# F2837xD Firmware Development Package

# USER'S GUIDE


F2837xD-FRM-EX-UG-100

# Copyright



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

# Revision Information

This is version 100 of this document, last updated on Mon Dec 9 12:58:58 CST 2013.

# Table of Contents

# 1 Introduction

The Texas Instruments® F2837xD Firmware development library is a group of example applications and helper libraries that demonstrate the basics of getting started with a F2837xD device.

**The following chapter (chapter 2) provides a step by step guide for from scratch project creation for each core as well as debug. It is highly recommended that users new to the F2837xD family of devices start by reading this section first.**

Because the F2837xD devices have two cores the example applications have been broken up to distinguish which examples run on each core.

- The example applications which run exclusively on the CPU 1 core can be found in the `F2837xD_examples_Cpu1` directory.
- The example applications which require both cores to run can be found in the `F2837xD_examples_Dual` 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. Example projects have a heirarchy as follows:

- Main project directory
  - CPU 1 project folder (cpu01)
    - CPU 1 project sources (∗.c, ∗.h)
    - CCS folder (ccs)
      - CCS project specific files
  - CPU 2 project folder (cpu02)
    - CPU 2 project sources (∗.c, ∗.h)
    - CCS folder (ccs)
      - CCS project specific files

TI also recommends that users append either _cpu01 or _cpu02 to project names to help developers differentiate between projects with similar names.

# 2 Getting Started and Troubleshooting



## 2.1 Introduction

Because of the sheer complexity of the F2837xD 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. This guide will focus on the user of a F2837xD controlCARD, but these same ideas should apply to other boards with minimal translation.

## 2.2 Project Creation

A typical F2837xD application consists of two seperate CCS projects: one for CPU 1 and one for CPU2. The two projects are completely independent and have no real linking between them as far as CCS is concerned.

### CPU 1 Subsystem 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.

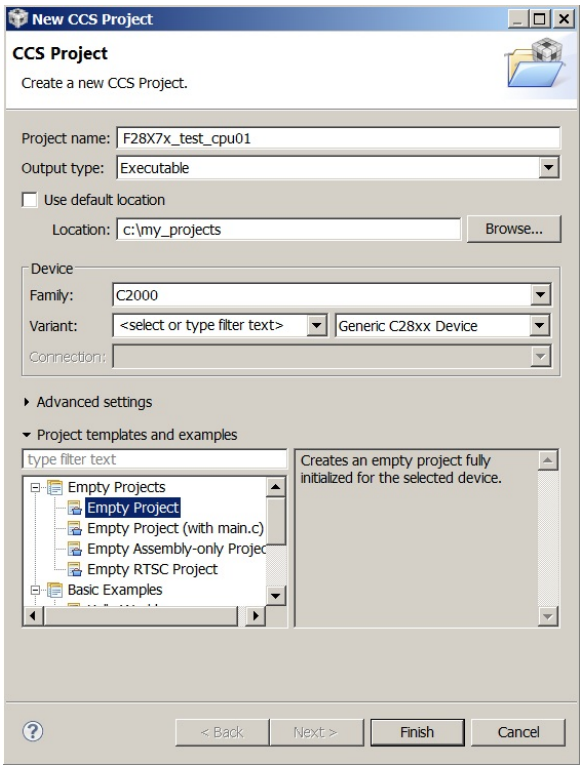


Figure 2.1: Creating a new C28 project

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:

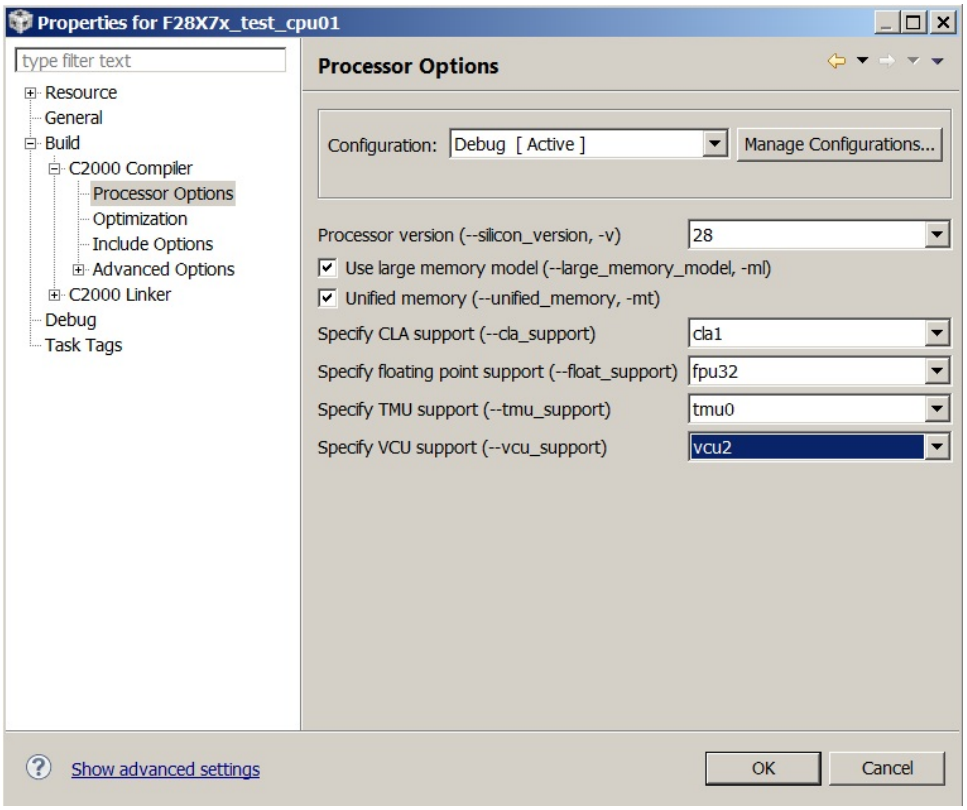


Figure 2.2: Project configuration dialog box

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 `F2837xD_common\include` folder of your controlSUITE installation (typically `C:\TI\controlSUITE\device_support\F2837xD\VERSION\F2837xD_common\include`). Click ok to add this path, and repeat this same process to add the `F2837xD_headers\include` directory.

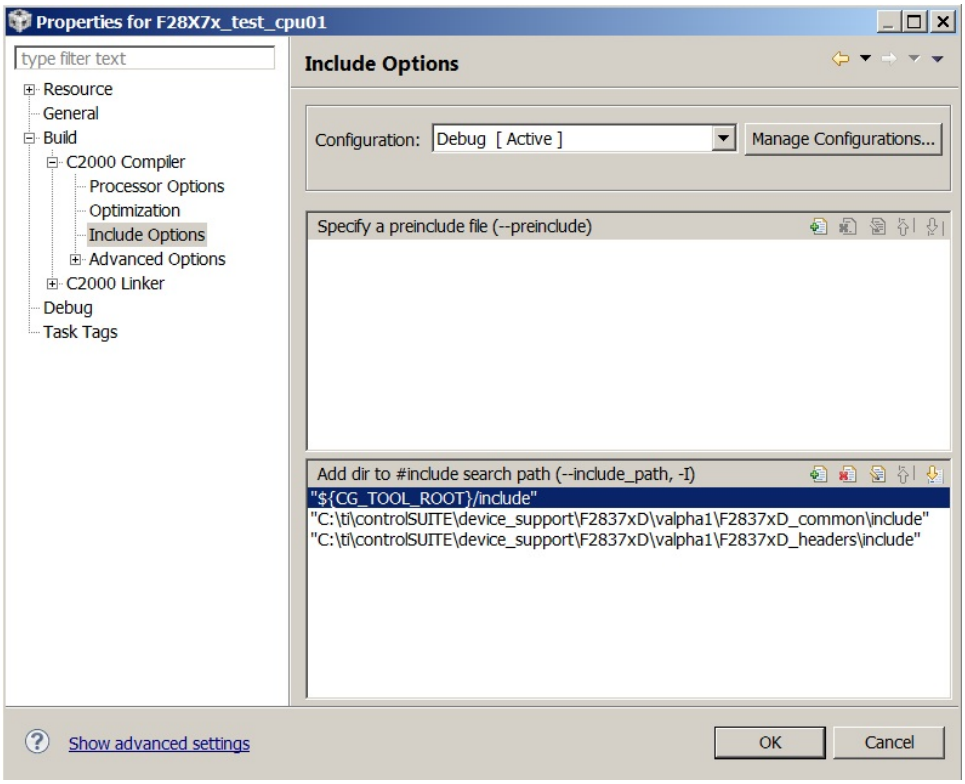


Figure 2.3: Project configuration dialog box

4. Expand the Advanced Options and look for the Predefined Symbol entry. Add a Pre-define NAME called "CPU1". This ensures that the header files build correct for this CPU.

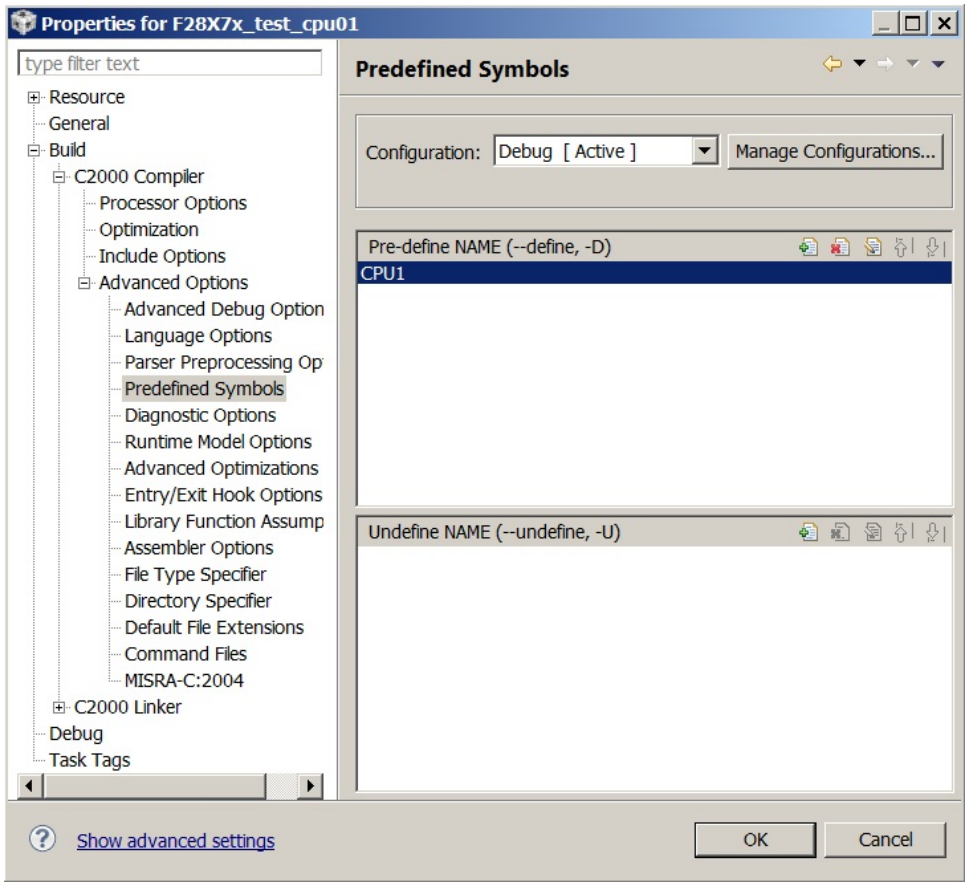


Figure 2.4: Project configuration dialog box

5. Click on the Linker File Search Path. Add these directories to the search path: `F2837xD_common\cmd` and `F2837xD_headers\cmd`. Then you'll also want to add the following files: `rts2800_fpu32.lib`, `2837x_RAM_lnk_cpu1.cmd`, and `F2837x_Headers_nonBIOS_cpu1.cmd`. Finally, delete libc.a, we will use rts2800_fpu32.lib as our run time support library instead.

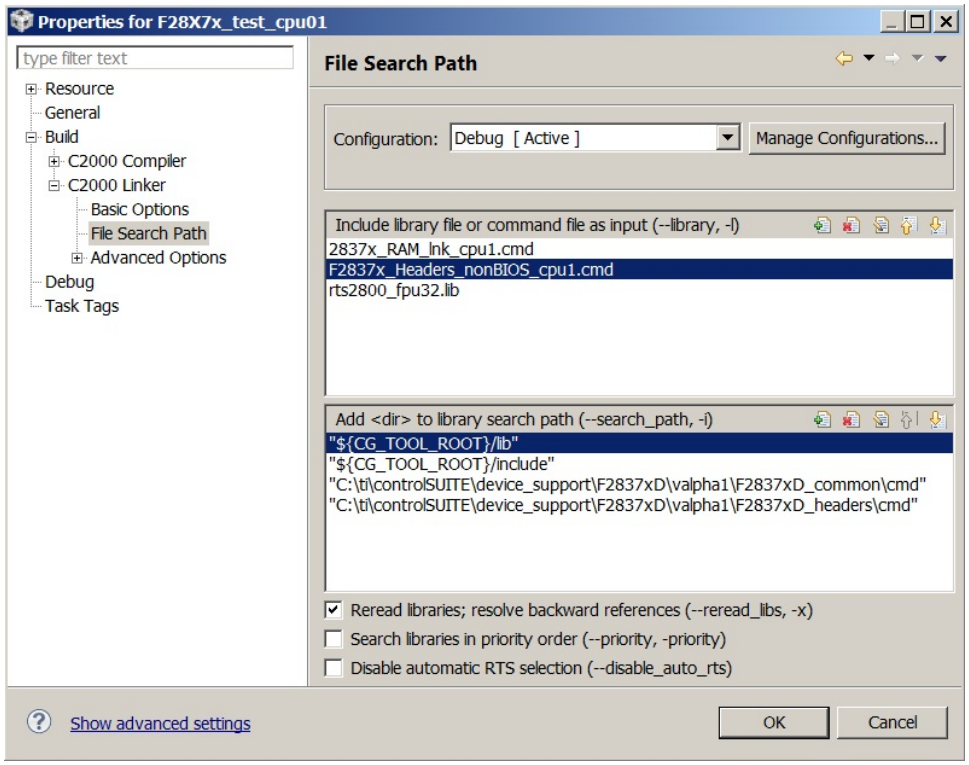


Figure 2.5: Project configuration dialog box

6. 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.

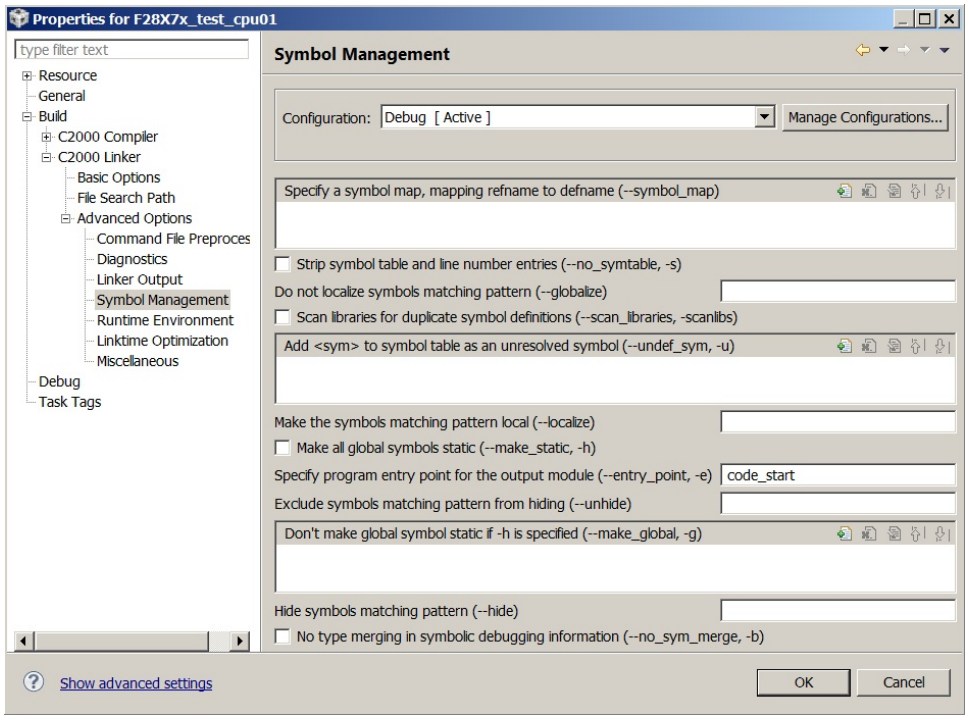


Figure 2.6: Include path setup

7. 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 `F2837xD_headers\source` directory, and select F2837xD_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. Link in the following files as well:
   - `F2837xD_common\source\F2837xD_CodeStartBranch.asm`
   - `F2837xD_common\source\F2837xD_usDelay.asm`
   - `F2837xD_common\source\F2837xD_SysCtrl.c`
   - `F2837xD_common\source\F2837xD_Gpio.c`
   - `F2837xD_common\source\F2837xD_Ipc.c`

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

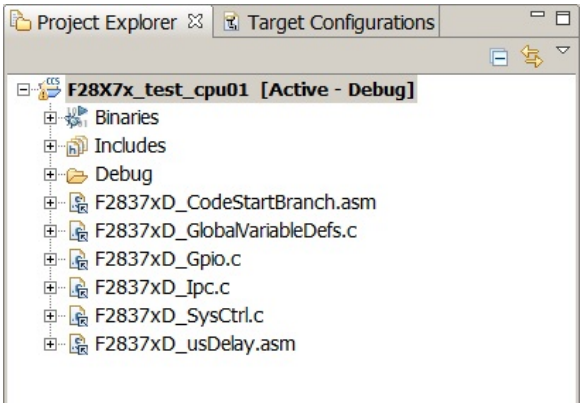


Figure 2.7: Linking files to project

8. 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 "F28x_Project.h"

void main(void)
{
uint32_t delay;

InitSysCtrl();

    // Set pin direction
    EALLOW;
    GpioCtrlRegs.GPADIR.bit.GPIO10 = 1;
    EDIS;

    GPIO_SetupPinOptions(14, GPIO_OUTPUT, GPIO_PUSHPULL);
    GPIO_SetupPinMux(14, GPIO_MUX_CPU2, 0);


    // turn off LED
    GpioDataRegs.GPADAT.bit.GPIO10 = 1;


    while(1)
    {

        // Turn on LED
     GpioDataRegs.GPADAT.bit.GPIO10 = 0;

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

        // Turn off LED
        GpioDataRegs.GPADAT.bit.GPIO10 = 1;

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

    }

}
```

9. Save main.c and then attemp to build the project by right click on it and selecting Build Project. Assuming the project builds try debugging this project on a F2837xD device. When the code runs you should see GPIO 10 toggle.

### CPU 2 Subsystem 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.

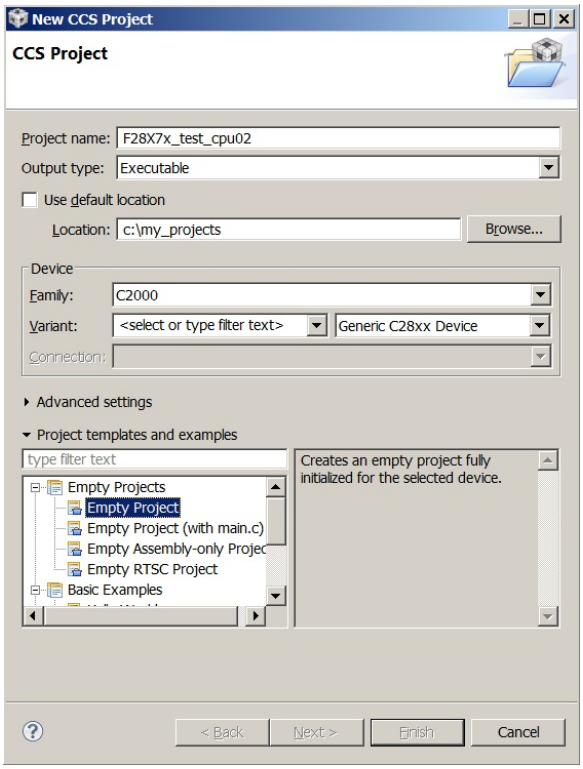


Figure 2.8: Creating a new C28 project

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:

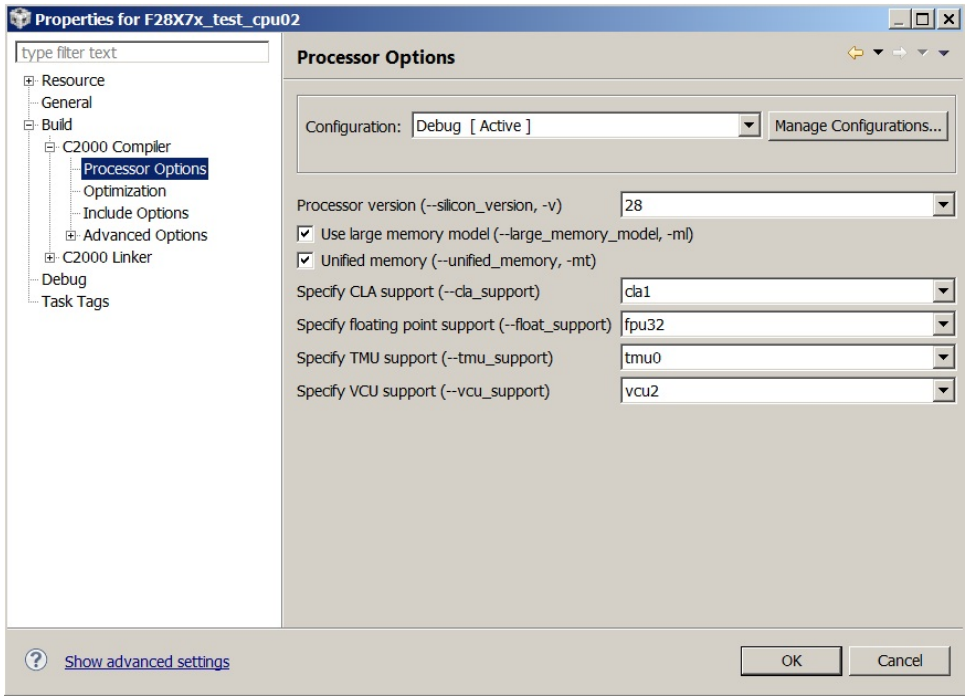


Figure 2.9: Project configuration dialog box

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 `F2837xD_common\include` folder of your controlSUITE installation (typically `C:\TI\controlSUITE\device_support\F2837xD\VERSION\F2837xD_common\include`). Click ok to add this path, and repeat this same process to add the `F2837xD_headers\include` directory.

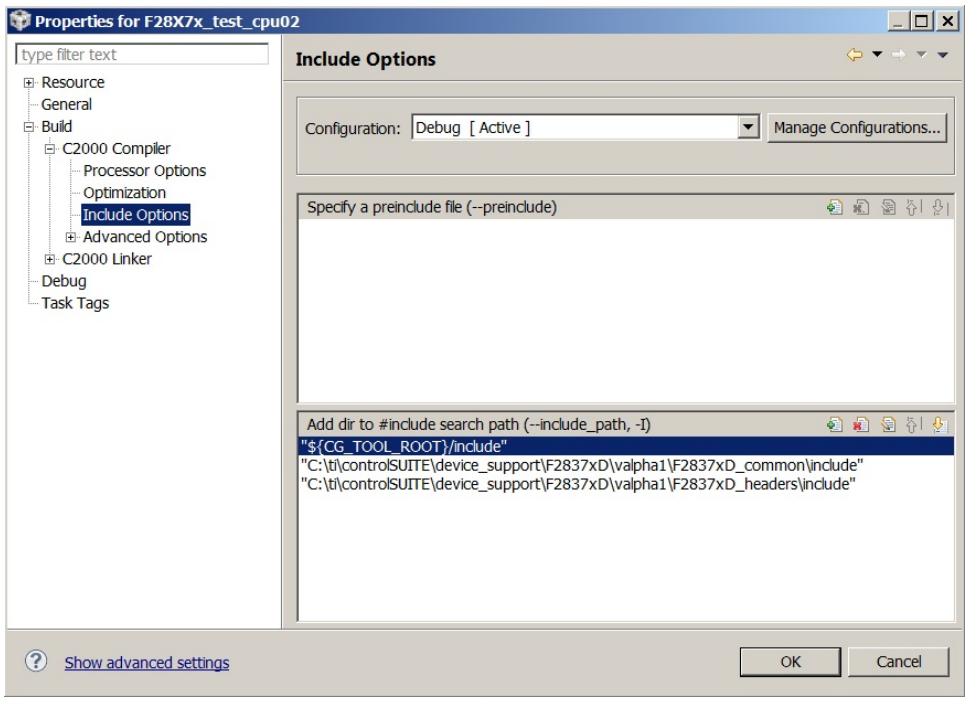


Figure 2.10: Project configuration dialog box

4. Expand the Advanced Options and look for the Predefined Symbol entry. Add a Pre-define NAME called "CPU2". This ensures that the header files build correct for this CPU.

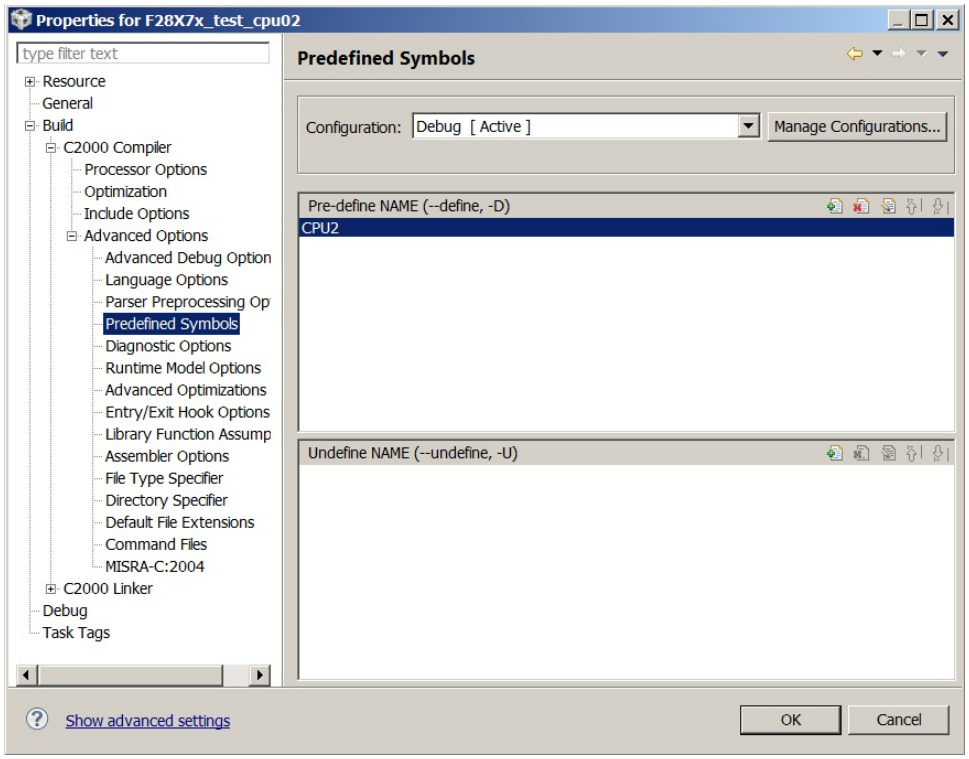


Figure 2.11: Project configuration dialog box

5. Click on the Linker File Search Path. Add these directories to the search path: `F2837xD_common\cmd` and `F2837xD_headers\cmd`. Then you'll also want to add the following files: `rts2800_fpu.lib`, `2837x_RAM_lnk_cpu2.cmd`, and `F2837x_Headers_nonBIOS_cpu2.cmd`. Finally, delete libc.a, we will use rts2800_fpu.lib as our run time support library instead.

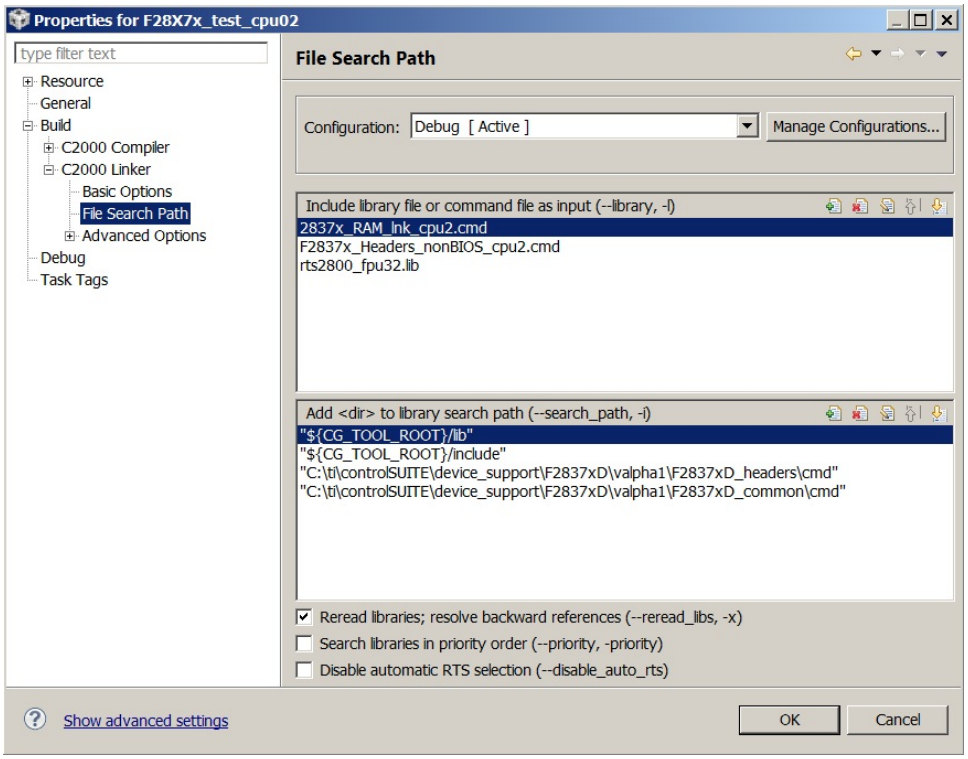


Figure 2.12: Project configuration dialog box

6. 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.

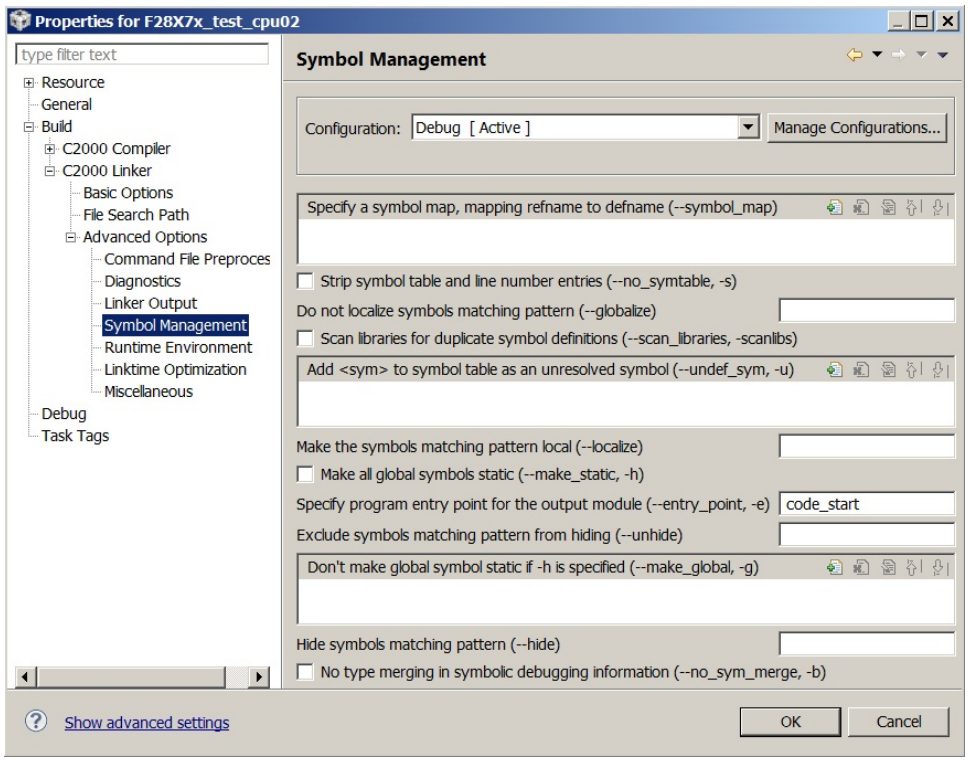


Figure 2.13: Include path setup

7. 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 `F2837xD_headers\source` directory, and select F2837xD_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. Link in the following files as well:
   - `F2837xD_common\source\F2837xD_CodeStartBranch.asm`
   - `F2837xD_common\source\F2837xD_usDelay.asm`
   - `F2837xD_common\source\F2837xD_SysCtrl.c`
   - `F2837xD_common\source\F2837xD_Gpio.c`
   - `F2837xD_common\source\F2837xD_Ipc.c`

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

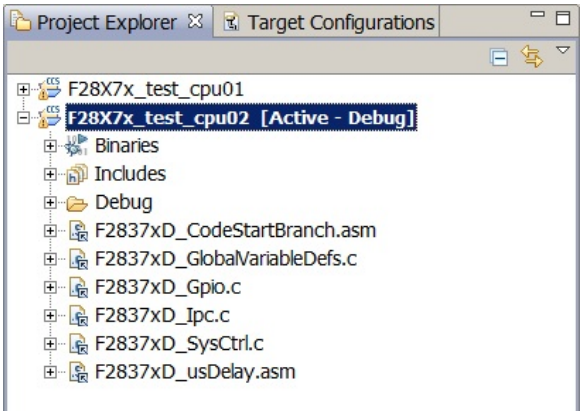


Figure 2.14: Linking files to project

8. 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 "F28x_Project.h"

void main(void)
{
uint32_t delay;

    // turn off LED
    GpioDataRegs.GPADAT.bit.GPIO14 = 1;


    while(1)
    {

        // Turn on LED
     GpioDataRegs.GPADAT.bit.GPIO14 = 0;

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

        // Turn off LED
        GpioDataRegs.GPADAT.bit.GPIO14 = 1;

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

    }

}
```

9. Save main.c and then attemp to build the project by right click on it and selecting Build Project. Assuming the project builds try debugging both these projects simultaneously on a F2837xD device, otherwise carefully examine the error and the above steps to determine what could have gone wrong.

## 2.3 Debugging Dual Core Applications

1. Ensure CCS version 5 is installed and up to date. You should have C2000 Code Generation Tools verion 6.2.0 or later.
2. Connect a USB Mini cable from the computer to the USB port on the left hand side of the controlCARD. Windows will enumerate and try to install drivers. As long as CCS is installed, Windows should automatically find and install drivers for the emulator.
3. Apply power either via USB or the 5V DC in jack on the docking station. While the emulator on the board is powered from the host computer's USB port, the rest of the board is not.

The reason for this is that the JTAG connection on the F2837xD controlCARDs is completely electrically isolated. Because of the typical applications these devices will be used in, it is neccessary to isolate the JTAG connection. However, for bench debug and evaluation (with low voltages), both halves of the board can be powered from the same supply (i.e. USB). Each power domain has an associated power LED which can be used to ensure that each domain has power.

4. Launch CCS and pick the workspace you would like to debug in.
5. Create a new target configuration. Click File -> New -> Target Configuration File and name the file appropriately (i.e. F2837xD_xds100.ccxml). Select the emulator you intend to use (XDS100v2) from the drop down list, and then select the device variant present on your board (F2837xD controlCARDs have a F2837xD). Save the target configuration and close the window.

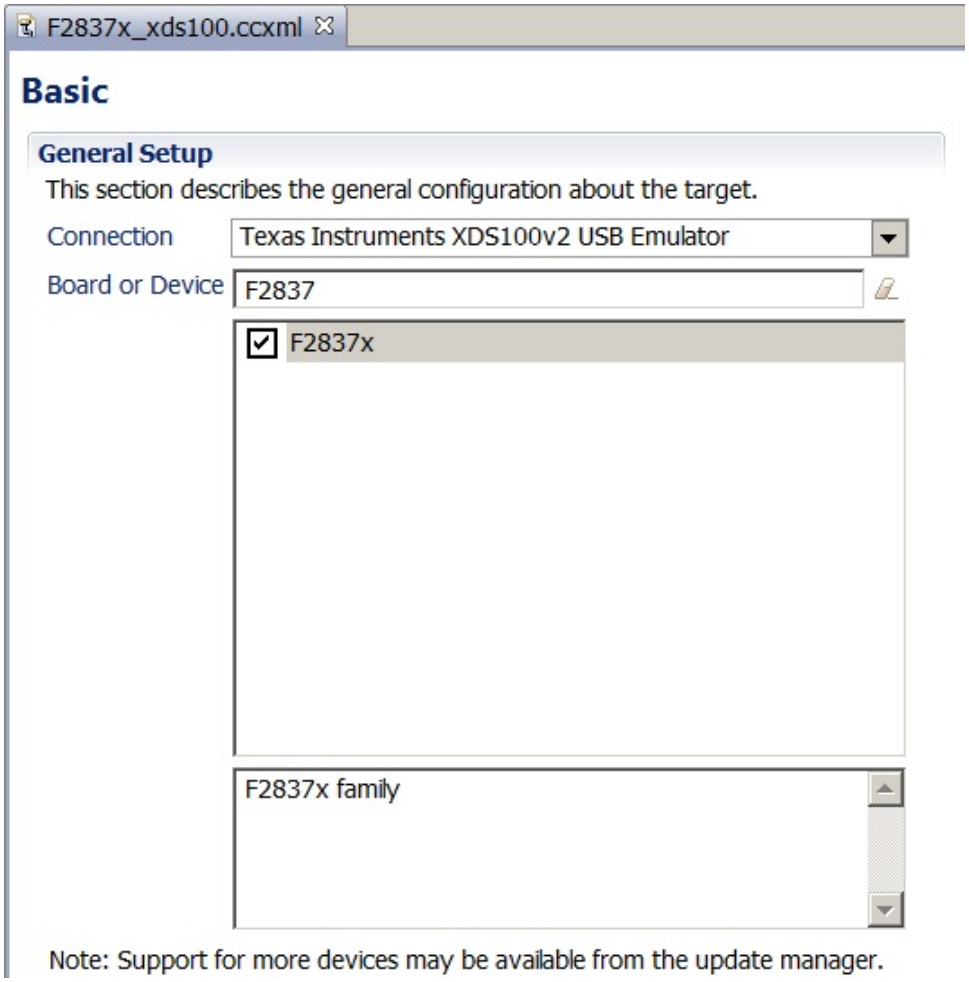


Figure 2.15: F2837xD Card Target Configuration Setup

6. Import the desired example projects (or skip this step if you are using projects you created in the Project Creation section). Click File -> Import, and in the CCS folder select Existing CCS/CCE Eclipse Projects before clicking Next. With the "Select search-directory" radio button checked, browse to the root of your controlSUITE installation. Device specific software as well as examples are stored in the `device_support/device_variant` folders. Navigate

to the `F2837xD` directory, and then to the `F2837xD_examples_Dual` directory. Click OK and CCS will parse all of the projects in this directory. Import any projects you wish to run into the workspace. **Do not select "Copy projects into workspace".** These projects link to external resources relatively, so taking them out of controlSUITE will break the project.

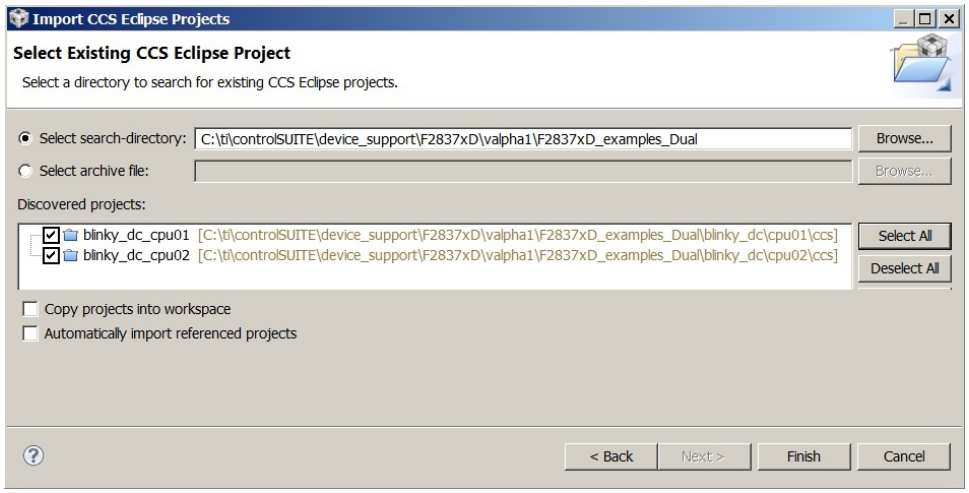


Figure 2.16: Importing Concerto Projects

7. Build each of the example projects. Right click on each project title and select build project.

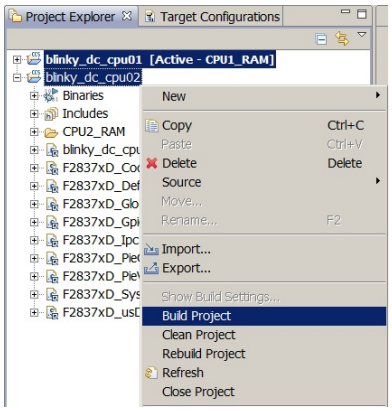


Figure 2.17: Building Concerto Projects

8. Launch the previously created target configuration. Click View -> Target Configurations. In the window that opens, find the target configuration you created previously, right click on it and select "Launch Target Configuration".

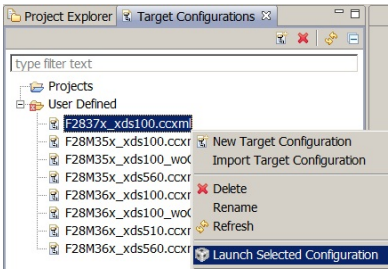


Figure 2.18: Launching a CCS Target Configuration

9. Connect to the device. Right click on each core in the debug window and select "Connect Target. This will connect CCS to the device and will allow you to load code and debug applications.

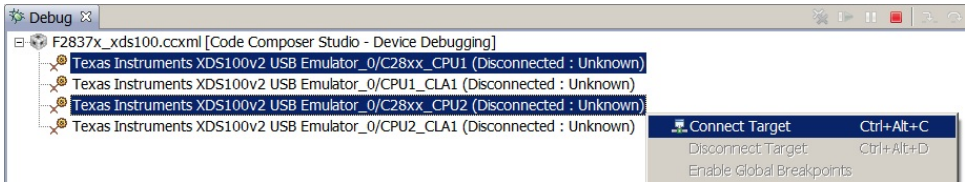


Figure 2.19: Connecting to a Target

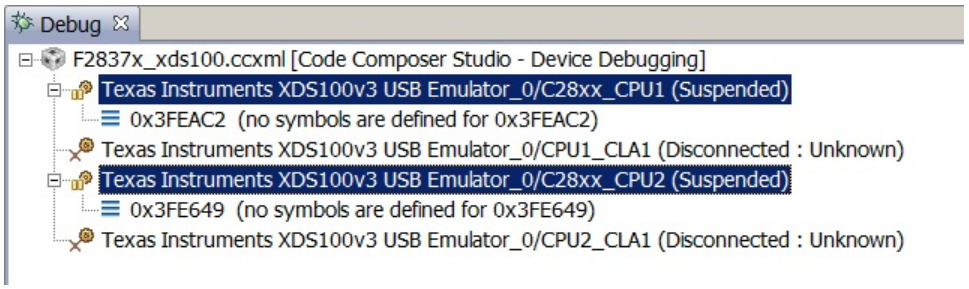


Figure 2.20: After connection to both cores

10. Load code on each of the cores. Select one of the cores in the debug window and then click Target -> Load Program. A dialog box is display which will allow you to select a program to load. Be careful to ensure that you load the appropriate out file on the appropriate core. Repeat this process for the other core by selecting it and following these same steps.
11. At this point both cores should have code loaded and be halted at main. From this point, users should be able to debug code just as they are used to with CCS. Please keep in mind that any action you take in CCS only has an effect on the core you currently have selected in the debug window. For instance if CPU 1 is selected, the memory window will display the memory map of of the system as seen by CPU 1. The opposite would be true if CPU 2 were selected.

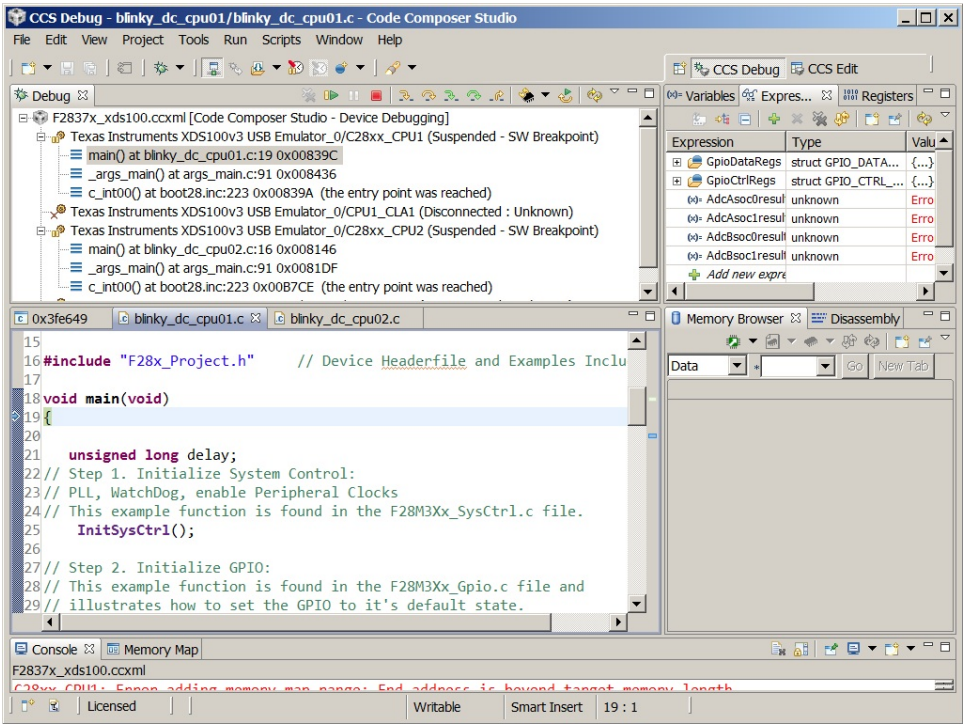


Figure 2.21: Projects loaded on each core

## 2.4 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 Concerto 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 F2837xD 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 F2837xD 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 deviceŠs memory map. If these appear correct, there is a chance there is something wrong in one of your GEL scripts.

**"The core doesn't run to main when I load code"**

Because the master subsystem is ultimately in control of the entire device, initially the C28x is held in reset by the M3 so it will not run. The GEL files shipped with controlSUITE take the C28x out of reset automatically, so you should not have this problem. However, if you do see behavior similar to this, look in the C28 GEL file you are using in the Device_Config function. You should see a line like the following:

- `*(unsigned long *)0x400FB8C0 |= 0x00030001; // Release C28 from Reset`

Comment or uncomment this line to your liking. In a stand alone application, the M3 application must manually release the C28 from reset before it will start running code.

A Non-Maskable Interrupt (NMI) can also potentially cause your debug session to not run to main. If during boot the device detects an NMI (such as missing clock, or brownout), the device will go to the bootROM NMI function and remain there.

**"When a core gets an interrupt it faults"**

Ensure that the interrupt vector table is where the interrupt controller thinks it is. On both cores 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 CPU1 comes up, it is not fresh out of reset"**

F2837xD devices support several boot modes, several of which allow program code to be loaded into and executed out of RAM via one of the deviceŠs 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 2803x 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 F2837xD 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 F2837xD 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 F2837xD 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 F2837xD 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 F28 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 F2837xD_common/include/F2837xD_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 F28_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 CPU 1 Example Applications

These example applications show how to make use of various peripherals of a F2837xD device. These applications are intended for demonstration and as a starting point for new applications.

All these examples contain two build configurations which allow you to build each project to run from either RAM or Flash. To change how the project is built simply right click on the project and select "Build Configurations". Then, move over to set the active build configuration, either RAM or Flash.

Because CPU 1 is ultimately in control of the entire F2837xD device and these applications contain no CPU 2 dependencies, these examples may be run completely on their own without any associated CPU2 program. The only exception to this in the CPU1 examples is the setup_cpu1 example. This example sets up all of the peripherals and GPIOs to be owned by CPU2. In addition, this example also has a special standalone flash build configuration which will send an IPC command to boot the second CPU and run the application in its flash memory.

All of these examples reside in the `device_support/F2837xD/<Version>/F2837xD_examples_Cpu1` subdirectory of the ControlSUITE package.

## 4.1 ADC PPB Delay Capture

This example demonstrates delay capture using the post-processing block.

Two asynchronous ADC triggers are setup:

ePWM1, with period 2048, triggering SOC0 to convert on pin A0

ePWM1, with period 9999, triggering SOC1 to convert on pin A1

Each conversion generates an ISR at the end of the conversion. In the ISR for SOC0, a conversion counter is incremented and the PPB is checked to determine if the sample was delayed.

After the program runs, the memory will contain:

**conversion** : the sequence of conversions using SOC0 that were delayed

**delay** : the corresponding delay of each of the delayed conversions

## 4.2 ADC PPB Limits

This example sets up the ePWM to periodically trigger the ADC. If the results are outside of the defined range, the post-processing block will generate an interrupt.

The default limits are 1000LSBs and 3000LSBs. With VREFHI set to 3.3V, the PPB will generate an interrupt if the input voltage goes above about 2.4V or below about 0.8V.

## 4.3 ADC PPB Offset

This example software triggers the ADC. Some SOCs have automatic offset adjustment applied by the post-processing block.

After the program runs, the memory will contain:

**AdcaResult** : a digital representation of the voltage on pin A0

**AdcaResult_offsetAdjusted** : a digital representation of the voltage on pin A0, plus 100 LSBs of automatically added offset

**AdcbResult** : a digital representation of the voltage on pin B0

**AdcbResult_offsetAdjusted** : a digital representation of the voltage on pin B0 minus 100 LSBs of automatically added offset

## 4.4 ADC Continuous Triggering

This example sets up the ADC to convert continuously, achieving maximum sampling rate.

After the program runs, the memory will contain:

**AdcaResults** : A sequence of analog-to-digital conversion samples from pin A0. The time between samples is the minimum possible based on the ADC speed.

## 4.5 ADC ePWM Triggering

This example sets up the ePWM to periodically trigger the ADC

After the program runs, the memory will contain:

**AdcaResults** : A sequence of analog-to-digital conversion samples from pin A0. The time between samples is determined based on the period of the ePWM timer.

## 4.6 ADC SOC Software Force

This example converts some voltages on ADCA and ADCB based on a software trigger.

After the program runs, the memory will contain:

**AdcaResult0** : a digital representation of the voltage on pin A0

**AdcaResult1** : a digital representation of the voltage on pin A1

**AdcbResult0** : a digital representation of the voltage on pin B0

**AdcbResult1** : a digital representation of the voltage on pin B1

## 4.7 Blinky

This example blinks LED X

## 4.8 Blinky

This example blinks LED X

## 4.9 Buffered DAC Enable

This example enables Buffered DACA output DACOUTA/ADCINA0 (HSEC Pin 9) and uses the default DAC reference setting of VDAC.

When the DAC reference is set to VDAC, an external reference voltage must be applied to the VDAC pin. This can accomplished by connecting a jumper wire from 3.3V to ADCINB0 (HSEC pin 12)

## 4.10 CAN External Loopback (can_loopback)

This example shows the basic setup of CAN in order to transmit and receive messages on the CAN bus. The CAN peripheral is configured to transmit messages with a specific CAN ID. A message is then transmitted once per second, using a simple delay loop for timing. The message that is sent is a 4 byte message that contains an incrementing pattern. A CAN interrupt handler is used to confirm message transmission and count the number of messages that have been sent.

This example sets up the CAN controller in External Loopback test mode. Data transmitted is visible on the CAN0TX pin and can be received with an appropriate mailbox configuration.

## 4.11 CAN External Loopback (can_loopback)

This example shows the basic setup of CAN in order to transmit and receive messages on the CAN bus. The CAN peripheral is configured to transmit messages with a specific CAN ID. A message is then transmitted once per second, using a simple delay loop for timing. The message that is sent is a 4 byte message that contains an incrementing pattern. A CAN interrupt handler is used to confirm message transmission and count the number of messages that have been sent.

This example sets up the CAN controller in External Loopback test mode. Data transmitted is visible on the CAN0TX pin and can be received with an appropriate mailbox configuration.

## 4.12 CAN External Loopback with Interrupts (can_loopback_interrupts)

This example shows the basic setup of CAN in order to transmit and receive messages on the CAN bus. The CAN peripheral is configured to transmit messages with a specific CAN ID. A message is then transmitted once per second, using a simple delay loop for timing. The message that is sent is a 4 byte message that contains an incrementing pattern. A CAN interrupt handler is used to confirm message transmission and count the number of messages that have been sent.

This example sets up the CAN controller in External Loopback test mode. Data transmitted is visible on the CAN0TX pin and can be received with an appropriate mailbox configuration.

This example uses the following interrupt handlers:

- INT_CAN0 - CANIntHandler

## 4.13 CLA 5 Tap Finite Impulse Response Filter (cla_adc_fir32_cpu01)

This example implements a 5 Tap FIR filter. It will setup EPWM1 to trigger ADCA at a frequency of 50KHz. Once the ADC completes sampling, it will trigger task 7 of the CLA which runs the filter on the ADC sample

EPWM2 is setup to switch at 10KHz. Connect pin EPWM2A to ADCA0 on the board to see the filtering effect

**Memory Allocation**

- CPU to CLA1 Message RAM
  - A - Filter Coefficients
- CLA1 to CPU Message RAM
  - voltFilt - Filtered sample
  - X - filter sample delay line

**Watch Variables**

- voltFilt - Filtered sample
- X - filter sample delay line

**External Connections**

- EPWM2A (GPIO2) to ADCA0

## 4.14 CLA $arcsine(x)$ using a lookup table (cla_asin_cpu01)

In this example, Task 1 of the CLA will calculate the arcsine of an input argument in the range (-1.0 to 1.0) using a lookup table

**Memory Allocation**

- CLA1 Math Tables (RAMLS0)
  - CLAasinTable - Lookup table
- CLA1 to CPU Message RAM
  - fResult - Result of the lookup algorithm
- CPU to CLA1 Message RAM
  - fVal - Sample input to the lookup algorithm

**Watch Variables**

- fVal - Argument to task 1
- fResult - Result of $arcsin(fVal)$

## 4.15 CLA $arctangent(x)$ using a lookup table (cla_atan_cpu01)

In this example, Task 1 of the CLA will calculate the arctangent of an input argument using a lookup table

## 4.16 CLA CRC8 Table-Lookup Algorithm (cla_crc8_cpu01)

This example implements a table lookup method of determining the 8-bit CRC of a message sequence. The polynomial used is 0x07

## 4.17 CLA CRC8 Table-generation Algorithm (cla_crc8table1_cpu01)

This example will generate the lookup table for an 8bit CRC checker with the polynomial 0x07

**Memory Allocation**

- CLA1 Data RAM 0(RAMLS0)
  - table - CRC Lookup table

**Watch Variables**

- table - Lookup table

## 4.18 CLA Determinant of 3X3 Matrix (cla_det_3by3_cpu01)

In this example, Task 1 of the CLA will calculate the determinant of a 3x3 matrix

## 4.19 CLA Division: Newton Raphson Approximation (cla_divide_cpu01)

In this example, Task 1 of the CLA will divide two input numbers using multiple approximations in the Newton Raphson method

**Memory Allocation**

- CLA1 to CPU Message RAM
  - Res - Result of the division operation
- CPU to CLA1 Message RAM
  - Num - Numerator of input
  - Den - Denominator of input

**Watch Variables**

- Num - Numerator of input
- Den - Denominator of input
- Res - Result of the division operation

## 4.20 CLA $10^X$ using a lookup table (cla_exp2_cpu01)

In this example, Task 1 of the CLA will calculate the Xth power of 10 using a table lookup method

**Memory Allocation**

- CLA1 Math Tables (RAMLS0)
  - CLAexpTable - Lookup table
- CLA1 to CPU Message RAM
  - ExpRes - Result of the exponentiation operation
- CPU to CLA1 Message RAM
  - Val - The exponent

**Watch Variables**

- Val - Input
- ExpRes - Result of $10^{Val}$

## 4.21 CLA $e^{\frac{A}{B}}$ using a lookup table (cla_exp2_cpu01)

In this example, Task 1 of the CLA will divide two input numbers using multiple approximations in the Newton Raphson method and then calculate the exponent of the result using a lookup table

**Memory Allocation**

- CLA1 Math Tables (RAMLS0)
  - CLAexpTable - Lookup table
- CLA1 to CPU Message RAM
  - ExpRes - Result of the exponentiation operation
- CPU to CLA1 Message RAM
  - Num - Numerator of input
  - Den - Denominator of input

**Watch Variables**

- Num - Numerator of input
- Den - Denominator of input
- ExpRes - Result of $e^{\frac{Num}{Den}}$

## 4.22 CLA 5 Tap Finite Impulse Response Filter (cla_fir32_cpu01)

This example implements a 5 Tap FIR filter. The input vector, stored in a lookup table, is filtered and then stored in an output buffer for storage.

**Memory Allocation**

- CLA1 Data RAM 0 (RAMLS0)
  - fCoeffs - Filter Coefficients
  - fDelayLine - Delay line memory elements
- CLA1 to CPU Message RAM
  - xResult - Result of the FIR operation
- CPU to CLA1 Message RAM
  - xAdcInput - Simulated ADC input

**Watch Variables**

- xResult - Result of the FIR operation
- xAdcInput - Simulated ADC input
- pass
- fail

## 4.23 CLA 2 Pole 2 Zero Infinite Impulse Response Filter (cla_iir2p2z_cpu01)

This example implements a Transposed Direct Form II IIR filter, commonly known as a Biquad. The input vector is a software simulated noisy signal that is fed to the biquad one sample at a time, filtered and then stored in an output buffer for storage.

**Memory Allocation**

- CLA1 Data RAM 1 (RAML2)
  - S1_A - Feedback coefficients
  - S1_B - Feedforward coefficients
- CLA1 to CPU Message RAM
  - yn - Output of the Biquad
- CPU to CLA1 Message RAM
  - xn - Sample input to the filter

**Watch Variables**

- fBiquadOutput
- pass
- fail

## 4.24 CLA Logic Test (cla_logic_cpu01)

In this example, Task 1 of the CLA implements a set of logic tests. More information about these logic statements can be found at: http://graphics.stanford.edu/∼seander/bithacks.html#OperationCounting

**Memory Allocation**

- CLA1 to CPU Message RAM
  - cla_pass_count - Logic test pass count
  - cla_fail_count - Logic test fail count

**Watch Variables**

- cla_pass_count - Logic test pass count
- cla_fail_count - Logic test fail count

## 4.25 CLA Matrix Multiplication (cla_matrix_mpy_cpu01)

In this example, Task 1 of the CLA multiplies two 3x3 matrices

**Memory Allocation**

- CLA1 to CPU Message RAM
  - z - Result of the matrix multiplication
- CPU to CLA1 Message RAM
  - x - 3X3 Input Matrix
  - y - 3X3 Input Matrix

**Watch Variables**

- x - 3X3 Input Matrix
- y - 3X3 Input Matrix
- z - Result of the matrix multiplication

## 4.26 CLA Matrix Transpose (cla_matrix_transpose_cpu01)

In this example, Task 1 of the CLA calculates the transpose of a 3x3 matrix

**Memory Allocation**

- CLA1 to CPU Message RAM
  - z - Transposed Matrix
- CPU to CLA1 Message RAM
  - x - 3X3 Input Matrix

**Watch Variables**

- x - 3X3 Input Matrix
- z - Transposed Matrix

## 4.27 CLA Mixed C and Assembly Code (cla_mixed_c_asm_cpu01)

This example shows the use of both C and assembly code on the CLA. The arc-cosine function uses a table lookup method and polynomial interpolation to determine the angle corresponding to the argument. The tables are stored in the CLA data ROM and can be accessed by

The tables needed by the acos routine are located in the CLA data ROM. A symbol table library is included with this example: c1bootROM_CLADataROMSymbols(_fpu32).lib The user must add this to the inclusion list in the upper window of the "File Search Path" options which can be found under properties->c2000 linker->File Search Path Since this library is present in the source directory, the user must also add the search path to the bottom window ${PROJECT_ROOT}/../

**Watch Variables**

- y1 - Accumulated results (angles in radians) from C routine
- y2 - Accumulated results (angles in radians) from asm routine

## 4.28 CLA Primes (cla_prime_cpu01)

In this example, Task 1 of the CLA calculates the set of prime numbers upto a length defined by the user

**Memory Allocation**

- CLA1 Data RAM 0 (RAMLS0)
  - out - Set of primes

**Watch Variables**

- out - Set of primes

# 4.29 CLA Shell Sort (cla_shellsort_cpu01)

In this example, Task 1 will perform the shell sort iteratively. Task 2 will do the same with mswapf intrinsic and Task 3 will also implement an in-place sort on an integer vector

**Memory Allocation**

- CLA1 Data RAM 1 (RAML2)
  - vector3 - Input/Output to task 3(in-place sorting)
- CLA1 to CPU Message RAM
  - vector1_sorted - Sorted output Task 1
  - vector2_sorted - Sorted output Task 2
- CPU to CLA1 Message RAM
  - vector1 - Input vector to task 1
  - vector2 - Input vector to task 2

**Watch Variables**

- vector3 - Input/Output to task 3(in-place sorting)
- vector1_sorted - Sorted output Task 1
- vector2_sorted - Sorted output Task 2
- vector1 - Input vector to task 1
- vector2 - Input vector to task 2

# 4.30 CLA Square Root (cla_sqrt_cpu01)

In this example, Task 1 calculates the square root of a number using multiple iterations of the Newton-Raphson approximation

**Memory Allocation**

- CLA1 to CPU Message RAM
  - fResult - $\sqrt{fVal}$
- CPU to CLA1 Message RAM
  - fVal - Input value

**Watch Variables**

- fVal - Input value
- fResult - $\sqrt{fVal}$

## 4.31 CLA Vector Inverse (cla_inverse_cpu01)

In this example, Task 1 calculates the element-wise inverse of a vector while Task 2 calculates the element-wise inverse of a vector and saves the result in the same vector

**Memory Allocation**

- CLA1 Data RAM 1 (RAML2)
  - vector2 - Input/Output vector for task 2
- CLA1 to CPU Message RAM
  - vector1_inverse - Inverse of input vector1
- CPU to CLA1 Message RAM
  - vector1 - Input vector to task 1

**Watch Variables**

- vector1 - Input vector to task 1
- vector1_inverse - Inverse of input vector1
- vector2 - Input/Output vector for task 2

## 4.32 CLA Vector Maximum (cla_vmaxfloat_cpu01)

Task 1 calculates the vector max moving backward through the array. Task 2 calculates the vector max moving forward through the array. Task 3 calculates the vector max using the ternary operator. Task 2 calculates the vector max using min/max intrinsics

**Memory Allocation**

- CLA1 to CPU Message RAM
  - max1 - Maximum value in vector 1
  - index1 - Index of the maximum value in vector 1
  - max2 - Maximum value in vector 2
  - index2 - Index of the maximum value in vector 2
  - max3 - Maximum value in vector 3
  - index3 - Index of the maximum value in vector 3
  - max4 - Maximum value in vector 4
  - min4 - Minimum value in vector 4
- CPU to CLA1 Message RAM
  - vector1 - Input vector to task 1
  - vector2 - Input vector to task 2
  - vector3 - Input vector to task 3
  - vector4 - Input vector to task 4
  - length1 - Length of vector 1
  - length2 - Length of vector 2

**Watch Variables**

- vector1 - Input vector to task 1
- vector2 - Input vector to task 2
- vector3 - Input vector to task 3
- vector4 - Input vector to task 4
- max1 - Maximum value in vector 1
- index1 - Index of the maximum value in vector 1
- max2 - Maximum value in vector 2
- index2 - Index of the maximum value in vector 2
- max3 - Maximum value in vector 3
- index3 - Index of the maximum value in vector 3
- max4 - Maximum value in vector 4
- min4 - Minimum value in vector 4

## 4.33 CLA Vector Minimum (cla_vminfloat_cpu01)

Task 1 calculates the vector min moving backward through the array. Task 2 calculates the vector min moving forward through the array. Task 3 calculates the vector min using the ternary operator.

**Memory Allocation**

- CLA1 to CPU Message RAM
  - min1 - Minimum value in vector 1
  - index1 - Index of the minimum value in vector 1
  - min2 - Minimum value in vector 2
  - index2 - Index of the minimum value in vector 2
  - min3 - Minimum value in vector 3
  - index3 - Index of the minimum value in vector 3
- CPU to CLA1 Message RAM
  - vector1 - Input vector to task 1
  - vector2 - Input vector to task 2
  - vector3 - Input vector to task 3
  - length1 - Length of vector 1
  - length2 - Length of vector 2
  - length3 - Length of vector 3

**Watch Variables**

- vector1 - Input vector to task 1
- vector2 - Input vector to task 2
- vector3 - Input vector to task 3
- min - Minimum value in vector 1
- index1 - Index of the minimum value in vector 1
- min2 - Minimum value in vector 2
- index2 - Index of the minimum value in vector 2
- min3 - Minimum value in vector 3
- index3 - Index of the minimum value in vector 3

## 4.34 CMPSS Asynchronous Trip

This example enables the CMPSS1 COMPH comparator and feeds the asynch CTRIPOUTH to GPIO14/XTRIPOUT3 pin and CTRIPH to GPIO15/EPWM8B

The COMPH inputs are: POS signal from CMPIN1P pin NEG signal from internal DACH

## 4.35 CMPSS Digital Filter

This example enables the CMPSS1 COMPH comparator and feeds the output through the digital filter to the GPIO14/XTRIPOUT3 pin.

The COMPH inputs are: POS signal from CMPIN1P pin NEG signal from internal DACH

## 4.36 CPU Timers

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

**Watch Variables**

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

## 4.37 ECAP APWM Example

This program sets up the eCAP pins in the APWM mode. This program runs at 200 MHz SYSCLK assuming a 20 MHz OSCCLK.

eCAP1 will come out on the GPIO5 pin This pin is configured to vary between frequencies using the shadow registers to load the next period/compare values

## 4.38 ECAP Capture PWM Example

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 GPIO19
- ePWM3A is on GPIO4
- Connect GPIO4 to GPIO19.

**Watch Variables**

- **ECap1PassCount** , Successful captures
- **ECap1IntCount** , Interrupt counts

# 4.39 EPWM dead band control (epwm_deadband)

During the test, monitor ePWM1, ePWM2, and/or ePWM3 outputs on a scope.

- ePWM1A is on GPIO0
- ePWM1B is on GPIO1
- ePWM2A is on GPIO2
- ePWM2B is on GPIO3
- ePWM3A is on GPIO4
- ePWM3B is on GPIO5

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.

View the EPWM1A/B, EPWM2A/B and EPWM3A/B waveforms via an oscilloscope

# 4.40 EPWM Action Qualifier (epwm_up_aq)

This example configures ePWM1, ePWM2, ePWM3 to produce an waveform with independant modulation on EPWMxA and EPWMxB.

The compare values CMPA and CMPB are modified within the ePWM's ISR.

The TB counter is in up count mode for this example.

View the EPWM1A/B(PA0_GPIO0 & PA1_GPIO1), EPWM2A/B(PA2_GPIO2 & PA3_GPIO3) and EPWM3A/B(PA4_GPIO4 & PA5_GPIO5) waveforms via an oscilloscope.

## 4.41 EPWM Action Qualifier (epwm_updown_aq)

This example configures ePWM1, ePWM2, ePWM3 to produce an waveform with independant 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.

View the EPWM1A/B(PA0_GPIO0 & PA1_GPIO1), EPWM2A/B(PA2_GPIO2 & PA3_GPIO3) and EPWM3A/B(PA4_GPIO4 & PA5_GPIO5) waveforms via an oscilloscope.

## 4.42 Frequency measurement using EQEP peripheral

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 64 edges 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

# 4.43 EQEP Speed and Position Measurement

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.

- GPIO20/eQEP1A <- GPIO0/ePWM1A (simulates eQEP Phase A signal)
- GPIO21/eQEP1B <- GPIO1/ePWM1B (simulates eQEP Phase B signal)
- GPIO23/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)

## 4.44 External Interrupts

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). The user is required to externally connect these signals for the program to work properly.

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

Connect GPIO30 to GPIO0. GPIO0 will be assigned to XINT1 Connect GPIO31 to GPIO1. GPIO1 will be assigned to XINT2

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

Watch Variables: Xint1Count for the number of times through XINT1 interrupt Xint2Count for the number of times through XINT2 interrupt LoopCount for the number of times through the idle loop

## 4.45 Device GPIO Setup

Configures the F2837xD 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

## 4.46 GPIO toggle test program

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

Toggle all of the GPIO PORT pins

The pins can be observed using Oscilloscope.

## 4.47 I2C EEPROM Example

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.

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

## 4.48 McBSP Loopback (mcbsp_loopback)

Three different serial word sizes can be tested. Before compiling this project, select the serial word size of 8, 16 or 32 by using the #define statements at the beginning of the code.

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.

This program will execute until terminated by the user.

**8-bit word example:**

The sent data looks like this:

00 01 02 03 04 05 06 07 .... FE FF

**16-bit word example:**

The sent data looks like this:

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

**32-bit word example:**

The sent data looks like this:

FFFF0000 FFFE0001 FFFD0002 .... 0000FFFF

**Watch Variables:**

- sdata1 - Sent data word: 8 or 16-bit or low half of 32-bit
- sdata2 - Sent data word: upper half of 32-bit
- rdata1 - Received data word: 8 or 16-bit or low half of 32-bit
- rdata2 - Received data word: upper half of 32-bit
- rdata1_point - Tracks last position in receive stream 1 for error checking
- rdata2_point - Tracks last position in receive stream 2 for error checking

**Note:**
sdata2 and rdata2 are not used for 8-bit or 16-bit word size

## 4.49 McBSP Loopback 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.

The sent data buffer will alternate between:

0000 0001 0002 0003 0004 0005 .... 007F

and

FFFF FFFE FFFD FFFC FFFB FFFA ....

Three different McBSP serial word sizes can be tested. Before compiling this project, select the serial word size of 8, 16 or 32 by using the #define statements at the beginning of the code.

This example uses DMA channel 1 and 2 interrupts. The incoming data is checked for accuracy. If an error is found the error() function is called and execution stops.

By default for the McBSP examples, the McBSP sample rate generator (SRG) input clock frequency is LSPCLK (80E6/4) assuming SYSCLKOUT = 80 MHz.

This example will execute until terminated by the user.

**Watch Variables:**

- sdata - Sent data buffer
- rdata - Received data buffer

## 4.50 McBSP Loopback with Interrupts (mcbsp_loopback_interrupts)

This program is a McBSP example that 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 80E6/4.

**Watch Variables:**

- sdata - Sent data word
- rdata - Received data word
- rdata_point - Tracks last position in receive stream for error checking

## 4.51 McBSP Loopback using SPI mode (mcbsp_spi_loopback)

This program will execute and transmit words until terminated by the user. SPI master mode transfer of 32-bit word size with digital loopback enabled.

**McBSP Signals** - **SPI equivalent**

- MCLKX - SPICLK (master)
- MFSX - SPISTE (master)
- MDX - SPISIMO
- MCLKR - SPICLK (slave - not used for this example)
- MFSR - SPISTE (slave - not used for this example)
- MDR - SPISOMI (not used for this example)

By default for the McBSP examples, the McBSP sample rate generator (SRG) input clock frequency is LSPCLK 80E6/4.

**Watch Variables:**

- sdata1 - Sent data word(1)
- sdata2 - Sent data word(2)
- rdata1 - Received data word(1)
- rdata2 - Received data word(2)

## 4.52 pbist_non_LS0_to_LS5

This example runs PBIST on all memories except for LS0 to LS5 since these memories are used for program/stack space

## 4.53 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 2806x 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.

**Note:**
If you are unable to open the .ht file, you can create a new one with the following settings

- Find correct COM port
- Bits per second = 9600
- Date Bits = 8
- Parity = None
- Stop Bits = 1
- Hardware Control = None

**Watch Variables**

- **LoopCount**, for the 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)

## 4.54 SCI FIFO Digital Loop Back Test

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

## 4.55 SCI Digital Loop Back with 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

01 02

02 03

....

FE FF

FF 00

etc..

The pattern is repeated forever.

**Watch Variables**

- **sdataA** , Data being sent
- **rdataA** , Data received
- **rdata_pointA** ,Keep track of where we are in the datastream. This is used to check the incoming data

# 4.56 SDFM Filter Sync CLA

In this example, SDFM filter data is read by CLA in Cla1Task1. The SDFM configuration is shown below:

- SDFM1 used in this example
- MODE0 Input control mode selected
- Comparator settings
  - Sinc3 filter selected
  - OSR = 32
  - HLT = 0x7FFF (Higher threshold setting)
  - LLT = 0x0000(Lower threshold setting)
- Sinc filter settings
  - All the 4 filter modules enabled
  - Sinc3 filter selected
  - OSR = 256
  - All the 4 filters are synchronized by using MFE (Master Filter enable bit)
- Integrator / demodulator settings
  - All the 4 integrator modules disabled
  - All the 4 demodulator modules disabled
  - The following settings don't have any bearings as the integrator module is disabled.
    - Sinc3 filter selected.
    - OSR = 32
  - Filter output represented in 16 bit format

    - Inorder to convert 25 bit Sinc filter / integrator output into 16 bit format user needs to right shift by 9 bits for Sinc3 filter with OSR = 256
- Interrupt module settings for SDFM filter
    - All the 4 higher threshold comparator interrupts disabled
    - All the 4 lower threshold comparator interrupts disabled
    - All the 4 modulator failure interrupts disabled
    - All the 4 filter will generate interrupt when a new filter data is available. **External Connections**

SDFM_PIN_MUX_OPTION1 Connect Sigma-Delta streams to (SD-D1, SD-C1 to SD-D8,SD-C8) on GPIO16-GPIO31

- SDFM_PIN_MUX_OPTION2 Connect Sigma-Delta streams to (SD-D1, SD-C1 to SD-D8,SD-C8) on GPIO48-GPIO63
- SDFM_PIN_MUX_OPTION3 Connect Sigma-Delta streams to (SD-D1, SD-C1 to SD-D8,SD-C8) on GPIO122-GPIO137

# 4.57 SDFM Filter Sync CPU

In this example, SDFM filter data is read by CPU in SDFM ISR routine. The SDFM configuration is shown below:

- SDFM used in this example - SDFM1
- Input control mode selected - MODE0
- Comparator settings
    - Sinc3 filter selected
    - OSR = 32
    - HLT = 0x7FFF (Higher threshold setting)
    - LLT = 0x0000(Lower threshold setting)
- Sinc filter settings
    - All the 4 filter modules enabled
    - Sinc3 filter selected
    - OSR = 256
    - All the 4 filters are synchronized by using MFE (Master Filter enable bit)
- Integrator / demodulator settings
    - All the 4 integrator modules disabled
    - All the 4 demodulator modules disabled
    - The following settings donŠt have any bearings as the integrator module is disabled.
        * Sinc3 filter selected.
        * OSR = 32
    - Filter output represented in 16 bit format
    - Inorder to convert 25 bit Sinc filter / integrator output into 16 bit format user needs to right shift by 9 bits for Sinc3 filter with OSR = 256

Interrupt module settings for SDFM filter

- All the 4 higher threshold comparator interrupts disabled
- All the 4 lower threshold comparator interrupts disabled
- All the 4 modulator failure interrupts disabled
- All the 4 filter will generate interrupt when a new filter data is available.

# 4.58 SDFM Filter Sync DMA

In this example, SDFM filter data is read by DMA. The SDFM configuration is shown below:

- SDFM1 used in this example
- MODE0 Input control mode selected
- Comparator settings
  - Sinc3 filter selected
  - OSR = 32
  - HLT = 0x7FFF (Higher threshold setting)
  - LLT = 0x0000(Lower threshold setting)
- Sinc filter settings
  - All the 4 filter modules enabled
  - Sinc3 filter selected
  - OSR = 256
  - All the 4 filters are synchronized by using MFE (Master Filter enable bit)
- Integrator / demodulator settings
  - All the 4 integrator modules disabled
  - All the 4 demodulator modules disabled
  - The following settings doesnŠt have any bearings as the integrator module is disabled.
    - Sinc3 filter selected.
    - OSR = 32
  - Filter output represented in 16 bit format
  - Inorder to convert 25 bit Sinc filter / integrator output into 16 bit format user needs to right shift by 9 bits for Sinc3 filter with OSR = 256
- Interrupt module settings for SDFM filter
  - All the 4 higher threshold comparator interrupts disabled
  - All the 4 lower threshold comparator interrupts disabled
  - All the 4 modulator failure interrupts disabled
  - All the 4 filter will generate interrupt when a new filter data is available

# 4.59 SDFM PWM Sync

In this example, SDFM filter data is read by CPU in SDFM ISR routine.The SDFM configuration is shown below:

- SDFM1 is used in this example

- MODE0 Input control mode selected
- Comparator settings
  - Sinc3 filter selected
  - OSR = 32
  - HLT = 0x7FFF (Higher threshold setting)
  - LLT = 0x0000(Lower threshold setting)
- Sinc filter settings
  - All the 4 filter modules enabled
  - Sinc3 filter selected
  - OSR = 256
  - All the 4 filters are synchronized by using PWM11.CMPC and CMPD synchronized signals instead of MFE bit
- Integrator / demodulator settings
  - All the 4 integrator modules disabled
  - All the 4 demodulator modules disabled
  - The following settings doesnŠt have any bearings as the integrator module is disabled.
  - Sinc3 filter selected.
    - OSR = 32
    - Filter output represented in 16 bit format
  - Inorder to convert 25 bit Sinc filter / integrator output into 16 bit format user needs to right shift by 9 bits for Sinc3 filter with OSR = 256
- Interrupt module settings for SDFM filter
  - All the 4 higher threshold comparator interrupts disabled
  - All the 4 lower threshold comparator interrupts disabled
  - All the 4 modulator failure interrupts disabled
  - All the 4 filter will generate interrupt when a new filter data is available

# 4.60 Setup CPU01

This example gives control of all shared GPIOs and peripherals to CPU02

# 4.61 SPI Digital Loop Back

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

## 4.62 SPI Digital Loop Back with 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

0001 0002

0002 0003

....

FFFE FFFF

FFFF 0000

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

## 4.63 LED Blink Getting Started Program

This example configures CPU Timer0 for a 500 msec period, and toggles the GPIO34 LED 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 GPIO34 LED blink on (for 500 msec) and off (for 500 msec) on the F2837xD control card.

## 4.64 Profiling $sine(x)$ using the TMU

In this example, we will use TMU intrinsics to calculate the sine for a series of per-unit arguments (the argument is not represented in radians, it is normalized to the range -1.0 to 1.0). We will profile the execution time of the TMU versus the conventional implementation in the run-time support library

$\forall x \in [-2\pi, 2\pi], x_{pu} = \frac{x}{2\pi}\ y = sin(x_{pu} * 2\pi)$

Instead of using intrinsics, the compiler can implement most of the RTS trigonometric functions through TMU instructions if the option *fp_mode* is set to *relaxed*. In this example, this option is left untouched; it defaults to the *strict* mode.

**Watch Variables**

- timeRTS - time to run RTS routine
- timeTMU - time to run TMU routine

## 4.65 USB Generic Bulk Device (usb_dev_bulk)

This example provides a generic USB device offering simple bulk data transfer to and from the host. The device uses a vendor-specific class ID and supports a single bulk IN endpoint and a single bulk OUT endpoint. Data received from the host is assumed to be ASCII text and it is echoed back with the case of all alphabetic characters swapped.

UART0, connected to the FTDI virtual COM port and running at 115,200, 8-N-1, is used to display messages from this application.

A Windows INF file for the device is provided in ControlSUITE. This INF contains information required to install the WinUSB subsystem on WindowsXP and Windows 7. WinUSB is a Windows subsystem allowing user mode applications to access the USB device without the need for a vendor-specific kernel mode driver.

A sample Windows command-line application, usb_bulk_example, illustrating how to connect to and communicate with the bulk device is also provided. Project files are included to allow the examples to be built using Microsoft VisualStudio. Source code for this application can be found in directory F2837xD_common/tools/usb_bulk_example/Release.

## 4.66 USB HID Keyboard Device (usb_dev_keyboard)

This example application turns the evaluation board into a USB keyboard supporting the Human Interface Device class. When GPIO0 is pulled high, a sequence of key presses is simulated to type a string. Care should be taken to ensure that the active window can safely receive the text; enter is not pressed at any point so no actions are attempted by the host if a terminal window is used (for example). The LED2 is used to indicate the current Caps Lock state and is updated in response to any other keyboard attached to the same USB host system.

The device implemented by this application also supports USB remote wakeup allowing it to request the host to reactivate a suspended bus. If the bus is suspended (as indicated on the application display), toggling GPIO0 will request a remote wakeup assuming the host has not specifically disabled such requests.

To run the example compile the project, load to the target, and run the example. After the example is running, connect a USB cable from the PC to the microUSB port on the controlCARD. Then toggle GPIO0 while the PC's window focus is in a window that can recieve keyboard input (i.e. NotePad).

## 4.67 USB HID Mouse Device (usb_dev_mouse)

This example application turns the evaluation board into a USB mouse supporting the Human Interface Device class. After loading and running the example simply connect the PC to the controlCARDs microUSB port using a usb cable, and the mouse pointer will move in a square pattern for the duration of the time it is plugged in.

UART0, connected to the FTDI virtual COM port and running at 115,200, 8-N-1, is used to display messages from this application.

## 4.68 USB MSC Device (usb_dev_msc)

This example application turns the evaluation board into a USB mass storage class device. The application will use the microSD card for the storage media for the mass storage device. The screen will display the current action occurring on the device ranging from disconnected, no media, reading, writing and idle. To get started simply connect your PC to the controlCARD's microUSB port on top of the card using a USB cable.

## 4.69 USB Serial Device (usb_dev_serial)

This example application turns the evaluation kit into a virtual serial port when connected to the USB host system. The application supports the USB Communication Device Class, Abstract Control Model to redirect UART0 traffic to and from the USB host system.

Connect USB cables from your PC to both the mini and microUSB connectors on the controlCARD.Figure out what COM ports your controlCARD is enumerating (typically done using Device Manager in Windows) and open a serial terminal to each of with the settings 115200 Baud 8-N-1. Characters typed in one terminal should be echoed in the other and vice versa.

Assuming you installed controlSUITE in the default directory, a driver information (INF) file for use with Windows XP and Windows7 can be found in C:/ti/controlSUITE/F2837xD/VERSION/F2837xD_common/windows_drivers.

## 4.70 USB HID Mouse Host (usb_host_mouse)

This application demonstrates the handling of a USB mouse attached to the evaluation kit. Once attached, the position of the mouse pointer and the state of the mouse buttons are output to the display.

The first UART, which is connected to the FTDI virtual serial port on the controlCARD board, is configured for 115,200 bits per second, and 8-N-1 mode. When a HID compliant mouse is connected to the microUSB port on the top of the controlCARD, position and button information will be displayed to the console.

## 4.71 USB Mass Storage Class Host (usb_host_msc)

This example application demonstrates reading a file system from a USB mass storage class device. It makes use of FatFs, a FAT file system driver. It provides a simple command console via the UART for issuing commands to view and navigate the file system on the mass storage device.

The first UART, which is connected to the FTDI virtual serial port on the controlCARD board, is configured for 115,200 bits per second, and 8-N-1 mode. When the program is started a message will be printed to the terminal. Type “help” for command help.

After loading and running the example, open a serial terminal with the above settings to open the command prompt. Then connect a USB MSC device to the microUSB port on the top of the controlCARD.

For additional details about FatFs, see the following site: http://elm-chan.org/fsw/ff/00index_e.html

## 4.72 Watchdog

This example shows how to service the watchdog or generate a wakeup interrupt using the watchdog. By default the example will generate a Wake interrupt. To service the watchdog and not generate the interrupt uncomment the ServiceDog() line the the main for loop.

# 5 Dual Core Example Applications

These example applications show how to make use of F2837xD device functions which span both the CPU 1 and CPU 2. All of these examples contain two example projects: one for CPU 1 and one for CPU 2.

Like the CPU1 only projects, these projects also contain different build configurations for RAM and Flash builds. All of the CPU1 projects contain RAM and Flash build configurations with debugger support, as well as a standalone flash build configuration which sends an IPC command to boot the second core and begin executing the application in its flash. The CPU2 projects all only contain a flash and RAM build configuration as there are no dependencies in the code regarding whether the application is running with or without a debugger.

To run one of these examples after compiling it, load the appropriate programs on each of the two cores. Then, for more example specific intructions please refer to the documentation regarding the example you wish to run on the following pages or in the comments of the example sources.

All of these examples can be found in the

`device_support/F2837xD/<Version>/F2837xD_examples_Dual` subdirectory of the ControlSUITE package.

## 5.1 ADC & EPWM on CPU2

This example demonstrates how to make use of the ADC and EPWM peripherals from CPU2. Device clocking (PLL) and GPIO setup are done using CPU1, while all other configuration of the peripherals is done using CPU2.

CPU2 configures EPWM1 in up count mode in a similar fashion to what is done in the epwm_up_aq example. The ADC is configured in continuous conversion mode similar to the adc_soc_continuous example. GPIO0 can be connected to ADCINA0 and the results buffer AdcaResults graphed in CCS to view the duty cycle of the generated waveform.

## 5.2 Blinky

Dual Core Blinky Example. This example demonstrates how to implement and run a standalone application on both cores.

## 5.3 CLA $arcsine(x)$ using a lookup table (cla_asin_cpu01)

In this example, cpu1 will be used to inialize the clocks for cpu2.cla1. Task 1 of the CLA on cpu2 will calculate the arcsine of an input argument in the range (-1.0 to 1.0) using a lookup table

**Memory Allocation**

- CLA1 Math Tables (RAMLS0)
  - CLAasinTable - Lookup table

- CLA1 to CPU Message RAM
  - fResult - Result of the lookup algorithm
- CPU to CLA1 Message RAM
  - fVal - Sample input to the lookup algorithm

**Watch Variables**

- fVal - Argument to task 1
- fResult - Result of $arcsin(fVal)$

**Note:**
CPU2 must turn on the CLA clock by writing a 1 to CpuSysRegs.PCLKCR0.bit.CLA1.

## 5.4 CLA 2 Pole 2 Zero Infinite Impulse Response Filter (cla_iir2p2z_cpu01)

This example implements a Transposed Direct Form II IIR filter, commonly known as a Biquad. The input vector is a software simulated noisy signal that is fed to the biquad one sample at a time, filtered and then stored in an output buffer for storage.

**Memory Allocation**

- CLA1 Data RAM 1 (RAML2)
  - S1_A - Feedback coefficients
  - S1_B - Feedforward coefficients
- CLA1 to CPU Message RAM
  - yn - Output of the Biquad
- CPU to CLA1 Message RAM
  - xn - Sample input to the filter

**Watch Variables**

- fBiquadOutput
- pass
- fail

**Note:**
CPU2 must turn on the CLA clock by writing a 1 to CpuSysRegs.PCLKCR0.bit.CLA1.

## 5.5 CPU01 to CPU02 IPC Driver

This example tests all of the basic read/write CPU01 to CPU02 IPC Driver functions available in F2837xD_Ipc_Driver.c. The CPU01 project sends commands to the CPU02 project, which then processes the commands. The CPU02 project responds to the commands sent from the CPU01 project. Note that IPC INT1 and IPC INT2 are used for this example to process IPC commands.

**Watch Variables for CPU01** :

- ErrorCount - Counts # of errors
- pusCPU01BufferPt - Stores 256 16-bit words block to write to CPU02
- pusCPU02BufferPt - Points to beginning of 256 word block received back from CPU02
- usWWord16 - 16-bit word to write to CPU02
- ulWWord32 - 32-bit word to write to CPU02
- usRWord16 - 16-bit word to read from CPU02
- ulRWord32 - 32-bit word to read from CPU02

**Watch Variables for CPU02** :

- ErrorFlag - Indicates an unrecognized command was sent from CPU01 to CPU02.

## 5.6 CPU01 to CPU02 IPC Lite Drivers (cpu01_to_cpu2_ipcdrivers_lite)

This example application demonstrates the use of the CPU01 to CPU02 IPC Lite Driver Functions which allow the CPU01 to read/write to addresses on the CPU02. CPU02 to CPU01 MSG RAM is used to pass the addresses of local variables between the processors.

**Watch Variables** on CPU01:

- ErrorCount - Counts # of errors
- usWWord16 - 16-bit word to write to CPU02
- ulWWord32 - 32-bit word to write to CPU02
- usRWord16 - 16-bit word to read from CPU02
- ulRWord32 - 32-bit word to read from CPU02

**Watch Variables** on CPU02:

- ErrorFlag - Indicates an unrecognized command was sent from CPU01 to CPU02.

## 5.7 CPU01 to CPU02 IPC Write Protect Driver

This example tests all of the basic read/write CPU01 to CPU02 IPC Write Protect Driver functions available in F2837xD_Ipc_Driver.c. The CPU01 project sends commands to the CPU02 project, which then processes the commands. The CPU02 project responds to the commands sent from the CPU01 project. Note that IPC INT1 and IPC INT2 are used for this example to process IPC commands.

**Watch Variables for CPU01** :

- ErrorCount - Counts # of errors
- ulCPU01Buffer - Stores 4 32-bit words block to write to CPU02
- pulCPU01BufferPt - Points to beginning of 256 word block received back from CPU02

- usWWord16 - 16-bit word to write to CPU02
- ulWWord32 - 32-bit word to write to CPU02
- usRWord16 - 16-bit word to read from CPU02
- ulRWord32 - 32-bit word to read from CPU02

**Watch Variables for CPU02** :

- ErrorFlag - Indicates an unrecognized command was sent from CPU01 to CPU02.

## 5.8 CPU02 to CPU01 IPC Driver

This example tests all of the basic read/write CPU02 to CPU01 IPC Driver functions available in F2837xD_Ipc_Driver.c. The CPU02 project sends commands to the CPU01 project, which then processes the commands. The CPU01 project responds to the commands sent from the CPU02 project. Note that IPC INT1 and IPC INT2 are used for this example to process IPC commands.

**Watch Variables for CPU02** :

- ErrorCount - Counts # of errors
- usCPU02Buffer - Stores 256 16-bit words block to write to CPU01
- pusCPU01BufferPt - Points to beginning of 256 word block received back from CPU01
- usWWord16 - 16-bit word to write to CPU01
- ulWWord32 - 32-bit word to write to CPU01
- usRWord16 - 16-bit word to read from CPU01
- ulRWord32 - 32-bit word to read from CPU01

**Watch Variables for CPU01** :

- ErrorFlag - Indicates an unrecognized command was sent from CPU02 to CPU01.

## 5.9 CPU02 to CPU01 IPC Lite Drivers (cpu02_to_cpu1_ipcdrivers_lite)

This example application demonstrates the use of CPU02 to CPU01 IPC Lite Driver Functions which allow the CPU02 to read/write to addresses on the CPU01. CPU01toCPU02 MSG RAM is used to pass the addresses of local variables between the processors.

**Watch Variables for CPU02:**

- ErrorCount - Counts # of errors
- usWWord16 - 16-bit word to write to CPU01
- ulWWord32 - 32-bit word to write to CPU01
- usRWord16 - 16-bit word to read from CPU01
- ulRWord32 - 32-bit word to read from CPU01

**Watch Variables for CPU01** :

- ErrorFlag - Indicates an unrecognized command was sent from CPU02 to CPU01.

## 5.10 CPU02 to CPU01 IPC Write Protect Driver

This example tests all of the basic read/write CPU02 to CPU01 IPC Write Protect Driver functions available in F2837xD_Ipc_Driver.c. The CPU02 project sends commands to the CPU01 project, which then processes the commands. The CPU01 project responds to the commands sent from the CPU02 project. Note that IPC INT1 and IPC INT2 are used for this example to process IPC commands.

**Watch Variables for CPU02** :

- ErrorCount - Counts # of errors
- ulCPU02Buffer - Stores 4 32-bit words block to write to CPU01
- pulCPU01BufferPt - Points to beginning of 256 word block received back from CPU01
- usWWord16 - 16-bit word to write to CPU01
- ulWWord32 - 32-bit word to write to CPU01
- usRWord16 - 16-bit word to read from CPU01
- ulRWord32 - 32-bit word to read from CPU01

**Watch Variables for CPU01** :

- ErrorFlag - Indicates an unrecognized command was sent from CPU01 to CPU02.

## 5.11 IPC GPIO toggle

This example shows GPIO input on the local CPU triggering an output on the remote CPU. A GPIO input change on CPU01 causes an output change on CPU02 and vice versa. CPU1 has control of GPIO31 , GPIO15 and GPIO14. CPU2 has control of GPIO34 , GPIO12 and GPIO11.

**Hardware Connections**

- connect GPIO15 to GPIO11
- connect GPIO14 to GPIO12

**Watch Pins**

- GPIO31 - output on CPU2 (LED blinking if using control card)
- GPIO11 - input on CPU2
- GPIO34 - output on CPU1 (LED blinking if using control card)
- GPIO14 - input on CPU1
- GPIO12 - square wave output on CPU02
- GPIO15 - square wave output on CPU01

# 5.12 Shared RAM management (RAM_management)

This example shows how to assign shared RAM for use by both the CPU02 and CPU01 core. Shared RAM regions are defined in both the CPU02 and CPU01 linker files. In this example GS0 and GS14 are assigned to/owned by CPU02. The remaining shared RAM regions are owned by CPU01. In this example

A pattern is written to c1_r_w_array and then IPC flag is sent to notify CPU02 that data is ready to be read. CPU02 then reads the data from c2_r_array and writes a modified pattern to c2_r_w_array. Once CPU02 acknowledges the IPC flag to , CPU01 reads the data from c1_r_array and compares with expected result.

A Timed ISR is also serviced in both CPUs. The ISRs are copied into the shared RAM region owned by the respective CPUs. Each ISR toggles a GPIO. Watch GPIO31 and GPIO34 on oscilloscope. If using the control card watch LED1 and LED2 bliink at different rates.

- c1_r_w_array[] is mapped to shared RAM GS1
- c1_r_array[] is mapped to shared RAM GS0
- c2_r_array[] is mapped to shared RAM GS1
- c2_r_w_array[] is mapped to shared RAM GS0
- cpu_timer0_isr in CPU02 is copied to shared RAM GS14 , toggles GPIO31
- cpu_timer0_isr in CPU01 is copied to shared RAM GS15 , toggles GPIO34

**Watch Variables**

- error Indicates that the data written is not correctly received by the other CPU.

# 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