F2833x-F2823x Firmware Development Package

USER'S GUIDE

Copyright

Texas Instruments
12203 Southwest Freeway
Houston, TX 77477
http://www.ti.com/c2000

Revision Information

This is version v140 of this document, last updated on March 4, 2015.

Table of Contents

1 Introduction

The Texas Instruments® F2833x Firmware Development Package is a collection of device header files, common source files, helper libraries and example applications for the 2833x/2823x line of devices in the Delfino portfolio.

The package comes with a complete set of example projects that demonstrate the basics of getting started with a Delfino device and working with its different peripheral modules.

Chapter 4 covers all the examples provided in the F2823x development package; what each example does, its setup and observation procedures and, in a few cases, the mathematics involved in setting up control values for peripherals.
Chapter 5 covers all the examples provided in the F2833x development package; what each example does, its setup and observation procedures and, in a few cases, the mathematics involved in setting up control values for peripherals.

The examples for Delfino F2823x can be found in the *DSP2823x_examples_ccsv5* directory. The examples for Delfino F2833x can be found in the *DSP2833x_examples_ccsv5* directory. As users move past evaluation, and get started developing their own application, TI recommends they maintain a similar project directory structure to that used in the example projects.

1.1 Detailed Revision History

V1.40

- Added CCSv5 Examples (Compatible with CCSv6)
- Pievect table set to volatile
- XINTF example updated to handle sharing TZ3 on GPIO14
- McBSP examples updated to setup correct PIE Group
- SW Prioritized interrupts updated to align with pipeline behavior
- McBSP ABIS mode removed. Not supported.
- "interrupt" and "asm" keywords changed to "__interrupt" and "__asm"
- Fixed source and header file case-sensitive issues
- Flash examples updated with appropriate EALLOW and EDIS
- Timer examples updated to write correct value to TCR register
- eCANBack2Back updated to read correct mailboxes
- SPI examples updated to remove ambiguous code
- CAN example comments updated
- DMA examples updated to halt DMA correctly
- Removed MemCopy.c file
- Added missing I2CEMCDR register to header file

V1.33

- Fixed issue in adc_soc example

V1.32

- Updated CCSv4 examples to fit within the controlSUITE directory structure
- Added the search path to the controlSUITE install of the fastRTS and IQmath libraries in each project
- Removed the local copy of the IQmath library and header file
- Linked the fastRTS library to each project
- Removed the "release" configuration from the example projects
- Linked in documents to the example projects
- Fixed delay issue in GPIO toggle example

V1.31

- Fixed PJT files for easy migration to CCSv4

V1.30

- Removed incorrect bits from DEVICECNF register
- Fixed CSM_PWL address in BIOS header linker file
- Fixed XRESET register so that it uses XRESET bit structure instead of XBANK bit structure
- Cleared up comments in DSP2833x_SysCtrl.c
- Moved BOOT_RSVD in linker files to PAGE1
- Added new gel files for CCSv4.0
- Updated comments to state that CPUTimer 2 is reserved for use by DSP/BIOS
- Renamed McBSP examples from McBSP to Mcbsp
- Updated HaltWake example description comments
- Added folders for CCSv4.x examples that support the new CCS format

V1.20

- Updated SPI SPIPRI register bit 6 to reserved
- Added DSP2833x_DualMap_EPWM.gel file to enable dual-mapping of EPWM registers to DMA-accessible memory
- Updated McBSP MFFINT register bits 1 and 3 to reserved
- Added new register structure PartIdRegs
- Added comments to include EPWM SOC signals as DMA triggers
- Added delay_loop() function to use in McBSP
- Corrected I2C_DEINFES to I2C_DEFINES in DSP280x_I2c_defines.h
- Updated ECan baud rate frequencies to account for CANCLK = SYSCLK/2
- Added NOP to DMAInitialize() function after HARDRESET
- Added f2823x gel files
- Added version 1.5 of IQ math libraries (fixed and floating)
- Added example to demonstrate dual-mapping of EPWM registers
- Added code to enable HRPWM logic prior to calling SFO_MepDis_V5() in HRPWM SFO example
- Updated description of UPPS bit settings in EQEP freqcal example
- Fixed various example warnings

- Added example build options to EQEP examples to use FPU

V1.10

- Collapsed eCAN register submenus into one submenu to reduce GEL submenu size
- Added types for int64 and Uint64
- Fixed incorrect comments in BIO and nonBIOS header linker files
- Removed EMPTY_ISR() references from default ISR source files
- Added DSP2823x_examples folder with DSP2833x examples compiled with fixed-point instead of floating-point

V1.03

- Added ECAP5/ECAP6 sections in BIOS header linker file and removed PIE_VECT sections
- Added missing GPIO bit field, QUALPRD1, to GPBCTRL_BITS struct
- Fixed some PIEIER number typos
- Added SFO_TI_BUILD_V5B.lib and SFO_TI_BUILD_V5Bfpu.lib libraries that includes SFO_MepEN() function that supports all available HRPWM configurations
- Updated ECanBack2Back example with ECan initialize function call located within DSP2833x_ECan.c
- Fixed HRPWM example to use correct 50

V1.02

- Removed references to SPI-B to SPI-D (Legacy from DSP280x)
- Removed McBSP GPAQSEL bit field updates to output only MDXA and MDXB GPIO pin configurations
- Added ADC calibration function to GEL files
- Fixed register access issue within Xintf examples

V1.01

- Corrected location of external interrupt registers in DSP2833x_Peripheral.gel
- Added hooks to configure Flash and OTP waitstates at 150MHz SYSCLKOUT
- Removed McBSP MFFST register
- Renamed DSP2833x_DefaultISR.h to DSP2833x_DefaultIsr.h
- Updated GPIO toggling pin in Example_2833xFlash.c example from 34 to 32

V1.00

- This version is the first release of the DSP2833x header files and examples.

1.2 Where Files are Located (Directory Structure)

As installed, the F2833x C/C++ Header Files and Peripheral Examples is partitioned into a well-defined directory structure. This is described in the table below.

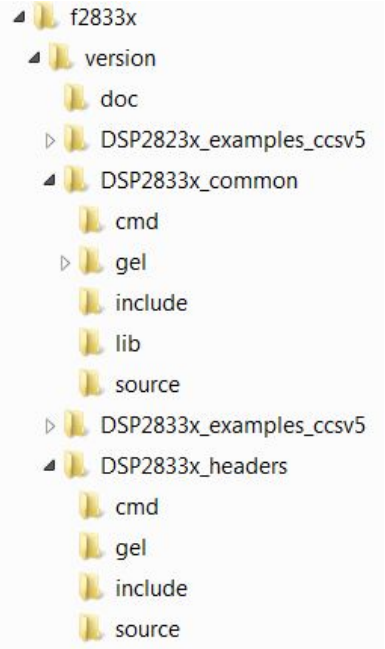

Figure 1.1: The contents of the main directories used by F2833x header files and peripheral examples

Directory	Description
<base>	Base install directory
<base>$\backslash doc$	Documentation including the revision history from the previous release.
<base>$\backslash DSP2833x\_headers$	Files required to incorporate the peripheral header files into a project.
<base>$\backslash DSP2823x\_examples\_ccsv5$	Example Code Composer Studio v5 projects. These example projects illustrate how to configure many of the on-chip peripherals. These examples have the main software configured for the 2823x device.
<base>$\backslash DSP2833x\_examples\_ccsv5$	Example Code Composer Studio v5 projects. These example projects illustrate how to configure many of the on-chip peripherals. These examples have the main software configured for the 2833x device.
<base>$\backslash DSP2833x\_common$	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.

F2833x Main Directory Structure

Under the DSP2833x_headers and DSP2833x_common the source files are further broken down into sub-directories each indicating the type of file. This table lists the subdirectories and describes the types of files found within each:

Sub-Directory	Description
DSP2833x_headers\cmd	Linker command files that allocate the bit-field structures.
DSP2833x_headers\$source$	Source files required to incorporate the header files into a new or existing project.
DSP2833x_headers\$include$	Header files for each of the on-chip peripherals.
DSP2833x_common\cmd	Example memory command files that allocate memory on the devices.
DSP2833x_common\$include$	Common .h files that are used by the peripheral examples.
DSP2833x_common\$source$	Common .c files that are used by the peripheral examples.
DSP2833x_common\lib	Common library (.lib) files that are used by the peripheral examples.
DSP2833x_common\$gel\ccsv4$	Code Composer Studio v4.x GEL files for each device. These are optional.

F2833x Sub-Directory Structure

2 Getting Started and Troubleshooting

2.1 Introduction

Because of the sheer complexity of the F2833x/F2823x devices, it is not uncommon for new users to have trouble bringing up the device their first time. This guide aims to give you, the user, a step by step guide for how to create and debug projects from scratch.

2.2 Project Creation

A typical F2833x application consists of setting up a CCS project, which involves configuring the build settings, file linking, and adding in any source code.

CCS Project Creation

1. From the main CCS window select File -> New -> CCS Project. Name your project and choose a location for it to reside. Click Finish and your project will be created.

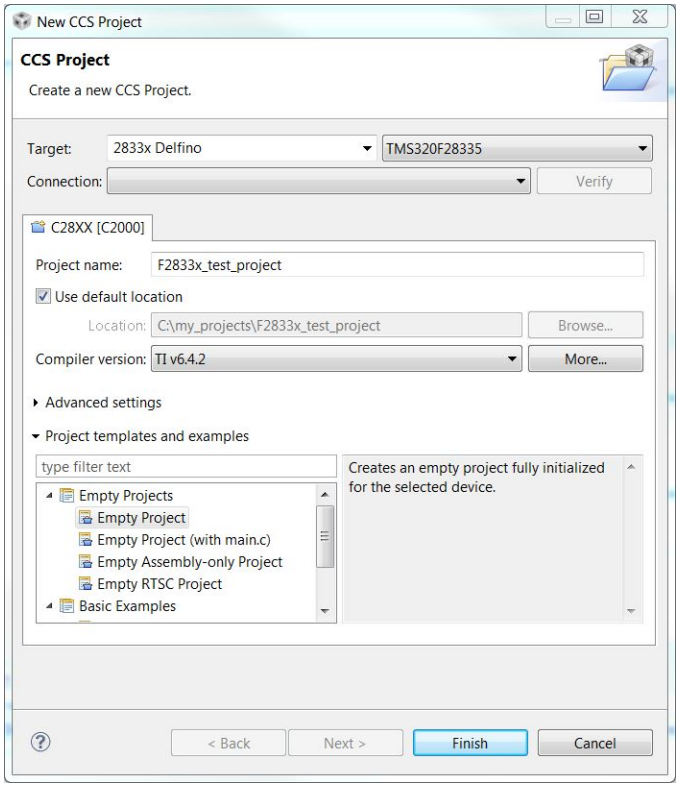

Figure 2.1: Creating a new C28 project

2. Before we can successfully build a project we need to setup some build specific settings. Right click on your project and select Properties. Look at the Processor Options and ensure they match the below image:

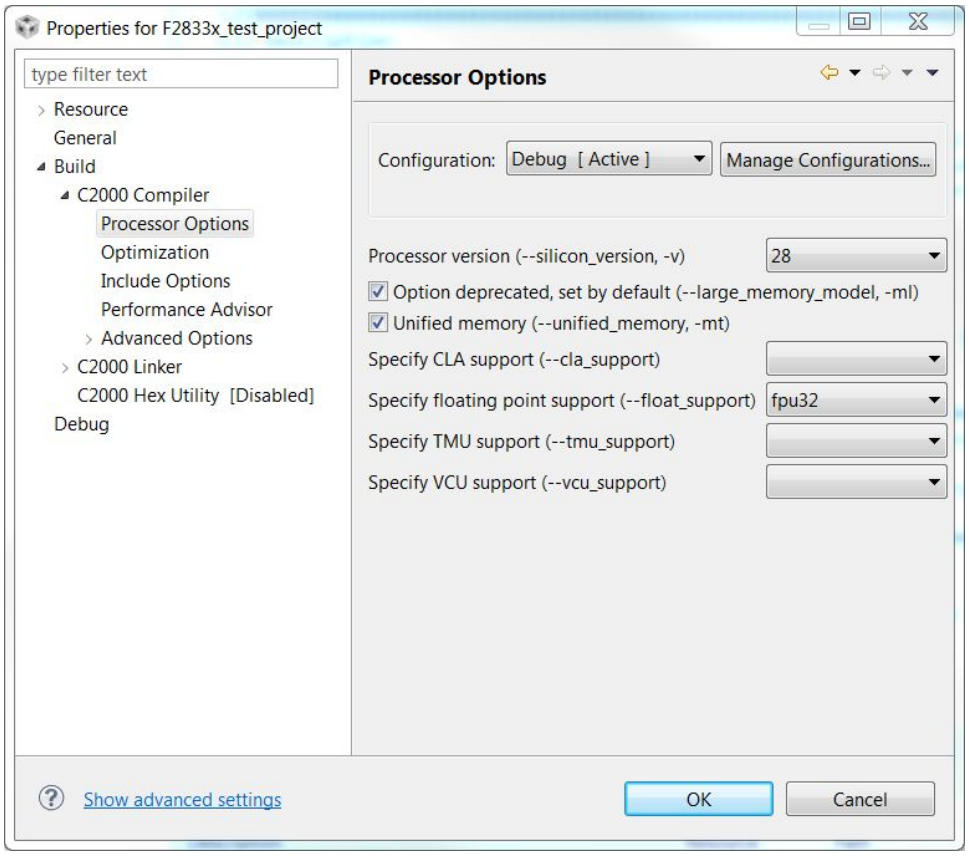

Figure 2.2: Setup Processor Options

3. In the C2000 Compiler entry look for and select the Include Options. Click on the add directory icon to add a directory to the search path. Click the File System button to browse to the `DSP2833x_common\include` folder of your controlSUITE installation (typically `C:\TI\controlSUITE\device_support\f2833x\<Version>\DSP2833x_common\include`). Click ok to add this path, and repeat this same process to add the `DSP2833x_headers\include` directory.

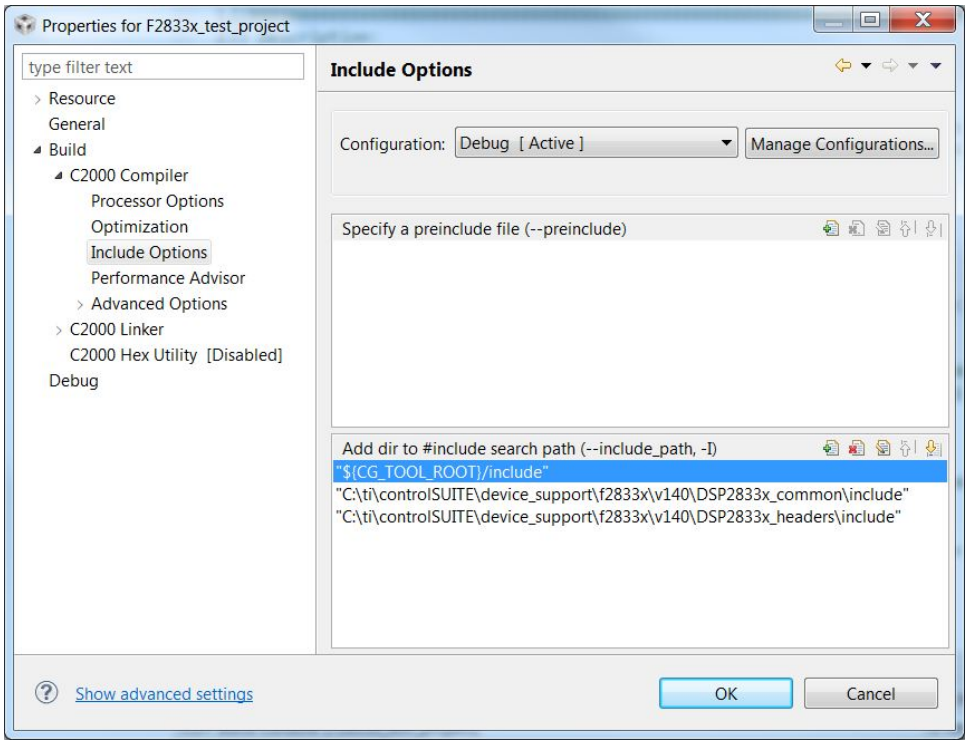

Figure 2.3: Setup Compiler Include Paths

4. Click on the Linker File Search Path. Add these directories to the search path: `DSP2833x_common\cmd` and `DSP2833x_headers\cmd`. Then you'll also want to add the following files: `rts2800_fpu32.lib`, `28335_RAM_lnk.cmd`, and `DSP2833x_Headers_nonBIOS.cmd`. Finally, delete libc.a, we will use rts2800_fpu32.lib as our run time support library instead.

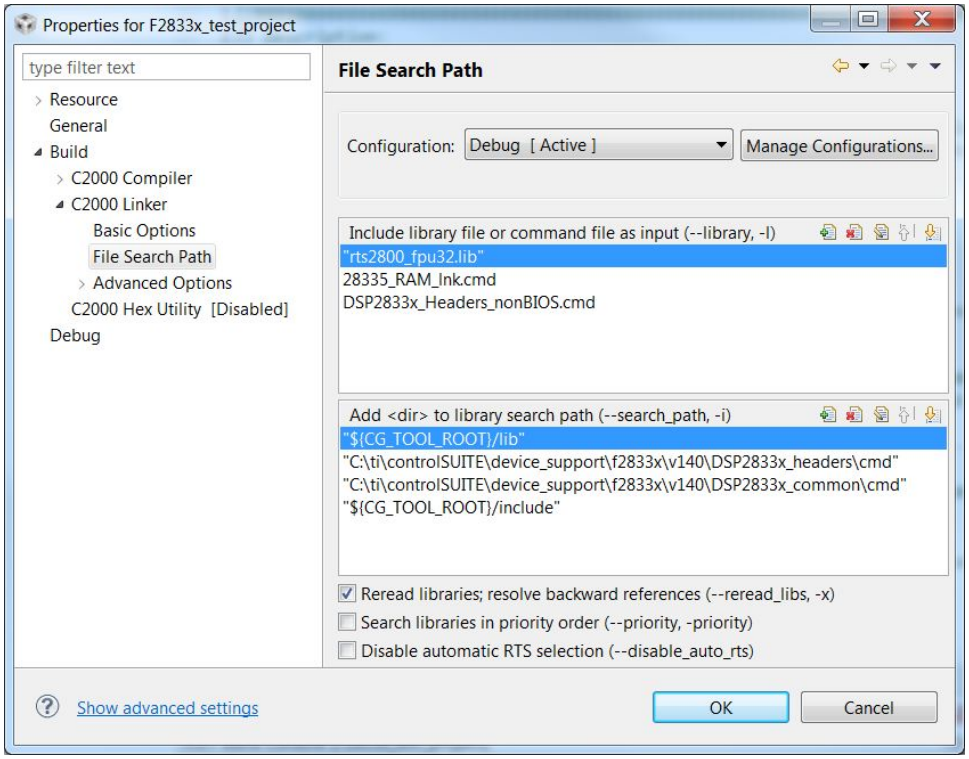

Figure 2.4: Setup Linker Include Paths

5. While you have this window open select the Symbol Management options under C2000 Linker Advanced Options. Specify the program entry point to be `code_start`. Select ok to close out of the Build Properties.

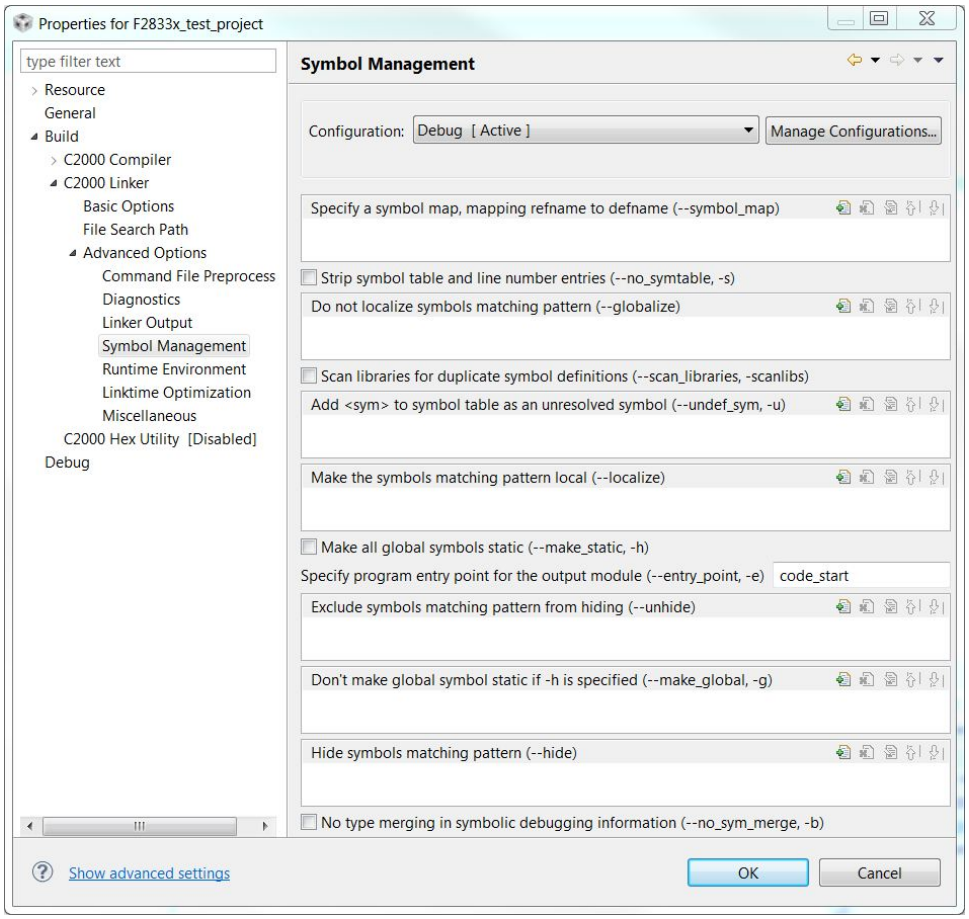

Figure 2.5: Setup Program Entry Point

6. Next we need to link in a few files which are used by the header files. To do this right click on your project in the workspace and select Add Files... Navigate to the `DSP2833x_headers\source` directory, and select DSP2833x_GlobalVariableDefs.c. After you select the file you'll have the option to copy the file into the project or link it. We recommend you link files like this to the project as you will probably not modify these files. Remove any linker command files that were added by default when the project was setup, then link in the following files as well:
   - `DSP2833x_common\source\DSP2833x_ADC_cal.asm`
   - `DSP2833x_common\source\DSP2833x_CodeStartBranch.asm`
   - `DSP2833x_common\source\DSP2833x_Gpio.c`
   - `DSP2833x_common\source\DSP2833x_SysCtrl.c`
   - `DSP2833x_common\source\DSP2833x_usDelay.asm`

   At this point your project workspace should look like the following:

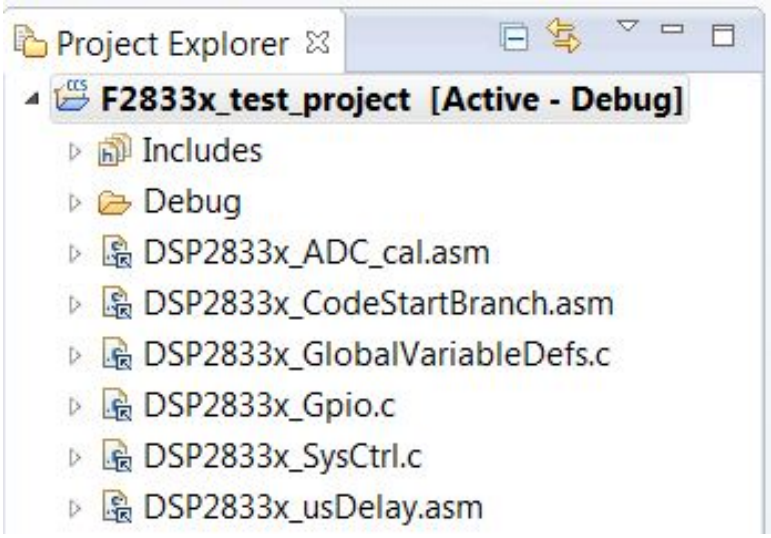

Figure 2.6: Linking files to project

7. Create a new file by right clicking on the project and selecting New -> File. Name this file main.c and copy the following code into it:

```
#include "DSP28x_Project.h"

void main(void)
{
   Uint32 delay;

   InitSysCtrl();

   // Configure GPIO34 as a GPIO output pin
   EALLOW;
   GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 0;
   GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1;
   EDIS;

   while(1)
   {
        // Toggle LED
        GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1;

        // Delay for a bit
        for(delay = 0; delay < 2000000; delay++)
        {

        }

        // Toggle LED
        GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1;

        // Delay for a bit
        for(delay = 0; delay < 2000000; delay++)
        {

        }
   }
}
```

8. Save main.c and then attempt to build the project by right click on it and selecting Build Project. Assuming the project builds try debugging this project on a f2833x device. When the code runs you should see GPIO 34 toggle.

2.3 Troubleshooting

There are a number of things that can cause the user trouble while bringing up a debug session the first time. This section will try to provide solutions to the most common problems encountered with the Delfino devices.

"I get a managed make error when I import the example projects"

This occurs when one imports a project for which he or she doesn't have the code generation tools for. Please ensure that you have at least version 6.2.0 of the C2000 Code Generation Tools.

"I cannot build the example projects"

This is caused by linked resources not being where the project expects them to be. For instance, if you imported the projects and selected "Copy projects to workspace", the projects would no longer build because the files they reference aren't a part of your workspace. Always build and run the examples directly in the controlSUITE tree.

"My F2833x/F2823x device isn't in the target configuration selection list"

The list of available device for debug is determined based on a number of factors, including drivers and tools chains available on the host system. If you system has previously been used only for development on previous C2000 devices, you may not have the required CCS device files. In CCS click on "Help, Check for updates" and follow the dialog boxes to update your CCS installation.

"I cannot connect to the target"

This is most often times caused by either a bad target configuration, or simply the emulator being physically disconnected. If you are unable to connect to a target check the following things:

1. Ensure the target configuration is correct for the device you have.
2. Ensure the emulator is plugged in to both the computer and the device to be debugged.
3. Ensure that the target device is powered.

"I cannot load code"

This is typically caused by an error in the GEL script or improperly linked code. GEL files shipped in controlSUITE are tested and should work without modification with f2833x devices, but advanced users may potentially alter GEL files depending on their overall system configuration. If you are having trouble loading code, check the linker command files and maps to ensure that they match the device memory map. If these appear correct, there is a chance there is something wrong in one of your GEL scripts.

"When a core gets an interrupt, it faults"

Ensure that the interrupt vector table is where the interrupt controller thinks it is. On the core, the interrupt vector table may be mapped to either RAM or flash. Please ensure that your vector table is where the interrupt controller thinks it is.

"When the CPU comes up, it is not fresh out of reset"

F2833x/F2823x devices support several boot modes, several of which allow program code to be loaded into and executed out of RAM via one of the device many serial peripherals. If the boot mode pins are in the wrong state at power up, one of these peripheral boot modes may be entered accidently before the debugger is connected. This leaves the chip in an unclean state with potentially several of the peripherals configured as well as the interrupt vector table setup. If you are seeing strange behavior check to ensure that the "Boot to Flash" or "Boot to RAM" boot mode is selected.

3 Interrupt Service Routine Priorities

3.1 Interrupt Hardware Priority Overview

With the PIE block enabled, the interrupts are prioritized in hardware by default as follows:
Global Priority (CPU Interrupt level):

CPU Interrupt	Hardware Priority
Reset	1(Highest)
INT1	5
INT2	6
INT3	7
INT4	8
INT5	9
INT6	10
INT7	11
...	...
INT12	16
INT13	17
INT14	18
DLOGINT	19(Lowest)
RTOSINT	20
reserved	2
NMI	3
ILLEGAL	-
USER1	-(Software Interrupts)
USER2	-
...	...

CPU Interrupts INT1 - INT14, DLOGINT and RTOSINT are maskable interrupts. These interrupts can be enabled or disabled by the CPU Interrupt enable register (IER).

Group Priority (PIE Level):
If the Peripheral Interrupt Expansion (PIE) block is enabled, then CPU interrupts INT1 to INT12 are connected to the PIE. This peripheral expands each of these 12 CPU interrupt into 8 interrupts. Thus the total possible number of available interrupts in the PIE is 96. Note, not all of the 96 are used on a 2833x device.

Each of the PIE groups has its own interrupt enable register (PIEIERx) to control which of the 8 interrupts (INTx.1 - INTx.8) are enabled and permitted to issue an interrupt.

CPU Interrupt	PIE Group	PIE Interrupts							
		Highest————Hardware Priority Within the Group————-Lowest							
INT1	1	INT1.1	INT1.2	INT1.3	INT1.4	INT1.5	INT1.6	INT1.7	INT1.8
INT2	2	INT2.1	INT2.2	INT2.3	INT2.4	INT2.5	INT2.6	INT2.7	INT2.8
INT3	3	INT3.1	INT3.2	INT3.3	INT3.4	INT3.5	INT3.6	INT3.7	INT3.8
... etc ...									
... etc ...									
INT12	12	INT12.1	INT12.2	INT12.3	INT12.4	INT12.5	INT12.6	INT12.7	INT4.8

Table 3.1: PIE Group Hardware Priority

3.2 PIE Interrupt Priorities

The PIE block is organized such that the interrupts are in a logical order. Interrupts that typically require higher priority, are organized higher up in the table and will thus be serviced with a higher priority by default.

The interrupts in a control subsystem can be categorized as follows (ordered highest to lowest priority):

1. **Non-Periodic, Fast Response**
   These are interrupts that can happen at any time and when they occur, they must be serviced as quickly as possible. Typically these interrupts monitor an external event.

   On the F2833x devices, such interrupts are allocated to the first few interrupts within PIE Group 1 and PIE Group 2. This position gives them the highest priority within the PIE group. In addition, Group 1 is multiplexed into the CPU interrupt INT1. CPU INT1 has the highest hardware priority. PIE Group 2 is multiplexed into the CPU INT2 which is the 2nd highest hardware priority.

2. **Periodic, Fast Response**
   These interrupts occur at a known period, and when they do occur, they must be serviced as quickly as possible to minimize latency. The A/D converter is one good example of this. The A/D sample must be processed with minimum latency.

   On the F2833x devices, such interrupts are allocated to the group 1 in the PIE table. Group 1 is multiplexed into the CPU INT1. CPU INT1 has the highest hardware priority

3. **Periodic**
   These interrupts occur at a known period and must be serviced before the next interrupt. Some of the PWM interrupts are an example of this. Many of the registers are shadowed, so the user has the full period to update the register values.

   In the F2833x device's PIE modules, such interrupts are mapped to group 2 - group 5. These groups are multiplexed into CPU INT3 to INT5 (the ePWM and eCAP), which are the next lowest hardware priority.

4. **Periodic, Buffered**
   These interrupts occur at periodic events, but are buffered and hence the processor need

only service such interrupts when the buffers are ready to filled/emptied. All of the serial ports (SCI / SPI / I2C / CAN) either have FIFOs or multiple mailboxes such that the CPU has plenty of time to respond to the events without fear of losing data.

In the F2833x device, such interrupts are mapped to INT6, INT8, and INT9, which are the next lowest hardware priority.

3.3 Software Prioritization of Interrupts

The user will probably find that the PIE interrupts are organized where they should be for most applications. However, some software prioritization may still be required for some applications.

Recall that the basic software priority scheme on the C28x works as follows:

- **Global Priority**
  This priority can be managed by manipulating the CPU IER register. This register controls the 16 maskable CPU interrupts (INT1 - INT16).

- **Group Priority**
  This can be managed by manipulating the PIE block interrupt enable registers (PIEIERx). There is one PIEIERx per group and each control the 8-interrupts multiplexed within that group.

The DSP28 software prioritization of interrupt example demonstrates how to configure the Global priority (via IER) and group priority (via PIEIERx) within an ISR in order to change the interrupt service priority based on user assigned levels. The steps required to do this are:

1. **Set the global priority**
   Modify the IER register to allow CPU interrupts with a higher user priority to be serviced.

2. **Set the Group priority**
   Modify the appropriate PIEIERx register to allow group interrupts with a higher user set priority to be serviced.

3. **Enable interrupts**

The software prioritized interrupts example provides a method using mask values that are configured during compile time to allow you to manage this easily.

To setup software prioritization for the example, the user must first assign the desired global priority levels and group priority levels.

This is done in the DSP2833x_common/include/DSP2833x_SWPrioritizedIsrLevels.h file as follows:

1. *User assigns global priority levels*
   INT1PL - INT16PL
   These values are used to assign a priority level to each of the 16 interrupts controlled by the CPU IER register. A value of 1 is the highest priority while a value of 16 is the lowest. More then one interrupt can be assigned the same priority level. In this case the default hardware priority would determine which would be serviced first. A priority of 0 is used to indicate that

the interrupt is not used.

2. *User assigns PIE group priority levels*
   GxyPL (where x = PIE group number 1 - 12 and y = interrupt number 1 - 8)

   These values are used to assign a priority level to each of the 8 interrupts within a PIE group. A value of 1 is the highest priority while a value of 8 is the lowest. More then one interrupt can be assigned the same priority level. In this case the default hardware priority would determine which would be serviced first. A priority of 0 is used to indicate that the interrupt is not used.

Once the user has defined the global and group priority levels, the compiler will generate mask values that can be used to change the IER and PIEIERx registers within each ISR. In this manner the interrupt software prioritization will be changed. The masks that are generated at compile time are:

- **IER mask values**
  MINT1 - MINT16

  The user assigned INT1PL - INT16PL values are used at compile time to calculate an IER mask for each CPU interrupt. This mask value will be used within an ISR to allow CPU interrupts with a higher priority to interrupt the current ISR and thus be serviced at a higher priority level.
- **PIEIERxy mask values**
  MGxy (where x = PIE group number 1 - 12 and y = interrupt number 1 - 8)

  The assigned group priority levels (GxyPL) are used at compile time to calculate PIEIERx masks for each PIE group. This mask value will be used within an ISR to allow interrupts within the same group that have a higher assigned priority to interrupt the current ISR and thus be serviced at a higher priority level.

3.3.1 Using the IER/PIEIER Mask Values

Within an interrupt service routine, the global and group priority can be changed by software to allow other interrupts to be serviced. The procedure for setting an interrupt priority using the mask values created in the DSP28_SWPrioritizedIsrLevels.h is the following:

1. **Set the global priority**
   - Modify IER to allow CPU interrupts from the same PIE group as the current ISR.
   - Modify IER to allow CPU interrupts with a higher user defined priority to be serviced.
2. **Set the group priority**
   - Save the current PIEIERx value to a temporary register.
   - The PIEIER register is then set to allow interrupts with a higher priority within a PIE group to be serviced.
3. **Enable interrupts**
   - Enable all PIE interrupt groups by writing all 1's to the PIEACK register
   - Enable global interrupts by clearing INTM
4. **Execute ISR.** Interrupts that were enabled in steps 1-3 (those with a higher software priority) will be allowed to interrupt the current ISR and thus be serviced first.

5. **Restore the PIEIERx register**
6. **Exit**

3.3.2 Example Code

The sample C code below shows an EV-A Comparator 1 Interrupt service routine software prioritization written in C. This interrupt is connected to PIE group 2 interrupt 1.

```
// Connected to PIEIER2_1 (use MINT2 and MG21 masks):
#if (G21PL != 0)
interrupt void EPWM1_TZINT_ISR(void)   // EPWM1 Trip Zone
{
    // Set interrupt priority:
    volatile Uint16 TempPIEIER = PieCtrlRegs.PIEIER2.all;
    IER |= M_INT2;
    IER &= MINT2;                      // Set "global" priority
    PieCtrlRegs.PIEIER2.all &= MG21;   // Set "group"  priority
    PieCtrlRegs.PIEACK.all = 0xFFFF;   // Enable PIE interrupts
    asm(" NOP");
    EINT;

    // Insert ISR Code here.......
    // for now just insert a delay
    for(i = 1; i <= 10; i++) {}

    // Restore registers saved:
    DINT;
    PieCtrlRegs.PIEIER2.all = TempPIEIER;

    //  Add ISR to Trace
    ISRTrace[ISRTraceIndex] = 0x0021;
    ISRTraceIndex++;
}
#endif

CMP1INT_ISR:
            ASP
            ADDB    SP,#1
            CLRC    OVM,PAGE0
            MOVW    DP,#0x0033
            MOV     AL,@36
            MOV     *-SP[1],AL
            OR      IER,#0x0002
            AND     IER,#0x0002
            AND     @36,#0x000E
            MOV     @33,#0xFFFF
            CLRC    INTM

            User code goes here...
```

```
SETC    INTM
MOV     AL,*-SP[1]
MOV     @36,AL
SUBB    SP,#1
NASP
IRET
```

The interrupt latency is approx 22 cycles.

/*!

4 Delfino F2823x Example Applications

These example applications show the user how to make use of various peripherals present on the F2823x device. They are intended for demonstration purposes only and a good starting point for building new applications.

All of these examples reside in the `device_support/f2833x/<version>/DSP2823x_examples_ccsv5` subdirectory of the ControlSUITE package.

4.1 ADC to DMA (adc_dma)

This ADC example uses ADC to convert 4 channels for each SOC received, with total of 10 SOCs. Each SOC initiates 4 conversions. DMA is used to capture the data on each SEQ1_INT. DMA will re-sort the data by channel sequentially, i.e. all channel0 data will be together and all channel1 data will be together.

Watch Variables

- DMABuf1 - DMA Buffer

4.2 ADC Seq Override Mode Test (add_seq_ovd_test)

In this example, channel A0 is converted forever and logged in a buffer (SampleTable) using sequencer1 in sequence override mode. Sequencer is Sequential mode with sample rate of 1/(3∗40ns) = 8.3 MHZ.

Watch Variables

- SampleTable - Log of converted values.
- GPIO34 - Toggles on every ADC sequencer flag

4.3 ADC Seq Mode Test (adc_seqmode_test)

In this example, channel A0 is converted forever and logged in a buffer (SampleTable)

Watch Variables

- SampleTable - Log of converted values

4.4 ADC Start of Conversion (adc_soc)

This ADC example uses ePWM1 to generate a periodic ADC SOC on SEQ1. Two channels are converted, ADCINA3 and ADCINA2.

Watch Variables

- Voltage1[10] - Last 10 ADCRESULT0 values
- Voltage2[10] - Last 10 ADCRESULT1 values
- ConversionCount - Current result number 0-9
- LoopCount - Idle loop counter

4.5 Cpu Timer (cpu_timer)

This example configures CPU Timer0, 1, and 2 and increments a counter each time the timers asserts an interrupt.

Watch Variables

- CputTimer0.InterruptCount
- CpuTimer1.InterruptCount
- CpuTimer2.InterruptCount

4.6 DMA Ram to Ram (dma_ram_to_ram)

This example will perform a block copy from L5 SARAM to L4 SARAM of 1024 words. Transfer will be started by Timer0. Will use 32-bit data size to decrease the transfer time.

Watch Variables

- DMABuf1
- DMABuf2

4.7 eCAN-A to eCAN-B Transmit Loop (ecan_a_to_b_xmit)

This example TRANSMITS data to another CAN module using MAILBOX5 This program could either loop forever or transmit "n" # of times, where "n" is the TXCOUNT value.

This example can be used to check CAN-A and CAN-B. Since CAN-B is initialized in DSP2833x_ECan.c, it will acknowledge all frames transmitted by the node on which this code runs. Both CAN ports of the 2833x DSP need to be connected to each other (via CAN transceivers)

External Connections

- eCANA is on GPIO31 and GPIO30
- eCANB is on GPIO8 and GPIO10
- Connect GPIO31 to GPIO8
- Connect GPIO30 to GPIO10

4.8 eCAN back to back (ecan_back2back)

This example tests eCAN by transmitting data back-to-back at high speed without stopping. The received data is verified. Any error is flagged. MBX0 transmits to MBX16, MBX1 transmits to MBX17 and so on....

This program illustrates the use of self-test mode.

Watch Variables

- PassCount
- ErrorCount
- MessageReceivedCount

4.9 eCAP APWM (ecap_epwm)

This program sets up eCAP pins in the APWM mode. This program runs at 150 MHz SYSCLKOUT assuming a 30 MHz XCLKIN or 100 MHz SYSCLKOUT assuming a 20 MHz XCLKIN.

For 150 MHz devices:

- eCAP1 will come out on the GPIO24 pin. This pin is configured to vary between 7.5 Hz and 15 Hz using the shadow registers to load the next period/compare values.
- eCAP2 will come out on the GPIO7 pin. This pin is configured as a 7.5 Hz output.
- eCAP3 will come out on the GPIO9 pin. This pin is configured as a 1.5 Hz output.
- eCAP4 will come out on the GPIO11 pin. This pin is configured as a 30 kHz output.
- All frequencies assume a 30 Mhz input clock. The XCLKOUT pin should show 150Mhz.

For 100 MHz devices:

- eCAP1 will come out on the GPIO24 pin. This pin is configured to vary between 5 Hz and 10 Hz using the shadow registers to load the next period/compare values.
- eCAP2 will come out on the GPIO7 pin. This pin is configured as a 5 Hz output.
- eCAP3 will come out on the GPIO9 pin. This pin is configured as a 1 Hz output.
- eCAP4 will come out on the GPIO11 pin. This pin is configured as a 20kHz output.
- All frequencies assume a 20 Mhz input clock. The XCLKOUT pin should show 100Mhz.

4.10 eCap capture PWM (ecap_capture_pwm)

This example configures ePWM3A for:

- Up count
- Period starts at 2 and goes up to 1000
- Toggle output on PRD

eCAP1 is configured to capture the time between rising and falling edge of the ePWM3A output.

External Connections

- eCap1 is on GPIO24
- ePWM3A is on GPIO4
- Connect GPIO4 to GPIO24.

Watch Variables

- ECap1IntCount - Successful captures
- ECap1PassCount - Interrupt counts

4.11 ePWM Deadband Generation (epwm_deadband)

This example configures ePWM1, ePWM2 and ePWM3 for:

- Count up/down
- Deadband

3 Examples are included:

- ePWM1: Active low PWMs
- ePWM2: Active low complementary PWMs
- ePWM3: Active high complementary PWMs

Each ePWM is configured to interrupt on the 3rd zero event when this happens the deadband is modified such that 0 <= DB <= DB_MAX. That is, the deadband will move up and down between 0 and the maximum value.

External Connections

- EPWM1A is on GPIO0
- EPWM1B is on GPIO1
- EPWM2A is on GPIO2
- EPWM2B is on GPIO3
- EPWM3A is on GPIO4
- EPWM3B is on GPIO5

4.12 ePWM to DMA (epwm_dma)

This example demonstrates several cases where the DMA is triggered from SOC signals generated by ePWM modules.

Watch Variables

- EPwm1Regs.TBPRD

- EPwm1Regs.CMPA.all
- ADCbuffer
- InterruptCount

4.13 ePWM Timer Interrupt (epwm_timer_interrupts)

This example configures the ePWM Timers and increments a counter each time an interrupt is taken. In this example:

- All ePWM's are initialized.
- All timers have the same period.
- The timers are started sync'ed.
- An interrupt is taken on a zero event for each ePWM timer.
- ePWM1: takes an interrupt every event.
- ePWM2: takes an interrupt every 2nd event.
- ePWM3: takes an interrupt every 3rd event.
- ePWM4-ePWM6: takes an interrupt every event.

Thus the Interrupt count for ePWM1, ePWM4, ePWM5, and ePWM6 should be equal.The interrupt count for ePWM2 should be about half that of ePWM1 and the interrupt count for ePWM3 should be about 1/3 that of ePWM1.

Watch Variables

- EPwm1TimerIntCount
- EPwm2TimerIntCount
- EPwm3TimerIntCount
- EPwm4TimerIntCount
- EPwm5TimerIntCount
- EPwm6TimerIntCount

4.14 ePWM Trip Zone (epwm_trip_zone)

This example configures ePWM1 and ePWM2 as follows

- ePWM1 has TZ1 and TZ2 as one shot trip sources
- ePWM2 has TZ1 and TZ2 as cycle by cycle trip sources

Initially tie TZ1 and TZ2 high. During the test, monitor ePWM1 or ePWM2 outputs on a scope. Pull TZ1 or TZ2 low to see the effect.

External Connections

- EPWM1A is on GPIO0
- EPWM1B is on GPIO1

- EPWM2A is on GPIO2
- EPWM2B is on GPIO3
- TZ1 is on GPIO12
- TZ2 is on GPIO13

4.15 ePWM Action Qualifier Module using Upcount mode (epwm_up_aq)

This example configures ePWM1, ePWM2, ePWM3 to produce a waveform with independent modulation on EPWMxA and EPWMxB. The compare values CMPA and CMPB are modified within the ePWM's ISR. The TB counter is in upmode.

Monitor the ePWM1 - ePWM3 pins on an oscilloscope.

External Connections

- EPWM1A is on GPIO0
- EPWM1B is on GPIO1
- EPWM2A is on GPIO2
- EPWM2B is on GPIO3
- EPWM3A is on GPIO4
- EPWM3B is on GPIO5

4.16 ePWM Action Qualifier Module using up/down count (epwm_updown_aq)

This example configures ePWM1, ePWM2, ePWM3 to produce an waveform with independent modulation on EPWMxA and EPWMxB. The compare values CMPA and CMPB are modified within the ePWM's ISR. The TB counter is in up/down count mode for this example.

Monitor ePWM1-ePWM3 pins on an oscilloscope as described

External Connections

- EPWM1A is on GPIO0
- EPWM1B is on GPIO1
- EPWM2A is on GPIO2
- EPWM2B is on GPIO3
- EPWM3A is on GPIO4
- EPWM3B is on GPIO5

4.17 eQEP, Frequency measurement (eqep_freqcal)

This test will calculate the frequency and period of an input signal using eQEP module.

EPWM1A is configured to generate a frequency of 5 kHz.

See also:
section on Frequency Calculation for more details on the frequency calculation performed in this example.

In addition to the main example file, the following files must be included in this project:

- **Example_freqcal.c** , includes all eQEP functions
- **Example_EPwmSetup.c** , sets up EPWM1A for use with this example
- **Example_freqcal.h** , includes initialization values for frequency structure.

The configuration for this example is as follows

- Maximum frequency is configured to 10KHz (BaseFreq)
- Minimum frequency is assumed at 50Hz for capture pre-scalar selection

SPEED_FR: High Frequency Measurement is obtained by counting the external input pulses for 10ms (unit timer set to 100Hz).

$$SPEED\_FR = \frac{Count\ Delta}{10ms}$$

SPEED_PR: Low Frequency Measurement is obtained by measuring time period of input edges. Time measurement is averaged over 64edges for better results and capture unit performs the time measurement using pre-scaled SYSCLK

Note that pre-scaler for capture unit clock is selected such that capture timer does not overflow at the required minimum frequency This example runs forever until the user stops it.

External Connections

Connect GPIO20/EQEP1A to GPIO0/EPWM1A

Watch Variables

- **freq.freqhz_fr** , Frequency measurement using position counter/unit time out
- **freq.freqhz_pr** , Frequency measurement using capture unit

4.17.1 EPWM Setup (Example_EPwmSetup.c)

This file contains source for the ePWM initialization for the freq calculation module. EPWM1 is set to operate in up-down count mode at a frequency of 5KHz

4.17.2 Frequency Calculation (Example_freqal.c)

This file includes the EQEP initialization and frequency calculation functions called by **Example_2823xEqep_freqcal.c**. The frequency calculation steps performed by FREQCAL_Calc()at SYSCLKOUT = 150 MHz and 100 MHZ are described below:

1. This program calculates: $**$**freqhz_fr**$**$ for SYSCLKOUT = 150MHz

$$freqhz\_fr\ or\ v = \frac{x_2 - x_1}{T} .....1$$

If

$$\frac{max}{base}\ freq = 10kHz => 10kHz = \frac{x_2 - x_1}{(2/100Hz)} .....2$$

$$max(x_2 - x_1) = 200counts = freqScaler\_fr$$

Note:
$T = \frac{2}{100Hz}$. 2 is from $\frac{x_2 - x_1}{2}$ because QPOSCNT counts 2 edges per cycle (rising and falling)

If both sides of Equation 2 are divided by 10 kHz, then:

$$1 = \frac{x_2 - x_1}{10kHz * (2/100Hz)}$$

where,

$$[10kHz * \frac{2}{100Hz}] = 200$$

Because

$$x_2 - x_1 < 200(max)$$
$$\frac{x_2 - x_1}{200} < 1$$

for all frequencies less than max

$$freq\_fr = \frac{x_2 - x_1}{200}\ or\ \frac{x_2 - x_1}{10kHz * (2/100Hz)} .....3$$

To get back to original velocity equation, Equation 1, multiply Equation 3 by 10 kHz

$$freqhz\_fr(or\ velocity) = 10kHz * \frac{x_2 - x_1}{10kHz * (2/100Hz)}$$

$$= \frac{x_2 - x_1}{(2/100Hz)} .....final\ equation$$

1. $**$**min freq**$**$ $= \frac{1\ count}{(2/100Hz)} = 50Hz$
2. $**$**freqhz_pr**$**$

$$freqhz\_pr\ or\ v = \frac{X}{t_2 - t_1} .....4$$

If

$$\frac{max}{base}\ freq = 10kHz => 10kHz = \frac{(4/2)}{T} = \frac{4}{2T}$$

where,

- 4 = QCAPCTL [UPPS] (Unit timeout - once every 4 edges)
- 2 = divide by 2 because QPOSCNT counts 2 edges per cycle (rising and falling)
- T = time in seconds = $\frac{t_2-t_1}{(150MHz/128)}$, $t_2 - t_1$ = # of QCAPCLK cycles, and $1\ QCAPCLK\ cycle = \frac{1}{(150MHz/128)} = QCPRDLAT$

So:

$$10kHz = 4 * \frac{(150MHz/128)}{2 * (t_2 - t_1)}$$

$$t_2 - t_1 = 4 * \frac{(150MHz/128)}{10kHz * 2} = \frac{(150MHz/128)}{((2 * 10KHz)/4)} .....5$$

$$= 234\ QCAPCLK\ cycles = maximum(t_2 - t_1) = freqScaler\_pr$$

Divide both sides by $(t_2 - t_1)$, and:

$$1 = \frac{234}{t_2 - t_1} = \frac{(150MHz/128)/((2 * 10KHz)/4)}{t_2 - t_1}$$

Because $(t_2 - t_1) < 234(max)$, $\frac{234}{t_2-t_1} < 1$ for all frequencies less than max

$$freq\_pr = \frac{234}{t_2 - t_1}\ or\ \frac{(150MHz/128)/((2 * 10KHz)/4)}{t_2 - t_1} .....6$$

Now within velocity limits, to get back to original velocity equation, Equation 1, multiply Equation 6 by 10 kHz:

$$freqhz\_fr(or\ velocity) = 10kHz * \frac{(150MHz/128)/((2 * 10KHz)/4)}{t_2 - t_1}$$

$$= \frac{(150MHz/128) * 4}{2 * (t_2 - t_1)}$$

or

$$\frac{4}{2 * (t_2 - t_1) * (QCPRDLAT)} .....final\ equation$$

For 100 MHz Operation:

The same calculations as above are performed, but with 100 MHz instead of 150MHz when calculating freqhz_pr, and at UPPS of 8 instead of 4. The value for freqScaler_pr becomes: (100MHz/128)/(2∗10kHz/8) = 313

More detailed calculation results can be found in the Example_freqcal.xls spreadsheet included in the example folder.

4.18 eQEP Speed and Position measurement (eqep_pos_speed)

This example provides position measurement,speed measurement using the capture unit, and speed measurement using unit time out. This example uses the IQMath library. It is used merely to simplify high-precision calculations.

The example requires the following hardware connections from EPWM1 and GPIO pins (simulating QEP sensor) to QEP peripheral.

- eQEP1A <- ePWM1A (simulates eQEP Phase A signal)
- eQEP1B <- ePWM1B (simulates eQEP Phase B signal)
- eQEP1I <- GPIO4 (simulates eQEP Index Signal)

See DESCRIPTION in Example_posspeed.c for more details on the calculations performed in this example.

In addition to this file, the following files must be included in this project:

- Example_posspeed.c - includes all eQEP functions
- Example_EPwmSetup.c - sets up ePWM1A and ePWM1B as simulated QA and QB encoder signals
- Example_posspeed.h - includes initialization values for pos and speed structure

Note:

- Maximum speed is configured to 6000rpm(BaseRpm)
- Minimum speed is assumed at 10rpm for capture pre-scalar selection
- Pole pair is configured to 2 (pole_pairs)
- QEP Encoder resolution is configured to 4000counts/revolution (mech_scaler)
- which means: 4000/4 = 1000 line/revolution quadrature encoder (simulated by EPWM1)
- EPWM1 (simulating QEP encoder signals) is configured for 5kHz frequency or 300 rpm (=4∗5000 cnts/sec ∗ 60 sec/min)/4000 cnts/rev)
- SPEEDRPM_FR: High Speed Measurement is obtained by counting the QEP input pulses for 10ms (unit timer set to 100Hz).
- SPEEDRPM_FR = (Position Delta/10ms) ∗ 60 rpm
- SPEEDRPM_PR: Low Speed Measurement is obtained by measuring time period of QEP edges. Time measurement is averaged over 64edges for better results and capture unit performs the time measurement using pre-scaled SYSCLK
- pre-scaler for capture unit clock is selected such that capture timer does not overflow at the required minimum RPM speed.

External Connections

- Connect eQEP1A(GPIO20) to ePWM1A(GPIO0)(simulates eQEP Phase A signal)
- Connect eQEP1B(GPIO21) to ePWM1B(GPIO1)(simulates eQEP Phase B signal)
- Connect eQEP1I(GPIO23) to GPIO4 (simulates eQEP Index Signal)

Watch Variables

- qep_posspeed.SpeedRpm_fr - Speed meas. in rpm using QEP position counter
- qep_posspeed.SpeedRpm_pr - Speed meas. in rpm using capture unit
- qep_posspeed.theta_mech - Motor mechanical angle (Q15)
- qep_posspeed.theta_elec - Motor electrical angle (Q15)

4.18.1 EPWM Setup (Example_EPwmSetup.c)

This file contains source for the ePWM initialization for the freq calculation module. EPWM1 is set to operate in up-down count mode at a frequency of 5KHz

4.18.2 Position/Speed Calculation (Example_posspeed.c)

This file includes the EQEP initialization and position and speed calculation functions called by Example_2833xEqep_posspeed.c. The position and speed calculation steps performed by POSSPEED_Calc() at SYSCLKOUT = 150 MHz and 100 MHz are described below:

1. This program calculates: $**$**theta_mech**$**$ for SYSCLKOUT = 150Mhz

$$theta\_mech\ = \frac{QPOSCNT}{mech\_scaler} = \frac{QPOSCNT}{4000}$$

   where 4000 is the number of counts in 1 revolution.(4000/4 = 1000 line/rev. quadrature encoder)

2. This program calculates: $**$**theta_elec**$**$ for SYSCLKOUT = 150MHz

$$theta\_elec\ = (polepairs) * theta\_mech\ = \frac{2 * QPOSCNT}{4000}$$

3. This program calculates: $**$**SpeedRpm_fr**$**$ for SYSCLKOUT = 150Mhz

$$SpeedRpm\_fr\ = \frac{x_2 - x_1}{4000] * T} .....1$$

Note:

$x_2 - x_1$ is the difference in number of QPOSCNT counts. Dividing $x_2 - x_1$ by 4000 gives position relative to Index in one revolution.

If $baseRPM = 6000rpm$:

$$6000rpm = \frac{x_2 - x_1}{4000 * 10ms} .....2$$
$$= \frac{\frac{x_2 - x_1}{4000}}{\frac{.01s * 1min}{60sec}}$$
$$= \frac{\frac{x_2 - x_1}{4000}}{\frac{1}{6000}min}$$

max $x_2 - x_1 = 4000$ counts, or 1 revolution in 10 ms

If both sides of Equation 2 are divided by 6000 rpm, then:

$$1 = \frac{\frac{x_2 - x_1}{4000} rev.}{\frac{1}{6000} min * 6000rpm}$$

Because $x_2 - x_1$ must be $< 4000(max)$ for QPOSCNT increment, $\frac{x_2-x_1}{4000} < 1$ for CW rotation

And because $x_2 - x_1$ must be > -4000 for QPOSCNT decrement, $\frac{x_2-x_1}{4000} > -1$ for CCW rotation

$$speed\_fr = \frac{\frac{x_2-x_1}{4000}}{\frac{1}{6000}min * 6000rpm} = \frac{x_2 - x1}{4000} .....3$$

To convert speed_fr to RPM, multiply Equation 3 by 6000 rpm

$$SpeedRpm\_fr = 6000rpm * \frac{x_2 - x_1}{4000} .....final$$

1. $**$min rpm $**$ = selected at 10 rpm based on CCPS prescaler options available (128 is greatest)
2. $**$SpeedRpm_pr$**$

   $$SpeedRpm\_pr = \frac{X}{t_2 - t_1} .....4$$

   where X = QCAPCTL [UPPS]/4000 rev. (position relative to Index in 1 revolution)

If $\frac{max}{base} speed = 6000rpm : 6000 = \frac{\frac{32}{4000}}{\frac{t_2-t_1}{\frac{150MHz}{128}}}$ where,

- 32 = QCAPCTL [UPPS] (Unit timeout - once every 32 edges)
- $\frac{32}{4000}$ = position in 1 revolution (position as a fraction of 1 revolution)
- $\frac{t_2-t_1}{\frac{150MHz}{128}}, t_2 - t_1$ = # of QCAPCLK cycles
- 1 QCAPCLK cycle = $\frac{1}{\frac{150MHz}{128}}$ = QCPRDLAT

So:

$$6000rpm = \frac{32\frac{150MHz}{128} * 60s/min}{4000(t_2 - t_1)}$$

$$t_2 - t_1 = \frac{32\frac{150MHz}{128} * 60s/min}{4000 * 6000rpm} .....5$$

$$= 94CAPCLKcycles = maximum(t2 - t1) = SpeedScaler$$

Divide both sides by $t_2 - t_1$, and:

$$1 = \frac{94}{t_2 - t_1} = \frac{\frac{32\frac{150MHz}{128} * 60s/min}{4000 * 6000rpm}}{t_2 - t_1}$$

Because $t_2 - t_1$ must be < 94 for QPOSCNT increment: $\frac{94}{t_2-t_1} < 1$ for CW rotation

And because $t_2 - t_1$ must be > -94 for QPOSCNT decrement: $\frac{94}{t_2-t_1} > -1$ for CCW rotation

$$speed_pr = \frac{94}{t_2 - t_1}$$

or

$$\frac{\frac{32\frac{150MHz}{128} * 60s/min}{4000 * 6000rpm)}}{t_2 - t_1} .....6$$

To convert speed_pr to RPM:

Multiply Equation 6 by 6000rpm:

$$SpeedRpm\_fr = 6000rpm * \frac{\frac{32\frac{150MHz}{128} * 60s/min}{4000 * 6000rpm)}}{t_2 - t_1}$$

$$= \frac{32\frac{150MHz}{128} * 60s/min}{4000 * (t_2 - t_1)}$$

or

$$\frac{\frac{32}{4000rev} * 60s/min}{(t_2 - t_1)(QCPRDLAT)} .....FinalEquation$$

For 100 MHz Operation: The same calculations as above are performed, but with 100 MHz instead of 150MHz when calculating SpeedRpm_pr.

The value for freqScaler_pr becomes: [32∗(100MHz/128)∗60s/min]/(4000∗6000rpm) = 63

More detailed calculation results can be found in the Example_freqcal.xls spreadsheet included in the example folder.

4.19 External Interrupt (external_interrupt)

This program sets up GPIO0 as XINT1 and GPIO1 as XINT2. Two other GPIO signals are used to trigger the interrupt (GPIO30 triggers XINT1 and GPIO31 triggers XINT2). XINT1 input is synched to SYSCLKOUT XINT2 has a long qualification - 6 samples at 510∗SYSCLKOUT each. GPIO34 will go high outside of the interrupts and low within the interrupts. This signal can be monitored on a scope. Each interrupt is fired in sequence - XINT1 first and then XINT2.

Monitor GPIO34 with an oscilloscope. GPIO34 will be high outside of the ISRs and low within each ISR.

External Connections

- Connect GPIO30 to GPIO0. GPIO0 is assigned to XINT1
- Connect GPIO31 to GPIO1. GPIO1 is assigned to XINT2

Watch Variables

- Xint1Count - XINT1 interrupt count
- Xint2Count - XINT2 interrupt count
- LoopCount - Idle loop count

4.20 ePWM Timer Interrupt From Flash (flash_f28335)

This example runs the ePWM interrupt example from flash. ePwm1 Interrupt will run from RAM and puts the flash into sleep mode. ePwm2 Interrupt will run from RAM and puts the flash into standby mode. ePWM3 Interrupt will run from FLASH. All timers have the same period. The timers are started sync'ed. An interrupt is taken on a zero event for each ePWM timer.GPIO32 is toggled while in the background loop.

Note:

- ePWM1: takes an interrupt every event
- ePWM2: takes an interrupt every 2nd event
- ePWM3: takes an interrupt every 3rd event

Thus the Interrupt count for ePWM1, ePWM4-ePWM6 should be equal The interrupt count for ePWM2 should be about half that of ePWM1 and the interrupt count for ePWM3 should be about 1/3 that of ePWM1

Follow these steps to run the program.

- Build the project
- Flash the .out file into the device.
- Set the hardware jumpers to boot to Flash
- Use the included GEL file to load the project, symbols defined within the project and the variables into the watch window.

Steps that were taken to convert the ePWM example from RAM to Flash execution:

- Change the linker cmd file to reflect the flash memory map.
- Make sure any initialized sections are mapped to Flash. In SDFlash utility this can be checked by the View->Coff/Hex status utility. Any section marked as "load" should be allocated to Flash.
- Make sure there is a branch instruction from the entry to Flash at 0x33FFF6 to the beginning of code execution. This example uses the DSP2833x_CodeStartBranch.__asm file to accomplish this.
- Set boot mode Jumpers to "boot to Flash"
- For best performance from the flash, modify the waitstates and enable the flash pipeline as shown in this example. Note: any code that manipulates the flash waitstate and pipeline control must be run from RAM. Thus these functions are located in their own memory section called ramfuncs.

Watch Variables

- EPwm1TimerIntCount
- EPwm2TimerIntCount
- EPwm3TimerIntCount

4.21 GPIO Setup (gpio_setup)

This example Configures the 2833x GPIO into two different configurations This code is verbose to illustrate how the GPIO could be setup.In a real application, lines of code can be combined for improved code size and efficiency.

This example only sets-up the GPIO.. nothing is actually done with the pins after setup.

In general:

- All pullup resistors are enabled. For ePWMs this may not be desired.
- Input qual for communication ports (eCAN, SPI, SCI, I2C) is asynchronous

- Input qual for Trip pins (TZ) is asynchronous
- Input qual for eCAP and eQEP signals is synch to SYSCLKOUT
- Input qual for some I/O's and interrupts may have a sampling window

4.22 GPIO Toggle (gpio_toggle)

Note:
ALL OF THE I/O'S TOGGLE IN THIS PROGRAM. MAKE SURE THIS WILL NOT DAMAGE YOUR HARDWARE BEFORE RUNNING THIS EXAMPLE.

Three different examples are included. Select the example (data, set/clear or toggle) to execute before compiling using the macros found at the top of the code.

Each example toggles all the GPIOs in a different way, the first through writing values to the GPIO DATA registers, the second through the SET/CLEAR registers and finally the last through the TOGGLE register

The pins can be observed using Oscilloscope.

4.23 High Resolution PWM (hrpwm)

This example modifies the MEP control registers to show edge displacement due to the HRPWM control extension of the respective EPwm module All EPwm1A,2A,3A,4A channels (GPIO0, GPIO2, GPIO4, GPIO6) will have fine edge movement due to HRPWM logic

1. 15MHz PWM (for 150 MHz SYSCLKOUT) or 10MHz PWM (for 100MHz SYSCLKOUT)
   ePWM1A toggle low/high with MEP control on rising edge
   ePWM1B toggle low/high with NO HRPWM control
2. 7.5MHz PWM (for 150 MHz SYSCLKOUT) or 5MHz PWM (for 100MHz SYSCLKOUT),
   ePWM2A toggle low/high with MEP control on rising edge
   ePWM2B toggle low/high with NO HRPWM control
3. 15MHz PWM (for 150 MHz SYSCLKOUT) or 10MHz PWM (for 100MHz SYSCLKOUT),
   ePWM3A toggle as high/low with MEP control on falling edge
   ePWM3B toggle low/high with NO HRPWM control
4. 7.5MHz PWM (for 150 MHz SYSCLKOUT) or 5MHz PWM (for 100MHz SYSCLKOUT),
   ePWM4A toggle as high/low with MEP control on falling edge
   ePWM4B toggle low/high with NO HRPWM control

External Connections

Monitor ePWM1-ePWM4 pins on an oscilloscope as described below:

- ePWM1A is on GPIO0
- ePWM1B is on GPIO1
- ePWM2A is on GPIO2
- ePWM2B is on GPIO3

- ePWM3A is on GPIO4
- ePWM3B is on GPIO5
- ePWM4A is on GPIO6
- ePWM4B is on GPIO7

4.24 High Resolution PWM SFO

This example modifies the MEP control registers to show edge displacement due to the HRPWM control extension of the respective ePWM module.

Note:
By default, this example project is configured for floating-point math. All included libraries must be pre-compiled for floating-point math. Therefore, SFO_TI_Build_fpu.lib (compiled for floating-point) is included in the project instead of the SFO_TI_Build.lib (compiled for fixed-point). To convert the example for fixed-point math, follow the instructions in sfo_readme.txt in the /doc directory of the header files and peripheral examples package.

This example calls the following TI's MEP Scale Factor Optimizer (SFO) software library functions:

- **void SFO_MepEn(int i)**; initialize MEP_Scalefactor[i] dynamically when HRPWM is in use.
- **void SFO_MepDis(int i)**; initialize MEP_Scalefactor[i] when HRPWM is not used

Where MEP_ScaleFactor[5] is a global array variable used by the SFO library

This example is intended to explain the HRPWM capabilities. The code can be optimized for code efficiency. Refer to TI's Digital power application examples and TI Digital Power Supply software libraries for details.

All ePWM1A,2A,3A,4A channels (GPIO0, GPIO2, GPIO4, GPIO6) will have fine edge movement due to the HRPWM logic

1. 5MHz PWM (SYSCLK=150MHz) or 3.33MHz PWM (SYSCLK=100MHz), ePWM1A toggle low/high with MEP control on falling edge
2. 5MHz PWM (SYSCLK=150MHz) or 3.33MHz PWM (SYSCLK=100MHz) ePWM2A toggle low/high with MEP control on falling edge
3. 5MHz PWM (SYSCLK=150MHz) or 3.33MHz PWM (SYSCLK=100MHz) ePWM3A toggle high/low with MEP control on falling edge
4. 5MHz PWM (SYSCLK=150MHz) or 3.33MHz PWM (SYSCLK=100MHz) ePWM4A toggle high/low with MEP control on falling edge

Running the Application

1. Run this example at 150MHz SYSCLKOUT (or 100 MHz SYSCLKOUT for 100 MHz devices)
2. Load the Example_2833xHRPWM_SFO.gel and observe variables in the watch window
3. Activate Real time mode
4. Run the code
5. Watch ePWM1A-4A waveforms on a Oscilloscope

6. In the watch window: Set the variable UpdateFine = 1 to observe the ePWMxA output with HRPWM capabilities (default). Observe the duty cycle of the waveform changes in fine MEP steps
7. In the watch window: Change the variable UpdateFine to 0, to observe the ePWMxA output without HRPWM capabilities. Observe the duty cycle of the waveform changes in coarse steps of 10nsec.

External Connections

Monitor ePWM1-ePWM4 pins on an oscilloscope as described below.

- EPWM1A is on GPIO0
- EPWM2A is on GPIO2
- EPWM3A is on GPIO4
- EPWM4A is on GPIO6

Watch Variables

- UpdateFine
- MEP_ScaleFactor
- EPwm1Regs.CMPA.all
- EPwm2Regs.CMPA.all
- EPwm3Regs.CMPA.all
- EPwm4Regs.CMPA.all

Note:
THE SFO.H FUNCTIONS INCLUDED WITH THIS EXAMPLE ONLY SUPPORTS EPWM1-EPWM4. FOR SUPPORT FOR MORE THAN 4 EPWMS, USE SFO_V5.H WITH THE SFO_TI_BUILD_V5.LIB LIBRARY. SEE THE HRPWM REFERENCE GUIDE (SPRU924) FOR USAGE INFORMATION AND DIFFERENCES BETWEEN VERSIONS.

4.25 High Resolution PWM SFO V5

This example modifies the MEP control registers to show edge displacement due to the HRPWM control extension of the respective ePWM module.

Note:
By default, this example project is configured for floating-point math. All included libraries must be pre-compiled for floating-point math. Therefore, SFO_TI_Build_V5B_fpu.lib (compiled for floating-point) is included in the project instead of the SFO_TI_Build_V5B.lib (compiled for fixed-point). To convert the example for fixed-point math, follow the instructions in sfo_readme.txt in the /doc directory of the header files and peripheral examples package.

This program requires the DSP2833x header files, which include the following files required for this example: SFO_V5.h and SFO_TI_Build_V5B_fpu.lib (or SFO_TI_Build_V5B.lib for fixed point)

This example calls the following TI's MEP Scale Factor Optimizer (SFO) software library V5 functions:

- **int SFO_MepEn_V5(int i)**; updates MEP_ScaleFactor[i] dynamically when HRPWM is in use.
- **Returns**
  - 1 when complete for the specified channel
  - 0 if not complete for the specified channel
  - 2 if there is a scale factor out-of-range error (MEP_ScaleFactor[n] differs from seed MEP_ScaleFactor[0] by more than +/-15)
- **int SFO_MepDis_V5(int i)**; updates MEP_ScaleFactor[i] when HRPWM is not used
- **Returns**
  - 1 when complete for the specified channel
  - 0 if not complete for the specified channel

This example is intended to explain the HRPWM capabilities. The code can be optimized for code efficiency. Refer to TI's Digital power application examples and TI Digital Power Supply software libraries for details. All ePWM1A-6A channels will have fine edge movement due to the HRPWM logic

- 5MHz PWM (for 150 MHz SYSCLKOUT), ePWMxA toggle high/low with MEP control on rising edge
- 3.33MHz PWM (for 100 MHz SYSCLKOUT), ePWMxA toggle high/low with MEP control on rising edge

Running the Application

1. ∗∗**!!IMPORTANT!!**∗∗ - in SFO_V5.h, set PWM_CH to the max number of HRPWM channels plus one. For example, for the F28335, the maximum number of HRPWM channels is 6. 6+1=7, so set #define PWM_CH 7 in SFO_V5.h. (Default is 7)
2. Run this example at 150/100MHz SYSCLKOUT
3. Load the Example_2833xHRPWM_SFO.gel and observe variables in the watch window
4. Activate Real time mode
5. Run the code
6. Watch ePWM1-6 waveforms on a Oscilloscope
7. In the watch window: Set the variable UpdateFine = 1 to observe the ePWMxA output with HRPWM capabilities (default). Observe the duty cycle of the waveform changes in fine MEP steps
8. In the watch window: Change the variable UpdateFine to 0, to observe the ePWMxA output without HRPWM capabilities. Observe the duty cycle of the waveform changes in coarse steps of 10nsec.

External Connections

Monitor the following pins on an oscilloscope:

- ePWM1A is on GPIO0
- ePWM2A is on GPIO2
- ePWM3A is on GPIO4
- ePWM4A is on GPIO6
- ePWM5A is on GPIO8

- ePWM6A is onGPIO10

Watch Variables

- UpdateFine
- MEP_ScaleFactor
- EPwm1Regs.CMPA.all
- EPwm2Regs.CMPA.all
- EPwm3Regs.CMPA.all
- EPwm4Regs.CMPA.all
- EPwm5Regs.CMPA.all
- EPwm6Regs.CMPA.all

4.26 High Resolution PWM with slider(hrpwm_slider)

This example modifies the MEP control registers to show edge displacement due to HRPWM control blocks of the respective ePWM module, ePWM1A, 2A, 3A, and 4A channels (GPIO0, GPIO2, GPIO4, and GPIO6) will have fine edge movement due to HRPWM logic.

Running the Application

1. Launch the target configuration and connect to the target first
2. Load the program and set it up to run in real time mode. Do not run yet!
3. Load the Example_2833xHRPWM_slider.gel file (provided in the folder)
4. Select the HRPWM FineDutySlider from the GEL menu. A FineDuty slider graphics will show up in CCS.
   (a) Add "DutyFine" variable to the watch window
5. Load the program and run. Use the Slider to and observe the epwm edge displacement for each slider step change.

This explains the MEP control on the ePWMxA channels,

1. 15MHz PWM (for 150 MHz SYSCLKOUT) or 10MHz PWM (for 100MHz SYSCLKOUT),
   ePWM1A toggle low/high with MEP control on rising edge
   ePWM1B toggle low/high with NO HRPWM control
2. 7.5MHz PWM (for 150 MHz SYSCLKOUT) or 5MHz PWM (for 100MHz SYSCLKOUT),
   ePWM2A toggle low/high with MEP control on rising edge
   ePWM2B toggle low/high with NO HRPWM control
3. 15MHz PWM (for 150 MHz SYSCLKOUT) or 10MHz PWM (for 100MHz SYSCLKOUT),
   ePWM3A toggle as high/low with MEP control on falling edge
   ePWM3B toggle low/high with NO HRPWM control
4. 7.5MHz PWM (for 150 MHz SYSCLKOUT) or 5MHz PWM (for 100MHz SYSCLKOUT),
   ePWM4A toggle as high/low with MEP control on falling edge
   ePWM4B toggle low/high with NO HRPWM control

External Connections

- EPWM1A is on GPIO0
- EPWM1B is on GPIO1
- EPWM2A is on GPIO2
- EPWM2B is on GPIO3
- EPWM3A is on GPIO4
- EPWM3B is on GPIO5

Watch Variables

- DutyFine

4.27 I2C EEPROM (i2c_eeprom)

This program requires an external I2C EEPROM connected to the I2C bus at address 0x50.

This program will write 1-14 words to EEPROM and read them back. The data written and the EEPROM address written to are contained in the message structure, **I2cMsgOut1**. The data read back will be contained in the message structure **I2cMsgIn1**.

Watch Variables

- I2cMsgIn1
- I2cMsgOut1

4.28 Low Power Modes: Halt Mode and Wakeup (lpm_haltwake)

This example puts the device into HALT mode. If the lowest possible current consumption in HALT mode is desired, the JTAG connector must be removed from the device board while the device is in HALT mode.

The example then wakes up the device from HALT using GPIO0. GPIO0 wakes the device from HALT mode when a high-to-low signal is detected on the pin. This pin must be pulsed by an external agent for wakeup.

The wakeup process begins as soon as GPIO0 is held low for the time indicated in the device datasheet. After the device wakes up, GPIO1 can be observed to go high.

GPIO0 is configured as the LPM wakeup pin to trigger a WAKEINT interrupt upon detection of a low pulse. Initially, pull GPIO0 high externally. To wake device from halt mode, pull GPIO0 low for at least the crystal startup time + 2 OSCLKS, then pull it high again.

External Connections

- To observe when device wakes from HALT mode, monitor GPIO1 with an oscilloscope (set to 1 in WAKEINT ISR)

4.29 Low Power Modes: Device Idle Mode and Wakeup (lpm_idlewake)

This example puts the device into IDLE mode then wakes up the device from IDLE using XINT1 which triggers on a falling edge from GPIO0.

This pin must be pulled from high to low by an external agent for wakeup. GPIO0 is configured as an XINT1 pin to trigger an XINT1 interrupt upon detection of a falling edge.

Initially, pull GPIO0 high externally. To wake device from idle mode by triggering an XINT1 interrupt, pull GPIO0 low (falling edge)

External Connections

- To observe the device wakeup from IDLE mode, monitor GPIO1 with an oscilloscope, which goes high in the XINT_1_ISR.

4.30 Low Power Modes: Device Standby Mode and Wakeup (lpm_standbywake)

This example puts the device into STANDBY mode. If the lowest possible current consumption in STANDBY mode is desired, the JTAG connector must be removed from the device board while the device is in STANDBY mode.

GPIO0 is configured as the LPM wakeup pin to trigger a WAKEINT interrupt upon detection of a low pulse. Initially, pull GPIO0 high externally. To wake device from standby mode, pull GPIO0 low for at least (2+QUALSTDBY) OSCLKS, then pull it high again.

The example then wakes up the device from STANDBY using GPIO0. GPIO0 wakes the device from STANDBY mode when a low pulse (signal goes high->low->high)is detected on the pin. This pin must be pulsed by an external agent for wakeup.

As soon as GPIO0 goes high again after the pulse, the device should wake up, and GPIO1 can be observed to toggle.

External Connections

- To observe when device wakes from STANDBY mode, monitor GPIO1 with an oscilloscope (set to 1 in WAKEINT ISR)

4.31 McBSP Digital Loop Back (mcbsp_loopback)

This example performs digital loopback tests for the McBSP peripheral. This example does not use interrupts. Instead, a polling method is used to check the receive data. The incoming data is checked for accuracy. If an error is found the error() function is called and execution stops.

Watch Variables

- sdata1

- sdata2
- rdata1
- rdata2
- rdata1_point
- rdata2_point

4.32 McBSP Digital Loop Back with DMA (mcbsp_loopback_dma)

This program is a McBSP example that uses the internal loopback of the peripheral and utilizes the DMA to transfer data from one buffer to the McBSP, and then from the McBSP to another buffer.

Initially, sdata[] is filled with values from 0x0000- 0x007F. The DMA moves the values in sdata[] one by one to the DXRx registers of the McBSP. These values are transmitted and subsequently received by the McBSP. Then, the DMA moves each data value to rdata[] as it is received by the McBSP.

By default for the McBSP examples, the McBSP sample rate generator (SRG) input clock frequency is LSPCLK (150E6/4 or 100E6/4) assuming SYSCLKOUT = 150 MHz or 100 MHz respectively. If while testing, the SRG input frequency is changed, the define MCBSP_SRG_FREQ (150E6/4 or 100E6/4) in the Mcbsp.c file must also be updated accordingly. This define is used to determine the Mcbsp initialization delay after the SRG is enabled, which must be at least 2 SRG clock cycles.

Watch Variables

- sdata
- rdata

4.33 McBSP Digital Loop Back with Interrupts (mcbsp_loopback_interrupts)

This program uses the internal loopback of the peripheral. Both Rx and Tx interrupts are enabled.

Incrementing values from 0x0000 to 0x00FF are being sent and received.

This pattern is repeated forever.

By default for the McBSP examples, the McBSP sample rate generator (SRG) input clock frequency is LSPCLK (150E6/4 or 100E6/4) assuming SYSCLKOUT = 150 MHz or 100 MHz respectively. If while testing, the SRG input frequency is changed, the define MCBSP_SRG_FREQ (150E6/4 or 100E6/4) in the Mcbsp.c file must also be updated accordingly. This define is used to determine the Mcbsp initialization delay after the SRG is enabled, which must be at least 2 SRG clock cycles.

Watch Variables

- sdata
- rdata
- rdata_point

4.34 McBSP Digital Loop Back using SPI Mode (mcbsp_spi_loopback)

This program will execute and transmit words until terminated by the user.

By default for the McBSP examples, the McBSP sample rate generator (SRG) input clock frequency is LSPCLK (150E6/4 or 100E6/4) assuming SYSCLKOUT = 150 MHz or 100 MHz respectively. If while testing, the SRG input frequency is changed, the define MCBSP_SRG_FREQ (CPU_SPD/4) in the Mcbsp.c file must also be updated accordingly. This define is used to determine the Mcbsp initialization delay after the SRG is enabled, which must be at least 2 SRG clock cycles.

Watch Variables

- sdata1
- sdata2
- rdata1
- rdata2

4.35 SCI Autobaud (sci_autobaud)

This test will perform autobaud lock at a variety of baud rates, including very high baud rates.

For this test to properly run, connect the SCI-A pins to the SCI-B pins without going through a transceiver. At higher baud rates, the slew rate of the incoming data bits can be affected by transceiver and connector performance. This slew rate may limit reliable autobaud detection at higher baud rates.

SCIA: Slave, autobaud locks, receives characters and echos them back to the host. Uses the RX interrupt to receive characters.

SCIB: Host, known baud rate, sends characters to the slave and checks that they are echoed back.

External Connections

- SCITXDA is on GPIO29
- SCIRXDB is on GPIO19
- SCIRXDA is on GPIO28
- SCITXDB is on GPIO18
- Connect GPIO29 to GPIO19
- Connect GPIO28 to GPIO18

Watch Variables

- BRRVal - current BRR value used for SCIB
- ReceivedAChar - character received by SCIA
- ReceivedBChar - character received by SCIB
- SendChar - character being sent by SCIB
- SciaRegs.SCILBAUD - SCIA baud register set by autobaud lock
- SciaRegs.SCIHBAUD - SCIA baud register set by autobaud lock

4.36 SCI Echo Back (sci_echoback)

This test receives and echo-backs data through the SCI-A port.

The PC application 'hyperterminal' can be used to view the data from the SCI and to send information to the SCI. Characters received by the SCI port are sent back to the host.

Running the Application

1. Configure hyperterminal: Use the included hyperterminal configuration file SCI_96.ht. To load this configuration in hyperterminal
   (a) Open hyperterminal
   (b) Go to file->open
   (c) Browse to the location of the project and select the SCI_96.ht file.
2. Check the COM port. The configuration file is currently setup for COM1. If this is not correct, disconnect (Call->Disconnect) Open the File-Properties dialog and select the correct COM port.
3. Connect hyperterminal Call->Call and then start the 2833x SCI echoback program execution.
4. The program will print out a greeting and then ask you to enter a character which it will echo back to hyperterminal.

As is, the program configures SCI-A for 9600 baud with SYSCLKOUT = 150MHz and LSPCLK = 37.5 MHz or SYSCLKOUT = 100MHz and LSPCLK = 25.0 Mhz

Watch Variables

- LoopCount - Number of characters sent
- ErrorCount

External Connections

Connect the SCI-A port to a PC via a transceiver and cable.

- GPIO28 is SCI_A-RXD (Connect to Pin3, PC-TX, of serial DB9 cable)
- GPIO29 is SCI_A-TXD (Connect to Pin2, PC-RX, of serial DB9 cable)

4.37 SCI Digital Loop Back (scia_loopback)

This program uses the internal loop back test mode of the peripheral. Other then boot mode pin configuration, no other hardware configuration is required.

This test uses the loopback test mode of the SCI module to send characters starting with 0x00 through 0xFF. The test will send a character and then check the receive buffer for a correct match.

Watch Variables

- LoopCount - Number of characters sent
- ErrorCount - Number of errors detected
- SendChar - Character sent
- ReceivedChar - Character received

4.38 SCI Digital Loop Back with Interrupts (spi_loopback_interrupts)

This program uses the internal loop back test mode of the peripheral. Other then boot mode pin configuration, no other hardware configuration is required. Both interrupts and the SCI FIFOs are used.

A stream of data is sent and then compared to the received stream.

The SCI-A sent data looks like this:

00 01 02 03 04 05 06 07

01 02 03 04 05 06 07 08

02 03 04 05 06 07 08 09

....

FE FF 00 01 02 03 04 05

FF 00 01 02 03 04 05 06

etc..

The SCI-B sent data looks like this:

FF FE FD FC FB FA F9 F8

FE FD FC FB FA F9 F8 F7

FD FC FB FA F9 F8 F7 F6

....

01 00 FF FE FD FC FB FA

00 FF FE FD FC FB FA F9

etc..

Both patterns are repeated forever.

Watch Variables

- sdataA, sdataB - Data to send
- rdataA, rdataB - Received data
- rdata_pointA, rdata_pointB - Used to keep track of the last position in the receive stream for error checking

4.39 SPI Digital Loop Back (spi_loopback)

This program uses the internal loop back test mode of the peripheral. Other then boot mode pin configuration, no other hardware configuration is required. Interrupts are not used.

A stream of data is sent and then compared to the received stream. The sent data looks like this:

0000 0001 0002 0003 0004 0005 0006 0007 .... FFFE FFFF

This pattern is repeated forever.

Watch Variables

- sdata - Sent data
- rdata - Received data

4.40 SPI Digital Loop Back with Interrupts (spi_loopback_interrupts)

This program uses the internal loop back test mode of the peripheral. Other then boot mode pin configuration, no other hardware configuration is required. Both interrupts and the SPI FIFOs are used.

A stream of data is sent and then compared to the received stream. The sent data looks like this:

0000 0001 0002 0003 0004 0005 0006 0007

0001 0002 0003 0004 0005 0006 0007 0008

0002 0003 0004 0005 0006 0007 0008 0009

....

FFFE FFFF 0000 0001 0002 0003 0004 0005

FFFF 0000 0001 0002 0003 0004 0005 0006

etc..

This pattern is repeated forever.

Watch Variables

- sdata - Data to send
- rdata - Received data
- rdata_point - Used to keep track of the last position in the receive stream for error checking

4.41 Software Prioritized Interrupts (sw_prioritized_interrupts)

For most applications, the hardware prioritizing of the the PIE module is sufficient. For applications that need custom prioritizing, this example illustrates how this can be done through software.

This program simulates interrupt conflicts by writing to the PIEIFR registers. This will cause multiple interrupt requests to come into the PIE block at the same time.

The interrupt service routines are software prioritized as per the table found in the DSP2833x_SWPrioritizedIsrLevels.h file.

Running the Application

1. Before compiling you must set the Global and Group interrupt priorities in the DSP2803x_SWPrioritizedIsrLevels.h file.

2. Select which test case you'd like to run with the #define CASE directive (1-9, default 1).
3. Compile the code, load, and run
4. At the end of each test there is a hard coded breakpoint (ESTOP0). When code stops at the breakpoint, examine the ISRTrace buffer to see the order in which the ISR's completed. All PIE interrupts will be added to the ISRTrace. The ISRTrace will consist of a list of hex values as shown:

   0x00wx <- PIE Group w interrupt x finished first

   0x00yz <- PIE Group y interrupt z finished next
5. If desired, set a new set of Global and Group interrupt priorities and repeat the test to see the change.

Watch Variables

- ISRTrace - Trace of ISR's in the order they complete.

After each test, examine this buffer to determine if the ISR's completed in the order desired.

4.42 Timer based blinking LED (timed_led_blink)

This example configures CPU Timer0 for a 500 msec period, and toggles the GPIO32 LED on the 2833x eZdsp once per interrupt. For testing purposes, this example also increments a counter each time the timer asserts an interrupt.

Watch Variables

- CpuTimer0.InterruptCount

External Connections

- Monitor the GPIO32 LED blink on (for 500 msec) and off (for 500 msec) on the 2833x eZdsp.

4.43 Watchdog interrupt Test (watchdog)

This program exercises the watchdog.

First the watchdog is connected to the WAKEINT interrupt of the PIE block. The code is then put into an infinite loop.

The user can select to feed the watchdog key register or not by commenting the following line of code in the infinite loop: **ServiceDog()**;

If the watchdog key register is fed by the ServiceDog function then the WAKEINT interrupt is not taken. If the key register is not fed by the ServiceDog function then WAKEINT will be taken.

Watch Variables

- **LoopCount** - Number of times through the infinite loop
- **WakeCount** - Number of times through WAKEINT

4.44 Code Run from XINTF (xintf_run_from)

This example configures CPU Timer0 and increments a counter each time the timer asserts an interrupt.

The code is loaded into SARAM. The XINTF Zone 7 is configured for x16-bit data bus. A portion of the code is copied to XINTF for execution there.

Watch Variables

- CpuTimer0.InterruptCount
- CpuTimer1.InterruptCount
- CpuTimer2.InterruptCount

5 Delfino F2833x Example Applications

These example applications show the user how to make use of various peripherals present on the F2833x device. They are intended for demonstration purposes only and a good starting point for building new applications.

All of these examples reside in the `device_support/f2833x/<version>/DSP2833x_examples_ccsv5` subdirectory of the ControlSUITE package.

5.1 ADC to DMA (adc_dma)

This ADC example uses ADC to convert 4 channels for each SOC received, with total of 10 SOCs. Each SOC initiates 4 conversions. DMA is used to capture the data on each SEQ1_INT. DMA will re-sort the data by channel sequentially, i.e. all channel0 data will be together and all channel1 data will be together.

Watch Variables

- DMABuf1 - DMA Buffer

5.2 ADC Seq Override Mode Test (add_seq_ovd_test)

In this example, channel A0 is converted forever and logged in a buffer (SampleTable) using sequencer1 in sequence override mode. Sequencer is Sequential mode with sample rate of 1/(3∗40ns) = 8.3 MHZ.

Watch Variables

- SampleTable - Log of converted values.
- GPIO34 - Toggles on every ADC sequencer flag

5.3 ADC Seq Mode Test (adc_seqmode_test)

In this example, channel A0 is converted forever and logged in a buffer (SampleTable)

Watch Variables

- SampleTable - Log of converted values

5.4 ADC Start of Conversion (adc_soc)

This ADC example uses ePWM1 to generate a periodic ADC SOC on SEQ1. Two channels are converted, ADCINA3 and ADCINA2.

Watch Variables

- Voltage1[10] - Last 10 ADCRESULT0 values
- Voltage2[10] - Last 10 ADCRESULT1 values
- ConversionCount - Current result number 0-9
- LoopCount - Idle loop counter

5.5 Cpu Timer (cpu_timer)

This example configures CPU Timer0, 1, and 2 and increments a counter each time the timers asserts an interrupt.

Watch Variables

- CputTimer0.InterruptCount
- CpuTimer1.InterruptCount
- CpuTimer2.InterruptCount

5.6 DMA Ram to Ram (dma_ram_to_ram)

This example will perform a block copy from L5 SARAM to L4 SARAM of 1024 words. Transfer will be started by Timer0. Will use 32-bit data size to decrease the transfer time.

Watch Variables

- DMABuf1
- DMABuf2

5.7 DMA XINTF to RAM (dma_xintf_to_ram)

This example will perform a block copy of 1024 words from Zone 7 XINTF (DMABuf2) to L4 SARAM (DMABuf1) . Transfer will be started by Timer0. We will use 32-bit DMA datasize. Note this is independent from the XINTF data width which is x16.

Watch Variables

- DMABuf1
- DMABuf2

5.8 eCAN-A to eCAN-B Trasmit Loop (ecan_a_to_b_xmit)

This example TRANSMITS data to another CAN module using MAILBOX5 This program could either loop forever or transmit "n" # of times, where "n" is the TXCOUNT value.

This example can be used to check CAN-A and CAN-B. Since CAN-B is initialized in DSP2833x_ECan.c, it will acknowledge all frames transmitted by the node on which this code runs. Both CAN ports of the 2833x DSP need to be connected to each other (via CAN transceivers)

External Connections

- eCANA is on GPIO31 (CANTXA) and GPIO30 (CANRXA)
- eCANB is on GPIO8 (CANTXB) and GPIO10 (CANRXB)
- Connect eCANA to eCANB via CAN transceivers

5.9 eCAN back to back (ecan_back2back)

This example tests eCAN by transmitting data back-to-back at high speed without stopping. The received data is verified. Any error is flagged. MBX0 transmits to MBX16, MBX1 transmits to MBX17 and so on....

This program illustrates the use of self-test mode

Watch Variables

- PassCount
- ErrorCount
- MessageReceivedCount

5.10 eCAP APWM (ecap_epwm)

This program sets up eCAP pins in the APWM mode. This program runs at 150 MHz SYSCLKOUT assuming a 30 MHz XCLKIN or 100 MHz SYSCLKOUT assuming a 20 MHz XCLKIN.

For 150 MHz devices:

- eCAP1 will come out on the GPIO24 pin. This pin is configured to vary between 7.5 Hz and 15 Hz using the shadow registers to load the next period/compare values.
- eCAP2 will come out on the GPIO7 pin. This pin is configured as a 7.5 Hz output.
- eCAP3 will come out on the GPIO9 pin. This pin is configured as a 1.5 Hz output.
- eCAP4 will come out on the GPIO11 pin. This pin is configured as a 30 kHz output.
- All frequencies assume a 30 Mhz input clock. The XCLKOUT pin should show 150Mhz.

For 100 MHz devices:

- eCAP1 will come out on the GPIO24 pin. This pin is configured to vary between 5 Hz and 10 Hz using the shadow registers to load the next period/compare values.
- eCAP2 will come out on the GPIO7 pin. This pin is configured as a 5 Hz output.
- eCAP3 will come out on the GPIO9 pin. This pin is configured as a 1 Hz output.
- eCAP4 will come out on the GPIO11 pin. This pin is configured as a 20kHz output.
- All frequencies assume a 20 Mhz input clock. The XCLKOUT pin should show 100Mhz.

5.11 eCap capture PWM (ecap_capture_pwm)

This example configures ePWM3A for:

- Up count
- Period starts at 2 and goes up to 1000
- Toggle output on PRD

eCAP1 is configured to capture the time between rising and falling edge of the ePWM3A output.

External Connections

- eCap1 is on GPIO24
- ePWM3A is on GPIO4
- Connect GPIO4 to GPIO24.

Watch Variables

- ECap1IntCount - Successful captures
- ECap1PassCount - Interrupt counts

5.12 ePWM Deadband Generation (epwm_deadband)

This example configures ePWM1, ePWM2 and ePWM3 for:

- Count up/down
- Deadband 3 Examples are included:
- ePWM1: Active low PWMs
- ePWM2: Active low complementary PWMs
- ePWM3: Active high complementary PWMs

Each ePWM is configured to interrupt on the 3rd zero event when this happens the deadband is modified such that 0 <= DB <= DB_MAX. That is, the deadband will move up and down between 0 and the maximum value.

External Connections

- EPWM1A is on GPIO0
- EPWM1B is on GPIO1
- EPWM2A is on GPIO2
- EPWM2B is on GPIO3
- EPWM3A is on GPIO4
- EPWM3B is on GPIO5

5.13 ePWM to DMA (epwm_dma)

This example demonstrates several cases where the DMA is triggered from SOC signals generated by ePWM modules.

Watch Variables

- EPwm1Regs.TBPRD
- EPwm1Regs.CMPA.all
- ADCbuffer
- InterruptCount

5.14 ePWM Timer Interrupt (epwm_timer_interrupts)

This example configures the ePWM Timers and increments a counter each time an interrupt is taken.

In this example:

- All ePWM's are initialized.
- All timers have the same period.
- The timers are started sync'ed.
- An interrupt is taken on a zero event for each ePWM timer.
- ePWM1: takes an interrupt every event.
- ePWM2: takes an interrupt every 2nd event.
- ePWM3: takes an interrupt every 3rd event.
- ePWM4-ePWM6: takes an interrupt every event.

Thus the Interrupt count for ePWM1, ePWM4, ePWM5, and ePWM6 should be equal.The interrupt count for ePWM2 should be about half that of ePWM1 and the interrupt count for ePWM3 should be about 1/3 that of ePWM1.

Watch Variables

- EPwm1TimerIntCount
- EPwm2TimerIntCount
- EPwm3TimerIntCount
- EPwm4TimerIntCount
- EPwm5TimerIntCount
- EPwm6TimerIntCount

5.15 ePWM Trip Zone (epwm_trip_zone)

This example configures ePWM1 and ePWM2 as follows

- ePWM1 has TZ1 and TZ2 as one shot trip sources
- ePWM2 has TZ1 and TZ2 as cycle by cycle trip sources

Initially tie TZ1 and TZ2 high. During the test, monitor ePWM1 or ePWM2 outputs on a scope. Pull TZ1 or TZ2 low to see the effect.

External Connections

- EPWM1A is on GPIO0
- EPWM1B is on GPIO1
- EPWM2A is on GPIO2
- EPWM2B is on GPIO3
- TZ1 is on GPIO12
- TZ2 is on GPIO13

5.16 ePWM Action Qualifier Module using Upcount mode (epwm_up_aq)

This example configures ePWM1, ePWM2, ePWM3 to produce a waveform with independent modulation on EPWMxA and EPWMxB. The compare values CMPA and CMPB are modified within the ePWM's ISR. The TB counter is in upmode.

Monitor the ePWM1 - ePWM3 pins on an oscilloscope.

External Connections

- EPWM1A is on GPIO0
- EPWM1B is on GPIO1
- EPWM2A is on GPIO2
- EPWM2B is on GPIO3
- EPWM3A is on GPIO4
- EPWM3B is on GPIO5

5.17 ePWM Action Qualifier Module using up/down count (epwm_updown_aq)

This example configures ePWM1, ePWM2, ePWM3 to produce an waveform with independent modulation on EPWMxA and EPWMxB. The compare values CMPA and CMPB are modified within the ePWM's ISR. The TB counter is in up/down count mode for this example.

Monitor ePWM1-ePWM3 pins on an oscilloscope as described

External Connections

- EPWM1A is on GPIO0

- EPWM1B is on GPIO1
- EPWM2A is on GPIO2
- EPWM2B is on GPIO3
- EPWM3A is on GPIO4
- EPWM3B is on GPIO5

5.18 eQEP, Frequency measurement (eqep_freqcal)

This test will calculate the frequency and period of an input signal using eQEP module.

EPWM1A is configured to generate a frequency of 5 kHz.

See also:
section on Frequency Calculation for more details on the frequency calculation performed in this example.

In addition to the main example file, the following files must be included in this project:

- **Example_freqcal.c** , includes all eQEP functions
- **Example_EPwmSetup.c** , sets up EPWM1A for use with this example
- **Example_freqcal.h** , includes initialization values for frequency structure.

The configuration for this example is as follows

- Maximum frequency is configured to 10KHz (BaseFreq)
- Minimum frequency is assumed at 50Hz for capture pre-scalar selection

SPEED_FR: High Frequency Measurement is obtained by counting the external input pulses for 10ms (unit timer set to 100Hz).

$$SPEED\_FR = \frac{Count\ Delta}{10ms}$$

SPEED_PR: Low Frequency Measurement is obtained by measuring time period of input edges. Time measurement is averaged over 64edges for better results and capture unit performs the time measurement using pre-scaled SYSCLK

Note that pre-scaler for capture unit clock is selected such that capture timer does not overflow at the required minimum frequency This example runs forever until the user stops it.

External Connections

- Connect GPIO20/EQEP1A to GPIO0/EPWM1A

Watch Variables

- **freq.freqhz_fr** , Frequency measurement using position counter/unit time out
- **freq.freqhz_pr** , Frequency measurement using capture unit

5.18.1 EPWM Setup (Example_EPwmSetup.c)

This file contains source for the ePWM initialization for the freq calculation module. EPWM1 is set to operate in up-down count mode at a frequency of 5KHz

5.18.2 Frequency Calculation (Example_freqal.c)

This file includes the EQEP initialization and frequency calculation functions called by **Example_2833xEqep_freqcal.c**. The frequency calculation steps performed by FREQCAL_Calc()at SYSCLKOUT = 150 MHz and 100 MHZ are described below:

1. This program calculates: $**$**freqhz_fr**$**$ for SYSCLKOUT = 150MHz

$$freqhz\_fr\ or\ v = \frac{x_2 - x_1}{T} .....1$$

If

$$\frac{max}{base}\ freq = 10kHz => 10kHz = \frac{x_2 - x_1}{(2/100Hz)} .....2$$

$$max(x_2 - x_1) = 200counts = freqScaler\_fr$$

Note:
$T = \frac{2}{100Hz}$. 2 is from $\frac{x_2 - x_1}{2}$ because QPOSCNT counts 2 edges per cycle (rising and falling)

If both sides of Equation 2 are divided by 10 kHz, then:

$$1 = \frac{x_2 - x_1}{10kHz * (2/100Hz)}$$

where,

$$[10kHz * \frac{2}{100Hz}] = 200$$

Because

$$x_2 - x_1 < 200(max)$$

$$\frac{x_2 - x_1}{200} < 1$$

for all frequencies less than max

$$freq\_fr = \frac{x_2 - x_1}{200}\ or\ \frac{x_2 - x_1}{10kHz * (2/100Hz)} .....3$$

To get back to original velocity equation, Equation 1, multiply Equation 3 by 10 kHz

$$freqhz\_fr(or\ velocity) = 10kHz * \frac{x_2 - x_1}{10kHz * (2/100Hz)}$$

$$= \frac{x_2 - x_1}{(2/100Hz)} .....final\ equation$$

1. $**$**min freq**$**$ = $\frac{1\ count}{(2/100Hz)} = 50Hz$
2. $**$**freqhz_pr**$**$

$$freqhz\_pr\ or\ v = \frac{X}{t_2 - t_1} .....4$$

If

$$\frac{max}{base}\ freq = 10kHz => 10kHz = \frac{(4/2)}{T} = \frac{4}{2T}$$

where,

- 4 = QCAPCTL [UPPS] (Unit timeout - once every 4 edges)
- 2 = divide by 2 because QPOSCNT counts 2 edges per cycle (rising and falling)
- T = time in seconds = $\frac{t_2 - t_1}{(150MHz/128)}, t_2 - t_1$ = # of QCAPCLK cycles, and $1\ QCAPCLK\ cycle = \frac{1}{(150MHz/128)} = QCPRDLAT$

So:

$$10kHz = 4 * \frac{(150MHz/128)}{2 * (t_2 - t_1)}$$

$$t_2 - t_1 = 4 * \frac{(150MHz/128)}{10kHz * 2} = \frac{(150MHz/128)}{((2 * 10KHz)/4)} .....5$$

$$= 234\ QCAPCLK\ cycles = maximum(t_2 - t_1) = freqScaler\_pr$$

Divide both sides by $(t_2 - t_1)$, and:

$$1 = \frac{234}{t_2 - t_1} = \frac{(150MHz/128)/((2 * 10KHz)/4)}{t_2 - t_1}$$

Because $(t_2 - t_1) < 234(max)$, $\frac{234}{t_2 - t_1} < 1$ for all frequencies less than max

$$freq\_pr = \frac{234}{t_2 - t_1}\ or\ \frac{(150MHz/128)/((2 * 10KHz)/4)}{t_2 - t_1} .....6$$

Now within velocity limits, to get back to original velocity equation, Equation 1, multiply Equation 6 by 10 kHz:

$$freqhz\_fr(or\ velocity) = 10kHz * \frac{(150MHz/128)/((2 * 10KHz)/4)}{t_2 - t_1}$$

$$= \frac{(150MHz/128) * 4}{2 * (t_2 - t_1)}$$

or

$$\frac{4}{2 * (t_2 - t_1) * (QCPRDLAT)} .....final\ equation$$

For 100 MHz Operation:

The same calculations as above are performed, but with 100 MHz instead of 150MHz when calculating freqhz_pr, and at UPPS of 8 instead of 4. The value for freqScaler_pr becomes: (100MHz/128)/(2∗10kHz/8) = 313

More detailed calculation results can be found in the Example_freqcal.xls spreadsheet included in the example folder.

5.19 eQEP Speed and Position measurement (eqep_pos_speed)

This example provides position measurement,speed measurement using the capture unit, and speed measurement using unit time out. This example uses the IQMath library. It is used merely to simplify high-precision calculations.

The example requires the following hardware connections from EPWM1 and GPIO pins (simulating QEP sensor) to QEP peripheral.

- eQEP1A <- ePWM1A (simulates eQEP Phase A signal)
- eQEP1B <- ePWM1B (simulates eQEP Phase B signal)
- eQEP1I <- GPIO4 (simulates eQEP Index Signal)

See DESCRIPTION in Example_posspeed.c for more details on the calculations performed in this example. In addition to this file, the following files must be included in this project:

- Example_posspeed.c - includes all eQEP functions
- Example_EPwmSetup.c - sets up ePWM1A and ePWM1B as simulated QA and QB encoder signals
- Example_posspeed.h - includes initialization values for pos and speed structure

Note:

- Maximum speed is configured to 6000rpm(BaseRpm)
- Minimum speed is assumed at 10rpm for capture pre-scalar selection
- Pole pair is configured to 2 (pole_pairs)
- QEP Encoder resolution is configured to 4000counts/revolution (mech_scaler)
- which means: 4000/4 = 1000 line/revolution quadrature encoder (simulated by EPWM1)
- EPWM1 (simulating QEP encoder signals) is configured for 5kHz frequency or 300 rpm (=4∗5000 cnts/sec ∗ 60 sec/min)/4000 cnts/rev)
- SPEEDRPM_FR: High Speed Measurement is obtained by counting the QEP input pulses for 10ms (unit timer set to 100Hz).
- SPEEDRPM_FR = (Position Delta/10ms) ∗ 60 rpm
- SPEEDRPM_PR: Low Speed Measurement is obtained by measuring time period of QEP edges. Time measurement is averaged over 64edges for better results and capture unit performs the time measurement using pre-scaled SYSCLK
- pre-scaler for capture unit clock is selected such that capture timer does not overflow at the required minimum RPM speed.

External Connections

- Connect eQEP1A(GPIO20) to ePWM1A(GPIO0)(simulates eQEP Phase A signal)
- Connect eQEP1B(GPIO21) to ePWM1B(GPIO1)(simulates eQEP Phase B signal)
- Connect eQEP1I(GPIO23) to GPIO4 (simulates eQEP Index Signal)

Watch Variables

- qep_posspeed.SpeedRpm_fr - Speed meas. in rpm using QEP position counter
- qep_posspeed.SpeedRpm_pr - Speed meas. in rpm using capture unit
- qep_posspeed.theta_mech - Motor mechanical angle (Q15)
- qep_posspeed.theta_elec - Motor electrical angle (Q15)

5.19.1 EPWM Setup (Example_EPwmSetup.c)

This file contains source for the ePWM initialization for the freq calculation module. EPWM1 is set to operate in up-down count mode at a frequency of 5KHz

5.19.2 Position/Speed Calculation (Example_posspeed.c)

This file includes the EQEP initialization and position and speed calculation functions called by Example_2833xEqep_posspeed.c. The position and speed calculation steps performed by POSSPEED_Calc() at SYSCLKOUT = 150 MHz and 100 MHz are described below:

1. This program calculates: $**$**theta_mech**$**$ for SYSCLKOUT = 150Mhz

$$theta\_mech\ = \frac{QPOSCNT}{mech\_scaler} = \frac{QPOSCNT}{4000}$$

   where 4000 is the number of counts in 1 revolution.(4000/4 = 1000 line/rev. quadrature encoder)

2. This program calculates: $**$**theta_elec**$**$ for SYSCLKOUT = 150MHz

$$theta\_elec\ = (polepairs) * theta\_mech\ = \frac{2 * QPOSCNT}{4000}$$

3. This program calculates: $**$**SpeedRpm_fr**$**$ for SYSCLKOUT = 150Mhz

$$SpeedRpm\_fr\ = \frac{x_2 - x_1}{4000] * T} .....1$$

Note:

$x_2 - x_1$ is the difference in number of QPOSCNT counts. Dividing $x_2 - x_1$ by 4000 gives position relative to Index in one revolution.

If $baseRPM = 6000rpm$:

$$6000rpm = \frac{x_2 - x_1}{4000 * 10ms} .....2$$
$$= \frac{\frac{x_2 - x_1}{4000}}{\frac{.01s * 1min}{60sec}}$$
$$= \frac{\frac{x_2 - x_1}{4000}}{\frac{1}{6000}min}$$

max $x_2 - x_1 = 4000$ counts, or 1 revolution in 10 ms

If both sides of Equation 2 are divided by 6000 rpm, then:

$$1 = \frac{\frac{x_2 - x_1}{4000} rev.}{\frac{1}{6000} min * 6000rpm}$$

Because $x_2 - x_1$ must be $< 4000(max)$ for QPOSCNT increment, $\frac{x_2-x_1}{4000} < 1$ for CW rotation

And because $x_2 - x_1$ must be > -4000 for QPOSCNT decrement, $\frac{x_2-x_1}{4000} > -1$ for CCW rotation

$$speed\_fr = \frac{\frac{x_2-x_1}{4000}}{\frac{1}{6000}min * 6000rpm} = \frac{x_2 - x1}{4000} .....3$$

To convert speed_fr to RPM, multiply Equation 3 by 6000 rpm

$$SpeedRpm\_fr = 6000rpm * \frac{x_2 - x_1}{4000} .....final$$

1. $**$min rpm $**$ = selected at 10 rpm based on CCPS prescaler options available (128 is greatest)
2. $**$SpeedRpm_pr$**$

   $$SpeedRpm\_pr = \frac{X}{t_2 - t_1} .....4$$

   where X = QCAPCTL [UPPS]/4000 rev. (position relative to Index in 1 revolution)

If $\frac{max}{base} speed = 6000rpm : 6000 = \frac{\frac{32}{4000}}{\frac{t_2-t_1}{\frac{150MHz}{128}}}$ where,

- 32 = QCAPCTL [UPPS] (Unit timeout - once every 32 edges)
- $\frac{32}{4000}$ = position in 1 revolution (position as a fraction of 1 revolution)
- $\frac{t_2-t_1}{\frac{150MHz}{128}}, t_2 - t_1$ = # of QCAPCLK cycles
- 1 QCAPCLK cycle = $\frac{1}{\frac{150MHz}{128}}$ = QCPRDLAT

So:

$$6000rpm = \frac{32\frac{150MHz}{128} * 60s/min}{4000(t_2 - t_1)}$$

$$t_2 - t_1 = \frac{32\frac{150MHz}{128} * 60s/min}{4000 * 6000rpm} .....5$$

$$= 94CAPCLKcycles = maximum(t2 - t1) = SpeedScaler$$

Divide both sides by $t_2 - t_1$, and:

$$1 = \frac{94}{t_2 - t_1} = \frac{\frac{32\frac{150MHz}{128} * 60s/min}{4000 * 6000rpm}}{t_2 - t_1}$$

Because $t_2 - t_1$ must be < 94 for QPOSCNT increment: $\frac{94}{t_2-t_1} < 1$ for CW rotation

And because $t_2 - t_1$ must be > -94 for QPOSCNT decrement: $\frac{94}{t_2-t_1} > -1$ for CCW rotation

$$speed_pr = \frac{94}{t_2 - t_1}$$

or

$$\frac{\frac{32\frac{150MHz}{128} * 60s/min}{4000 * 6000rpm)}}{t_2 - t_1} .....6$$

To convert speed_pr to RPM:

Multiply Equation 6 by 6000rpm:

$$SpeedRpm\_fr = 6000rpm * \frac{32\frac{150MHz}{128} * 60s/min}{4000 * 6000rpm)}{t_2 - t_1}$$

$$= \frac{32\frac{150MHz}{128} * 60s/min}{4000 * (t_2 - t_1)}$$

or

$$\frac{\frac{32}{4000rev} * 60s/min}{(t_2 - t_1)(QCPRDLAT)} .....Final Equation$$

For 100 MHz Operation: The same calculations as above are performed, but with 100 MHz instead of 150MHz when calculating SpeedRpm_pr.

The value for freqScaler_pr becomes: [32∗(100MHz/128)∗60s/min]/(4000∗6000rpm) = 63

More detailed calculation results can be found in the Example_freqcal.xls spreadsheet included in the example folder.

5.20 External Interrupt (external_interrupt)

This program sets up GPIO0 as XINT1 and GPIO1 as XINT2. Two other GPIO signals are used to trigger the interrupt (GPIO30 triggers XINT1 and GPIO31 triggers XINT2). XINT1 input is synched to SYSCLKOUT XINT2 has a long qualification - 6 samples at 510∗SYSCLKOUT each. GPIO34 will go high outside of the interrupts and low within the interrupts. This signal can be monitored on a scope. Each interrupt is fired in sequence - XINT1 first and then XINT2.

Monitor GPIO34 with an oscilloscope. GPIO34 will be high outside of the ISRs and low within each ISR.

External Connections

- Connect GPIO30 to GPIO0. GPIO0 is assigned to XINT1
- Connect GPIO31 to GPIO1. GPIO1 is assigned to XINT2

Watch Variables

- Xint1Count - XINT1 interrupt count
- Xint2Count - XINT2 interrupt count
- LoopCount - Idle loop count

5.21 F28335 Flash Kernel (f28335_flash_kernel)

This example is for use with the SerialLoader2000 utility. This application is intended to be loaded into the device's RAM via the SCI boot mode. After successfully loaded this program implements a modified version of the SCI boot protocol that allows a user application to be programmed into flash

5.22 ePWM Timer Interrupt From Flash (flash_f28335)

This example runs the ePWM interrupt example from flash. ePwm1 Interrupt will run from RAM and puts the flash into sleep mode. ePwm2 Interrupt will run from RAM and puts the flash into standby mode. ePWM3 Interrupt will run from FLASH. All timers have the same period. The timers are started sync'ed. An interrupt is taken on a zero event for each ePWM timer.GPIO32 is toggled while in the background loop.

Note:

- ePWM1: takes an interrupt every event
- ePWM2: takes an interrupt every 2nd event
- ePWM3: takes an interrupt every 3rd event

Thus the Interrupt count for ePWM1, ePWM4-ePWM6 should be equal The interrupt count for ePWM2 should be about half that of ePWM1 and the interrupt count for ePWM3 should be about 1/3 that of ePWM1

Follow these steps to run the program.

- Build the project
- Flash the .out file into the device.
- Set the hardware jumpers to boot to Flash
- Use the included GEL file to load the project, symbols defined within the project and the variables into the watch window.

Steps that were taken to convert the ePWM example from RAM to Flash execution:

- Change the linker cmd file to reflect the flash memory map.
- Make sure any initialized sections are mapped to Flash. In SDFlash utility this can be checked by the View->Coff/Hex status utility. Any section marked as "load" should be allocated to Flash.
- Make sure there is a branch instruction from the entry to Flash at 0x33FFF6 to the beginning of code execution. This example uses the DSP2833x_CodeStartBranch.__asm file to accomplish this.
- Set boot mode Jumpers to "boot to Flash"
- For best performance from the flash, modify the waitstates and enable the flash pipeline as shown in this example. Note: any code that manipulates the flash waitstate and pipeline control must be run from RAM. Thus these functions are located in their own memory section called ramfuncs.

Watch Variables

- EPwm1TimerIntCount
- EPwm2TimerIntCount
- EPwm3TimerIntCount

5.23 Floating Point Unit (fpu_hardware)

The code calculates two y=mx+b equations. The variables are all 32-bit floating-point.

The compiler will generate floating point instructions to do these calculations. To compile the project for floating point, the following Build Options were used:

1. Project->Properties-> C/C++ Build window-> Basic Settings-> C2000 Compiler Vx.x
   In All Options textbox: add "–float_support=fpu32" .
   OR in Runtime Model Options, under "Specify floating point support (–float_support) pull-down menu: Select "fpu32".
2. Project->Properties-> C/C++ Build window-> Basic Settings-> C2000 Linker Vx.x-> File Search Path
   In "Include linker file or command file as input (–library, -l)" box, click green plus sign and add rts2800_fpu32.lib (run-time support library).
3. Not included in this example: If the project includes any other libraries, they must also be compiled with floating point instructions.

Watch Variables

- y1
- y2
- FPU registers (optional)

5.24 Floating Point Unit (fpu_oftware)

The code calculates two y=mx+b equations. The variables are all 32-bit floating-point.

The compiler will only used fixed point instructions. This means the runtime support library will be used to emulate floating point. This will also run on C28x devices without the floating point unit. To compile the project for fixed point, the following Build Options were used:

1. Project->Properties-> C/C++ Build window-> Basic Settings-> C2000 Compiler Vx.x
   In All Options textbox: "–float_support=fpu32" is removed.
   OR in Runtime Model Options, under "Specify floating point support (–float_support) pull-down menu: Select "None".
2. Project->Properties-> C/C++ Build window-> Basic Settings-> C2000 Linker Vx.x-> File Search Path
   In "Include linker file or command file as input (–library, -l)" box, click green plus sign and add rts2800.lib or rts2800_ml.lib (run-time support library).
3. Not included in this example: If the project includes any other libraries, they must also be compiled with fixed point instructions.

Watch Variables

- y1
- y2
- FPU registers (optional)

5.25 GPIO Setup (gpio_setup)

This example Configures the 2833x GPIO into two different configurations This code is verbose to illustrate how the GPIO could be setup.In a real application, lines of code can be combined for improved code size and efficiency.

This example only sets-up the GPIO and nothing is actually done with the pins after setup.

In general:

- All pullup resistors are enabled. For ePWMs this may not be desired.
- Input qual for communication ports (eCAN, SPI, SCI, I2C) is asynchronous
- Input qual for Trip pins (TZ) is asynchronous
- Input qual for eCAP and eQEP signals is synch to SYSCLKOUT
- Input qual for some I/O's and interrupts may have a sampling window

5.26 GPIO Toggle (gpio_toggle)

Note:
ALL OF THE I/O'S TOGGLE IN THIS PROGRAM. MAKE SURE THIS WILL NOT DAMAGE YOUR HARDWARE BEFORE RUNNING THIS EXAMPLE.

Three different examples are included. Select the example (data, set/clear or toggle) to execute before compiling using the macros found at the top of the code.

Each example toggles all the GPIOs in a different way, the first through writing values to the GPIO DATA registers, the second through the SET/CLEAR registers and finally the last through the TOGGLE register

The pins can be observed using Oscilloscope.

5.27 High Resolution PWM (hrpwm)

This example modifies the MEP control registers to show edge displacement due to the HRPWM control extension of the respective EPwm module All EPwm1A,2A,3A,4A channels (GPIO0, GPIO2, GPIO4, GPIO6) will have fine edge movement due to HRPWM logic

1. 15MHz PWM (for 150 MHz SYSCLKOUT) or 10MHz PWM (for 100MHz SYSCLKOUT)
   ePWM1A toggle low/high with MEP control on rising edge
   ePWM1B toggle low/high with NO HRPWM control
2. 7.5MHz PWM (for 150 MHz SYSCLKOUT) or 5MHz PWM (for 100MHz SYSCLKOUT),
   ePWM2A toggle low/high with MEP control on rising edge
   ePWM2B toggle low/high with NO HRPWM control
3. 15MHz PWM (for 150 MHz SYSCLKOUT) or 10MHz PWM (for 100MHz SYSCLKOUT),
   ePWM3A toggle as high/low with MEP control on falling edge
   ePWM3B toggle low/high with NO HRPWM control

4. 7.5MHz PWM (for 150 MHz SYSCLKOUT) or 5MHz PWM (for 100MHz SYSCLKOUT), ePWM4A toggle as high/low with MEP control on falling edge
   ePWM4B toggle low/high with NO HRPWM control

External Connections

Monitor ePWM1-ePWM4 pins on an oscilloscope as described below:

- ePWM1A is on GPIO0
- ePWM1B is on GPIO1
- ePWM2A is on GPIO2
- ePWM2B is on GPIO3
- ePWM3A is on GPIO4
- ePWM3B is on GPIO5
- ePWM4A is on GPIO6
- ePWM4B is on GPIO7

5.28 High Resolution PWM SFO

This example modifies the MEP control registers to show edge displacement due to the HRPWM control extension of the respective ePWM module.

Note:
By default, this example project is configured for floating-point math. All included libraries must be pre-compiled for floating-point math. Therefore, SFO_TI_Build_fpu.lib (compiled for floating-point) is included in the project instead of the SFO_TI_Build.lib (compiled for fixed-point). To convert the example for fixed-point math, follow the instructions in sfo_readme.txt in the /doc directory of the header files and peripheral examples package.

This example calls the following TI's MEP Scale Factor Optimizer (SFO) software library functions:

- **void SFO_MepEn(int i)**; initialize MEP_Scalefactor[i] dynamically when HRPWM is in use.
- **void SFO_MepDis(int i)**; initialize MEP_Scalefactor[i] when HRPWM is not used

Where MEP_ScaleFactor[5] is a global array variable used by the SFO library.

This example is intended to explain the HRPWM capabilities. The code can be optimized for code efficiency. Refer to TI's Digital power application examples and TI Digital Power Supply software libraries for details.

All ePWM1A,2A,3A,4A channels (GPIO0, GPIO2, GPIO4, GPIO6) will have fine edge movement due to the HRPWM logic

1. 5MHz PWM (SYSCLK=150MHz) or 3.33MHz PWM (SYSCLK=100MHz), ePWM1A toggle low/high with MEP control on falling edge
2. 5MHz PWM (SYSCLK=150MHz) or 3.33MHz PWM (SYSCLK=100MHz) ePWM2A toggle low/high with MEP control on falling edge
3. 5MHz PWM (SYSCLK=150MHz) or 3.33MHz PWM (SYSCLK=100MHz) ePWM3A toggle high/low with MEP control on falling edge

4. 5MHz PWM (SYSCLK=150MHz) or 3.33MHz PWM (SYSCLK=100MHz) ePWM4A toggle high/low with MEP control on falling edge

Running the Application

1. Run this example at 150MHz SYSCLKOUT (or 100 MHz SYSCLKOUT for 100 MHz devices)
2. Load the Example_2833xHRPWM_SFO.gel and observe variables in the watch window
3. Activate Real time mode
4. Run the code
5. Watch ePWM1A-4A waveforms on a Oscilloscope
6. In the watch window: Set the variable UpdateFine = 1 to observe the ePWMxA output with HRPWM capabilities (default). Observe the duty cycle of the waveform changes in fine MEP steps
7. In the watch window: Change the variable UpdateFine to 0, to observe the ePWMxA output without HRPWM capabilities. Observe the duty cycle of the waveform changes in coarse steps of 10nsec.

External Connections

Monitor ePWM1-ePWM4 pins on an oscilloscope as described below.

- EPWM1A is on GPIO0
- EPWM2A is on GPIO2
- EPWM3A is on GPIO4
- EPWM4A is on GPIO6

Watch Variables

- UpdateFine
- MEP_ScaleFactor
- EPwm1Regs.CMPA.all
- EPwm2Regs.CMPA.all
- EPwm3Regs.CMPA.all
- EPwm4Regs.CMPA.all

Note:
THE SFO.H FUNCTIONS INCLUDED WITH THIS EXAMPLE ONLY SUPPORTS EPWM1-EPWM4. FOR SUPPORT FOR MORE THAN 4 EPWMS, USE SFO_V5.H WITH THE SFO_TI_BUILD_V5.LIB LIBRARY. SEE THE HRPWM REFERENCE GUIDE (SPRU924) FOR USAGE INFORMATION AND DIFFERENCES BETWEEN VERSIONS.

5.29 High Resolution PWM SFO V5

This example modifies the MEP control registers to show edge displacement due to the HRPWM control extension of the respective ePWM module.

Note:

By default, this example project is configured for floating-point math. All included libraries must be pre-compiled for floating-point math. Therefore, SFO_TI_Build_V5B_fpu.lib (compiled for floating-point) is included in the project instead of the SFO_TI_Build_V5B.lib (compiled for fixed-point). To convert the example for fixed-point math, follow the instructions in sfo_readme.txt in the /doc directory of the header files and peripheral examples package.

This program requires the DSP2833x header files, which include the following files required for this example: SFO_V5.h and SFO_TI_Build_V5B_fpu.lib (or SFO_TI_Build_V5B.lib for fixed point)

This example calls the following TI's MEP Scale Factor Optimizer (SFO) software library V5 functions:

- **int SFO_MepEn_V5(int i)**; updates MEP_ScaleFactor[i] dynamically when HRPWM is in use.
- **Returns**
  - 1 when complete for the specified channel
  - 0 if not complete for the specified channel
  - 2 if there is a scale factor out-of-range error (MEP_ScaleFactor[n] differs from seed MEP_ScaleFactor[0] by more than +/-15)
- **int SFO_MepDis_V5(int i)**; updates MEP_ScaleFactor[i] when HRPWM is not used
- **Returns**
  - 1 when complete for the specified channel
  - 0 if not complete for the specified channel

This example is intended to explain the HRPWM capabilities. The code can be optimized for code efficiency. Refer to TI's Digital power application examples and TI Digital Power Supply software libraries for details. All ePWM1A-6A channels will have fine edge movement due to the HRPWM logic

- 5MHz PWM (for 150 MHz SYSCLKOUT), ePWMxA toggle high/low with MEP control on rising edge
- 3.33MHz PWM (for 100 MHz SYSCLKOUT), ePWMxA toggle high/low with MEP control on rising edge

Running the Application

1. $**$**!!IMPORTANT!!**$**$ - in SFO_V5.h, set PWM_CH to the max number of HRPWM channels plus one. For example, for the F28335, the maximum number of HRPWM channels is 6. 6+1=7, so set #define PWM_CH 7 in SFO_V5.h. (Default is 7)
2. Run this example at 150/100MHz SYSCLKOUT
3. Load the Example_2833xHRPWM_SFO.gel and observe variables in the watch window
4. Activate Real time mode
5. Run the code
6. Watch ePWM1-6 waveforms on a Oscilloscope
7. In the watch window: Set the variable UpdateFine = 1 to observe the ePWMxA output with HRPWM capabilities (default). Observe the duty cycle of the waveform changes in fine MEP steps

8. In the watch window: Change the variable UpdateFine to 0, to observe the ePWMxA output without HRPWM capabilities. Observe the duty cycle of the waveform changes in coarse steps of 10nsec.

External Connections

Monitor the following pins on an oscilloscope:

- ePWM1A is on GPIO0
- ePWM2A is on GPIO2
- ePWM3A is on GPIO4
- ePWM4A is on GPIO6
- ePWM5A is on GPIO8
- ePWM6A is onGPIO10

Watch Variables

- UpdateFine
- MEP_ScaleFactor
- EPwm1Regs.CMPA.all
- EPwm2Regs.CMPA.all
- EPwm3Regs.CMPA.all
- EPwm4Regs.CMPA.all
- EPwm5Regs.CMPA.all
- EPwm6Regs.CMPA.all

5.30 High Resolution PWM with slider(hrpwm_slider)

This example modifies the MEP control registers to show edge displacement due to HRPWM control blocks of the respective ePWM module, ePWM1A, 2A, 3A, and 4A channels (GPIO0, GPIO2, GPIO4, and GPIO6) will have fine edge movement due to HRPWM logic.

Running the Application

1. Launch the target configuration and connect to the target first
2. Load the program and set it up to run in real time mode. Do not run yet!
3. Load the Example_2833xHRPWM_slider.gel file (provded in the folder)
4. Select the HRPWM FineDutySlider from the GEL menu. A FineDuty slider graphics will show up in CCS.
   (a) Add "DutyFine" variable to the watch window
5. Load the program and run. Use the Slider to and observe the epwm edge displacement for each slider step change.

This explains the MEP control on the ePWMxA channels,

1. 15MHz PWM (for 150 MHz SYSCLKOUT) or 10MHz PWM (for 100MHz SYSCLKOUT),
   ePWM1A toggle low/high with MEP control on rising edge
   ePWM1B toggle low/high with NO HRPWM control

2. 7.5MHz PWM (for 150 MHz SYSCLKOUT) or 5MHz PWM (for 100MHz SYSCLKOUT),
   ePWM2A toggle low/high with MEP control on rising edge
   ePWM2B toggle low/high with NO HRPWM control
3. 15MHz PWM (for 150 MHz SYSCLKOUT) or 10MHz PWM (for 100MHz SYSCLKOUT),
   ePWM3A toggle as high/low with MEP control on falling edge
   ePWM3B toggle low/high with NO HRPWM control
4. 7.5MHz PWM (for 150 MHz SYSCLKOUT) or 5MHz PWM (for 100MHz SYSCLKOUT),
   ePWM4A toggle as high/low with MEP control on falling edge
   ePWM4B toggle low/high with NO HRPWM control

External Connections

- EPWM1A is on GPIO0
- EPWM1B is on GPIO1
- EPWM2A is on GPIO2
- EPWM2B is on GPIO3
- EPWM3A is on GPIO4
- EPWM3B is on GPIO5

Watch Variables

- DutyFine

5.31 High Resolution PWM Symmetric Duty Cycle SFO V5

This example modifies the MEP control registers to show symmetric edge displacement due to the HRPWM control extension of the respective ePWM module.

Note:
By default, this example project is configured for floating-point math. All included libraries must be pre-compiled for floating-point math. Therefore, SFO_TI_Build_V5B_fpu.lib (compiled for floating-point) is included in the project instead of the SFO_TI_Build_V5B.lib (compiled for fixed-point). To convert the example for fixed-point math, follow the instructions in sfo_readme.txt in the /doc directory of the header files and peripheral examples package.

This program requires the DSP2833x header files, which include the following files required for this example: SFO_V5.h and SFO_TI_Build_V5B_fpu.lib (or SFO_TI_Build_V5B.lib for fixed point)

This example calls the following TI's MEP Scale Factor Optimizer (SFO) software library V5 functions:

- **int SFO_MepDis_V5(int i)**; updates MEP_ScaleFactor[i] when HRPWM is not used
- **Returns**
  - 1 when complete for the specified channel
  - 0 if not complete for the specified channel

Channel ePWM1A will have fine edge movement due to the HRPWM logic when the duty cycle is altered.

Channel ePWM1B has a fixed 50% duty cycle.

- 5MHz PWM (for 150 MHz SYSCLKOUT), ePWM1A toggle high/low with MEP control on rising edge
- 3.33MHz PWM (for 100 MHz SYSCLKOUT), ePWM1A toggle high/low with MEP control on rising edge

Running the Application

1. ∗∗**!!IMPORTANT!!**∗∗ - in SFO_V5.h, set PWM_CH to the max number of HRPWM channels plus one. For example, for the F28335, the maximum number of HRPWM channels is 6. 6+1=7, so set #define PWM_CH 7 in SFO_V5.h. (Default is 7)
2. Run this example at 150/100MHz SYSCLKOUT
3. Watch ePWM1 waveforms on a Oscilloscope

External Connections

Monitor the following pins on an oscilloscope:

- ePWM1A is on GPIO0
- ePWM1B is on GPIO1

Watch Variables

- MEP_ScaleFactor
- EPwm1Regs.CMPA.all
- EPwm1Regs.CMPB.all

5.32 I2C EEPROM (i2c_eeprom)

This program requires an external I2C EEPROM connected to the I2C bus at address 0x50.

This program will write 1-14 words to EEPROM and read them back. The data written and the EEPROM address written to are contained in the message structure, **I2cMsgOut1**. The data read back will be contained in the message structure **I2cMsgIn1**.

Watch Variables

- I2cMsgIn1
- I2cMsgOut1

5.33 Low Power Modes: Halt Mode and Wakeup (lpm_haltwake)

This example puts the device into HALT mode. If the lowest possible current consumption in HALT mode is desired, the JTAG connector must be removed from the device board while the device is in

HALT mode.

The example then wakes up the device from HALT using GPIO0. GPIO0 wakes the device from HALT mode when a high-to-low signal is detected on the pin. This pin must be pulsed by an external agent for wakeup.

The wakeup process begins as soon as GPIO0 is held low for the time indicated in the device datasheet. After the device wakes up, GPIO1 can be observed to go high.

GPIO0 is configured as the LPM wakeup pin to trigger a WAKEINT interrupt upon detection of a low pulse. Initially, pull GPIO0 high externally. To wake device from halt mode, pull GPIO0 low for at least the crystal startup time + 2 OSCLKS, then pull it high again.

External Connections

- To observe when device wakes from HALT mode, monitor GPIO1 with an oscilloscope (set to 1 in WAKEINT ISR)

5.34 Low Power Modes: Device Idle Mode and Wakeup (lpm_idlewake)

This example puts the device into IDLE mode then wakes up the device from IDLE using XINT1 which triggers on a falling edge from GPIO0.

This pin must be pulled from high to low by an external agent for wakeup. GPIO0 is configured as an XINT1 pin to trigger an XINT1 interrupt upon detection of a falling edge.

Initially, pull GPIO0 high externally. To wake device from idle mode by triggering an XINT1 interrupt, pull GPIO0 low (falling edge).

External Connections

- To observe the device wakeup from IDLE mode, monitor GPIO1 with an oscilloscope, which goes high in the XINT_1_ISR.

5.35 Low Power Modes: Device Standby Mode and Wakeup (lpm_standbywake)

This example puts the device into STANDBY mode. If the lowest possible current consumption in STANDBY mode is desired, the JTAG connector must be removed from the device board while the device is in STANDBY mode.

GPIO0 is configured as the LPM wakeup pin to trigger a WAKEINT interrupt upon detection of a low pulse. Initially, pull GPIO0 high externally. To wake device from standby mode, pull GPIO0 low for at least (2+QUALSTDBY) OSCLKS, then pull it high again.

The example then wakes up the device from STANDBY using GPIO0. GPIO0 wakes the device from STANDBY mode when a low pulse (signal goes high->low->high)is detected on the pin. This pin must be pulsed by an external agent for wakeup.

As soon as GPIO0 goes high again after the pulse, the device should wake up, and GPIO1 can be observed to toggle.

External Connections

- To observe when device wakes from STANDBY mode, monitor GPIO1 with an oscilloscope (set to 1 in WAKEINT ISR)

5.36 McBSP Digital Loop Back (mcbsp_loopback)

This example performs digital loopback tests for the McBSP peripheral. This example does not use interrupts. Instead, a polling method is used to check the receive data. The incoming data is checked for accuracy. If an error is found the error() function is called and execution stops.

Watch Variables

- sdata1
- sdata2
- rdata1
- rdata2
- rdata1_point
- rdata2_point

5.37 McBSP Digital Loop Back with DMA (mcbsp_loopback_dma)

This program is a McBSP example that uses the internal loopback of the peripheral and utilizes the DMA to transfer data from one buffer to the McBSP, and then from the McBSP to another buffer.

Initially, sdata[] is filled with values from 0x0000- 0x007F. The DMA moves the values in sdata[] one by one to the DXRx registers of the McBSP. These values are transmitted and subsequently received by the McBSP. Then, the DMA moves each data value to rdata[] as it is received by the McBSP.

By default for the McBSP examples, the McBSP sample rate generator (SRG) input clock frequency is LSPCLK (150E6/4 or 100E6/4) assuming SYSCLKOUT = 150 MHz or 100 MHz respectively. If while testing, the SRG input frequency is changed, the define MCBSP_SRG_FREQ (150E6/4 or 100E6/4) in the Mcbsp.c file must also be updated accordingly. This define is used to determine the Mcbsp initialization delay after the SRG is enabled, which must be at least 2 SRG clock cycles.

Watch Variables

- sdata
- rdata

5.38 McBSP Digital Loop Back with Interrupts (mcbsp_loopback_interrupts)

This program uses the internal loopback of the peripheral. Both Rx and Tx interrupts are enabled.

Incrementing values from 0x0000 to 0x00FF are being sent and received.

This pattern is repeated forever.

By default for the McBSP examples, the McBSP sample rate generator (SRG) input clock frequency is LSPCLK (150E6/4 or 100E6/4) assuming SYSCLKOUT = 150 MHz or 100 MHz respectively. If while testing, the SRG input frequency is changed, the define MCBSP_SRG_FREQ (150E6/4 or 100E6/4) in the Mcbsp.c file must also be updated accordingly. This define is used to determine the Mcbsp initialization delay after the SRG is enabled, which must be at least 2 SRG clock cycles.

Watch Variables

- sdata
- rdata
- rdata_point

5.39 McBSP Digital Loop Back using SPI Mode (mcbsp_spi_loopback)

This program will execute and transmit words until terminated by the user.

By default for the McBSP examples, the McBSP sample rate generator (SRG) input clock frequency is LSPCLK (150E6/4 or 100E6/4) assuming SYSCLKOUT = 150 MHz or 100 MHz respectively. If while testing, the SRG input frequency is changed, the define MCBSP_SRG_FREQ (CPU_SPD/4) in the Mcbsp.c file must also be updated accordingly. This define is used to determine the Mcbsp initialization delay after the SRG is enabled, which must be at least 2 SRG clock cycles.

Watch Variables

- sdata1
- sdata2
- rdata1
- rdata2

5.40 SCI Autobaud (sci_autobaud)

This test will perform autobaud lock at a variety of baud rates, including very high baud rates.

For this test to properly run, connect the SCI-A pins to the SCI-B pins without going through a transceiver. At higher baud rates, the slew rate of the incoming data bits can be affected by transceiver and connector performance. This slew rate may limit reliable autobaud detection at higher baud rates.

SCIA: Slave, autobaud locks, receives characters and echos them back to the host. Uses the RX interrupt to receive characters.

SCIB: Host, known baud rate, sends characters to the slave and checks that they are echoed back.

External Connections

- SCITXDA is on GPIO29
- SCIRXDB is on GPIO19
- SCIRXDA is on GPIO28
- SCITXDB is on GPIO18
- Connect GPIO29 to GPIO19
- Connect GPIO28 to GPIO18

Watch Variables

- BRRVal - current BRR value used for SCIB
- ReceivedAChar - character received by SCIA
- ReceivedBChar - character received by SCIB
- SendChar - character being sent by SCIB
- SciaRegs.SCILBAUD - SCIA baud register set by autobaud lock
- SciaRegs.SCIHBAUD - SCIA baud register set by autobaud lock

5.41 SCI Echo Back (sci_echoback)

This test receives and echo-backs data through the SCI-A port.

The PC application 'hypterterminal' can be used to view the data from the SCI and to send information to the SCI. Characters received by the SCI port are sent back to the host.

Running the Application

1. Configure hyperterminal: Use the included hyperterminal configuration file SCI_96.ht. To load this configuration in hyperterminal
   (a) Open hyperterminal
   (b) Go to file->open
   (c) Browse to the location of the project and select the SCI_96.ht file.
2. Check the COM port. The configuration file is currently setup for COM1. If this is not correct, disconnect (Call->Disconnect) Open the File-Properties dialog and select the correct COM port.
3. Connect hyperterminal Call->Call and then start the 2833x SCI echoback program execution.
4. The program will print out a greeting and then ask you to enter a character which it will echo back to hyperterminal.

As is, the program configures SCI-A for 9600 baud with SYSCLKOUT = 150MHz and LSPCLK = 37.5 MHz or SYSCLKOUT = 100MHz and LSPCLK = 25.0 Mhz

Watch Variables

- LoopCount - Number of characters sent
- ErrorCount

External Connections

Connect the SCI-A port to a PC via a transceiver and cable.

- GPIO28 is SCI_A-RXD (Connect to Pin3, PC-TX, of serial DB9 cable)
- GPIO29 is SCI_A-TXD (Connect to Pin2, PC-RX, of serial DB9 cable)

5.42 SCI Digital Loop Back (scia_loopback)

This program uses the internal loop back test mode of the peripheral. Other then boot mode pin configuration, no other hardware configuration is required.

This test uses the loopback test mode of the SCI module to send characters starting with 0x00 through 0xFF. The test will send a character and then check the receive buffer for a correct match.

Watch Variables

- LoopCount - Number of characters sent
- ErrorCount - Number of errors detected
- SendChar - Character sent
- ReceivedChar - Character received

5.43 SCI Digital Loop Back with Interrupts (scia_loopback_interrupts)

This program uses the internal loop back test mode of the peripheral. Other then boot mode pin configuration, no other hardware configuration is required. Both interrupts and the SCI FIFOs are used.

A stream of data is sent and then compared to the received stream.

The SCI-A sent data looks like this:

00 01 02 03 04 05 06 07

01 02 03 04 05 06 07 08

02 03 04 05 06 07 08 09

....

FE FF 00 01 02 03 04 05

FF 00 01 02 03 04 05 06

etc..

The SCI-B sent data looks like this:

FF FE FD FC FB FA F9 F8

FE FD FC FB FA F9 F8 F7

FD FC FB FA F9 F8 F7 F6

....

01 00 FF FE FD FC FB FA

00 FF FE FD FC FB FA F9

etc..

Both patterns are repeated forever.

Watch Variables

- sdataA, sdataB - Data to send
- rdataA, rdataB - Received data
- rdata_pointA, rdata_pointB - Used to keep track of the last position in the receive stream for error checking

5.44 SPI Digital Loop Back (spi_loopback)

This program uses the internal loop back test mode of the peripheral. Other then boot mode pin configuration, no other hardware configuration is required. Interrupts are not used.

A stream of data is sent and then compared to the received stream. The sent data looks like this:

0000 0001 0002 0003 0004 0005 0006 0007 .... FFFE FFFF

This pattern is repeated forever.

Watch Variables

- sdata - Sent data
- rdata - Received data

5.45 SPI Digital Loop Back with Interrupts (spi_loopback_interrupts)

This program uses the internal loop back test mode of the peripheral. Other then boot mode pin configuration, no other hardware configuration is required. Both interrupts and the SPI FIFOs are used.

A stream of data is sent and then compared to the received stream. The sent data looks like this:

0000 0001 0002 0003 0004 0005 0006 0007

0001 0002 0003 0004 0005 0006 0007 0008

0002 0003 0004 0005 0006 0007 0008 0009

....

FFFE FFFF 0000 0001 0002 0003 0004 0005

FFFF 0000 0001 0002 0003 0004 0005 0006

etc..

This pattern is repeated forever.

Watch Variables

- sdata - Data to send
- rdata - Received data
- rdata_point - Used to keep track of the last position in the receive stream for error checking

5.46 Software Prioritized Interrupts (sw_prioritized_interrupts)

For most applications, the hardware prioritizing of the the PIE module is sufficient. For applications that need custom prioritizing, this example illustrates how this can be done through software.

This program simulates interrupt conflicts by writing to the PIEIFR registers. This will cause multiple interrupt requests to come into the PIE block at the same time.

The interrupt service routines are software prioritized as per the table found in the DSP2833x_SWPrioritizedIsrLevels.h file.

Running the Application

1. Before compiling you must set the Global and Group interrupt priorities in the DSP2833x_SWPrioritizedIsrLevels.h file.
2. Select which test case you'd like to run with the #define CASE directive (1-9, default 1).
3. Compile the code, load, and run
4. At the end of each test there is a hard coded breakpoint (ESTOP0). When code stops at the breakpoint, examine the ISRTrace buffer to see the order in which the ISR's completed. All PIE interrupts will be added to the ISRTrace. The ISRTrace will consist of a list of hex values as shown:
   0x00wx <- PIE Group w interrupt x finished first
   0x00yz <- PIE Group y interrupt z finished next
5. If desired, set a new set of Global and Group interrupt priorities and repeat the test to see the change.

Watch Variables

- ISRTrace - Trace of ISR's in the order they complete.
  After each test, examine this buffer to determine if the ISR's completed in the order desired.

5.47 Timer based blinking LED (timed_led_blink)

This example configures CPU Timer0 for a 500 msec period, and toggles the GPIO32 LED on the 2833x eZdsp once per interrupt. For testing purposes, this example also increments a counter each time the timer asserts an interrupt.

Watch Variables

- CpuTimer0.InterruptCount

External Connections

- Monitor the GPIO32 LED blink on (for 500 msec) and off (for 500 msec) on the 2833x eZdsp.

5.48 Watchdog interrupt Test (watchdog)

This program exercises the watchdog.

First the watchdog is connected to the WAKEINT interrupt of the PIE block. The code is then put into an infinite loop.

The user can select to feed the watchdog key register or not by commenting the following line of code in the infinite loop: **ServiceDog()**;

If the watchdog key register is fed by the ServiceDog function then the WAKEINT interrupt is not taken. If the key register is not fed by the ServiceDog function then WAKEINT will be taken.

Watch Variables

- **LoopCount** - Number of times through the infinite loop
- **WakeCount** - Number of times through WAKEINT

5.49 Code Run from XINTF (xintf_run_from)

This example configures CPU Timer0 and increments a counter each time the timer asserts an interrupt.

The code is loaded into SARAM. The XINTF Zone 7 is configured for x16-bit data bus. A portion of the code is copied to XINTF for execution there.

Watch Variables

- CpuTimer0.InterruptCount
- CpuTimer1.InterruptCount
- CpuTimer2.InterruptCount

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	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265