

2834x C/C++ Header Files and Peripheral Examples Quick Start

1	Device Support:	2
2	Introduction:	2
	2.1 Revision History.....	3
	2.2 Where Files are Located (Directory Structure).....	3
3	Understanding The Peripheral Bit-Field Structure Approach	5
4	Peripheral Example Projects	6
	4.1 Getting Started	6
	4.2 Example Program Structure.....	11
	4.2.1 Include Files.....	12
	4.2.2 Source Code	12
	4.2.3 Linker Command Files	13
	4.2.4 Documentation.....	14
	4.3 Example Program Flow:.....	15
	4.4 Included Examples:	16
	4.5 Converting Floating-Point Compiled Examples to Fixed-Point and Vice Versa	17
5	Steps for Incorporating the Header Files and Sample Code	22
	5.1 Before you begin.....	22
	5.2 Including the DSP2834x Peripheral Header Files	22
	5.3 Including Common Example Code.....	26
6	Troubleshooting Tips & Frequently Asked Questions	29
	6.1 Effects of read-modify-write instructions.	31
	6.1.1 Registers with multiple flag bits in which writing a 1 clears that flag.....	32
	6.1.2 Registers with Volatile Bits.....	32
7	Migration Tips for moving from the TMS320x2833x/TMS320x2823x header files to the TMS320x2834x header files	33
8	Packet Contents:	34
	8.1 Header File Support – DSP2834x_headers	34
	8.1.1 2834x Header Files – Main Files	34
	8.1.2 DSP2834x Header Files – Peripheral Bit-Field and Register Structure Definition Files	36
	8.1.3 Code Composer .gel Files.....	37
	8.1.4 Variable Names and Data Sections.....	37
	8.2 Common Example Code – DSP2834x_common.....	39
	8.2.1 Peripheral Interrupt Expansion (PIE) Block Support	39
	8.2.2 Peripheral Specific Files.....	40
	8.2.3 Utility Function Source Files.....	41
	8.2.4 Example Linker .cmd files	41
	8.2.5 Included Library .lib Files.....	42
	8.2.6 IQmath and FPU fastRTS Library.....	43
9	Detailed Revision History	44

1 Device Support:

This software package supports 2834x devices. This includes the following: TMS320C28346, TMS320C28345, TMS320C28344, TMS320C28343, TMS320C28342, and TMS320C28341.

Throughout this document, TMS320C28346, TMS320C28345, TMS320C28344, TMS320C28343, TMS320C28342, TMS320C28341, and TMS320C2834F (custom-device) are abbreviated as C28346, C28345, C28344, C28343, C28342, C28341, and C2834F respectively.

2 Introduction:

The 2834x C/C++ peripheral header files and example projects facilitate writing in C/C++ Code for the Texas Instruments TMS320x2834x devices. The code can be used as a learning tool or as the basis for a development platform depending on the current needs of the user.

- Learning Tool:

This download includes several example Code Composer Studio™† projects for a ‘2834x development platform. One such platform is the C28346 Experimenter’s Kit.

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 C2834x C/C++ peripheral header files.
- Overview of the included peripheral example projects.
- Steps for integrating the peripheral header files into a new or existing project.
- Troubleshooting tips and frequently asked questions.
- Migration tips for users moving from the 2833x/2823x header files to the 2834x header files.

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

Finally, this document does not provide a tutorial on writing C code, using Code Composer Studio, or the C28x Compiler and Assembler. It is assumed that the reader already has a 2834x hardware platform setup and connected to a host with Code Composer Studio installed. The user should have a basic understanding of how to use Code Composer Studio to download code through JTAG and perform basic debug operations.

2.1 Revision History

Version 1.11 – controlSUITE update

- ❑ This update makes numerous improvements to the Code Composer Studio 4 projects.
- ❑ A detailed revision history can be found in Section 9.

Version 1.10

- ❑ This version includes minor corrections and comment fixes to the header files and examples, adds the 28345, 28344, 28342, and 28341 devices, and also adds a separate example folder, DSP2834x_examples_ccsv4, with examples supported by the Eclipse-based Code Composer Studio v4. A detailed revision history can be found in Section 9.

Version 1

- ❑ This version is the first release of the 2834x header files and examples.

2.2 Where Files are Located (Directory Structure)

As installed, the *C2834x 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:\ti\controlSUITE\device_support\c2834x\<version> directory.

The included example projects also link in the following math libraries:

- fastRTS library located at c:\ti\controlSUITE\libs\math\FPUfastRTS\V100
- IQmath library located at c:\ti\controlSUITE\libs\math\IQmath\v15c

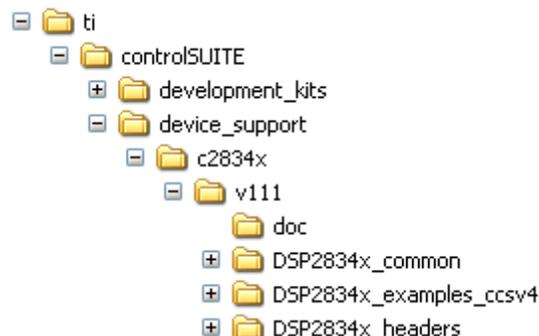


Table 1 describes the contents of the main directories used by 2834x header files and peripheral examples:

Table 1. DSP2834x Main Directory Structure

Directory	Description
<base>	Base install directory. By default this is c:\tidcs\c28\DSP2834x\<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>\DSP2834x_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>\DSP2834x_examples_ccsv4	Example Code Composer Studio v4 projects compiled with floating point unit <i>enabled</i> . These example are identical to those in the \DSP2834x_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>DSP2834x_common	Common source files shared across example projects to illustrate how to perform tasks using header file approach. Use of these files is optional, but may be useful in new projects. A list of these files is in Section 8.

Under the *DSP2834x_headers* and *DSP2834x_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. DSP2834x Sub-Directory Structure

Sub-Directory	Description
DSP2834x_headers\cmd	Linker command files that allocate the bit-field structures described in Section 3.
DSP2834x_headers\source	Source files required to incorporate the header files into a new or existing project.
DSP2834x_headers\include	Header files for each of the on-chip peripherals.
DSP2834x_common\cmd	Example memory command files that allocate memory on the devices.
DSP2834x_common\include	Common .h files that are used by the peripheral examples.
DSP2834x_common\source	Common .c files that are used by the peripheral examples.
DSP2834x_common\lib	Common library (.lib) files that are used by the peripheral examples.
DSP2834x_common\gel	Code Composer Studio v3.x GEL files for each device. These are optional.
DSP2834x_common\gel\ccsv4	Code Composer Studio v4.x GEL files for each device. These are optional.

3 Understanding The Peripheral Bit-Field Structure Approach

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

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

Programming TMS320x28xx and 28xxx Peripherals in C/C++ (SPRAA85)

4 Peripheral Example Projects

This section describes how to get started with and configure the peripheral examples included in the 2834x Header Files and Peripheral Examples software package.

NOTE:

Because the '2834x devices are floating-point devices, the '2834x peripheral examples are configured for floating-point by default. C2000 CodeGenTools V5.x or later, which includes fpu32 floating-point support, is required to build and run these examples. (For more information, see Section 4.5).

4.1 Getting Started

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 4 installed.**

NOTE: As supplied, the '2834x example projects are built for the '28346 device. If you are using another 2834x device, the memory definition in the linker command file (.cmd) will need to be changed and the project rebuilt.

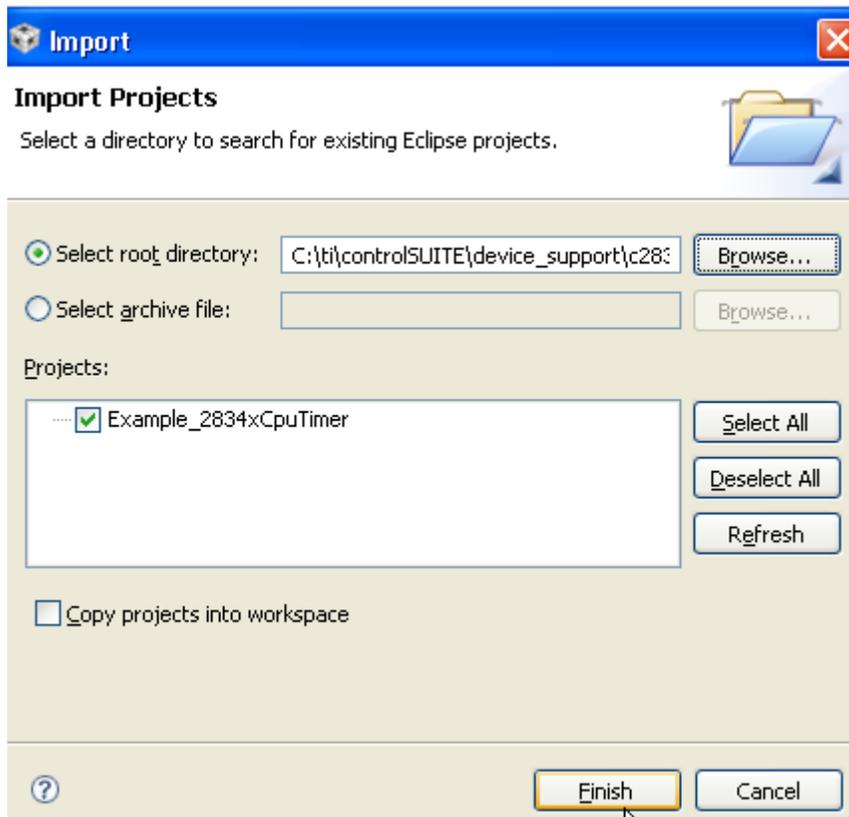
2. **Open the example project.**

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

To open the '2834x CPU-Timer example project directory, follow the following steps:

- a. In Code Composer Studio v 4.x: Project->Import Existing CCS/CCE Eclipse Project.
- b. Next to "Select Root Directory", browse to the CPU Timer example directory:
DSP2834x_examples_ccsv4\cpu_timer.

Note: do NOT check the "copy projects to workspace" option. This will break the project configuration.



Select the *Finish* button.

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

3. Edit DSP28_Device.h

Edit the DSP2834x_Device.h file and make sure the appropriate device is selected. By default the 28346 is selected.

```

/*****
* DSP2834x_headers\include\DSP2834x_Device.h
*****/

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

#define DSP28_28346 TARGET
#define DSP28_28345 0
#define DSP28_28344 0
#define DSP28_28343 0
#define DSP28_28342 0
#define DSP28_28341 0
#define DSP28_2834F 0

```

4. Edit DSP2834x_Examples.h

Edit DSP2834x_Examples.h and specify the clock rate, the PLL control register value (PLLCR and DIVSEL). These values will be used by the examples to initialize the PLLCR register and DIVSEL bits.

The default values will result in a 300Mhz SYSCLKOUT frequency assuming a 20 MHz input clock.

```

/*****
* DSP2834x_common\include\DSP2834x_Examples.h
*****/
/-----*
Specify the PLL control register (PLLCR) and divide select (DIVSEL) value.
-----*/
// #define DSP28_DIVSEL      0 // Enable /4 for SYSCLKOUT(default at reset)
// #define DSP28_DIVSEL      1 // Disable /4 for SYSCLKOUT
#define DSP28_DIVSEL        2 // Enable /2 for SYSCLKOUT
// #define DSP28_DIVSEL      3 // Enable /1 for SYSCLKOUT

#define DSP28_PLLCR         29 // Fout=Fin x 30
// #define DSP28_PLLCR       28 // Fout=Fin x 29
// #define DSP28_PLLCR       27 // Fout=Fin x 28
// #define DSP28_PLLCR       26 // Fout=Fin x 27
// #define DSP28_PLLCR       25 // Fout=Fin x 26
// #define DSP28_PLLCR       24 // Fout=Fin x 25
// #define DSP28_PLLCR       23 // Fout=Fin x 24
// #define DSP28_PLLCR       22 // Fout=Fin x 23
// #define DSP28_PLLCR       21 // Fout=Fin x 22
// #define DSP28_PLLCR       20 // Fout=Fin x 21
// #define DSP28_PLLCR       19 // Fout=Fin x 20
...
// #define DSP28_PLLCR        3 // Fout=Fin x 4
// #define DSP28_PLLCR        2 // Fout=Fin x 3
// #define DSP28_PLLCR        1 // Fout=Fin x 2
// #define DSP28_PLLCR        0 // (Default at reset) PLL is bypassed in this mode
/-----

```

In DSP2834x_Examples.h, also specify the SYSCLKOUT rate. This value is used to scale a delay loop used by the examples. The default value is for a 300 Mhz SYSCLKOUT. If you have a 250 MHz or 200 MHz device you will need to adjust these settings accordingly.

```

/*****
* DSP2834x_common\include\DSP2834x_Examples.h
*****/
...
#define CPU_RATE            3.333L // for 300MHz CPU clock speed (SYSCLKOUT)
// #define CPU_RATE          4.000L // for 250MHz CPU clock speed (SYSCLKOUT)
// #define CPU_RATE          5.000L // for 200 MHz CPU clock speed (SYSCLKOUT)
...

```

In DSP2834x_Examples.h the device selected in DSP2834x_Device.h also specifies the maximum SYSCLKOUT frequency (300MHz, 250MHz, or 200MHz) by setting it to 1 and the others to 0. This value is used by those examples with timing dependent code (i.e. baud rates or other timing parameters) to determine whether 300MHz code, 250MHz code, or 200MHz code should be run.

**5. Review the comments at the top of the main source file:
Example_2834xCpuTimer.c.**

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

6. Perform any hardware setup required by the example.

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

The following table shows a listing of the boot mode pin settings for your reference. Refer to the documentation for your hardware platform for information on configuring the boot mode pins. For more information on the ‘2834x boot modes refer to the device specific *Boot ROM Reference Guide*.

Table 3. 2834x Boot Mode Settings

GPIO87 XA15 PU	GPIO86 XA14 PU	GPIO85 XA13 PU	GPIO84 XA12 PU	Mode
1	1	1	1	TI Test Only
1	1	1	0	Call SCI-A boot loader
1	1	0	1	Call SPI-A boot loader
1	1	0	0	Call I2C-A timing 1 boot loader
1	0	1	1	Call eCAN-A timing 1 boot loader
1	0	1	0	Call McBSP-A boot loader
1	0	0	1	Boot to XINTF x16 0x100000
1	0	0	0	Reserved
0	1	1	1	Call eCAN-A timing 2 boot loader
0	1	1	0	Call parallel GPIO boot loader
0	1	0	1	Call parallel XINTF boot loader
0	1	0	0	Boot to M0 SARAM 0x000000
0	0	1	1	Branch to check boot mode
0	0	1	0	Call I2C-A timing 2 boot loader
0	0	0	1	Reserved
0	0	0	0	TI test only

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 **Error! Reference source not found.** 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*.

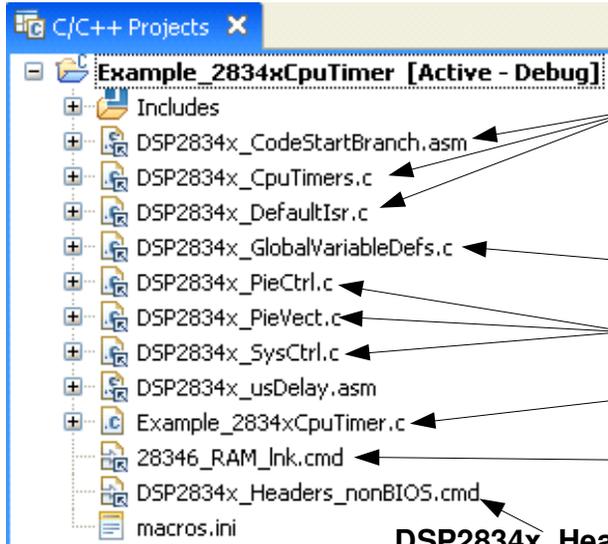
Note: Make sure the “*Do not delete contents*” button is selected, then select *Yes*.

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 *DSP2834x_headers* directory and shared source in the *DSP2834x_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



Common (shared) Source Code
Used by more than one example. These files contain generic functions for setting up peripherals to a defined state or functions that may be useful to re-use in different applications.

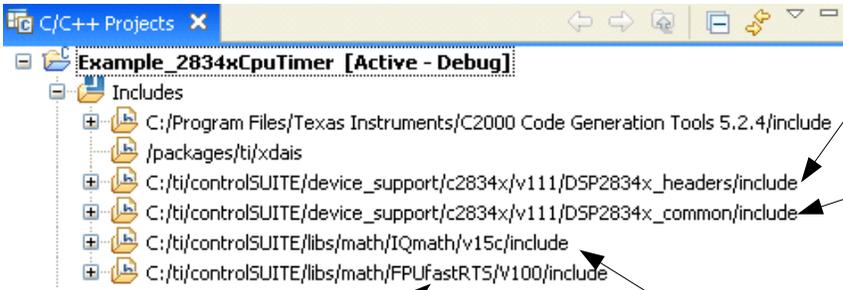
DSP2834x_GlobalVariableDefs.c
This source file is required to use the header files.

Shared Source Code

Example Specific Source Code

Memory block specific linker command file

DSP2834x_Headers_nonBIOS.cmd
Linker file required by the peripheral specific header files.



Header File Include Directory
Files include bit-field structures for each Peripheral.

Common File Include Directory
Not required for the bit-field definitions, but used by the example projects.

IQmath library – Used by some examples.
Refer to the documentation in the /doc directory of the IQmath library install for more information.

fastRTS library. Linked into all of the projects.
Refer to the documentation in the /doc directory of the fastRTS library install for more information.

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

4.2.1 Include Files

```

/*****
* DSP2834x_examples\cpu_timer\Example_2834xCpuTimer.c
*****/

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

```

- **DSP28x_Project.h**

This header file includes DSP2834x_Device.h and DSP2834x_Examples.h. Because the name is device-generic, example/custom projects can be easily ported between different device header files. With this file included in the example source files, only the example/custom project (.pj) file and DSP28x_Project.h file would need to be modified when porting source code between different devices. This file is found in the `<base>\DSP2834x_common\include` directory.

- **DSP2834x_Device.h**

This header file is required to use the header files. This file includes all of the required peripheral specific header files and includes device specific macros and typedef statements. This file is found in the `<base>\DSP2834x_headers\include` directory.

- **DSP2834x_Examples.h**

This header file defines parameters that are used by the example code. This file is not required to use just the DSP2834x peripheral header files but is required by some of the common source files. This file is found in the `<base>\DSP2834x_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.

- **DSP2834x_GlobalVariableDefs.c**

Any project that uses the DSP2834x 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>\DSP2834x_headers\source` directory.

- **Example specific source code:**

Files that are specific to a particular example have the prefix Example_2834x in their filename. For example Example_2834xCpuTimer.c is specific to the CPU Timer example and not used for any other example. Example specific files are located in the `<base>\DSP2834x_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 DSP2834x_common\source directory. Users may choose to incorporate none, some, or the entire shared source into their own new or existing projects.

4.2.3 Linker Command Files

Each example uses two linker command files. These files specify the memory where the linker will place code and data sections. One linker file is used for assigning compiler generated sections to the memory blocks on the device while the other is used to assign the data sections of the peripheral register structures used by the DSP2834x peripheral header files.

- **Memory block linker allocation:**

The linker files shown in Table 4 are used to assign sections to memory blocks on the device. These linker files are located in the *<base>\DSP2834x_common\cmd* directory. Each example will use one of the following files depending on the memory used by the example.

Table 4. Included Memory Linker Command Files

Memory Linker Command File Examples	Location	Description
28346_RAM_Ink.cmd	DSP2834x_common\cmd	28346 memory linker command file. Includes all of the internal SARAM blocks on a 28346 device. "RAM" linker files do not include flash or OTP blocks.
28345_RAM_Ink.cmd	DSP2834x_common\cmd	28345 SARAM memory linker command file.
28344_RAM_Ink.cmd	DSP2834x_common\cmd	28344 SARAM memory linker command file.
28343_RAM_Ink.cmd	DSP2834x_common\cmd	28343 SARAM memory linker command file.
28342_RAM_Ink.cmd	DSP2834x_common\cmd	28342 SARAM memory linker command file.
28341_RAM_Ink.cmd	DSP2834x_common\cmd	28341 SARAM memory linker command file.

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

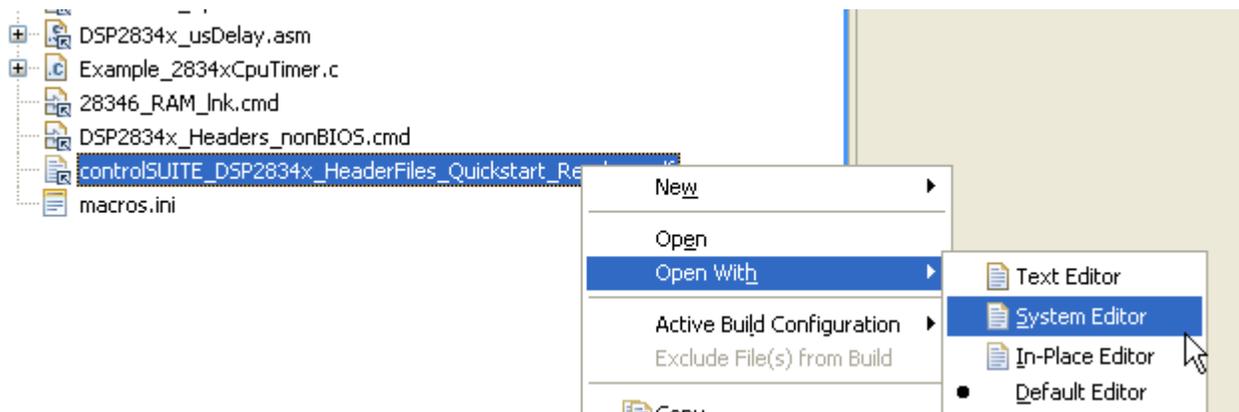
Any project that uses the header file peripheral structures must include a linker command file that assigns the peripheral register structure data sections to the proper memory location. These files are described in Table 5.

Table 5. DSP2834x Peripheral Header Linker Command File

Header File Linker Command File	Location	Description
DSP2834x_Headers_BIOS.cmd	DSP2834x_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.
DSP2834x_Headers_nonBIOS.cmd	DSP2834x_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.

4.2.4 Documentation

This document is linked into each project so it can easily be opened through the project view. To do this, right click on the document, select “open with” and “system editor”.



4.3 Example Program Flow:

All of the example programs follow a similar recommended flow for setting up a 2834x device.

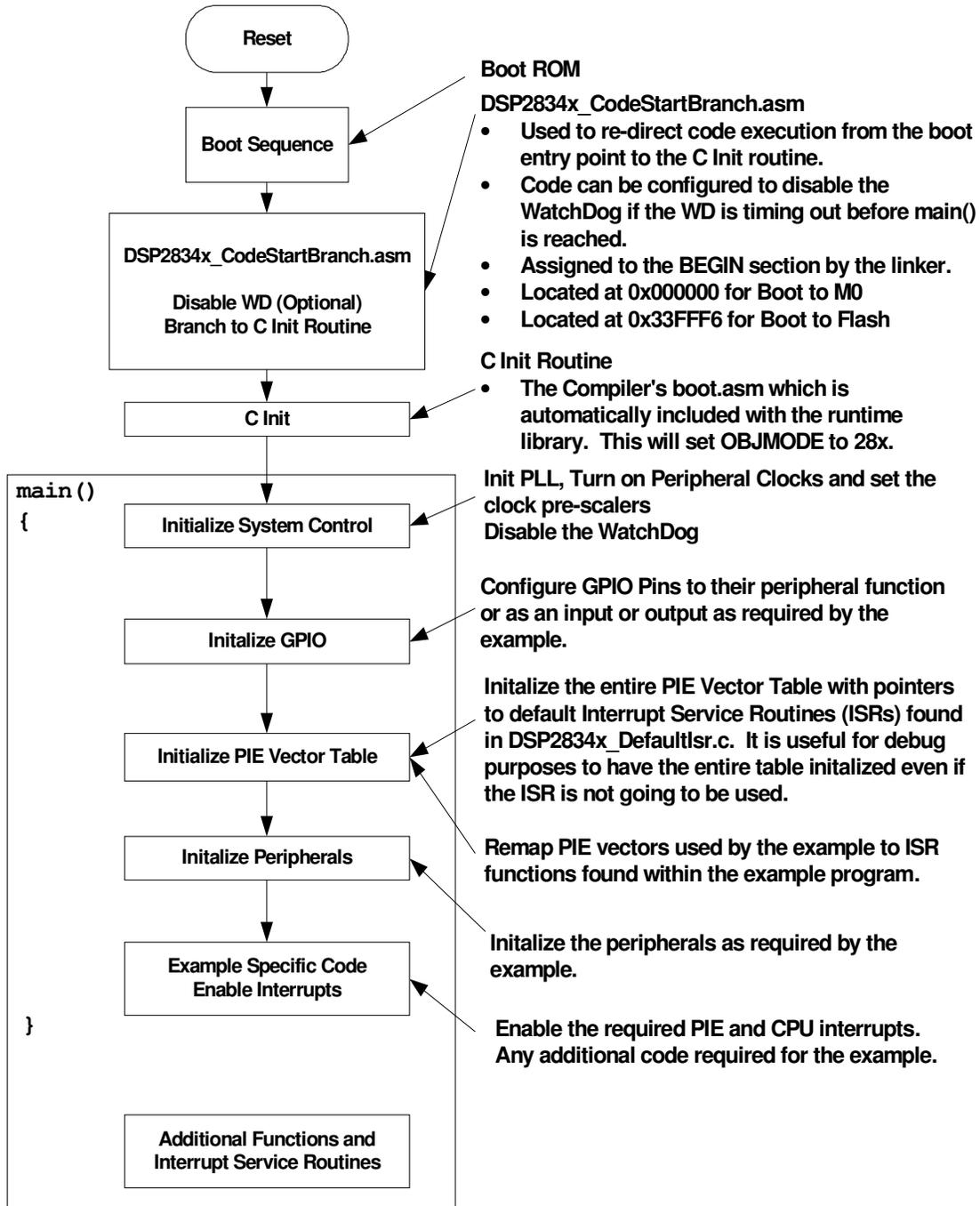


Figure 1. Flow for Example Programs

4.4 Included Examples:

Table 6. Included Examples

Example	Description
cpu_timer	Configures CPU Timer0 and increments a count each time the ISR is serviced.
dma_ram_to_ram	Example of RAM to RAM data block transfer using the DMA.
dma_xintf_to_ram	Example of XINTF to RAM data block transfer using the DMA.
ecan_a_to_b_xmit	Transmit from eCANa to eCANb
ecan_back2back	eCAN self-test mode example. Transmits eCAN data back-to-back at high speed without stopping.
ecap_apwm	This example sets up the alternate eCAP pins in the APWM mode
ecap_capture_pwm	Captures the edges of a ePWM signal.
epwm_deadband	Example deadband generation via ePWM3
epwm_timer_interrupts	Starts ePWM1-ePWM6 timers. Every period an interrupt is taken for each ePWM.
epwm_trip_zone	Uses the trip zone signals to set the ePWM signals to a particular state.
epwm_up_aq	Generate a PWM waveform using an up count time base ePWM1-ePWM3 are used.
epwm_updown_aq	Generate a PWM waveform using an up/down time base. ePWM- ePWM3 are used.
eqep_freqcal	Frequency cal using eQEP1
eqep_pos_speed	Pos/speed calculation using eQEP1
external_interrupt	Configures GPIO0 as XINT1 and GPIO1 as XINT2. The interrupts are fired by toggling GPIO30 and GPIO31 which are connected to XINT1 (GPIO0) and XINT2 (GPIO1) externally by the user.
fpu	Two projects illustrating the difference between code compiled with floating-point hardware (FPU) and fixed-point hardware (using software to simulate floating-point).
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 an EEPROM via I2C
lpm_haltwake	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_idlewake	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.

Included Examples Continued...

lpm_standbywake	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.
mcbsp_loopback	McBSP-A example that uses the peripheral's loop-back testmode to send data.
mcbsp_loopback_dma	McBSP-A example that uses the peripheral's loop-back testmode with the DMA to send and receive data.
mcbsp_loopback_interrupts	McBSP-A example that uses the peripheral's loop-back testmode to send data. Interrupts are used in this example.
mcbsp_spi_loopback	McBSP-A example that configures the peripheral for SPI mode and uses the loop-back testmode to send data.
sci_autobaud	Externally connect SCI-A to SCI-B and send data between the two peripherals. Baud lock is performed using the autobaud feature of the SCI. This test is repeated for different baud rates.
sci_echoback	SCI-A example that can be used to echoback to a terminal program such as hyperterminal. A transceiver and a connection to a PC is required.
scia_loopback	SCI-A example that uses the peripheral's loop-back test mode to send data.
scia_loopback_interrupts	SCI-A example that uses the peripheral's loop-back test mode to send data. Both interrupts and FIFOs are used in this example.
spi_loopback	SPI-A example that uses the peripherals loop-back test mode to send data.
spi_loopback_interrupts	SPI-A example that uses the peripherals loop-back test mode to send data. Both interrupts and FIFOs are used in this example.
sw_prioritized_interrupts	The standard hardware prioritization of interrupts can be used for most applications. This example shows a method for software to re-prioritize interrupts if required.
timed_led_blink	This example blinks GPIO34 (LED on the control card) at a rate of 1 Hz using CPU Timer 0.
watchdog	Illustrates feeding the dog and re-directing the watchdog to an interrupt.
xintf_run_from	This example shows how to run from XINTF zone 7 and configure the XINTF memory

4.5 Converting Floating-Point Compiled Examples to Fixed-Point and Vice Versa

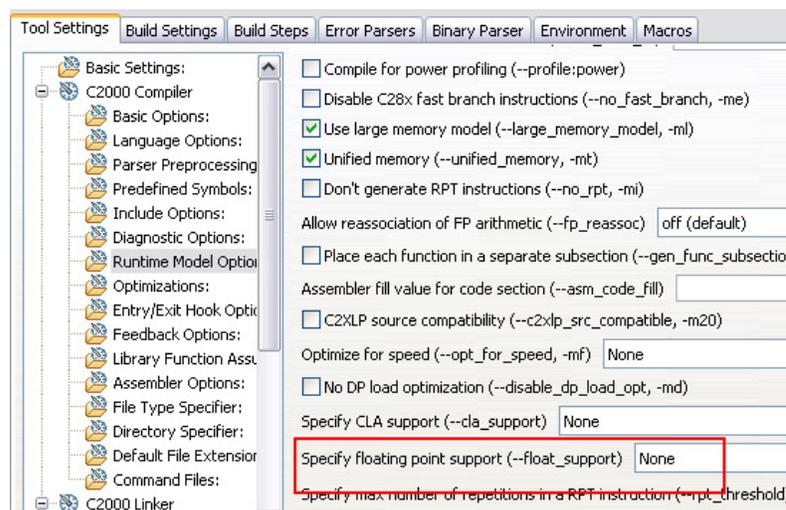
Because the '2834x is a floating-point device, all of the DSP2834x examples (unless otherwise denoted in the example description) are configured for floating-point. In some cases, it may be desirable to compile the code for fixed-point instead of floating-point.

To convert the examples so they compile for fixed-point, certain steps must be taken. The following steps are demonstrated on the example in *DSP2834x_examples\fpu_software* and *\fpu_hardware*. The directory includes two projects with identical C-code— one compiled using fixed-point instructions and the other compiled using floating-point instructions.

All instructions in this section apply to Code Composer Studio V4.

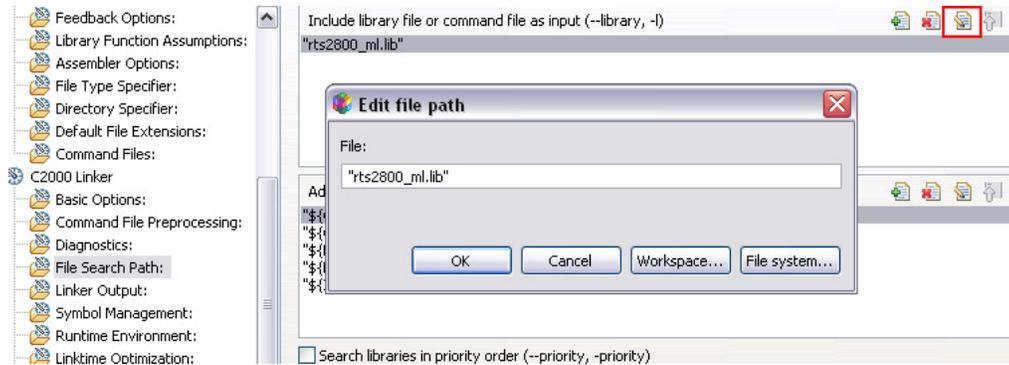
1. Configure the compiler build options for fixed-point instead of floating-point.

- a. Go to *Project-> Properties*
- b. Select *C/C++ Build*.
- c. Select the *Tool Settings* tab,
- d. Select “*C2000 Compiler*”->*Runtime Model Options*.
- e. In the screen that appears on the right, select “*None*” from the “*Specify floating point support (--float_support)*” pull-down menu.
- f. Then select the *Apply* button at the bottom of the window.



2. Use the fixed-point version of the rts2800.lib library instead of the floating-point version.

- a. Go to *Project -> Properties*
- b. Select the *Tool Settings* tab
- c. Select “*C2000 Linker->File Search Path*”
- d. In the upper box on the right labeled “*Include library file or command file as input (-library, -l)*”,
 - i. Click on “*rts2800_fpu32.lib*” to select it, and then and click on the “*Edit*” icon in the top right
 - ii. Change “*rts2800_fpu32.lib*” (for floating-point version of the rts2800 library) to “*rts2800_ml.lib*” (for fixed-point large memory model version of the library).
 - iii. Select the “*rts2800_fpu32_fast_supplement.lib*” and click on the ‘X’ to remove it. This library is only used for floating-point code.



3. Replace any floating-point compiled libraries included in the project with their fixed-point equivalents.

If your project is compiled for floating-point (fpu32 option), then any libraries included by your project must also be compiled for floating-point. Likewise, if your project is compiled for fixed-point, the included libraries must also be compiled for fixed-point.

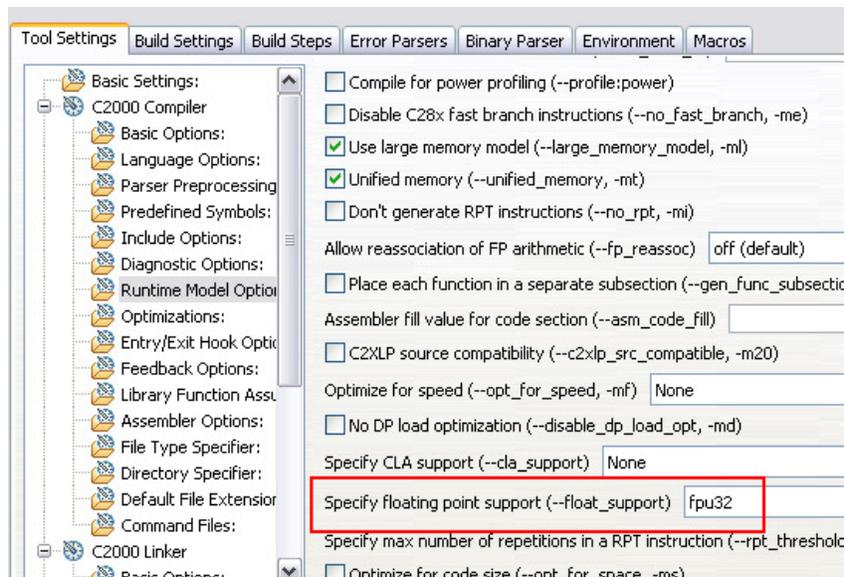
- a. In the Project window, right click on “SFO_TI_Build_V5_fpu.lib”, and select “Delete”.
- b. Then, right-click on the project name and select “New->File”.
- c. Select your project. Then at the bottom of the window, click the “Advanced>>” button.
- d. Check the “Link to file in the file system” checkbox.
- e. Now you have 2 options for adding the floating-point version of the library to your project. Option A is the quickest way, but only works on your own computer as long as the library remains at the same location on your computer (the project will break on another computer or if you move the library to a different location on your computer). Option B takes a few more steps, but it can be used if you plan to move the project and associated files to another location on the same or different computer.
 - i. Option A: Select “Browse...” and navigate to where the floating-point version of your library is located, and select the library to replace the fixed-point version. (for instance, “SFO_TI_Build_V5_fpu.lib” is replaced with “SFO_TI_Build_V5.lib”).
 - ii. Option B: Select “Variables...”, and select the macro pointing to the path of your header file installation direction. (i.e. select INSTALLROOT_2834X_V110 which points to the default installation directory path for V1.10 of the 2834x header files and peripheral examples). Then select “Extend...”, and navigate to where the fixed-point version of your library is located, and select the library to replace the floating-point version. (for instance, “SFO_TI_Build_V5_fpu.lib” is replaced with “SFO_TI_Build_V5.lib”).

Repeat the same procedure for the IQmath_fpu32.lib library.

After these 3 steps are performed, the floating-point example project has been converted to fixed-point and can be re-compiled and built for fixed-point. To convert a fixed-point example back into floating-point, the following steps must be taken:

1. Configure the compiler build options for floating-point instead of fixed-point.

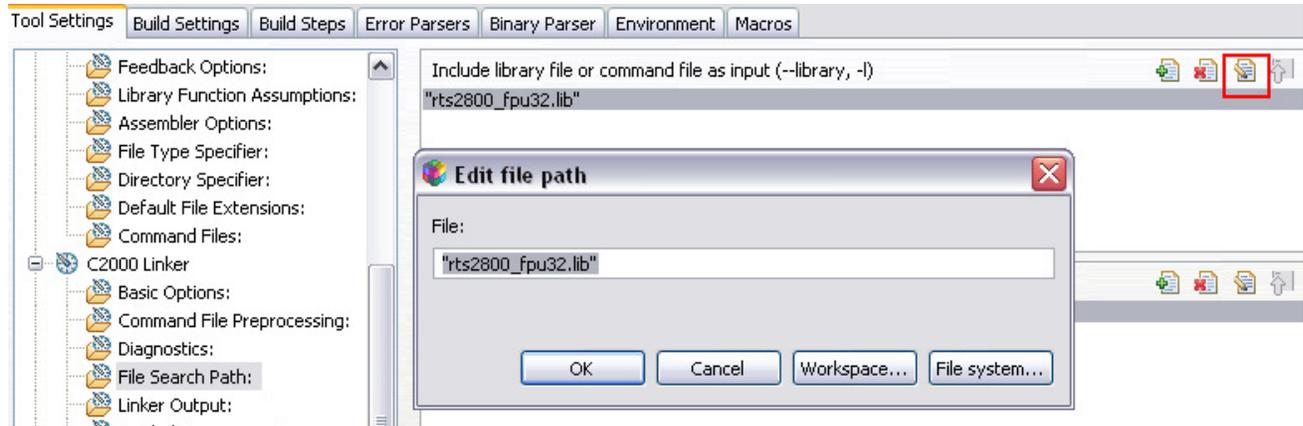
- a. Go to *Project-> Properties*.
- b. In CCStudio v4.0: Select *C/C++ Build*. Then in the *Tool Settings* tab, select “*C2000 Compiler*”->*Runtime Model Options*. In the screen that appears on the right, select “*fpu32*” from the “*Specify floating point support (--float_support)*” pull-down menu. Then select the *Apply* button at the bottom of the window.



2. Use the floating-point version of the rts2800.lib library instead of the fixed-point version.

In Code Composer Studio v4.x:

- a. In the *Tool Settings* tab, go to “*C2000 Linker->File Search Path:*”
- b. In the upper box on the right labeled “*Include library file or command file as input (-library, -l)*”, select the “*Edit*” icon in the top right, and change “*rts2800_ml.lib*” (for fixed-point large memory version of the rts2800 library) to “*rts2800_fpu32.lib*” (for the floating-point version of the library).



3. Replace any fixed-point compiled libraries included in the project with their floating-point equivalents.

- a. In the Project window, right click on “SFO_TI_Build_V5.lib”, and select “Delete”.
- b. Then, right-click on the project name and select “New->File”.
- c. Select your project. Then at the bottom of the window, click the “Advanced>>” button.
- d. Check the “Link to file in the file system” checkbox.
- e. Now you have 2 options for adding the floating-point version of the library to your project. Option A is the quickest way, but only works on your own computer as long as the library remains at the same location on your computer (the project will break on another computer or if you move the library to a different location on your computer). Option B takes a few more steps, but it can be used if you plan to move the project and associated files to another location on the same or different computer.
 - iii. Option A: Select “Browse...” and navigate to where the floating-point version of your library is located, and select the library to replace the fixed-point version. (for instance, “SFO_TI_Build_V5.lib” is replaced with “SFO_TI_Build_V5_fpu.lib”).
 - iv. Option B: Select “Variables...”, and select the macro pointing to the path of your header file installation direction. (i.e. select INSTALLROOT_2834X_V110 which points to the default installation directory path for V1.10 of the 2834x header files and peripheral examples). Then select “Extend...”, and navigate to where the floating-point version of your library is located, and select the library to replace the fixed-point version. (for instance, “SFO_TI_Build_V5.lib” is replaced with “SFO_TI_Build_V5_fpu.lib”).

After these 3 steps are performed, the fixed-point example project has been converted to floating-point and can be re-compiled and built for floating-point.

5 Steps for Incorporating the Header Files and Sample Code

Follow these steps to incorporate the peripheral header files and sample code into your own projects. If you already have a project that uses the DSP280x or DSP281x header files then also refer to Section 7 for migration tips.

All instructions in this section apply to Code Composer Studio V4.

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.

DSP2834x_headers: code required to incorporate the header files into your project
DSP2834x_common: shared source code much of which is used in the example projects.

DSP2834x_examples: '2834x floating-point compiled example projects that use the header files and shared code.

5.2 Including the DSP2834x Peripheral Header Files

Including the DSP2834x 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 "DSP2834x_Device.h" (or #include "DSP28x_Project.h") in your source files.

The DSP2834x_Device.h 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 "DSP2834x_Device.h"

```

Another option is to #include "DSP28x_Project.h" in your source files, which in-turn includes "DSP2834x_Device.h" and "DSP2834x_Examples.h" (if it is not necessary to include common source files in the user project, the #include "DSP2834x_Examples.h" line can be deleted). Due to the device-generic nature of the file name, user code is easily ported between different device header files. With this file included in the user's

source files, only the project (.pj) file and DSP28x_Project.h file would need to be modified when porting source code between different devices.

```

/*****
* User's source file
*****/

#include "DSP28x_Project.h"

```

Edit DSP2834x_Device.h and select the target you are building for:

In the below example, the file is configured to build for the '28346 device.

```

/*****
* DSP2834x_headers\include\DSP2834x_Device.h
*****/

#define TARGET 1
#define DSP28_28346 TARGET // Selects '28346
#define DSP28_28343 0 // Selects '28343
#define DSP28_2834F 0 // Selects '2834F... etc

```

By default, the '28346 device is selected.

2. Add the source file *DSP2834x_GlobalVariableDefs.c* to the project.

This file is found in the *DSP2834x_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.

3. Add the appropriate DSP2834x header linker command file to the project.

As described in Section 3, when using the DSP2834x 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 *DSP2834x_headers\cmd* must be included in your project:

- For non-DSP/BIOS[†] projects: *DSP2834x_Headers_nonBIOS.cmd*
- For DSP/BIOS projects: *DSP2834x_Headers_BIOS.cmd*

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

2) Add the directory path to the DSP2834x peripheral linker .cmd file to your project.

Method #1:

Note: The example as provided use Method #2.

- 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 *DSP2833x_headers\cmd* directory on your system and select the desired .cmd file.

Note:

The limitation with Method #1 is that the path to the .cmd file 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](http://tiexpressdsp.com/index.php/Portable%20Projects%20in%20CCSv4%20for%20C2000)

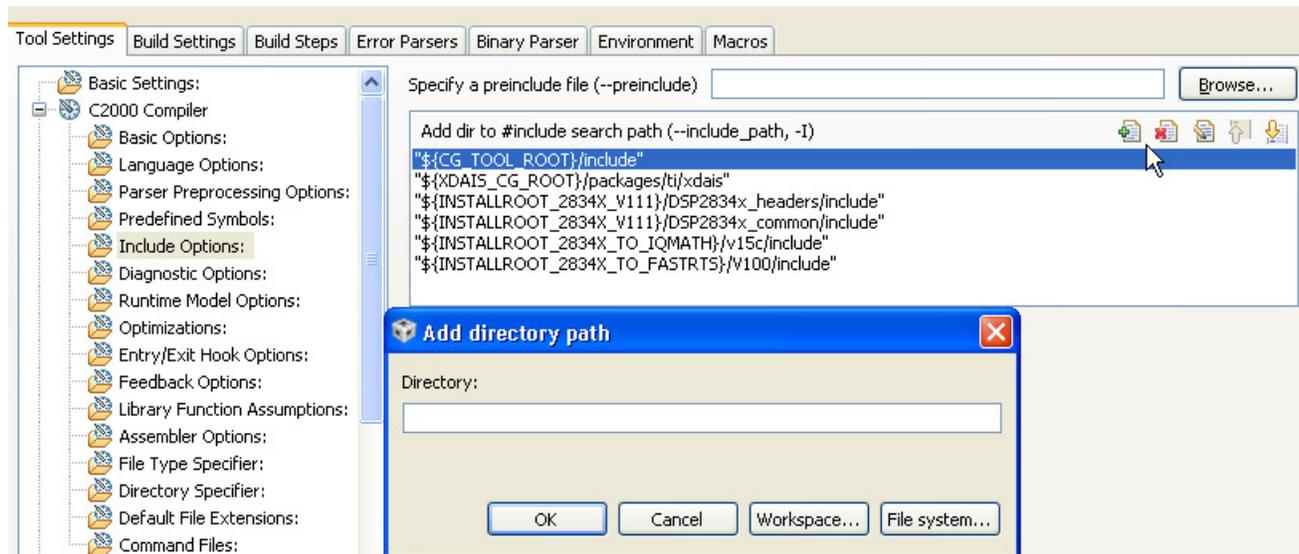
Method #2:

- 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 2833x header files, this is *INSTALLROOT_2833X_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](http://tiexpressdsp.com/index.php/Portable%20Projects%20in%20CCSv4%20for%20C2000#Method.232%20for%20Linking%20Files%20to%20Project):
- f. Click on the *Extend...* button. Navigate to the desired .cmd file and select *OK*.

4. Add the directory path to the DSP2834x header files to your project.

To specify the directory where the header files are located:

- Open the menu: *Project->Properties*.
- In the menu on the left, select “C/C++ Build”.
- In the “*Tool Settings*” tab, Select “*C2000 Compiler -> Include Options:*”
- In the “Add dir to #include search path (--include_path, -I)” window, select the “Add” “+” icon in the top right corner.
- Select the “*File system...*” button and navigate to the directory path of *DSP2833x_headers\include* on your system.



5. 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 DSP2834x examples, this is the symbol “code_start”. This symbol is defined in the DSP2834x_common\source\DSP2834x_CodeStartBranch.asm file. When you load the code in Code Composer Studio, the debugger will set the PC to the address of this symbol. If you do not define a entry point using the –e option, then the linker will use _c_int00 by default.

5.3 Including Common Example Code

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

1. #include “DSP2834x_Examples.h” (or “DSP28x_Project.h”) in your source files.

The “DSP2834x_Examples.h” include file will include common definitions and declarations used by the example code.

```

/*****
* User's source file
*****/

#include "DSP2834x_Examples.h"

```

Another option is to #include “DSP28x_Project.h” in your source files, which in-turn includes “DSP2834x_Device.h” and “DSP2834x_Examples.h”. Due to the device-generic nature of the file name, user code is easily ported between different device header files. With this file included in the user’s source files, only the project (.pj) file and DSP28x_Project.h file would need to be modified when porting source code between different devices.

```

/*****
* User's source file
*****/

#include "DSP28x_Project.h"

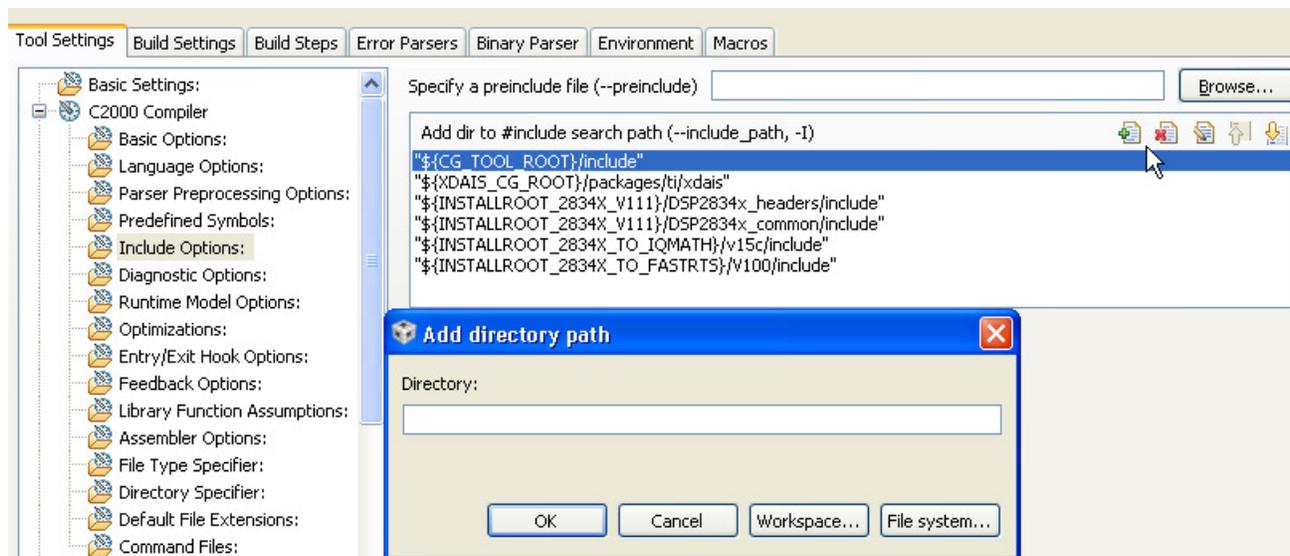
```

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

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 *DSP2834x_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 *DSP2834x_common\cmd* directory. For getting started the basic *28346_RAM_Ink.cmd* file is suggested and used by most of the examples.

Table 7. Included Main Linker Command Files

Memory Linker Command File Examples	Location	Description
28346_RAM_Ink.cmd	DSP2834x_common\cmd	28346 memory linker command file. Includes all of the internal SARAM blocks on a 28346 device. "RAM" linker files do not include flash or OTP blocks.
28345_RAM_Ink.cmd	DSP2834x_common\cmd	28345 SARAM memory linker command file.
28344_RAM_Ink.cmd	DSP2834x_common\cmd	28344 SARAM memory linker command file.
28343_RAM_Ink.cmd	DSP2834x_common\cmd	28343 SARAM memory linker command file.
28342_RAM_Ink.cmd	DSP2834x_common\cmd	28342 SARAM memory linker command file.
28341_RAM_Ink.cmd	DSP2834x_common\cmd	28341 SARAM memory linker command file.
2834F_RAM_Ink.cmd	DSP2834x_common\cmd	2834F SARAM memory linker command file.

4. Set the CPU Frequency

In the *DSP2834x_common\include\DSP2834x_Examples.h* file specify the proper CPU frequency. Some examples are included in the file.

```
/******  
* DSP2834x_common\include\DSP2834x_Examples.h  
*****/  
...  
#define CPU_RATE      3.333L      // for 300MHz CPU clock speed (SYSCLKOUT)  
//#define CPU_RATE    4.000L      // for 250MHz CPU clock speed (SYSCLKOUT)  
//#define CPU_RATE    5.000L      // for 200 MHz CPU clock speed (SYSCLKOUT)  
...
```

5. Add desired common source files to the project.

The common source files are found in the *DSP2834x_common\source* directory.

6. Include .c files for the PIE.

Since all catalog '2834x 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 DSP2834x_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 *TMS320x2834x System Control and Interrupts Reference Guide*.

- **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 DSP2834x_common\source directory shows an example of enabling the peripheral clocks.
- Some peripherals are not present on all 2834x 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. *TMS320x2834x System Control and Interrupts Reference Guide (SPRUFB0)* for a complete list of EALLOW protected registers.

- **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. *TMS320x2834x System Control and Interrupts Reference Guide* (SPRUFB0) for a complete list of EALLOW protected registers.

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

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

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

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

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

- The entire PIE vector table is enabled, even if the ISR is not used within the example. This can be very useful for debug purposes.
- The default ISR file is left un-modified for use with other examples or your own project as you see fit.
- It illustrates how the PIE table can be updated at a later time.

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

This warning is given when a symbol other than `_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 `DSP2834x_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 DSP2834x 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 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 `DSP2834x_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 than one bit is set, performing a read-modify-write on the register may clear more bits than intended.

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

```

/*****
* User's source file
*****/

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

```

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

```

/*****
* User's source file
*****/

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

```

6.1.2 Registers with Volatile Bits.

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

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

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

7 Migration Tips for moving from the TMS320x2833x/TMS320x2823x header files to the TMS320x2834x header files

This section includes suggestions for moving a project from the 2833x/2823x header files to the 2834x header files.

1. **Create a copy of your project to work with or back-up your current project.**
2. **Open the project file(s) in a text editor**

In Code Composer Studio v3.x:

Open the .pj1 project file for your project. Replace all instances of 2833x or 2823x with 2834x so that the appropriate source files and build options are used. Check the path names to make sure they point to the appropriate header file and source code directories.

In Code Composer Studio v4.x:

Open the *.project*, *.cdtbuild*, and *macros.ini* files in your example folder. Replace all instances of 2833/2823x with 2834x so that the appropriate source files and build options are used. Check the path names to make sure they point to the appropriate header file and source code directories. Also replace the header file version number for the paths and macro names as well where appropriate. For instance, if a macro name was `INSTALLROOT_2833X_V130` for your 2833x project using 2833x header files V1.30, change this to `INSTALLROOT_2834X_V110` to migrate to the 2834x header files V1.10. If not using the default macro name for your header file version, be sure to change your macros according to your chosen macro name in the *.project*, *.cdtbuild*, and *macros.ini* files.

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

Use the Edit-> find in files dialog to find instances of `DSP2833x_Device.h` and `DSP2833x_Example.h`. Replace these with `DSP2834x_Device.h` and `DSP2834x_Example.h` respectively.

4. **Make sure you are using the correct linker command files (.cmd) appropriate for your device and for the DSP2834x header files.**

You will have one file for the memory definitions and one file for the header file structure definitions. Using a 2833x/2823x memory file can cause issues since the L0-L3 memory blocks are no longer mirrored on the 2834x devices and the flash memories on a 2833x/2823x memory file also no longer exist on the 2834x devices (replaced with H0-H5 SARAM memories).

5. **Build the project.**

The compiler will highlight areas that have changed. If migrating from the TMS320x2833x/2823x header files, code should be mostly compatible after all instances of 2833x/2823x are replaced with 2834x in all relevant files, and the above steps are taken. Additionally, several bits have been modified. See Table 8.

Table 8. Summary of Register and Bit-Name Changes from 2833x/2823x V1.60 to 2834x V1.00

Peripheral	Register	Bit Name		Comment
		Old	New	
SysCtrlRegs				
	HISPCP	HSPCLK (bits 3:0)	HSPCLK (bits 4:0)	Previously, on 2833x/2823x: HSPCLK = SYSCLKOUT/(HISPCP X 2). On 2834x: HSPCLK = SYSCLKOUT / [(HISPCP +1)* 2]
	PLLCR	DIV (bits 3:0)	DIV (bits 4:0)	Previously, on 2833x/2823x : CLKIN * DIV = VCOCLK frequency with DIV = 0 (bypass) to 10. On 2834x: CLKIN * (DIV+1) = VCOCLK frequency with DIV = 0 (bypass) to 31.
	PLLSTS	MCLKSTS	Removed	Limp mode function was removed
		MCLKCLR	Removed	Limp mode function was removed
		OSCOFF	Removed	Limp mode function was removed
		MCLKOFF	Removed	Limp mode function was removed
DevEmuRegs				
	PARTID	Register at 0x380090	Register at 0x882	Because there is no OTP on the 2834x devices, the PARTID register has been moved to address 0x882 and replaces the CLASSID register
	CLASSID	Register at 0x882	Removed and replaced by PARTID register.	CLASSID register does not exist for 2834x devices
AdcRegs				
	All registers			Internal ADC module does not exist on 2834x devices, and therefore code written for ADC on the 2833x/2823x is not applicable to the 2834x.

8 Packet Contents:

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

8.1 Header File Support – DSP2834x_headers

The 2834x header files are located in the `<base>\DSP2834x_headers\` directory.

8.1.1 2834x Header Files – Main Files

The following files must be added to any project that uses the 2834x header files. Refer to section 5.2 for information on incorporating the header files into a new or existing project.

Table 9. DSP2834x Header Files – Main Files

File	Location	Description
DSP2834x_Device.h	DSP2834x_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.
DSP2834x_GlobalVariableDefs.c	DSP2834x_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.
DSP2834x_Headers_BIOS.cmd	DSP2834x_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.
DSP2834x_Headers_nonBIOS.cmd	DSP2834x_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 DSP2834x 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 2834x devices. These files are automatically included in the project by including *DSP2834x_Device.h*. Refer to section 4.2 for more information on incorporating the header files into a new or existing project.

Table 10. DSP2834x Header File Bit-Field & Register Structure Definition Files

File	Location	Description
DSP2834x_CpuTimers.h	DSP2834x_headers\include	CPU-Timer register structure and bit-field definitions.
DSP2834x_DevEmu.h	DSP2834x_headers\include	Emulation register definitions
DSP2834x_DMA.h	DSP2834x_headers\include	DMA register structures and bit-field definitions.
DSP2834x_ECan.h	DSP2834x_headers\include	eCAN register structures and bit-field definitions.
DSP2834x_ECap.h	DSP2834x_headers\include	eCAP register structures and bit-field definitions.
DSP2834x_EPwm.h	DSP2834x_headers\include	ePWM register structures and bit-field definitions.
DSP2834x_EQep.h	DSP2834x_headers\include	eQEP register structures and bit-field definitions.
DSP2834x_Gpio.h	DSP2834x_headers\include	General Purpose I/O (GPIO) register structures and bit-field definitions.
DSP2834x_I2c.h	DSP2834x_headers\include	I2C register structure and bit-field definitions.
DSP2834x_Mcbsp.h	DSP2834x_headers\include	McBSP register structure and bit-field definitions.
DSP2834x_PieCtrl.h	DSP2834x_headers\include	PIE control register structure and bit-field definitions.
DSP2834x_PieVect.h	DSP2834x_headers\include	Structure definition for the entire PIE vector table.
DSP2834x_Sci.h	DSP2834x_headers\include	SCI register structure and bit-field definitions.
DSP2834x_Spi.h	DSP2834x_headers\include	SPI register structure and bit-field definitions.
DSP2834x_SysCtrl.h	DSP2834x_headers\include	System register definitions. Includes Watchdog, PLL, CSM, Flash/OTP, Clock registers.
DSP2834x_Xintf.h	DSP2834x_headers\include	XINTF register structure and bit-field definitions.
DSP2834x_XIntrupt.h	DSP2834x_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 DSP2834x Header File peripheral register structures.

Table 11. DSP2834x Included GEL Files

File	Location	Description
DSP2834x_Peripheral.gel	DSP2834x_headers\gel	This is relevant to Code Composer Studio V3.x only. Provides GEL pull-down menus to load the DSP2834x data structures into the watch window. You may want to have CCS load this file automatically by adding a GEL_LoadGel("<base>DSP2834x_headers\gel\DSP2834x_peripheral.gel") function to the standard C28346.gel that was included with CCS.

8.1.4 Variable Names and Data Sections

This section is a summary of the variable names and data sections allocated by the DSP2834x_headers\source\DSP2834x_GlobalVariableDefs.c file. Note that all peripherals may not be available on a particular 2834x device. Refer to the device datasheet for the peripheral mix available on each 2834x family derivative.

Table 12. 2834x Variable Names and Data Sections

Peripheral	Starting Address	Structure Variable Name
Code Security Module	0x000AE0	CsmRegs
Code Security Module Password Locations	0x33FFF8-0x33FFFF	CsmPwl
CPU Timer 0	0x000C00	CpuTimer0Regs
CPU Timer 1	0x000C08	CpuTimer1Regs
CPU Timer 2	0x000C10	CpuTimer2Regs
Device and Emulation Registers	0x000880	DevEmuRegs
DMA Registers	0x001000	DmaRegs
eCAN-A	0x006000	ECanaRegs
eCAN-A Mail Boxes	0x006100	ECanaMboxes
eCAN-A Local Acceptance Masks	0x006040	ECanaLAMRegs
eCAN-A Message Object Time Stamps	0x006080	ECanaMOTSRegs
eCAN-A Message Object Time-Out	0x0060C0	ECanaMOTORRegs
eCAN-B	0x006200	ECanbRegs
eCAN-B Mail Boxes	0x006300	ECanbMboxes
eCAN-B Local Acceptance Masks	0x006240	ECanbLAMRegs
Peripheral	Starting Address	Structure Variable Name
eCAN-B Message Object Time Stamps	0x006280	ECanbMOTSRegs
eCAN-B Message Object Time-Out	0x0062C0	ECanbMOTORRegs

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
eCAP1	0x006A00	ECap1Regs
eCAP2	0x006A20	ECap2Regs
eCAP3	0x006A40	ECap3Regs
eCAP4	0x006A60	ECap4Regs
eCAP5	0x006A80	ECap5Regs
eCAP6	0x006AA0	ECap6Regs
eQEP1	0x006B00	EQep1Regs
eQEP2	0x006B40	EQep2Regs
eQEP3	0x006B80	EQep3Regs
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
McBSP-A	0x005000	McbspaRegs
McBSP-B	0x005040	McbspbRegs
PIE Control	0x000CE0	PieCtrlRegs
SCI-A	0x007050	SciaRegs
SCI-B	0x007750	ScibRegs
SCI-C	0x007770	ScicRegs
SPI-A	0x007040	SpiaRegs
SPI-D	0x007780	SpidRegs
System Control	0x007010	SysCtrlRegs
XINTF	0x000B20	XintfRegs

8.2 Common Example Code – DSP2834x_common

8.2.1 Peripheral Interrupt Expansion (PIE) Block Support

In addition to the register definitions defined in DSP2834x_PieCtrl.h, this packet provides the basic ISR structure for the PIE block. These files are:

Table 13. Basic PIE Block Specific Support Files

File	Location	Description
DSP2834x_DefaultIsr.c	DSP2834x_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.
DSP2834x_DefaultIsr.h	DSP2834x_common\include	Function prototype statements for the ISRs in DSP2834x_DefaultIsr.c. Note: This file is not used for DSP/BIOS projects.
DSP2834x_PieVect.c	DSP2834x_common\source	Creates an instance of the PIE vector table structure initialized with pointers to the ISR functions in DSP2834x_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 *DSP2834x_examples\sw_prioritized_interrupts* for more information.

Table 14. Software Prioritized Interrupt PIE Block Specific Support Files

File	Location	Description
DSP2834x_SWPrioritizedDefaultIsr.c	DSP2834x_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.
DSP2834x_SWPrioritizedIsrLevels.h	DSP2834x_common\include	Function prototype statements for the ISRs in DSP2834x_SWPrioritizedDefaultIsr.c. Note: This file is not used for DSP/BIOS projects.
DSP2834x_SWPrioritizedPieVect.c	DSP2834x_common\source	Creates an instance of the PIE vector table structure initialized with pointers to the default ISR functions that are included in DSP2834x_SWPrioritizedDefaultIsr.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 *DSP2834x_common\src* directory. These files include:

Table 15. Included Peripheral Specific Files

File	Description
DSP2834x_GlobalPrototypes.h	Function prototypes for the peripheral specific functions included in these files.
DSP2834x_CpuTimers.c	CPU-Timer specific functions and macros.
DSP2834x_DMA.c	DMA specific functions and macros.
DSP2834x_Dma_defines.h	#define macros that are used for the DMA examples.
DSP2834x_ECan.c	Enhanced CAN specific functions and macros.
DSP2834x_ECap.c	eCAP module specific functions and macros.
DSP2834x_EPwm.c	ePWM module specific functions and macros.
DSP2834x_EPwm_defines.h	#define macros that are used for the ePWM examples
DSP2834x_EQep.c	eQEP module specific functions and macros.
DSP2834x_Gpio.c	General-purpose IO (GPIO) specific functions and macros.
DSP2834x_I2C.c	I2C specific functions and macros.
DSP2834x_I2c_defines.h	#define macros that are used for the I2C examples
DSP2834x_Mcbsp.c	McBSP specific functions and macros.
DSP2834x_PieCtrl.c	PIE control specific functions and macros.
DSP2834x_Sci.c	SCI specific functions and macros.
DSP2834x_Spi.c	SPI specific functions and macros.
DSP2834x_SysCtrl.c	System control (watchdog, clock, PLL etc) specific functions and macros.
DSP2834x_Xintf.c	XINTF 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 16. Included Utility Function Source Files

File	Description
DSP2834x_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.
DSP2834x_DBGIER.asm	Assembly function to manipulate the DEBIER register from C.
DSP2834x_DisInt.asm	Disable interrupt and restore interrupt functions. These functions allow you to disable INTM and DBGM and then later restore their state.
DSP2834x_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 <i>DSP2834x_examples\adc</i> for an example of its use.

8.2.4 Example Linker .cmd files

Example memory linker command files are located in the *DSP2834x_common\cmd* directory. For getting started the basic 28346_RAM_Ink.cmd file is suggested and used by many of the included examples.

The SARAM blocks L0, L1, L2, and L3 are mirrored on these devices. For simplicity these memory maps only include one instance of these memory blocks.

Table 17. Included Main Linker Command Files

Memory Linker Command File Examples	Location	Description
28346_RAM_Ink.cmd	DSP2834x_common\cmd	28346 memory linker command file. Includes all of the internal SARAM blocks on a 28346 device. "RAM" linker files do not include flash or OTP blocks.
28345_RAM_Ink.cmd	DSP2834x_common\cmd	28345 SARAM memory linker command file.
28344_RAM_Ink.cmd	DSP2834x_common\cmd	28344 SARAM memory linker command file.
28343_RAM_Ink.cmd	DSP2834x_common\cmd	28343 SARAM memory linker command file.
28342_RAM_Ink.cmd	DSP2834x_common\cmd	28342 SARAM memory linker command file.
28341_RAM_Ink.cmd	DSP2834x_common\cmd	28341 SARAM memory linker command file.
2834F_RAM_Ink.cmd	DSP2834x_common\cmd	2834F SARAM memory linker command file.

8.2.5 Included Library .lib Files

Example library files are located in the *DSP2834x_common\lib* directory. The SFO libraries are included for use in the example projects. Please refer to the device-specific HRPWM reference guide for more information on SFO library usage and the HRPWM module.

Table 18. Included Library Files

Main Liner Command File Examples	Description
SFO_TI_Build.lib	Please refer to the <i>device-specific HRPWM Reference Guide</i> for more information on the SFO library
SFO_TI_Build_fpu.lib	The floating-point equivalent of SFO_TI_Build.lib. See Section 4.5 for information about including fixed and floating point libraries.
SFO.h	SFO header file
SFO_TI_Build_V5B.lib	Please refer to the device-specific HRPWM Reference Guide 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_TI_Build_V5B_fpu.lib	The floating-point equivalent of SFO_TI_Build_V5.lib. See Section 4.5 for information about including fixed and floating point libraries. Updated versions will be marked with alphabetical characters after "V5" (i.e. SFO_TI_Build_V5B.lib)
SFO_V5.h	SFO V5 header file

8.2.6 IQmath and FPU fastRTS Library

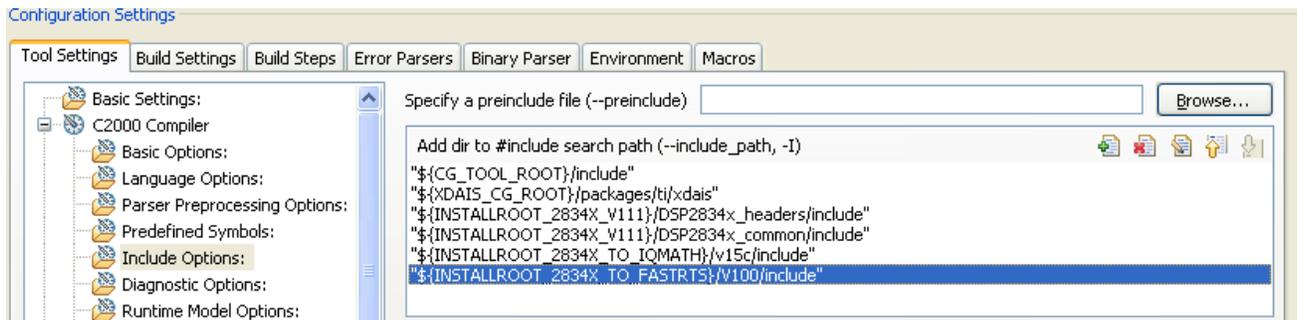
The example projects are configured to look for the IQmath and fastRTS libraries in their controlSUITE install location. For more information on using these two libraries, refer to the documentation in the /doc directory of their installation directory.

- fastRTS library located at c:\ti\controlSUITE\libs\math\FPUfastRTS\<version>

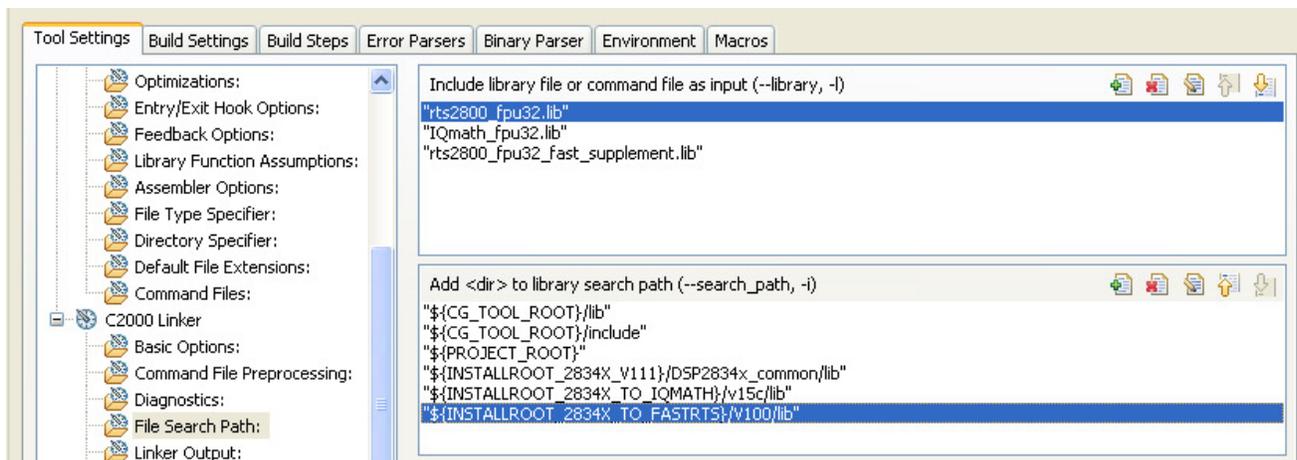
Note: As provided, the fastRTS library is linked before the normal 28x RTS library. This means the fastRTS functions will take priority. To change this configuration, please consult the instructions in the fastRTS library documentation.

- IQmath library located at c:\ti\controlSUITE\libs\math\IQmath\<version>

To modify the <version> of the library used, edit the search path parameters for the include file and the library file in the Project->Properties, C/C++ Options, Include Options:



And the Project->Properties, C2000 Linker, Library File Search Path:



The directory macros INSTALLROOT_2834X_TO_FASTRTS and INSTALLROOT_2834X_TO_IQMATH are defined in the macros.ini file in the directory of each project.

9 Detailed Revision History

Version 1.11 – controlSUITE update

- This update makes numerous improvements to the Code Composer Studio 4 projects.
 - Updated the CCS 4 projects to fit within the controlSUITE directory structure.
 - Added the search path to the controlSUITE install of the fastRTS and IQmath libraries to each project.
 - Removed the local copy of the IQmath library and header file.
 - Linked in the fastRTS library to each project. The fastRTS is linked in before the normal RTS. Refer to the fastRTS library documentation for information.
 - Removed the “release” configuration of each project as it has not been properly setup and tested.
 - Reviewed the compiler and linker switches and updated them as needed.
 - Linked this document to each project so it can be easily opened. Right click on the document and select “open with” and “system editor”.
 - Increased the delay in the GPIO Toggle example.

Changes from V1.00 to V1.10

Changes to Header Files:

- a) **DSP2834x_Device.h-** Added new devices: 28345, 28344, 28342, and 28341.
- b) **DSP2834x_Xintf.h-** Replaced XBANK_BITS in XRESET structure with XRESET_BITS.

Changes to Common Files:

- a) **DSP2834x_SysCtrl.c-** Change HISPCP register to equal 0x1F by default instead of 0x0001.
- b) **Added /DSP2834x_common/gel/ccsv4/ directory-** Because CCSv4.0 does not support the “WatchAdd()” gel command, new device gels without any “WatchAdd()” entries have been generated for CCSv4.0. These are located in the /ccsv4/ directory.
- c) **All gel files-** Removed XINTF_Enable() function call from OnPreFileLoaded() function but maintained the function in the file.
- d) **Added new C28345, C28344, C28342, C28341 gel files** – Added new gel files for C28345, C28344, C28342, and C28341 devices.
- e) **DSP2834x_CpuTimers.c-** Modified comments – CPUtimer 2 is reserved for use by DSP/BIOS. When using DSP/BIOS, all CPUtimer 2 code should be commented out.

- f) **All .cmd files-** Moved BOOT_RSVD to PAGE1 instead of PAGE0.
- g) **Added new C28345, C28344, C28342, C28341 RAM_Ink .cmd files-** Added new linker command files for C28345, C28344, C28342, and C28341 devices.
- h) **DSP2834x_Examples.h-** Updated comments such that DIVSEL=4 (/8 by default). Added new devices – 28345, 28344, 28342, 28341.

Changes to Example Files:

- 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_2834xHaltWake.c** – Updated Description comments concerning halt wakeup procedure using GPIO0.
- c) **Added DSP2834x_examples_ccsv4 directory** - Added directory for CCSv4.x projects. The example projects in this directory are identical to those found in the normal CCSv3.x DSP2834x_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.

V1.00

- This version is the first customer release of the DSP2834x header files and examples.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Communications and Telecom	www.ti.com/communications
DSP	dsp.ti.com	Computers and Peripherals	www.ti.com/computers
Clocks and Timers	www.ti.com/clocks	Consumer Electronics	www.ti.com/consumer-apps
Interface	interface.ti.com	Energy	www.ti.com/energy
Logic	logic.ti.com	Industrial	www.ti.com/industrial
Power Mgmt	power.ti.com	Medical	www.ti.com/medical
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video and Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless-apps

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2010, Texas Instruments Incorporated