TEXAS INSTRUMENTS

C281x C/C++ Header Files and Peripheral Examples Quick Start

1 Device Support:

This software package supports 281x devices. This includes the following: TMS320F2812, TMS320F2811, TMS320F2810, TMS320C2812, TMS320C2811, TMS320C2810, TMS320R2812, and TMS320R2811.

Throughout this document, TMS320F2812, TMS320F2811, TMS320F2810, TMS320C2812, TMS320C2811, TMS320C2810, TMS320R2812, and TMS320R2811 are abbreviated as F2812, F2811, F2810, C2812, C2811, C2810, R2812, and R2811, respectively.

2 Introduction:

The DSP281x peripheral header files and example projects included in (SPRC097) facilitate writing in C/C++ Code for the Texas Instruments '281x 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 '281x development platform. One such platform is the eZdsp™[††] F2812 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 DSP281x 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.

[†] 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.

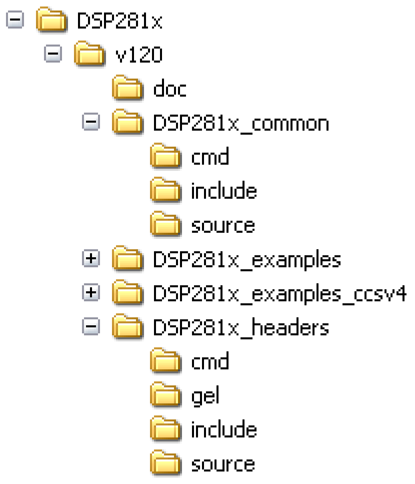
- Troubleshooting tips and frequently asked questions.
- Migration tips for users moving from the previous release V.58 to V1.00 and newer.

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 281x 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

V1.20 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. V1.11 is a minor release with only a few updates to fix bit name typos and add clarity to examples. V1.10 is a release to fix multiple typos and errors in V1.00 and to add new examples. A detailed revision history can be found in Section 9. An errata listing known typos and errors not fixed in this release to prevent code incompatibility for users between V1.00 and the newest version can be found in Section 10.

2.2 Where Files are Located (Directory Structure)



As installed, the *C281x C/C++ Header Files and Peripheral Examples* (SPRC097) is partitioned into a well-defined directory structure. By default, the source code is installed into the c:\tidcs\c28\DSP281x\<version> directory.

Table 1 describes the contents of the main directories:

Table 1. DSP281x Main Directory Structure

Directory	Description
<base>	Base install directory. By default this is c:\tidcs\c28\DSP281x\v111. 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>\DSP281x_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>\DSP281x_examples	Example Code Composer Studio projects based on the DSP281x header files. These example projects illustrate how to configure many of the '281x on-chip peripherals. An overview of the examples is given in Section 4.
<base>\DSP281x_examples_ccsv4	Example Code Composer Studio v4 projects compiled with floating point unit *enabled*. These examples are identical to those in the \DSP281x_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>DSP281x_common	Common source files shared across a number of the DSP281x example projects to illustrate how to perform tasks using the DSP281x 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 *DSP281x_headers* and *DSP281x_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. DSP281x Sub-Directory Structure

Sub-Directory	Description
DSP281x_headers\cmd	Linker command files that allocate the bit-field structures described in Section 3.
DSP281x_headers\source	Source files required to incorporate the header files into a new or existing project.
DSP281x_headers\include	Header files for each of the 281x on-chip peripherals.
DSP281x_common\cmd	Example memory command files that allocate memory on the '281x devices.
DSP281x_common\include	Common .h files that are used by the DSP281x peripheral examples.
DSP281x_common\source	Common .c files that are used by the DSP281x peripheral examples.

3 Understanding The Peripheral Bit-Field Structure Approach

The DSP281x header files and peripheral examples use a bit-field structure approach for mapping and accessing peripheral registers on the TI '281x based DSPs. For more information on using this technique, refer to the application note *Programming TMS32028xx and 28xxx Peripherals in C/C++* (SPRAA85).
.

This application note explores the hardware abstraction layer implementation to make C/C++ coding easier on 28x DSPs. This method is compared to traditional #define macros and topics of code efficiency and special case registers are also addressed.

4 Peripheral Example Projects

In the *DSP281x_examples* directory of *C281x C/C++ Header Files and Peripheral Examples* (SPRC097) there are several example projects that use the DSP281x header files to configure the on-chip peripherals.

4.1 Getting Started

4.1.1 Getting Started in Code Composer Studio v3.x

To get started, follow these steps to load the DSP281x CPU-Timer example. Other examples are set-up in a similar manner.

1. **Have an F2812 eZdsp or other hardware platform connected to a host with Code Composer Studio installed.**
2. **Load the example's GEL file (.gel) or Project file (.pjt).**

   Each example includes a Code Composer Studio GEL file to automate loading of the project, compiling of the code and populating of the watch window. Alternatively, the project itself 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: *DSP281x_examples\cpu_timer*

   c. Select *Example_281xCpuTimer.gel* and click on *open*.

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

      DSP281x CpuTimerExample-> Load_and_Build_Project

      This will load the project and build compile the project.

3. **Edit DSP28_Device.h**

   Edit the DSP281x_Device.h file in the *DSP281x_headers\include* directory and make sure the appropriate device is selected. By default the 2812 is selected.

```
/*********************************************************************
* DSP281x_headers\include\DSP281x_Device.h
*********************************************************************/

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

#define   DSP28_F2812   TARGET
#define   DSP28_F2811   0
#define   DSP28_F2810   0
```

4. **Edit DSP281x_Examples.h**

   Edit DSP281x_Examples.h and specify the clock rate.

   In DSP281x_Examples.h specify the SYSCLKOUT period. This value is used to scale a delay loop used by the examples. The default value is for 150Mhz SYSCLKOUT.

```
/*********************************************************************
* DSP281x_common\include\DSP281x_Examples.h
*********************************************************************/
……
#define CPU_RATE    6.667L   // for a 150MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE    7.143L   // for a 140MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE    8.333L   // for a 120MHz CPU clock speed (SYSCLKOUT)
//#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)
//#define CPU_RATE   33.333L   // for a 30MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE   41.667L   // for a 24MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE   50.000L   // for a 20MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE   66.667L   // for a 15MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE  100.000L   // for a 10MHz CPU clock speed  (SYSCLKOUT)//-----------
-----------------------------------------------------------------
```

5. **Review the comments at the top of the main source file: Example_281xCpuTimer.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.

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

   Perform any hardware setup indicated by the comments in the main source. The DSP281x CPU-Timer example only requires that the hardware be setup for “Boot to H0” 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 reference. For users with the F2812 eZdsp from Spectrum Digital, refer to the eZdsp’s user’s guide for the jumpers corresponding to the boot mode selection. For more information on the ‘281x boot modes refer to the *TMS320F28x Boot ROM Reference Guide* (SPRU095).

Table 3. 281x Boot Mode Settings

GPIOF4	GPIOF12	GPIOF3	GPIOF2	Mode
1	X	x	x	Boot to flash 0x3F7FF6
0	1	X	X	Call SPI boot loader
0	0	1	1	Call SCI boot loader
0	0	1	0	Boot to H0 SARAM 0x3F8000
0	0	0	1	Boot to OTP 0x3D7800
0	0	0	0	Call parallel boot loader

Note: X = Don't Care

7. **Load the code**

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

   DSP281x 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 example.**

   If you wish to modify the examples it is suggested that you make a copy of the entire DSP281x 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 *DSP281x_headers* directory and shared source in the *DSP281x_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 contained in the SPRC097 download 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:

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

   f. 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 DSP281x_Device.h file in the *DSP281x_headers\include* directory and make sure the appropriate device is selected. By default the 2812 is selected.

```
/*********************************************************************
* DSP281x_headers\include\DSP281x_Device.h
*********************************************************************/

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

#define   DSP28_F2812   TARGET
#define   DSP28_F2811   0
#define   DSP28_F2810   0
```

4. **Edit DSP281x_Examples.h**

   Edit DSP281x_Examples.h and specify the clock rate.

   In DSP281x_Examples.h specify the SYSCLKOUT period. This value is used to scale a delay loop used by the examples. The default value is for 150Mhz SYSCLKOUT.

```
/********************************************************************
* DSP281x_common\include\DSP281x_Examples.h
*******************************************************************/
……
#define CPU_RATE    6.667L   // for a 150MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE    7.143L   // for a 140MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE    8.333L   // for a 120MHz CPU clock speed (SYSCLKOUT)
//#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)
//#define CPU_RATE   33.333L   // for a 30MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE   41.667L   // for a 24MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE   50.000L   // for a 20MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE   66.667L   // for a 15MHz CPU clock speed  (SYSCLKOUT)
//#define CPU_RATE  100.000L   // for a 10MHz CPU clock speed  (SYSCLKOUT)//-----------
-----------------------------------------------------------------
```

5. **Review the comments at the top of the main source file: Example_281xCpuTimer.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.

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

   Perform any hardware setup indicated by the comments in the main source. The DSP281x CPU-Timer example only requires that the hardware be setup for "Boot to H0" 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 reference. For users with the F2812 eZdsp from Spectrum Digital, refer to the eZdsp's user's guide for the jumpers corresponding to the boot mode selection. For more information on the '281x boot modes refer to the *TMS320F28x Boot ROM Reference Guide* (SPRU095).

Table 4. 281x Boot Mode Settings

GPIOF4	GPIOF12	GPIOF3	GPIOF2	Mode
1	X	x	x	Boot to flash 0x3F7FF6
0	1	X	X	Call SPI boot loader
0	0	1	1	Call SCI boot loader
0	0	1	0	Boot to H0 SARAM 0x3F8000
0	0	0	1	Boot to OTP 0x3D7800
0	0	0	0	Call parallel boot loader

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.

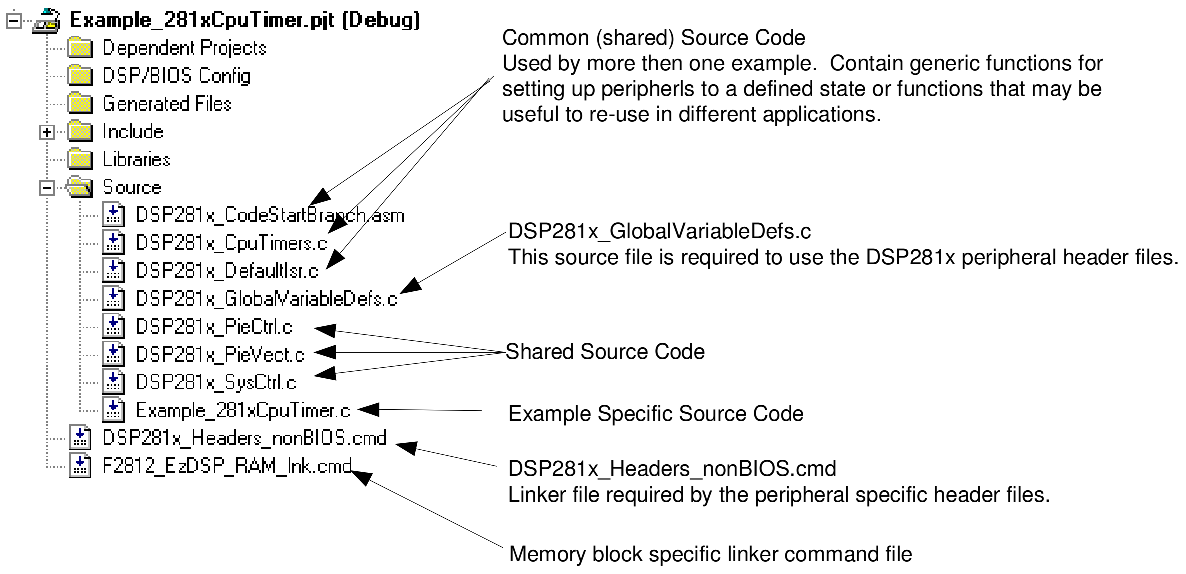
Go *to View->C/C++ Projects* to open up your project view. To remove/delete the project from the workspace, right click on the project's name and select *delete*. Make sure the *Do not delete* contents button is selected, then select *Yes.* This does not delete the project itself. It merely removes the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

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.


Example_281xCpuTimer.pjt (Debug)
Dependent Projects
DSP/BIOS Config
Generated Files
Include
Libraries
Source
DSP281x_CodeStartBranch.asm
DSP281x_CpuTimers.c
DSP281x_DefaultIsr.c
DSP281x_GlobalVariableDefs.c
DSP281x_PieCtrl.c
DSP281x_PieVect.c
DSP281x_SysCtrl.c
Example_281xCpuTimer.c
DSP281x_Headers_nonBIOS.cmd
F2812_EzDSP_RAM_lnk.cmd
Common (shared) Source Code
Used by more then one example. Contain generic functions for setting up peripherls to a defined state or functions that may be useful to re-use in different applications.
DSP281x_GlobalVariableDefs.c
This source file is required to use the DSP281x peripheral header files.
Shared Source Code
Example Specific Source Code
DSP281x_Headers_nonBIOS.cmd
Linker file required by the peripheral specific header files.
Memory block specific linker command file

4.2.1 *Include Files*

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

```
/*********************************************************************
* DSP281x_examples\cpu_timer\Example_281xCpuTimer.c
*********************************************************************/

#include "DSP281x_Device.h"     // DSP281x Headerfile Include File
#include "DSP281x_Examples.h"   // DSP281x Examples Include File
```

- **DSP281x_Device.h**

  This header file is required to use the DSP281x peripheral 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 *DSP281x_headers\include* directory.

- **DSP281x_Examples.h**

  This header file defines parameters that are used by the example code. This file is not required to use just the DSP281x peripheral header files but is required by some of the common source files. This file is found in the *DSP281x_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.

- **DSP281x_GlobalVariableDefs.c**

  Any project that uses the DSP281x 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 DSP281x_headers\source directory.

- **Example specific source code:**

  Files that are specific to a particular example have the prefix Example_281x on their filename. For example Example_281xCpuTimer.c is specific to the CPU Timer example and not used for any other example. Example specific files are located in the DSP281x_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 DSP281x_common\source directory. Users may choose to incorporate none, some, or all of the 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 DSP281x 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 *DSP281x_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
F2812_EzDSP_RAM_lnk.cmd	DSP281x_common\cmd	eZdsp memory map that only allocates SARAM locations. No Flash, OTP, or CSM password protected locations are used.
F2810.cmd	DSP281x_common\cmd	F2810 memory linker command file. Includes all Flash, OTP and CSM password protected memory locations.
F2812.cmd	DSP281x_common\cmd	F2812 memory linker command file. . Includes all Flash, OTP and CSM password protected memory locations. This linker command file is valid for F2811 as well.
F2812_XintfBoot.cmd	DSP281x_common\cmd	F2812 boot from XINTF Zone 7

- **DSP281x header file structure data section allocation:**

  Any project that uses the DSP281x 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. DSP281x Peripheral Header Linker Command File

DSP281x Peripheral Header File Linker Command File	Location	Description
DSP281x_Headers_BIOS.cmd	DSP281x_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.
DSP281x_Headers_nonBIOS.cmd	DSP281x_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 the 281x devices. Figure 1 outlines this basic flow:

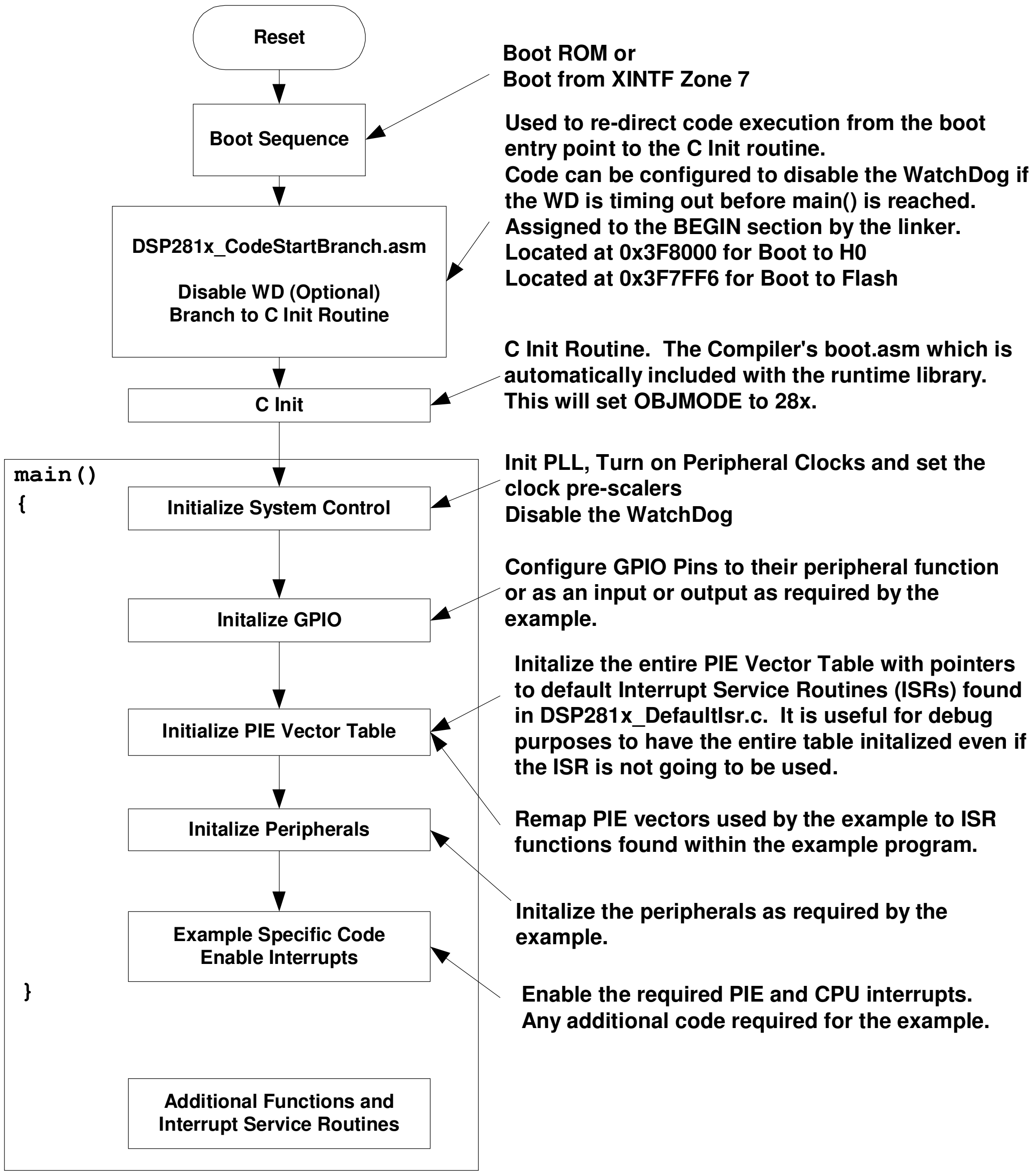

Reset
Boot Sequence
Boot ROM or
Boot from XINTF Zone 7
DSP281x_CodeStartBranch.asm
Disable WD (Optional)
Branch to C Init Routine
Used to re-direct code execution from the boot entry point to the C Init routine.
Code can be configured to disable the WatchDog if the WD is timing out before main() is reached.
Assigned to the BEGIN section by the linker.
Located at 0x3F8000 for Boot to H0
Located at 0x3F7FF6 for Boot to Flash
C Init
C Init Routine. The Compiler's boot.asm which is automatically included with the runtime library.
This will set OBJMODE to 28x.
main()
{
Initialize System Control
Init PLL, Turn on Peripheral Clocks and set the clock pre-scalers
Disable the WatchDog
Initalize GPIO
Configure GPIO Pins to their peripheral function or as an input or output as required by the example.
Initialize PIE Vector Table
Initalize the entire PIE Vector Table with pointers to default Interrupt Service Routines (ISRs) found in DSP281x_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 Peripherals
Initalize the peripherals as required by the example.
Example Specific Code
Enable Interrupts
Enable the required PIE and CPU interrupts.
Any additional code required for the example.
}
Additional Functions and
Interrupt Service Routines

Figure 1. Flow for Example Programs

4.4 Included Examples:

Table 7. Included Examples

Example	Description
adc_seqmode_test	ADC Seq Mode Test. Channel A0 is converted forever and logged in a buffer
adc_seq_ovd_tests	ADC test using the sequencer override feature available as of silicon Rev C.
adc_soc	ADC example to convert two channels: ADCINA3 and ADCINA2. Interrupts are enabled and EVA 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_back2back	eCAN self-test mode example. Transmits eCAN data back-to-back at high speed without stopping.
ev_pwm	Event Manager PWM example. This program sets up the EV timers to generate PWM waveforms. The user can then observe the waveforms using a an oscilloscope.
ev_timer_period	Event Manager Timer example. This program sets up EVA and EVB timers to fire an interrupt on a period overflow. A count is kept each time each interrupt passes through the interrupt service routine.
Flash	EV Timer Example 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_loopback	General Purpose I/O loop back test. In this test, 8 bits of a GPIO Port are configured as outputs and 8 bits of the same port are configured as inputs. The pins configured as outputs are externally looped back to the pins configured as inputs. The output data is read back on the input pins.
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.
lpm_haltwake	Puts device into low power halt mode. XNMI is configured to wake the device from halt when an external high-low signal is applied to it.
lpm_idlewake	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_standbywake	Puts device into low power standby mode. The watchdog interrupt will wake the device from standby mode.
mcbsp_loopback	McBSP is configured for loop-back test. Polling is used instead of interrupts.
mcbsp_loopback_interrupts	McBSP is configured for loop-back test. Both interrupts and FIFOs are used.
mcbsp_spi_loopback	McBSP is configured in SPI-mode for loop-back test. Polling and FIFOs are used.
run_from_xintf	Shows how to boot from XINTF zone 7 and configure the XINTF on the F2812 eZdsp.
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 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.
sci_loopback	SCI example code that uses the loop-back test mode of the SCI module to send characters This example uses bit polling and does not use interrupts.
sci_loopback_interrupts	SCI example code that uses the internal loop-back test mode to transfer data through SCI-A. Interrupts and FIFOs are both used in this example.
spi_loopback	SPI example that uses the peripherals loop-back test mode to send data.
spi_loopback_interrupts	SPI 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 DSP281x examples execute from SARAM in “boot to H0” mode. One example, *DSP281x_examples\Flash*, executes from flash memory in “boot to flash” mode. This example is the Event Manager timer example with the following changes made to execute out of flash:

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

   Remove F2812_EzDSP_RAM_lnk.cmd from the project and add F2812.cmd or F2810.cmd. Both F2810.cmd and F2812.cmd are located in the *DSP281x_common\cmd\* directory.

2. **Add the DSP281x_common\source\DSP281x_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 *TMS320F28x System Control and Interrupts Reference Guide* (SPRU078).

3. **Modify the code to copy functions that must be executed in 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 DSP281x 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.

```
/********************************************************************
* DSP281x_common\source\DSP281x_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:

```
/********************************************************************
* DSP281x_common\include\F2812.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	RamfuncsLoadStart
Load end	RamfuncsLoadEnd
Run start	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 *DSP281x_common\source\DSP281x_MemCopy.c* to the project.

b. Add the following function prototype to the example source code. This is done for you in the *DSP281x_Examples.h* file.

```
/********************************************************************
* DSP281x_common\include\DSP281x_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 DSP281x example code this has is already done in *DSP281x_Examples.h*.

```
/********************************************************************
* DSP281x_common\include\DSP281x_Examples.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.

```
/********************************************************************
* DSP281x_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.

```
/*********************************************************************
* DSP281x 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. 281x Boot Mode Settings

GPIOF4	GPIOF12	GPIOF3	GPIOF2	Mode
1	x	x	x	Boot to flash 0x3F7FF6
0	1	X	X	Call SPI boot loader
0	0	1	1	Call SCI boot loader
0	0	1	0	Boot to H0 SARAM 0x3F8000
0	0	0	1	Boot to OTP 0x3D7800
0	0	0	0	Call parallel boot loader

Note: X = Don’t Care

For users with the F2812 eZdsp from Spectrum Digital, refer to the eZdsp’s user’s guide for the jumpers corresponding to the boot mode selection.

For more information on the ‘281x boot modes refer to the *TMS320F28x Boot ROM Reference Guide* (SPRU095).

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 V.58 of the 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.**

   – *DSP281x_headers:* code required to incorporate the header files into your project

   – *DSP281x_common*: shared source code much of which is used in the example projects.

   – *DSP281x_examples*: example projects that use the header files and shared code.

5.2 Including the DSP281x Peripheral Header Files

Including the DSP281x 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 "DSP281x_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 "DSP281x_Device.h"
```

2. **Edit DSP281x_Device.h and select the target you are building for:**

   In the below example, the file is configured to build for the F2812 device.

```
/********************************************************************
* DSP281x_headers\include\DSP281x_Device.h
********************************************************************/

#define    TARGET              1
#define    DSP28_F2812         TARGET
#define    DSP28_F2811         0
#define    DSP28_F2810         0
```

3. **Add the source file *DSP281x_GlobalVariableDefs.c* to the project.**

   This file is found in the *DSP281x_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 DSP281x header linker command file to the project.**

   As described in Section 3, when using the DSP281x 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 *DSP281x_headers\cmd\* must be included in your project:

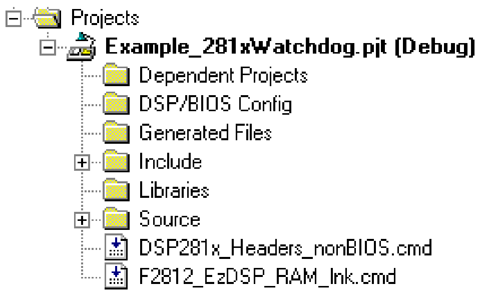
– For non-DSP/BIOS[†] projects:	*DSP281x_Headers_nonBIOS.cmd*
– For DSP/BIOS projects:	*DSP281x_Headers_BIOS.cmd*

   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.



```
Projects
  Example_281xWatchdog.pjt (Debug)
    Dependent Projects
    DSP/BIOS Config
    Generated Files
    Include
    Libraries
    Source
    DSP281x_Headers_nonBIOS.cmd
    F2812_EzDSP_RAM_lnk.cmd
```

   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 DSP281x header files, perform the following two steps:

   1) **Update the project's main linker command (.cmd) file to call one of the supplied DSP281x peripheral structure linker command files using the -l option.**

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

/* Use this include file only for non-BIOS applications */
-l DSP281x_Headers_nonBIOS.cmd
/* Use this include file only for BIOS applications */
/* -l DSP281x_Headers_BIOS.cmd */
```

[†] DSP/BIOS is a trademark of Texas Instruments

2) **Add the directory path to the DSP281x 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 *DSP281x_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*.

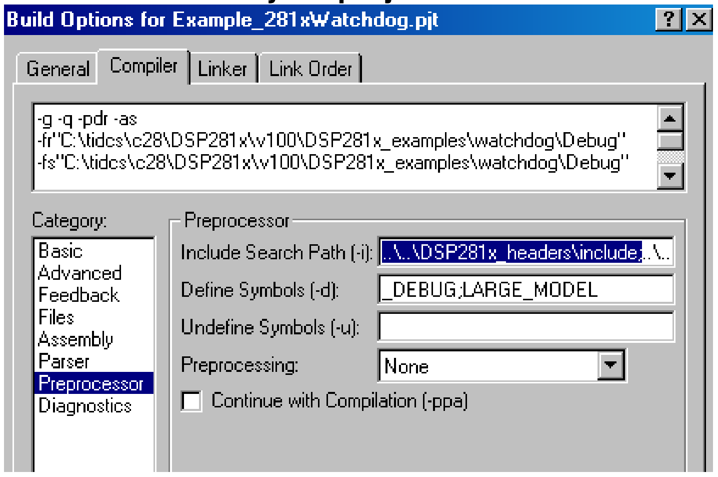
c. Click on the *Advanced>>* button to expand the window.

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 DSP281x header files to your project.


Build Options for Example_281xWatchdog.pjt

General | Compiler | Linker | Link Order

```
-g -q -pdr -as
-fr"C:\tidcs\c28\DSP281x\v100\DSP281x_examples\watchdog\Debug"
-fs"C:\tidcs\c28\DSP281x\v100\DSP281x_examples\watchdog\Debug"
```

Category: Basic, Advanced, Feedback, Files, Assembly, Parser, Preprocessor, Diagnostics

Preprocessor

Include Search Path (-i): ..\..\DSP281x_headers\include;..\..

Define Symbols (-d): _DEBUG;LARGE_MODEL

Undefine Symbols (-u):

Preprocessing: None

Continue with Compilation (-ppa)

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 *DSP281x_headers\include* on your system.

Code Composer Studio 4.x

To specify the directory where the header files are located:

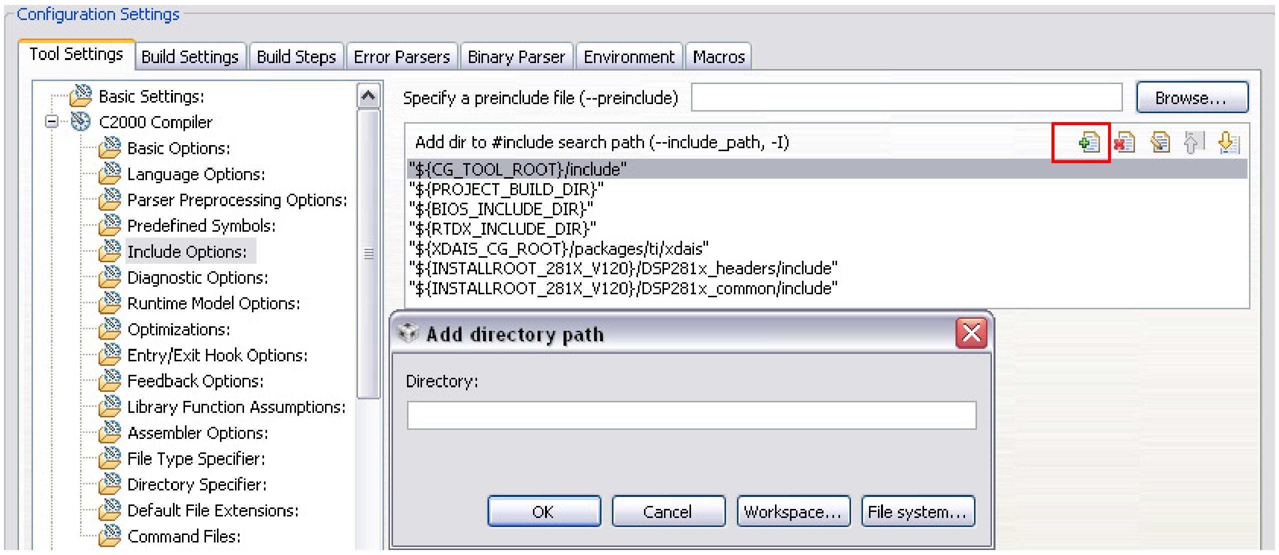
a. Open the menu: *Project->Properties*.

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.


Configuration Settings

Tool Settings | Build Settings | Build Steps | Error Parsers | Binary Parser | Environment | Macros

Basic Settings:
C2000 Compiler
Basic Options:
Language Options:
Parser Preprocessing Options:
Predefined Symbols:
Include Options:
Diagnostic Options:
Runtime Model Options:
Optimizations:
Entry/Exit Hook Options:
Feedback Options:
Library Function Assumptions:
Assembler Options:
File Type Specifier:
Directory Specifier:
Default File Extensions:
Command Files:

Specify a preinclude file (--preinclude) Browse...

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_281X_V120}/DSP281x_headers/include"
"${INSTALLROOT_281X_V120}/DSP281x_common/include"
```

Add directory path

Directory:

OK | Cancel | Workspace... | File system...

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.

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 “DSP281x_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 “DSP281x_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:

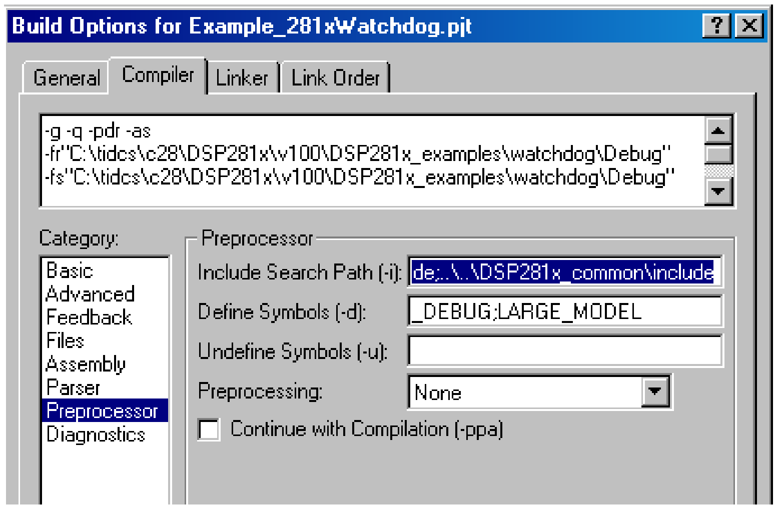
a. Open the menu:

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 DSP281x_common/include on your system.
Use a semicolon between directories.



```
-g -q -pdr -as
-fr"C:\tidcs\c28\DSP281x\v100\DSP281x_examples\watchdog\Debug"
-fs"C:\tidcs\c28\DSP281x\v100\DSP281x_examples\watchdog\Debug"
```

Include Search Path (-i): de;..\..\DSP281x_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:
..\..\DSP281x_headers\include;..\..\DSP281x_common\include

Code Composer Studio 4.x

To specify the directory where the header files are located:

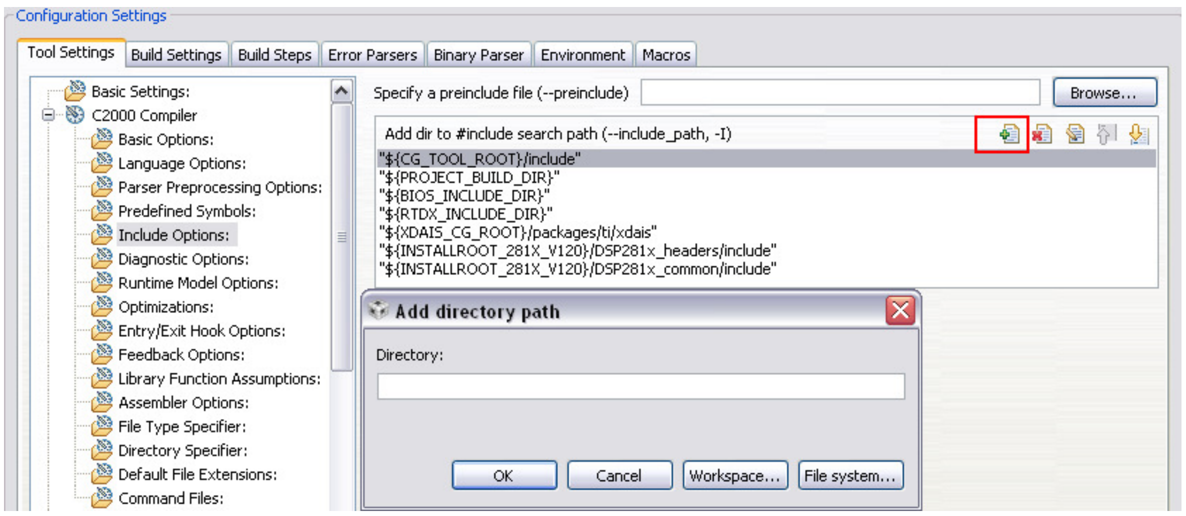
a. Open the menu: *Project->Properties*.

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_281X_V120}/DSP281x_headers/include"
"${INSTALLROOT_281X_V120}/DSP281x_common/include"
```

3. Add a linker command file to your project.

The following memory linker .cmd files are provided as examples in the *DSP281x_common\cmd* directory. For getting started the basic *F2812_EzDSP_RAM_lnk.cmd* file is suggested and used by most of the examples.

Table 9. Included Main Linker Command Files

Main Liner Command File Examples	Description
F2812_EzDSP_RAM_lnk.cmd	Main eZdsp example linker file. Only uses only SARAM locations that are not protected by the code security module. This memory map is used for all of the examples to run out of the box on an F2812 EzDSP. No Flash or OTP locations are used.
F2812_XintfBoot.cmd	Linker command file used for booting from XINTF Zone 7
F2810.cmd	Main F2810 linker command file. Includes all Flash and OTP memory locations.
F2812.cmd	Main F2812 linker command file. Includes all Flash, OTP and XINTF memory. This linker file can be used for F2811 as well.

4. Set the CPU Frequency

In the *DSP281x_common\include\DSP281x_Examples.h* file specify the proper CPU frequency. Some examples are included in the file.

```
/*********************************************************************
* DSP281x_common\include\DSP281x_Examples.h
*********************************************************************/

#define CPU_RATE    6.667L   // for a 150MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE    7.143L   // for a 140MHz CPU clock speed (SYSCLKOUT)
//#define CPU_RATE    8.333L   // for a 120MHz CPU clock speed (SYSCLKOUT)
```

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

   The common source files are found in the *DSP281x_common\source\* directory.

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

   Since all catalog ‘281x 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 DSP281x_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 *TMS320F28x Control and Interrupts Reference Guide* (SPRU078).

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

  Peripheral registers cannot be modified or read unless the clock to the specific peripheral is enabled. The function InitPeripheralClocks() in the DSP281x_common\source directory shows an example of enabling the peripheral clocks.

- **Memory block L0, L1 reads 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 *TMS320F28x Control and Interrupts Reference Guide* (SPRU078) 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 *TMS320F28x Control and Interrupts Reference Guide* (SPRU078) 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. Refer to the *TMS320F28x Control and Interrupts Reference Guide* (SPRU078) 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 that all initialized sections, such as .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 many of the examples, the compiler outputs the following: remark: controlling expression is constant. What does this mean?**

  Many 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?**

  Many 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. For example in the McBSP loopback program, depending on which serial word size the example is compiled for, some code may never be used.

- **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 DSP281x_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 from V.58 to V1.00 and newer

This section will guide you through the steps needed to migrate projects that are currently built using V.58 of the header files to V1.00 and newer.

1. **Create a copy of your project to work with or back-up your current project.**
2. **Create a copy of the header file source you want to use or create a back-up of the header files.**
   - *DSP281x_headers*: code required to incorporate the header files into your project
   - *DSP281x_common:* shared source code much of which is used in the example projects.
   - *DSP281x_examples*: example projects that use the header files and shared code.
3. **File name changes**

   The filenames of the standard files have changed slightly since the V.58 release. Previously all standard header files and example code files began with DSP28. In anticipation of future '28x devices, the prefix DSP28 has been changed to DSP281x.
   - **Update the project file:**

     If your project uses the example .c files, then open the project file (.pjt) in a text editor. Using a search and replace method, change all instances of DSP28 to DSP281x. It is advised that you review the change before it is made. That is, use the find next option, review the change and then perform the replacement.
   - **Update your source:**

     In your source code if you have included DSP28_Device.h this should be changed to DSP281x_Device.h.
4. **Load the project into Code Composer Studio**

   Code Composer will complain that it cannot find some of the source files. This is due to the new directory structure used for V1.00 and newer. This change was done to better partition the header files from the example code.

   As you are prompted for each source file location, browse to the new location of the file.
   - DSP281x_GlobalVariableDefs.c is located in *DSP281x_headers\source*
   - All other .c files can be found in *DSP281x_common\source*

– Memory linker .cmd files are located in *DSP281x_common\cmd*

– If you were using the file: EzDSP_RAM_lnk.cmd, then remove this file from the project and replace it with F2812_EzDSP_RAM_lnk.cmd located in *DSP281x_common\cmd*

5. **Follow all of the steps in Section 5 to incorporate the header files and example source into your existing project. Some of these steps may already be complete.**

   Some of the major differences between V.58 and V1.00 and newer are highlighted below:

   – Section 5.2 step 6: Include the header linker command file. The linker files have now been split into memory specific and peripheral header file specific files.

   – Section 5.2 step 7: Update the include search path for the new location of the header files.

   – Section 5.3 step 1 & 2: If your project uses any of the sample code, include DSP281x_Examples.h in your source code. This file contains the example specific information that used to be part of DSP281x_Device.h.

6. **Build the project.**

   The compiler will highlight areas that have changed. Most of the changes will be bit-name or register name corrections to align with the peripheral user guides. Some example errors and their solutions are outlined below.

   – **Register name has changed to align with the user's guide:**

   Example: `struct "EVA_REGS" has no field "CAPCON".`

   Solution: Refer to Table 10 for register changes. Table 10 shows that CAPCON for EV-A was changed to CAPCONA. Update the code to use CAPCONA.

   – **Bit field name has changed to align with the user's guide:**

   Example error: `struct "FOTPWAIT_BITS" has no field "OPTWAIT"` Solution:

   Solution: Refer to Table 11 for bit name changes. Table 11 shows that OPTWAIT was changed to OTPWAIT. Update the code to use OTPWAIT.

   – **Register was removed and is no longer used.**

   Example: struct "DEV_EMU_REGS" has no field "M0RAMDFT"

   Solution: Refer to Table 10 for register changes. Table 10 indicates that this register was removed and no longer needs to be initialized. Remove the code that initializes this register.

   – **Register bit-field definitions for a register were removed:**

   Example: `expression must have struct or union type`

   This error occurs when the .bit or the .all is used to access a register that no longer has a union defined.

   Solution: Examine the source code that caused this error. For example:

```
SysCtrlRegs.SCSR.all = 0x0002;
```

Refer to Table 11 for bit name changes. Table 11 indicates the bit field was removed for this register because of the sensitivity of other bits to read-modify-write instructions. Modify the code to not use .bit or .all:

```
SysCtrlRegs.SCSR = 0x0002;
```

– **Register bit-field definitions for a register were added:**

Example: `a value of type "int" cannot be assigned to an entity of type "union PLLCR_REG"`

This error occurs when a register that has a bit-field definition is accessed without specifying the .bit or the .all union member.

Solution: Look at the source that caused the error. For example:

```
SysCtrlRegs.PLLCR = 0x000A;
```

Refer to Table 11 for bit name changes. Table 11 indicates that bit fields were added for this register. The solution is to access the register using the .all union member:

```
SysCtrlRegs.PLLCR.all = 0x000A;
```

7. **Enabling the PIE.**

   In V.58 the PIE block was enabled in the IntPieCtrl() function. In the examples this occurred before the PIE vector table was initialized. The PIE enable has been removed from the IntPieCtrl() function and is now done after the PIE table initialization. Users should take care to insure the PIE is properly enabled in their projects.

8. **PLL lock time change.**

   As of Rev C F2810/12 silicon, the lock time of the PLL has changed to 131072 CLKIN cycles. Make sure this change is reflected in your code.

9. **M0RAMDFT, M1RAMDFT, L0RAMDFT, L1RAMDFT and H0RAMDFT were removed:**

   On F2810/12 prior to Rev C silicon initialization of these registers was required. This is no longer required as of Rev C silicon and the code that initializes them should be removed.

Table 10. Register Name Changes

	Register Name		
Peripheral	**Old**	**New**	**Comment**
DevEmuRegs			
	M0RAMDFT	-	Register removed. Init no longer needed.
	M1RAMDFT	-	Register removed. Init no longer needed.
	L0RAMDFT	-	Register removed. Init no longer needed.
	L1RAMDFT	-	Register removed. Init no longer needed.
	H0RAMDFT	-	Register removed. Init no longer needed.
EcanaRegs			
	CANLNT	CANTSC	Alignment with user's guide.
	CANMID	CANMSGID	Alignment with user's guide.
	CANMCF	CANMSGCTRL	Alignment with user's guide.
	MDRL	MDL	Alignment with user's guide. Register can now be accessed as .byte or .word
	MDRH	MDH	Alignment with user's guide. Register can now be accessed as .byte or .word
EvaRegs			
	EXTCON	EXTCONA	Alignment with user's guide.
	CAPCON	CAPCONA	Alignment with user's guide
	CAPFIFO	CAPFIFOA	Alignment with user's guide
McbspaRegs			
	PCR1	PCR	Alignment with user's guide.

Table 11. Summary of Bit-Name Changes from V.58 to V1.00 and Newer

Peripheral	Register	Bit Name		Comment
		Old	New	
AdcRegs				
	ADCMAXCONV	MAX_CONV	MAX_CONV1 MAX_CONV2	Field was split into two parts: MAX_CONV1 0:3 & MAX_CONV2 4:6
	ADCTRL1	rsvd2	SEQ_OVRD	New Feature as of Rev C
CpuTimerRegs				
	TCR	OUTSTS	reserved	Feature not implemented on F281x
		FORCE	reserved	Feature not implemented on F281x
		POL	reserved	Feature not implemented on F281x
		TOG	reserved	Feature not implemented on F281x
		FRCEN	reserved	Feature not implemented on F281x
		PWIDTH	reserved	Feature not implemented on F281x
DevEmuRegs				
	DEVICEID	PARTID	reserved	Feature no longer supported
	M0RAMDFT	-	-	Removed. Init no longer needed.
	M1RAMDFT	-	-	Removed. Init no longer needed.
	L0RAMDFT	-	-	Removed. Init no longer needed.
	L1RAMDFT	-	-	Removed. Init no longer needed.
	H0RAMDFT	-	-	Removed. Init no longer needed.
EcanaRegs				
	CANMC	SCM	SCB	Alignment with user's guide.
		LNTM	TCC	Alignment with user's guide.
		LNTC	MBCC	Alignment with user's guide.
	CANBTC	TSEG2	TSEG2REG	Alignment with user's guide.
		TSEG1	TSEG1REG	Alignment with user's guide.
		SJW	SJWREG	Alignment with user's guide.
		ERM	reserved	Feature not implemented on F281x
		ERM	reserved	Alignment with user's guide.
		BRP	BRPREG	
	CANGIFO	TCOIFO	TCOFO	Alignment with user's guide.
		MAIFO	MTOFO	Alignment with user's guide.
	CANGIM	SIL	GIL	Alignment with user's guide.
		TCOIM	TCOM	Alignment with user's guide.
		MAIM	MTOM	Alignment with user's guide.
	CANGIF1	TCOIF1	TCOF1	Alignment with user's guide.
		MAIF1	MTOF1	Alignment with user's guide.

Table 10 Continued - Summary of Bit-Name Changes from V.58 to V1.00 and Newer

		Bit Name		
Peripheral	**Register**	**Old**	**New**	**Comment**
EcanaRegs continued				
	CANTIOC	TXIN	Reserved	Feature not implemented
		TXOUT	Reserved	Feature not implemented
		TXDIR	Reserved	Feature not implemented
	CANRIOC	RXIN	Reserved	Feature not implemented
		RXOUT	Reserved	Feature not implemented
		RXDIR	Reserved	Feature not implemented
	CANMSGID	MSGID_L	EXTMSGID_L	Alignment with user's guide.
		MSGID_H	EXTMSGID_H STDMSGID	Due to 16-bit size limit for bit-fields, this was broken into two parts
EvaRegs				
	GPTCONA	TCOMPOE	TCMPOE	Alignment with user's guide.
		rsvd2	T1CTRIPE T2CTRIPE	Correction
	EXTCONA	QEPIQEL	QEPIQUAL	Correction
	COMCONA	rsvd	C1TRIPE C2TRIPE C3TRIPE FCMP1OE FCMP2OE FCMP3OE	Correction
	CAPCONA	CAPQEPN	CAP12EN	Alignment with user's guide.
EvbRegs				
	GPTCONB	TCOMPOE	TCMPOE	Alignment with user's guide.
		T1CTRIP	T3CTRIPE	Correction
		T2CTRIP	T4CTRIPE	Correction
	EXTCONB	QEPIQEL	QEPIQUAL	Correction
	COMCONB	rsvd3	C4TRIPE C5TRIPE C6TRIPE FCMP4OE FCMP5OE FCMP6OE	Correction
	CAPCONB	CAPQEPN	CAP45EN	Alignment with user's guide.

Table 10 Continued - Summary of Bit-Name Changes from V.58 to V1.00 and Newer

		Bit Name		
Peripheral	**Register**	**Old**	**New**	**Comment**
McbspaRegs				
	XCERA	XCEA0-XCEA15	XCERA0-XCERA15	Alignment with user's guide.
	XCERB	XCEB0-XCEB15	XCERB0-XCERB15	Alignment with user's guide.
	XCERC	XCEC0-XCEC15	XCERC0-XCERC15	Alignment with user's guide.
	XCERD	XCED0-XCED15	XCERD0-XCERD15	Alignment with user's guide.
	XCERE	XCEE0-XCEE15	XCERE0-XCERE15	Alignment with user's guide.
	XCERC	XCEF0-XCEF15	XCERF0-XCERF15	Alignment with user's guide.
	XCERG	XCEG0-XCEG15	XCERG0-XCERG15	Alignment with user's guide.
	MFFCT	TXDLY	FFTXDLY	Alignment with user's guide.
	MFFRX	IL	RXFFIL	Alignment with user's guide
		INT_CLR	RXFFINT_CLEAR	Alignment with user's guide
		INT	RXFFINT_FLAG	Alignment with user's guide
		ST	RXFFST	Alignment with user's guide
		RRESET	RXFIFO_RESET	Alignment with user's guide
		OVF_CLR	RXFFOVF_CLEAR	Alignment with user's guide
		OVF	RXFFOVF_FLAG	Alignment with user's guide
	MFFTX	IL	TXFFIL	Alignment with user's guide
		INT_CLR	TXFFINT_CLEAR	Alignment with user's guide
		INT	TXFFINT_FLAG	Alignment with user's guide
		ST	TXFFST	Alignment with user's guide
		XRESET	TXFIFO_RESET	Alignment with user's guide
	SPCR1	REMPTY	RFULL	Correction
	SRGR2	GYSNC	GSYNC	Correction
		CLKSP	resvd	Correction
SciaRegs				
	SCIRXST	RXERR	RXERROR	Alignment with user's guide.
	SCIFFTX	resvd	SCIRST	Correction
	SCIFFRX	RSOVF_CLR	RXFFOVRCLR	Alignment with user's guide.
ScibRegs				
	SCIRXST	RXERR	RXERROR	Alignment with user's guide.
	SCIFFTX	resvd	SCIRST	Correction
	SCIFFRX	RSOVF_CLR	RXFFOVRCLR	Alignment with user's guide.
SpiaRegs				
	SPIFFTX	TXFFINTINTCLR	TSFFINTCLR	Alignment with user's guide.
		TXFIFORESET	TXFIFO	Alignment with user's guide.
		rsvd	SPIFFENA SPIRST	Correction
	SPICCR	RESET	SPISWRESET	Alignment with user's guide.
	SPICTL	OVERRUN	OVERRUNINTENA	Alignment with user's guide.

Table 10 Continued - Summary of Bit-Name Changes from V.58 to V1.00 and Newer

		Bit Name		
Peripheral	**Register**	**Old**	**New**	**Comment**
SysCtrlRegs				
	PCLKCR	SCIENCLKA	SCIAENCLK	Alignment with user's guide.
		SCIENCLKB	SCIBENCLK	Alignment with user's guide.
	SCSR	WDOVERRIDE WDENINT		Register bit fields were removed due to WDOVERRIDE sensitivity to read-modify-write instructions Use: SysCtrlRegs.SCSR = MASK
	LPMCR0	-	LPM QUALSTDBY	Bit fields added. Use .all or .bit to access this register.
	LPMCR1	-	XINT1 XNMI WDINT etc…	Bit fields added. Use .all or .bit to access this register.
	PLLCR		DIV	Bit fields added. Use .all or .bit to access this register.
FlashRegs				
	FBANKWAIT	OPTWAIT	OTPWAIT	Typo correction

8 Packet Contents:

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

8.1 Header File Support – DSP281x_headers

The DSP281x header files are located in the *<base>\DSP281x_headers\* directory.

8.1.1 DSP281x Header Files – Main Files

The following files must be added to any project that uses the DSP281x header files. Refer to section 5.2 for information on incorporating the header files into a new or existing project.

Table 12. DSP281x Header Files – Main Files

File	Location	Description
DSP281x_Device.h	DSP281x_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.
DSP281x_GlobalVariableDefs.c	DSP281x_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.
DSP281x_Headers_BIOS.cmd	DSP281x_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.
DSP281x_Headers_nonBIOS.cmd	DSP281x_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 DSP281x 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 281x devices. These files are automatically included in the project by including *DSP281x_Device.h.* Refer to section 4.2 for more information on incorporating the header files into a new or existing project.

Table 13. DSP281x Header File Bit-Field & Register Structure Definition Files

File	Location	Description
DSP281x_Adc.h	DSP281x_headers\include	ADC register structure and bit-field definitions.
DSP281x_CpuTimers.h	DSP281x_headers\include	CPU-Timer register structure and bit-field definitions.
DSP281x_DevEmu.h	DSP281x_headers\include	Emulation register definitions
DSP281x_ECan.h	DSP281x_headers\include	eCAN register structures and bit-field definitions.
DSP281x_Ev.h	DSP281x_headers\include	Event manager (EV) register structures and bit-field definitions.
DSP281x_Gpio.h	DSP281x_headers\include	General Purpose I/O (GPIO) register structures and bit-field definitions.
DSP281x_Mcbsp.h	DSP281x_headers\include	McBSP register structure and bit-field definitions.
DSP281x_PieCtrl.h	DSP281x_headers\include	PIE control register structure and bit-field definitions.
DSP281x_PieVect.h	DSP281x_headers\include	Structure definition for the entire PIE vector table.
DSP281x_Sci.h	DSP281x_headers\include	SCI register structure and bit-field definitions.
DSP281x_Spi.h	DSP281x_headers\include	SPI register structure and bit-field definitions.
DSP281x_SysCtrl.h	DSP281x_headers\include	System register definitions. Includes Watchdog, PLL, CSM, Flash/OTP, Clock registers.
DSP281x_Xintf.h	DSP281x_headers\include	External memory interface (XINTF) register structure and bit-field definitions.
DSP281x_XIntrupt.h	DSP281x_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 DSP281x Header File peripheral register structures.

Table 14. Included GEL Files

File	Location	Description
DSP281x_Peripheral.gel	DSP281x_headers\gel	This is relevant for CCSv3.x only Provides GEL pull-down menus to load the DSP281x data structures into the watch window. You may want to have CCS load this file automatically by adding a GEL_LoadGel("<base>DSP281x_headers\/gel\DSP281xperipheral.gel") function to the standard F2812.gel that was included with CCS.
DSP281x_GpioQuickRef.gel	DSP281x_headers\gel	Provides a quick reference for the General Purpose I/O ports on the F281x DSPs. It simply prints out the MUX information into a debugger window.

8.1.4 Variable Names and Data Sections

This section is a summary of the variable names used and data sections allocated by the DSP281x_headers\source\DSP281x_GlobalVariableDefs.c file.

Peripheral	Starting Address	Structure Variable Name
ADC	0x007100	AdcRegs
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	0x006000	ECanaRegs
eCAN Mail Boxes	0x006100	ECanaMboxes
eCAN Local Acceptance Masks	0x006040	ECanaLAMRegs
eCAN Message Object Time Stamps	0x006080	ECanaMOTSRegs
eCAN Message Object Time-Out	0x0060C0	ECanaMOTORegs
Event Manager A (EV-A)	0x007400	EvaRegs
Event Manager B (EV-B)	0x007500	EvbRegs
Flash & OTP Configuration Registers	0x000A80	FlashRegs
General Purpose I/O Data Registers	0x0070E0	GpioDataRegs
General Purpose MUX Registers	0x0070C0	GpioMuxRegs
McBSP Registers	0x007800	McbspaRegs
PIE Control	0x000CE0	PieCtrlRegs

8.2 Common Example Code – DSP281x_common

8.2.1 Peripheral Interrupt Expansion (PIE) Block Support

In addition to the register definitions defined in DSP281x_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
DSP281x_DefaultIsr.c	DSP281x_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.**
DSP281x_DefaultIsr.h	DSP281x_common\include	Function prototype statements for the ISRs in DSP281x_DefaultIsr.c. **Note: This file is not used for DSP/BIOS projects.**
DSP281x_PieVect.c	DSP281x_common\source	Creates an instance of the PIE vector table structure initialized with pointers to the ISR functions in DSP281x_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 and documentation in *DSP281x_examples\sw_prioritized_interrupts* for more information.

Table 16. Software Prioritized Interrupt PIE Block Specific Support Files

File	Location	Description
DSP281x_SWPrioritizedDefaultIsr.c	DSP281x_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.**
DSP281x_SWPrioritizedIsrLevels.h	DSP281x_common\include	Function prototype statements for the ISRs in DSP281x_DefaultIsr.c. **Note: This file is not used for DSP/BIOS projects.**
DSP281x_SWPrioritizedPieVect.c	DSP281x_common\source	Creates an instance of the PIE vector table structure initialized with pointers to the default ISR functions that are included in DSP281x_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 *DSP281x_common\src\* directory. These files include:

Table 17. Included Peripheral Specific Files

File	Description
DSP281x_GlobalPrototypes.h	Function prototypes for the peripheral specific functions included in these files.
DSP281x_Adc.c	ADC specific functions and macros.
DSP281x_CpuTimers.c	CPU-Timer specific functions and macros.
DSP281x_ECan.c	Enhanced CAN specific functions and macros.
DSP281x_Ev.c	Event Manager (EV) specific functions and macros.
DSP281x_Gpio.c	General-purpose IO (GPIO) specific functions and macros.
DSP281x_Mcbsp.c	McBSP specific functions and macros.
DSP281x_PieCtrl.c	PIE control specific functions and macros.
DSP281x_Sci.c	SCI specific functions and macros.
DSP281x_Spi.c	SPI specific functions and macros.
DSP281x_SysCtrl.c	System control (watchdog, clock, PLL etc) specific functions and macros.
DSP281x_Xintf.c	External memory interface (XINTF) specific functions and macros.
DSP281x_XIntrupt.c	External interrupts 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
DSP281x_CodeStartBranch.asm	Branch to the start of code execution. This is used to re-direct code execution when booting to Flash, OTP or H0 SARAM memory. An option to disable the watchdog before the C init routine is included. If booting from XINTF Zone 7, use DSP281x_XintfBootReset.asm instead.
DSP281x_XintfBootReset.asm	This file is used to boot from XINTF Zone 7. An option to disable the watchdog before the C init routine is included. If booting to H0, Flash or OTP, use DSP281x_CodeStartBranch.asm instead.
DSP281x_DBGIER.asm	Assembly function to manipulate the DEBIER register from C.
DSP281x_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 *DSP281x_examples\adc* for an example of its use.
DSP281x_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 *DSP281x_common\cmd* directory. For getting started using the 281x devices, the basic F2812_EzDSP_RAM_lnk.cmd file is suggested and used by many of the included examples.

Table 19. Included Main Linker Command Files

Main Liner Command File Examples	Description
F2812_EzDSP_RAM_lnk.cmd	eZdsp memory linker example. Only allocates SARAM locations. This memory map is used for all of the examples that run out of the box on an F2812 EzDSP. No Flash, OTP, or CSM password protected locations are used.
F2810.cmd	F2810 memory linker command file. Includes all Flash, OTP and CSM password protected memory locations.
F2812.cmd	F2812 memory linker command file. . Includes all Flash, OTP and CSM password protected memory locations. This linker command file can be used for the F2811 also.
F2812_XintfBoot.cmd	F2812 memory linker command file to illustrate booting from XINTF Zone 7

9 Detailed Revision History:

<u>**Changes from V1.11 to V1.20**</u>

Changes to Header Files:

a) **DSP281x_Device.h –** Added int64 and Uint64 type definitions.

b) **DSP281x_CpuTimers.h-** Uncommented CPUTimer 1 and CPUTimer 2 code.

Changes to Common Files:

a) **DSP281x_Mcbsp.c** – Added #define CPU_SPD to resolve warnings when compiling example projects.

b) **DSP281x_GlobalPrototypes.c**- Added function prototype for delay_loop() defined in DSP281x_Mcbsp.c. Uncommented CPUTimer1 and CPUTimer2 function prototypes.

c) **DSP281x_CpuTimers.c-** Uncommented CPUTimer1 and CPUTimer2 code, and modified note such that only CPUTimer2 code must be commented out when DSP/BIOS is used.

Changes to examples:

a) **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.

b) **Example_281xMcBSP_SPI_FFDLB.c-** Replaced while(1) with for(;;) and removed #defines for McBSP word length to eliminate warnings when building project.

<u>**Changes from V1.10 to V1.11**</u>

Changes to Header Files:

a) Added release date to all header files.

b) **DSP281x_Ev.h** - In the T2CON_BITS, T3CON_BITS, and T4CON_BITS structure, FREE is now bit 15 and SOFT is now bit 14 to match the User Guide.

Changes to examples:

a) **DSP281x_Mcbsp.c** – Updated init function so clock generator reset occurs in init function in an order corresponding to the User Guide.

b) **Example_281xSci_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. Also clarified comments above the delay loop in the scia_AutobaudLock function as follows:

```
// Note the lower the baud rate the longer
// this delay has to be to allow the other end
// to echo back a character (about 4 characters long)
```

```
        // Make this really long since we are going through all
        // the baud rates.
```

c) Added low-power mode examples in the following directories under the DSP281x_examples directory: lpm_standbywake, lpm_haltwake, and lpm_idlewake.

Changes from V1.00 to V1.10

Changes to Header Files:

c) **DSP281x_DevEmu.h** - DEVICEID_REG type for “all” changed to Uint16 instead Uint32. Also, to match with the User Guide, in DEV_EMU_REGS, PARTID now replaces rsvd1, and a #define REVID DEVICEID was added. This maintains code compatibility with V1.00 and adds support for the User Guide implementation of the device emulation registers. Additionally, a union and struct were created for PARTID_REG and PARTID_BITS respectively.

d) **DSP281x_Ev.h** - In the TXCON_BITS structure, FREE is now bit 15 and SOFT is now bit 14 to match the User Guide.

e) **DSP281x_Adc.h** – Added ADCEXTREF as bit 8 of ADCTRL4_BITS struct.

f) **DSP281x_ECan.h**—Added extern prototype references to CanDoubleRead and CanDoubleWrite to support errata.

g) **DSP281x_Xintf.h** – In XBANK_BITS, BANK is corrected to be 3 bits long instead of 2 bits.

h) **DSP281x_Device.h** – For the sake of completeness, added DSP28_F2811 as a target device.

i) **DSP281x_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.

j) **DSP281x_Headers_BIOS.cmd and DSP281x_Headers_nonBIOS.cmd** – Peripheral Frame 1 and Peripheral Frame 2 comment headings are now above the appropriate peripheral regfiles.

Changes to examples:

d) The root of the default path in all example project files was changed from C:\tidcs\c28\DSP281x\v100\ to C:\tidcs\c28\DSP281x\v110\ to reflect the version change.

e) DSP281x_DefaultIsr.h – This file was included twice – once in the DSP281x_common\include\ directory, and once in the DSP281x_headers\include\ directory. The DSP281x_headers\include\ version was deleted because this file is used for the code examples and is not part of the headers themselves.

f) Example_281xSpi_FFDLB_int.c – Changed comment regarding RXFIFO level set at 31 levels to 8 levels.

g) Example_281xSci_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)`.

h) DSP281x_Mcbsp.c – Updated with new initialization functions for Mcbsp GPIO, data lengths, and general initialization for digital loopback mode. Also added #define for SRG cycle delay required in initialization routine.

i) DSP281x_ECan.c – Removed system initialization functions, corrected notation for TSEGn fields in the comments section for bit timing configuration, and implemented reads/writes using shadow registers where required.

j) **DSP281x_SysCtrl.c** – Added CsmUnlock() function which allows user to unlock the CSM in code, if desired.

k) **DSP281x_GlobalPrototypes.h** – Added extern function prototype for CsmUnlock(). Added extern function prototype for new Mcbsp functions in DSP281x_Mcbsp.c. Also, declared extern function prototype for usDelay.asm here.

l) Mcbsp examples updated to utilize McBSP GPIO initialization function and McBSP initialization functions in the new DSP281x_Mcbsp.c.

m) **Example_281xMcBSP_FFDLB_int.c** – For clarity, updated initialization writes to RCR2 and XCR2 – "132" part of 0x1321 is "don't care" because there is no phase 2 in this configuration.

n) **Example_281xSWPrioritizedDefaultIsr.c** – Fixed numerous typos so that priorities are set properly in the ISR's.

o) **Example_281xECanBack2Back.c** – Incorporated updates to DSP281x_ECan.c in example. Also added a note about CanDoubleWrite and CanDoubleRead when user incorporates code into an application which does not use self-test mode.

p) Added sci_echoback example which transmits and echoes characters through SCI-A via Hyperterminal on the PC.

q) **F2812_EzDSP_RAM_lnk.cmd** – changed line from:

```
   // For BIOS applications add:      DSP281x_Headers_nonBIOS.cmd
```

to the below:

```
   // For BIOS applications add:      DSP281x_Headers_BIOS.cmd
```

r) **DSP281x_Xintf.c** – changed "#if F2812" to "#if DSP38_F2812".

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) **DSP281x_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 DSP281x_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
};
```

b) The following table lists typos in the header files which were not corrected to preserve compatibility with previous versions of the header files.

Table 20. Known Typos in DSP281x Header Files

		Bit Name		
Peripheral	**Register**	**Incorrect**	**Correct**	**Comment**
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	
PieCtrlRegs				
	PIECTRL	PIECRTL	PIECTRL	Typo
SciaRegs, ScibRegs				
	SCIFFTX	TXFFILIL	TXFFIL	Typo
		TXINTCLR	TXFFINTCLR	Align with user guide.
	SCIFFRX	RXFIFST	RXFFST	Typo – Also corrected in user guide