Important Notice

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI’s standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer’s applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI’s publication of information regarding any third party’s products or services does not constitute TI’s approval, warranty or endorsement thereof.

Copyright © 2003 – 2009 Texas Instruments Incorporated

Revision History

February 2003 – Revision 1.0
March 2003 – Revision 1.1
July 2004 – Revision 2.0
August 2005 – Revision 3.0
October 2005 – Revision 3.1
April 2007 – Revision 4.0
June 2008 – Revision 5.0
November 2008 – Revision 5.1
January 2009 – Revision 5.2

Mailing Address

Texas Instruments
Training Technical Organization
7839 Churchill Way
M/S 3984
Dallas, Texas 75251-1903
Workshop Topics

Workshop Topics................................................................................................................ .........................3
Workshop Introduction ...............................................................................................................................4
Architecture Overview .................................................................................................................................7
Programming Development Environment ..................................................................................................................10
  Code Composer Studio ................................................................................................................. .................10
  Linking Sections in Memory ................................................................................................................12
Lab 1: Linker Command File .......................................................................................................................15
Peripheral Register Header Files ....................................................................................................................19
Reset, Interrupts and System Initialization ........................................................................................................26
  Reset .....................................................................................................................................................26
  Interrupts ..............................................................................................................................................28
  Peripheral Interrupt Expansion (PIE) ...................................................................................................29
  Oscillator / PLL Clock Module ............................................................................................................31
  Watchdog Timer Module .......................................................................................................................32
  GPIO .....................................................................................................................................................33
Lab 2: System Initialization .........................................................................................................................35
Control Peripherals ..................................................................................................................................39
  ADC Module ........................................................................................................................................39
  Pulse Width Modulation .........................................................................................................................40
  ePWM ..................................................................................................................................................41
  eCAP ..................................................................................................................................................52
  eQEP ..................................................................................................................................................54
Lab 3: Control Peripherals ..........................................................................................................................56
Flash Programming ..................................................................................................................................62
  Flash Programming Basics ...................................................................................................................62
  Programming Utilities and CCS Plug-in ...............................................................................................63
  Code Security Module and Password .................................................................................................64
Lab 4: Programming the Flash ....................................................................................................................66
The Next Step ..............................................................................................................................................73
  Training ................................................................................................................................................73
  Development Tools .............................................................................................................................74
  Development Support ..........................................................................................................................77
Workshop Introduction

C28x 1-Day Workshop Outline

- Workshop Introduction
- Architecture Overview
- Programming Development Environment
  - Lab: Linker command file
- Peripheral Register Header Files
- Reset, Interrupts and System Initialization
  - Lab: Watchdog and interrupts
- Control Peripherals
  - Lab: Generate and graph a PWM waveform
- Flash Programming
  - Lab: Run the code from flash memory
- The Next Step…
Introductions

- Name
- Company
- Project Responsibilities
- DSP / Microcontroller Experience
- TMS320 DSP Experience
- Hardware / Software - Assembly / C
- Interests

C2000 Portfolio Expanding with Price/Performance Optimized Derivatives

- High-Precision Control
  - F2834x/24x up to 300 MIPS
  - F2833x/23x 150 MIPS
  - F281x 150 MIPS

- High-end Derivatives
  - F28x xx 100 MIPS

- Cost optimized versions
  - 24x™ up to 40 MIPS
  - F2803x/2x up to 60 MIPS
Workshop Introduction

**Broad C28x™ Application Base**

- **Solar Inverters**
- **Optical Networking**
  - Control of laser diode
- **Digital Power Supply**
  - Provides control, sensing, PFC, and other functions
- **Other Segments**
  - eg. Musical Instruments, HDTV/Displays
- **Automotive**
- **Medical**
- **Industrial Motor Control**
- **Printer**
  - Print head control
  - Paper path motor control
- **Non-traditional Motor Control**
  - Many new cool applications to come

**High Performance Controllers**

<table>
<thead>
<tr>
<th></th>
<th>MHz</th>
<th>FPU</th>
<th>Flash</th>
<th>RAM</th>
<th>DMA</th>
<th>PWM/HRPWM</th>
<th>CAP/QEP</th>
<th>Communication Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>F28335</td>
<td>150</td>
<td>Yes</td>
<td>256</td>
<td>34</td>
<td>Yes</td>
<td>18/6</td>
<td>6/2</td>
<td>SPI, 3x SCI, 1°C, 2x McBSP, 2x CAN</td>
</tr>
<tr>
<td>F28334</td>
<td>150</td>
<td>Yes</td>
<td>128</td>
<td>34</td>
<td>Yes</td>
<td>18/6</td>
<td>4/2</td>
<td>SPI, 3x SCI, 1°C, 2x McBSP, 2x CAN</td>
</tr>
<tr>
<td>F28332</td>
<td>100</td>
<td>Yes</td>
<td>64</td>
<td>26</td>
<td>Yes</td>
<td>16/4</td>
<td>4/2</td>
<td>SPI, 2x SCI, 1°C, McBSP, 2x CAN</td>
</tr>
<tr>
<td>F28325</td>
<td>150</td>
<td>No</td>
<td>256</td>
<td>34</td>
<td>Yes</td>
<td>18/6</td>
<td>6/2</td>
<td>SPI, 3x SCI, 1°C, 2x McBSP, 2x CAN</td>
</tr>
<tr>
<td>F28324</td>
<td>150</td>
<td>No</td>
<td>128</td>
<td>34</td>
<td>Yes</td>
<td>18/6</td>
<td>4/2</td>
<td>SPI, 3x SCI, 1°C, 2x McBSP, 2x CAN</td>
</tr>
<tr>
<td>F28322</td>
<td>100</td>
<td>No</td>
<td>64</td>
<td>26</td>
<td>Yes</td>
<td>16/4</td>
<td>4/2</td>
<td>SPI, 2x SCI, 1°C, McBSP, 2x CAN</td>
</tr>
</tbody>
</table>

- All devices above are 100% pin-compatible and 100% Software compatible
- All devices have 16/32-bit EMIF, 16 channel ADC at 12.5 MSPS, and 88 GPIO

*For details and information on other C28x family members refer to the "DSP Selection Guide" and specific "Data Manuals"*
Architecture Overview

C28x Block Diagram

TMS320F28335 Memory Map
C28x Fast Interrupt Response Manager

- 96 dedicated PIE vectors
- No software decision making required
- Direct access to RAM vectors
- Auto flags update
- Concurrent auto context save

Auto Context Save

<table>
<thead>
<tr>
<th>T</th>
<th>ST0</th>
</tr>
</thead>
<tbody>
<tr>
<td>AH</td>
<td>AL</td>
</tr>
<tr>
<td>PH</td>
<td>PL</td>
</tr>
<tr>
<td>AR1 (L)</td>
<td>AR0 (L)</td>
</tr>
<tr>
<td>DP</td>
<td>ST1</td>
</tr>
<tr>
<td>DBSTAT</td>
<td>IER</td>
</tr>
<tr>
<td>PC(msw)</td>
<td>PC(lsw)</td>
</tr>
</tbody>
</table>

Reset – Bootloader

Reset

OBJMODE = 0  AMODE = 0
ENPIE = 0  INTM = 1

Bootloader sets
OBJMODE = 1  AMODE = 0
Boot determined by state of GPIO pins

Reset vector fetched from boot ROM
0x3F FFC0

Note:
Details of the various boot options will be discussed in the Reset and Interrupts module
Architecture Overview

eZdsp™ F28335 Hardware

- JTAG Interface (P1)
- Power Connector (P6) +5V
- LED (DS1) +5V
- USB JTAG Controller Interface (J201)
- 30 MHz Clock
- Bootloader GPIO Pins
- I/O Interface (P4/P8/P7)
- Expansion (P2/P10)
- LED (DS2) GPIO32
- SCI-A (P12)
- SCI-B (J12)
- SCI-A (P11)
- eCAN-A (P11)
- eCAN-B (J11)
- On-Chip: 34Kw RAM
  256Kw Flash
  1Kw OTP
- Analog Interface (P5/P9)
- TMS320F28335 150 MIPS
- 34Kw RAM
- 256Kw Flash
- 1Kw OTP
- TMS320F28335 One-Day Workshop
Programming Development Environment

Code Composer Studio

Code Composer Studio: IDE

- Integrates: edit, code generation, and debug
- Single-click access using buttons
- Powerful graphing/profiling tools
- Automated tasks using GEL scripts and CCS scripting
- Built-in access to BIOS functions
- Supports TI and 3rd party plug-ins

The CCS Project

Project (.pjt) files contain:

- List of files:
  - Source (C, assembly)
  - Libraries
  - DSP/BIOS configuration file
  - Linker command files
- Project settings:
  - Build options (compiler, Linker, assembler, and DSP/BIOS)
  - Build configurations
Build Options GUI - Compiler

- GUI has 8 pages of categories for code generation tools
- Controls many aspects of the build process, such as:
  - Optimization level
  - Target device
  - Compiler/assembly/link options

Build Options GUI - Linker

- GUI has 3 categories for linking
  - Specify various link options
- `.Debug` means the directory called Debug one level below the `.pjt` file directory
- `${Proj_dir}\Debug` is an equivalent expression
Linking Sections in Memory

Sections

- All code consists of different parts called sections
- All default section names begin with "." 
- The compiler has default section names for initialized and uninitialized sections

```
int x = 2;
int y = 7;

void main(void)
{
    long z;
    z = x + y;
}
```

Global vars (.ebss)  Init values (.cinit)

Local vars (.stack)  Code (.text)

Compiler Section Names

### Initialized Sections

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Link Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td>code</td>
<td>FLASH</td>
</tr>
<tr>
<td>.cinit</td>
<td>initialization values for global and static variables</td>
<td>FLASH</td>
</tr>
<tr>
<td>.econst</td>
<td>constants (e.g. const int k = 3;)</td>
<td>FLASH</td>
</tr>
<tr>
<td>.switch</td>
<td>tables for switch statements</td>
<td>FLASH</td>
</tr>
<tr>
<td>.pinit</td>
<td>tables for global constructors (C++)</td>
<td>FLASH</td>
</tr>
</tbody>
</table>

### Uninitialized Sections

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Link Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>.ebss</td>
<td>global and static variables</td>
<td>RAM</td>
</tr>
<tr>
<td>.stack</td>
<td>stack space</td>
<td>low 64Kw RAM</td>
</tr>
<tr>
<td>.esysmem</td>
<td>memory for far malloc functions</td>
<td>RAM</td>
</tr>
</tbody>
</table>

Note: During development initialized sections could be linked to RAM since the emulator can be used to load the RAM
Programming Development Environment

Placing Sections in Memory

Memory

0x00 0000
M0SARAM (0x400)

0x00 0400
M1SARAM (0x400)

0x30 0000
FLASH (0x40000)

Sections

.ebss

.stack

.cinit

.text

Linking

- Memory description
- How to place s/w into h/w

Link.cmd

.obj → Linker → .out

.map
Linker Command File

MEMORY
{
    PAGE 0: /* Program Memory */
        FLASH: origin = 0x300000, length = 0x40000

    PAGE 1: /* Data Memory */
        M0SARAM: origin = 0x000000, length = 0x400
        M1SARAM: origin = 0x000400, length = 0x400
}
SECTIONS
{
    .text:> FLASH PAGE = 0
    .ebss:> M0SARAM PAGE = 1
    .cinit:> FLASH PAGE = 0
    .stack:> M1SARAM PAGE = 1
}
Lab 1: Linker Command File

Objective

Use a linker command file to link the C program file (Lab1.c) into the system described below.

System Description:
- TMS320F28335
- All internal RAM blocks allocated

Placement of Sections:
- .text into RAM Block L0123SARAM on PAGE 0 (program memory)
- .cinit into RAM Block L0123SARAM on PAGE 0 (program memory)
- .ebss into RAM Block L4SARAM on PAGE 1 (data memory)
- .stack into RAM Block M1SARAM on PAGE 1 (data memory)

System Description
- TMS320F28335
- All internal RAM blocks allocated

Placement of Sections:
- .text into RAM Block L0123SARAM on PAGE 0 (program memory)
- .cinit into RAM Block L0123SARAM on PAGE 0 (program memory)
- .ebss into RAM Block L4SARAM on PAGE 1 (data memory)
- .stack into RAM Block M1SARAM on PAGE 1 (data memory)

Procedure

Open a Project

1. Double click on the Code Composer Studio icon on the desktop. Maximize Code Composer Studio to fill your screen. Code Composer Studio has a Connect/Disconnect feature which allows the target to be dynamically connected and disconnected. This will reset the JTAG link and also enable “hot swapping” a target board. Connect to the target.
Click: Debug ➔ Connect

The menu bar (at the top) lists File ... Help. Note the horizontal tool bar below the menu bar and the vertical tool bar on the left-hand side. The window on the left is the project window and the large right-hand window is your workspace.

2. A project is all the files you will need to develop an executable output file (.out) which can be run on the DSP hardware. A project named Lab1.pjt has been created for this lab. Open the project by clicking:

Project ➔ Open...

and look in C:\C28x\LABS\LAB1. This .pjt file will invoke all the necessary tools (compiler, assembler, linker) to build the project. It will also create a debug folder that will hold immediate output files.

3. In the project window on the left, click the plus sign (+) to the left of Project. Now, click on the plus sign next to Lab1.pjt. Notice that the Lab1.cmd file is listed. Click on Source to see the current source file list (i.e. Lab1.c).

4. A test file named Lab1.c has been added to the project. This file will be used in this exercise to demonstrate some features of Code Composer Studio.

**Project Build Options**

5. There are numerous build options in the project. The default option settings are sufficient for getting started. We will inspect a couple of the default linker options at this time.

Click: Project ➔ Build Options...

6. Select the Linker tab. Notice that .out and .map files are being created. The .out file is the executable code that will be loaded into the DSP. The .map file will contain a linker report showing memory usage and section addresses in memory. The Stack Size has been set to 0x200.

7. Select OK and the Build Options window will close.

**Linker Command File – Lab1.cmd**

8. Open and inspect Lab1.cmd by double clicking on the filename in the project window. Notice that the Memory{} declaration describes the system memory shown on the “Lab1: Linker Command File” slide in the objective section of this lab exercise. Memory blocks L0SARAM, L1SARAM, L2SARM, and L3SARAM have been combined into a single memory block called L0123SARAM. This combined memory block has been placed in program memory on page 0, and the other memory blocks have been placed in data memory on page 1.

9. In the Sections{} area notice that the sections defined on the slide have been “linked” into the appropriate memories. Also, notice that a section called .reset has been allocated. The .reset section is part of the rts2800_ml.lib, and is not needed. By putting the TYPE =
DSECT modifier after its allocation, the linker will ignore this section and not allocate it. Close the inspected file.

**Build and Load the Project**

10. The top four buttons on the horizontal toolbar control code generation. Hover your mouse over each button as you read the following descriptions:

<table>
<thead>
<tr>
<th>Button</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compile File</td>
<td>Compile, assemble the current open file</td>
</tr>
<tr>
<td>2</td>
<td>Incremental Build</td>
<td>Compile, assemble only changed files, then link</td>
</tr>
<tr>
<td>3</td>
<td>Rebuild All</td>
<td>Compile, assemble all files, then link</td>
</tr>
<tr>
<td>4</td>
<td>Stop Build</td>
<td>Stop code generation</td>
</tr>
</tbody>
</table>

11. Code Composer Studio can automatically load the output file after a successful build. On the menu bar click: **Option → Customize...** and select the “Program/Project/CIO” tab, check “Load Program After Build”.

Also, Code Composer Studio can automatically connect to the target when started. Select the “Debug Properties” tab, check “Connect to the target at startup”, then click OK.

12. Click the “Build” button and watch the tools run in the build window. Check for errors (we have deliberately put an error in Lab1.c). When you get an error, scroll the build window at the bottom of the Code Composer Studio screen until you see the error message (in red), and simply double-click the error message. The editor will automatically open the source file containing the error, and position the mouse cursor at the correct code line.

13. Fix the error by adding a semicolon at the end of the "z = x + y" statement. For future knowledge, realize that a single code error can sometimes generate multiple error messages at build time. This was not the case here.

14. Rebuild the project (there should be no errors this time). The output file should automatically load. The Program Counter should be pointing to _c_int00 in the Disassembly Window.

15. Under **Debug** on the menu bar click “Go Main”. This will run through the DSP/BIOS C-environment initialization routine and stop at main() in Lab1.c.

**Debug Enviroment Windows**

It is standard debug practice to watch local and global variables while debugging code. There are various methods for doing this in Code Composer Studio. We will examine two of them here: memory windows, and watch windows.

16. Open a *memory window* to view the global variable “z”.

   Click: **View → Memory...** on the menu bar.
Type “&z” into the address field and then enter. Note that you must use the ampersand (meaning “address of”) when using a symbol in a memory window address box. Also note that Code Composer Studio is case sensitive.

Set the properties format to “Hex 16 Bit – TI style” at the bottom of the window. This will give you more viewable data in the window. You can change the contents of any address in the memory window by double-clicking on its value. This is useful during debug.

17. Open the watch window to view the local variables x and y.

Click: View → Watch Window on the menu bar.

Click the “Watch Locals” tab and notice that the local variables x and y are already present. The watch window will always contain the local variables for the code function currently being executed.

(Note that local variables actually live on the stack. You can also view local variables in a memory window by setting the address to “SP” after the code function has been entered).

18. We can also add global variables to the watch window if desired. Let's add the global variable “z”.

Click the “Watch 1” tab at the bottom of the watch window. In the empty box in the “Name” column, type “z” and then enter. Note that you do not use an ampersand here. The watch window knows you are specifying a symbol. Check that the watch window and memory window both report the same value for “z”. Trying changing the value in one window, and notice that the value also changes in the other window.

**Single-stepping the Code**

19. Click the “Watch Locals” tab at the bottom of the watch window. Single-step through main() by using the <F11> key (or you can use the Single Step button on the vertical toolbar). Check to see if the program is working as expected. What is the value for “z” when you get to the end of the program?

**End of Exercise**
Peripheral Register Header Files

**Traditional Approach to C Coding**

```c
#define ADCTRL1 (volatile unsigned int *)0x00007100
#define ADCTRL2 (volatile unsigned int *)0x00007101

void main(void)
{
    *ADCTRL1 = 0x1234; //write entire register
    *ADCTRL2 |= 0x4000; //reset sequencer #1
}
```

**Advantages**
- Simple, fast and easy to type
- Variable names exactly match register names (easy to remember)

**Disadvantages**
- Requires individual masks to be generated to manipulate individual bits
- Cannot easily display bit fields in Watch window
- Will generate less efficient code in many cases

---

**Structure Approach to C Coding**

```c
void main(void)
{
    AdcRegs.ADCTRL1.all = 0x1234; //write entire register
    AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1; //reset sequencer #1
}
```

**Advantages**
- Easy to manipulate individual bits.
- Watch window is amazing! (next slide)
- Generates most efficient code (on C28x)

**Disadvantages**
- Can be difficult to remember the structure names (Editor Auto Complete feature to the rescue!)
- More to type (again, Editor Auto Complete feature to the rescue)
Peripheral Register Header Files

The CCS Watch Window using \#define

The CCS Watch Window using Structures
Structure Naming Conventions

- The DSP2833x header files define:
  - All of the peripheral structures
  - All of the register names
  - All of the bit field names
  - All of the register addresses

```
PeripheralName.RegisterName.all // Access full 16 or 32-bit register
PeripheralName.RegisterName.half.LSW // Access low 16-bits of 32-bit register
PeripheralName.RegisterName.half.MSW // Access high 16-bits of 32-bit register
PeripheralName.RegisterName.bit.FieldName // Access specified bit fields of register
```

Notes:
1. “PeripheralName” are assigned by TI and found in the DSP2833x header files. They are a combination of capital and small letters (i.e. CpuTimer0Regs).
2. “RegisterName” are the same names as used in the data sheet. They are always in capital letters (i.e. TCR, TIM, TPR,...).
3. “FieldName” are the same names as used in the data sheet. They are always in capital letters (i.e. POL, TOG, TSS,...).
Peripheral Register Header Files

**DSP2833x Header File Package**
(http://www.ti.com, literature # SPRC530)

- Contains everything needed to use the structure approach
- Defines all peripheral register bits and register addresses
- Header file package includes:

  - `
    \DSP2833x_headers\include
    \DSP2833x_headers\cmd
    \DSP2833x_headers\gel
    \DSP2833x_examples
    \DSP2823x_examples
    \doc
  `


documentation

**Peripheral Structure .h files (1 of 2)**

- Contain bits field structure definitions for each peripheral register

![Partial code listing showing structure definitions for ADC register](image)

Your C-source file (e.g., Adc.c)

```c
#include "DSP2833x_Device.h"
Void InitAdc(void)
{
    /* Reset the ADC module */
    AdcRegs.ADCTRL1.bit.RESET = 1;

    /* configure the ADC register */
    AdcRegs.ADCTRL1.all = 0x0710;
}
```

**DSP2833x_Adc.h**

```c
/* ADC Individual Register Bit Definitions */
struct ADCTRL1_BITS {
    Uint16 rsvd1:4; // 3:0 reserved
    Uint16 SEQ_CASC:1; // 4 Cascaded sequencer mode
    Uint16 SEQ_OVRD:1 // 5 Sequencer override
    Uint16 CONT_RUN:1; // 6 Continuous run
    Uint16 CPS:1; // 7 ADC core clock prescaler
    Uint16 ACQ_PS:4; // 11:8 Acquisition window size
    Uint16 SUSMOD:2; // 13:12 Emulation suspend mode
    Uint16 RESET:1; // 14 ADC reset
    Uint16 rsvd2:1; // 15 reserved
};
```

/* Allow access to the bit fields or entire register */
union ADCTRL1_REG {
    Uint16 all;
    struct ADCTRL1_BITS bit;
};

/* External References & Function Declarations:*/
extern volatile struct ADC_REGS AdcRegs;

```
Peripheral Structure .h files (2 of 2)

- The header file package contains a .h file for each peripheral in the device

<table>
<thead>
<tr>
<th>DSP2833x_Device.h</th>
<th>DSP2833x_DevEmu.h</th>
<th>DSP2833x_SysCtrl.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP2833x_PieCtrl.h</td>
<td>DSP2833x_Adc.h</td>
<td>DSP2833x_CpuTimers.h</td>
</tr>
<tr>
<td>DSP2833x_ECan.h</td>
<td>DSP2833x_ECap.h</td>
<td>DSP2833x_EPwm.h</td>
</tr>
<tr>
<td>DSP2833x_EQep.h</td>
<td>DSP2833x_Gpio.h</td>
<td>DSP2833x_I2c.h</td>
</tr>
<tr>
<td>DSP2833x_Sci.h</td>
<td>DSP2833x_Spi.h</td>
<td>DSP2833x_XIntrupt.h</td>
</tr>
<tr>
<td>DSP2833x_PieVect.h</td>
<td>DSP2833x_DefaultIsr.h</td>
<td>DSP2833x_DMA.h</td>
</tr>
<tr>
<td>DSP2833x_Mcbsp.h</td>
<td>DSP2833x_XIntf.h</td>
<td></td>
</tr>
</tbody>
</table>

- **DSP2833x_Device.h**
  - Main include file (for ‘2833x and ‘2823x devices)
  - Will include all other .h files
  - Include this file in each source file:
    ```c
    #include "DSP2833x_Device.h"
    ```

Global Variable Definitions File

**DSP2833x_GlobalVariableDefs.c**

- Declares a global instantiation of the structure for each peripheral
- Each structure put in its own section using a DATA_SECTION pragma to allow linking to correct memory (see next slide)

```c
#include "DSP2833x_Device.h"
...
#pragma DATA_SECTION(AdcRegs,"AdcRegsFile");
volatile struct ADC_REGS AdcRegs;
...
```

- Add this file to your CCS project:
  ```c
  DSP2833x_GlobalVariableDefs.c
  ```
Peripheral Register Header Files

Linker Command Files for the Structures

- DSP2833x_nonBIOS.cmd and DSP2833x_BIOS.cmd

- Links each structure to the address of the peripheral using the structures named section

- non-BIOS and BIOS versions of the .cmd file

- Add one of these files to your CCS project:
  - DSP2833x_nonBIOS.cmd
  - DSP2833x_BIOS.cmd

DSP2833x_GlobalVariableDefs.c

```c
#include "DSP2833x_Device.h"

#pragma DATA_SECTION(AdcRegs,"AdcRegsFile");
volatile struct ADC_REGS AdcRegs;
```

DSP2833x_Headers_nonBIOS.cmd

```c
MEMORY
{
  PAGE1:
  ...
  ADC: origin=0x007100, length=0x000020
  ...
}

SECTIONS
{
  ...
  AdcRegsFile: > ADC PAGE = 1
  ...
}
```

Peripheral Specific Examples

- Example(s) projects for each peripheral
- Helpful to get you started
- Separate projects for ‘2833x and ‘2823x
  - ‘2823x projects configured for no FPU
Peripheral Register Header Files

Summary

- Easier code development
- Easy to use
- Generates most efficient code
- Increases effectiveness of CCS watch window
- TI has already done all the work!
  - Use the correct header file package for your device:
    - F2833x and F2823x # SPRC530
    - F280x and F2801x # SPRC191
    - F2804x # SPRC324
    - F281x # SPRC097

Go to http://www.ti.com and enter the literature number in the keyword search box
Reset, Interrupts and System Initialization

Reset

Reset Sources

Watchdog Timer → XRS pin active → C28x Core → XRS → To XRS pin

Reset – Bootloader

Reset
OBJMODE = 0  AMODE = 0
ENPIE = 0  INTM = 1

Bootloader sets
OBJMODE = 1
AMODE = 0

Boot determined by state of GPIO pins

Reset vector fetched from boot ROM
0x3F FFC0

Execution Entry Point
FLASH
M0 SARAM
OTP
XINTF

Bootloading Routines
SCI-A / SPI-A
I2C
eCAN-A
McBSP-A
GPIO / XINTF
### Bootloader Options

<table>
<thead>
<tr>
<th>GPIO pins</th>
<th>87 / 86 / 85 / 84 / XA15 XA14 XA13 XA12</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1</td>
<td>jump to FLASH address 0x33 FFF6</td>
<td></td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>bootloader code to on-chip memory via SCI-A</td>
<td></td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>bootloader external EEPROM to on-chip memory via SPI-A</td>
<td></td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>bootloader external EEPROM to on-chip memory via I2C</td>
<td></td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>Call CAN_Boot to load from eCAN-A mailbox 1</td>
<td></td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>bootloader code to on-chip memory via McBSP-A</td>
<td></td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>jump to XINTF Zone 6 address 0x10 0000 for 16-bit data</td>
<td></td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>jump to XINTF Zone 6 address 0x10 0000 for 32-bit data</td>
<td></td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>jump to OTP address 0x38 0400</td>
<td></td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>bootloader code to on-chip memory via GPIO port A (parallel)</td>
<td></td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>bootloader code to on-chip memory via XINTF (parallel)</td>
<td></td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>jump to M0 SARAM address 0x00 0000</td>
<td></td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>branch to check boot mode</td>
<td></td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>branch to Flash without ADC calibration (TI debug only)</td>
<td></td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>branch to M0 SARAM without ADC calibration (TI debug only)</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>branch to SCI-A without ADC calibration (TI debug only)</td>
<td></td>
</tr>
</tbody>
</table>

### Reset Code Flow - Summary

- **Reset Code Flow - Summary**
  - **M0 SARAM (1Kw)**
  - **OTP (1Kw)**
  - **FLASH (256Kw)**
  - **Boot ROM (8Kw)**
  - **Boot Code** 0x3F F9A9
  - **BROM vector (64w)** 0x3F F9A9
  - 0x3F FFC0
  - **XINTF Zone 6 (x16 / x32) 0x10 0000**
  - **Execution Entry Point Determined By GPIO Pins**
Interrupts

### Internal Sources
- TINT2
- TINT1
- TINT0
- ePWM, eCAP, eQEP, ADC, SCI, SPI, I2C, eCAN, McBSP, DMA, WD

### External Sources
- XINT1 – XINT7
- TZx
- XRS
- XNMI_XINT13

### PIE (Peripheral Interrupt Expansion)

**C28x CORE**
- XRS
- NMI
- INT1
- INT2
- INT3
- INT12
- INT13
- INT14

### Maskable Interrupt Processing

- **Core Interrupt**
  - INT1
  - INT2
  - INT3
  - INT12
  - INT13
  - INT14

- **(IFR) “Latch”**
  - INT1: 1
  - INT2: 0

- **(IER) “Switch”**
  - INT1: 1
  - INT2: 0

- **(INTM) “Global Switch”**

- **C28x Core**

- **A valid signal on a specific interrupt line causes the latch to display a “1” in the appropriate bit**
- **If the individual and global switches are turned “on” the interrupt reaches the core**
Core Interrupt Registers

Interrupt Flag Register (IFR)

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTOSINT</td>
<td>DLOGINT</td>
<td>INT14</td>
<td>INT13</td>
<td>INT12</td>
<td>INT11</td>
<td>INT10</td>
<td>INT9</td>
</tr>
<tr>
<td>INT8</td>
<td>INT7</td>
<td>INT6</td>
<td>INT5</td>
<td>INT4</td>
<td>INT3</td>
<td>INT2</td>
<td>INT1</td>
</tr>
</tbody>
</table>

Interrupt Enable Register (IER)

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTOSINT</td>
<td>DLOGINT</td>
<td>INT14</td>
<td>INT13</td>
<td>INT12</td>
<td>INT11</td>
<td>INT10</td>
<td>INT9</td>
</tr>
<tr>
<td>INT8</td>
<td>INT7</td>
<td>INT6</td>
<td>INT5</td>
<td>INT4</td>
<td>INT3</td>
<td>INT2</td>
<td>INT1</td>
</tr>
</tbody>
</table>

Interrupt Global Mask Bit (INTM)

Bit 0

ST1

Peripheral Interrupt Expansion (PIE)

Peripheral Interrupt Expansion - PIE

PIE module for 96 Interrupts

Interrupt Group 1

PIEIR1

Interrupt Group 1

INT1.1

INT1.2

INT1.8

INT1 – INT12

12 Interrupts

28x Core Interrupt logic
Reset, Interrupts and System Initialization

PIE Registers

PIEFRx register (x = 1 to 12)

<table>
<thead>
<tr>
<th>15 - 8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>reserved</td>
<td>INTx.8</td>
<td>INTx.7</td>
<td>INTx.6</td>
<td>INTx.5</td>
<td>INTx.4</td>
<td>INTx.3</td>
<td>INTx.2</td>
<td>INTx.1</td>
</tr>
</tbody>
</table>

PIEIEx register (x = 1 to 12)

<table>
<thead>
<tr>
<th>15 - 8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>reserved</td>
<td>INTx.8</td>
<td>INTx.7</td>
<td>INTx.6</td>
<td>INTx.5</td>
<td>INTx.4</td>
<td>INTx.3</td>
<td>INTx.2</td>
<td>INTx.1</td>
</tr>
</tbody>
</table>

PIE Interrupt Acknowledge Register (PIEACK)

<table>
<thead>
<tr>
<th>15 - 12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>reserved</td>
<td>PIEACKx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PIECTRl register

<table>
<thead>
<tr>
<th>15 - 1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIEVECT</td>
<td>ENPIE</td>
</tr>
</tbody>
</table>

#include "DSP2833x_Device.h"
PiecCtrlRegs.PIEFR1.bit.INTx4 = 1;  //manually set IFR for XINT1 in PIE group 1
PiecCtrlRegs.PIEFR3.bit.INTx5 = 1;  //enable EPWM5_INT in PIE group 3
PiecCtrlRegs.PIEACK.all = 0x0004;  //acknowledge the PIE group 3
PiecCtrlRegs.PIECTRL.bit.ENPIE = 1;  //enable the PIE

F2833x PIE Interrupt Assignment Table

<table>
<thead>
<tr>
<th>INTx.8</th>
<th>INTx.7</th>
<th>INTx.6</th>
<th>INTx.5</th>
<th>INTx.4</th>
<th>INTx.3</th>
<th>INTx.2</th>
<th>INTx.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT1</td>
<td>WAKEINT</td>
<td>TINT0</td>
<td>ADCINT</td>
<td>XINT2</td>
<td>XINT1</td>
<td>SEQ2INT</td>
<td>SEQ1INT</td>
</tr>
<tr>
<td>INT2</td>
<td>EPWM6_TZINT</td>
<td>EPWM5_TZINT</td>
<td>EPWM4_TZINT</td>
<td>EPWM3_TZINT</td>
<td>EPWM2_TZINT</td>
<td>EPWM1_TZINT</td>
<td></td>
</tr>
<tr>
<td>INT3</td>
<td>EPWM6_INT</td>
<td>EPWM5_INT</td>
<td>EPWM4_INT</td>
<td>EPWM3_INT</td>
<td>EPWM2_INT</td>
<td>EPWM1_INT</td>
<td></td>
</tr>
<tr>
<td>INT4</td>
<td>ECAP6_INT</td>
<td>ECAP5_INT</td>
<td>ECAP4_INT</td>
<td>ECAP3_INT</td>
<td>ECAP2_INT</td>
<td>ECAP1_INT</td>
<td></td>
</tr>
<tr>
<td>INT5</td>
<td></td>
<td>EEP2_INT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT6</td>
<td>MXINTA</td>
<td>MRINTA</td>
<td>MXINTB</td>
<td>MRINTB</td>
<td>SPIXINTA</td>
<td>SPIRXINTA</td>
<td></td>
</tr>
<tr>
<td>INT7</td>
<td>DINTCH6</td>
<td>DINTCH5</td>
<td>DINTCH4</td>
<td>DINTCH3</td>
<td>DINTCH2</td>
<td>DINTCH1</td>
<td></td>
</tr>
<tr>
<td>INT8</td>
<td>SCITXINTC</td>
<td>SCIRXINTC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT9</td>
<td>ECAN1_INTB</td>
<td>ECAN0_INTB</td>
<td>ECAN1_INTA</td>
<td>ECAN0_INTA</td>
<td>SCITXINTB</td>
<td>SCIRXINTB</td>
<td></td>
</tr>
<tr>
<td>INT10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT12</td>
<td>LUF</td>
<td>LVF</td>
<td>XINT7</td>
<td>XINT6</td>
<td>XINT5</td>
<td>XINT4</td>
<td>XINT3</td>
</tr>
</tbody>
</table>

#30  TMS320F28335 One-Day Workshop
Default Interrupt Vector Table at Reset

<table>
<thead>
<tr>
<th>Vector</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET</td>
<td>00</td>
</tr>
<tr>
<td>INT1</td>
<td>02</td>
</tr>
<tr>
<td>INT2</td>
<td>04</td>
</tr>
<tr>
<td>INT3</td>
<td>06</td>
</tr>
<tr>
<td>INT4</td>
<td>08</td>
</tr>
<tr>
<td>INT5</td>
<td>0A</td>
</tr>
<tr>
<td>INT6</td>
<td>0C</td>
</tr>
<tr>
<td>INT7</td>
<td>0E</td>
</tr>
<tr>
<td>INT8</td>
<td>10</td>
</tr>
<tr>
<td>INT9</td>
<td>12</td>
</tr>
<tr>
<td>INT10</td>
<td>14</td>
</tr>
<tr>
<td>INT11</td>
<td>16</td>
</tr>
<tr>
<td>INT12</td>
<td>18</td>
</tr>
<tr>
<td>INT13</td>
<td>1A</td>
</tr>
<tr>
<td>INT14</td>
<td>1C</td>
</tr>
<tr>
<td>DATALOG</td>
<td>1E</td>
</tr>
<tr>
<td>RTOSINT</td>
<td>20</td>
</tr>
<tr>
<td>EMUINT</td>
<td>22</td>
</tr>
<tr>
<td>NMI</td>
<td>24</td>
</tr>
<tr>
<td>ILLEGAL</td>
<td>26</td>
</tr>
<tr>
<td>USER 1-12</td>
<td>28-3E</td>
</tr>
</tbody>
</table>

Default Vector Table Re-mapped when ENPIE = 1

Oscillator / PLL Clock Module

F2833x Oscillator / PLL Clock Module
(lab file: SysCtrl.c)

<table>
<thead>
<tr>
<th>DIVSEL</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1/4*</td>
</tr>
<tr>
<td>01</td>
<td>1/2</td>
</tr>
<tr>
<td>11</td>
<td>1/1</td>
</tr>
</tbody>
</table>

* default

Note: /1 mode can only be used when PLL is bypassed.

Input Clock Fail Detect Circuitry
PLL will issue a “limp mode” clock (1-4 MHz) if input clock is removed after PLL has locked.
An internal device reset will also be issued (XRSn pin not driven).
Watchdog Timer Module

Watchdog Timer

- Resets the C28x if the CPU crashes
  - Watchdog counter runs independent of CPU
  - If counter overflows, a reset or interrupt is triggered (user selectable)
  - CPU must write correct data key sequence to reset the counter before overflow
- Watchdog must be serviced or disabled within 131,072 instructions after reset
- This translates to 4.37 ms with a 30 MHz OSCCLK

---

Watchdog Timer Module (lab file: Watchdog.c)

6 - Bit Free - Running Counter

System Reset

55 + AA Detector

Watchdog Reset Key Register

64
32
16
8
4
2
1

OSCCLK

/512

/64

/16

/8

/4

/2

CLR

111
101
100
011
010
001
000

WDPS

WDCR . 2 - 0

WDCR . 0

WDOVERRIDE

WDCNTR . 7 - 0

8 - Bit Watchdog Counter

WDKEY . 7 - 0

WDCR . 5 - 3

WDCHK 2-0

One-Cycle Delay

WDKEY . 7 - 0

Output Pulse

WDRST

WDINT

WDFLAG

WDCR . 7

Good Key

SCSR . 1

WENINT

WDENINT

WDOVERRIDE

WDIFFLAG

Bad WDCR Key

SCSR . 0

WDOVERRIDE

WDIFFLAG

Bad WDCR Key

SCSR . 0

WDOVERRIDE

WDIFFLAG

Bad WDCR Key

SCSR . 0

WDOVERRIDE

WDIFFLAG

Bad WDCR Key

SCSR . 0

WDOVERRIDE

WDIFFLAG

Bad WDCR Key

SCSR . 0

WDOVERRIDE

WDIFFLAG

Bad WDCR Key
GPIO

F2833x GPIO Grouping Overview

- GPIO Port A Mux1 Register (GPAMUX1) [GPIO 0 to 15]
- GPIO Port A Mux2 Register (GPAMUX2) [GPIO 16 to 31]
- GPIO Port B Mux1 Register (GPBMUX1) [GPIO 32 to 47]
- GPIO Port B Mux2 Register (GPBMUX2) [GPIO 48 to 63]
- GPIO Port C Mux1 Register (GPCMUX1) [GPIO 64 to 79]
- GPIO Port C Mux2 Register (GPCMUX2) [GPIO 80 to 87]

- GPIO Port A Direction Register (GPADIR) [GPIO 0 to 31]
- GPIO Port B Direction Register (GPBDIR) [GPIO 32 to 63]
- GPIO Port C Direction Register (GPCDIR) [GPIO 64 to 87]

- Input Qual

F2833x GPIO Pin Block Diagram

- GPxSET
- GPxCLEAR
- GPxTOGGLE
- GPxDAT
- I/O DIR Bit
- 0 = Input
- 1 = Output
- GPxDIR
- In
- Out
- GPxPUD
- Internal Pull-Up
- 0 = enable (default GPIO 12-31)
- 1 = disable (default GPIO 0-11)

- Peripheral
- 1
- 10
- 01
- 11
- GPxMUX1
- GPxMUX2
- MUX Control Bits *
- 00 = GPIO
- 01 = Peripheral 1
- 10 = Peripheral 2
- 11 = Peripheral 3

- Input Qualification (GPIO 0-63 only)

- * See device datasheet for pin function selection matrices
Lab 2: System Initialization

- LAB2 files have been provided
- LAB2 consists of two parts:
  Part 1
  - Test behavior of watchdog when disabled and enabled
  Part 2
  - Initialize peripheral interrupt expansion (PIE) vectors and use watchdog to generate an interrupt
- Modify, build, and test code using Code Composer Studio
Lab 2: System Initialization

Objective

The objective of this lab is to perform the processor system initialization. Additionally, the peripheral interrupt expansion (PIE) vectors will be initialized and tested. The system initialization for this lab will consist of the following:

- Setup the clock module – PLL, HISPCP = /1, LOSPCP = /4, low-power modes to default values, enable all module clocks
- Disable the watchdog – clear WD flag, disable watchdog, WD prescale = 1
- Setup watchdog system control register – DO NOT clear WD OVERRIDE bit, WD generate a DSP reset
- Setup shared I/O pins – set all GPIO pins to GPIO function (e.g. a "00" setting for GPIO function, and a “01”, “10”, or “11” setting for peripheral function.)

The first part of the lab exercise will setup the system initialization and test the watchdog operation by having the watchdog cause a reset. In the second part of the lab exercise the PIE vectors will tested by using the watchdog to generate an interrupt. This lab will make use of the DSP2833x C-code header files to simplify the programming of the device, as well as take care of the register definitions and addresses. Please review these files, and make use of them in the future, as needed.

Procedure

Project File

1. A project named Lab2.pjt has been created for this lab. Open the project by clicking on Project Æ Open... and look in C:\C28x\LABS\LAB2. All Build Options have been configured. The files used in this lab are:

   - CodeStartBranch.asm
   - DefaultIsr_2.c
   - DelayUs.asm
   - DSP2833x_GlobalVariableDefs.c
   - DSP2833x_Headers_nonBIOS.cmd
   - Gpio.c
   - Lab_2_3.cmd
   - Main_2.c
   - PieCtrl.c
   - PieVect.c
   - SysCtrl.c
   - Watchdog.c

   Note that include files, such as DSP2833x_Device.h and Lab.h, are automatically added at project build time. (Also, DSP2833x_DefaultIsr.h is automatically added and will be used with the interrupts in the second part of this lab exercise).

Modified Memory Configuration

2. Open and inspect the linker command file Lab_2_3.cmd. Notice that the user defined section “codestart” is being linked to a memory block named BEGIN_M0. The
codestart section contains code that branches to the code entry point of the project. The bootloader must branch to the codestart section at the end of the boot process. Recall that the "Jump to M0 SARAM" bootloader mode branches to address 0x000000 upon bootloader completion.

The linker command file (Lab_2_3.cmd) has a new memory block named BEGIN_M0: origin = 0x000000, length = 0x0002, in program memory. Additionally, the existing memory block MOSARAM in data memory has been modified to avoid overlaps with this new memory block.

System Initialization

3. Open and inspect SysCtrl.c. Notice that the PLL and module clocks have been enabled.

4. Open and inspect Watchdog.c. Notice that watchdog control register (WDCR) is configured to disable the watchdog, and the system control and status register (SCSR) is configured to generate a reset.

5. Open and inspect Gpio.c. Notice that the shared I/O pins have been set to the GPIO function, except for GPIO0 which will be used in the next lab exercise. Close the inspected files.

Build and Load

6. Click the “Build” button and watch the tools run in the build window. The output file should automatically load.

7. Under Debug on the menu bar click “Reset CPU”.

8. Under Debug on the menu bar click “Go Main”. You should now be at the start of Main().

Run the Code – Watchdog Reset

9. Place the cursor on the first line of code in main() and set a breakpoint by right clicking the mouse key and select Toggle Software Breakpoint. Notice that line is highlighted with a red dot indicating that the breakpoint has been set. Alternately, you can double-click in the gray field to the left of the code line to set the breakpoint. The breakpoint is set to prove that the watchdog is disabled. If the watchdog causes a reset, code execution will stop at this breakpoint.

10. Place the cursor in the “main loop” section (on the asm(" NOP"); instruction line) and right click the mouse key and select Run To Cursor. This is the same as setting a breakpoint on the selected line, running to that breakpoint, and then removing the breakpoint.

11. Run your code for a few seconds by using the <F5> key, or using the Run button on the vertical toolbar, or using Debug ➔ Run on the menu bar. After a few seconds halt your code by using Shift <F5>, or the Halt button on the vertical toolbar. Where did your
12. Modify the `InitWatchdog()` function to enable the watchdog – in `Watchdog.c` change the `WDCR` register value to `0x00A8`. This will enable the watchdog to function and cause a reset. Save the file and click the “Build” button. Then reset the CPU by clicking on `Debug` → `Reset CPU`. Under Debug on the menu bar click “Go Main”.


14. Run your code. Where did your code stop? Are the results as expected? If things went as expected, your code should stop at the breakpoint.

**Setup PIE Vector for Watchdog Interrupt**

The first part of this lab exercise used the watchdog to generate a CPU reset. This was tested using a breakpoint set at the beginning of `main()`. Next, we are going to use the watchdog to generate an interrupt. This part will demonstrate the interrupt concepts learned in the previous module.

15. Notice that the following files are included in the project:
   - `DefaultIsr_2.c`
   - `PieCtrl.c`
   - `PieVect.c`

16. In `Main_2.c`, the following code is used to call the `InitPieCtrl()` function. There are no passed parameters or return values, so the call code is simply:

   ```c
   InitPieCtrl();
   ```

17. Using the “PIE Interrupt Assignment Table” shown in the slides find the location for the watchdog interrupt, “WAKEINT”. This is used in the next step.

   PIE group #: ___________  # within group: ______

18. In `main()` notice the code used to enable global interrupts (INTM bit), and in `InitWatchdog()` the code used to enable the “WAKEINT” interrupt in the PIE (using the `PieCtrlRegs` structure) and to enable core INT1 (IER register).

19. Modify the system control and status register (SCSR) to cause the watchdog to generate a WAKEINT rather than a reset – in `Watchdog.c` change the `SCSR` register value to `0x0002`. Save this modified file.

20. Open and inspect `DefaultIsr_2.c`. This file contains interrupt service routines. The ISR for WAKEINT has been trapped by an emulation breakpoint contained in an inline assembly statement using “ESTOP0”. This gives the same results as placing a breakpoint in the ISR. We will run the lab exercise as before, except this time the watchdog will generate an interrupt. If the registers have been configured properly, the code will be trapped in the ISR.
21. Open and inspect `PieCtrl.c`. This file is used to initialize the PIE RAM and enable the PIE. The interrupt vector table located in `PieVect.c` is copied to the PIE RAM to setup the vectors for the interrupts. Close the modified and inspected files.

**Build and Load**

22. Click the “Build” button. Then reset the CPU, and then “Go Main”.

**Run the Code – Watchdog Interrupt**

23. Place the cursor in the “main loop” section, right click the mouse key and select Run To Cursor.

24. Run your code. Where did your code stop? Are the results as expected? If things went as expected, your code should stop at the “ESTOP0” instruction in the WAKEINT ISR.

**End of Exercise**

---

**Note:** By default, the watchdog timer is enabled out of reset. Code in the file `CodeStartBranch.asm` has been configured to disable the watchdog. This can be important for large C code projects (ask your instructor if this has not already been explained). During this lab exercise, the watchdog was actually re-enabled (or disabled again) in the file `Watchdog.c`.
Control Peripherals

ADC Module

ADC Module Block Diagram (Cascaded Mode)

ADC Module Block Diagram (Dual-Sequencer mode)
ADC Control Registers (file: Adc.c)

- **ADCTRL1** (ADC Control Register 1)
  - module reset
  - continuous run / stop EOS
  - sequencer mode (cascaded / dual)
  - acquisition time prescale (S/H)
- **ADCTRL2** (ADC Control Register 2)
  - ePWM SOC; start conversion (s/w trigger); ePWM SOC mask bit
  - reset SEQ
  - interrupt enable; interrupt mode: every EOS / every other EOS
- **ADCTRL3** (ADC Control Register 3)
  - ADC clock prescale
  - sampling mode (sequential / simultaneous)
- **ADCMAXCONV** (ADC Maximum Conversion Register)
  - maximum number of autoconversions
- **ADCCHSELSEQx {x=1-4}** (ADC Channel Select Register)
  - Channel select sequencing
- **ADCRESULTx {x=0-15}** (ADC Results Register)

*Note: refer to the reference guide for a complete listing of registers*

Pulse Width Modulation

What is Pulse Width Modulation?

- **PWM** is a scheme to represent a signal as a sequence of pulses
  - fixed carrier frequency
  - fixed pulse amplitude
  - pulse width proportional to instantaneous signal amplitude
  - PWM energy $\approx$ original signal energy

![Diagram of original signal and PWM representation](image)
Why use PWM with Power Switching Devices?

- Desired output currents or voltages are known
- Power switching devices are transistors
  - Difficult to control in proportional region
  - Easy to control in saturated region
- PWM is a digital signal $\Rightarrow$ easy for DSP to output

---

ePWM

ePWM Block Diagram

![ePWM Block Diagram](image-url)
Control Peripherals

**ePWM Time-Base Module**

- Clock Prescaler
- Compare Logic
- Action Qualifier
- Dead Band

**16-Bit Time-Base Counter**

- TBCTR : 15 - 0
- TBCLK
- EPWMxSYNCI
- SYSCLKOUT

**Period Register**

- TBPRD : 15 - 0
- TBCTL : 12 - 7

**Compare Registers**

- CMPA : 15 - 0
- CMPB : 15 - 0
- AQCTLA : 11 - 0
- AQCTLB : 11 - 0

**Action Qualifier Registers**

- AQCTLB : 11 - 0
- DBCTL : 4 - 0

**Timer Control**

- EPWMxA
- EPWMxB
- TZy

**ePWM Time-Base Count Modes**

- **Count Up Mode**
  - Asymmetrical Waveform

- **Count Down Mode**
  - Asymmetrical Waveform

- **Count Up and Down Mode**
  - Symmetrical Waveform
Control Peripherals

### ePWM Compare Event Waveforms

- **Count Up Mode**
  - Asymmetrical Waveform
  - `TBCTR` and `TBPRD`
  - CMPA and CMPB

- **Count Down Mode**
  - Asymmetrical Waveform
  - `TBCTR` and `TBPRD`
  - CMPA and CMPB

- **Count Up and Down Mode**
  - Symmetrical Waveform
  - `TBCTR` and `TBPRD`
  - CMPA and CMPB

- `= compare events are fed to the Action Qualifier Module`

### ePWM Action Qualifier Module

- **16-Bit Time-Base Counter**
- **Compare Logic**
- **Action Qualifier**
- **Dead Band**
- **Trip Zone**
- **PWM Chopper**

- **Clock Prescaler**
- **Compare Registers**
- **Period Register**
- **Action Qualifier Registers**
- **Dead Band Control**

- **TBCLK** and **EPWMxSYNCI**
- **SYSCLKOUT**

- `EPWMxA`, `EPWMxB`, `EPWMxA`
### ePWM Action Qualifier Actions
for EPWMA and EPWMB

<table>
<thead>
<tr>
<th>S/W Force</th>
<th>Time-Base Counter equals:</th>
<th>EPWM Output Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Zero</td>
<td>CMPA</td>
</tr>
<tr>
<td>SWX</td>
<td>Z</td>
<td>CA</td>
</tr>
<tr>
<td>SW↓</td>
<td>Z</td>
<td>CA</td>
</tr>
<tr>
<td>SW↑</td>
<td>Z</td>
<td>CA</td>
</tr>
<tr>
<td>SWT</td>
<td>Z</td>
<td>CA</td>
</tr>
</tbody>
</table>

### ePWM Count Up Asymmetric Waveform
with Independent Modulation on EPWMA / B

![Waveform Diagram](attachment:image.png)
**Control Peripherals**

### ePWM Count Up Asymmetric Waveform
with Independent Modulation on EPWMA

- **TBCTR**
- **TBPRD**

- EPWMA
- EPWMB

### ePWM Count Up-Down Symmetric Waveform
with Independent Modulation on EPWMA / B

- **TBCTR**
- **TBPRD**

- EPWMA
- EPWMB
ePWM Count Up-Down Symmetric Waveform
with Independent Modulation on EPWMA

ePWM Dead-Band Module
**Motivation for Dead-Band**

- Transistor gates turn on faster than they shut off
- Short circuit if both gates are on at the same time!

**ePWM PWM Chopper Module**

![Diagram of ePWM PWM Chopper Module](image-url)
**ePWM Chopper Waveform**

- Allows a high frequency carrier signal to modulate the PWM waveform generated by the Action Qualifier and Dead-Band modules
- Used with pulse transformer-based gate drivers to control power switching elements

**ePWM Trip-Zone Module**

- 16-Bit Time-Base Counter
- Compare Logic
- Action Qualifier
- Dead Band
- PWM Chopper
- Trip Zone
- EPWMxA
- EPWMxB
- EPWMxSYNCl
- EPWMxSYNC0
- SYSCLKOUT
- TBCLK
- TBCTRL . 12 - 7
- TBCTR . 15 - 0
- CMPA . 15 - 0
- CMB . 15 - 0
- AQCTLA . 11 - 0
- AQCTLB . 11 - 0
- DBCTL . 4 - 0
- TBPDR . 15 - 0
- PCCTL . 10 - 0
- TZSEL . 15 - 0
Trip-Zone Module Features

- Trip-Zone has a fast, clock independent logic path to high-impedance the EPWMxA/B output pins
- Interrupt latency may not protect hardware when responding to over current conditions or short-circuits through ISR software
- Supports: 
  1) one-shot trip for major short circuits or over current conditions
  2) cycle-by-cycle trip for current limiting operation

![Diagram of Trip-Zone Module Features]

---

ePWM Event-Trigger Module

![Diagram of ePWM Event-Trigger Module]
Control Peripherals

Hi-Resolution PWM (HRPWM)

- Significantly increases the resolution of conventionally derived digital PWM
- Uses 8-bit extensions to Compare registers (CMPxHR) and Phase register (TBPHSHR) for edge positioning control
- Typically used when PWM resolution falls below ~9-10 bits which occurs at frequencies greater than ~300 kHz (with system clock of 150 MHz)
- Not all ePWM outputs support HRPWM feature (see device datasheet)

PWM Period

Device Clock (i.e. 150 MHz)

HRPWM divides a clock cycle into smaller steps called Micro Steps
(Step Size ~ 150 ps)

Calibration Logic tracks the number of Micro Steps per clock to account for variations caused by Temp/Volt/Process

HRPWM Micro Step (~150 ps)

Regular PWM Step (i.e. 6.67 ns)

Calibration Logic

EPMW Event-Trigger Interrupts and SOC

TBCTR
TBPRD
CMPB
CMPA
EPWMA
EPWMB
CTR = 0
CTR = PRD
CTRU = CMPA
CTRD = CMPA
CTRU = CMPB
CTRD = CMPB

Hi-Resolution PWM (HRPWM)

- Significantly increases the resolution of conventionally derived digital PWM
- Uses 8-bit extensions to Compare registers (CMPxHR) and Phase register (TBPHSHR) for edge positioning control
- Typically used when PWM resolution falls below ~9-10 bits which occurs at frequencies greater than ~300 kHz (with system clock of 150 MHz)
- Not all ePWM outputs support HRPWM feature (see device datasheet)
ePWM Control Registers (file: EPwm.c)

- **TBCTL** (Time-Base Control)
  - counter mode (up, down, up & down, stop); clock prescale; period shadow load; phase enable/direction; sync select
- **CMPCTL** (Compare Control)
  - compare load mode; operating mode (shadow / immediate)
- **AQCTLA/B** (Action Qualifier Control Output A/B)
  - action on up/down CTR = CMPA/B, PRD, 0 (nothing/set/clear/toggle)
- **DBCTL** (Dead-Band Control)
  - in/out-mode (disable / delay PWMxA/B); polarity select
- **PCCTL** (PWM-Chopper Control)
  - enable / disable; chopper CLK freq. & duty cycle; 1-shot pulse width
- **TZCTL** (Trip-Zone Control)
  - enable / disable; action (force high / low / high-Z/nothing)
- **ETSEL** (Event-Trigger Selection)
  - interrupt & SOCA/B enable / disable; interrupt & SOCA/B select

Note: refer to the reference guide for a complete listing of registers

---

eCAP

Capture Units (eCAP)

- The eCAP module timestamps transitions on a capture input pin
eQEP

**What is an Incremental Quadrature Encoder?**

A digital (angular) position sensor

-[photo sensors spaced $\theta/4$ deg. apart](#)
- [slots spaced $\theta$ deg. apart](#)
- [light source (LED)](#)

Incremental Optical Encoder

Quadrature Output from Photo Sensors

**How is Position Determined from Quadrature Signals?**

Position resolution is $\theta/4$ degrees

-Ch. A: (00) | (10) | (01)
-Ch. B: (11)

Quadrature Decoder

State Machine

Increment counter
-Decrement counter

Illegal Transitions; generate phase error interrupt

Quadrature Decoder State Machine
eQEP Connections
Lab 3: Control Peripherals

- **Objective**

The objective of this lab is to demonstrate the techniques discussed in this module and become familiar with the operation of the on-chip analog-to-digital converter and ePWM. ePWM1A will be setup to generate a 2 kHz, 25% duty cycle symmetric PWM waveform. The waveform will then be sampled with the on-chip analog-to-digital converter and displayed using the graphing feature of Code Composer Studio. The ADC has been setup to sample a single input channel at a 48 kHz sampling rate and store the conversion result in a buffer in the DSP memory. This buffer operates in a circular fashion, such that new conversion data continuously overwrites older results in the buffer.

Two ePWM modules have been configured for this lab exercise:

- **ePWM1A – PWM Generation**
  - Used to generate a 2 kHz, 25% duty cycle symmetric PWM waveform

- **ePWM2 – ADC Conversion Trigger**
  - Used as a timebase for triggering ADC samples (period match trigger SOC A)

The software in this exercise configures the ePWM modules and the ADC. It is entirely interrupt driven. The ADC end-of-conversion interrupt will be used to prompt the CPU to copy the results of the ADC conversion into a results buffer in memory. This buffer pointer will be managed in a circular fashion, such that new conversion results will continuously overwrite older conversion.
results in the buffer. The ADC interrupt service routine (ISR) will also toggle LED DS2 on the eZdsp™ as a visual indication that the ISR is running.

**Notes**
- ePWM1A is used to generate a 2 kHz PWM waveform
- Program performs conversion on ADC channel A0 (ADCINA0 pin)
- ADC conversion is set at a 48 kHz sampling rate
- ePWM2 is triggering the ADC on period match using SOC A trigger
- Data is continuously stored in a circular buffer
- Data is displayed using the graphing feature of Code Composer Studio
- ADC ISR will also toggle the eZdsp™ LED DS2 as a visual indication that it is running

➢ Procedure

**Project File**

1. A project named Lab3.pjt has been created for this lab. Open the project by clicking on Project ➔ Open... and look in C:\C28x\LABS\LAB3. All Build Options have been configured. The files used in this lab are:

   - Adc.c
   - CodeStartBranch.asm
   - DefaultIsr_3_4.c
   - DelayUs.asm
   - DSP2833x_GlobalVariableDefs.c
   - DSP2833x_Headers_nonBIOS.cmd
   - EPwm.c
   - Gpio.c
   - Lab_2_3.cmd
   - Main_3.c
   - PieCtrl.c
   - PieVect.c
   - SysCtrl.c
   - Watchdog.c

**Setup of Shared I/O, General-Purpose Timer1 and Compare1**

**Note:** DO NOT make any changes to Gpio.c and EPwm.c — ONLY INSPECT

2. Open and inspect Gpio.c by double clicking on the filename in the project window. Notice that the shared I/O pin in GPIO0 has been set for the ePWM1A function. Next, open and inspect EPwm.c and see that the ePWM1 has been setup to implement the PWM waveform as described in the objective for this lab. Notice the values used in the following registers: TBCTL (set clock prescales to divide-by-1, no software force, sync and phase disabled), TBPRD, CMPA, CMPCTL (load on 0 or PRD), and AQCTLA (set on up count and clear on down count for output A). Software force, deadband, PWM chopper and trip action has been disabled. (Note that the last steps enable the timer count mode and enable the clock to the ePWM module). See the global variable names and values that have been set using #define in the beginning of the Lab.h file. Notice that ePWM2 has been initialized earlier in the code for the ADC. Close the inspected files.
Lab 3: Control Peripherals

**Build and Load**

3. Click the “Build” button and watch the tools run in the build window. The output file should automatically load.

4. Under Debug on the menu bar click “Reset CPU”.

5. Under Debug on the menu bar click “Go Main”. You should now be at the start of Main().

**Run the Code – PWM Waveform**

6. Open a memory window to view some of the contents of the ADC results buffer. To open a memory window click: View → Memory... on the menu bar. The address label for the ADC results buffer is AdcBuf.

---

**Note:** Exercise care when connecting any wires, as the power to the eZdsp™ is on, and we do not want to damage the eZdsp™! Details of pin assignments can be found in Appendix A.

---

7. Using a connector wire provided, connect the PWM1A (pin # P8-9) to ADCINA0 (pin # P9-2) on the eZdsp™.

8. Run your code for a few seconds by using the <F5> key, or using the Run button on the vertical toolbar, or using Debug → Run on the menu bar. After a few seconds halt your code by using Shift <F5>, or the Halt button on the vertical toolbar. Verify that the ADC result buffer contains the updated values.

9. Open and setup a graph to plot a 48-point window of the ADC results buffer. Click: View → Graph → Time/Frequency... and set the following values:

<table>
<thead>
<tr>
<th>Start Address</th>
<th>AdcBuf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acquisition Buffer Size</td>
<td>48</td>
</tr>
<tr>
<td>Display Data Size</td>
<td>48</td>
</tr>
<tr>
<td>DSP Data Type</td>
<td>16-bit unsigned integer</td>
</tr>
<tr>
<td>Sampling Rate (Hz)</td>
<td>48000</td>
</tr>
<tr>
<td>Time Display Unit</td>
<td>μs</td>
</tr>
</tbody>
</table>

Select OK to save the graph options.

10. The graphical display should show the generated 2 kHz, 25% duty cycle symmetric PWM waveform. The period of a 2 kHz signal is 500 μs. You can confirm this by measuring the period of the waveform using the graph (you may want to enlarge the graph window using the mouse). The measurement is best done with the mouse. The lower left-hand corner of the graph window will display the X and Y-axis values.
Subtract the X-axis values taken over a complete waveform period (you can use the PC calculator program found in Microsoft Windows to do this).

**Frequency Domain Graphing Feature of Code Composer Studio**

11. Code Composer Studio also has the ability to make frequency domain plots. It does this by using the PC to perform a Fast Fourier Transform (FFT) of the DSP data. Let's make a frequency domain plot of the contents in the ADC results buffer (i.e. the PWM waveform).

Click: **View → Graph → Time/Frequency**... and set the following values:

<table>
<thead>
<tr>
<th>Display Type</th>
<th>FFT Magnitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Address</td>
<td>AdcBuf</td>
</tr>
<tr>
<td>Acquisition Buffer Size</td>
<td>48</td>
</tr>
<tr>
<td>FFT Framesize</td>
<td>48</td>
</tr>
<tr>
<td>DSP Data Type</td>
<td>16-bit unsigned integer</td>
</tr>
<tr>
<td>Sampling Rate (Hz)</td>
<td>48000</td>
</tr>
</tbody>
</table>

Select **OK** to save the graph options.

12. On the plot window, left-click the mouse to move the vertical marker line and observe the frequencies of the different magnitude peaks. Do the peaks occur at the expected frequencies?

**Using Real-time Emulation**

Real-time emulation is a special emulation feature that allows the windows within Code Composer Studio to be updated at up to a 10 Hz rate while the DSP is running. This not only allows graphs and watch windows to update, but also allows the user to change values in watch or memory windows, and have those changes affect the DSP behavior. This is very useful when tuning control law parameters on-the-fly, for example.

13. Reset the CPU, and then enable real-time mode by selecting:

**Debug → Real-time Mode**

14. A message box may appear. Select **YES** to enable debug events. This will set bit 1 (DBGM bit) of status register 1 (ST1) to a “0”. The DBGM is the debug enable mask bit. When the DBGM bit is set to “0”, memory and register values can be passed to the host processor for updating the debugger windows.
15. The memory and graph windows displaying AdcBuf should still be open. The connector wire between PWM1A (pin # P8-9) and ADCINA0 (pin # P9-2) should still be connected. In real-time mode, we would like to have our window continuously refresh. Click:

View → Real-time Refresh Options...

and check “Global Continuous Refresh”. Use the default refresh rate of 100 ms and select OK. Alternately, we could have right clicked on each window individually and selected “Continuous Refresh”.

Note: “Global Continuous Refresh” causes all open windows to refresh at the refresh rate. This can be problematic when a large number of windows are open, as bandwidth over the emulation link is limited. Updating too many windows can cause the refresh frequency to bog down. In that case, either close some windows, or disable global refresh and selectively enable “Continuous Refresh” for individual windows of interest instead.

16. Run the code and watch the windows update in real-time mode. Carefully remove and replace the connector wire from PWM1A (pin # P8-9). Are the values updating as expected?

17. Fully halting the DSP when in real-time mode is a two-step process. First, halt the processor with Debug → Halt. Then uncheck the “Real-time mode” to take the DSP out of real-time mode (Debug → Real-time Mode).

Real-time Mode using GEL Functions

18. Code Composer Studio includes GEL (General Extension Language) functions which automate entering and exiting real-time mode. Four functions are available:

- Run_Realtime_with_Reset (reset DSP, enter real-time mode, run DSP)
- Run_Realtime_with_Restart (restart DSP, enter real-time mode, run DSP)
- Full_Halt (exit real-time mode, halt DSP)
- Full_Halt_with_Reset (exit real-time mode, halt DSP, reset DSP)

These GEL functions can be executed by clicking:

GEL → Realtime Emulation Control → GEL Function

If you would like, try repeating the previous step using the following GEL functions:

GEL → Realtime Emulation Control → Run_Realtime_with_Reset

GEL → Realtime Emulation Control → Full_Halt

Optional Exercise

You might want to experiment with this code by changing some of the values or just modify the code. Try generating another waveform of a different frequency and duty cycle. Also, try to generate complementary pair PWM outputs. Next, try to generate additional simultaneous waveforms by using other ePWM modules. Hint: don’t forget to setup the proper shared I/O pins,
etc. (This optional exercise requires some further working knowledge of the ePWM. Additionally, it may require more time than is allocated for this lab. Therefore, the student may want to try this after the class).

End of Exercise
Flash Programming

Flash Programming Basics

- The DSP CPU itself performs the flash programming
- The CPU executes Flash utility code from RAM that reads the Flash data and writes it into the Flash
- We need to get the Flash utility code and the Flash data into RAM

![Diagram of Flash Programming Basics]

Flash Programming Basics

- Sequence of steps for Flash programming:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Erase</td>
<td>- Set all bits to zero, then to one</td>
</tr>
<tr>
<td>2. Program</td>
<td>- Program selected bits with zero</td>
</tr>
<tr>
<td>3. Verify</td>
<td>- Verify flash contents</td>
</tr>
</tbody>
</table>

- Minimum Erase size is a sector (32Kw or 16Kw)
- Minimum Program size is a bit!
- Important not to lose power during erase step:
  If CSM passwords happen to be all zeros, the CSM will be permanently locked!
- Chance of this happening is quite small! (Erase step is performed sector by sector)
Programming Utilities and CCS Plug-in

Flash Programming Utilities

- Code Composer Studio Plug-in (uses JTAG)
- Third-party JTAG utilities
  - SDFlash JTAG from Spectrum Digital (requires SD emulator)
  - Signum System Flash utilities (requires Signum emulator)
  - BlackHawk Flash utilities (requires Blackhawk emulator)
- SDFlash Serial utility (uses SCI boot)
- Gang Programmers (use GPIO boot)
  - BP Micro programmer
  - Data I/O programmer
- Build your own custom utility
  - Use a different ROM bootloader method than SCI
  - Embed flash programming into your application
  - Flash API algorithms provided by TI

* TI web has links to all utilities (http://www.ti.com/c2000)

Code Composer Studio Flash Plug-In
Code Security Module and Password

**Code Security Module (CSM)**

- Access to the following on-chip memory is restricted:
  - Flash Registers
  - L0 SARAM (4Kw)
  - L1 SARAM (4Kw)
  - L2 SARAM (4Kw)
  - L3 SARAM (4Kw)
  - FLASH (256Kw)
  - 128-Bit Password
  - OTP (1Kw)
  - L0 SARAM (4Kw)
  - L1 SARAM (4Kw)
  - L2 SARAM (4Kw)
  - L3 SARAM (4Kw)
  - 0x00B000
  - 0x3FB000

- Data reads and writes from restricted memory are only allowed for code running from restricted memory
- All other data read/write accesses are blocked:
  - JTAG emulator/debugger, ROM bootloader, code running in external memory or unrestricted internal memory

---

### CSM Password

- Prevents reverse engineering and protects valuable intellectual property

- 128-bit user defined password is stored in Flash
- 128-bit Key Register used to lock and unlock the device
  - Mapped in memory space 0x00 0AE0 – 0x00 0AE7
- 128-bits = 2^{128} = 3.4 \times 10^{38} possible passwords
- To try 1 password every 8 cycles at 150 MHz, it would take at least 5.8 \times 10^{23} years to try all possible combinations!
Flash Programming

CSM Password Match Flow

Start

Flash device secure after reset or runtime

Do dummy reads of PWL 0x33 FFF8 – 0x33 FFFF

Is PWL = all 0s?
Yes → Device permanently locked
No

Is PWL = all Fs?
Yes → Write password to KEY registers 0x00 0AE0 – 0x00 0AE7 (EALLOW) protected
No

Correct password?
Yes → Device unlocked
User can access on-chip secure memory
No
Lab 4: Programming the Flash

Objective

The objective of this lab is to demonstrate the techniques discussed in this module and to program and execute code from the on-chip flash memory. The TMS320F28335 device has been designed for standalone operation in an embedded system. Using the on-chip flash eliminates the need for external non-volatile memory or a host processor from which to bootload. In this lab, the steps required to properly configure the software for execution from internal flash memory will be covered.

Procedure

Project File

1. A project named Lab4.pjt has been created for this lab. Open the project by clicking on Project ➔ Open... and look in C:\C28x\Labs\Lab4. All Build Options have been configured like the previous lab. The files used in this lab are:

   - Adc.c
   - Gpio.c
   - CodeStartBranch.asm
   - Lab_4.cmd
   - DefaultIsr_3_4.c
   - Main_4.c
   - DelayUs.asm
   - PieCtrl.c
   - DSP2833x_GlobalVariableDefs.c
   - PieVect.c
   - DSP2833x_Headers_nonBIOS.cmd
   - SysCtrl.c
   - EPwm.c
   - Watchdog.c
Link Initialized Sections to Flash

Initialized sections, such as code and constants, must contain valid values at device power-up. For a stand-alone embedded system with the F28335 device, these initialized sections must be linked to the on-chip flash memory. Note that a stand-alone embedded system must operate without an emulator or debugger in use, and no host processor is used to perform bootloading.

Each initialized section actually has two addresses associated with it. First, it has a LOAD address which is the address to which it gets loaded at load time (or at flash programming time). Second, it has a RUN address which is the address from which the section is accessed at runtime. The linker assigns both addresses to the section. Most initialized sections can have the same LOAD and RUN address in the flash. However, some initialized sections need to be loaded to flash, but then run from RAM. This is required, for example, if the contents of the section needs to be modified at runtime by the code.

2. Open and inspect the linker command file Lab_4.cmd. Notice that a memory block named FLASH_ABCDEFGH has been created at origin = 0x300000, length = 0x03FF80 on Page 0. This flash memory block length has been selected to avoid conflicts with other required flash memory spaces. See the reference slide at the end of this lab exercise for further details showing the address origins and lengths of the various memory blocks used.

3. In Lab_4.cmd the following compiler sections have been linked to on-chip flash memory block FLASH_ABCDEFGH:

<table>
<thead>
<tr>
<th>Compiler Sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
</tr>
<tr>
<td>.cinit</td>
</tr>
<tr>
<td>.const</td>
</tr>
<tr>
<td>.econst</td>
</tr>
<tr>
<td>.pinit</td>
</tr>
<tr>
<td>.switch</td>
</tr>
</tbody>
</table>

Copying Interrupt Vectors from Flash to RAM

The interrupt vectors must be located in on-chip flash memory and at power-up needs to be copied to the PIE RAM as part of the device initialization procedure. The code that performs this copy is located in InitPieCtrl(). The C-compiler runtime support library contains a memory copy function called memcpy() which will be used to perform the copy.

4. Open and inspect InitPieCtrl() in PieCtrl.c. Notice the memcpy() function used to initialize (copy) the PIE vectors. At the end of the file a structure is used to enable the PIE.
Initializing the Flash Control Registers

The initialization code for the flash control registers cannot execute from the flash memory (since it is changing the flash configuration!). Therefore, the initialization function for the flash control registers must be copied from flash (load address) to RAM (run address) at runtime. The memory copy function `memcpy()` will again be used to perform the copy. The initialization code for the flash control registers `InitFlash()` is located in the `Flash.c` file.

5. Open and inspect `Flash.c`. The C compiler `CODE_SECTION` pragma is used to place the `InitFlash()` function into a linkable section named “secureRamFuncs”.

6. The “secureRamFuncs” section will be linked using the user linker command file `Lab_4.cmd`. Open and inspect `Lab_4.cmd`. The “secureRamFuncs” will load to flash (load address) but will run from `L0123SARAM` (run address). Also notice that the linker has been asked to generate symbols for the load start, load end, and run start addresses.

While not a requirement from a DSP hardware or development tools perspective (since the C28x DSP has a unified memory architecture), historical convention is to link code to program memory space and data to data memory space. Therefore, notice that for the `L0123SARAM` memory we are linking “secureRamFuncs” to, we are specifying “PAGE = 0” (which is program memory).

7. Open and inspect `Main_4.c`. Notice that the memory copy function `memcpy()` is being used to copy the section “secureRamFuncs”, which contains the initialization function for the flash control registers.

8. The following line of code in `main()` is used call the `InitFlash()` function. Since there are no passed parameters or return values the code is just:

   ```c
   InitFlash();
   ```

   at the desired spot in `main()`.

Code Security Module and Passwords

The CSM module provides protection against unwanted copying (i.e. pirating!) of your code from flash, OTP memory, and the L0, L1, L2 and L3 RAM blocks. The CSM uses a 128-bit password made up of 8 individual 16-bit words. They are located in flash at addresses 0x33FFF8 to 0x33FFFF. During this lab, dummy passwords of 0xFFFF will be used – therefore only dummy reads of the password locations are needed to unsecure the CSM. **DO NOT PROGRAM ANY REAL PASSWORDS INTO THE DEVICE.** After development, real passwords are typically placed in the password locations to protect your code. We will not be using real passwords in the workshop.

The CSM module also requires programming values of 0x0000 into flash addresses 0x33FF80 through 0x33FFFF in order to properly secure the CSM. Both tasks will be accomplished using a simple assembly language file `Passwords.asm`. 
9. Open and inspect **Passwords.asm**. This file specifies the desired password values *(DO NOT CHANGE THE VALUES FROM 0xFFFF)* and places them in an initialized section named “passwords”. It also creates an initialized section named “csm_rsrd” which contains all 0x0000 values for locations 0x33FF80 to 0x33FFF5 (length of 0x76).

10. Open Lab_4.cmd and notice that the initialized sections for “passwords” and “csm_rsrd” are linked to memories named PASSWORDS and CSM_RSVD, respectively.

### Executing from Flash after Reset

The F28335 device contains a ROM bootloader that will transfer code execution to the flash after reset. When the boot mode selection pins are set for “Jump to Flash” mode, the bootloader will branch to the instruction located at address 0x33FFF6 in the flash. An instruction that branches to the beginning of your program needs to be placed at this address. Note that the CSM passwords begin at address 0x33FFF8. There are exactly two words available to hold this branch instruction, and not coincidentally, a long branch instruction “LB” in assembly code occupies exactly two words. Generally, the branch instruction will branch to the start of the C-environment initialization routine located in the C-compiler runtime support library. The entry symbol for this routine is _c_int00. Recall that C code cannot be executed until this setup routine is run. Therefore, assembly code must be used for the branch. We are using the assembly code file named **CodeStartBranch.asm**.

11. Open and inspect **CodeStartBranch.asm**. This file creates an initialized section named “codestart” that contains a long branch to the C-environment setup routine. This section has been linked to a block of memory named BEGIN_FLASH.

12. In the earlier lab exercises, the section “codestart” was directed to the memory named BEGIN_M0. Open and inspect Lab_4.cmd and notice that the section “codestart” will now be directed to BEGIN_FLASH. Close the inspected files.

13. The eZdsp™ board needs to be configured for “Jump to Flash” bootmode. Move switch SW1 positions 1, 2, 3 and 4 to the “1” position (all switches to the Left) to accomplish this. Details of switch positions can be found in Appendix A. This switch controls the pullup/down resistor on the GPIO84, GPIO85, GPIO86 and GPIO87 pins, which are the pins sampled by the bootloader to determine the bootmode. (For additional information on configuring the “Jump to Flash” bootmode see the TMS320x2833x DSP Boot ROM Reference Guide, and also the eZdsp F28335 Technical Reference).

### Build – Lab.out

14. At this point we need to build the project, but not have CCS automatically load it since CCS cannot load code into the flash! (the flash must be programmed). On the menu bar click: Option → Customize... and select the “Program/Project CIO” tab. Uncheck “Load Program After Build”.

CCS has a feature that automatically steps over functions without debug information. This can be useful for accelerating the debug process provided that you are not interested
in debugging the function that is being stepped-over. While single-stepping in this lab exercise we do not want to step-over any functions. Therefore, select the “Debug Properties” tab. Uncheck “Step over functions without debug information when source stepping”, then click OK.

15. Click the “Build” button to generate the Lab.out file to be used with the CCS Flash Plug-in.

**CCS Flash Plug-in**

16. Open the Flash Plug-in tool by clicking:

   Tools → F28xx On-Chip Flash Programmer

17. A Clock Configuration window may open. If needed, in the Clock Configuration window set “OSCCLK (MHz):” to 30, “DIVSEL:” to /2, and “PLLCR Value:” to 10. Then click OK. In the next Flash Programmer Settings window confirm that the selected DSP device to program is F28335 and all options have been checked. Click OK.

18. Notice that the eZdsp™ board uses a 30 MHz oscillator (located on the board near LED DS1). Confirm the “Clock Configuration” in the upper left corner has the OSCCLK set to 30 MHz, the DIVSEL set to /2, and the PLLCR value set to 10. Recall that the PLL is divided by two, which gives a SYSCLKOUT of 150 MHz.

19. Confirm that all boxes are checked in the “Erase Sector Selection” area of the plug-in window. We want to erase all the flash sectors.

20. We will not be using the plug-in to program the “Code Security Password”. Do not modify the Code Security Password fields.

21. In the “Operation” block, notice that the “COFF file to Program/Verify” field automatically defaults to the current .out file. Check to be sure that “Erase, Program, Verify” is selected. We will be using the default wait states, as shown on the slide in this module.

22. Click “Execute Operation” to program the flash memory. Watch the programming status update in the plug-in window.

23. After successfully programming the flash memory, close the programmer window.

**Running the Code – Using CCS**

24. In order to effectively debug with CCS, we need to load the symbolic debug information (e.g., symbol and label addresses, source file links, etc.) so that CCS knows where everything is in your code. Click:

   File → Load Symbols → Load Symbols Only...

   and select Lab4.out in the Debug folder.
25. Reset the DSP. The program counter should now be at 0x3FF9A9, which is the start of the bootloader in the Boot ROM.

26. Single-Step <F11> through the bootloader code until you arrive at the beginning of the codestart section in the CodeStartBranch.asm file. (Be patient, it will take about 125 single-steps). Notice that we have placed some code in CodeStartBranch.asm to give an option to first disable the watchdog, if selected.

27. Step a few more times until you reach the start of the C-compiler initialization routine at the symbol _e_int00.

28. Now do Debug → Go Main. The code should stop at the beginning of your main() routine. If you got to that point successfully, it confirms that the flash has been programmed properly, and that the bootloader is properly configured for jump to flash mode, and that the codestart section has been linked to the proper address.

29. You can now RUN the DSP, and you should observe the LED on the board blinking. Try resetting the DSP and hitting RUN (without doing all the stepping and the Go Main procedure). The LED should be blinking again.

Running the Code – Stand-alone Operation (No Emulator)


31. Disconnect the USB cable (emulator) from the eZdsp™ board.

32. Remove the power from the board.

33. Re-connect the power to the board.

34. The LED should be blinking, showing that the code is now running from flash memory.

Return Switch SW1 Back to Default Positions

35. Remove the power from the board.

36. Please return the settings of switch SW1 back to the default positions “Jump to M0 SARAM” bootmode as shown in the table below (see Appendix A for switch position details):

<table>
<thead>
<tr>
<th>Position 4</th>
<th>Position 3</th>
<th>Position 2</th>
<th>Position 1</th>
<th>Boot Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIO87</td>
<td>GPIO86</td>
<td>GPIO85</td>
<td>GPIO84</td>
<td>M0 SARAM</td>
</tr>
<tr>
<td>Right – 0</td>
<td>Left – 1</td>
<td>Right – 0</td>
<td>Right – 0</td>
<td>M0 SARAM</td>
</tr>
</tbody>
</table>

End of Exercise
Lab 4 Reference: Programming the Flash

Flash Memory Section Blocks

<table>
<thead>
<tr>
<th>Section</th>
<th>Origin</th>
<th>Length</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH</td>
<td>0x30 0000</td>
<td>0xFFFF80</td>
<td>0</td>
</tr>
<tr>
<td>CSM_RSVD</td>
<td>0x33 FF80</td>
<td>0x76</td>
<td>0</td>
</tr>
<tr>
<td>BEGIN_FLASH</td>
<td>0x33 FFF6</td>
<td>0x2</td>
<td>0</td>
</tr>
<tr>
<td>PASSWORDS</td>
<td>0x33 FFF8</td>
<td>0x8</td>
<td>0</td>
</tr>
</tbody>
</table>

Lab_4.cmd

```
SECTIONS
{
  codestart  :>  BEGIN_FLASH, PAGE = 0
  passwords  :>  PASSWORDS,   PAGE = 0
  csm_rsvd   :>  CSM_RSVD,      PAGE = 0
}
```

Startup Sequence from Flash Memory

1. RESET
2. BROM vector (32w)
3. Boot Code
   - 0x3F F9A9
4. "rts2800_ml.lib"
5. "user" code sections
   - main ()
      { ...... ...... }

 Origins:
- 0x30 0000
- 0x33 7FF6
- 0x3F F000
- 0x3F FFC0
The Next Step…

Training

**C28x Multi-day Training Course**

- Architectural Overview
- Programming Development Environment
- Peripheral Register Header Files
- Reset and Interrupts
- System Initialization
- Analog-to-Digital Converter
- Control Peripherals
- Numerical Concepts and IQmath
- Direct Memory Access (DMA)
- System Design
- Communications
- DSP/BIOS
- Support Resources

**C2000 Digital Power Supply Workshop**

- Introduction to Digital Power Supply Design
- Driving the Power Stage with PWM Waveforms
- Controlling the Power Stage with Feedback
- Tuning the Loop for Good Transient Response
- Summary and Conclusion

Digital Power Experimenter Kit

Provides hands-on Introduction to Digital Power Concepts
Development Tools

C2000 controlCARDs

- New low cost single-board controllers perfect for initial software development and small volume system builds
- Small form factor (9cm x 2.5cm) with standard 100-pin DIMM interface
  - analog I/O, digital I/O, and JTAG signals available at DIMM interface
- Galvanically isolated RS-232 interface
- Single 5V power supply required (not included)
- Available through TI authorized distributors and on the TI web
  - Part Numbers:
    - TMDCSNCD2808 (100 MHz F2808)
    - TMDCSNCD28044 (100 MHz F28044)
    - TMDCSNCD28335 (150 MHz F28335)

C2000 Experimenter Kits

- Experimenter Kits include
  - F2808 or F28335 controlCARD
  - Docking station (motherboard)
  - C2000 Applications Software CD with example code and full hardware details
  - Code Composer Studio v3.3 with code size limit of 32KB
  - 5V DC power supply
- Docking station features
  - Access to all controlCARD signals
  - Breadboard areas
  - RS-232 an JTAG connectors
- Available through TI authorized distributors and on the TI web
  - Part Numbers:
    - TMDSDOCK2808
    - TMDSDOCK28335
C2000 Peripheral Explorer Kit

- **Experimenter Kit includes**
  - F28335 controlCARD
  - Peripheral Explorer (motherboard)
  - C2000 Applications Software CD with example code and full hardware details
  - Code Composer Studio v3.3 with code size limit of 32KB
  - 5V DC power supply

- **Peripheral Explorer features**
  - ADC input variable resistors
  - GPIO hex encoder & push buttons
  - eCAP infrared sensor
  - GPIO LEDs, I2C & CAN connection
  - Analog I/O (AIC+McBSP)

- **Available through TI authorized distributors and on the TI web**
  - Part Number: TMDSPREX28335

C2000 Digital Power Experimenter Kit

- **DPEK includes**
  - 2-rail DC/DC EVM using TI PowerTrain™ modules (10A)
  - F2808 controlCARD
  - On-board digital multi-meter and active load for transient response tuning
  - C2000 Applications Software CD with example code and full hardware details
  - Digital Power Supply Workshop teaching material and lab software
  - Code Composer Studio v3.3 with code size limit of 32KB
  - 9V DC power supply

- **Available through TI authorized distributors and on the TI web**
  - Part Number: TMDSDCDC2KIT
C2000 DC/DC Developer’s Kit

- DC/DC Kit includes
  - 8-rail DC/DC EVM using TI PowerTrain™ modules (10A)
  - F28044 controlCARD
  - C2000 Applications Software CD with example code and full hardware details
  - Code Composer Studio v3.3 with code size limit of 32KB
  - 9V DC power supply
- Available through TI authorized distributors and on the TI web
  - Part Number: TMDSDCDC8KIT

C2000 AC/DC Developer’s Kit

- AC/DC Kit includes
  - AC/DC EVM with interleaved PFC and phase-shifted full-bridge
  - F2808 controlCARD
  - C2000 Applications Software CD with example code and full hardware details
  - Code Composer Studio v3.3 with code size limit of 32KB
- AC/DC EVM features
  - 12VAC in, 80W/10A output
  - Primary side control
  - Synchronous rectification
  - Peak current mode control
  - Two-phase PFC with current balancing
- Available through TI authorized distributors and on the TI web
  - Part Number: TMDSACDCKIT
# Development Support

## C28x Signal Processing Libraries

<table>
<thead>
<tr>
<th>Signal Processing Libraries &amp; Applications Software</th>
<th>Literature #</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACI3-1: Control with Constant V/Hz</td>
<td>SPRC194</td>
</tr>
<tr>
<td>ACI3-3: Sensored Indirect Flux Vector Control</td>
<td>SPRC207</td>
</tr>
<tr>
<td>ACI3-3: Sensored Indirect Flux Vector Control (simulation)</td>
<td>SPRC208</td>
</tr>
<tr>
<td>ACI3-4: Sensorless Direct Flux Vector Control</td>
<td>SPRC195</td>
</tr>
<tr>
<td>ACI3-4: Sensorless Direct Flux Vector Control (simulation)</td>
<td>SPRC209</td>
</tr>
<tr>
<td>PMSM3-1: Sensored Field Oriented Control using QEP</td>
<td>SPRC210</td>
</tr>
<tr>
<td>PMSM3-2: Sensorless Field Oriented Control</td>
<td>SPRC197</td>
</tr>
<tr>
<td>PMSM3-3: Sensored Field Oriented Control using Resolver</td>
<td>SPRC211</td>
</tr>
<tr>
<td>PMSM3-4: Sensored Position Control using QEP</td>
<td>SPRC212</td>
</tr>
<tr>
<td>BLDTC3-1: Sensored Trapezoidal Control using Hall Sensors</td>
<td>SPRC213</td>
</tr>
<tr>
<td>BLDTC3-2: Sensorless Trapezoidal Drive</td>
<td>SPRC196</td>
</tr>
<tr>
<td>DCMOTOR: Speed &amp; Position Control using QEP without Index</td>
<td>SPRC214</td>
</tr>
<tr>
<td>Digital Motor Control Library (F/C280x)</td>
<td>SPRC215</td>
</tr>
<tr>
<td>Communications Driver Library</td>
<td>SPRC183</td>
</tr>
<tr>
<td>DSP Fast Fourier Transform (FFT) Library</td>
<td>SPRC081</td>
</tr>
<tr>
<td>DSP Filter Library</td>
<td>SPRC082</td>
</tr>
<tr>
<td>DSP Fixed-Point Math Library</td>
<td>SPRC085</td>
</tr>
<tr>
<td>DSP IQ Math Library</td>
<td>SPRC087</td>
</tr>
<tr>
<td>DSP Signal Generator Library</td>
<td>SPRC083</td>
</tr>
<tr>
<td>DSP Software Test Bench (STB) Library</td>
<td>SPRC084</td>
</tr>
<tr>
<td>C28x FPU Fast RTS Library</td>
<td>SPRC664</td>
</tr>
<tr>
<td>C2833x C/C++ Header Files and Peripheral Examples</td>
<td>SPRC350</td>
</tr>
</tbody>
</table>


---

## TI Workshops Download Site

**TI Workshops Download Site**

Login Name: c20001day
Password: tto2


1. F28000 ezDSP 1-day Workshop Labs
2. F28000 ezDSP 1-day Workshop Solutions
3. F28000 ezDSP 1-day Workshop Student Guide
4. F28012 ezDSP 1-day Workshop Labs
5. F28012 ezDSP 1-day Workshop Solutions
6. F28012 ezDSP 1-day Workshop Student Guide
7. F28335 ezDSP 1-day Workshop Labs
8. F28335 ezDSP 1-day Workshop Solutions
9. F28335 ezDSP 1-day Workshop Student Guide
10. LF2407 ezDSP 1-day Workshop Labs and Solutions
11. LF2407 ezDSP 1-day Workshop Student Guide

---

TMS320F28335 One-Day Workshop 77
Customers Are Using C2000™ Products For …

<table>
<thead>
<tr>
<th>Motor Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart/lung machines</td>
</tr>
<tr>
<td>Human transporters</td>
</tr>
<tr>
<td>Industrial drives</td>
</tr>
<tr>
<td>Inverters</td>
</tr>
<tr>
<td>Lawn mowers</td>
</tr>
<tr>
<td>Magnetic bearings</td>
</tr>
<tr>
<td>Mass flow control</td>
</tr>
<tr>
<td>Medical pumps</td>
</tr>
<tr>
<td>Mass control</td>
</tr>
<tr>
<td>Plotters</td>
</tr>
<tr>
<td>Postal sorters</td>
</tr>
<tr>
<td>Power assisted steering</td>
</tr>
<tr>
<td>Power drills</td>
</tr>
<tr>
<td>Power generators</td>
</tr>
<tr>
<td>Printers</td>
</tr>
<tr>
<td>Refrigerators</td>
</tr>
<tr>
<td>Robot control</td>
</tr>
<tr>
<td>Sewing machines</td>
</tr>
<tr>
<td>Ship propulsion control</td>
</tr>
<tr>
<td>Stepper motor control</td>
</tr>
<tr>
<td>Textile machines</td>
</tr>
<tr>
<td>Toy trains</td>
</tr>
<tr>
<td>Treadmills</td>
</tr>
<tr>
<td>Vacuum cleaners</td>
</tr>
<tr>
<td>Vibration control</td>
</tr>
<tr>
<td>Wafer testers</td>
</tr>
<tr>
<td>Washing machines</td>
</tr>
<tr>
<td>Windmill control</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Digital Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battery charging</td>
</tr>
<tr>
<td>Frequency converters</td>
</tr>
<tr>
<td>Fuel cell control</td>
</tr>
<tr>
<td>Server power control</td>
</tr>
<tr>
<td>Solar energy control</td>
</tr>
<tr>
<td>Uninterrupted power supplies</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optical Networking</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEC control</td>
</tr>
<tr>
<td>Optical switch control</td>
</tr>
<tr>
<td>Tunable laser control</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive cruise control</td>
</tr>
<tr>
<td>Airbag control</td>
</tr>
<tr>
<td>Anti-theft systems</td>
</tr>
</tbody>
</table>


For More Information . . .

Internet
Website: http://www.ti.com

FAQ: http://www-k.ext.ti.com/sc/technical_support/knowledgebase.htm
- Device information
- Application notes
- Technical documentation
- my.ti.com
- News and events
- Training

Enroll in Technical Training: http://www.ti.com/sc/training

USA - Product Information Center (PIC)
Phone: 800-477-8924 or 972-644-5580
Email: support@ti.com
- Information and support for all TI Semiconductor products/tools
- Submit suggestions and errata for tools, silicon and documents
### European Product Information Center (EPIC)

**Web:** [http://www-k.ext.ti.com/sc/technical_support/pic/euro.htm](http://www-k.ext.ti.com/sc/technical_support/pic/euro.htm)

<table>
<thead>
<tr>
<th>Phone</th>
<th>Language</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belgium (English)</td>
<td>+32 (0) 27 45 55 32</td>
<td></td>
</tr>
<tr>
<td>France</td>
<td>+33 (0) 1 30 70 11 64</td>
<td></td>
</tr>
<tr>
<td>Germany</td>
<td>+49 (0) 8161 80 33 11</td>
<td></td>
</tr>
<tr>
<td>Israel (English)</td>
<td>1800 949 0107 (free phone)</td>
<td></td>
</tr>
<tr>
<td>Italy</td>
<td>800 79 11 37 (free phone)</td>
<td></td>
</tr>
<tr>
<td>Netherlands (English)</td>
<td>+31 (0) 546 87 95 45</td>
<td></td>
</tr>
<tr>
<td>Spain</td>
<td>+34 902 35 40 28</td>
<td></td>
</tr>
<tr>
<td>Sweden (English)</td>
<td>+46 (0) 8587 555 22</td>
<td></td>
</tr>
<tr>
<td>United Kingdom</td>
<td>+44 (0) 1604 66 33 99</td>
<td></td>
</tr>
<tr>
<td>Finland (English)</td>
<td>+358 (0) 9 25 17 39 48</td>
<td></td>
</tr>
</tbody>
</table>

**Fax:** All Languages +49 (0) 8161 80 2045

**Email:** [epic@ti.com](mailto:epic@ti.com)

- Literature, Sample Requests and Analog EVM Ordering
- Information, Technical and Design support for all Catalog TI Semiconductor products/tools
- Submit suggestions and errata for tools, silicon and documents
Note: This appendix only provides a description of the eZdsp™ F28335 interfaces used in this workshop. For a complete description of all features and details, please see the eZdsp™ F28335 Technical Reference manual.
Module Topics

Appendix A – eZdsp™ F28335

Module Topics........................................................................................................................................... A-2

eZdsp™ F28335 ........................................................................................................................................ A-3
  eZdsp™ F28335 Connector / Header and Pin Diagram........................................................................... A-3
  P2 – Expansion Interface...................................................................................................................... A-5
  P4/P8/P7 – I/O Interface....................................................................................................................... A-6
  P5/P9 – Analog Interface..................................................................................................................... A-8
  P10 – Expansion Interface.................................................................................................................. A-9
  SW1 – Boot Load Option Switch........................................................................................................ A-10
  DS1/DS2 – LEDs ............................................................................................................................... A-11
  TP1/TP2/TP3/TP4 – Test Points ........................................................................................................ A-11
eZdsp™ F28335

eZdsp™ F28335 Connector / Header and Pin Diagram

Figure 1, eZdsp™ F28335 PCB Outline (Top)
Table 1: eZdsp™ F28335 Connectors

<table>
<thead>
<tr>
<th>Connector</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>JTAG Interface</td>
</tr>
<tr>
<td>P2</td>
<td>Expansion</td>
</tr>
<tr>
<td>P4/P8/P7</td>
<td>I/O Interface</td>
</tr>
<tr>
<td>P5/P9</td>
<td>Analog Interface</td>
</tr>
<tr>
<td>P6</td>
<td>Power Connector</td>
</tr>
<tr>
<td>P10</td>
<td>Expansion</td>
</tr>
<tr>
<td>P11</td>
<td>CAN-A</td>
</tr>
<tr>
<td>P12</td>
<td>SCI-A</td>
</tr>
<tr>
<td>J11</td>
<td>CAN-B</td>
</tr>
<tr>
<td>J12</td>
<td>SCI-B</td>
</tr>
<tr>
<td>J201</td>
<td>Embedded JTAG</td>
</tr>
</tbody>
</table>
# P2 – Expansion Interface

The positions of the 60 pins on the P2 connector are shown in the figure below.

![Figure 2: Connector P2 Pin Locations](image)

The definition of P2, which has the I/O signal interface is shown below.

## Table 2: P2, Expansion Interface Connector

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal</th>
<th>Pin #</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+3.3V/+5V/NC *</td>
<td>2</td>
<td>+3.3V/+5V/NC *</td>
</tr>
<tr>
<td>3</td>
<td>GPIO79_XD0</td>
<td>4</td>
<td>GPIO78_XD1</td>
</tr>
<tr>
<td>5</td>
<td>GPIO77_XD2</td>
<td>6</td>
<td>GPIO76_XD3</td>
</tr>
<tr>
<td>7</td>
<td>GPIO75_XD4</td>
<td>8</td>
<td>GPIO74_XD5</td>
</tr>
<tr>
<td>9</td>
<td>GPIO73_XD6</td>
<td>10</td>
<td>GPIO72_XD7</td>
</tr>
<tr>
<td>11</td>
<td>GPIO71_XD8</td>
<td>12</td>
<td>GPIO70_XD9</td>
</tr>
<tr>
<td>13</td>
<td>GPIO69_XD10</td>
<td>14</td>
<td>GPIO68_XD11</td>
</tr>
<tr>
<td>15</td>
<td>GPIO67_XD12</td>
<td>16</td>
<td>GPIO66_XD13</td>
</tr>
<tr>
<td>17</td>
<td>GPIO65_XD14</td>
<td>18</td>
<td>GPIO64_XD15</td>
</tr>
<tr>
<td>19</td>
<td>GPIO40_XAO_XWE1n</td>
<td>20</td>
<td>GPIO41_XA1</td>
</tr>
<tr>
<td>21</td>
<td>GPIO42_XA2</td>
<td>22</td>
<td>GPIO43_XA3</td>
</tr>
<tr>
<td>23</td>
<td>GPIO44_XA4</td>
<td>24</td>
<td>GPIO45_XA5</td>
</tr>
<tr>
<td>25</td>
<td>GPIO46_XA8</td>
<td>26</td>
<td>GPIO47_XA7</td>
</tr>
<tr>
<td>27</td>
<td>GPIO80_XA8</td>
<td>28</td>
<td>GPIO81_XA9</td>
</tr>
<tr>
<td>29</td>
<td>GPIO82_XA10</td>
<td>30</td>
<td>GPIO83_XA11</td>
</tr>
<tr>
<td>31</td>
<td>GPIO84_XA12</td>
<td>32</td>
<td>GPIO85_XA13</td>
</tr>
<tr>
<td>33</td>
<td>GPIO86_XA14</td>
<td>34</td>
<td>GPIO87_XA15</td>
</tr>
<tr>
<td>35</td>
<td>GND</td>
<td>36</td>
<td>GND</td>
</tr>
<tr>
<td>37</td>
<td>GPIO36_SCI_RXD_A-XZCS0n</td>
<td>38</td>
<td>GPIO37_ECAP2-XZCS7n</td>
</tr>
<tr>
<td>39</td>
<td>GPIO34_ECAP1_XREADY</td>
<td>40</td>
<td>B_GPIO32_SCI_RXD_A-XZCS8n</td>
</tr>
<tr>
<td>41</td>
<td>GPIO35_SCI_RXD_A-XRNW</td>
<td>42</td>
<td>10K Pull-up</td>
</tr>
<tr>
<td>43</td>
<td>GPIO38_WE0n</td>
<td>44</td>
<td>XRDn</td>
</tr>
<tr>
<td>45</td>
<td>+3.3V</td>
<td>46</td>
<td>No connect</td>
</tr>
<tr>
<td>47</td>
<td>DSP_RS0n</td>
<td>48</td>
<td>XCLKOUT</td>
</tr>
<tr>
<td>49</td>
<td>GND</td>
<td>50</td>
<td>GND</td>
</tr>
<tr>
<td>51</td>
<td>GND</td>
<td>52</td>
<td>GND</td>
</tr>
<tr>
<td>53</td>
<td>GPIO30_XA16</td>
<td>54</td>
<td>GPIO31_CANTXA_XA17</td>
</tr>
<tr>
<td>55</td>
<td>GPIO30_CANRXA_XA18</td>
<td>56</td>
<td>GPIO14_Tzi3h_XHOLDn_SCI_TXB_MCLKXB</td>
</tr>
<tr>
<td>57</td>
<td>GPIO15_XHOLDAn_SCI_RXD8_MFSXB</td>
<td>58</td>
<td>GPIO29_SCI_TXD_A_XA19</td>
</tr>
<tr>
<td>59</td>
<td>No connect</td>
<td>60</td>
<td>No connect</td>
</tr>
</tbody>
</table>

* Default is No Connect (NC). User can jumper to +3.3V or +5V on backside of eZdsp with JR5.
Appendix

P4/P8/P7 – I/O Interface

The connectors P4, P8, and P7 present the I/O signals from the DSC. The layout of these connectors are shown below.

![Diagram of P4/P8/P7 Connectors](image)

Figure 3, P4/P8/P7 Connectors

The pin definition of the P4 connector is shown in the table below.

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+3.3V/+5V/NC ^</td>
</tr>
<tr>
<td>2</td>
<td>No connect</td>
</tr>
<tr>
<td>3</td>
<td>GPIO22_EGEP1S_MCLKRA_SCITXDB</td>
</tr>
<tr>
<td>4</td>
<td>GPIO27_EPWM4B_MCLKRA_ECAP2</td>
</tr>
<tr>
<td>5</td>
<td>GPIO23_EGEP1_MFSXA_SCIRXDB</td>
</tr>
<tr>
<td>6</td>
<td>GPIO25_EPWM3B_MFSRA_ECAP1</td>
</tr>
<tr>
<td>7</td>
<td>GPIO20_EAEAP1A_MXDA_CANTXB</td>
</tr>
<tr>
<td>8</td>
<td>GPIO21_EAEAP1B_MDRA_CANRXB</td>
</tr>
<tr>
<td>9</td>
<td>No connect</td>
</tr>
<tr>
<td>10</td>
<td>GND</td>
</tr>
<tr>
<td>11</td>
<td>GPIO3_EPWM2B_ECAP5_MCLKRB</td>
</tr>
<tr>
<td>12</td>
<td>GPIO1_EPWM1B_ECAP6_MFSRB</td>
</tr>
<tr>
<td>13</td>
<td>No connect</td>
</tr>
<tr>
<td>14</td>
<td>No connect</td>
</tr>
<tr>
<td>15</td>
<td>No connect</td>
</tr>
<tr>
<td>16</td>
<td>No connect</td>
</tr>
<tr>
<td>17</td>
<td>No connect</td>
</tr>
<tr>
<td>18</td>
<td>GPIO14_TZ3n_XHOLD_SCITXDB_MCLKXB</td>
</tr>
<tr>
<td>19</td>
<td>GPIO15_TZ4n_XHOLDA_SCIRXDB_MFSXB</td>
</tr>
<tr>
<td>20</td>
<td>GND</td>
</tr>
</tbody>
</table>

Table 3: P4, I/O Connectors
## Appendix

### Table 4: P8, I/O Connectors

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal</th>
<th>Pin #</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+3.3V+5V NC *</td>
<td>2</td>
<td>+3.3V+5V NC *</td>
</tr>
<tr>
<td>3</td>
<td>MUX_GPIO28_SCIRXDA_XA19</td>
<td>4</td>
<td>MUX_GPIO28_SCIRXDA_XZCS6n</td>
</tr>
<tr>
<td>5</td>
<td>GPIO14_TZ2n_XHOLD_SCITXD0_MCLKXB</td>
<td>6</td>
<td>GPIO20_EAEPIA_MXDA_CANTXB</td>
</tr>
<tr>
<td>7</td>
<td>GPIO21_EQEP18_MDRA_CANRXB</td>
<td>8</td>
<td>GPIO23_EQEP1_MFSXA_SCIRXDG</td>
</tr>
<tr>
<td>9</td>
<td>GPIO00_EPWM1A</td>
<td>10</td>
<td>GPIO17_EPWM6B/ECAP6/MFSRB</td>
</tr>
<tr>
<td>11</td>
<td>GPIO2_EPWM2A</td>
<td>12</td>
<td>GPIO3_EPWM2B ECAP5_MCLKRB</td>
</tr>
<tr>
<td>13</td>
<td>GPIO4_EPWM3A</td>
<td>14</td>
<td>GPIO5_EPWM3B MFSRA_ECAP1</td>
</tr>
<tr>
<td>15</td>
<td>GPIO27_ECAP4_EQEP2S_MFSXB</td>
<td>16</td>
<td>GPIO6_EPWM4A_EPWMSYNCO/EPWMSYNC1</td>
</tr>
<tr>
<td>17</td>
<td>GPIO13_TZ2n_CANRXB_MDRB</td>
<td>18</td>
<td>GPIO34_ECAP1_XREADY</td>
</tr>
<tr>
<td>19</td>
<td>GND</td>
<td>20</td>
<td>GND</td>
</tr>
<tr>
<td>21</td>
<td>GPIO7_EPWM4B_MCLKRA_ECAP2</td>
<td>22</td>
<td>GPIO15_TZ4n_XHOLD_A_SCI_RXDB_MFSX9</td>
</tr>
<tr>
<td>23</td>
<td>GPIO16_SPISOMOA_CANTXB_T25n</td>
<td>24</td>
<td>GPIO17_SPISOMIA_CANTXB_T26n</td>
</tr>
<tr>
<td>25</td>
<td>GPIO18_SPICLKA_SCITXD0_CANRXA</td>
<td>26</td>
<td>GPIO19_SPISTAN_SCI_RXDB_CANTX0</td>
</tr>
<tr>
<td>27</td>
<td>_MUX_GPIO31_CANRXA_XA17</td>
<td>28</td>
<td>MUX_GPIO30_CANRXA_XA18</td>
</tr>
<tr>
<td>29</td>
<td>MUX_GPIO11_EPWM4B_SCIRXDB_ECAP4</td>
<td>30</td>
<td>MUX_GPIO8_EPWM5A_CANTXB_ADCSOMCAp3</td>
</tr>
<tr>
<td>31</td>
<td>MUX_GPIO09_EPWM5B_SCITXD2_ECAP3</td>
<td>32</td>
<td>MUX_GPIO10_EPWM6A_CANTXB_ADCSOMCAp3</td>
</tr>
<tr>
<td>33</td>
<td>MUX_GPIO22</td>
<td>34</td>
<td>GPIO25_ECAP2_EPEQ2B_MDRB</td>
</tr>
<tr>
<td>35</td>
<td>GPIO26_ECAP2_EPEQ2B_MCLKXB</td>
<td>36</td>
<td>GPIO32_SDAA_EPWM3SYNC1_ADCSOMCAp3</td>
</tr>
<tr>
<td>37</td>
<td>GPIO12_TZ1n_CANTXB_MDXB</td>
<td>38</td>
<td>GPIO33_SCLA_EPWM3SYNC2_ADCSOMCAp3</td>
</tr>
<tr>
<td>39</td>
<td>GND</td>
<td>40</td>
<td>GND</td>
</tr>
</tbody>
</table>

* Default is No Connect (NC). User can jumper to +3.3V or +5V on backside of ezdsp with JR4.

The P7 connector is supplied for backwards compatibility. Signals from other connectors can be wired to this connector to support existing user interfaces. The pin definition of P7 connector is shown in the table below.

### Table 5: P7, I/O Connector

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal</th>
<th>Pin #</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No connect</td>
<td>11</td>
<td>No connect</td>
</tr>
<tr>
<td>2</td>
<td>No connect</td>
<td>12</td>
<td>No connect</td>
</tr>
<tr>
<td>3</td>
<td>No connect</td>
<td>13</td>
<td>No connect</td>
</tr>
<tr>
<td>4</td>
<td>No connect</td>
<td>14</td>
<td>No connect</td>
</tr>
<tr>
<td>5</td>
<td>No connect</td>
<td>15</td>
<td>No connect</td>
</tr>
<tr>
<td>6</td>
<td>No connect</td>
<td>16</td>
<td>No connect</td>
</tr>
<tr>
<td>7</td>
<td>No connect</td>
<td>17</td>
<td>No connect</td>
</tr>
<tr>
<td>8</td>
<td>No connect</td>
<td>18</td>
<td>No connect</td>
</tr>
<tr>
<td>9</td>
<td>No connect</td>
<td>19</td>
<td>No connect</td>
</tr>
<tr>
<td>10</td>
<td>No connect</td>
<td>20</td>
<td>GND</td>
</tr>
</tbody>
</table>
Appendix

P5/P9 – Analog Interface

The position of the 30 pins on the P5/P9 connectors are shown in the diagram below as viewed from the top of the eZdsp.

![Figure 4, Connector P5/P9 Pin Locations](image)

The definition of P5/P9 signals are shown in the table below.

<table>
<thead>
<tr>
<th>P5 Pin #</th>
<th>Signal</th>
<th>P9 Pin #</th>
<th>Signal</th>
<th>P9 Pin #</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ADCINB0</td>
<td>1</td>
<td>GND</td>
<td>2</td>
<td>ADCINA0</td>
</tr>
<tr>
<td>2</td>
<td>ADCINB1</td>
<td>3</td>
<td>GND</td>
<td>4</td>
<td>ADCINA1</td>
</tr>
<tr>
<td>3</td>
<td>ADCINB2</td>
<td>5</td>
<td>GND</td>
<td>6</td>
<td>ADCINA2</td>
</tr>
<tr>
<td>4</td>
<td>ADCINB3</td>
<td>7</td>
<td>GND</td>
<td>8</td>
<td>ADCINA3</td>
</tr>
<tr>
<td>5</td>
<td>ADCINB4</td>
<td>9</td>
<td>GND</td>
<td>10</td>
<td>ADCINA4</td>
</tr>
<tr>
<td>6</td>
<td>ADCINB5</td>
<td>11</td>
<td>GND</td>
<td>12</td>
<td>ADCINA5</td>
</tr>
<tr>
<td>7</td>
<td>ADCINB6</td>
<td>13</td>
<td>GND</td>
<td>14</td>
<td>ADCINA6</td>
</tr>
<tr>
<td>8</td>
<td>ADCINB7</td>
<td>15</td>
<td>GND</td>
<td>16</td>
<td>ADCINA7</td>
</tr>
<tr>
<td>9</td>
<td>ADCREFM</td>
<td>17</td>
<td>GND</td>
<td>18</td>
<td>ADCLO *</td>
</tr>
<tr>
<td>10</td>
<td>ADCREFP</td>
<td>19</td>
<td>GND</td>
<td>20</td>
<td>No connect</td>
</tr>
</tbody>
</table>

* Connect ADCLO to AGND or ADCLO of target system for proper ADC operation.
P10 – Expansion Interface

The positions of the 60 pins on the P10 connector are shown in the figure below.

![Figure 5, Connector P10 Pin Locations]

The definition of P10, which has the I/O signal interface is shown below.

**Table 7: P10, Expansion Interface Connector**

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal</th>
<th>Pin #</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+3.3V/+5V/NC</td>
<td>2</td>
<td>+3.3V/+5V/NC</td>
</tr>
<tr>
<td>3</td>
<td>GPIO33_SCITXDC_XD16</td>
<td>4</td>
<td>GPIO32_SCIRXDC_XD17</td>
</tr>
<tr>
<td>5</td>
<td>GPIO51_MFSRB_XD18</td>
<td>6</td>
<td>GPIO50_MCLKRB_XD19</td>
</tr>
<tr>
<td>7</td>
<td>GPIO59_MFSRA_XD20</td>
<td>8</td>
<td>GPIO58_MCLKRA_XD21</td>
</tr>
<tr>
<td>9</td>
<td>GPIO57_SPISTE_An_XD22</td>
<td>10</td>
<td>GPIO56_SPICLU_XD23</td>
</tr>
<tr>
<td>11</td>
<td>GPIO55_SPISOMIA_XD24</td>
<td>12</td>
<td>GPIO54_SPISIMO_A_XD25</td>
</tr>
<tr>
<td>13</td>
<td>GPIO53_EOEFP1_XD26</td>
<td>14</td>
<td>GPIO52_EOEFP1_S_XD27</td>
</tr>
<tr>
<td>15</td>
<td>GPIO51_EAEP1_B_XD28</td>
<td>16</td>
<td>GPIO50_EOEPT1_A_XD29</td>
</tr>
<tr>
<td>17</td>
<td>GPIO49_ECAPE_XD30</td>
<td>18</td>
<td>GPIO48_ECAPE_XD31</td>
</tr>
<tr>
<td>19</td>
<td>No connect</td>
<td>20</td>
<td>No Connect</td>
</tr>
<tr>
<td>21</td>
<td>No connect</td>
<td>22</td>
<td>No Connect</td>
</tr>
<tr>
<td>23</td>
<td>No connect</td>
<td>24</td>
<td>No Connect</td>
</tr>
<tr>
<td>25</td>
<td>No connect</td>
<td>26</td>
<td>No Connect</td>
</tr>
<tr>
<td>27</td>
<td>No connect</td>
<td>28</td>
<td>No Connect</td>
</tr>
<tr>
<td>29</td>
<td>No connect</td>
<td>30</td>
<td>No Connect</td>
</tr>
<tr>
<td>31</td>
<td>No connect</td>
<td>32</td>
<td>No Connect</td>
</tr>
<tr>
<td>33</td>
<td>No connect</td>
<td>34</td>
<td>No Connect</td>
</tr>
<tr>
<td>35</td>
<td>No connect</td>
<td>36</td>
<td>No Connect</td>
</tr>
<tr>
<td>37</td>
<td>No connect</td>
<td>38</td>
<td>No Connect</td>
</tr>
<tr>
<td>39</td>
<td>No connect</td>
<td>40</td>
<td>No Connect</td>
</tr>
<tr>
<td>41</td>
<td>No connect</td>
<td>42</td>
<td>No Connect</td>
</tr>
<tr>
<td>43</td>
<td>No connect</td>
<td>44</td>
<td>No Connect</td>
</tr>
<tr>
<td>45</td>
<td>No connect</td>
<td>46</td>
<td>No Connect</td>
</tr>
<tr>
<td>47</td>
<td>No connect</td>
<td>48</td>
<td>No Connect</td>
</tr>
<tr>
<td>49</td>
<td>No connect</td>
<td>50</td>
<td>No Connect</td>
</tr>
<tr>
<td>51</td>
<td>No connect</td>
<td>52</td>
<td>No Connect</td>
</tr>
<tr>
<td>53</td>
<td>No connect</td>
<td>54</td>
<td>No Connect</td>
</tr>
<tr>
<td>55</td>
<td>No connect</td>
<td>56</td>
<td>No Connect</td>
</tr>
<tr>
<td>57</td>
<td>No connect</td>
<td>58</td>
<td>No Connect</td>
</tr>
<tr>
<td>59</td>
<td>GND</td>
<td>60</td>
<td>GND</td>
</tr>
</tbody>
</table>
SW1 – Boot Load Option Switch

Switch SW1 is used to select the boot load option used by the F28335 processor on power up. These selections are shown in the table below.

Table 8: SW1, Boot Load Option Switch

<table>
<thead>
<tr>
<th>PIN</th>
<th>Position 4 Boot-3 XA15</th>
<th>Position 3 Boot-2 XA14</th>
<th>Position 2 Boot-1 XA13</th>
<th>Position 1 Boot-0 XA12</th>
<th>Boot Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>Jump to Flash</td>
</tr>
<tr>
<td>1110</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>SCI-A boot</td>
</tr>
<tr>
<td>1101</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>OFF</td>
<td>SPI-A boot *</td>
</tr>
<tr>
<td>1100</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>I²C-A boot</td>
</tr>
<tr>
<td>1011</td>
<td>OFF</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>eCAN-A boot</td>
</tr>
<tr>
<td>1010</td>
<td>OFF</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>McBSP-A boot</td>
</tr>
<tr>
<td>1001</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>Jump to XINTX x16</td>
</tr>
<tr>
<td>1000</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>Jump to XINTX x32</td>
</tr>
<tr>
<td>0111</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>Jump to OTP</td>
</tr>
<tr>
<td>0110</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>Parallel GPIO I/O boot</td>
</tr>
<tr>
<td>0101</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>OFF</td>
<td>Parallel XINTF boot</td>
</tr>
<tr>
<td>0100</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>Jump to SARAM</td>
</tr>
<tr>
<td>0011</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>Branch to check boot mode</td>
</tr>
<tr>
<td>0010</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>Branch to Flash, skip ADC CAL</td>
</tr>
<tr>
<td>0001</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>Branch to SARAM, skip ADC CAL</td>
</tr>
<tr>
<td>0000</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>Branch to SCI, skip ADC CAL</td>
</tr>
</tbody>
</table>

* As shipped from the factory.

The figure below shows the layout of SW1.

![Figure 6, SW1, Boot Load Option Switch (default)](image_url)

<table>
<thead>
<tr>
<th>Position 4 GPIO87</th>
<th>Position 3 GPIO86</th>
<th>Position 2 GPIO85</th>
<th>Position 1 GPIO84</th>
<th>Boot Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right – 0</td>
<td>Left – 1</td>
<td>Right – 0</td>
<td>Right – 0</td>
<td>M0 SARAM</td>
</tr>
<tr>
<td>Left – 1</td>
<td>Left – 1</td>
<td>Left – 1</td>
<td>Left – 1</td>
<td>FLASH</td>
</tr>
</tbody>
</table>
DS1/DS2 – LEDs

The eZdspTM F28335 has three light-emitting diodes. DS1 indicates the presence of +5 volts and is normally ‘on’ when power is applied to the board. LED DS2 is under control of the GPIO32 line from the processor. DS201 is connected to the embedded USB emulator and shows the status of the emulation link. These are shown in the table below.

<table>
<thead>
<tr>
<th>LED #</th>
<th>Color</th>
<th>Controlling Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS1</td>
<td>Green</td>
<td>+5 Volts</td>
</tr>
<tr>
<td>DS2</td>
<td>Green</td>
<td>GPIO32</td>
</tr>
<tr>
<td>DS201</td>
<td>Green</td>
<td>Embedded emulation link status</td>
</tr>
</tbody>
</table>

TP1/TP2/TP3/TP4 – Test Points

<table>
<thead>
<tr>
<th>Test Point</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP1</td>
<td>AGND</td>
</tr>
<tr>
<td>TP2</td>
<td>XCLKOUT</td>
</tr>
<tr>
<td>TP3</td>
<td>U8(DSP) Pin 81, TEST1</td>
</tr>
<tr>
<td>TP4</td>
<td>U8(DSP) Pin 82, TEST2</td>
</tr>
</tbody>
</table>