C2804x C/C++ Header Files and Peripheral Examples Quick Start

1 Introduction:

The DSP2804x C/C++ peripheral header files and example projects facilitate writing in C/C++ Code for the Texas Instruments TMS320x2804x 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 '2804x development platform.

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

  These projects can also be migrated to any future '2804x 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 DSP2804x C/C++ peripheral header files.
- Overview of the included peripheral example projects.
- Steps for integrating the peripheral header files into a new or existing project.
- Troubleshooting tips and frequently asked questions.

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 2804x 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.

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

1.1 Revision History

Version 1.30

- This version includes minor corrections and comment fixes to the header files and examples, and also adds separate example folders, DSP2804x_examples_ccsv4, with examples supported by the Eclipse-based Code Composer Studio v4. A detailed revision history can be found in Section 7.

Version 1.20

- This version includes SFO_TI_Build_V5B.lib, which supports all HRPWM configurations and fixes a few typos and minor errors in the DSP2804x header files and examples. A detailed revision history can be found in Section 7.

Version 1.10

- This version fixes various typos and minor errors in the DSP2804x header files and examples. A detailed revision history can be found in Section 7.

Version 1

- This version is the first formal release of the DSP2804x header files and examples. The name has been changed to align with marketing nomenclature of the supported devices.

Beta 1

- This version is the first preliminary customer release of the DSP2804x header files and examples. At this point the headers were named DSP282x.

1.2 Where Files are Located (Directory Structure)

As installed, the *C2804x C/C++ Header Files and Peripheral Examples* is partitioned into a well-defined directory structure. By default, the source code is installed into the c:\tidcs\c28\DSP2804x\<version> directory.

Table 1 describes the contents of the main directories used by DSP2804x:

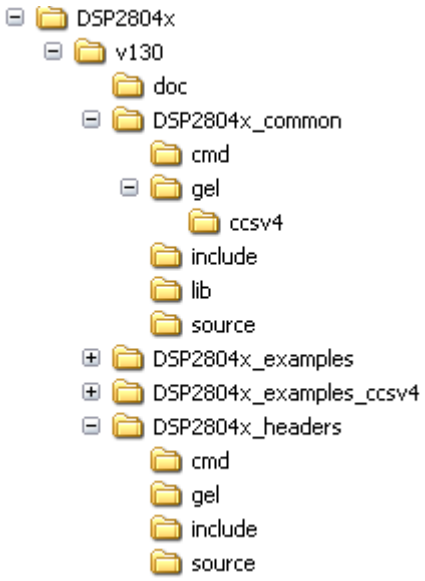


Table 1. DSP2804x Main Directory Structure

Directory	Description
<base>	Base install directory. By default this is c:\tidcs\c28\DSP2804x\<version>. 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>\DSP2804x_headers	Files required to incorporate the peripheral header files into a project. The header files use the bit-field structure approach described in Section 2. Integrating the header files into a new or existing project is described in Section 4.
<base>\DSP2804x_examples	Example Code Composer Studio projects based on the DSP2804x header files. These example projects illustrate how to configure many of the '2804x on-chip peripherals. An overview of the examples is given in Section 3.
<base>\DSP2804x_examples_ccsv4	Example Code Composer Studio v4 projects compiled with floating point unit *enabled*. These example are identical to those in the \DSP2804x_examples directory, but are generated for CCSv4 and cannot be run in CCSv3.x. An overview of the examples is given in Section 3.
<base>DSP2804x_common	Common source files shared across a number of the DSP2804x example projects to illustrate how to perform tasks using the DSP2804x header file approach. Use of these files is optional, but may be useful in new projects. A list of these files is in Section 6.

Under the *DSP2804x_headers* and *DSP2804x_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. DSP2804x Sub-Directory Structure

Sub-Directory	Description
DSP2804x_headers\cmd	Linker command files that allocate the bit-field structures described in Section 2.
DSP2804x_headers\source	Source files required to incorporate the header files into a new or existing project.

DSP2804x_headers\include	Header files for each of the 2804x on-chip peripherals.

DSP2804x_common\cmd	Example memory command files that allocate memory on the ‘2804x devices.
DSP2804x_common\include	Common .h files that are used by the peripheral examples.
DSP2804x_common\source	Common .c files that are used by the peripheral examples.
DSP2804x_common\lib	Common library (.lib) files that are used by the peripheral examples.
DSP2804x_common\gel	Code Composer Studio v3.x GEL files for each device. These are optional.
DSP2804x_common\gel\ccsv4	Code Composer Studio v4.x GEL files for each device. These are optional.

2 Understanding The Peripheral Bit-Field Structure Approach

The *DSP2804x C/C++ Header Files and Peripheral Examples in C* use a bit-field structure approach for mapping and accessing peripheral registers on the TI '2804x 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.

3 Peripheral Example Projects

In the *DSP2804x_examples*\ directory of *C2804x C/C++ Header Files and Peripheral Examples in C/C++* there are several example projects that use the DSP2804x header files to configure the on-chip peripherals. A listing of the examples is included in Section 3.4.

3.1 Getting Started

3.1.1 Getting Started in Code Composer Studio v3.x

To get started, follow these steps to load the DSP2804x CPU-Timer example. Other examples are set-up in a similar manner.

1. **Have a 2804x hardware platform connected to a host with Code Composer Studio installed.**

   NOTE: As supplied, the example projects are built for the '28044 device. If you are using another device within the '2804x family, the memory definition in the linker command file (.cmd) will need to be modified and the project rebuilt.

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

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

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

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

   b. Browse to the CPU Timer example directory: *DSP2804x_examples\cpu_timer*

   c. Select *Example_2804xCpuTimer.gel* and click on *open*.

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

      DSP2804x CpuTimerExample-> Load_and_Build_Project
      This will load the project and build compile the project.

3. **Review the comments at the top of the main source file: Example_2804xCpuTimer.c.**

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

4. Perform any hardware setup required by the example.

Perform any hardware setup indicated by the comments in the main source. The DSP2804x CPU-Timer example only requires that the hardware be setup for "Boot to SARAM" mode. Other examples may require additional hardware configuration such as connecting pins together or pulling a pin high or low.

Table 3 shows a listing of the boot mode pin settings for your reference. Refer to the documentation for your hardware platform for information on configuring the boot mode pins. The '2804x boot modes are identical to those on the 280x. Refer to the *TMS320x280x Boot ROM Reference Guide (SPRU722) for more information*.

Table 3. 2804x Boot Mode Settings

GPIO18	GPIO29	GPIO34	Mode
1	1	1	Boot to flash 0x3F7FF6
1	1	0	Call SCI-A boot loader
1	0	1	Call SPI-A boot loader
1	0	0	Call I2C boot loader
0	1	1	Reserved (Was 280x eCAN-A boot loader)
0	1	0	Boot to M0 SARAM 0x000000
0	0	1	Boot to OTP 0x3D7800
0	0	0	Call parallel boot loader

5. Load the code

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

DSP2804x 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*.

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

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

If you wish to modify the examples it is suggested that you make a copy of the entire DSP2804x 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 3.2 and 3.3 describe the structure and flow of the examples in more detail.

8. **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 *DSP2804x_headers* directory and shared source in the *DSP2804x_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.

3.1.2 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.

9. **Review the comments at the top of the main source file: Example_2804xCpuTimer.c.**

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

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

    Perform any hardware setup indicated by the comments in the main source. The DSP2804x CPU-Timer example only requires that the hardware be setup for "Boot to SARAM" mode.

Other examples may require additional hardware configuration such as connecting pins together or pulling a pin high or low.

Table 3 shows a listing of the boot mode pin settings for your reference. Refer to the documentation for your hardware platform for information on configuring the boot mode pins. The '2804x boot modes are identical to those on the 280x. Refer to the *TMS320x280x Boot ROM Reference Guide (SPRU722) for more information*.

Table 4. 2804x Boot Mode Settings

GPIO18	GPIO29	GPIO34	Mode
1	1	1	Boot to flash 0x3F7FF6
1	1	0	Call SCI-A boot loader
1	0	1	Call SPI-A boot loader
1	0	0	Call I2C boot loader
0	1	1	Reserved (Was 280x eCAN-A boot loader)
0	1	0	Boot to M0 SARAM 0x000000
0	0	1	Boot to OTP 0x3D7800
0	0	0	Call parallel boot loader

3. 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.

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

11. 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.

5. 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 3.2 and 3.3 describe the structure and flow of the examples in more detail.

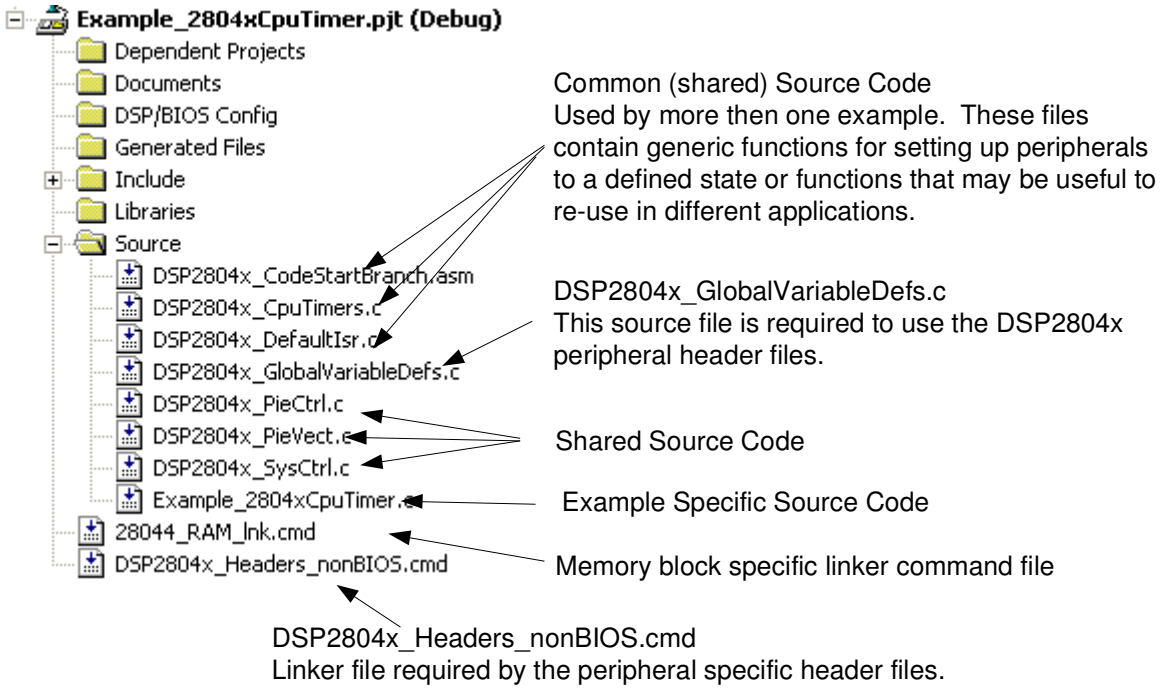
6. When done, delete the project from the Code Composer Studio v4 workspace.

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.

3.2 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.

3.2.1 Include Files

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

```
/*************************************************************************
* DSP2804x_examples\cpu_timer\Example_2804xCpuTimer.c
*************************************************************************/

#include "DSP2804x_Device.h"     // DSP2804x Headerfile Include File
#include "DSP2804x_Examples.h"   // DSP2804x Examples Include File
```

- **DSP2804x_Device.h**

  This header file is required to use the DSP2804x 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 *<base>\DSP2804x_headers\include* directory.

- **DSP2804x_Examples.h**

  This header file defines parameters that are used by the example code. This file is not required to use just the DSP2804x peripheral header files but is required by some of the common source files. This file is found in the *<base>\DSP2804x_common\include* directory.

3.2.2 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.

- **DSP2804x_GlobalVariableDefs.c**

  Any project that uses the DSP2804x peripheral header files must include this source file. In this file are the declarations for the peripheral register structure variables and data section assignments. This file is found in the *<base>\DSP2804x_headers\source* directory.

- **Example specific source code:**

  Files that are specific to a particular example have the prefix Example_2804x on their filename. For example Example_2804xCpuTimer.c is specific to the CPU Timer example and not used for any other example. Example specific files are located in the *<base>\DSP2804x_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 DSP2804x_common\source directory. Users may choose to incorporate none, some, or the entire shared source into their own new or existing projects.

3.2.3 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 DSP2804x peripheral header files.

- **Memory block linker allocation:**

The linker files shown in Table 5 are used to assign sections to memory blocks on the device. These linker files are located in the *<base>\DSP2804x_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
28044_RAM_lnk.cmd	DSP2804x_common\cmd	28044 memory linker command file. Includes all of the internal SARAM blocks on a 28044 device. Does not include flash or OTP blocks.
F28044.cmd	DSP2804x_common\cmd	F28044 memory linker command file. Includes all Flash, OTP and CSM password protected memory locations.

- **DSP2804x header file structure data section allocation:**

  Any project that uses the DSP2804x 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. DSP2804x Peripheral Header Linker Command File

DSP2804x Peripheral Header File Linker Command File	Location	Description
DSP2804x_Headers_BIOS.cmd	DSP2804x_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 4.2.
DSP2804x_Headers_nonBIOS.cmd	DSP2804x_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 4.2.

3.3 Example Program Flow

All of the example programs follow a similar recommended flow for setting up the 2804x devices. Figure 1 outlines this basic flow:

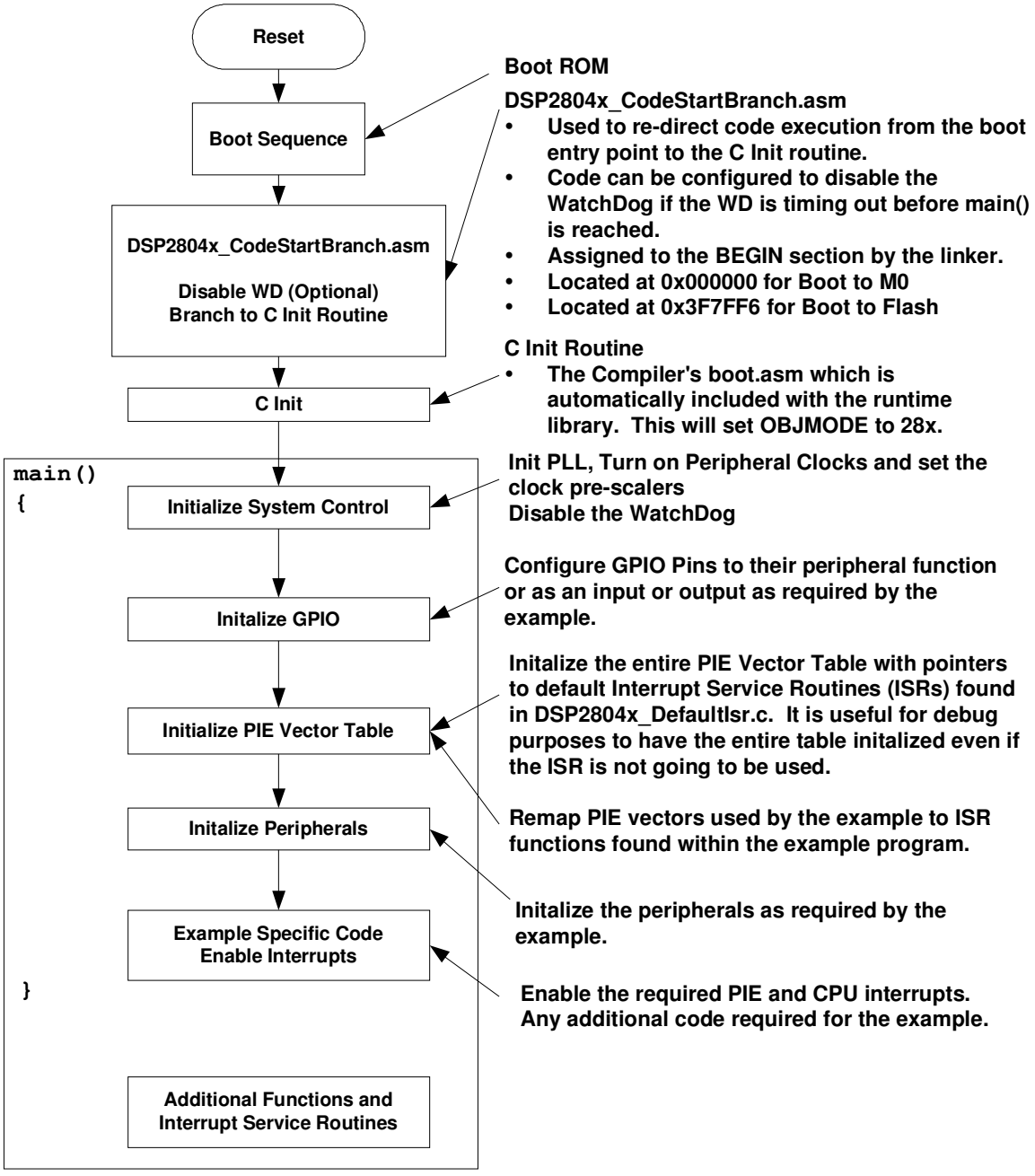


Figure 1. Flow for Example Programs

3.4 Included Examples:

Table 7. Included Examples

Example	Description
adc_seq_ovd_tests	ADC test using the sequencer override feature.
adc_seqmode_test	ADC Seq Mode Test. Channel A0 is converted forever and logged in a buffer
adc_soc	ADC example to convert two channels: ADCINA3 and ADCINA2. Interrupts are enabled and PWM1 is configured to generate a periodic ADC SOC on SEQ1.
cpu_timer	Configures CPU Timer0 and increments a count each time the ISR is serviced.
epwm_Achannel_updown	Configures GPIO0-15 for ePWM1-16 A channels (EPWMxA)
epwm_deadband	Example deadband generation via ePWM3
epwm_timer_interrupts	Starts ePWM1-ePWM6 timers. Every period an interrupt is taken for each ePWM.
epwm_trip_zone	Uses the trip zone signals to set the ePWM signals to a particular state.
epwm_up_aq	Generate a PWM waveform using an up count time base ePWM1-ePWM3 are used.
epwm_updown_aq	Generate a PWM waveform using an up/down time base. ePWM1 – ePWM3 are used.
external_interrupt	Configures GPIO0 as XINT1 and GPIO1 as XINT2. The interrupts are fired by toggling GPIO30 and GPIO31 which are connected to XINT1 (GPIO0) and XINT2 (GPIO1) externally by the user.
flash	ePWM timer interrupt project moved from SARAM to Flash. Includes steps that were used to convert the project from SARAM to Flash. Some interrupt service routines are copied from FLASH to SARAM for faster execution.
gpio_setup	Three examples of different pinout configurations.
gpio_toggle	Toggles all of the I/O pins using different methods – DATA, SET/CLEAR and TOGGLE registers. The pins can be observed using an oscilloscope.
hrpwm	Sets up ePWM1-ePWM4 and controls the edge of output A using the HRPWM extension. Both rising edge and falling edge are controlled.
hrpwm_sfo	Use TI's MEP Scale Factor Optimizer (SFO) library to change the HRPWM. This version of the SFO library supports HRPWM on ePWM channels 1-4 only.
hrpwm_sfo_v5	Use TI's MEP Scale Factor Optimizer (SFO) library version 5 to change the HRPWM. This version of the SFO library supports HRPWM on up to 16 ePWM channels (if available)
hrpwm_slider	This is the same as the hrpwm example except the control of CMPAHR is now controlled by the user via a slider bar. The included .gel file sets up the slider.
i2c_eeprom	Communicate with the EEPROM via I2C
lpm_halt	Puts device into low power halt mode. GPIO0 is configured to wake the device from halt when an external high-low-high pulse is applied to it.
lpm_idle	Puts device into low power idle mode. GPIO0 is configured as XINT1 pin. When an XINT1 interrupt occurs due to a falling edge on GPIO0, the device is woken from idle.
lpm_standby	Puts device into low power standby mode. GPIO0 is configured to wake the device from halt when an external high-low-high pulse is applied to it.
sci_echoback	SCI-A example that can be used to echoback to a terminal program such as hyperterminal. A transceiver and a connection to a PC is required.
scia_loopback	SCI-A example 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.
scia_loopback_interrupts	SCI-A 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-A example that uses the peripherals loop-back test mode to send data.

Included Examples Continued…

spi_loopback_interrupts	SPI-A example that uses the peripherals loop-back test mode to send data. Both interrupts and FIFOs are used in this example.
sw_prioritized_interrupts	The standard hardware prioritization of interrupts can be used for most applications. This example shows a method for software to re-prioritize interrupts if required.
watchdog	Illustrates feeding the dog and re-directing the watchdog to an interrupt.

3.5 Executing the Examples From Flash

Most of the DSP2804x examples execute from SARAM in “boot to SARAM” mode. One example, *DSP2804x_examples\Flash*, executes from flash memory in “boot to flash” mode. This example is the PWM timer interrupt example with the following changes made to execute out of flash:

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

   Remove 28044_RAM_lnk.cmdfrom the project and add F28044.cmd. F28044.cmd is located in the *<base>DSP2804x_common\cmd\ directory*.

2. **Add the *DSP2804x_common\source\DSP2804x_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. The 2804x CSM is identical to that on the 280x. For more information on the CSM refer to the *TMS320x280x, 2801x, 2804x System Control and Interrupts Reference Guide* (spru712).

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

   In particular, the flash wait state initialization routine must be executed out of SARAM. In the DSP2804x 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.

```
/*********************************************************************
* DSP2804x_common\source\DSP2804x_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:

```
/*********************************************************************
* DSP2804x_common\include\F28044.cmd
*********************************************************************/
SECTIONS
{
   ramfuncs    : LOAD = FLASHD,
                 RUN = RAML0,
                 LOAD_START(_RamfuncsLoadStart),
                 LOAD_END(_RamfuncsLoadEnd),
                 RUN_START(_RamfuncsRunStart),
                 PAGE = 0
}
```

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

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

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

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

a. Add the file *DSP2804x_common\source\DSP2804x_MemCopy.c* to the project.

b. Add the following function prototype to the example source code. This is done for you in the *DSP2804x_Examples.h* file.

```
/*************************************************************
* DSP2804x_common\include\DSP2804x_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 DSP2804x example code this has is already done in *DSP2804x_Examples.h*.

```
/*************************************************************
* DSP2804x_common\include\DSP2804x_GlobalPrototypes.h
*************************************************************/

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

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

```
/*************************************************************
* DSP2804x_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.

```
/*************************************************************
* DSP2804x 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. 2804x Boot Mode Settings

GPIO18	GPIO29	GPIO34	Mode
1	1	1	Boot to flash 0x3F7FF6
1	1	0	Call SCI-A boot loader
1	0	1	Call SPI-A boot loader
1	0	0	Call I2C boot loader
0	1	1	**Reserved (280x eCAN-A boot loader)**
0	1	0	Boot to M0 SARAM 0x000000
0	0	1	Boot to OTP 0x3D7800
0	0	0	Call parallel boot loader

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

   The 2804x boot modes are identical to the 280x. For more information refer to the *TMS320x280x, 2801x, 2804x DSP Boot ROM Reference Guide* (SPRU722).

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

   In Code Composer Studio v4.0, when code is loaded into the device during debug, it automatically programs to flash memory.

   This can 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 v3.0 can be used.

   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.

4 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.

4.1 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 3.

2. **Create a copy of the source files you want to use.**
   - *DSP2804x_headers:* code required to incorporate the header files into your project
   - *DSP2804x_common*: shared source code much of which is used in the example projects.
   - *DSP2804x_examples*: example projects that use the header files and shared code.

4.2 Including the DSP2804x Peripheral Header Files

Including the DSP2804x 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 "DSP2804x_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 "DSP2804x_Device.h"
```

2. **Edit DSP2804x_Device.h and select the target you are building for:**

   In the below example, the file is configured to build for the '28044 device.

```
/*************************************************************
* DSP2804x_headers\include\DSP2804x_Device.h
*************************************************************/

#define    TARGET                1
#define    DSP28_28044           TARGET
```

   By default, the '28044 device is selected.

3. Add the source file *DSP2804x_GlobalVariableDefs.c* to the project.

This file is found in the *DSP2804x_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 DSP2804x header linker command file to the project.

As described in Section 2, when using the DSP2804x 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 *DSP2804x_headers\cmd* must be included in your project:

– For non-DSP/BIOS† projects: *DSP2804x_Headers_nonBIOS.cmd*

– For DSP/BIOS projects: *DSP2804x_Headers_BIOS.cmd*

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

Code Composer Studio V2.2 and later:

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

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

Code Composer Studio prior to V2.2

Prior to CCS 2.2, each project contained only one main linker command file. This file can, however, call additional .cmd files as needed. To include the required memory allocations for the DSP2804x header files, perform the following two steps:

1) Update the project's main linker command (.cmd) file to call one of the supplied DSP2804x peripheral structure linker command files using the -l option.

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

/* Use this include file only for non-BIOS applications */
-l DSP2804x_Headers_nonBIOS.cmd
/* Use this include file only for BIOS applications */
/* -l DSP2804x_Headers_BIOS.cmd */
```

† DSP/BIOS is a trademark of Texas Instruments

2) Add the directory path to the DSP2804x 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 *DSP2804x_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 *DSP2804x_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>\DSP2804x_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 2804x header files, this is INSTALLROOT_2804X_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 DSP2804x header files to your project.

Code Composer Studio 3.x:

To specify the directory where the header files are located:

a. Open the menu: Project->Build Options

b. Select the *Compiler tab*

c. Select *pre-processor*.

d. In the *Include Search Path*, add the directory path to the location of *DSP2804x_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 *DSP2804x_headers\include* on your 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.
  - **-e Select *Basic* and enter Code Entry Point –e**

Defines a global symbol that specifies the primary entry point for the output module. For the DSP2804x examples, this is the symbol "code_start". This symbol is defined in the DSP2804x_common\source\DSP2804x_CodeStartBranch.asm file. When you load the code in Code Composer Studio, the debugger will set the PC to the address of this symbol. If you do not define a entry point using the –e option, then the linker will use _c_int00 by default.

4.3 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 "DSP2804x_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 "DSP2804x_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 DSP2804x_common/include on your system.
Use a semicolon between directories.

For example the directory path for the included projects is:
..\..\DSP2804x_headers\include;..\..\DSP2804x_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 *DSP2804x_headers\include* on your system.

3. Add a linker command file to your project.

The following memory linker .cmd files are provided as examples in the *DSP2804x_common\cmd* directory. For getting started the basic *28044_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
28044_RAM_lnk.cmd	F28044 memory linker example. Only allocates SARAM locations.
F28044.cmd	F28044 memory linker command file. Includes all Flash, OTP and CSM password protected memory locations.

4. Set the CPU Frequency

In the *DSP2804x_common\include\DSP2804x_Examples.h* file specify the proper CPU frequency. Some examples are included in the file.

```
/*-----------------------------------------------------------------
* DSP2804x_common\include\DSP2804x_Examples.h
-----------------------------------------------------------------*/

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

5. Add desired common source files to the project.

The common source files are found in the *DSP2804x_common\source* directory.

6. Include .c files for the PIE.

Since all catalog '2804x 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 6.2.1.

5 Troubleshooting Tips & Frequently Asked Questions

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

  EALLOW; is a macro defined in DSP2804x_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 preliminary list of protected registers, refer to *TMS320x280x Control and Interrupts Reference Guide* (SPRU712).

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

  There are a few things to check:

  - Peripheral registers cannot be modified or unless the clock to the specific peripheral is enabled. The function InitPeripheralClocks() in the DSP2804x_common\source directory shows an example of enabling the peripheral clocks.
  - Some peripherals are not present on all 2804x family derivatives. Refer to the device datasheet for information on which peripherals are available.
  - The EALLOW bit protects some registers from spurious writes by the CPU. If your program seems unable to write to a register, then check to see if it is EALLOW protected. If it is, then enable access using the EALLOW assembly instruction. *TMS320x280x Control and Interrupts Reference Guide* (SPRU712) for a preliminary list of EALLOW protected registers.

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

  In this case most likely the code security module is locked and thus the protected memory locations are reading back all 0x0000. Refer to the *TMS320x280x Control and Interrupts Reference Guide* (SPRU712) for information on the code security module.

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

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

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

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

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

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

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

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

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

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

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

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

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

  This warning is given when a symbol other then _c_int00 is defined as the code entry point of the project. For these examples, the symbol code_start is the first code that is executed after exiting the boot ROM code and thus is defined as the entry point via the –e linker option. This symbol is defined in the DSP2804x_CodeStartBranch.asm file. The entry point symbol is used by the debugger and by the hex utility. When you load the code, CCS will set the PC to the entry point symbol. By default, this is the _c_int00 symbol which marks the start of the C initialization routine. For the DSP2804x examples, the code_start symbol is used instead. Refer to the source code for more information.

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

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

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

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

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

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

  – Compiler Tab, Preprocessor: Include search path

  – Linker Tab, Basic: Library search path

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

  Refer to section 4 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.

5.1 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:

5.1.1 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!
```

5.1.2 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.

6 Packet Contents:

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

6.1 Header File Support – DSP2804x_headers

The DSP2804x header files are located in the *<base>\DSP2804x_headers* directory.

6.1.1 DSP2804x Header Files – Main Files

The following files must be added to any project that uses the DSP2804x header files. Refer to section 4.2 for information on incorporating the header files into a new or existing project.

Table 10. DSP2804x Header Files – Main Files

File	Location	Description
DSP2804x_Device.h	DSP2804x_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 4.2.
DSP2804x_GlobalVariableDefs.c	DSP2804x_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 4.2.
DSP2804x_Headers_BIOS.cmd	DSP2804x_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 4.2.
DSP2804x_Headers_nonBIOS.cmd	DSP2804x_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 4.2.

6.1.2 DSP2804x 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 2804x devices. These files are automatically included in the project by including *DSP2804x_Device.h.* Refer to section 4.2 for more information on incorporating the header files into a new or existing project.

Table 11. DSP2804x Header File Bit-Field & Register Structure Definition Files

File	Location	Description
DSP2804x_Adc.h	DSP2804x_headers\include	ADC register structure and bit-field definitions.
DSP2804x_CpuTimers.h	DSP2804x_headers\include	CPU-Timer register structure and bit-field definitions.
DSP2804x_DevEmu.h	DSP2804x_headers\include	Emulation register definitions
DSP2804x_EPwm.h	DSP2804x_headers\include	ePWM register structures and bit-field definitions.
DSP2804x_Gpio.h	DSP2804x_headers\include	General Purpose I/O (GPIO) register structures and bit-field definitions.
DSP2804x_I2c.h	DSP2804x_headers\include	I2C register structure and bit-field definitions.
DSP2804x_PieCtrl.h	DSP2804x_headers\include	PIE control register structure and bit-field definitions.
DSP2804x_PieVect.h	DSP2804x_headers\include	Structure definition for the entire PIE vector table.
DSP2804x_Sci.h	DSP2804x_headers\include	SCI register structure and bit-field definitions.
DSP2804x_Spi.h	DSP2804x_headers\include	SPI register structure and bit-field definitions.
DSP2804x_SysCtrl.h	DSP2804x_headers\include	System register definitions. Includes Watchdog, PLL, CSM, Flash/OTP, Clock registers.
DSP2804x_XIntrupt.h	DSP2804x_headers\include	External interrupt register structure and bit-field definitions.

6.1.3 Code Composer .gel Files

The following Code Composer Studio .gel files are included for use with the DSP2804x Header File peripheral register structures.

Table 12. DSP2804x Included GEL Files

File	Location	Description
DSP2804x_Peripheral.gel	DSP2804x_headers\gel	This is relevant for CCSv3.3 only. Provides GEL pull-down menus to load the DSP2804x data structures into the watch window. You may want to have CCS load this file automatically by adding a GEL_LoadGel("<base>\DSP2804x_headers\gel\DSP2804x_peripheral.gel") function to the standard F28044.gel that was included with CCS.

6.1.4 Variable Names and Data Sections

This section is a summary of the variable names and data sections allocated by the DSP2804x_headers\source\DSP2804x_GlobalVariableDefs.c file. Note that all peripherals may not be available on a particular 2804x device. Refer to the device datasheet for the peripheral mix available on each 2804x family derivative.

Table 13. DSP2804x Variable Names and Data Sections

Peripheral	Starting Address	Structure Variable Name
ADC	0x007100	AdcRegs
ADC Mirrored Result Registers	0x000B00	AdcMirror
Code Security Module	0x000AE0	CsmRegs
Code Security Module Password Locations	0x3F7FF8- 0x3F7FFF	CsmPwl
CPU Timer 0	0x000C00	CpuTimer0Regs
Device and Emulation Registers	0x000880	DevEmuRegs
ePWM1	0x006800	EPwm1Regs
ePWM2	0x006840	EPwm2Regs
ePWM3	0x006880	EPwm3Regs
ePWM4	0x0068C0	EPwm4Regs
ePWM5	0x006900	EPwm5Regs
ePWM6	0x006940	EPwm6Regs
ePWM7	0x006980	EPwm7Regs
ePWM8	0x0069C0	EPwm8Regs
ePWM9	0x006600	EPwm9Regs
ePWM10	0x006640	EPwm10Regs
ePWM11	0x006680	EPwm11Regs
ePWM12	0x0066C0	EPwm12Regs
ePWM13	0x006700	EPwm13Regs
ePWM14	0x006740	EPwm14Regs
ePWM15	0x006780	EPwm15Regs
ePWM16	0x0067C0	EPwm16Regs
External Interrupt Registers	0x007070,	XIntruptRegs
Flash & OTP Configuration Registers	0x000A80	FlashRegs
General Purpose I/O Data Registers	0x006fC0	GpioDataRegs
General Purpose Control Registers	0x006F80	GpioCtrlRegs
General Purpose Interrupt Registers	0x006fE0	GpioIntRegs
I2C	0x007900	I2caRegs
PIE Control	0x000CE0	PieCtrlRegs
SCI-A	0x007050	SciaRegs
SPI-A	0x007040	SpiaRegs

6.2 Common Example Code – DSP2804x_common

6.2.1 Peripheral Interrupt Expansion (PIE) Block Support

In addition to the register definitions defined in DSP2804x_PieCtrl.h, this packet provides the basic ISR structure for the PIE block. These files are:

Table 14. Basic PIE Block Specific Support Files

File	Location	Description
DSP2804x_DefaultIsr.c	DSP2804x_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.**
DSP2804x_DefaultIsr.h	DSP2804x_common\include	Function prototype statements for the ISRs in DSP2804x_DefaultIsr.c. **Note: This file is not used for DSP/BIOS projects.**
DSP2804x_PieVect.c	DSP2804x_common\source	Creates an instance of the PIE vector table structure initialized with pointers to the ISR functions in DSP2804x_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 *DSP2804x_examples\sw_prioritized_interrupts* for more information.

Table 15. Software Prioritized Interrupt PIE Block Specific Support Files

File	Location	Description
DSP2804x_SWPrioritizedDefaultIsr.c	DSP2804x_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.**
DSP2804x_SWPrioritizedIsrLevels.h	DSP2804x_common\include	Function prototype statements for the ISRs in DSP2804x_DefaultIsr.c. **Note: This file is not used for DSP/BIOS projects.**
DSP2804x_SWPrioritizedPieVect.c	DSP2804x_common\source	Creates an instance of the PIE vector table structure initialized with pointers to the default ISR functions that are included in DSP2804x_DefaultIsr.c. This instance can be copied to the PIE vector table in order to initialize it with the default ISR locations.

6.2.2 Peripheral Specific Files

Several peripheral specific initialization routines and support functions are included in the peripheral .c source files in the *DSP2804x_common\src* directory. These files include:

Table 16. Included Peripheral Specific Files

File	Description
DSP2804x_GlobalPrototypes.h	Function prototypes for the peripheral specific functions included in these files.
DSP2804x_Adc.c	ADC specific functions and macros.
DSP2804x_CpuTimers.c	CPU-Timer specific functions and macros.
DSP2804x_EPwm.c	ePWM module specific functions and macros.
DSP2804x_EPwm_defines.h	#define macros that are used for the ePWM examples
DSP2804x_Gpio.c	General-purpose IO (GPIO) specific functions and macros.
DSP2804x_I2C.c	I2C specific functions and macros.
DSP2804x_I2c_defines.h	#define macros that are used for the I2C examples
DSP2804x_PieCtrl.c	PIE control specific functions and macros.
DSP2804x_Sci.c	SCI specific functions and macros.
DSP2804x_Spi.c	SPI specific functions and macros.
DSP2804x_SysCtrl.c	System control (watchdog, clock, PLL etc) specific functions and macros.

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

6.2.3 Utility Function Source Files

Table 17. Included Utility Function Source Files

File	Description
DSP2804x_CodeStartBranch.asm	Branch to the start of code execution. This is used to re-direct code execution when booting to Flash, OTP or M0 SARAM memory. An option to disable the watchdog before the C init routine is included.
DSP2804x_DBGIER.asm	Assembly function to manipulate the DEBIER register from C.
DSP2804x_DisInt.asm	Disable interrupt and restore interrupt functions. These functions allow you to disable INTM and DBGM and then later restore their state.
DSP2804x_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 *DSP2804x_examples\adc* for an example of its use.
DSP2804x_CSMPasswords.asm	Include in a project to program the code security module passwords and reserved locations.

6.2.4 Example Linker .cmd files

Example memory linker command files are located in the *DSP2804x_common\cmd* directory. For getting started using the 2804x devices, the basic 28044_RAM_lnk.cmd file is suggested and used by many of the included examples.

On 2804x devices, the SARAM blocks L0 and L1 are mirrored. For simplicity these memory maps only include one instance of these memory blocks.

Table 18. Included Main Linker Command Files

Main Liner Command File Examples	Description
28044_RAM_lnk.cmd	F28044 memory linker example. Only allocates SARAM locations.
F28044.cmd	F28044 memory linker command file. Includes all Flash, OTP and CSM password protected memory locations.

6.2.5 Example Library .lib Files

Example library files are located in the *DSP2804x_common\lib* directory. For this release the IQMath library is included for use in the example projects. Please refer to the *C28x IQMath Library - A Virtual Floating Point Engine (*SPRC087) for more information on IQMath and the most recent IQMath library. The SFO libraries are also included for use in the example projects. Please refer to *TMS320x28xx, 28xxx HRPWM Reference Guide* (SPRU924) for more information on SFO library usage and the HRPWM module.

Table 19. Included Library Files

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

7 Detailed Revision History:

Changes from V1.20 to V1.30

Changes to Header Files:

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

b) **DSP2804x_Device.h –** Added int64 and Uint64 typedef definitions.

c) **DSP2804x_Spi.h-** Changed SPIPRI register bit 6 to reserved bit.

d) **DSP2804x_Gpio.h-** In GPIO_DATA_REGS struct changed GPBPUD_REG for GPBDAT register to GPBDAT_REG.

Changes to Common Files:

e) **DSP2804x_CpuTimers.c –** Updated comments to indicate only CpuTimer2 is reserved for DSP/BIOS use. User must comment out CpuTimer2 code when using DSP/BIOS.

f) **DSP2804x_I2c_defines.h-** Fixed typo for DSP280x_I2CDEINFES_H and replaced with DSP280x_I2CDEFINES_H.

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

Changes to Example Files:

h) **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.

i) **Example_2804xHRPWM.c and Example_2804xHRPWM_slider.c** – Changed initialization code to set TBPRD register to period-1 instead of period to achieve correct period and duty cycle.

j) **Example_2804xHaltWake.c**- Update WAKE_ISR to toggle GPIO1 instead of set GPIO1 in ISR. Additionally, updated description comments for wakeup.

k) **Example_2804xHRPWM_SFO_V5.c-** Added line of code before calling MepDis() function to enable HRPWM logic for the channel first.

l) **Added DSP2804x_examples_ccsv4 directories** - Added directories for CCSv4.x projects. The example projects in these directories are identical to those found in the normal CCSv3.x DSP2804x_examples directory with the exception that the examples now support the Code Composer Studio v4.x project folder format instead of the Code Composer Studio v3.x PJT files. The example gel files have also been removed for the CCSv4 example projects because the gel file functions used in the example gels are no longer supported.

Changes from V1.10 to V1.20

Changes to Header Files:

m) **DSP2804x_DevEmu.h –** Removed MONPRIV, EMU0SEL, and EMU1SEL bits in the DEVICECNF register.

Changes to Common Files:

n) **DSP2804x_SWPrioritizedDefaultIsr.c –** Fixed some PIEIER number typos.

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

p) **DSP2804x_SysCtrl.h –** Added EALLOW access to code which sets CLKINDIV bit to 0. Also removed "!" from code which sets CLKINDIV to divider value.

Changes from V1.00 to V1.10

Changes to Header Files:

a) **Gel files (f28044.gel and sim28044.gel)** – configuration of addressing modes updated such that AMODE and OBJMODE are set by manipulating the ST register instead of by directly setting AMODE and OBJMODE via legacy method.

b) **DSP2804x_Headers_BIOS.cmd and DSP2804x_Headers_nonBIOS.cmd** – Peripheral Frame 1 and Peripheral Frame 2 comment headings are now above the appropriate peripheral regfiles.

c) **DSP2804x_SysCtrl.h** – Edited XCLKOUT (XCLK) register bit descriptions to read "reserved for TI internal use only" to align with System Control User Guide (with the exception of XCLKOUTDIV bits).

Changes to Example Files:

a) **Example_2804xSpi_FFDLB_int.c** – Changed comment regarding RXFIFO level set at 31 levels to 8 levels.

b) All HRPWM example .pjt files – Fixed pathnames for Debug and Release folders so that they are generated within the hrpwm example folders instead of the global DSP280x_examples folder.

c) Added 3 low power mode examples – Example_2804xHaltWake.c in lpm_haltwake, Example_2804xIdleWake.c in lpm_idlewake, and Example_2804xStandbyWake.c in lpm_standbywake.

d) **Example_2804xSci_FFDLB_int.c** – fixed baud rate calculation defines at the top of file by adding: `#define LSPCLK_FREQ CPU_FREQ/4` and fixing the SCI baud rate period formula such that: `#define SCI_PRD LSPCLK_FREQ/((SCI_FREQ*8)-1)`.

e) **DSP2804x_SysCtrl.c** – Added CsmUnlock() function which allows user to unlock the CSM in code, if desired.

f) **DSP2804x_GlobalPrototypes.h** – Added extern function prototype for CsmUnlock().

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

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

i) The root of the default path in all example project files was changed from C:\tidcs\c28\DSP2804x\v100\ to C:\tidcs\c28\DSP2804x\v110\ to reflect the version change.

8 Errata

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

a) **DSP2804x_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 DSP2804x_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
};
```