

F28M35x Workshop

Workshop Guide and Lab Manual



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 © 2012 – 2014 Texas Instruments Incorporated

Revision History

December 2012 – Revision 1.0

May 2013 – Revision 2.0

March 2014 – Revision 3.0

Mailing Address

Texas Instruments
Training Technical Organization
6500 Chase Oaks Blvd Building 2
M/S 8437
Plano, Texas 75023

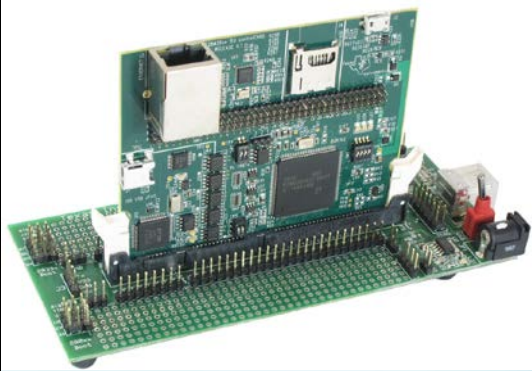
Workshop Topics

<i>Workshop Topics</i>	3
<i>Workshop Introduction</i>	5
Outline	5
Required Workshop Materials	6
The F28M35x	6
F28M35H52C1 Experimenter's Kit	8
F28M35x Functional Block Diagram	9
<i>Programming Development Environment</i>	10
Programming Model	10
Code Composer Studio	11
Software Development and COFF Concepts	11
Edit and Debug Perspective	13
Target Configuration	14
CCS Project and Build Options	15
CCSv5 Debug Environment	18
Dual Subsystem Debug	20
Lab File Directory Structure	21
<i>Lab 1: Dual Core Debug with F28M35x</i>	22
<i>Reset and System Initialization</i>	35
Device Power and Reset	35
Master and Control Subsystem Boot	36
System Clock / PLL	38
GPIO Multiplexer	42
<i>Architectural Overview – Master M3 Subsystem</i>	46
Architectural Block Diagram	46
Memory System	47
Interrupt System	53
External Peripheral Interface (EPI)	56
M3 Serial Communication	57
<i>Lab 2: Generating a Master M3 Subsystem Interrupt</i>	63
<i>Architectural Overview – Control C28 Subsystem</i>	66
Architectural Block Diagram	66
Memory System	67
Interrupt System	71
Viterbi, Complex Math, CRC Unit (VCU)	72
ePWM Module	73
<i>Lab 3: Generating a Control C28 Subsystem Interrupt</i>	75
<i>Analog Subsystem</i>	79
Module Block Diagram	80
Input Pin Connections	81
Conversion Trigger	82
Conversion Priority	83
Interrupts	86
Clock Configuration	86
Analog Comparators	87
<i>Lab 4: Capture ePWM waveform with ADC</i>	89
<i>Inter-Processor Communications (IPC)</i>	97
Shared SARAM and Message SARAM	97

Interrupts and Flags	99
IPC Data Transfer	100
<i>Lab 5: Data Transfer using IPC</i>	<i>102</i>
<i>Safety Features.....</i>	<i>107</i>
Self Test Controller (STC) / Memory BIST.....	107
Access Protection - Shared / Dedicated Memory	108
Error Detection and Correction	109
Register Protection.....	111
Peripheral and Communication Verification.....	111
Missing Clock Detection.....	112
<i>Other Resources – The Next Step... ..</i>	<i>113</i>
C2000 MCU Multi-day Training Course	113
controlSUITE™	113
Experimenter's Kits	114
Peripheral Explorer Kit	114
controlSTICK Evaluation Tool.....	115
LaunchPad Evaluation Kit	115
Application Kits.....	116
C2000 Workshop Download Wiki	116
For More Information... ..	117
<i>Appendix – F28M35xx Experimenter's Kit.....</i>	<i>118</i>
F28M35xx controlCARD	119
Docking Station	123
<i>Notes:.....</i>	<i>126</i>

Workshop Introduction

F28M35x Workshop



Technical Training Organization

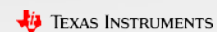
C2000 is a trademark of Texas Instruments.
Copyright © 2014 Texas Instruments. All rights reserved.



Outline

Outline

- ◆ **Workshop Introduction**
- ◆ **Programming Development Environment**
 - ◆ Lab 1: Using Code Composer Studio with the F28M35x
- ◆ **Reset and System Initialization**
- ◆ **Architectural Overview - Master M3 Subsystem**
 - ◆ Lab 2: Generating a Master M3 Subsystem Interrupt
- ◆ **Architectural Overview - Control C28 Subsystem**
 - ◆ Lab 3: Generating a Control C28 Subsystem Interrupt
- ◆ **Analog Subsystem**
 - ◆ Lab 4: Capture ePWM waveform with ADC
- ◆ **Inter-Processor Communications (IPC)**
 - ◆ Lab 5: Data Transfer using Inter-Processor Communications
- ◆ **Safety Features**
- ◆ **Other Resources - The Next Step...**



Required Workshop Materials

Required Workshop Materials

- ◆ http://processors.wiki.ti.com/index.php/F28M35x_Workshop
- ◆ F28M35H52C1 Experimenter's Kit
- ◆ Install Code Composer Studio v5.5.0
- ◆ Run the workshop installer
 - F28M35x Workshop-3.0-Setup.exe*
 - ◆ Lab Files / Solution Files
 - ◆ Student Guide and Documentation

The F28M35x

F28M35x - Best of Both Worlds

Real-Time Control



Connectivity



- ◆ Single Device Combining a Real-Time Control Subsystem with a Master MCU Subsystem
 - ◆ TI 32-bit F28x with Floating-Point Unit
 - ◆ ARM® 32-bit Cortex™ M3 MCU
- ◆ Optimized subsystems reduce both hardware and software complexity
- ◆ Fast inter-processor communication reduces latency
- ◆ Improved Safety – EDAC, Redundancy, and Security

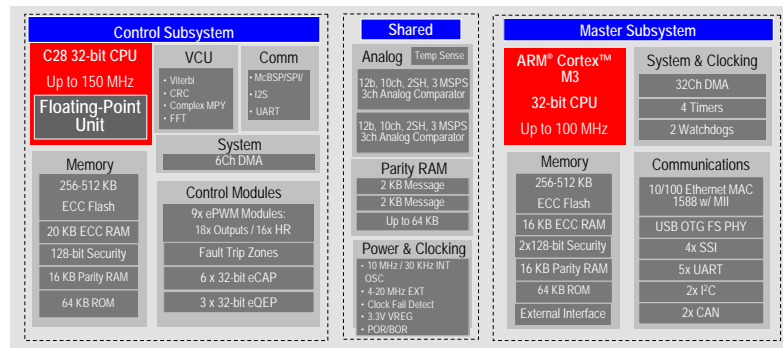
F28M35x = Control + Connectivity

Control Subsystem

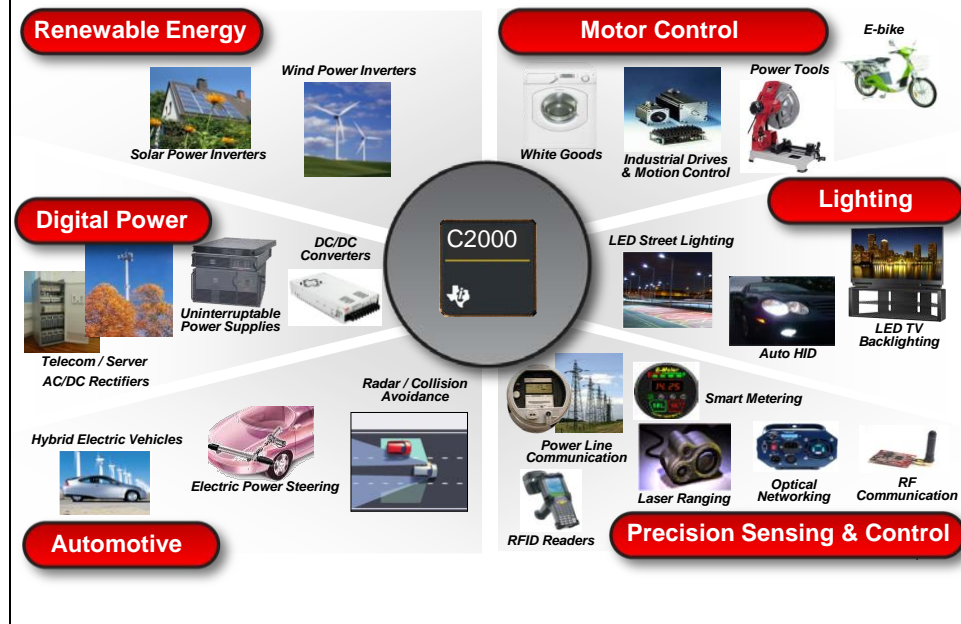
- Precision Control
- Industry leading computational performance
 - Expanded instruction set
 - Industry's highest-resolution PWMs
 - Low-latency control loops
 - Real-world, modular control software
 - High-speed precision analog
 - Fine-tuned control architecture

Master Subsystem

- Ecosystem for Developers
- Operating System
 - Middleware
 - SW Infrastructure
- Robust Communications
- Ethernet
 - CAN
 - Fieldbus
 - Serial
 - USB
- Additional functions
- Natural user interface
 - Motion profile
 - Safety



Broad C2000™ Control Application Base



F28M35H52C1 Experimenter's Kit

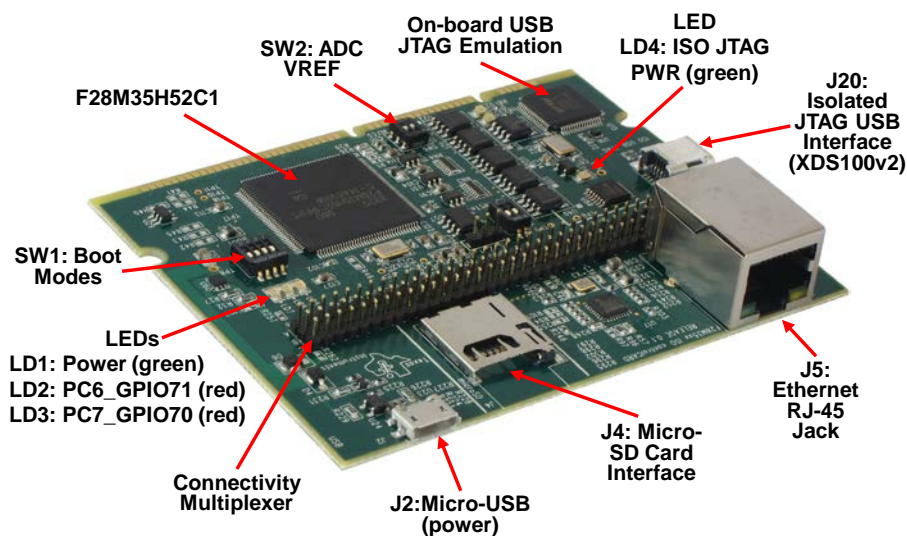
F28M35H52C1 Experimenter's Kit



Part Number: TMDXDOCKH52C1

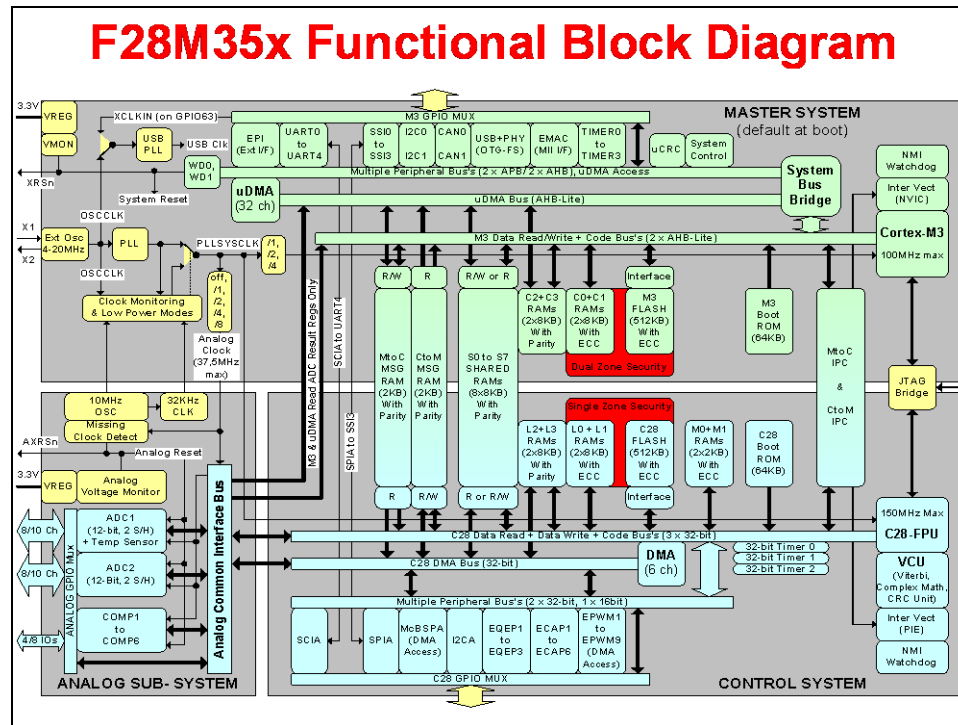
- ◆ **Experimenter's Kit includes:**
 - ◆ F28M35H52C1 controlCARD
 - ◆ USB docking station
 - ◆ USB and Ethernet cables
 - ◆ Code Composer Studio
- ◆ **Features:**
 - ◆ No external emulator required
 - ◆ Fully powered from USB connection; no external power supply needed
 - ◆ Access to controlCARD signals
 - ◆ Breadboard area
 - ◆ Standard 100-pin DIMM Interface
 - ◆ Analog I/O, digital I/O and JTAG signals at DIMM interface
 - ◆ Full open source software examples and hardware files for download in controlSUITE
- ◆ Available through TI authorized distributors and the TI eStore

F28M35H52C1 controlCARD



Part Number: TMDXCNCDS52C1

F28M35x Functional Block Diagram

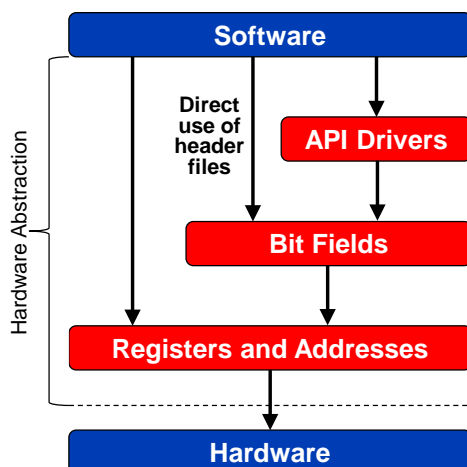


Programming Development Environment

Programming Model

Programming Model: Header Files & API

- ◆ Master M3 → API Drivers
- ◆ Control C28 → Header Files



- ◆ **API Drivers**
 - ◆ C functions automatically set register bit fields
 - ◆ Common tasks and peripheral modes supported
 - ◆ Reduces learning curve and simplifies programming
- ◆ **Bit Fields**
 - ◆ C structures – Peripheral Register Header Files
 - ◆ Register access whole or by bits and bit fields are manipulated without masking
 - ◆ Ease-of-use with CCS IDE
- ◆ **Registers and Addresses**
 - ◆ Assembly language used to program hardware registers and access addresses

Programming Model: Comparison

Direct Register Access

```

//Interrupts set up elsewhere
//Set duty cycle
MOVB    @9,#0x0F,UNC

//Set PWM1A on Zero Event
AND     AL,@11,#0xFFFC
ORB     AL,#0x02
MOV     @11,AL

//Clear PWM1A on Up-count
//CompareA event
AND     AL,@11,#0xFFCF
ORB     AL,#0x10
MOV     @11,AL

```

Bit Field Header Files

```

interrupt void IsrAdc(void)
{
    //Period of ePWM1 is set in
    //init; Multiply period by
    //desired duty to get CMPA
    //value;
    EPwm1Regs.CMPA.half.CMPA =
        EPwm1Regs.TBPRD * duty;
}

```

API Driver

```

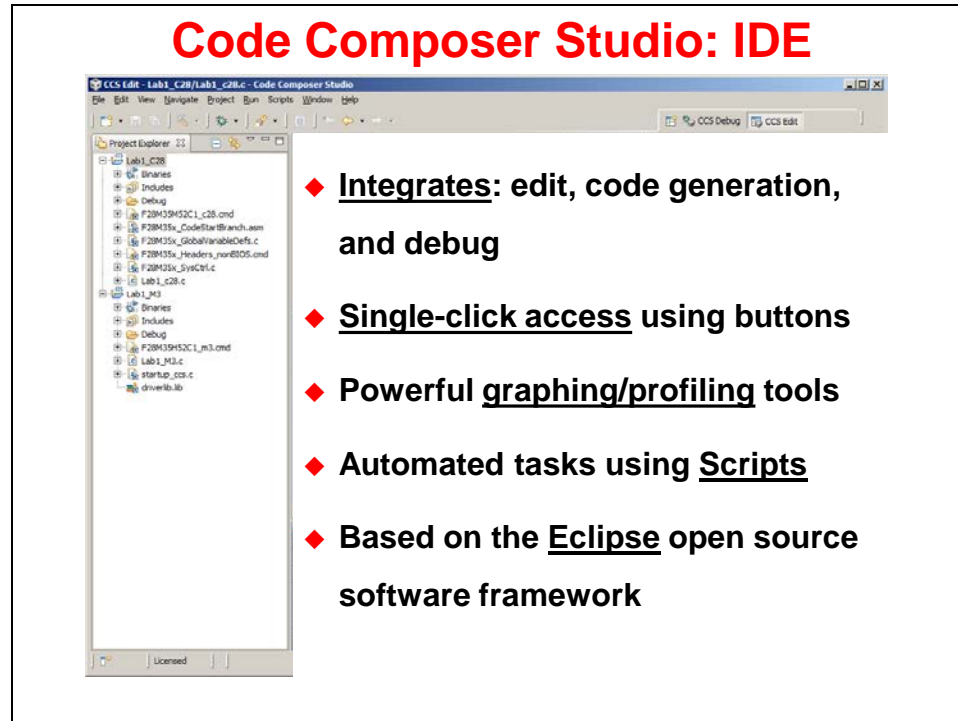
interrupt void IsrAdc(void)
{
    /* set a new pwm value */
    PWM_setDutyA(PWM_MODULE_2,
        duty);
}

```

- ◆ Device support package includes documentation and examples showing how to use the Bit Field Header Files or API Driver Library
- ◆ Device support packages located at:
C:\TI\controlSUITE\device_support\
- ◆ controlSUITE located at <http://www.ti.com> and enter “controlSUITE” in the keyword search box

Code Composer Studio

Code Composer Studio™ (CCS) is an integrated development environment (IDE) for Texas Instruments (TI) embedded processor families. CCS comprises a suite of tools used to develop and debug embedded applications. It includes compilers for each of TI's device families, source code editor, project build environment, debugger, profiler, simulators, real-time operating system and many other features. The intuitive IDE provides a single user interface taking you through each step of the application development flow. Familiar tools and interfaces allow users to get started faster than ever before and add functionality to their application thanks to sophisticated productivity tools.

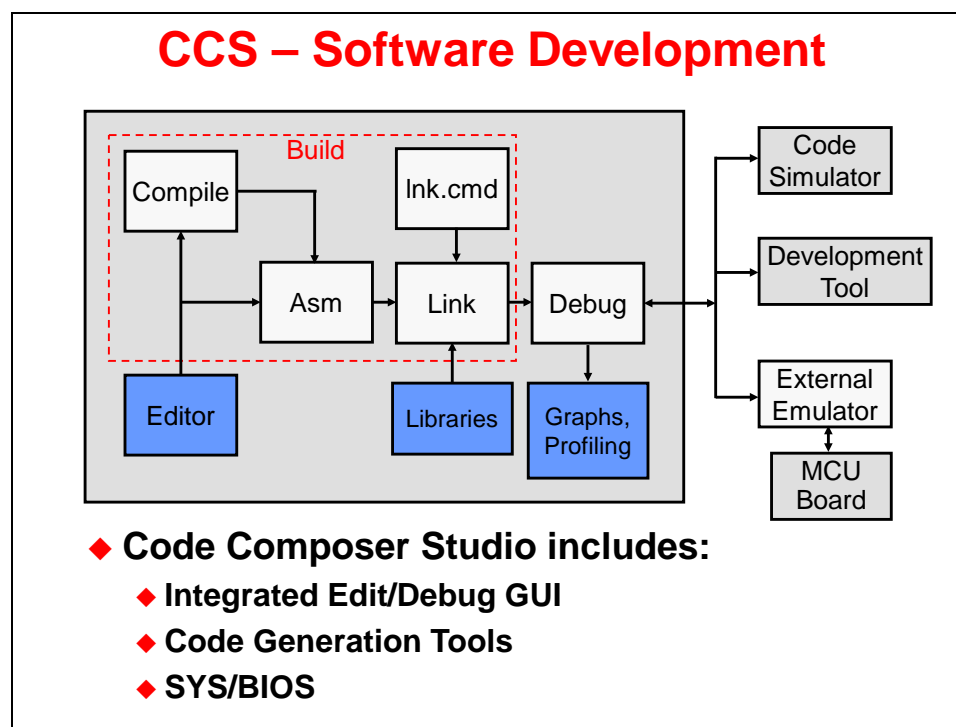


CCS is based on the Eclipse open source software framework. The Eclipse software framework was originally developed as an open framework for creating development tools. Eclipse offers an excellent software framework for building software development environments and it is becoming a standard framework used by many embedded software vendors. CCS combines the advantages of the Eclipse software framework with advanced embedded debug capabilities from TI resulting in a compelling feature-rich development environment for embedded developers. CCS supports running on both Windows and Linux PCs. Note that not all features or devices are supported on Linux.

Software Development and COFF Concepts

In an effort to standardize the software development process, TI uses the Common Object File Format (COFF). COFF has several features which make it a powerful software development system. It is most useful when the development task is split between several programmers.

Each file of code, called a *module*, may be written independently, including the specification of all resources necessary for the proper operation of the module. Modules can be written using CCS or any text editor capable of providing a simple ASCII file output. The expected extension of a source file is *.ASM* for *assembly* and *.C* for *C programs*.



CCS includes a built-in editor, compiler, assembler, linker, and an automatic build process. Additionally, tools to connect file input and output, as well as built-in graph displays for output are available. Other features can be added using the plug-ins capability

Numerous modules are joined to form a complete program by using the *linker*. The *linker* efficiently allocates the resources available on the device to each module in the system. The linker uses a command (.CMD) file to identify the memory resources and placement of where the various sections within each module are to go. Outputs of the linking process includes the linked object file (.OUT), which runs on the device, and can include a .MAP file which identifies where each linked section is located.

The high level of modularity and portability resulting from this system simplifies the processes of verification, debug and maintenance. The process of COFF development is presented in greater detail in the following paragraphs.

The concept of COFF tools is to allow modular development of software independent of hardware concerns. An individual assembly language file is written to perform a single task and may be linked with several other tasks to achieve a more complex total system.

Writing code in modular form permits code to be developed by several people working in parallel so the development cycle is shortened. Debugging and upgrading code is faster, since components of the system, rather than the entire system, is being operated upon. Also, new systems may be developed more rapidly if previously developed modules can be used in them.

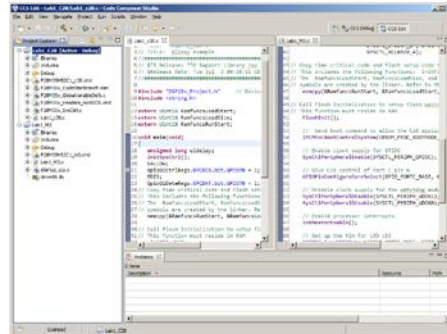
Code developed independently of hardware concerns increases the benefits of modularity by allowing the programmer to focus on the code and not waste time managing memory and moving code as other code components grow or shrink. A linker is invoked to allocate systems hardware to the modules desired to build a system. Changes in any or all modules, when re-linked, create a new hardware allocation, avoiding the possibility of memory resource conflicts.

Edit and Debug Perspective

A perspective defines the initial layout views of the workbench windows, toolbars, and menus that are appropriate for a specific type of task, such as code development or debugging. This minimizes clutter to the user interface.

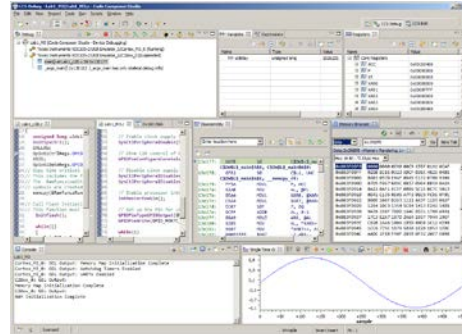
Edit and Debug Perspective

- ◆ Each perspective provides a set of functionality aimed at accomplishing a specific task



◆ Edit Perspective

- ◆ Displays views used during code development
 - ◆ C/C++ project, editor, etc.



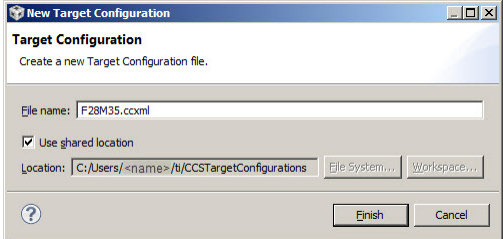
◆ Debug Perspective

- ◆ Displays views used for debugging
 - ◆ Menus and toolbars associated with debugging, watch and memory windows, graphs, etc.

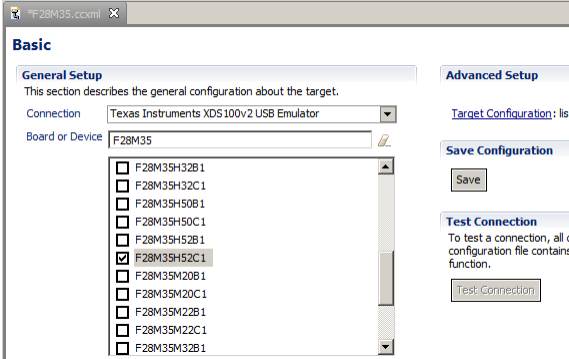
Target Configuration

A Target Configuration tells CCS how to connect to the device. It describes the device using GEL files and device configuration files. The configuration files are XML files and have a *.ccxml extension.

Creating a Target Configuration



◆ **File → New → Target Configuration File**



◆ **Select connection type**

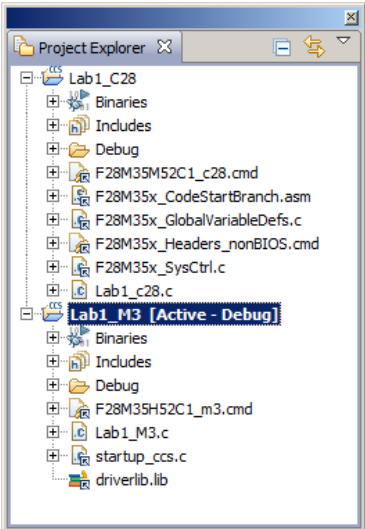
◆ **Select device**

◆ **Save configuration**

CCS Project and Build Options

CCS works with a *project* paradigm. Essentially, within CCS you create a project for each executable program you wish to create. Projects store all the information required to build the executable. For example, it lists things like: the source files, the header files, the target system's memory-map, and program build options.

CCSv5 Project



Project files contain:

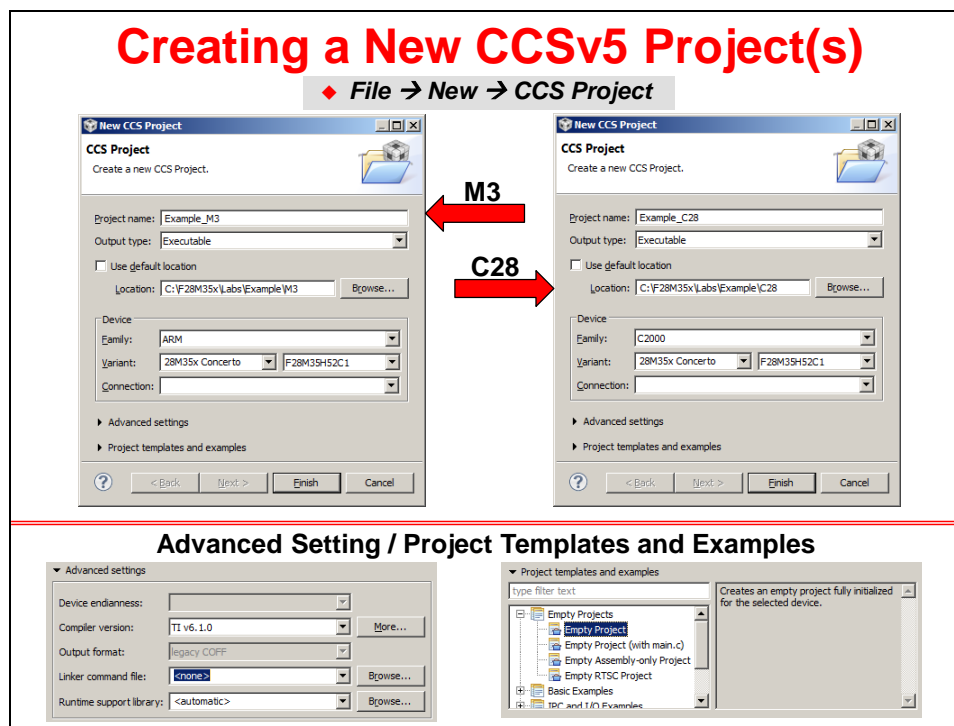
- ◆ **List of files:**
 - ◆ Source (C, assembly)
 - ◆ Libraries
 - ◆ SYS/BIOS configuration file
 - ◆ Linker command files
- ◆ **Project settings:**
 - ◆ Build options (compiler, assembler, linker, and SYS/BIOS)
 - ◆ Build configurations

To create a new project, you need to select the following menu items:

File → New → CCS Project

Along with the main Project menu, you can also manage open projects using the right-click popup menu. Either of these menus allows you to modify a project, such as add files to a project, or open the properties of a project to set the build options.

A graphical user interface (GUI) is used to assist in creating a new project. The GUI is shown in the slide below.



Project options direct the code generation tools (i.e. compiler, assembler, linker) to create code according to your system's needs. When you create a new project, CCS creates two sets of build options – called configurations: one called *Debug*, the other *Release* (you might think of as optimize).

To make it easier to choose build options, CCS provides a graphical user interface (GUI) for the various compiler and linker options. The following slide is a sample of the configuration options.

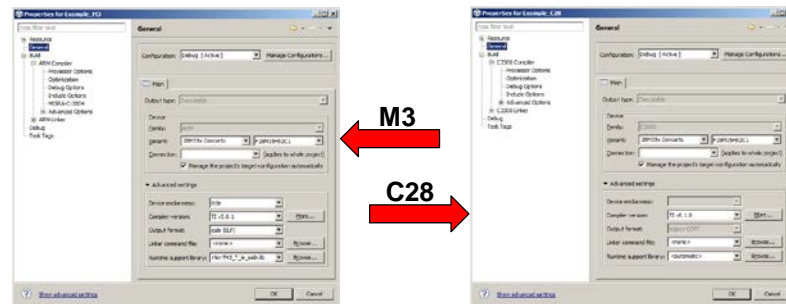
There is a one-to-one relationship between the items in the text box on the main page and the GUI check and drop-down box selections. Once you have mastered the various options, you can probably find yourself just typing in the options.

There are many linker options but these four handle all of the basic needs.

- `-o <filename>` specifies the output (executable) filename.
- `-m <filename>` creates a map file. This file reports the linker's results.
- `-c` tells the compiler to autoinitialize your global and static variables.
- `-x` tells the compiler to exhaustively read the libraries. Without this option libraries are searched only once, and therefore backwards references may not be resolved.

To help make sense of the many compiler options, TI provides two default sets of options (configurations) in each new project you create. The Release (optimized) configuration invokes the optimizer with `-o3` and disables source-level, symbolic debugging by omitting `-g` (which disables some optimizations to enable debug).

CCSv5 Build Options – Compiler / Linker



- ◆ Separate build options for each project – M3 and C28
- ◆ Compiler
 - ◆ Categories for code generation tools – controls many aspects of the build process, such as:
 - ◆ Optimization level
 - ◆ Target device
 - ◆ Compiler / assembly / link options
- ◆ Linker
 - ◆ Categories for linking – specify various link options
 - ◆ `${PROJECT_ROOT}` specifies the current project directory

Compiler Support for F28M35x

- ◆ F28M35x devices require Code Composer Studio version 4.2.x or later
- ◆ Devices have a two different cores; *different compilers must be used for each core*:
 - ◆ TMS470 Code Generation Tools
 - ◆ Used for ARM® Cortex™ M3 Master Subsystem
 - ◆ Version 5.0.1 or later
 - ◆ C2000 Code Generation Tools
 - ◆ Used for TI C28 Control Subsystem
 - ◆ Version 6.1.0 or later





CCSv5 Debug Environment

The basic buttons that control the debug environment are located in the top of CCS:



The common debugging and program execution descriptions are shown below:

Start debugging


Image	Name	Description	Availability
	New Target Configuration	Creates a new target configuration file.	File New Menu Target Menu
	Debug	Opens a dialog to modify existing debug configurations. Its drop down can be used to access other launching options.	Debug Toolbar Target Menu
	Connect Target	Connect to hardware targets.	TI Debug Toolbar Target Menu Debug View Context Menu
	Terminate All	Terminates all active debug sessions.	Target Menu Debug View Toolbar

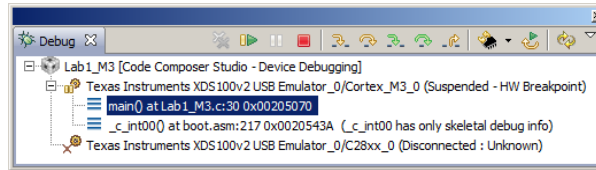
Program execution

Image	Name	Description	Availability
	Halt	Halts the selected target. The rest of the debug views will update automatically with most recent target data.	Target Menu Debug View Toolbar
	Run	Resumes the execution of the currently loaded program from the current PC location. Execution continues until a breakpoint is encountered.	Target Menu Debug View Toolbar
	Run to Line	Resumes the execution of the currently loaded program from the current PC location. Execution continues until the specific source/assembly line is reached.	Target Menu Disassembly Context Menu Source Editor Context Menu
	Go to Main	Runs the programs until the beginning of function main is reached.	Debug View Toolbar
	Step Into	Steps into the highlighted statement.	Target Menu Debug View Toolbar
	Step Over	Steps over the highlighted statement. Execution will continue at the next line either in the same method or (if you are at the end of a method) it will continue in the method from which the current method was called. The cursor jumps to the declaration of the method and selects this line.	Target Menu Debug View Toolbar
	Step Return	Steps out of the current method.	Target Menu Debug View Toolbar
	Reset	Resets the selected target. The drop-down menu has various advanced reset options, depending on the selected device.	Target Menu Debug View Toolbar
	Restart	Restores the PC to the entry point for the currently loaded program. If the debugger option "Run to main on target load or restart" is set the target will run to the specified symbol, otherwise the execution state of the target is not changed.	Target Menu Debug View Toolbar
	Assembly Step Into	The debugger executes the next assembly instruction, whether source is available or not.	TI Explicit Stepping Toolbar Target Advanced Menu
	Assembly Step Over	The debugger steps over a single assembly instruction. If the instruction is an assembly subroutine, the debugger executes the assembly subroutine and then halts after the assembly function returns.	TI Explicit Stepping Toolbar Target Advanced Menu

Dual Subsystem Debug

Launching Dual Subsystem Debug (1)

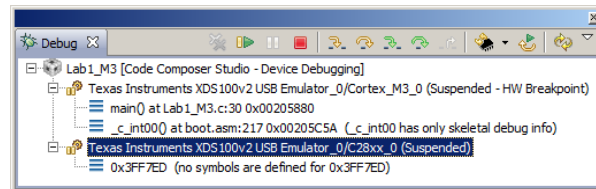
- ◆ **1st subsystem** (CCS Edit Perspective) -
 - ◆ Clicking “Debug” button  will automatically:
 - ◆ Launch the debugger
 - ◆ Connects to target
 - ◆ Programs flash memory



- ◆ Note 2nd subsystem is disconnected
- ◆ Next step will connect 2nd subsystem

Launching Dual Subsystem Debug (2)

- ◆ **2nd subsystem** (CCS Debug Perspective) -
 - ◆ In Debug window right-click on emulator and select “Connect target”
 - ◆ Highlight emulator and load program (flash)
 - ◆ Run → Load → Load Program...



- ◆ Both subsystems are connected
- ◆ Next step is dual subsystem start-up sequence

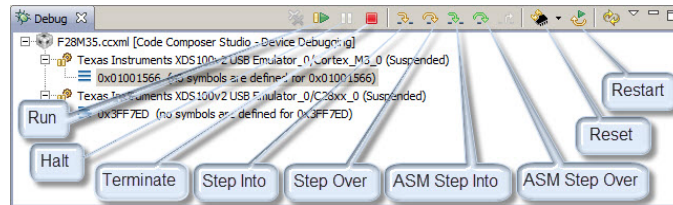
Dual Subsystem Debug Start-up

◆ Start-up sequence

1. Reset C28 subsystem
2. Reset M3 subsystem
3. Run C28 subsystem
4. Run M3 subsystem
5. Stop and debug either subsystem

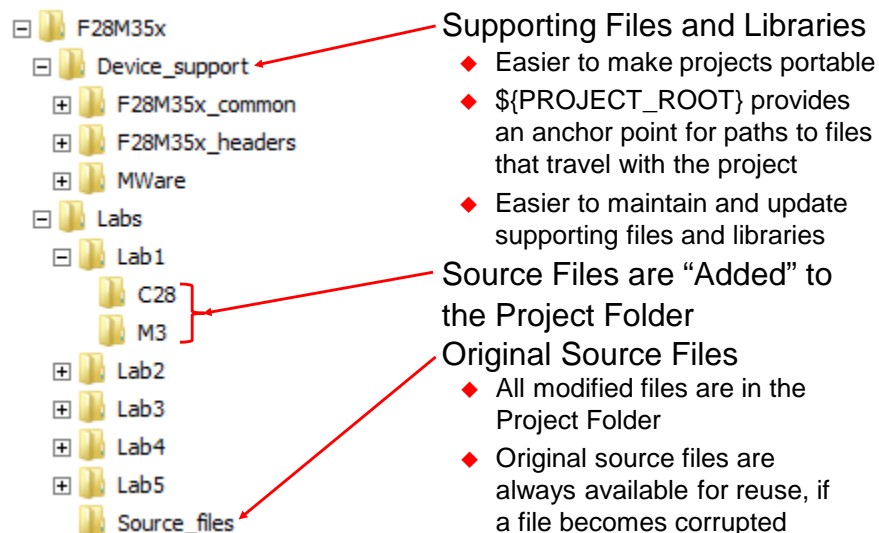
◆ Debug window controls “selected” subsystem for the debug interaction

◆ Highlight appropriate subsystem for debug



Lab File Directory Structure

Lab File Directory Structure



Note: CCSv5 will automatically add ALL files contained in the folder where the project is created

Lab 1: Dual Core Debug with F28M35x

➤ Objective

The objective of this lab exercise is to become familiar with the Code Composer Studio (CCS) development environment while using an F28M35x device (i.e. dual subsystem). Details on setting up the target configuration, creating a new project, setting build options, and connecting to the subsystems will be explained. A typical F28M35x application consists of two separate and completely independent CCS projects without any linking between them. One project is for the Master subsystem (M3), and the other project is for the Control subsystem (C28). A project contains all the files needed to develop an executable output file (.out) which can be run on the F28M35x device. In this lab exercise we will have the Master subsystem blink LED LD3 and the Control subsystem blink LED LD2.

Lab1: Dual Core Debug with F28M35x



◆ Use Code Composer Studio (CCS) in dual subsystem debug environment

- ◆ Set up target configuration
- ◆ Create M3 project
 - ◆ Master subsystem blinks LED LD3 (software delay loop)
- ◆ Create C28 project
 - ◆ Control subsystem blinks LED LD2 (software delay loop)

➤ Initial Hardware Set Up

Boot Mode Switch Settings

Note: The F28M35xx ISO controlCARD boot mode must be configured to “Boot from M3 FLASH” for the workshop lab exercises. Set the “BOOT” controlCARD switch SW1 as shown in the following table (see the Appendix for switch and jumper position details).

Position 1 / PG2_GPIO34	Position 2 / PG3_GPIO35	Position 3 / PG7_GPIO47	Position 4 / PG3_GPIO43	Boot Mode
Down – 1	Down – 1	Down – 1	Down – 1	M3 FLASH

Connect USB cables to controlCARD

Using the two (2) supplied USB cables, plug the standard USB connectors into the computer USB ports and the other ends to the controlCARD as follows:

- The mini USB connector to J20 – isolated XDS100V2 JTAG emulation
- The micro USB connector to J2 – board power

➤ Initial Software Set Up

Code Composer Studio must be installed in addition to the workshop files. A local copy of the required *controlSUITE* files is included with the lab files. This provides portability, making the workshop files self-contained and independent of other support files or resources. The lab directions for this workshop are based on all software installed in their default locations.

➤ Procedure

Start Code Composer Studio and Open a Workspace

1. Start Code Composer Studio (CCS) by double clicking the icon on the desktop or selecting it from the Windows Start menu. When CCS loads, a dialog box will prompt you for the location of a workspace folder. Use the default location for the workspace and click **OK**.

This folder contains all CCS custom settings, which includes project settings and views when CCS is closed so that the same projects and settings will be available when CCS is opened again. The workspace is saved automatically when CCS is closed.

2. The first time CCS opens a “Welcome to Code Composer Studio v5” page appears. Close the page by clicking the **x** on the “TI Resource Explorer” tab. You should now have an empty workbench. The term workbench refers to the desktop development environment. Maximize CCS to fill your screen.

The workbench will open in the “CCS Edit” perspective view. Notice the CCS Edit icon in the upper right-hand corner. A perspective defines the initial layout views of the workbench windows, toolbars, and menus which are appropriate for a specific type of task (i.e. code development or debugging). This minimizes clutter to the user interface. The “CCS Edit” perspective is used to create or build C/C++ projects. A “CCS Debug” perspective view will automatically be enabled when the debug session is started. This perspective is used for debugging C/C++ projects.

Setup Target Configuration

3. Open the emulator target configuration dialog box. On the menu bar click:

File → New → Target Configuration File

In the file name field type **F28M35.ccxml**. This is just a descriptive name since multiple target configuration files can be created. Leave the “Use shared location” box checked and select **Finish**.

4. In the next window that appears, select the emulator using the “Connection” pull-down list and choose “Texas Instruments XDS100v2 USB Emulator”. In the “Board or Device” box type **F28M35** to filter the options. In the box below, check the box to select “F28M35H52C1”. Click **Save** to save the configuration, then close the “F28M35.ccxml” setup window by clicking the **x** on the tab.
5. To view the target configurations, click:

View → Target Configurations

and click the plus sign (+) to the left of “User Defined”. Notice that the F28M35.ccxml file is listed and set as the default. If it is not set as the default, right-click on the .ccxml file and select “Set as Default”. Close the Target Configurations window by clicking the x on the tab.

Create a New Project – M3 Master Subsystem

6. To create a new project for the M3 Master subsystem click:

File → New → CCS Project

In the Project name field type **Lab1_M3**. Uncheck the “Use default location” box. Click the Browse... button and navigate to:

C:\F28M35x\Labs\Lab1\M3

Click OK.

7. The next section selects the device. Select the “Family” using the pull-down list and choose “ARM”. Set the “Variant” filter using the pull-down list to “28M35x Concerto” and choose the “F28M35H52C1” device. Leave the “Connection” box blank since we already set up the target configuration.
8. Next, open the “Advanced setting” section and set the “Linker command file” to “<none>”. We will be using our own linker command file, rather than the one supplied by CCS.
9. Then, open the “Project templates and examples” section and select the “Empty Project” template. Click Finish.

A new project has now been created. Notice the “Project Explorer” window contains Lab1_M3. The project is set Active and the output files will be located in the Debug folder. At this point, the project does not include any source files. The next step is to add the source files to the project.

Add Files to Project – M3 Master Subsystem

Note: The local copy of the supporting files and libraries in this workshop are identical to the required controlSUITE files. The workshop lab exercises will make use of these files as often as possible. When adding files to the project, a window will appear asking to “copy” or “link” the files. Selecting “Copy files” will make a copy of the original file to work with in the local project directory. Selecting “Link files” will set a reference to the original file and will use the original file. Typically, “link files” is used when the files will not be modified. To avoid accidentally modifying the original files, we will use “copy files” throughout this workshop and work with the local copy in the project directory.

10. To add the source files to the project, right-click on Lab1_M3 in the “Project Explorer” window and select:

Add Files...

or click: Project → Add Files...

Add (“copy files”) the following source files to the project:

- a) File: startup_ccs.c
Folder: C:\F28M35x\Device_support\MWare\utils

`startup_ccs.c` is used to call the C-runtime support library preparation function “_c_int00” and to provide interrupt service routines for the M35x core.

- b) File: `F28M35H52C1_m3.cmd`
Folder: `C:\F28M35x\Device_support\MWare\cmd`

`F28M35H52C1_m3.cmd` is used during linking to provide information about the physical memory locations, logical code and data sections and how these sections are placed in memory.

- c) File: `driverlib.lib`
Folder: `C:\F28M35x\Device_support\MWare\driverlib\ccs\Debug`

`driverlib.lib` is a precompiled version of the MWare source code files in the folder “driverlib”. This folder contains source and header files to initialize the M35x core and peripherals based on MWare.

In the Project Explorer window, click the plus sign (+) to the left of Lab1_M3 and notice that the files are listed.

M3 Project Build Options – Include Search Path

11. Setup the build options by right-clicking on Lab1_M3 in the “Project Explorer” window and select “Properties”. We need to setup the include search path to include the MWare files. Under “ARM Compiler” select “Include Options”. In the lower box that opens (“Add dir to #include search path”) click the Add icon (first icon with green plus sign). Then in the “Add directory path” window type:

`${PROJECT_ROOT}/../Device_support/MWare`

Click OK to include the search path. Finally, click OK to save and close the Properties window.

Add M3 Function “main()”

Recall that the objective of this lab exercise is to have the M3 Master subsystem blink LED LD3 and the C28 Control subsystem blink LD2. The M3 function “main()” will need to execute the basic M3 setup:

- Initialize the clock system
- Setup maximum FLASH performance
- Enable M3 master interrupts
- Setup GPIO_PIN 7 for output (LD3)

Finally, a “while(1) loop” with the control code will be used to toggle LD3 periodically. Normally to accomplish this, the Technical Reference Manual for the M3 would be used to setup the necessary registers and initialize them with the (hopefully) correct values. Since we have a limited amount of time for this workshop, the function “main()” has been completed.

12. Add (“copy files”) the function “main()” source file to the project:

File: `Lab1_M3.c`

Folder: C:\F28M35x\Labs\Source_files

13. Open and inspect Lab1_M3.c. Notice at the beginning of the file the required header files are included. Although all of the header files are not needed for this first lab exercise, it is not a problem to include them now for the following lab exercises. Recall that header files do not occupy any memory space on the target:

```
6 #include <string.h>
7
8 #include "inc/hw_ints.h"
9 #include "inc/hw_memmap.h"
10 #include "inc/hw_nvic.h"
11 #include "inc/hw_gpio.h"
12 #include "inc/hw_types.h"
13 #include "inc/hw_sysctl.h"
14 #include "driverlib/debug.h"
15 #include "driverlib/flash.h"
16 #include "driverlib/ipc.h"
17 #include "driverlib/interrupt.h"
18 #include "driverlib/sysctl.h"
19 #include "driverlib/gpio.h"
```

At the beginning of function “main()” the clock unit is initialized, so that the M3 will run at 75 MHz and C28 will run at 150 MHz. Also, the protection system of the M3 is disabled:

```
28 void main(void)
29 {
30     volatile unsigned long ulLoop;
31
32     // Disable Protection
33     HWREG(SYSCTL_MWRALLOW) = 0xA5A5A5A5;
34
35     // Sets up PLL, M3 running at 75 MHz and C28 running at 150 MHz
36     SysCtlClockConfigSet(SYSCTL_USE_PLL | (SYSCTL_SPLLIMULT_M & 0xF) |
37                          SYSCTL_SYSDIV_1 | SYSCTL_M3SSDIV_2 |
38                          SYSCTL_XCLKDIV_4);
```

Next, after function “FlashInit” is copied from FLASH to RAM it is called to optimize the performance of the FLASH:

```
44     memcpy(&RamfuncsRunStart, &RamfuncsLoadStart, (size_t)&RamfuncsLoadSize);
45
46     // Call Flash Initialization to setup flash waitstates
47     // This function must reside in RAM
48     FlashInit();
```

The function “FlashInit” is part of “driverlib.lib”. The source code of “FlashInit” can be found in the “flash.c” file located in folder C:\F28M35x\device_support\MWare\driverlib

The next part of function “main()” will disable the watchdogs (not to be done in an actual real world project), initialize the M3 Interrupts and GPIO_PIN_7 for the LED LD3:

```

59 // Disable clock supply for the watchdog modules
60 SysCtlPeripheralDisable(SYSCTL_PERIPH_WDOG1);
61 SysCtlPeripheralDisable(SYSCTL_PERIPH_WDOG0);
62
63 // Enable processor interrupts.
64 IntMasterEnable();
65
66 // Set up the Pin for LED LD3
67 GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_7);
68 GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, ~0);

```

Finally the endless loop is entered to toggle LED LD3:

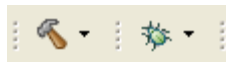
```

70 while(1)
71 {
72     GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0); // LD3 ON
73     for(ulLoop = 0; ulLoop < 2750000; ulLoop++); // delay
74     GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, ~0); // LD3 OFF
75     for(ulLoop = 0; ulLoop < 2750000; ulLoop++); // delay
76 }

```

Build the Project and Program the Flash Memory

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



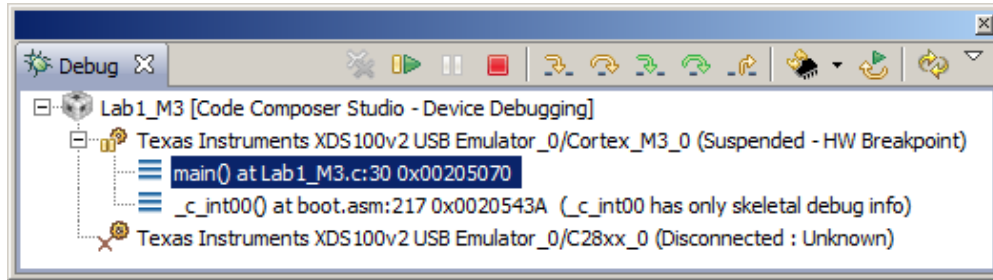
Button Name	Description
1 Build	Full build and link of all source files
2 Debug	Automatically build, link, load/program and launch debug-session

In CCS the on-chip flash programmer is integrated into the debugger. When the program is loaded CCS will automatically determine which sections reside in flash memory based on the linker command file. CCS will then program these sections into the on-chip flash memory. Additionally, in order to effectively debug with CCS, the symbolic debug information (e.g., symbol and label addresses, source file links, etc.) will automatically load so that CCS knows where everything is in your code. Clicking the “Debug” button in the “CCS Edit” perspective view will automatically launch the debugger, connect to the target, and program the flash memory in a single step.

14. Click the “Build” button and watch the tools run in the “Console” window. Check for errors in the “Problems” window. If you get an error, you will see the error message in the “Problems” window. Expand the error by clicking on the plus sign (+) to the left of the “Errors”. Then simply double-click the error message. The editor will automatically open to the source file containing the error, with the code line highlighted with a question mark (?).
15. Program the flash memory by clicking the “Debug” button (green bug) or click Run→Debug. (If needed, when the “Progress Information” box opens select “Details >>” in order to watch the programming operation and status). After successfully programming the flash memory the “Progress Information” box will close.

The CCS Debug icon in the upper right-hand corner indicates that we are now in the “CCS Debug” perspective view. The program ran through the C-environment initialization routine in the run-time support library and stopped at “main()” in Lab1_M3.c. The blue arrow in the left hand column of the source code window indicates the current position of the M3 program

counter (PC). The “Debug” window reflects the current status of the two subsystems – Cortex_M3_0 and C28xx_0.



Notice that the second subsystem (C28xx_0) is currently “Disconnected”. This means that CCS has no control over this subsystem thus far; it is freely running from the view of CCS. Of course the C28xx_0 is always under control of the Master subsystem and since we have not executed an Inter Processor Communication (IPC) command yet, the C28xx_0 is stopped by an “Idle” mode instruction in the Boot ROM.

Run the Code – Test the M3

Two buttons on the horizontal toolbar are commonly used to control program execution. Hover your mouse over each button as you read the following descriptions:



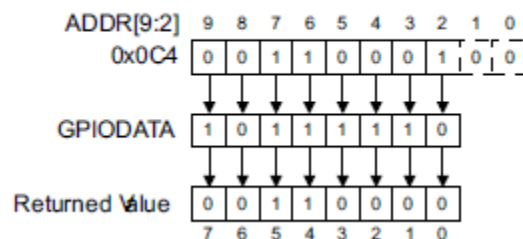
Button Name	Description
1 Resume	Run the selected target (F8)
2 Suspend	Halt the selected target (Alt+F8)

16. Run the M3 code using the “Resume” button on the toolbar, or click Run → Resume. LED LD3 should be blinking at a period of approximately 1 second.

Inspect M3 Registers

Let us inspect the M3 registers for GPIO – Port C. As explained in the Technical Reference Manual (spruh22), chapter 4.1.3.3.3 “Data Register Operation” the current value in port C register GPIODATA can be read by a base address + mask offset access. The ‘1’ bits in the mask will pass the current value of the corresponding bit-position into the result register (see Figure 4-3 of spruh22):

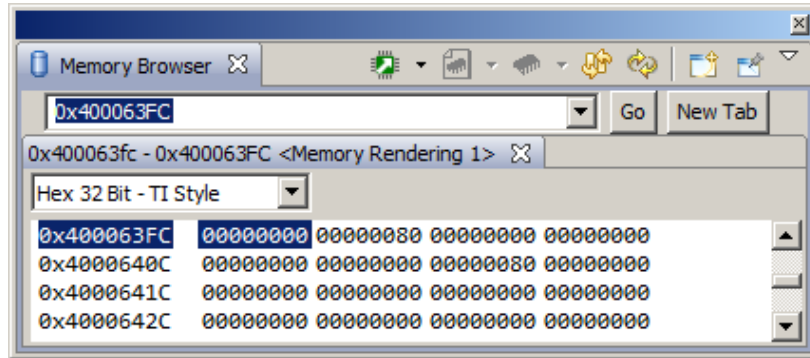
Figure 4-3. GPIODATA Read Example



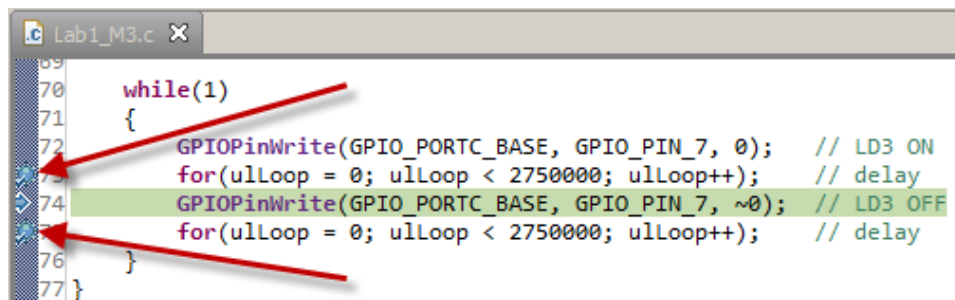
In Figure 4-3 a mask offset of 0xC4 is used to read 3 bits (0, 4 and 5).

In our lab exercise the base address for Port C GPIODATA is 0x40006000. To read all 8 bits we have to use a mask offset of $0xFF \ll 2 = 0x3FC$. The final address to watch register GPIODATA is 0x400063FC.

17. Halt the M3 code using the “Suspend” button on the toolbar, or click Run → Suspend.
18. Open a memory window (View → Memory Browser) and enter address 0x400063FC:



19. Set two breakpoints at the for-loops in “main()” by double clicking in the line number field to the left of the code line. Notice that line is highlighted with a blue dot indicating that the breakpoint has been set. (Alternately, you can set a breakpoint on the line by right-clicking the mouse and selecting Breakpoint (Code Composer Studio) → Breakpoint):



20. Run the M3 code. The execution stops at one of the two breakpoints and the memory browser shows the current value in GPIODATA to be either 0x00 or 0x80. Run again to reach the 2nd breakpoint. The memory browser will show the toggle in bit position 7.
21. Delete the two breakpoints, again with a double click at the left column.

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

22. The “Terminate” button will terminate the active debug session, close the debugger and return CCS to the “CCS Edit” perspective view.

Click: Run → Terminate or use the Terminate icon:

23. Disconnect the two USB connections (power and emulator) from the kit.
24. Re-connect the micro USB connector J2 to power the kit.

The M3 code will start immediately and LED LD3 should be blinking.

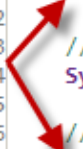
25. Now re-connect the mini USB connector J20 to the kit.

Activate the C28 Control Subsystem

Now we are ready to create the second project for this exercise – the one for the C28 Control subsystem. Before ending the M3 Master subsystem, this is a good time to edit the Lab1_M3.c file. The M3 Master subsystem is responsible to release the C28 out of its “Idle” state in the Boot ROM and to grant control for the LED LD2 to the C28 Control subsystem.

26. Open the Lab1_M3.c file and **uncomment** the “IPCMtoCBootControlSystem()” and the “GPIOPinConfigureCoreSelect()” function calls:

```
50 // Send boot command to allow the C28 application to begin execution
51 IPCMtoCBootControlSystem(CBROM_MTOC_BOOTMODE_BOOT_FROM_FLASH);
52
53 // Enable clock supply for GPIOC
54 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
55
56 // Give C28 control of Port C pin 6
57 GPIOPinConfigureCoreSelect(GPIO_PORTC_BASE, 0x40, GPIO_PIN_C_CORE_SELECT);
```



27. Rebuild the project (“hammer” icon) and then program the modified code into the target (green “bug” icon).

Create a New Project – C28 Control Subsystem

28. Switch to the “CCS Edit” perspective view by clicking the CCS Edit icon in the upper right-hand corner.
29. Create a new project for the C28 Control subsystem by clicking:

File → New → CCS Project

In the Project name field type **Lab1_C28**. Uncheck the “Use default location” box. Click the Browse... button and navigate to:

C:\F28M35x\Labs\Lab1\C28

Click OK.

30. The next section selects the device. Select the “Family” using the pull-down list and choose “C2000”. Set the “Variant” filter using the pull-down list to “28M35x Concerto” and choose the “F28M35H52C1” device. Leave the “Connection” box blank since we already set up the target configuration.
31. Next, open the “Advanced setting” section and set the “Linker command file” to “<none>”. We will be using our own linker command file, rather than the one supplied by CCS.
32. Then, open the “Project templates and examples” section and select the “Empty Project” template. Click Finish.

A new project has now been created. Notice the “Project Explorer” window contains Lab1_C28. Click on the Lab1_C28 project. The project is set Active and the output files will be located in the Debug folder. At this point, the project does not include any source files. The next step is to add the source files to the project.

Add Files to Project – C28 Control Subsystem

33. Add (“copy files”) the following source files to the project by right-clicking on Lab1_C28 in the “Project Explorer” window and selecting Add Files...

a) File: F28M35x_CodeStartBranch.asm

File: F28M35x_SysCtrl.c

Folder: C:\F28M35x\Device_support\F28M35x_common\source

F28M35x_CodeStartBranch.asm contains a single branch instruction to the C-runtime start function “_c_int00” and a label “code_start”, which is used by the linker.

F28M35x_SysCtrl.c defines two functions “InitFlash” and “InitSysCtrl”, which will be called shortly.

- b) File: F28M35x_GlobalVariableDefs.c

Folder: C:\F28M35x\Device_support\F28M35x_headers\source

F28M35x_GlobalVariableDefs.c defines all data memory mapped C28 peripheral registers as global variables.

- c) File: F28M35x_Headers_nonBIOS.cmd

Folder: C:\F28M35x\Device_support\F28M35x_headers\cmd

F28M35x_Headers_nonBIOS.cmd links all C28 registers to their physical addresses.

- d) File: F28M35H52C1_c28.cmd

Folder: C:\F28M35x\Device_support\F28M35x_common\cmd

F28M35H52C1_c28.cmd links all code and data sections to physical memory. All non-volatile sections are linked to the C28 FLASH.

In the Project Explorer window, click the plus sign (+) to the left of Lab1_C28 and notice that the files are listed.

C28 Project Build Options – Include Search Path

34. Setup the build options by right-clicking on Lab1_C28 in the “Project Explorer” window and select “Properties”. We need to setup the include search path to include the peripheral register header files. Under “C2000 Compiler” select “Include Options”. In the lower box that opens (“Add dir to #include search path”) click the Add icon (first icon with green plus sign). Then in the “Add directory path” window type (one at a time):

`${PROJECT_ROOT}/.././Device_support/F28M35x_headers/include`

`${PROJECT_ROOT}/.././Device_support/F28M35x_common/include`

Click OK to include the search path. Finally, click OK to save and close the Properties window.

Add C28 Function “main()”

Previously, we added an M3 function “main()” that is used to execute the basic M3 setup. Now we need to add a C28 function “main()” to execute the basic C28 setup. Recall that the objective of this lab exercise is to have the M3 Master subsystem blink LED LD3 and the C28 Control subsystem blink LD2. The C28 function “main()” will need to:

- Initialize the C28 subsystem

- Setup maximum FLASH performance
- Setup GPIO_PIN 70 for output (LD2)

Finally, a “while(1) loop” with the control code will be used to toggle LD2 periodically. As in the first part of this lab exercise, since we have a limited amount of time for this workshop, the function “main()” has been completed.

35. Add (“copy files”) the function “main()” source file to the project:

File: Lab1_C28.c

Folder: C:\F28M35x\Labs\Source_files

36. Open and inspect Lab1_C28.c. At the beginning of the file the required header files are included. In the function “main()” notice that “InitSysCtrl()” is called first to initialize the C28 subsystem, and then Pin GPIO70 is set as digital output:

```
13 void main(void)
14 {
15     unsigned long uldelay;
16     InitSysCtrl();           // Init C28 core
17     EALLOW;
18     GpioG1CtrlRegs.GPCDIR.bit.GPIO70 = 1; // GPIO70 as output
19     EDIS;
20     GpioG1DataRegs.GPCDAT.bit.GPIO70 = 1; // turn off LED
```

After function “FlashInit()” is copied from FLASH to RAM it is called to optimize the performance of the FLASH:

```
25 memcpy(&RamfuncsRunStart, &RamfuncsLoadStart, (size_t)&RamfuncsLoadSize);
26
27 // Call Flash Initialization to setup flash waitstates
28 // This function must reside in RAM
29 InitFlash();
```

Finally the endless loop is entered to toggle LED LD2:

```
31 while(1)
32 {
33     GpioG1DataRegs.GPCDAT.bit.GPIO70 = 0; // LD2 ON
34     for(uldelay = 0; uldelay < 1000000; uldelay++); // delay
35     GpioG1DataRegs.GPCDAT.bit.GPIO70 = 1; // LD2 OFF
36     for(uldelay = 0; uldelay < 1000000; uldelay++); // delay
37 }
```

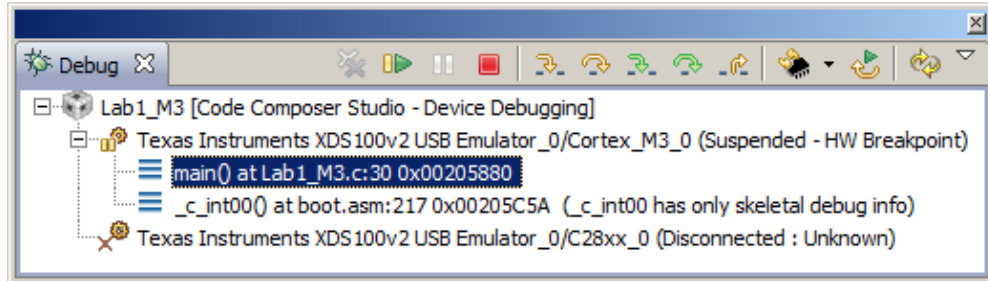
Build the Project and Program the Flash Memory

37. Click the “Build” button and watch the tools run in the “Console” window. Check for errors in the “Problems” window (see the following note).

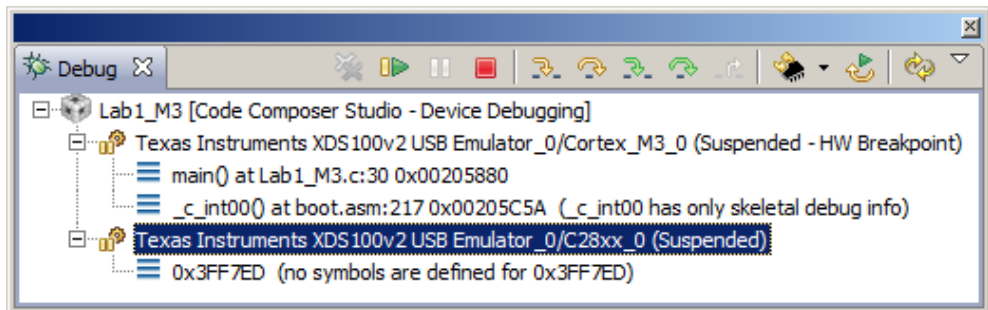
Note: When the project is built, the compiler will issue a warning about an unused variable. This is caused by a software routine which makes a dummy read from a hardware register in order to initialize the flash. The register contents are assigned to a local variable called 'temp' and as the variable is not subsequently used in the code the compiler issues a warning that the variable might be redundant. This is harmless and the warning can safely be ignored.

DO NOT continue with the “Debug” button! The JTAG connection is still active from the M3 project. We will use a different way to program the C28 Control subsystem.

38. Switch to the “CCS Debug” perspective view. The “Debug” window should reflect the current status of the two subsystems – Cortex_M3_0 = connected, and C28xx_0 = disconnected:



39. Right click at the line “Texas Instruments XDS100v2 USB Emulator_0/C28xx_0” and select “Connect Target”:



40. With the line “Texas Instruments XDS100v2 USB Emulator_0/C28xx_0” still highlighted, load the program:

Run → Load → Load Program...

Browse to the file: C:\F28M35x\Labs\Lab1\C28\Debug\Lab1_C28.out and select OK to load the program. The programming procedure will start to program the C28 flash memory.

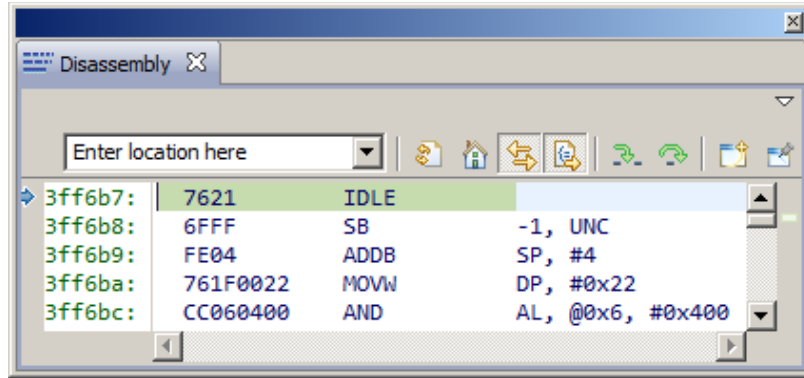
Dual Subsystem Debugging

41. In the “Debug” window left click on the line Texas Instruments XDSv2 USB Emulator_0/C28xx_0 and then click the “Reset CPU” icon:



or by clicking Run → Reset → Reset CPU

42. Next in the “Debug” window left click on the line Texas Instruments XDSv2 USB Emulator_0/Cortex_M3_0 and then click the “Reset CPU” icon.
43. In the “Debug” window highlight the C28 subsystem and run the code by clicking the Resume icon (or F8). The C2x runs into an “Idle” instruction in the boot ROM code and waits for an IPC command from the M3 subsystem.
44. Just for inspection, halt the C28 subsystem by clicking the Suspend icon (alt-F8). With a Disassembly window open notice the program counter is pointing to the “Idle” instruction in boot ROM:




45. Now run the C28 Subsystem again (Resume icon or F8)
46. Next in the “Debug” window highlight the M3 subsystem and run the code (Resume icon or F8). Both the C28 and M3 subsystems should now be running. The C28 subsystem is controlling the blinking of LED LD2, and the M3 subsystem is controlling the blinking of LED LD3. If you would like to stop, single step through the programs or set breakpoints for either M3 or C28 subsystems, just highlight the corresponding subsystem in the “Debug” window and apply the test command using the menu bar or the icon.

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

At this point, the code for both the C28 and M3 subsystems has been programmed in their respective flash memories. Recall the boot mode has been set as “Boot from Flash”.

47. The “Terminate” button will terminate the active debug session, close the debugger and return CCS to the “CCS Edit” perspective view.

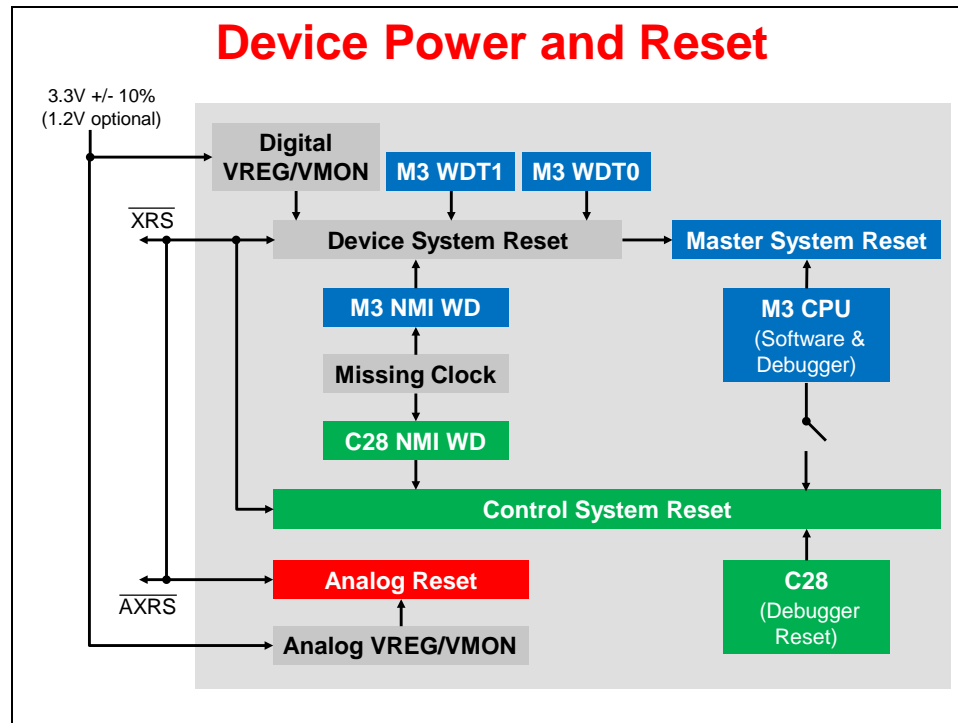
Click: Run → Terminate or use the Terminate icon: 

48. Disconnect the two USB connections (power and emulator) from the kit.
49. Re-connect the micro USB connector J2 to power the kit.
Both the M3 and C28 code will start immediately and both LED LD3 and LED LD2 should be blinking.
50. Now re-connect the mini USB connector J20 to the kit.

End of Exercise

Reset and System Initialization

Device Power and Reset



The device has two voltage regulators and monitors – Digital VREG and Analog VREG.

The Digital VREG supplies the required voltages for the digital logic. The total device power must meet package tolerances. If power usage is higher than package tolerances, then the user can disable the digital regulator and supply an external 1.2V.

The Analog VREG supplies voltages to the analog modules. This regulator cannot be disabled.

Reset pins AXRSn and XRSn are always tied together external to the device. The types of resets are:

1. Device Reset - this will reset the Master, Control and Analog modules. If the reset is caused by an internal module, it is fed back out through the XRSn pin.
2. Master Subsystem Reset – resets the Master M3 CPU and Master subsystem peripherals.
3. Control Subsystem Reset – resets the Control C28 CPU and peripherals affected by XRSn pin but does not drive out on XRSn.
4. Analog Reset - reset to the analog modules of the device occurs as part of a device reset.

The device reset is caused by:

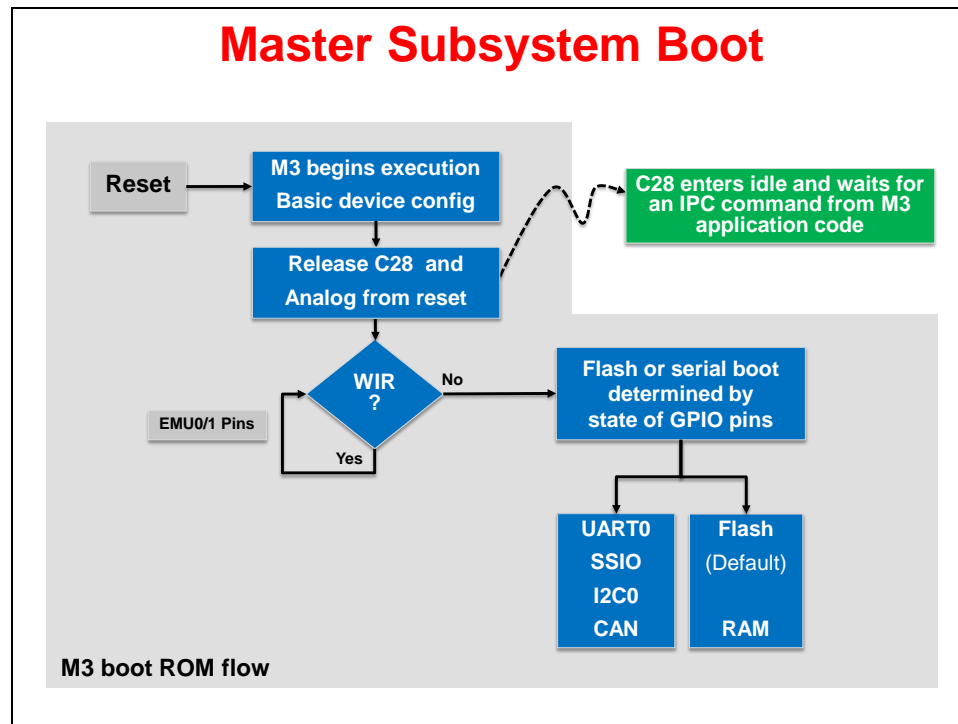
- XRSn pin (and AXRSn pin).
- Digital voltage monitor and the analog voltage monitor.
- M3 watchdog 0 and 1.
- M3 NMI watchdog/missing clock detection.

Control subsystem reset is caused by:

- A device system reset.
- C28 NMI WD timeout/missing clock detection.
- C28 debugger reset.
- M3 software / debugger – this can be disabled if desired.

Master subsystem is reset by any device reset and can be reset by software or the M3 debugger.

Master and Control Subsystem Boot

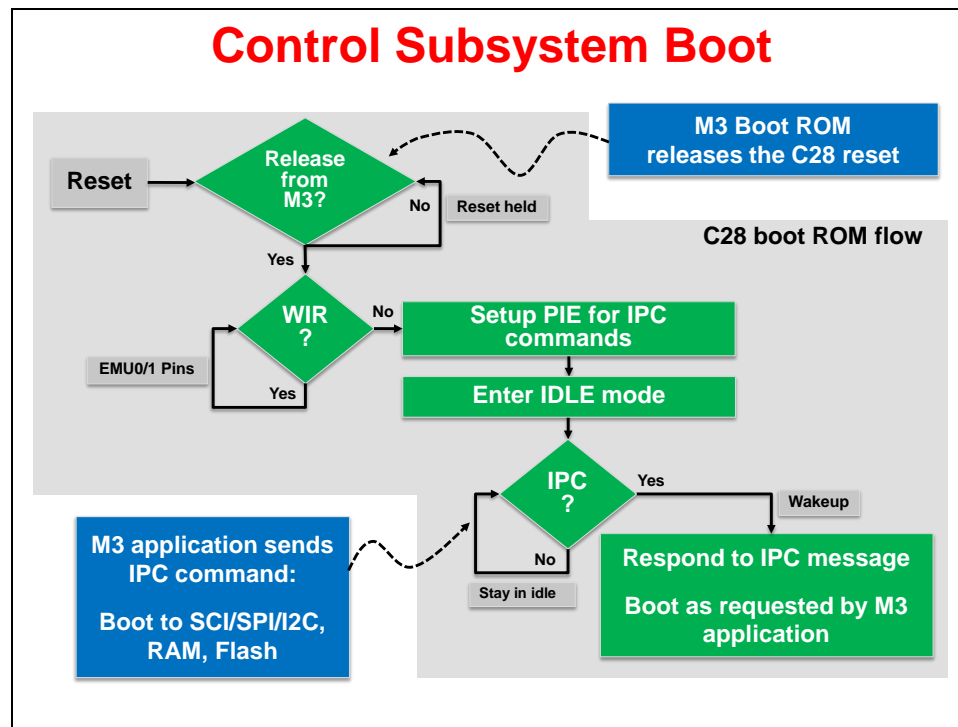


The M3 is the master for the boot process.

At reset, the M3 will begin execution. The rest of the device will stay in reset until reset is released by the M3 boot ROM. This includes the C28 Control subsystem and the Analog Common Interface Bus (Analog CIB). After being released from reset, the C28 simply performs some basic setup and then goes into the idle mode and waits for further instructions via an IPC command, discussed in the next slide.

The wait in reset (WIR) mode is entered by the state of the EMU0/EMU1 pins. Wait in reset will keep the M3 from continuing execution until an emulator is connected.

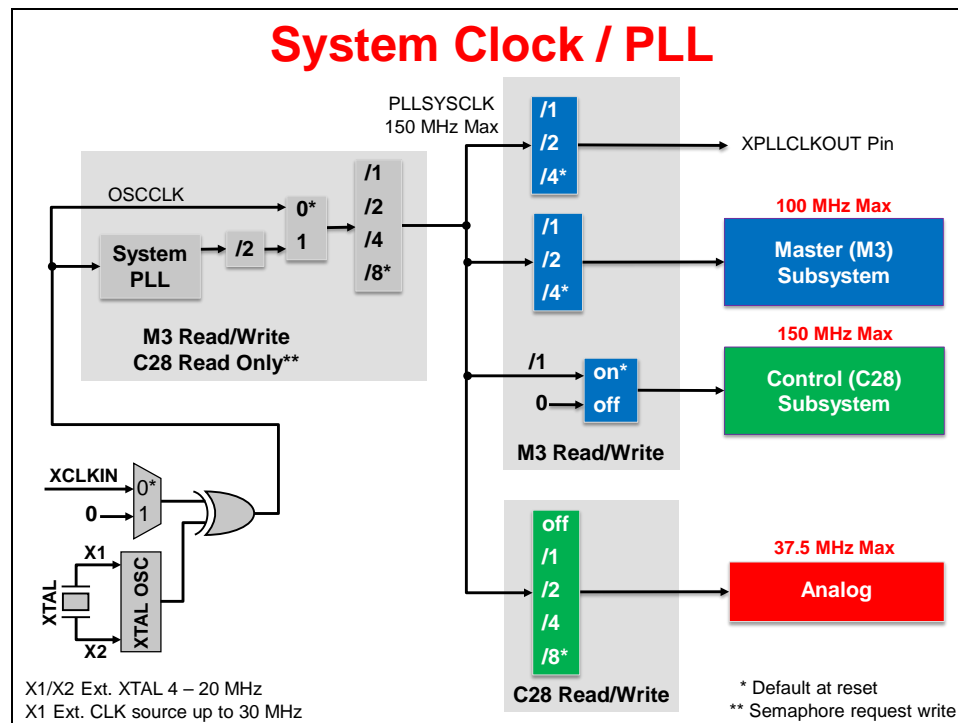
The boot mode, flash or serial, is determined by the state of GPIO pins. If the boot is to a serial port, then the code is downloaded from the peripheral to RAM and then execution is passed to the downloaded code. If the boot is to the flash, then execution is transferred to the entry point of the flash.



When the C28 CPU is released from reset, it will read the state of the EMU0/1 pins and if the state of the pins indicate wait in reset (WIR) mode, then it will sit in a loop continuously reading the state of the EMU0/1 bits.

The C28 sets up the PIE to handle MTOCIPCINT1. After booting, the M3 application will take over execution on the master subsystem. This application code sends IPC commands to tell the C28 what it should do next. All other interrupts in the PIE are disabled or ignored. The IPC message handlers at this point are within the boot ROM of the C28 Control subsystem. An API supplied by TI makes using the boot IPC commands easy.

System Clock / PLL



OSCLK – the source to generate the system clock using X1 or X1/X2:

- X1: external single-ended clock source up to 30MHz.
- X1/X2: external crystal (4-20MHz) or resonator.

System PLL - configuration of the System PLL clock (PLLSYSCLK):

- PLL control (bypass, multiplier, on/off) and divider from PLL output.
- Configured by the M3. The C28 can also freely read these values.
- The C28 can request write control to the PLL multiplier and divider by using the clock control semaphore. Typically this is only used for flash programming during development (e.g. via Code Composer Studio)

PLLSYSCLK dividers configured by the M3 Master **ONLY**:

- Divider from PLSYSCLK to XPLLCLKOUT Pin
- Divider from PLSYSCLK to the Master communications subsystem. This clock is known as M3SSCLK and can be /1, /2 or /4 of the PLSYSCLK. The divider must be configured to result in a Master subsystem (M3SSCLK) clock of 100 MHz or less.
- There is no divider from PLSYSCLK to the Control subsystem. The C28 CPU executes at the same rate as PLSYSCLK and can be a maximum of 150MHz.
- The M3 can, however, enable/disable the clock to the control subsystem. This can be done to save power.

PLLSYSCLK divider configured by the C28x **ONLY**:

- The Analog and the Control subsystem are related. Thus the divider from the PLL to the analog modules is controlled by the C28. The ADCCLK can be turned off, or configured for /1, /2.../8 of the PLSYSCLK. This divider must be configured to result in an ADCCLK of 37.5 MHz or less.

Example clock configurations are described on the following slide:

System Clock Configurations

Maximum Frequencies				
C28 MHz CLKIN	M3 MHz M3SSCLK	Analog MHz ADCCLK	ADC MSPS ADCCLK/13	EPI MHz M3SSCLK/2
150.0	100.0	37.5	2 x 2.885	50.0

Possible Combinations				
C28 MHz	M3 MHz	Analog MHz	ADC MSPS	EPI MHz
150.0	150.0 (/1)	150.0 (/1)		
	75.0 (/2)	75.0 (/2)		37.5
	37.5 (/4)	37.5 (/4)	2 x 2.885	18.75
100.0	100.0 (/1)	25.0 (/4)	2 x 1.923	50.0
60.0	60.0 (/1)	30.0 (/2)	2 x 2.308	30.0

API Example:

```

// Sets up PLL, M3 running at 75MHz and C28 running at 150MHz
SysCtlClockConfigSet(SYSCTL_USE_PLL | (SYSCTL_SPLLIMULT_M & 0xF) |
SYSCTL_SYSDIV_1 | SYSCTL_M3SSDIV_2 | SYSCTL_XCLKDIV_4);

```

C28 CLKIN == PLL System Clock

Recall the maximum Control subsystem clock is 150MHz, the maximum master subsystem clock is 100MHz, and the ANALOG SYSCLK is limited to 37.5 MHz. The last two columns show the possible ADC and EPI performance at these maximum frequencies.

Users will select clock divisors for the M3SSCLK and the Analog SYSCLK to always meet these maximum timings. Remember there is no divider on the C28x clock; it is the same as PLLSYSCLK.

The 2nd table explains the possibilities for three configurations: (1) maximum C28 system, (2) maximum M3 system, and (3) low end MIPs system.

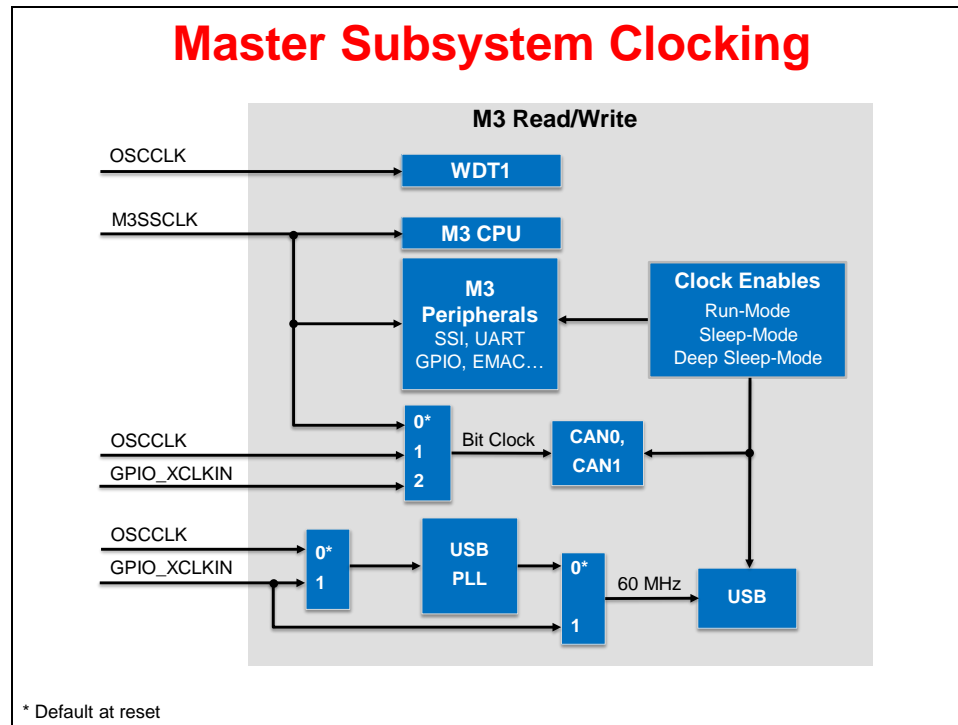
Row 1: The maximum C28 frequency system starts with C28 of 150MHz.

- If the M3 clock is set to /1 this will not work since 150 is above the M3 maximum frequency. If the M3 clock is configured as /2 then this will work since 75MHz is less than the maximum M3 clock. The M3 clock could also be set to /4 to yield a 37.5 MHz M3 clock.
- The Analog clock divider cannot be set to /1 (150 MHz) or /2 (75 MHz) because both are higher than the maximum Analog SYSCLK. The divider can be set to /4 (37.5 MHz) or set to /8 because these are both less than or equal to the maximum Analog SYSCLK.

Row 2: The maximum M3 frequency system has the maximum M3 system clock of 100MHz.

- Recall the M3 clock this is a divider off of PLLSYSCLK == C28x SYSCLKOUT. The Control subsystem must also run at 100 MHz and M3 clock divider must be /1 to. This results in the Master subsystem running at its maximum frequency of 100MHz.
- The Analog clock divider cannot be set to /1, /2 because this will violate the maximum Analog SYSCLK rate. The highest it can be set is /4 (it can also be set to /8 if desired).

Row 3: Low end MIPs system. In this case the whole system is run at a slower frequency: 60MHz C28 clock, 60MHz M3 clock, and 30.0 MHz Analog SYSCLK. The M3 could be divided down further to 30MHz if desired. Likewise the Analog SYSCLK could be reduced to (/4) or (/8).



All of the configurations shown on the slide above are performed by the M3 Master subsystem.

M3SSCLK (M3 subsystem clock) is the main clock for the Master subsystem for the M3 CPU and most of the M3 related peripherals such as UARTs, EMAC, GPIO, SSI, I2C, GPT. Recall that the maximum frequency for this clock is 100 MHz. Other peripherals are clocked as follows:

Watchdog timer 1 is clocked from a different clock domain - OSCCLK (from X1/X2 pins) - which is important for safety.

GPIO_XCLKIN is multiplexed on GPIO63. This clock can ONLY be used as a source to the USB or CAN modules

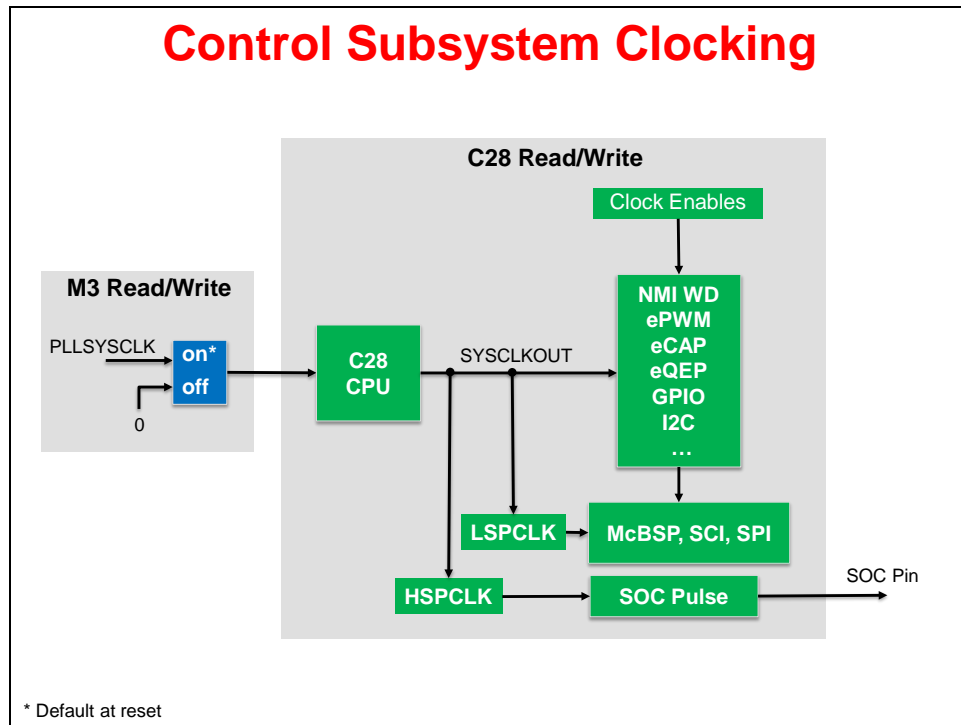
CAN:

- CAN is susceptible to clock jitter at higher bit rates.
- Allows a more accurate and stable bit clock to be selected.
- If the application uses a spread spectrum system clock the user can select an alternative clock and thus will not compromise communications.

USB:

- USB has its own PLL which can supply 60MHz to the peripheral. This allows USB to be independent of the system clock ratios used by the rest of the system.
- If GPIO_XCLKIN is used to clock the USB, then it is limited to 60 MHz.

Clocks to peripherals can be disabled/enabled to conserve power when modules are not in use or when in low-power modes.



In the M3 read/write section, the M3 Master can turn off the clock to the Control subsystem to save power. Everything else on the above slide is configured by the C28 Control subsystem.

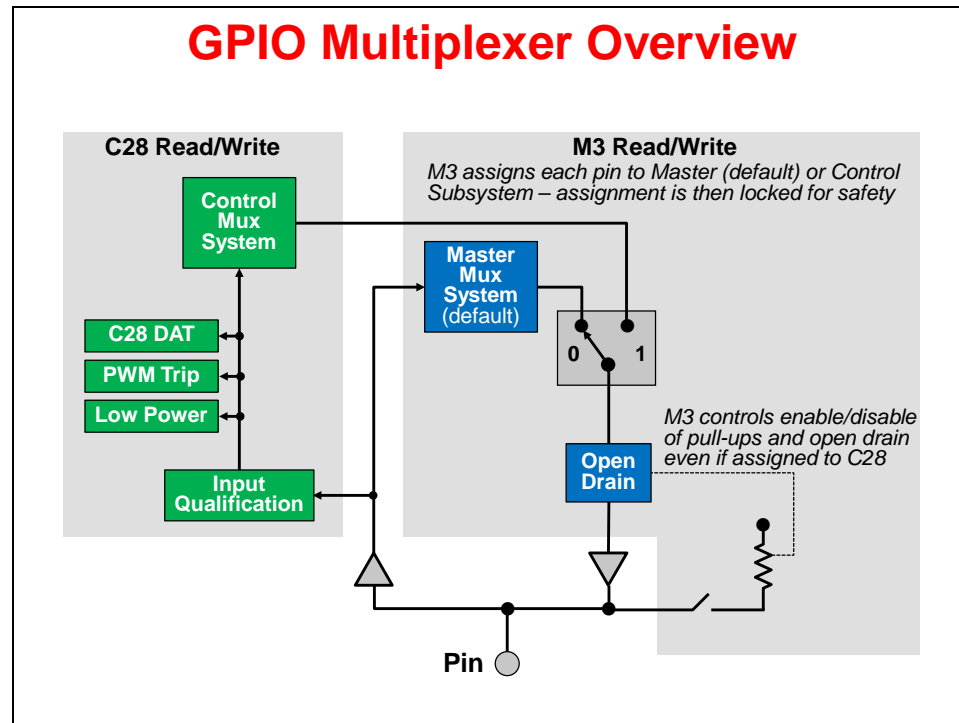
In the C28 only read/write section, the clock enables allow the C28 to turn off clocks to modules that are not being used to save power. The Control subsystem clcking is similar to Delfino and Piccolo C2000 class devices.

SYCLKOUT is a maximum of 150 MHz. It clocks the C28 CPU, control peripherals (PWM, capture, HRPWM, QEP), I2C, CPU Timers, GPIO, DMA, and the NMI watchdog.

The Low Speed Clock (LSPCLK) clocks all serial ports except I2C. LSPCLK is a ratio of SYCLKOUT (equal, /2, /4, /6... /12)

The High Speed Clock (HSPCLK) is a ratio of SYCLKOUT (equal, /2, /4... /14). It is used to control the width of an external SOC (start of conversion) pulse. This pulse is generated from the ePWM logic.

GPIO Multiplexer



There are two multiplexer systems on F28M35x – one on the M3 subsystem and one on the C28 subsystem. At reset all of the GPIO pins belong to the M3 multiplexer system.

The M3 is responsible for allocating pins to the C28 multiplexer system. Once the pins have been allocated, the assignment can be locked for safety. The next three slides will go into each multiplexer system in more detail.

The input always goes to the C28, even if the pin is assigned to the M3 multiplexer system. This means the M3 cannot block inputs to the C28 which is important for signals like trip inputs to the ePWM modules. The C28 can also read the pin state in its DAT (data) register even if it is configured for the M3 multiplexer system.

The input qualification for C28 is compatible with Delfino and Piccolo devices:

- Async (no qualification)
- Sync to C28 SYSCLKOUT
- Sync to C28 SYSCLKOUT + filter

Pull-up and open drain – only the M3 can enable/disable pull-ups, and control if a pin is open-drain or not, even if the pin is assigned to the C28 multiplexer system. Pull-ups are disabled on reset. Open drain is available on all GPIO outputs.

Note: The following I/O features are not supported on the F28M35x: I/O drive, slew-rate and pull-downs.

GPIO Multiplexer Assignment

◆ Multiplexer options are based on subsystem assignment

Pin	If assigned to the Master Subsystem				If assigned to the Control Subsystem	
GPIO4	M3 GPIO	SSIORX	CAN0RX	U1DSR	C28 GPIO	EPWM3A
GPIO5	M3 GPIO	SSIoTXX	CAN0TX	U1RTS	C28 GPIO	EPWM3B

API Example:

```
// Give C28 control of Port C pin 6
GPIOPinConfigureCoreSelect(GPIO_PORTC_BASE, 0x40, GPIO_PIN_C_CORE_SELECT);
```

Which peripherals are available on a particular pin first depends on whether the pin has been assigned to the Master subsystem (default) or the Control subsystem. Only the M3 can assign pins to either system. Once the assignment is made it can be locked by the M3 for safety.

Master subsystem peripherals are only available if the pin is assigned to the master multiplexer. This is the default assignment at reset. Likewise, Control subsystem peripherals are only available if the pin is assigned to the Control subsystem multiplexer.

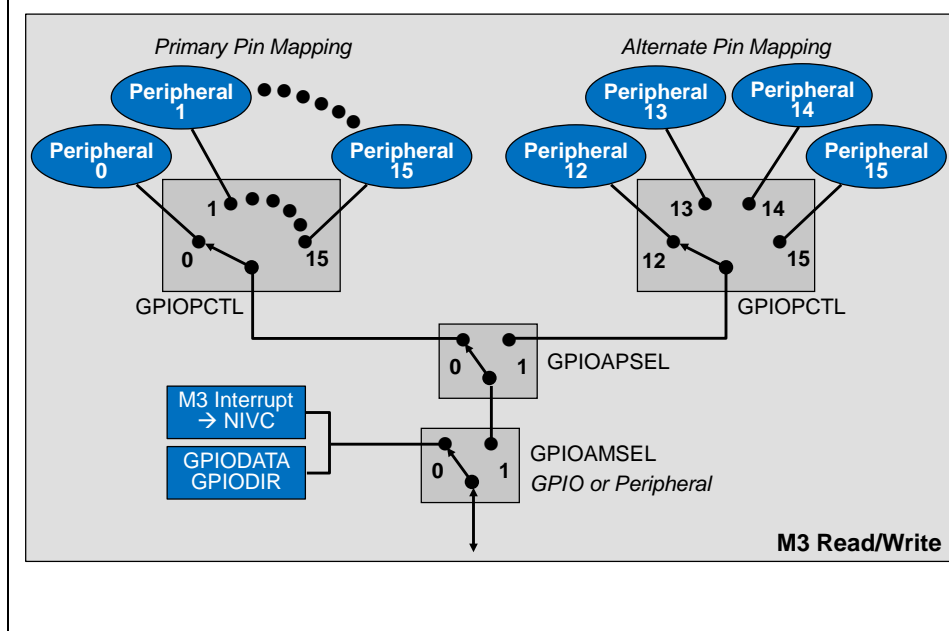
As an example, consider pins GPIO4 and GPIO5. If these pins are assigned to the Master subsystem, then the M3 can configure the pin to be an M3 GPIO or SSI, CAN or UART pin. If these two pins are instead assigned to the Control subsystem, then the C28 will configure whether the pin is a C28 GPIO or the EPWM3A or 3B output.

In summary:

- Only M3 GPIO and Master subsystem peripherals are available on the Master subsystem multiplexer.
- If the pin is to be used as an EPWM or C28 GPIO, then it should be assigned to the Control subsystem.
- Only C28 GPIO and Control subsystem peripherals are available on the Control subsystem multiplexer.

The next two slides will show the multiplexing within each of the subsystems.

Master GPIO Multiplexer Subsystem



This slide shows how the M3 can configure a pin that has been assigned to the Master subsystem multiplexer. All of the configurations on this slide are performed by the M3.

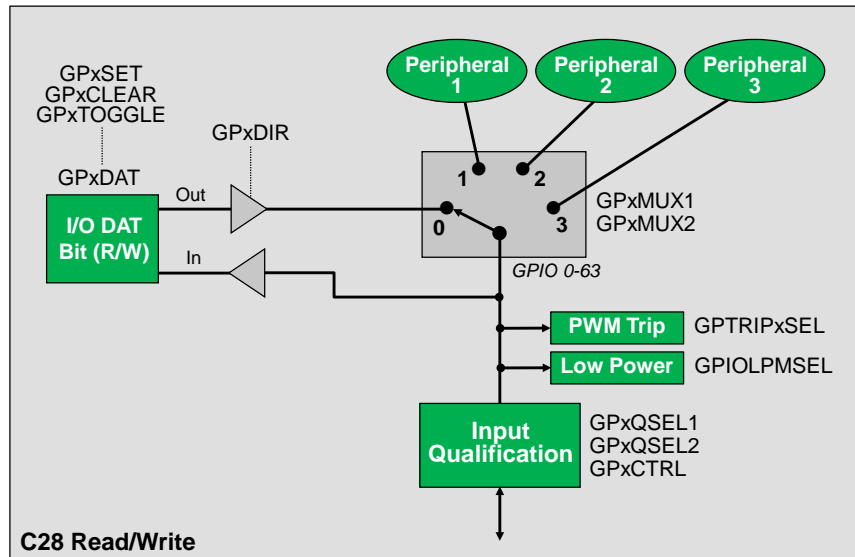
GPIO – the M3 then selects if the pin should be a M3 GPIO or if it should be connected to a peripheral function. The GPIO selection is the default. If the pin is configured to be a GPIO, then it will be routed to the M3 data and direction registers as well as the interrupt control logic that feeds the NVIC.

Peripheral – if the pin is configured to be a peripheral function, then the M3 will select which of the available peripherals will be connected. Note, only peripherals that can be used by the M3 are available to the Master multiplexer subsystem (i.e. no Control subsystem peripherals are available).

Flexible Multiplexer Pin Mapping – for a peripheral function two multiplexer schemes are provided. The primary (default) multiplexer mapping on the Mater subsystem provides compatibility with Tiva™ devices. To gain additional pin-mapping flexibility the user can select an alternate mapping. This alternate mapping is similar to existing C2000 devices.

Note: The pin can be enable/disable (not shown on this slide). If it is enabled, but assigned to the C28 multiplexer, then the M3 can still read the state of the pin.

Control GPIO Multiplexer Subsystem



If a pin is assigned to the C28 multiplexer system then the C28 has free use of the pin. Note that even in this case, the C28 will not be able to control the pull-up or the open drain enable/disable – this is an M3 only operation.

Only peripherals available to the C28 are available in the peripheral multiplexer of the Control subsystem. The mapping of peripherals is compatible to current C2000 devices.

Input Qualification is also compatible with Delfino and Piccolo devices:

- Async (no qualification)
- Sync to C28 SYSCLKOUT
- Sync to C28 SYSCLKOUT + filter

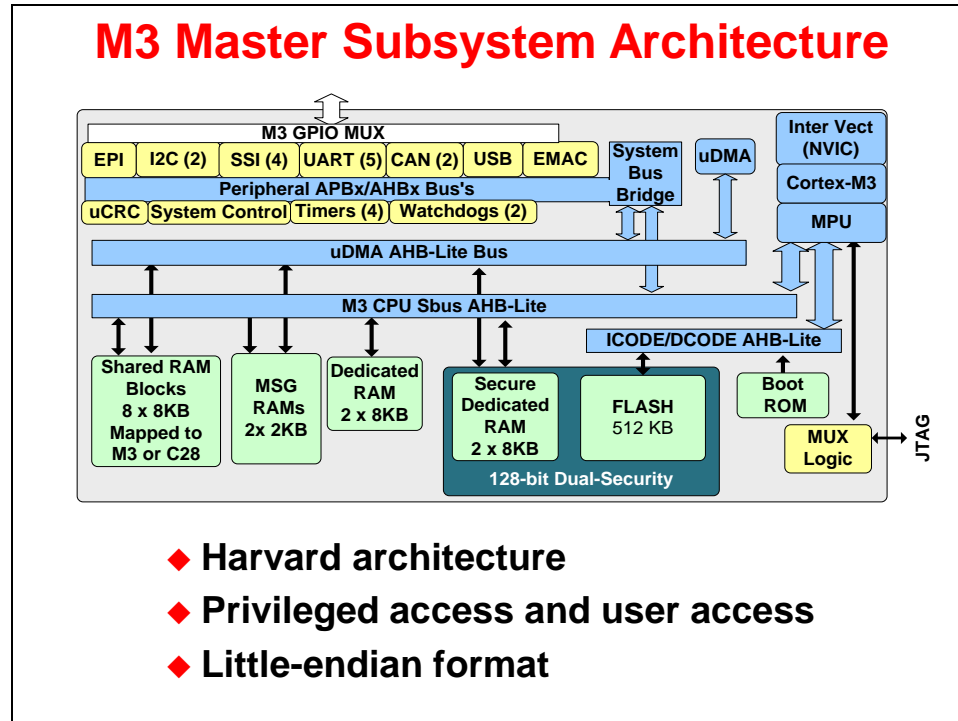
Additionally, GPIO0-63 can be configured as TRIP to ePWMs, inputs to eCAP and interrupt 1-3. This provides more flexibility than in previous C2000 devices.

Architectural Overview – Master M3 Subsystem

Architectural Block Diagram

The F28M35x Master subsystem is based on Cortex-M3 core which is a 32-bit machine. It features a Harvard architecture characterized by separate buses for instruction and data.

The Cortex-M3 supports 32-bit words, 16-bit half words, and 8-bit bytes. The processor also supports 64-bit data transfer instructions. All instruction and data memory accesses are little endian.



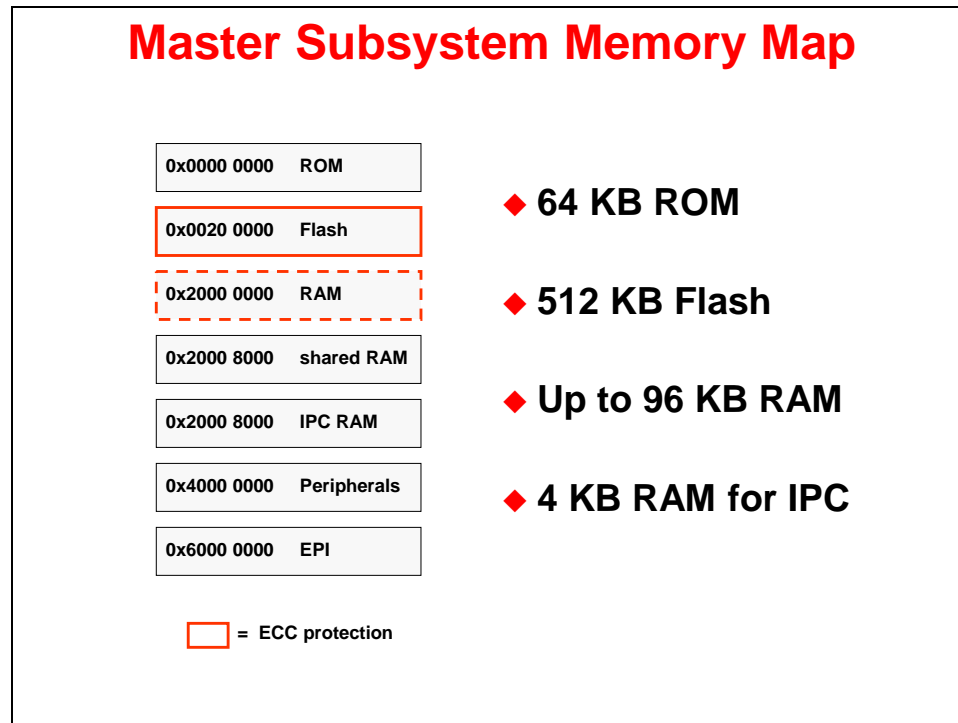
The Cortex-M3 processor integrates hardware support for interrupts through its nested interrupt controller (NVIC).

The Memory Protection Unit (MPU) improves system reliability. If a program accesses a memory location that is prohibited by the MPU, the processor generates a memory management fault, causing a fault exception and possibly causing termination of the process in an OS environment.

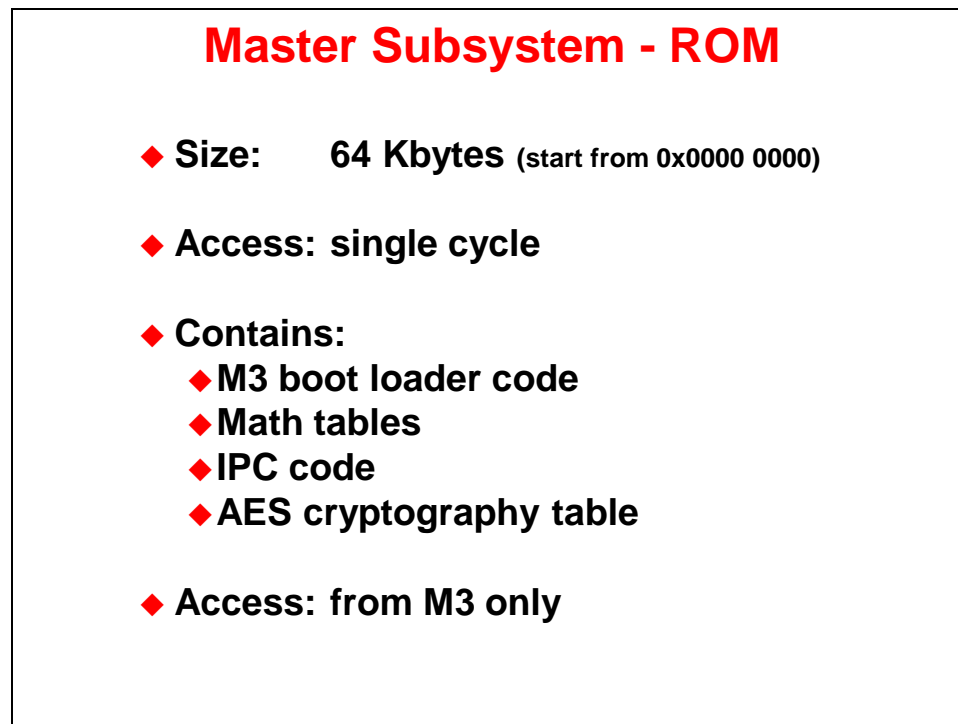
The M3-core also comes with an enhanced system debug with extensive breakpoint and trace capabilities. Please note that Serial Wire Debug is not supported by F28M35x.

F28M35x devices include additional write protection to critical registers. This is achieved by using a “double write” method. In this method there is a WRALLOW register, which if written with a particular value will allow writes to all other “PROTECTED” registers defined in this specification. The WRALLOW register is only writable in M3 privilege mode.

Memory System



The Master subsystem memory map starts at address 0x0000 0000 with on-chip boot ROM. The μ DMA controller can transfer data to and from the on-chip SRAM and peripherals. However, because the Flash memory and ROM are located on a separate internal bus, it is not possible to transfer data from the Flash memory or ROM with the μ DMA controller.

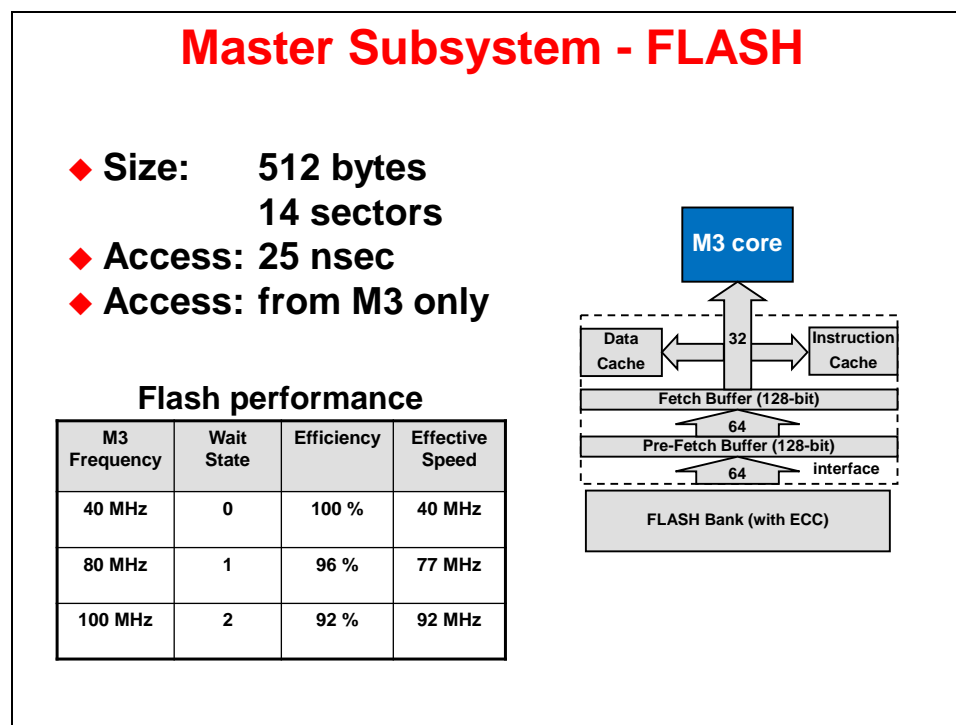


Note again that address 0x0000 0000 in the boot ROM contains the NVIC (Nested Vectored Interrupt Controller). The boot ROM performs a number of operations and then transfers control to the user application. The user application can then setup the NVIC and remap it to dedicated RAM.

The on-chip ROM is 64 Kbytes. On top of the M3 boot loader code, the M3 ROM contains some math tables.

The IPC (Inter-Processor Communication code) also resides in this ROM memory. M3 Boot ROM does not include Flash API. Nothing that is used specifically for Ethernet stack or USB is in the boot ROM. It does include AES cryptography table.

Only the M3 core can access its ROM block. This block cannot be accessed from the μ DMA or the C28 subsystem.



The F28M35x is based on 65nm FLASH Technology. The M3 Master subsystem and C28 Control subsystem operate from independent Flash banks. The M3 Flash is 512 Kbytes which is split into 14 distinct sectors. The Flash cannot be accessed from the μ DMA.

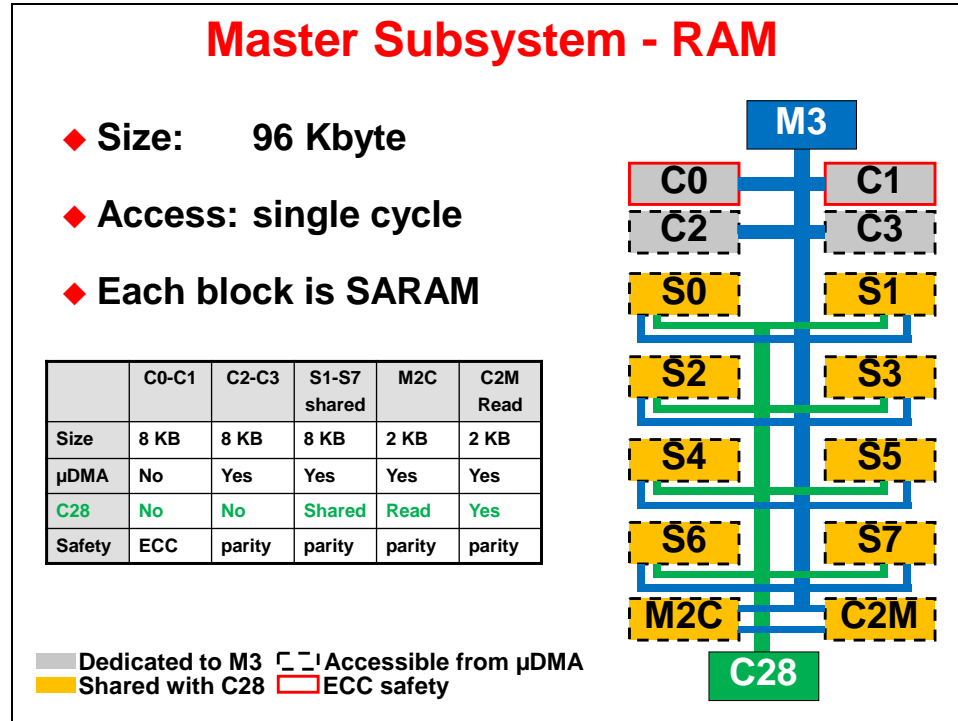
The Flash technology used in the F28M35x device has an overall cycle time of 25ns (or 40MHz). This number includes any ECC error detection and correction overhead and any overhead associated with performance acceleration mechanisms and caching.

To boost code execution performance, acceleration hardware is added. It consists of a 128-bit wide 2-level pre-fetch mechanism. After the first 128-bit wide fetch, the Flash interface will fetch ahead the next 128-bit wide word while the currently fetched instructions are fed to the CPU. In theory, 128-bit wide word can hold four 32-bit instructions or eight 16-bit instructions. Hence, this system can provide near single cycle performance even if the Flash wait states are as high as 3 (four cycle access).

In addition to the instruction pre-fetch mechanisms, the M3 Master System Flash interface also implements a 128-Byte instruction cache. Simulations have shown that this can significantly

boost performance for certain algorithms without severely impacting the "predictability" of execution, which can be critical in real time systems. An 8 byte data cache is also present.

During a Flash memory operation (write, page erase, or mass erase) access to the Flash memory is inhibited. Since the C28 and the M3 share the same pump, there is a Flash semaphore for pump ownership.



The 96 Kbytes of RAM are split in 12 distinct blocks of 8 Kbytes each.

Two additional 2 Kbytes blocks are reserved for inter-processor communication.

- M2C is for M3 to C28 exchange
- C2M is for C28 to M3 exchange
- C blocks are dedicated to the M3 subsystem
- S block are shared

After Reset all of the 8 shared RAM blocks are mapped to the M3 subsystem.

M3 Memory Protection Unit (MPU)

- ◆ Cortex M3 memory is split in 8 Protection regions + 1 background region
- ◆ Independent settings for each region
 - ◆ Location start address
 - ◆ Size configurable from 32 B to 4 GB
 - ◆ Independent attributes for each region
- ◆ **“Access Privilege” (AP)**
 - ◆ No access (NO)
 - ◆ Read and Write (R/W)
 - ◆ Read only (R)
- ◆ Overlapping protection regions with region priority
- ◆ MPU mismatches and permission violations invoke Memory Manager fault handler

Privileged Super-user	Unprivileged User	Permission Fault
R/W	R/W R NO	None Write by unprivileged Access from unprivileged
R	R No	Writes Writes + Read from unprivileged
NO	NO	All

The Memory Protection Unit (MPU) improves system reliability by defining the memory attributes (no access, R/W, Read only) for different memory regions.

If a program accesses a memory location that is prohibited by the MPU, the processor generates a memory management fault, causing a fault exception and possibly causing termination of the process in an OS environment. The MMU provides up to eight different regions and an optional predefined background region from 32B to 4GB range. The Cortex-M3 MPU memory map is unified, meaning that instruction accesses and data accesses have the same region settings.

The MPU registers can only be accessed from privileged mode.

Micro Direct Memory Access (μDMA)

- ◆ **32-channels - dedicated for supported peripherals**

- ◆ 8,16 or 32 bits data sizes
- ◆ Two levels of priority, Mask-able device requests

- ◆ **Multiple transfer modes:**

- ◆ Basic
- ◆ “Ping” – “pong”
- ◆ Scatter - gather

- ◆ **Interrupt on transfer completion with a separate interrupt per channel**

Channel Control Structure

Offset	Description
0x0	Source end pointer
0x4	Destination end pointer
0x8	Control word
0xC	unused

Control Structure Memory Map

Offset	Channel	
0x0	0	Primary
0x10	1	
...		
0x1F0	31	
0x200	0	Alternate
0x210	1	
...		
0x2F0	31	

The Cortex-M3 Prime Cell 32-channel (μDMA) controller can perform transfers between memory and peripherals.

The μDMA controller uses an area of system memory to store a set of channel control structures in a table. This control table can be located anywhere in system memory, but it must be contiguous and aligned on a 1024byte address boundary.

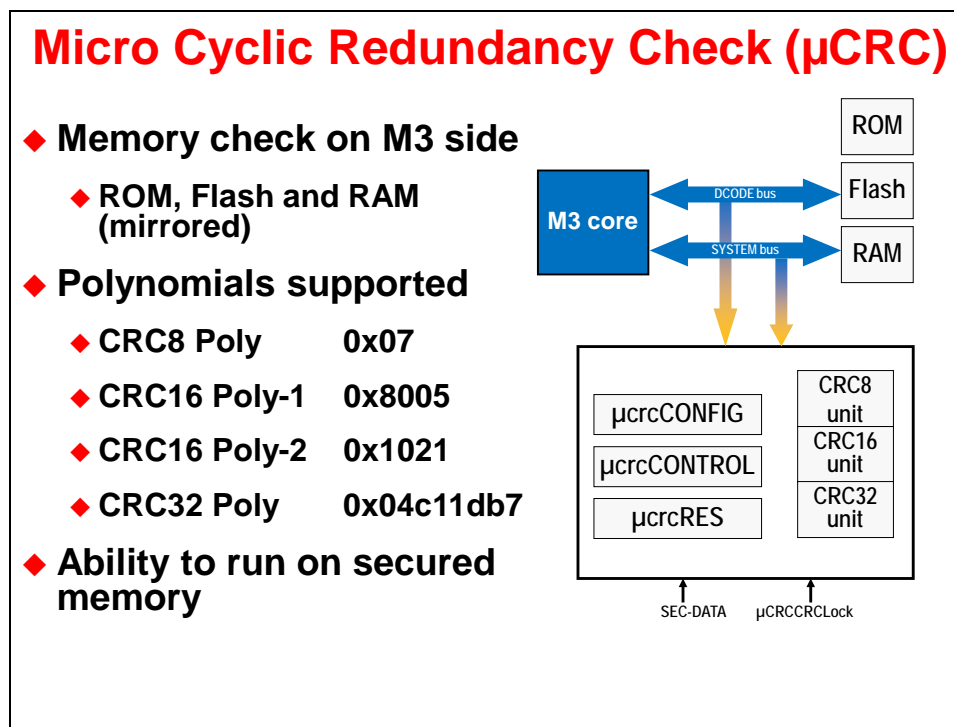
The control table may have two entries (primary, alternate) for each μDMA channel. Each entry in the table structure contains source and destination pointers, transfer size, and transfer mode.

Three transfer modes are supported:

- Basic, for simple transfer scenarios where the μDMA controller performs transfers as long as there are more items to transfer and a transfer request is present.
- Ping-Pong, for continuous data flow to/from peripherals. In this mode the transfer use 2 control structures so called the primary and the alternate control structure. The transfer is started using the primary control structure.
- Scatter-gather, from a programmable list of arbitrary transfers initiated from a single request. It is used when data must be transferred to or from varied locations in memory instead of a set of contiguous locations in a memory buffer.

The μDMA controller can transfer data to and from the on-chip SRAM. However, because the Flash memory and ROM are located on a separate internal bus, it is not possible to transfer data from the Flash memory or ROM with the μDMA controller.

Each channel has a priority level bit to provide two levels of priority: default priority and high priority. If the priority level bit is set, then that channel has higher priority than all other channels at default priority. If multiple channels are set for high priority, then the channel number is used to determine relative priority among all the high priority channels.



The F28M35x embeds a two dedicated modules for CRC, on the M3 subsystem and on the Control subsystem. There are 4 fixed pre-defined polynomials. Depending on the standards your application have to be compliant with, the CRC polynomial used in the CRC computation can be one of these 4 polynomials.

Once the uCRC module is configured using its configuration register (“μcrcCONFIG”), it intercepts the AHB buses of the M3 core to detect any read accesses from memory locations. These read data are used to perform a CRC. The computed CRC gets accumulated in the CRC result register (ucrcRES).

The SEC-DATA (Secure Data) strobe indicates whether the data appearing on the HRDATA bus is originating from secured memory or not. In case data is coming from a secured area, then the μCRCCRClock register is used to give the μCRC module the ability to calculate CRC on secured memories (including execute Only flash sectors).

The M3 ROM, flash and RAM are mapped to a mirrored memory location. It means that the memory blocks are physically the same but have 2 addresses. The μCRC module grabs data from the ABH bus only if the data is coming from an address that match the mirroring address.

Interrupt System

The Cortex-M3 processor supports interrupts and system exceptions. The processor and the Nested Vectored Interrupt Controller (NVIC) prioritize and handle all exceptions. An exception changes the normal flow of software control. The processor uses Handler mode to handle all exceptions except for reset. The NVIC registers control interrupt handling.

M3 Interrupt System

- ◆ **Nested Vectored Interrupt Controller (NVIC):**
 - ◆ **Prioritize and handle all exceptions**
 - ◆ **Automatic interrupt service**
 - ◆ **Pre-emptive/nested interrupts implementation**
 - ◆ **Full access from privileged mode**
- ◆ **Exceptions:**
 - ◆ **10 Cortex M3 core exceptions types**
 - ◆ **Up to 91 peripherals interrupts (GPIOs, UART, USB,..)**
 - ◆ **Priority grouping**
- ◆ **Process and Main (Handler) stacks**

There are 10 exceptions types. All exceptions have a configurable priority except Reset, Hard fault, and NMI which are granted with the highest priority. A lower priority value indicates a higher priority. By default all exceptions with a configurable priority have a priority of 0.

Interrupt service is automatically handled by the NVIC. When an interrupt is serviced, the initial state is automatically saved at the interrupt entry (and restored at the end). The vector table entry is automatically read as well. The NVIC supports nested interrupts and pre-emption which makes thread scheduling/service very flexible and perfect for RT operations. Full access to the NVIC controller is restricted to the privileged mode.

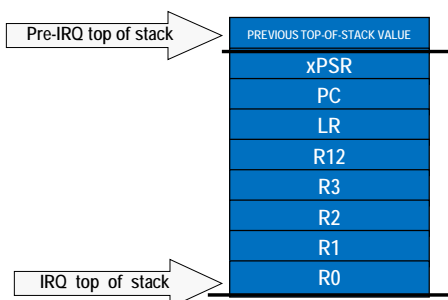
The Master subsystem supports both level-sensitive and edge-triggered interrupts.

Note that the user has to clear the interrupt source at the beginning of the interrupt handler.

M3 Exception Model

- ◆ **Exception Model handles all interrupts, synchronous faults and Supervisor Call (SVC) exceptions**
 - ◆ Exceptions cause current machine state to be stacked
 - ◆ Stacked registers conform to EABI
- ◆ **Exception handlers are trivial as register manipulation carried out in hardware**
 - ◆ No assembler code required
 - ◆ Simple 'C' interrupt service routines:


```
void IRQ(void) { /* my handler */ }
```



Offset	Vector	IRQ number	Exception number
0x0118	IRQ91	91	107
0x0048	IRQ1	1	17
0x0044	IRQ0	0	16
0x0040	Systick	-1	15
0x003C	PendSV	-2	14
0x0038	Reserved		
0x002C	SVCall	-5	11
0x0020	Reserved		
0x0018	Usage fault	-10	6
0x0014	Bus fault	-11	5
0x0010	Memory mgt fault	-12	4
0x000C	Hard Fault	-13	3
0x0008	NMI	-14	2
0x0004	Reset		1
0x0000	Initial SP value		

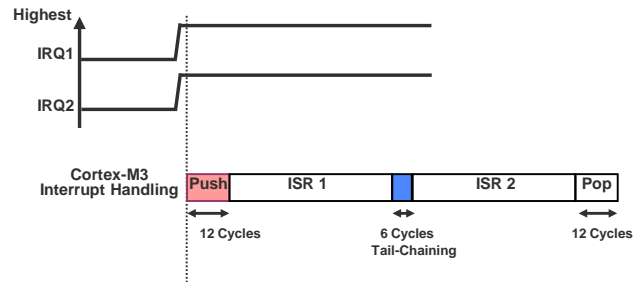
Exception Model - When an interrupt is serviced, the new Stack Pointer (SP) points to the IRQ top of stack. The Pre IRQ top of stack is where the old SP was pointing to. The context is automatically saved on the stack.

The NVIC mapping at address 0 is in the boot ROM. After the boot ROM passes control to the main application the NVIC can be initialized and remapped to RAM. Using a dedicated M3 RAM block for the NVIC is recommended to reduce latency.

On the F28M35x, other than the normal interrupts hooked to the NVIC, certain error conditions trigger exceptions:

- NMI
- clock fail condition detected
- External GPIO input signal is requesting an NMI (Note: similar to Stellaris devices)
- Error condition is generated on a C28 subsystem PIE NMI vector fetch
- C28 NMIWD timed out and issued a reset to C28 CPU
- Bus Fault: Also generated for memory access errors and RAM and flash uncorrectable data errors

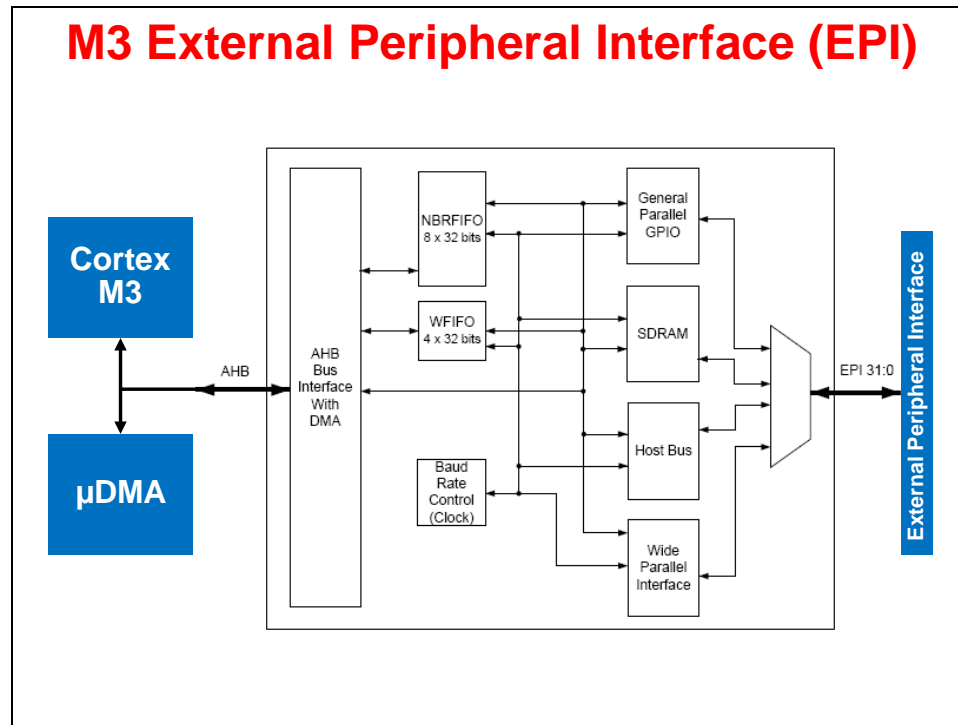
M3 Interrupt Cycle Count



- ◆ 12 cycles from IRQ1 to ISR1 (interruptible/continual LSM)
- ◆ 6 cycles from ISR1 exit to ISR2 entry
- ◆ 12 cycles to return from ISR2

Interrupt service is fast and efficient. The processor state is automatically stored to the stack on an exception and automatically restored from the stack at the end of the Interrupt Service Routine (ISR). The vector is fetched in parallel to the state saving, enabling efficient interrupt entry. The processor supports tail-chaining, which enables back-to-back interrupts to be performed without the overhead of state saving and restoration.

External Peripheral Interface (EPI)



The External Peripheral Interface (EPI) is a high-speed parallel bus for external peripherals or memory. The EPI is similar to a standard microprocessor address/data bus.

EPI can run several modes of operation to interface many types of external devices:

- **Host-Bus Interface:** Traditional x8 and 16 MCU bus interface capabilities which allow access to SRAM, NOR Flash, and other devices. In host bus mode interface mode, address and data can be de-multiplexed (allowing a 1 MB addressing) or multiplexed to address up to 256 Mbytes. Whatever the constraint of the external memory can be, the EPI can easily be configured to communicate with it through flexible setting such as read/write W/S for speed control.
- **Machine-to-Machine:** Wide parallel interfaces for fast communications (up to 150 Mbytes/second) with CPLDs and FPGAs and address width configurable in the range from 4 to 16 bits.
- **General parallel GPIO:** this mode supports communication with a parallel type of peripheral (actuator, data acquisition) from 1 to 32-bit width through a FIFO buffer. Again, the EPI offers much flexibility to control speed to fit any external device constraints.

Enhanced capabilities include μDMA support, clocking control and support for external FIFO buffers.

M3 Serial Communication

Universal Asynchronous RX/TX (UART)

- ◆ 5 independent UARTs
- ◆ Max baud rate 12.5 Mbps
- ◆ Fully programmable serial interface characteristics:
 - ◆ 5, 6, 7, or 8 data bits
 - ◆ Even, odd or no-parity bit generation/detection
 - ◆ 1 or 2 stop bit generation
- ◆ Efficient transfers using μ DMA
- ◆ Flexible modes
 - ◆ IrDA serial-IR encoder/decoder for up to 115.2 Kbps half-duplex
 - ◆ ISO 7816 Support (SmartCard communication)
 - ◆ Full modem handshake (UART1 only)

The F28M35x family includes 5 independent UARTs. Each UART is able to use separate transmit and receive FIFOs with programmable FIFO length to reduce CPU interrupt service loading with trigger levels of 1/8, 1/4, 1/2, 3/4, and 7/8. The UARTs support the standard asynchronous communication data frame. Providing the M3 system clock is running at 100 MHz, each UART can communicate at up to 12.5 Mbps.

Other features are:

- False start bit detection
- Line-break generation and detection
- IrDA serial-IR (SIR) encoder/decoder for data rates up to 115.2 Kbps half-duplex
- ISO 7816 Support (SmartCard communication)
- Full modem handshake (UART1 only)
- UARTs are accessible from/by μ DMA.

Inter-Integrated Circuit (I²C)

- ◆ **2 independent I²C modules**
- ◆ **Master or Slave - Simultaneous operation as both a master and a slave**
- ◆ **2 transmission speeds:**
 - ◆ **Standard (100 Kbps)**
 - ◆ **Fast (400 Kbps)**
- ◆ **Master and slave interrupts support**
- ◆ **Access from M3 only**
- ◆ **Loopback mode available**

The F28M35x offers 2 independent I²C modules. The I²C bus provides bi-directional data transfer through a two-wire design (a serial data line SDA and a serial clock line SCL). Each I²C module supports both sending and receiving data as either a master or a slave and can operate simultaneously as both a master and a slave.

There are a total of four I²C modes:

- Master Transmit,
- Master Receive
- Slave Transmit,
- Slave Receive

Both the I²C master and slave can generate interrupts. I²C master generates interrupts when a transmit or receive operation completes (or aborts).

I²C slave generates interrupts when data has been sent or requested by a master.

I²C modules are accessible from M3 core only. μ DMA access is not possible.

Synchronous Serial Interface (SSI)

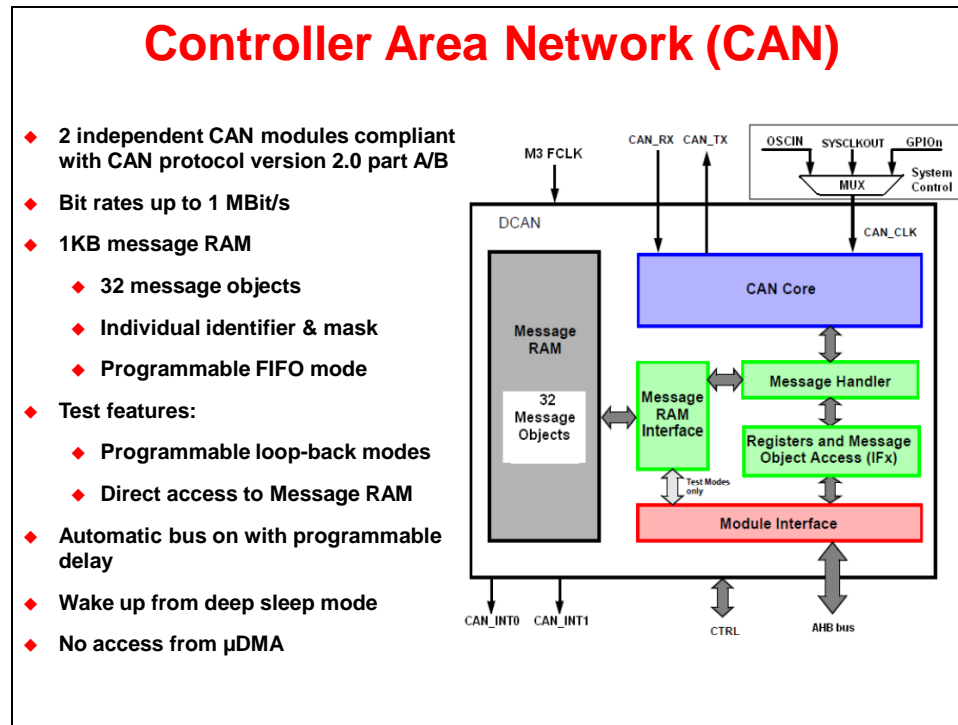
- ◆ **4 independent SSI**
- ◆ **Master or slave modes**
- ◆ **Maximum speed 25 Mbps**
- ◆ **Separate Tx and Rx FIFOs**
 - ◆ **16 bits wide**
 - ◆ **8 locations deep**
- ◆ **Frame size from 4 to 16 bits**
- ◆ **Efficient transfers using μ DMA**
- ◆ **Loopback mode available**

The F28M35x family includes 4 independent SSI on the master subsystem.

Each SSI is a master or slave interface for synchronous serial communication with peripheral devices that have Freescale SPI, MICROWIRE, or Texas Instruments Synchronous Serial Interfaces.

The SSI module has a programmable clock bit rate and prescaler. The SPI will support SPI communication at up to 25MHz in TX or RX mode. The data frame size is programmable from 4 to 16 bit wide. Tx and Rx FIFO are independent and 8 words deep and 16-bit wide.

Efficient data transfer can be ensured through μ DMA with single and burst capability. Internal loopback test mode is also available for diagnostic/debug testing.



The F28M35x includes 2 independent CAN modules. Each CAN module supports bit-rates up to 1 Mbit/s and is compliant to the CAN 2.0B protocol specification.

Two clock domains are provided to the CAN module:

- The CAN wrapper and registers are always clocked at the same rate as M3 SYSCLK (no divide).
- As CAN peripherals can be susceptible to clock jitter at higher bit rates, the CAN Protocol Kernel (CPK) is clocked from 3 possible sources which enables a more accurate and stable clock to be fed to the peripherals. Also, this enables the use of spread spectrum clocks without compromising CAN communications. The CAN bit clock can be:
 - the M3 System Clock again
 - from XCLKIN the external clock derived through a GPIO pin in the range from 8 to 25 MHz,
 - or the X1 input oscillator clock directly in the range from 8 to 25 MHz

The CAN Core has to be programmed to at least 8 clock cycles per bit time. For instance, to achieve a transfer rate of 1 MBaud when using the asynchronous clock domain as the clock source for CAN_CLK, an oscillator frequency of 8MHz or higher has to be used.

Each CAN module can handle up to 32 mailboxes which contains distinct message objects and their respective individual identifier are stored in the message RAM. Each mailbox is 32 bytes so that the total message RAM is 1kByte long. Multiple message objects with the same identifiers and masks can be concatenated to a FIFO Buffer.

Please note that the CAN message RAM parity check and additional robustness features for safety.

Each CAN module can send 2 distinct interrupts to the NVIC:

- Error interrupts & Status interrupts can only be routed to CANn_INT0

- Mailbox interrupts can be routed to either CANn_INT0 or CANn_INT1.

The CAN module features programmable loop-back modes for self-test operation. This feature is a key to comply with safety requirements necessary in many applications. The CAN modules come with other features that are useful for test such as direct access to the message RAM.

After the CAN has entered Bus-Off state, the CPU can start a Bus-Off-Recovery sequence. The CAN provides an automatic Auto-Bus-On that can delay this sequence. The CAN modules are able to wake up the device from deep-sleep mode.

Both CAN modules belong to the M3 subsystem, they cannot be accessed directly by the C28 but the C28 can access CAN through shared RAM.

There is no access from μ DMA.

CANTX/CANRX pins cannot be configured as general purpose IO pins when unused.

Ethernet MAC

- ◆ **MAC layer only**
- ◆ **10BASE-T and 100BASE-TX/RX IEEE 802.3 Full/Half-Duplex support**
- ◆ **Programmable MAC address**
- ◆ **Hardware support for Precision Time Protocol (IEEE 1588 PTP Promiscuous mode support)**
- ◆ **2KB Transmit FIFO / 2KB Receive FIFO**
- ◆ **Efficient transfers using μ DMA**

The F28M35x Ethernet Controller consists of a fully integrated Media Access Controller (MAC); it does not provide a network physical (PHY) interface. The MAC layer provides transmit and receive processing for Ethernet frames. The MAC layer also provides the interface to the PHY layer via an internal Media Independent Interface (MII).

The MAC conforms to IEEE 802.3 specifications and supports 10BASE-T and 100BASE-TX standards (full and half duplex at 10 or 100 MBPS).

The Ethernet controller provides many options which allow the user to configure the module as needed. Some of these options include a programmable MAC address, configurable interrupts, CRC error-rejection control, promiscuous mode, and LED indicator selection.

Other features of the Ethernet controller include power-saving modes, power-down modes, automatic cross-over correction, programmable transmit amplitude, and automatic polarity correction.

The unit also includes hardware assistance for an IEEE 1588 PTP-enabled system. IEEE 1588 adds a clock and ID address to permit synchronized operation between devices that support these additionally feature.

Promiscuous mode – ability to receive all packets sent on network – like CAN. Versus only those sent to specific MAC address.

Universal Serial Bus (USB)

- ◆ **Integrated controller with integrated PHY**
- ◆ **USB 2.0 full-speed (12 Mbps) / low-speed (1.5 Mbps)**
- ◆ **Devices with OTG/Host/Device or Host/Device**
- ◆ **Transfer: Control, Interrupt, Bulk and Isochronous**
- ◆ **Up to 32 Endpoints**
 - ◆ **1 dedicated control IN endpoint**
 - ◆ **1 dedicated control OUT endpoint**
 - ◆ **4 KB Dedicated Endpoint Memory**
- ◆ **μDMA efficient data transfer**

The F28M35x USB controller includes an integrated PHY for minimum system cost/easy connection. Note that the module requires a 60MHz input clock, generated by a second PLL.

The USB-unit operates as a full-speed or low-speed function controller during point-to-point communications with USB Host, Device, or OTG functions. The controller complies with the USB 2.0 standard, which includes SUSPEND and RESUME signaling. The controller complies with OTG standard's session request protocol (SRP) by supporting both the session request protocol (SRP) and the host negotiation protocol (HNP). The session request protocol allows devices on the B side of a cable to request the A side device turn on VBUS.

In addition, the USB controller provides support for connecting to non-OTG peripherals or Host controllers. The USB controller can be configured to act as either a dedicated Host or Device, in which case, the USB0VBUS and USB0ID signals can be used as GPIOs.

It features 32 endpoints including two hard-wired for control transfers (one endpoint for IN and one endpoint for OUT) plus 30 endpoints are defined by firmware along with a dynamic sizable FIFO support multiple packet queuing. 4 KB of memory are dedicated to endpoint memory: one endpoint may be defined for double-buffered with 1023-bytes isochronous packet size.

Software-controlled connect and disconnect allows flexibility during USB device start-up.


μDMA access to the FIFO allows minimal interference from system software. The USB peripheral provides an interface connected to the μDMA controller with separate channels for 3 transmit endpoints and 3 receive endpoints.

Lab 2: Generating a Master M3 Subsystem Interrupt

➤ Objective

The objective of this lab exercise is to add interrupt functionality to the Master M3 subsystem. A general purpose timer will be used to generate a hardware-based trigger for toggling LED LD3 on GPIO_PIN7. In the first lab exercise this was done using a software “for loop” delay toggle. This lab exercise will only be implemented on the Master M3 subsystem. In the next lab exercise the Control C28 subsystem will be added to the project performing a similar task.

Lab 2: Master M3 Subsystem



◆ **Add interrupt functionality to the Master M3 subsystem**

- ◆ **Create M3 project**
 - ◆ Master subsystem blinks LED LD3 using general purpose timer generated trigger
 - ◆ GP Timer0 replaces software delay “for” loop in Lab1

➤ Procedure

Note: To eliminate duplicate efforts and possibly reduce errors, the remaining lab exercise projects in this workshop will use a copy of an existing project as a starting point. If the previous project is not listed in the Project Explorer window it can be opened using Project → Import Existing CCS Eclipse Project.

Create a New Project – M3 Master Subsystem

1. The Lab1_M3 project will be used as a starting point for this lab exercise. Right click on project Lab1_M3 and select “Copy” followed by “Paste”. A “Copy Project” window will open.

In the Project name field rename the project to **Lab2_M3**. Uncheck the “Use default location” box. Click the Browse... button and navigate to:

C:\F28M35x\Labs\Lab2\M3

Click OK, and then OK again to copy the project.

2. Close the Lab1_M3 project by right-clicking on the project in the Project Explorer window and select `Close Project`.
3. Next, in the Lab2_M3 project delete the “Debug” folder.
4. Then delete file Lab1_M3.c from the project. A new function “main()” file will be added next.
5. Add (“copy files”) the function “main()” source file to the project by right-clicking on Lab2_M3 in the “Project Explorer” window and selecting `Add Files...`

File: Lab2_M3.c

Folder: C:\F28M35x\Labs\Source_files

Lab2_M3.c file is identical to the Lab1_M3.c file except for additional code sections needed for this lab exercise. The new additional code sections will be explained next.

Inspect Source File Lab2_M3.c

6. Open and inspect Lab2_M3.c. Notice the code modifications as compared to the Lab1_M3.c file. Like before, at the beginning of the code the required header files are included. Except this time a new line for the M3 Timer header declaration has been added:

```
19#include "driverlib/gpio.h"
20#include "driverlib/timer.h"           // new in Lab2_M3
21
```

The next code block section is used as the Timer 0 Interrupt Service Routine. A local static variable “toggle” is used to switch the LED on or off with every other interrupt. The interrupt is cleared using the MWare library function “TimerIntClear”:

```
26//*****
27// The interrupt handler for timer0 interrupt.
28//*****
29void Timer0IntHandler(void)
30{
31    static unsigned int toggle = 0;
32    // Clear the timer interrupt.
33    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
34    if(toggle == 0)
35    {
36        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0); // LD3 ON
37        toggle = 1;
38    }
39    else
40    {
41        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, ~0); // LD3 OFF
42        toggle = 0;
43    }
44}
```

In the function “main()” a new code sequence is added to setup the M3 Timer 0. Timer 0 is enabled (“SysCtlPeripheralEnable”), set to 32-bit operating mode (“TimerConfigure”) and the period is set (“TimerLoadSet”) by reading the system clock speed in Hz (in our example 75,000,000 Hz). By using this setup, the timer will count 75,000,000 / 2 input clock cycles of 13.33 nanoseconds and the resulting interrupt trigger will be 500 milliseconds:


```

69 // Enable timer 0 used by this example.
70 SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
71 // Configure the timer0 as 32-bit periodic timer.
72 TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);
73 TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet(SYSTEM_CLOCK_SPEED)/2);

```

Finally, a code sequence will enable the M3 interrupt system and start the timer. The address of the interrupt service routine is loaded into the vector table by the MWare library function “IntRegister”:

```

88 // Setup the interrupts for the timer timeouts.
89 IntEnable(INT_TIMER0A);
90 TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
91 IntRegister(INT_TIMER0A, Timer0IntHandler);
92
93 // Enable the timers.
94 TimerEnable(TIMER0_BASE, TIMER_A);

```

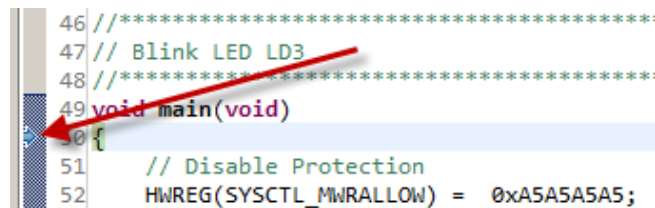
The “while(1);” instruction will loop forever, and all activities will be performed in the interrupt handler.

Build the Project and Program the Flash Memory

7. Click the “Build” button and watch the tools run in the “Console” window. Check for errors in the “Problems” window.
8. Program the flash memory by clicking the “Debug” button (green bug) or click Run→Debug. (If needed, when the “Progress Information” box opens select “Details >>” in order to watch the programming operation and status). After successfully programming the flash memory the “Progress Information” box will close.

Run the Code – Test the M3

9. The blue arrow (program counter) should now be pointing to the beginning of “main()”:



```

46 //*****:
47 // Blink LED LD3
48 //*****:
49 void main(void)
50 {
51 // Disable Protection
52 HWREG(SYSCTL_MWRALLOW) = 0xA5A5A5A5;

```

10. Run the M3 code using the “Resume” button on the toolbar (or F8 key). LED LD3 should be blinking at a period of exactly 1 second.
11. Halt the M3 code using the “Suspend” button on the toolbar (or alt+F8).

Terminate Debug Session

12. Terminate the active debug session using the Terminate button. This will close the debugger and return CCS to the “CCS Edit” perspective view.

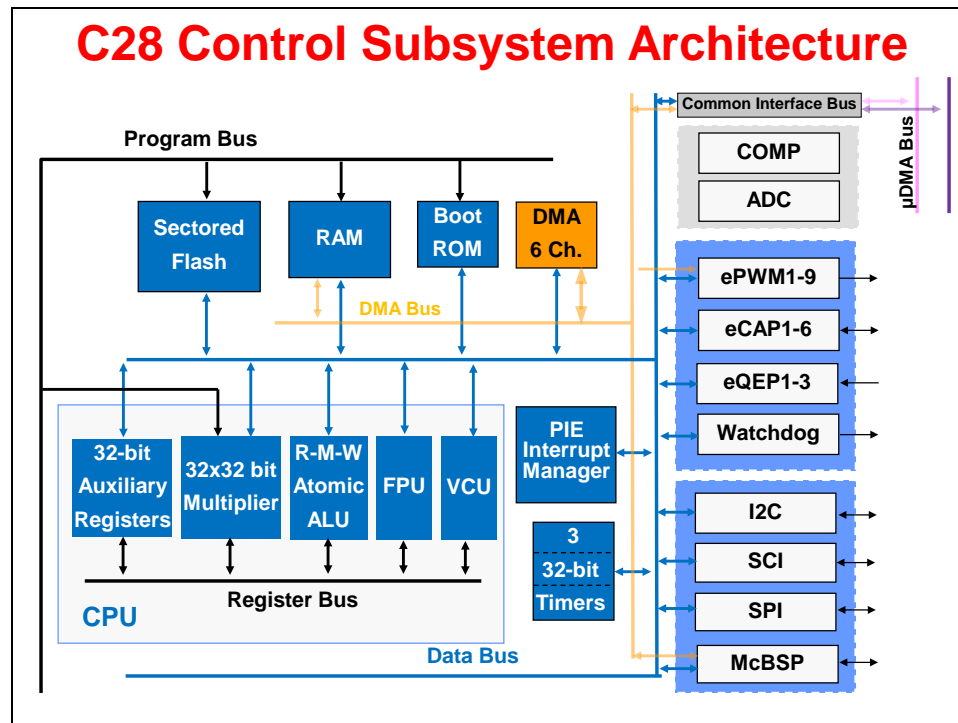
End of Exercise

Architectural Overview – Control C28 Subsystem

Architectural Block Diagram

The F28M35x Control subsystem very closely resembles the C2000 F283xx “Delfino” device family with some added enhancements from the F280xx “Piccolo” family.

Although most Delfino family devices include an on-chip ADC, the choice was made for the F28M35x to implement analog peripherals including the on-chip ADC and the analog comparators in a separate Analog subsystem. One of the key advantages of this is to allow both the Control and the Master subsystem to access these peripherals.



The legacy C28x architecture is based on a 32-bit fixed point CPU. When performing signed math, the CPU uses 2s-complement notation. The Control engine is based on the C28x plus floating-point (C28x+FPU) co-processor which extends the capabilities of the C28x 32-bit fixed-point CPU by adding registers and instructions to support IEEE single-precision 754 floating point operations. Double-precision operations will require support libraries.

No changes have been made to the C28x+FPU base set of instructions, pipeline, or memory bus architecture. Therefore, programs written for the C28x+FPU (i.e. Delfino) are completely compatible with the F28M35x.

The Viterbi, Complex math, CRC unit (VCU) enhances the digital signal processing capability of the C28 CPU core. The VCU includes its own resources.

The CPU features include circular addressing and a modified Harvard architecture that enables instruction and data fetches to be performed in parallel. The CPU can read instructions and data while it writes data simultaneously to maintain the single-cycle instruction operation across the pipeline.

Note that no system watchdog is available on the C28 subsystem, as with the Delfino or Piccolo family devices, however an NMI watchdog is available.

The F28M35x C28 subsystem includes 3 CPU timers which are exactly the same as on the Piccolo and Delfino family devices.

The C28 memory map is accessible outside the CPU by means of the memory interface. This connects the CPU logic to memories, peripherals, or other interfaces. The memory interface includes separate buses for program space and data space. This allows an instruction to be fetched from program memory while data memory is being accessed at the same time.

Control peripherals and communication peripherals are accessible through data bus.

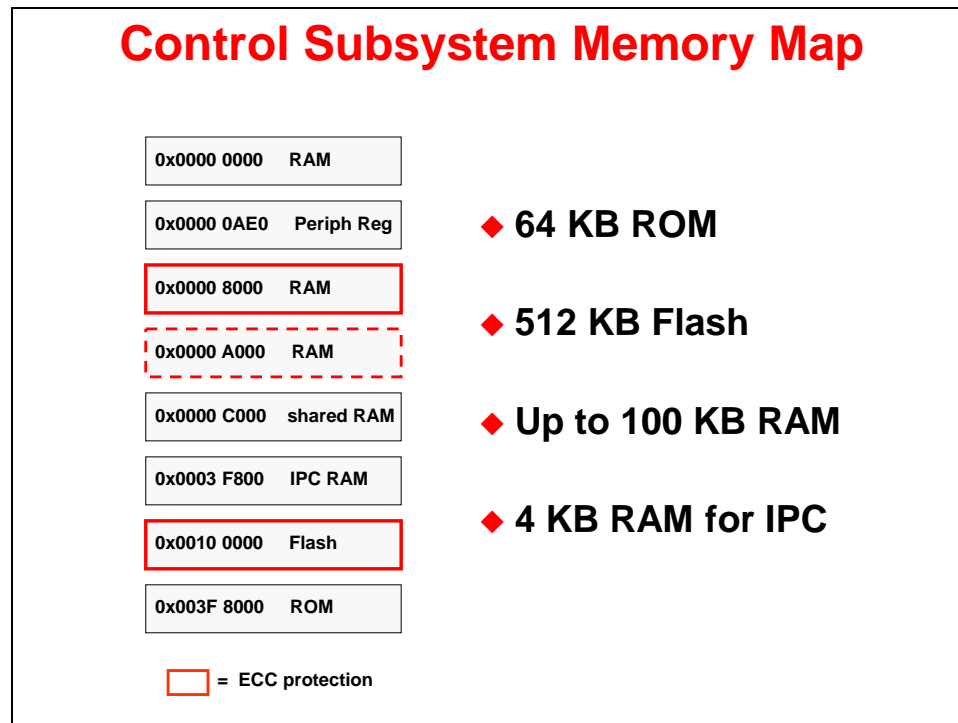
Although the Analog to Digital Converter and the Comparators are not part of the Control subsystem, they can be accessed through the Analog Common Interface Bus (ACIB). It is accessible from all the 4 masters namely C28 CPU, C28 DMA, M3 CPU and M3 DMA simultaneously.

Note that the CAN modules are mapped to the M3 subsystem and cannot be accessed from the Control subsystem.

The EPI which is used to interface to external memory or FPGA is also owned by the Master subsystem and cannot be addressed directly from the C28 subsystem.

At reset all of the GPIO pins belong to the Master M3 subsystem. The M3 is responsible for allocating pins to the C28 multiplexer system. Once the pins have been allocated the assignment can be locked for safety.

Memory System



Control Subsystem - ROM

- ◆ **Size:** 64 Kbytes (start from 0x003F 8000)
- ◆ **Access:** single cycle
- ◆ **Contains:**
 - ◆ C28 boot loader code
 - ◆ Math tables
 - ◆ IQmath
 - ◆ Floating-Point
 - ◆ PLC
 - ◆ IPC code
- ◆ **Access:** from C28 only

The on-chip ROM is 64 Kbytes. At the beginning of the boot loader code, the C28 ROM contains some math tables for PLC, IQ-math and FPU-math. It also contains code for the Inter Processor Communication (IPC).

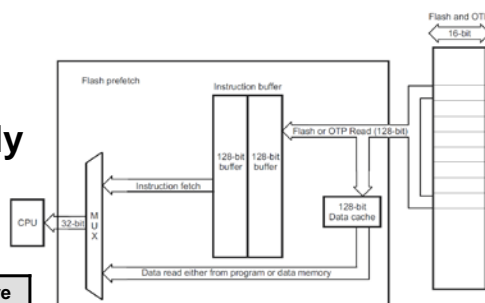
Only the C28 subsystem can access its ROM block. It cannot be accessed from the DMA or M3 subsystem.

Control Subsystem - FLASH

- ◆ **Size:** 512 Kbytes
14 sectors
- ◆ **Access:** 25 nsec
- ◆ **Access:** from C28 only

Flash performance

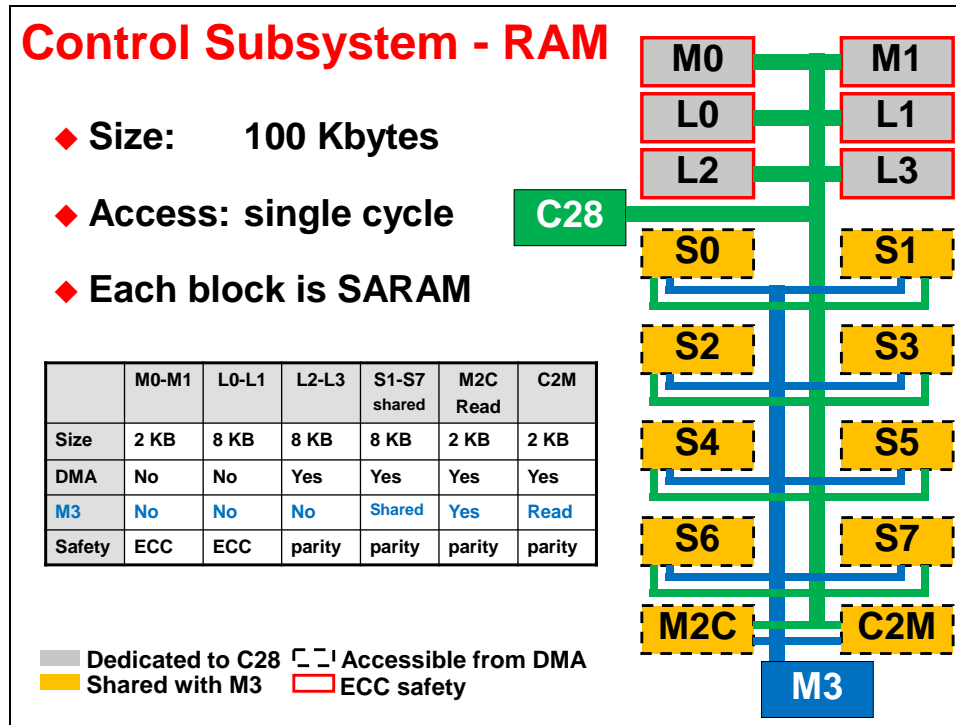
C28 Frequency	Wait State	Efficiency	Effective Speed
40 MHz	0	100 %	40 MHz
80 MHz	1	96 %	77 MHz
100 MHz	2	94 %	94 MHz
150 MHz	3	91 %	137 MHz



The F28M35x is based on 65nm FLASH Technology. The M3 Master subsystem and C28 Control subsystem operate from independent Flash banks. The C28 Flash is 512 Kbytes which is split into 14 sections. FLASH cannot be accessed from the DMA.

To boost code execution performance, acceleration hardware is added. It consists of a 128-bit wide 2-level pre-fetch mechanism. After the first 128-bit wide fetch, the Flash interface will fetch ahead the next 128-bit wide word while the currently fetched instructions are fed to the CPU. In theory, 128-bit wide word can hold four 32-bit instructions or eight 16-bit instructions. Hence, this system can provide near single cycle performance even if the Flash wait states are as high as 3 (four cycle access).

During a Flash memory operation (write, page erase, or mass erase) access to the Flash memory is inhibited. Although M3 Flash bank can only be programmed by M3 and C28 Flash bank can only be programmed by C28, the C28 and the M3 share the same charge pump. Hence, there is a Flash semaphore for pump ownership. By default on reset, the Flash Pump is mapped to M3. For the C28 to program its own bank it must request access to Flash Pump and the clock control registers via semaphores.

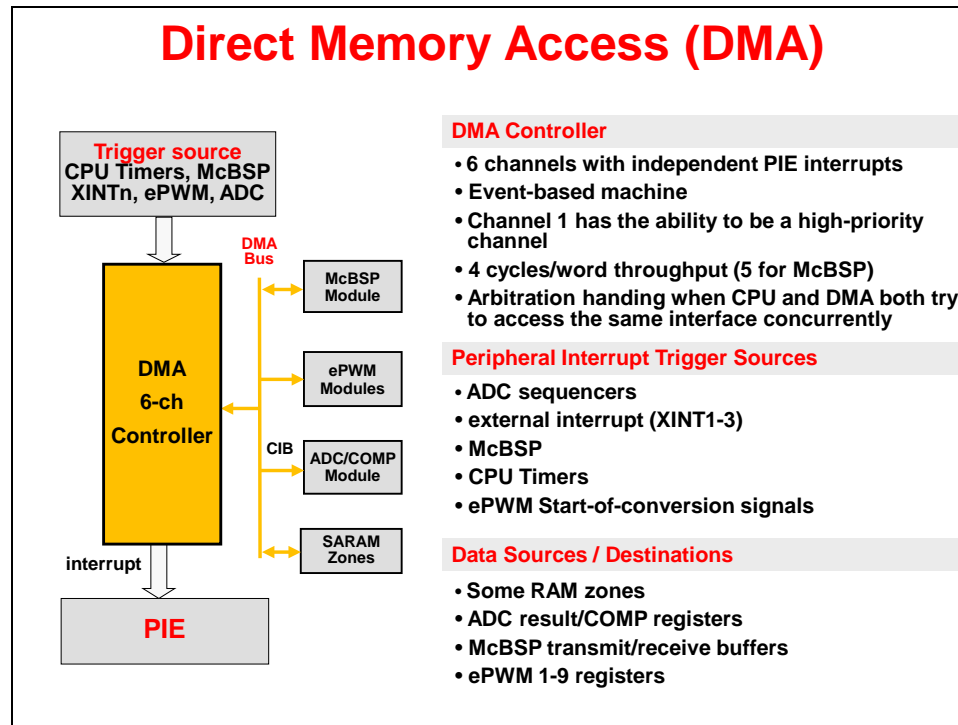


The 100 Kbytes of RAM are split in 14 distinct blocks. M0 and M1 are 2 Kbytes each whereas the other 12 blocks are 8 Kbytes each.

Two additional 2 Kbytes blocks are reserved for inter-processor communication.

- M2C is for M3 to C28 exchange
- C2M is for C28 to M3 exchange
- L blocks are dedicated

At Reset all of the 8 shared RAM blocks are mapped to the M3 subsystem.



The DMA module is an event-based machine; it requires a peripheral interrupt trigger to start a DMA transfer. The sources can be:

- The ADC sequencers from the Analog Common Interface Bus
- External interrupts (XINT1-3) that can be triggered from any of the 64 GPIOs
- McBSP
- CPU Timers
- ePWM Start-of-Conversion signals (A or B)

The trigger source for each of the six DMA channels can be configured separately and each channel contains its own independent PIE interrupt to let the CPU know when a DMA transfers has either started or completed.

Peripherals can also be used as:

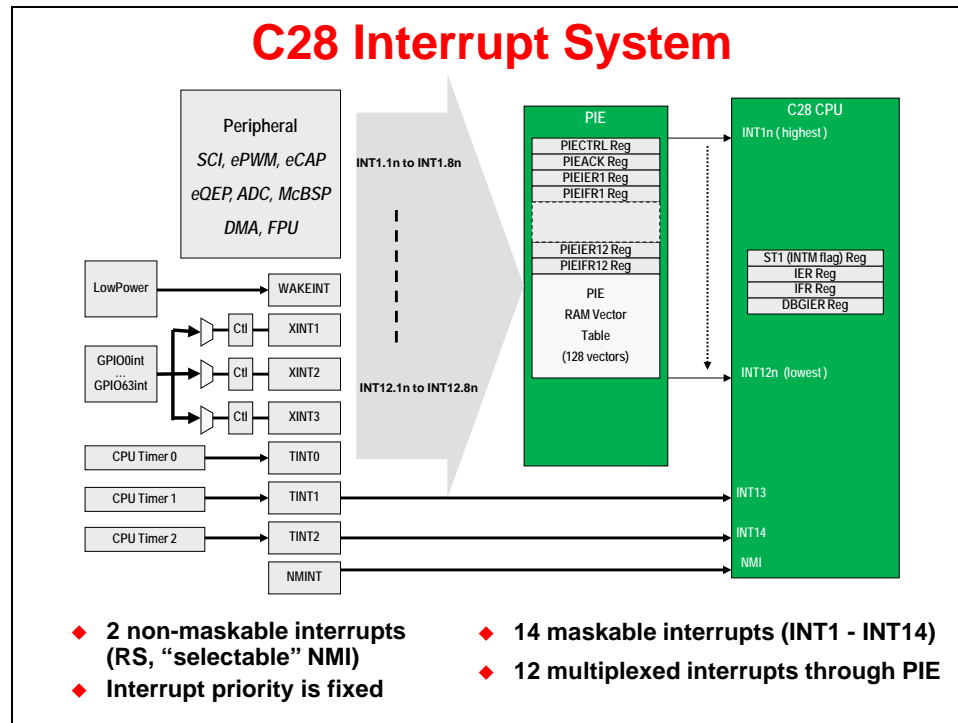
- Source: ADC result and Comparator registers
- Destination : ePWM registers
- Source and destination : McBSP

The DMA can transfer data to/from some RAM zones:

- Shared memory block S0 to S7, following the rules given by each block ownership
- L2 and L3 dedicated memory blocks
- Message RAM blocks that are dedicated to the inter-processor communication.

The DMA consists of a 4-stage pipeline. Each DMA channel can be configured with a word of either 16 bits or 32 bits. The transfer is defined a number of up to 64K burst to transfer. The burst is defined as the smallest entity to transfer at a time; it can be configured for each channel from 1 up to 32 words.

Interrupt System

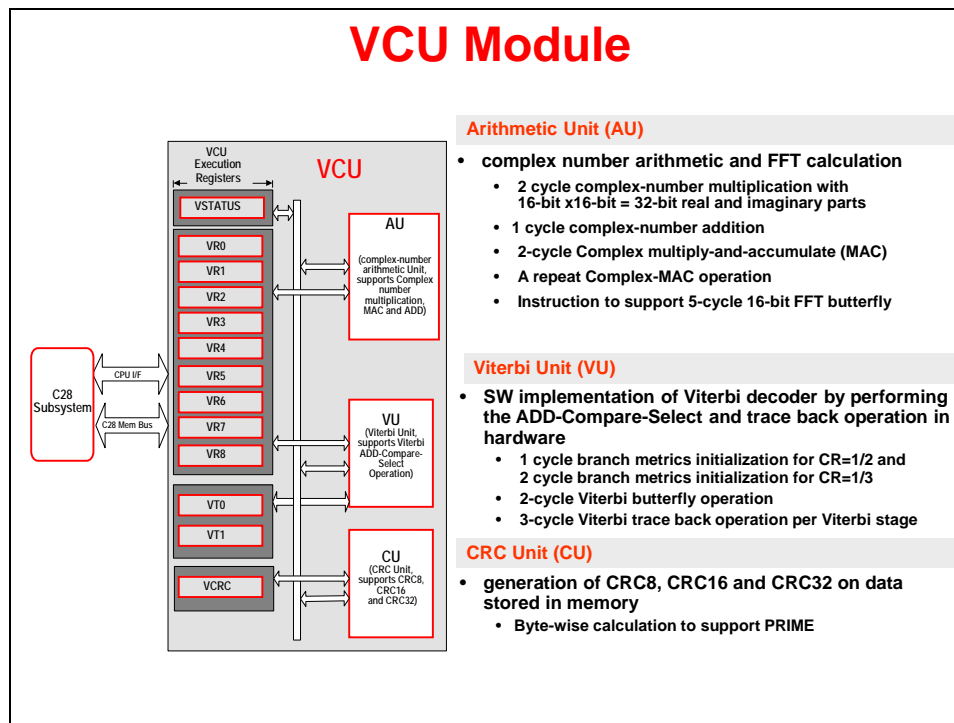


The C28 subsystem handles 15 interrupts. Interrupt 13 and 14 are direct connected to CPU on-chip timers 1 and 2. One Interrupt is dedicated to NMI (Non Maskable Interrupts) and Interrupts 1 to 12 are multiplexed by a Peripheral Interrupt Expansion (PIE) unit.

Interrupts are automatically prioritized by the C28 hardware. Group 1, which corresponds to CPU INT1, has the highest priority. Within each group there are 8 interrupts with INTx.1 being the highest priority and INTx.8 having the lowest.

There are 3 external interrupt available: XINT1, XINT 2 and XINT3. Any of the 64 GPIO can be used to trigger one of these external sources of interrupt. These XINT interrupt can be configured as either negative, positive or both edge triggered. Each of the 3 external interrupts has its own 16-bit counter that starts counting when a valid interrupt edge is detected and resets when another valid edge is detected.

Viterbi, Complex Math, CRC Unit (VCU)



Additionally to the fixed-point CPU and its floating point co-processor, the F28M35x includes a Viterbi CRC and Complex Arithmetic unit (VCU). This unit operates in three modes:

- **CRC Unit (CU):** Three standard polynoms are supported in hardware – CRC32, CRC16 and CRC8.
- **Viterbi Unit (VU):** Viterbi decoding is commonly used in baseband communications applications. Without a VCU, it takes 15 cycles for the C28 CPU to perform two Viterbi butterfly calculations. F28x MCUs with the VCU option can perform two simultaneous butterflies in 2 cycles. A Viterbi trace back implemented in S/W takes 22 cycles per stage compared to just 3 cycles per stage using the VCU.
- **Complex Arithmetic Unit (AU):** The Complex Math capability of the VCU enables the VCU to perform 16-bit FFT butterflies in 5 cycles and 16-bit complex filters in a single cycle/tap.

The VCU benefits of its own independent register space. These registers function as source and destination registers for VCU instructions. The VCU is optimized for 16-bit fixed point data operations, which are common in communication algorithms.

An application issues instructions to the VCU in the same way that an application issues an instruction to the FPU using dedicated registers with pointers to data and result buffers. This allows applications to utilize VCU-accelerated calculations at any time.

Communications is becoming increasingly important to engineers interested in the renewable energy space. One popular method of communication is Power Line Communications, or PLC, because the network already exists and a customer doesn't have to invest in installing a new network such as Ethernet. The main challenge is power lines are very noisy. Special encoding techniques are required to add extra bits to the transmitted data for error correction. Transmitting over the power line is not difficult but receiving information off the line is quite math intensive with lot of filtering, FFT, decoding and checking for data integrity.

Just as an FPU provides advanced mathematical capabilities, the VCU is extremely efficient at performing FFT, filtering, and CRC calculations.

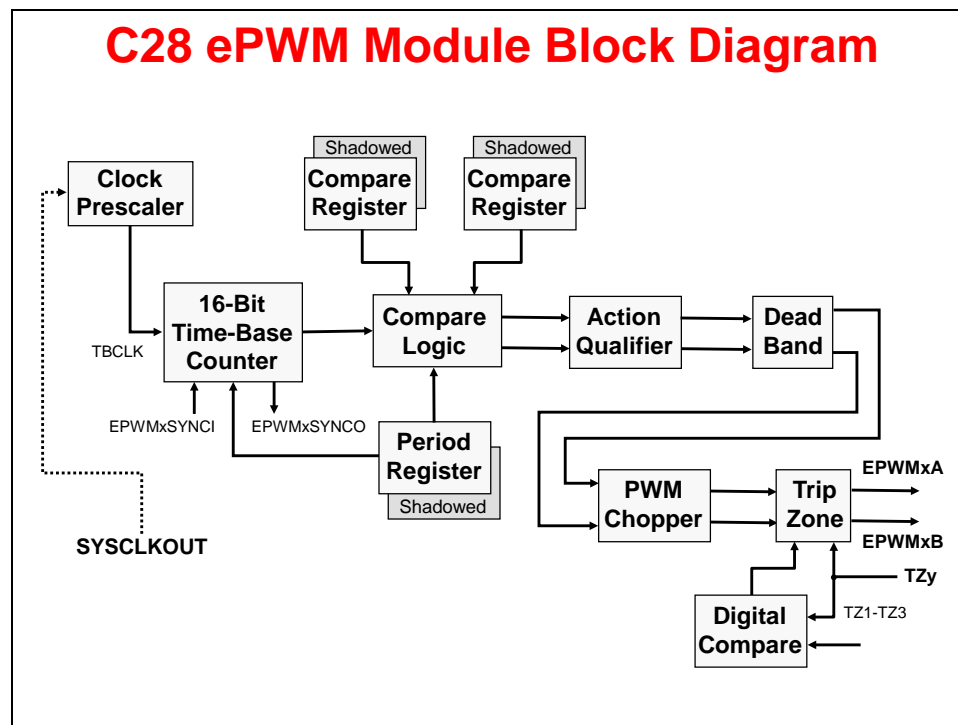
As an example, the following steps explain how the VCU processes the receive portion of a PLC modem. First, the complex math signal is received from the analog front-end and filtered by the CMU (Complex Math Unit). A complex FFT would follow. The CMU can run up to 4x faster 16-bit complex FFT butterfly vs. software since it can do 2 cycle complex number multiplication; single cycle complex number addition; and 2 cycle complex MAC. After the FFT the information is decoded via the VU or the Viterbi Decoding Unit. The VU can do up to 7x faster butterfly and 5x faster trace back. Then after the data is decoded, the CRC Unit (CU) can perform CRC checking for its data integrity in memory. This performance is increased up to 25 times because it is done in hardware.

Other application areas for the VCU are:

- Additional part for digital motor control in order to do vibration analysis on the motor.
- Radar sensor applications can take advantage of the FFT capabilities in the VCU to run the radar sensor algorithms.
- Orthogonal Frequency Division Multiplexing (OFDM) modulation techniques.
- Memory integrity checks can be accelerated using the VCU's CRC-32 capabilities.

Texas Instruments provides a complete library of VCU-functions that can be called from C.

ePWM Module



The ePWM module is similar to the one used on the Piccolo devices with some additional enhancements. One enhancement to consider is the high-resolution mode which enables achieving resolution up to ~150ps. This applies to:

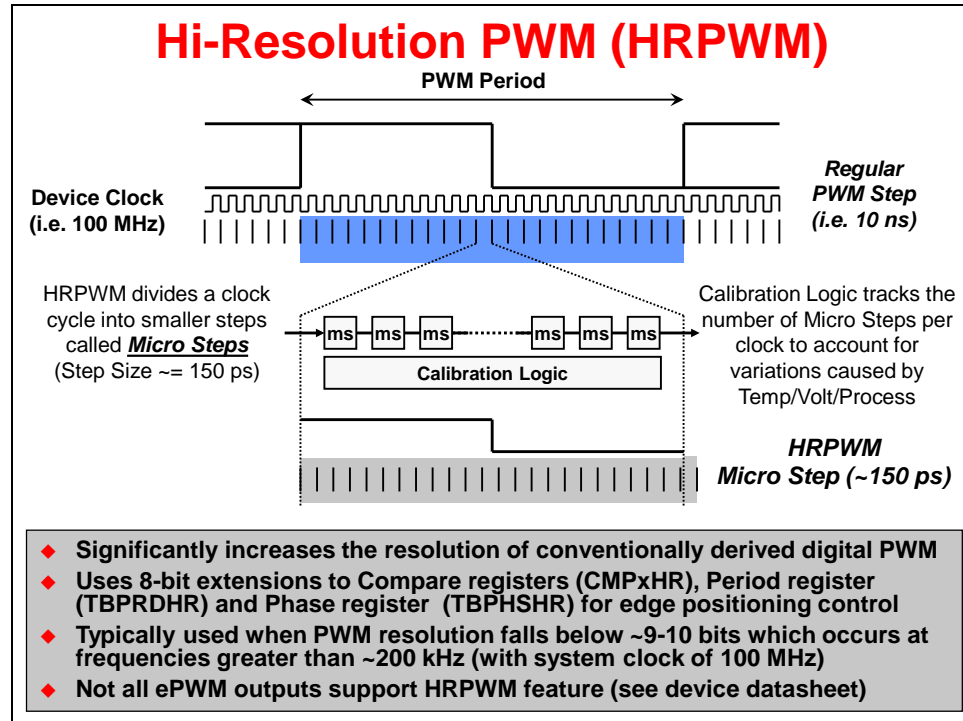
- The PWM frequency
- COMPA and COMPB to configure the duty cycle value
- The phase registers is also compliant with high resolution
- Both channel (A and B) can generate a high-resolution PWM waveform where it was only available to a single channel on previous C2000 devices
- The dead-band sub-module that enables a delay to the PWM falling and/or rising edge also makes use of the high-resolution feature

In addition to CMPA and CMPB, two new ePWM events are available (CMPC and CMPD) to trigger an interrupt or send a SOC event to the ADC.

Also a new feature is the ability to simultaneously update the period and compare registers across synchronized modules.

The event-trigger has been enhanced. The SOC/INT counters have been increased to 4-bits enabling the INT/SOC to generate a trigger up to every 15 events. This allows software initialization of event counters on SYNC event (user can initialize the ET SOC/INT counters synchronized between different PWM control loops).

The ePWM are accessible from the DMA which makes control loop highly efficient and fast.



The high-resolution feature extends the time resolution capabilities of the conventionally derived digital pulse width modulator (PWM). HRPWM is typically used when PWM resolution falls below ~ 9-10 bits. The key features of HRPWM are:


- Extended time resolution capability
- Used in both duty cycle and phase-shift control methods
- Finer time granularity control or edge positioning using extensions to the Compare A, Compare B and Phase registers
- Implemented using the A & B signal path of PWM, i.e., on the EPWMxA and EPWMxB output
- Dead band high-resolution control for falling and rising edge delay in half cycle clocking operation
- Self-check diagnostics software mode to check if the micro edge positioner (MEP) logic is running optimally
- Enables high resolution output swapping on the EPWMxA and EPWMxB output
- Enables high-resolution output on EPWMxB signal output via inversion of EPWMxA signal output
- Enables high-resolution period, duty and phase control on the EPWMxA and EPWMxB output on selected devices

Lab 3: Generating a Control C28 Subsystem Interrupt

➤ Objective

The objective of this lab exercise is to add interrupt functionality to the Control C28 subsystem. A general purpose timer will be used to generate a hardware-based trigger for toggling LED LD2 on GPIO_PIN 70. In the first lab exercise this was done using a software “for loop” delay toggle. This lab exercise combines both the Master M3 and Control C28 subsystems implementing interrupt driven hardware-based triggers for the LEDs.

Lab 3: Control C28 Subsystem



- ◆ **Add interrupt functionality to the Control C28 subsystem**
 - ◆ **Create C28 project**
 - ◆ Control subsystem blinks LED LD2 using CPU timer generated trigger
 - ◆ CPU Timer0 replaces software delay “for” loop in Lab1
 - ◆ **M3 project runs unchanged from Lab2**

➤ Procedure

Create a New Project – C28 Control Subsystem

1. The Lab1_C28 project will be used as a starting point for this part of the lab exercise. Right click on project Lab1_C28 and select “Copy” followed by “Paste”. A “Copy Project” window will open.

In the Project name field rename the project to **Lab3_C28**. Uncheck the “Use default location” box. Click the Browse... button and navigate to:

C:\F28M35x\Labs\Lab3\C28

Click OK, and then OK again to copy the project.

2. Close the Lab1_C28 project by right-clicking on the project in the Project Explorer window and select Close Project.
3. Next, in the Lab3_C28 project delete the “Debug” folder.
4. Then rename file Lab1_C28.c to **Lab3_C28.c**.

Add Files to Project – C28 Control Subsystem

5. Add (“copy files”) the following source files to the project by right-clicking on Lab3_C28 in the “Project Explorer” window and selecting Add Files...

File: F28M35x_CpuTimers.c
 File: F28M35x_DefaultIsr.c
 File: F28M35x_PieCtrl.c
 File: F28M35x_PieVect.c
 Folder: C:\F28M35x\Device_support\F28M35x_common\source

Edit Source File Lab3_C28.c

Note: For this and the remaining lab exercises, code will be added to various files. To assist in this process, the needed code for the lab exercise can be copied from the LabCode.txt file located in the folder C:\F28M35x\Labs\Source_files.

6. Edit Lab3_C28.c as follows – at the beginning of the code add a new define command:

```
#define C28_FREQ    150           //C28 CPU frequency in MHz
```

7. Next, at the beginning of the code before “main()” add a new function declaration for the local interrupt service routine of CPU-Timer 0:

```
interrupt void cpu_timer0_isr(void);
```

8. At the beginning of function “main()” delete the local variable “uldelay”.
9. After the function call “InitFlash()” add new instructions:

```
InitPieCtrl();  
InitPieVectTable();  
EALLOW; // This is needed to write to EALLOW protected registers  
PieVectTable.TINT0 = &cpu_timer0_isr;  
EDIS;  
InitCpuTimers();  
ConfigCpuTimer(&CpuTimer0, C28_FREQ, 125000);  
PieCtrlRegs.PIEIER1.bit.INTx7 = 1;  
IER |= 1;  
EINT; // Enable Global interrupt INTM  
ERTM; // Enable Global realtime interrupt DBGM
```

InitPieCtrl() and InitPieVectors() will initialize the PIE interrupt vector table with default functions. These functions can be found in F28M35_DefaultIsr.c.

The next line will replace the interrupt table entry for Timer 0 by the local function “cpu_timer0_isr”.

The next two lines will initialize CPU Timer 0 to a period of 125,000 microseconds = 0.125 seconds.

The last four lines will enable the CPU Timer 0 interrupt.

10. Now replace the “while(1)” loop with:

```
CpuTimer0Regs.TCR.bit.TSS = 0;    // start T0
while(1);
```

The TSS bit cleared to zero will start the timer. The endless while(1) loop is idle.

11. Finally, after the end of function “main()” add a new function:

```
interrupt void cpu_timer0_isr(void)
{
    GpioG1DataRegs.GPCDAT.bit.GPIO070 ^= 1;    // Toggle LED
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;    // ACK PIE - Int
}
```

Build the C28 Control Subsystem Project

12. Click the “Build” button and watch the tools run in the “Console” window. Check for errors in the “Problems” window.

Create a New Project – M3 Master Subsystem

13. The Lab2_M3 project will be used as a starting point for this part of the lab exercise. Right click on project Lab2_M3 and select “Copy” followed by “Paste”. A “Copy Project” window will open.

In the Project name field rename the project to **Lab3_M3**. Uncheck the “Use default location” box. Click the Browse... button and navigate to:

C:\F28M35x\Labs\Lab3\M3

Click OK, and then OK again to copy the project.

14. Close the Lab2_M3 project by right-clicking on the project in the Project Explorer window and select **Close Project**.
15. Next, in the Lab3_M3 project delete the “Debug” folder.
16. Then rename file Lab2_M3.c to **Lab3_M3.c**.

Edit Source File Lab3_M3.c

17. In Lab3_M3.c we need to enable the Inter-Processor Communication (IPC), and then we need to select control for GPIO70 to the C28 Control subsystem. Uncomment the following two lines (78 and 84):

```
IPCMtoCBootControlSystem(CBROM_MTOC_BOOTMODE_BOOT_FROM_FLASH);
GPIOPinConfigureCoreSelect(GPIO_PORTC_BASE, 0x40, GPIO_PIN_C_CORE_SELECT);
```

Build the M3 Master Subsystem Project and Program Flash

18. Click the “Build” button and watch the tools run in the “Console” window. Check for errors in the “Problems” window.
19. Program the flash memory by clicking the “Debug” button (green bug) or click **Run→Debug**. After successfully programming the flash memory the “Progress Information” box will close.

Dual Subsystem Debugging

20. The “Debug” window should reflect the current status of the two subsystems showing the Cortex_M3_0 as connected (suspended), and C28xx_0 as disconnected. Now in the “Debug” window highlight the C28xx_0 (Disconnected : Unknown) line and right click and select “Connect Target”. With the line still highlighted, load the program by clicking:

Run → Load → Load Program...

and Browse to C:\F28M35x\Labs\Lab3\C28\Debug\Lab3_C28.out

Select Open and then OK to program the flash memory.

21. In the “Debug” window left click on the line Texas Instruments XDSv2 USB Emulator_0/C28xx_0 and then click the “Reset CPU” icon.
22. Next in the “Debug” window left click on the line Texas Instruments XDSv2 USB Emulator_0/Cortex_M3_0 and then click the “Reset CPU” icon.

Running the Dual Subsystem

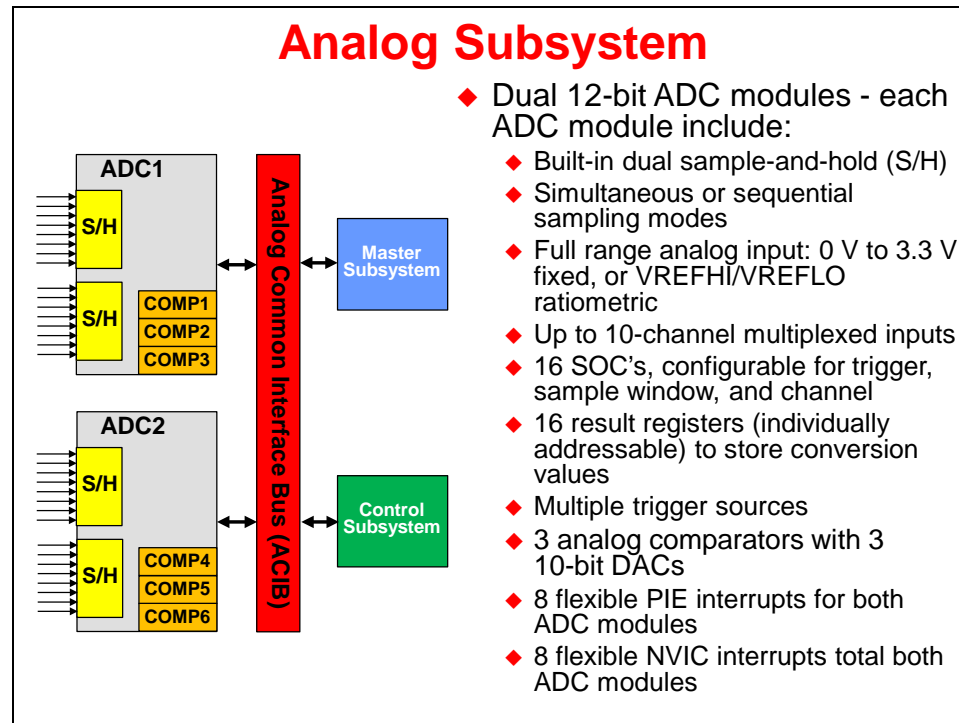
23. In the “Debug” window highlight the C28 subsystem and run the code by clicking the Resume icon (or F8). The C28 runs into an “Idle” instruction in the boot ROM code and waits for an IPC command from the M3 subsystem.
24. Next in the “Debug” window highlight the M3 subsystem and run the code (Resume icon or F8). Both the C28 and M3 subsystems should now be running. The C28 subsystem is controlling the blinking of LED LD2, and the M3 subsystem is controlling the blinking of LED LD3.

Terminate Debug Session

25. Terminate the active debug session using the Terminate button.

End of Exercise

Analog Subsystem



The Analog subsystem consists of dual 12-bit ADC modules and six comparators with six internal 10-bit DACs. Each ADC module block has two sample and hold circuits and three comparators plus three internal 10-bit DACs.

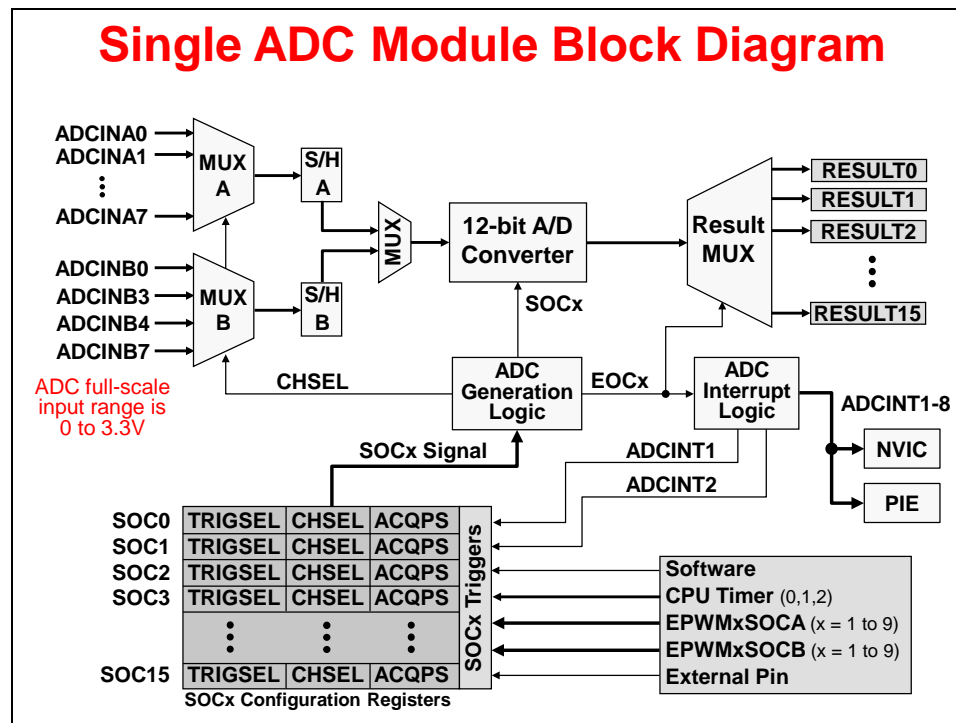
The Analog subsystem is accessed by the analog common interface bus (ACIB). The ACIB is responsible for transferring triggers initiated by the Control subsystem to the Analog subsystem and transferring interrupts initiated by the Analog subsystem to both the Master subsystem and Control subsystem. The ACIB also transfers analog register read and write data and ADC conversion results.

The internal ADC module is a 12-bit recyclic ADC; part SAR, part pipelined. The analog circuits of this converter include:

- The front-end analog multiplexers (MUXs)
- Sample-and-hold (S/H) circuits
- The conversion core
- Voltage regulators and other analog supporting circuits

Digital circuits include programmable conversions, result registers, interface to analog circuits, interface to the Analog Common Interface Bus (ACIB) and interface to other on-chip modules.

Module Block Diagram



Unlike the Delfino ADC, this ADC is not sequencer based. Instead, it is SOC based. The term “SOC” is the configuration set defining the single conversion of a single channel. In that set there are three configurations:

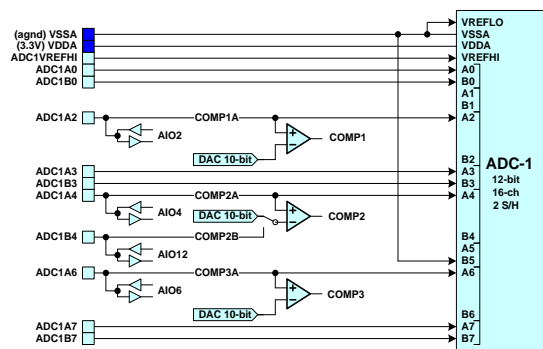
- Trigger source that starts the conversion
- Channel to convert
- Acquisition (sample) window size

Each SOC is independently configured and can have any combination of the trigger, channel, and sample window size available. Multiple SOCx can be configured for the same trigger, channel, and/or acquisition window as desired. This provides a very flexible means of configuring conversions ranging from individual samples of different channels with different triggers, to oversampling the same channel using a single trigger, to creating your own series of conversions of different channels all from a single trigger.

The trigger source for SOCx is configured by a combination of the TRIGSEL field in the ADCSOCxCTL register, the appropriate bits in the ADCINTSOCSEL1 or ADCINTSOCSEL2 register, and setting the correct bits in the TRIGxSEL registers. Software can also force an SOC event with the ADCSOCFRC1 register. The channel and sample window size for SOCx are configured with the CHSEL and ACQPS fields of the ADCSOCxCTL register.

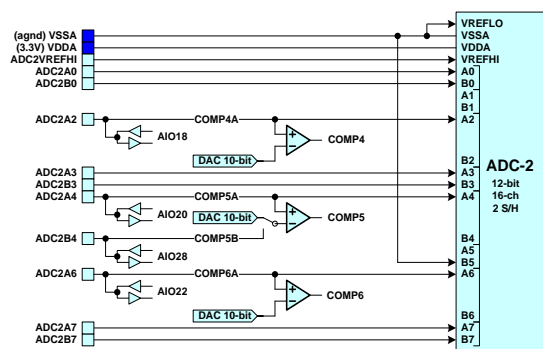
Input Pin Connections

ADC1 Input Pin Connections



- ◆ 6 analog-only input pins
 - ◆ A0, B0, A3, B3, A7, B7
- ◆ 4 shared with AIO & COMP
 - ◆ A2, A4, B4, A6
- ◆ 3 DAC for COMPx input thresholds
- ◆ COMP2 for comparing two analog signals
- ◆ VREFLO on B5
- ◆ AIO configured as inputs and disabled at reset

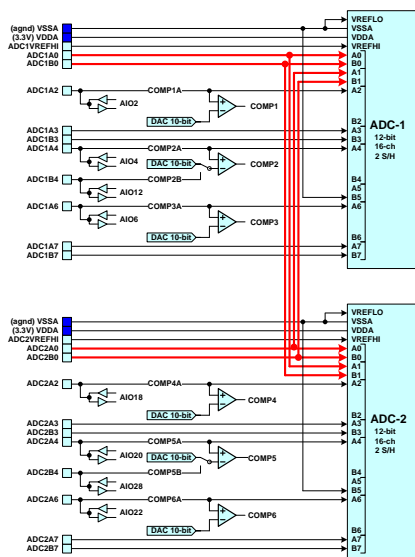
ADC2 Input Pin Connections



- ◆ 6 analog-only input pins
 - ◆ A0, B0, A3, B3, A7, B7
- ◆ 4 shared with AIO & COMP
 - ◆ A2, A4, B4, A6
- ◆ 3 DAC for COMPx input thresholds
- ◆ COMP5 for comparing two analog signals
- ◆ VREFLO on B5
- ◆ AIO configured as inputs and disabled at reset

Same connections as ADC1

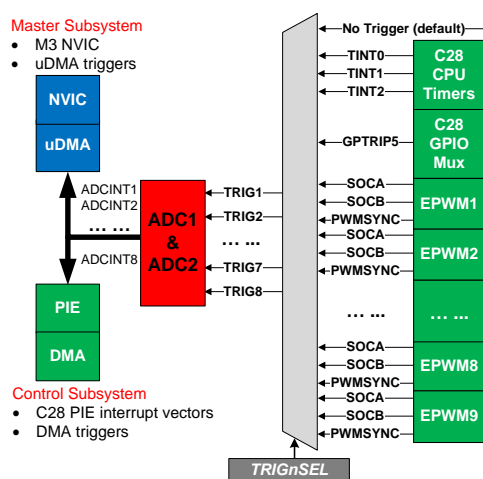
ADC Shared Inputs



- ◆ **4 Shared analog input pins:**
 - ◆ $ADC1A0 = ADC1A0 \& ADC2A1$
 - ◆ $ADC1B0 = ADC1B0 \& ADC2B1$
 - ◆ $ADC2A0 = ADC1A1 \& ADC2A0$
 - ◆ $ADC2B0 = ADC1B1 \& ADC2B0$
- ◆ **Redundant channels does not waste pins**
- ◆ **ADC safety verification**
 - ◆ Critical signals converted by 2 ADCs
 - ◆ 6 independent 10-bit DACs

Conversion Trigger

ADC Conversion Trigger



- ◆ **Multiplexer selects trigger source**
- ◆ **8 triggers shared by ADC1 & ADC2**
- ◆ **ADC configuration registers select one of the 8 triggers to initiate a conversion**
- ◆ **Can sample 4 signals simultaneously**
- ◆ **4 compare registers from ePWM adds more SOC flexibility**
 - ◆ SOCA/B from CMPA/B/C/D
- ◆ **Interrupts share by ADC1 & ADC2**
 - ◆ Goes to both Subsystems

Each SOC can be configured to start on one of many input triggers. Multiple SOC's can be configured for the same channel if desired. Following is a list of the available input triggers:

- Software
- CPU Timers 0/1/2 interrupts
- XINT2 SOC
- ePWM1-9 SOCA and SOCB

Additionally ADCINT1 and ADCINT2 can be fed back to trigger another conversion. This configuration is controlled in the ADCINTSOCSEL1/2 registers. This mode is useful if a continuous stream of conversions is desired

Each SOC can be configured to convert any of the available ADCIN input channels. When an SOC is configured for sequential sampling mode, the four bit CHSEL field of the ADCSOCxCTL register defines which channel to convert.

When an SOC is configured for simultaneous sampling mode, the most significant bit of the CHSEL field is dropped and the lower three bits determine which pair of channels is converted.

ADCINA0 is shared with VREFHI, and therefore cannot be used as a variable input source when using external reference voltage mode.

Conversion Priority

ADC Conversion Priority

- ◆ ***When multiple SOC flags are set at the same time – priority determines the order in which they are converted***

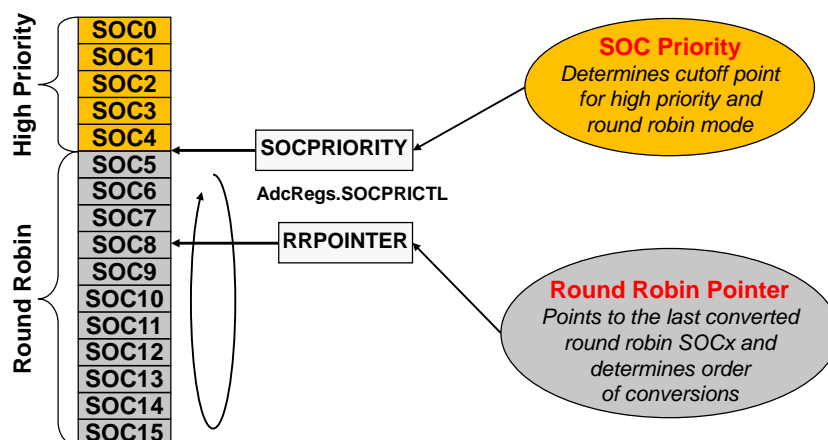
- ◆ **Round Robin Priority (default)**

- ◆ No SOC has an inherent higher priority than another
- ◆ Priority depends on the round robin pointer

- ◆ **High Priority**

- ◆ High priority SOC will interrupt the round robin wheel after current conversion completes and insert itself as the next conversion
- ◆ After its conversion completes, the round robin wheel will continue where it was interrupted

Conversion Priority Functional Diagram



Round Robin Priority Example

SOC PRIORITY configured as 0;
RRPOINTER configured as 15;
SOC0 is highest RR priority

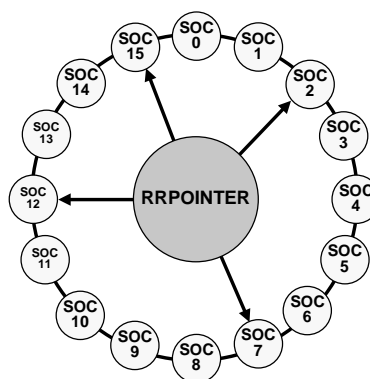
SOC7 trigger received

SOC7 is converted;
RRPOINTER now points to SOC7;
SOC8 is now highest RR priority

SOC2 & SOC12 triggers received simultaneously

SOC12 is converted;
RRPOINTER points to SOC12;
SOC13 is now highest RR priority

SOC2 is converted;
RRPOINTER points to SOC2;
SOC3 is now highest RR priority



High Priority Example

SOC PRIORITY configured as 4;
RRPOINTER configured as 15;
SOC4 is highest RR priority

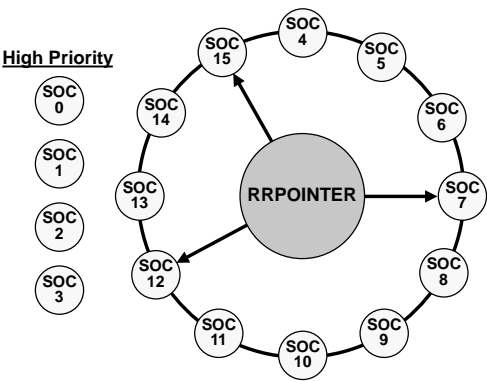
SOC7 trigger received

SOC7 is converted;
RRPOINTER points to SOC7;
SOC8 is now highest RR priority

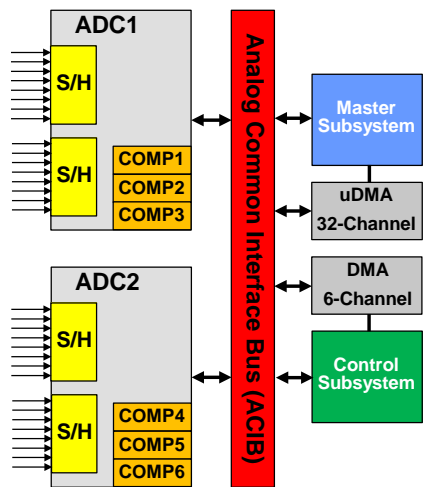
SOC2 & SOC12 triggers received simultaneously

SOC2 is converted;
RRPOINTER stays pointing to SOC7

SOC12 is converted;
RRPOINTER points to SOC12;
SOC13 is now highest RR priority



ADC Accessibility

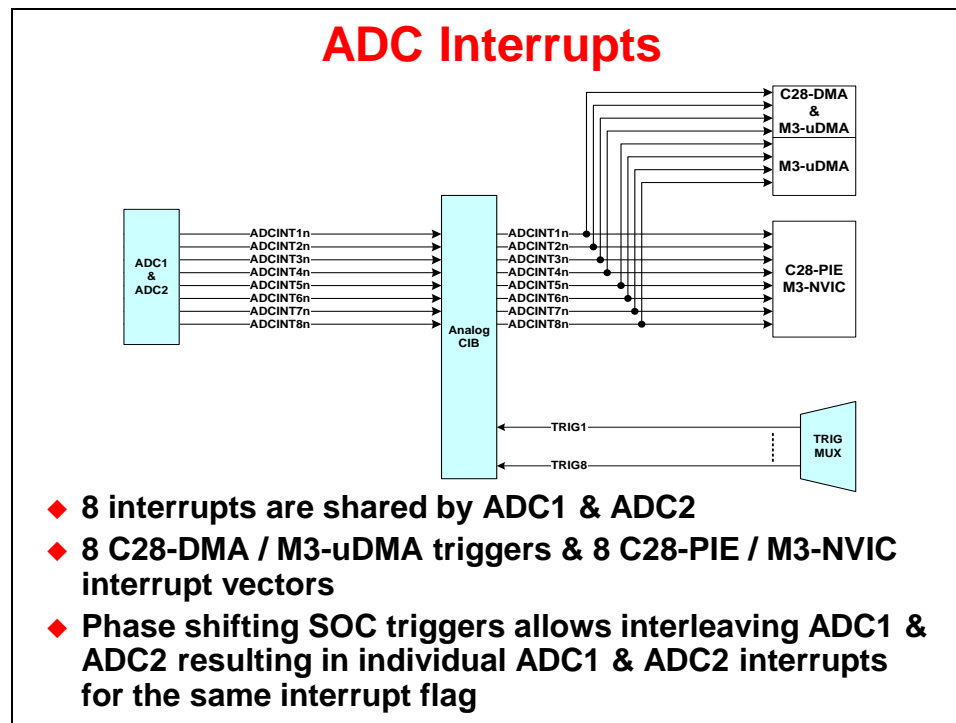


Round-robin arbitration

1. M3
2. M3 uDMA
3. C28 read
4. C28 write
5. C28 DMA read/write

Resource	C28 DMA	C28	M3 uDMA	M3
ADC Results	yes	yes	yes	yes
COMP_DAC	yes	yes	-	-
ADC Config	yes	yes	-	-
GPIO	yes	yes	-	-

Interrupts



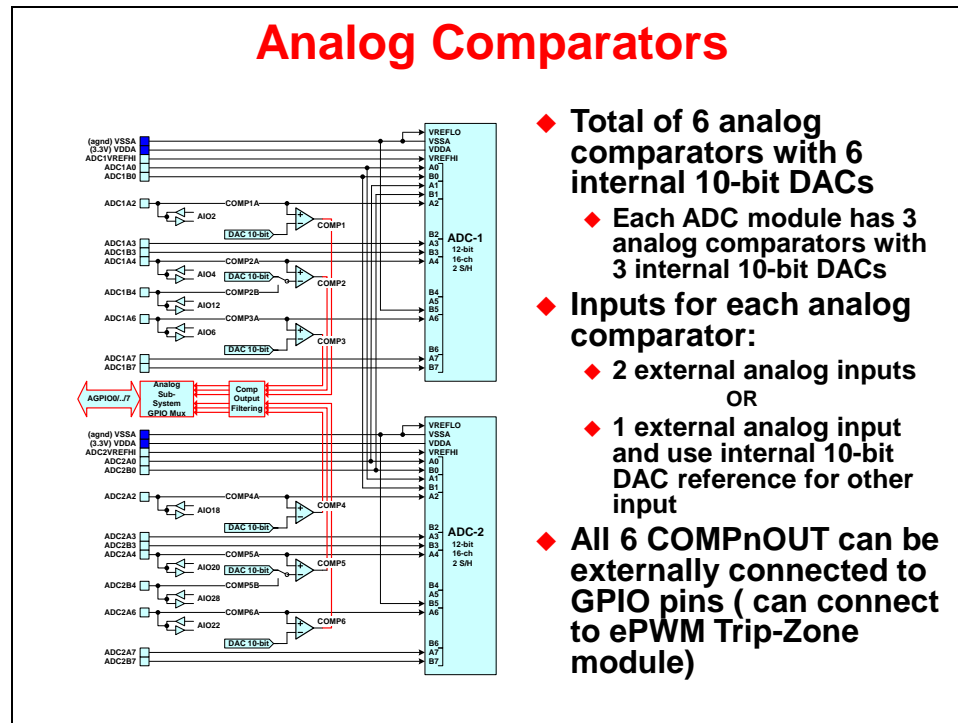
Clock Configuration

ADC Clock Configuration

Device Configuration Examples	C28	M3	ACIBCLK = ADCCLK	ADC1 & ADC2
	MHz	MHz (Div Ratio To C28 MHz)	MHz (Div Ratio To C28 MHz)	MSPS (ADCCLK/13)
Max MIPS Device	150.0	75.0 (/2)	37.5 (/4)	2 x 2.885
Max M3 MIPS Device	100.0	100.0 (/1)	25.0 (/4)	2 x 1.923
Low End MIPS Device	60.0	60.0 (/1)	30.0 (/2)	2 x 2.308

- ◆ ADCCLK is synchronous to C28
- ◆ ACIBCLK = C28CLK/CLKDIV
 - ◆ CLKDIV= 1, 2, 4, or 8 (default at reset)
 - ◆ ACIB = Analog Common Interface Bus
- ◆ ADCCLK = ACIBCLK
 - ◆ ADCCLK = (150MHz/4) = 37.5MHz
 - ◆ At reset, ADCCLK = 10MHz OSC
- ◆ 5.77 MSPS Max conversion rate
 - ◆ Each ADC Sample Rate
 - ◆ 37.5MHz/13 = 2.885MSPS

Analog Comparators



The comparator module is a true analog voltage comparator in the VDDA domain. The analog portion of the block includes:

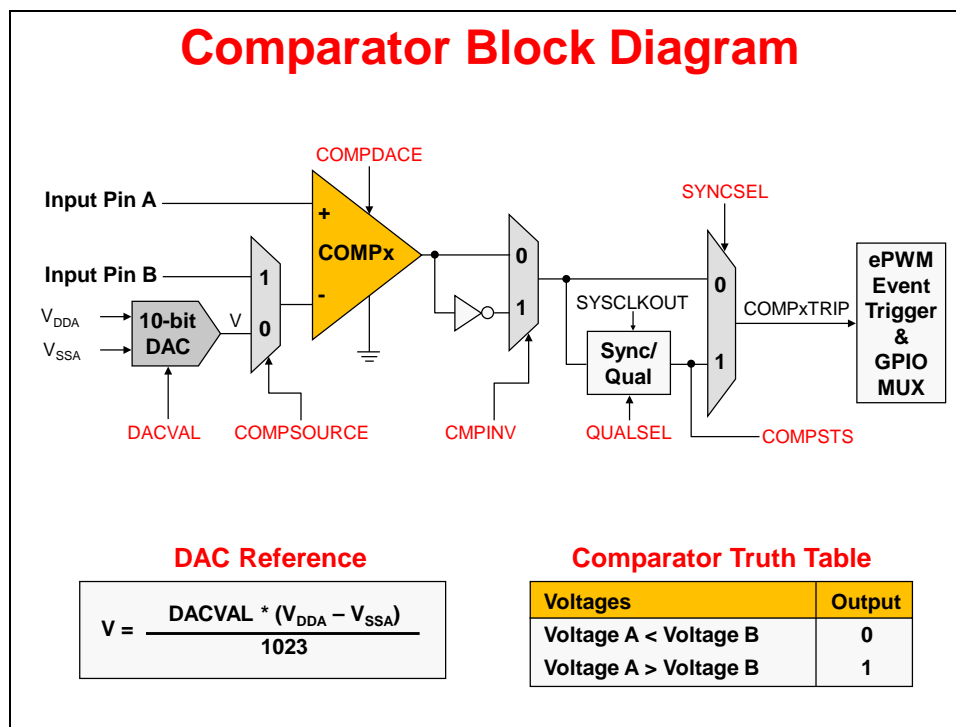
- Comparator
- Inputs and outputs
- Internal DAC reference

The digital circuits include the DAC controls, interface to other on-chip logic, output qualification block, and the control signals.

The comparator block can accommodate two external analog inputs or one external analog input using the internal DAC reference for the other input. The output of the comparator can be passed asynchronously or qualified and synchronized to the system clock period.

The comparator output can be externally connected to a GPIO in order to connect to an ePWM Trip Zone module.

Comparator Block Diagram

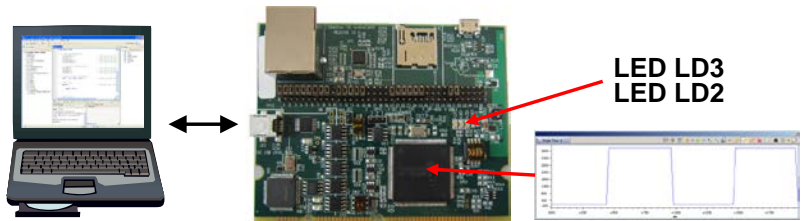


Lab 4: Capture ePWM waveform with ADC

➤ Objective

The objective of this lab exercise is to demonstrate and become familiar with the operation of the on-chip analog-to-digital converter and ePWM module. ePWM1A will be setup to generate a 2 kHz, 50% 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 50 kHz sampling rate and store the conversion result in a buffer in the C28 Control subsystem memory. This buffer operates in a circular fashion, such that new conversion data continuously overwrites older results in the buffer. This lab exercise consists of two parts. The first part will configure the ePWM waveform generation and the second part will configure the ADC capture.

Lab 4: Analog Subsystem



◆ **Capture PWM waveform with ADC**

◆ **C28 project:**

- ◆ Part 1 – C28 project generates 2 kHz waveform
- ◆ Part 2 – C28 project captures waveform with ADC and displayed with CCS
- ◆ Blinks LED LD2 unchanged from Lab3

◆ **M3 project runs unchanged from Lab2**

Part 1 – ePWM Waveform Generation

➤ Procedure

Create a New Project – C28 Control Subsystem

1. The Lab3_C28 project will be used as a starting point for this part of the lab exercise. Right click on project Lab3_C28 and select “Copy” followed by “Paste”. A “Copy Project” window will open.

In the Project name field rename the project to **Lab4-1_C28**. Uncheck the “Use default location” box. Click the Browse... button and navigate to:

C:\F28M35x\Labs\Lab4-1\C28

Click OK, and then OK again to copy the project.

2. Close the Lab3_C28 project by right-clicking on the project in the Project Explorer window and select `Close Project`.
3. Next, in the Lab4-1_C28 project delete the “Debug” folder.
4. Then rename file Lab3_C28.c to **Lab4-1_C28.c**.

Edit Source File Lab4-1_C28.c

5. Edit Lab4-1_C28.c as follows – at the beginning of the function “main()” add a new line after the “EALLOW” instruction to assign ePWM1A to pin GPIO0:

```
GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1;    // GPIO0 = ePWM1A
```

6. Next, after the function call to “InitFlash()” add new instructions to define ePWM1A:

```
EPwm1Regs.TBCTL.all = 0;                // clear all bits in TBCTL
EPwm1Regs.TBCTL.bit.CLKDIV = 0;          // CLKDIV = “div by 1”;
EPwm1Regs.TBCTL.bit.HSPCLKDIV = 0;       // HSPCLKDIV = “div by 1”
EPwm1Regs.TBCTL.bit.CTRMODE = 2;         // count mode = up-down mode
EPwm1Regs.AQCTLA.all = 0x0006;           // ZRO=set; PRD=clear
EPwm1Regs.TBPRD = 37500;                  // 2 kHz PWM frequency
// TBPRD = fcpu / (2* fpwm * CLKDIV * HSPCLKDIV)
// TBPRD = 150 MHz / (2 * 2 kHz * 1 * 1)
```

Build the C28 Control Subsystem Project

7. Click the “Build” button and watch the tools run in the “Console” window. Check for errors in the “Problems” window.

Create a New Project – M3 Master Subsystem

8. The Lab3_M3 project will be used as a starting point for this part of the lab exercise. Right click on project Lab3_M3 and select “Copy” followed by “Paste”. A “Copy Project” window will open.

In the Project name field rename the project to **Lab4-1_M3**. Uncheck the “Use default location” box. Click the Browse... button and navigate to:

```
C:\F28M35x\Labs\Lab4-1\M3
```

Click OK, and then OK again to copy the project.

9. Close the Lab3_M3 project by right-clicking on the project in the Project Explorer window and select `Close Project`.
10. Next, in the Lab4-1_M3 project delete the “Debug” folder.
11. Then rename file Lab3_M3.c to **Lab4-1_M3.c**.

Edit Source File Lab4-1_M3.c

12. Edit Lab4-1_M3.c – in the function “main()” we need to select control for pin GPIO0 to the C28 Control subsystem. The code in “main()” already selects control for GPIO70 (LED LD3) to the C28 Control subsystem. Now, add the following two lines to select control of GPIO0 after the existing instructions for GPIO70:

```
// Enable Clock Supply for GPIOA
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
// Give C28 control of Port A pin 0
GPIOPinConfigureCoreSelect(GPIO_PORTA_BASE, GPIO_PIN_0, GPIO_PIN_C_CORE_SELECT);
```

Build the M3 Master Subsystem Project and Program Flash

13. Click the “Build” button and watch the tools run in the “Console” window. Check for errors in the “Problems” window.
14. Program the flash memory by clicking the “Debug” button (green bug) or click Run→Debug. After successfully programming the flash memory the “Progress Information” box will close.

Dual Subsystem Debugging

15. The “Debug” window should reflect the current status of the two subsystems showing the Cortex_M3_0 as connected (suspended), and C28xx_0 as disconnected. Now in the “Debug” window highlight the C28xx_0 (Disconnected : Unknown) line and right click and select “Connect Target”. With the line still highlighted, load the program by clicking:

Run → Load → Load Program...

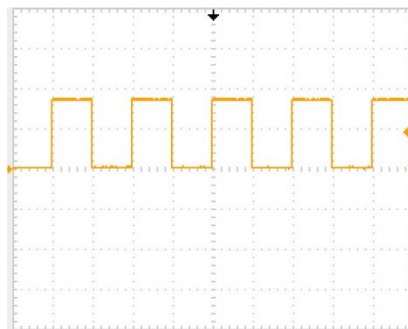
and Browse to C:\F28M35x\Labs\Lab4-1\C28\Debug\Lab4-1_C28.out

Select Open and then OK to program the flash memory.

16. In the “Debug” window left click on the line Texas Instruments XDSv2 USB Emulator_0/C28xx_0 and then click the “Reset CPU” icon.
17. Next in the “Debug” window left click on the line Texas Instruments XDSv2 USB Emulator_0/Cortex_M3_0 and then click the “Reset CPU” icon.

Running the Dual Subsystem

18. In the “Debug” window highlight the C28 subsystem and run the code by clicking the Resume icon (or F8). The C28 runs into an “Idle” instruction in the boot ROM code and waits for an IPC command from the M3 subsystem.
19. Next in the “Debug” window highlight the M3 subsystem and run the code (Resume icon or F8). Both the C28 and M3 subsystems should now be running and both LEDs LD2 and LD3 should be blinking.
20. If an oscilloscope is available, connect it to pin GPIO0. The 100-pin connector on the docking station is marked as J1 and the pins are numbered according to their GPIO numbers. GPIO0 is labeled as the “00” pin:



Note: The waveform will be viewed using CCS in the next part without an oscilloscope.

Terminate Debug Session

21. Terminate the active debug session using the Terminate button.

End of Part 1

Part 2 – ADC Capture

In this part the ADC will be configured to capture the ePWM1A waveform that was generated in the previous part. Samples will be stored in a circular buffer and displayed using the CCS graphing feature.

➤ Procedure

Create a New Project – C28 Control Subsystem

1. The Lab4-1_C28 project will be used as a starting point for this part of the lab exercise. Right click on project Lab4-1_C28 and select “Copy” followed by “Paste”. A “Copy Project” window will open.

In the Project name field rename the project to **Lab4-2_C28**. Uncheck the “Use default location” box. Click the Browse... button and navigate to:

C:\F28M35x\Labs\Lab4-2\C28

Click OK, and then OK again to copy the project.

2. Close the Lab4-1_C28 project by right-clicking on the project in the Project Explorer window and select `Close Project`.
3. Next, in the Lab4-2_C28 project delete the “Debug” folder.
4. Then rename file Lab4-1_C28.c to **Lab4-2_C28.c**.

Add Files to Project – C28 Control Subsystem

5. Add (“copy files”) the following source files to the project by right-clicking on Lab4-2_C28 in the “Project Explorer” window and selecting `Add Files...`

File: F28M35x_Adc.c

File: F28M35x_usDelay.asm

Folder: C:\F28M35x\Device_support\F28M35x_common\source

Edit Source File Lab4-2_C28.c

6. At the beginning of the file Lab4-2_C28.c, replace the function declaration “cpu_timer0_isr()” by a new declaration for the ADC interrupt service routine:

```
interrupt void adc1_isr(void);
```

7. Then add three new global variables “ConversionCount”, “Voltage_C28[100]”, and “GPIO70_count”. The first two variables will be used to buffer the ADC samples in a circular memory and the last variable will be used for the blink rate of LED LD2:

```
Uint16 ConversionCount = 0;
Uint16 Voltage_C28[100] = {0};
Uint16 GPIO70_count;
```

8. In function “main()”, change the code line for the adjustment of the interrupt table entry by removing the line for Timer0 and add a new line for the ADC:

```
PieVectTable.ADCINT1 = &adc1_isr;
```

9. Change the enable instruction for the PIEIER1 register to enable the ADC (1.1) instead of the Timer 0 (1.7):

```
PieCtrlRegs.PIEIER1.bit.INTx1 = 1;
```

10. Change the period for CPU Timer0 from 125 milliseconds into 20 microseconds (for an ADC sampling frequency of 50 kHz) by a modification to the following line:

```
ConfigCpuTimer(&CpuTimer0,C28_FREQ,20);
```

11. Before the code in “main()” enters the “while(1)” loop, add code to initialize the ADC1:

```
InitAdc1();
EALLOW;
Adc1Regs.ADCCTL2.bit.ADCNONOVERLAP = 1; // Enable non-overlap mode i.e.
// conversion and future sampling
// events don't overlap
Adc1Regs.ADCCTL1.bit.INTPULSEPOS = 1; // ADCINT1 trips after AdcResults latch
Adc1Regs.INTSEL1N2.bit.INT1E = 1; // Enabled ADCINT1
Adc1Regs.INTSEL1N2.bit.INT1CONT = 0; // Disable ADCINT1 Continuous mode
Adc1Regs.INTSEL1N2.bit.INT1SEL = 0; // setup EOC0 to trigger ADCINT1 to fire
Adc1Regs.ADCSOC0CTL.bit.CHSEL = 0; // set SOC0 channel select to ADC1in_A0
AnalogSysCtrlRegs.TRIG1SEL.all = 1; // Assigning TINT0 to ADC-TRIGGER 1
Adc1Regs.ADCSOC0CTL.bit.TRIGSEL = 5; // Set SOC0 start trigger to
// ADC Trigger 1 of the ADC
Adc1Regs.ADCSOC0CTL.bit.ACQPS = 6; // set SOC0 S/H Window to 7 ADC
// Clock Cycles, (6 ACQPS + 1)
EDIS;
```

12. At the end of the file replace function “cpu_timer0_isr” by a new function:

```
interrupt void adc1_isr(void)
{
    Voltage_C28[ConversionCount] = Adc1Result.ADCRESULT0;
    if(ConversionCount == 100) ConversionCount = 0;
    else ConversionCount++;
    if(GPIO70_count++ > 6250)
    {
        GpioG1DataRegs.GPCDAT.bit.GPIO70 ^= 1;
        GPIO70_count = 0;
    }
    Adc1Regs.ADCINTFLGCLR.bit.ADCINT1 = 1;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

Build the C28 Control Subsystem Project

13. Click the “Build” button and watch the tools run in the “Console” window. Check for errors in the “Problems” window.

Create a New Project – M3 Master Subsystem

14. The Lab4-1_M3 project will be used as a starting point for this part of the lab exercise. Right click on project Lab4-1_M3 and select “Copy” followed by “Paste”. A “Copy Project” window will open.

In the Project name field rename the project to **Lab4-2_M3**. Uncheck the “Use default location” box. Click the Browse... button and navigate to:

```
C:\F28M35x\Labs\Lab4-2\M3
```

Click OK, and then OK again to copy the project.

15. Close the Lab4-1_M3 project by right-clicking on the project in the Project Explorer window and select `Close Project`.
16. Next, in the Lab4-2_M3 project delete the “Debug” folder.
17. Then rename file Lab4-1_M3.c to **Lab4-2_M3.c**.

Build the M3 Master Subsystem Project and Program Flash

18. Click the “Build” button and watch the tools run in the “Console” window. Check for errors in the “Problems” window.
19. Program the flash memory by clicking the “Debug” button (green bug) or click `Run`→`Debug`. After successfully programming the flash memory the “Progress Information” box will close.

Dual Subsystem Debugging

20. The “Debug” window should reflect the current status of the two subsystems showing the Cortex_M3_0 as connected (suspended), and C28xx_0 as disconnected. Now in the “Debug” window highlight the C28xx_0 (Disconnected : Unknown) line and right click and select “Connect Target”. With the line still highlighted, load the program by clicking:

`Run` → `Load` → `Load Program...`

and Browse to `C:\F28M35x\Labs\Lab4-2\C28\Debug\Lab4-2_C28.out`

Select `Open` and then `OK` to program the flash memory.

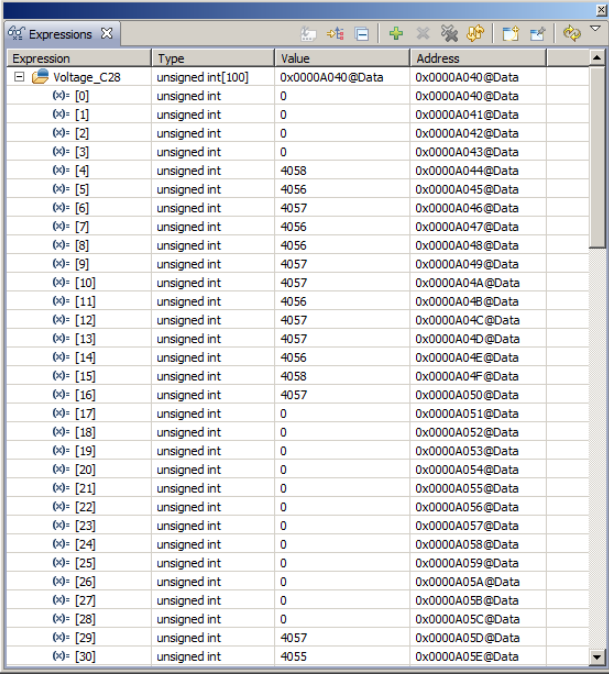
21. In the “Debug” window left click on the line Texas Instruments XDSv2 USB Emulator_0/C28xx_0 and then click the “Reset CPU” icon.
22. Next in the “Debug” window left click on the line Texas Instruments XDSv2 USB Emulator_0/Cortex_M3_0 and then click the “Reset CPU” icon.

Running the Dual Subsystem

23. In the “Debug” window highlight the C28 subsystem and run the code by clicking the Resume icon (or F8). The C28 runs into an “Idle” instruction in the boot ROM code and waits for an IPC command from the M3 subsystem.
24. Next in the “Debug” window highlight the M3 subsystem and run the code (Resume icon or F8). Both the C28 and M3 subsystems should now be running and both LEDs LD2 and LD3 should be blinking.

Inspect ADC Samples

25. Using a connector wire provided, connect the PWM1A (pin # GPIO-00) to ADCINA0 (pin # ADC-A0) on the Docking Station.
26. After a few seconds halt the C28 Control subsystem using the “Suspend” button on the toolbar (or alt+F8).
27. Add the variable “Voltage_C28” to the Expressions window by clicking the “Expressions” tab at the top of the window. In the empty box in the Expression column (*Add new expression*), type `Voltage_C28` and then enter. (Note that the expressions window can be manually opened by clicking `View` → `Expressions` on the menu bar).



Expression	Type	Value	Address
Voltage_C28	unsigned int[100]	0x0000A040@Data	0x0000A040@Data
[0]	unsigned int	0	0x0000A040@Data
[1]	unsigned int	0	0x0000A041@Data
[2]	unsigned int	0	0x0000A042@Data
[3]	unsigned int	0	0x0000A043@Data
[4]	unsigned int	4058	0x0000A044@Data
[5]	unsigned int	4056	0x0000A045@Data
[6]	unsigned int	4057	0x0000A046@Data
[7]	unsigned int	4056	0x0000A047@Data
[8]	unsigned int	4056	0x0000A048@Data
[9]	unsigned int	4057	0x0000A049@Data
[10]	unsigned int	4057	0x0000A04A@Data
[11]	unsigned int	4056	0x0000A04B@Data
[12]	unsigned int	4057	0x0000A04C@Data
[13]	unsigned int	4057	0x0000A04D@Data
[14]	unsigned int	4056	0x0000A04E@Data
[15]	unsigned int	4058	0x0000A04F@Data
[16]	unsigned int	4057	0x0000A050@Data
[17]	unsigned int	0	0x0000A051@Data
[18]	unsigned int	0	0x0000A052@Data
[19]	unsigned int	0	0x0000A053@Data
[20]	unsigned int	0	0x0000A054@Data
[21]	unsigned int	0	0x0000A055@Data
[22]	unsigned int	0	0x0000A056@Data
[23]	unsigned int	0	0x0000A057@Data
[24]	unsigned int	0	0x0000A058@Data
[25]	unsigned int	0	0x0000A059@Data
[26]	unsigned int	0	0x0000A05A@Data
[27]	unsigned int	0	0x0000A05B@Data
[28]	unsigned int	0	0x0000A05C@Data
[29]	unsigned int	4057	0x0000A05D@Data
[30]	unsigned int	4055	0x0000A05E@Data

The ePWM generated waveform is sampled by the ADC. The input range for the ADC is between 0V and 3.3V. For the PWM waveform, this would give 12-bit ADC samples close to 0 or almost 4095.

Graph PWM Waveform

28. Open and setup a graph to plot a 100-point window of the ADC results buffer.

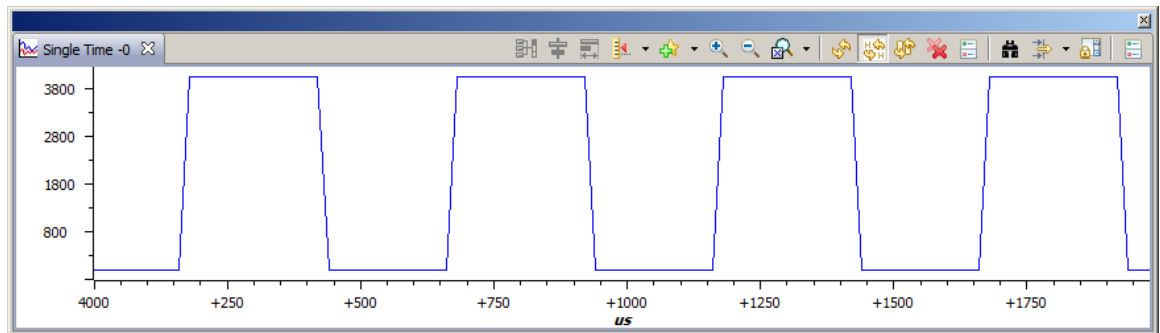
Click: Tools → Graph → Single Time and set the following values:

Acquisition Buffer Size	100
DSP Data Type	16-bit unsigned integer
Sampling Rate (Hz)	50000
Start Address	Voltage_C28
Display Data Size	100
Time Display Unit	μs

Select OK to save the graph options.

The graphical display should show the generated 1 kHz, 50% duty cycle symmetric PWM waveform. The period of a 1 kHz signal is 1 ms. You can confirm this by measuring the period of the waveform using the “measurement marker mode” graph feature. In the graph window toolbar, left-click on the ruler icon with the red arrow. Note when you hover your mouse over the icon, it will show “Toggle Measurement Marker Mode”. Move the mouse to the first measurement position and left-click. Again, left-click on the Toggle Measurement Marker Mode icon. Move the mouse to the second measurement position and left-click. The graph will automatically calculate the difference between the two values taken over a complete waveform period. When done, clear the measurement points by right-clicking on the graph and select Remove All Measurement Marks (or Ctrl+Alt+M).

Graph view of variable "Voltage_C28[100]":



Terminate Debug Session

29. Terminate the active debug session using the Terminate button.

End of Exercise

Inter-Processor Communications (IPC)

IPC Device Features

- ◆ Shared SARAM
- ◆ Message SARAM
- ◆ IPC Message Registers
- ◆ IPC Interrupts and Flags
- ◆ Clock Configuration Semaphore
- ◆ Flash Pump Semaphore
- ◆ Free Running Counter
- ◆ SSI to SPI and UART to I2C Loopback

Shared SARAM and Message SARAM

IPC Shared SARAM

- ◆ Up to 8 Blocks (S0 - S7)
- ◆ 8Kbytes each
 - ◆ Can be configured to be used by M3 or C28
 - ◆ Typically used by the application
 - ◆ Also can be used to pass messages

Ownership	Master Subsystem		Control Subsystem	
	M3 CPU	uDMA	C28	DMA
Master Subsystem*	R/W/Exe	R/W	R	R
Control Subsystem	R	R	R/W/Exe	R/W

* default

There are up to 8 blocks of shared SARAM on F28M35x devices. These shared SARAM blocks are typically used by the application, but can also be used for transferring messages and data.

Each block can individually be owned by either the master subsystem or the control subsystem.

Master Subsystem ownership:

At reset, the master subsystem owns all of the shared SARAM blocks. In this configuration the master subsystem can freely use the memory blocks. The M3 can read, write or execute from the block and the μ DMA can read or write.

On the control subsystem, the C28x and C28x DMA can only read from these blocks. Blocks owned by the master subsystem can be used by the M3 to send the C28x messages. This is referred to as “MtoC”.

Control Subsystem ownership:

After reset, the M3 application can assign ownership of blocks to the control subsystem. In this configuration, control subsystem can freely use the blocks. The C28x can read, write or execute from the block. The master subsystem, however can only read from the block. Blocks owned by the control subsystem can be used to send messages from the C28x to the M3. This is referred to as “CtoM”.

IPC Message SARAM

◆ 2 Blocks

◆ 2Kbytes each Block

◆ Used to pass messages or data between the M3 and C28 CPU's

◆ Always enabled – configuration is fixed

Message RAM	Master Subsystem		Control Subsystem	
	M3 CPU	μ DMA	C28	DMA
Master to Control (“MtoC”)	R/W	R	R	R
Control to Master (“CtoM”)	R	R	R/W	R

The F28M35x has two dedicated message RAM blocks. Each block is 2 Kbytes in length. Unlike the shared SARAM blocks, these blocks provide communication in one direction only and cannot be reconfigured.

Master to Control “MtoC” message RAM:

The first message SARAM is the Master to Control or MtoC. This block can be read or written to by the M3 and read by the C28. The M3 can write a message to this block and then the C28 can read it.

Control to Master “CtoM” message RAM:

The second message SARAM is the Control to Master or CtoM. This block can be read or written to by the C28 and read by the M3. This means the C28 can write a message to this block and then the M3 can read it. After the sending CPU writes a message it can inform the receiver CPU that it is available through an interrupt or flag.

IPC Message Registers

- ◆ Provides very simple and flexible messaging
- ◆ Dedicated registers mapped to both CPU's
 - ◆ IPCCOMMAND
 - ◆ IPCADDR
 - ◆ IPCDATAWRITE
 - ◆ IPCDATAREAD
- ◆ The definition (what the register content means) is up to the application software
- ◆ TI's IPC-Lite drivers use the IPC message registers

Interrupts and Flags

IPC Interrupts and Flags

- ◆ Master to Control: 4 interrupts & 28 flags w/o interrupt
- ◆ Control to Master: 4 interrupts & 28 flags w/o interrupt

Requesting CPU → Set, Flag and Clear registers

Register	
IPCSET	Message waiting (send interrupt and/or set flag)
IPCFLG	Bit is set by the "SET" register
IPCCLR	Clear the flag

Receiving CPU → Status and Acknowledge registers

Register	
IPCSTS	Status (reflects the FLG bit)
IPCAK	Clear STS and FLG

Prefix indicates request and receive CPU: **MT**OC**IPCSET** / **CT**OM**IPCSET**

When the sending CPU wishes to inform the receiver that a message is ready, it can make use of an interrupt or flag. There are identical IPC interrupt and flag resources reside on both the master sub-system and the control sub-system.

4 Interrupts:

There are 4 interrupts that the Master M3 can send to the Control C28 through the control system's peripheral interrupt expansion (PIE) module. Likewise there are 4 interrupts the C28 can send to the M3 Master through the NIVC (Nested Vectored Interrupt Controller). Each of the interrupts has a dedicated vector within the PIE or the NVIC.

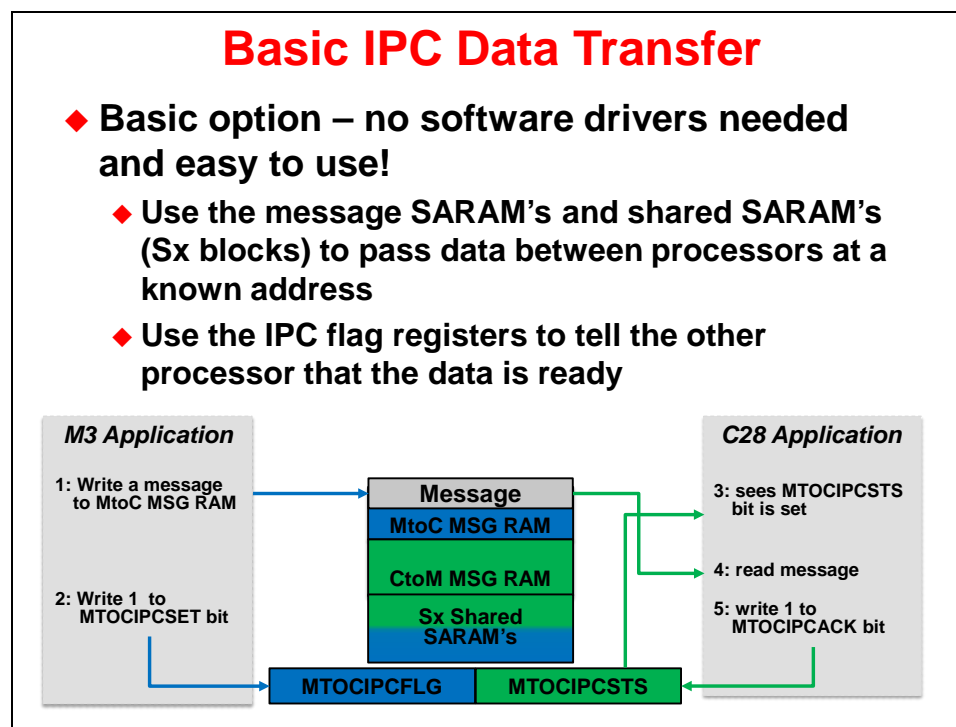
28 Flags:

In addition, there are 28 flags available to each of the subsystems. These flags can be used for messages that are not time critical or they can be used to send status back to originating processor. The flags and interrupts can be used however the application sees fit and are not tied to particular operation in hardware.

Registers: Set, Flag, Clear, Status and Acknowledge:

The registers to control the IPC interrupts and flags are 32-bits: Bits [3:0] = interrupt & flag, Bits [31:4] = flag only

IPC Data Transfer



The F28M35x IPC is very easy to use. Before looking into the details of the IPC software drivers, at the most basic level, the application doesn't need ANY separate software drivers to communicate between processors. It can utilize the message RAM's and shared SARAM blocks to pass data between processors at a fixed address known to both processors. Then the sending processor can use the IPC flag registers merely to flag to the receiving processor that the data is ready. Once the receiving processor has grabbed the data, it will then acknowledge the corresponding IPC flag to indicate that it is ready for more messages.

As an example:

1. First, the M3 would write a message to the C28 in MtoC MSG RAM.
2. Then the M3 would write a 1 to the appropriate flag bit in the MTOCIPCSET register. This sets the MTOCIPCFLG, which also sets the MTOCIPCSTS register on the C28, letting the C28 know that a message is available.

3. The C28 sees that a bit in the MTOCIPCSTS register is set.
4. The C28 reads the message from the MtoC MSG RAM and then
5. It writes a 1 to the same bit in the MTOCIPACK register to acknowledge that it has received the message. This subsequently clears the flag bit in MTCOIPCFLG and MTOCIPCSTS.
6. The M3 can then send more messages using that particular flag bit.

IPC Software Solutions Summary

◆ Basic Option

- ◆ No software drivers needed
- ◆ Uses IPC registers only (simple message passing)

◆ IPC-Lite Software API Driver

- ◆ Uses IPC registers only (no memory used)
- ◆ Limited to 1 IPC interrupt at a time
- ◆ Limited to 1 command/message at a time
- ◆ M3 can use IPC-Lite to communicate with C28 boot ROM

◆ Main IPC Software API Driver

- ◆ Uses circular buffers message RAMs
- ◆ Can queue up to 4 messages prior to processing (configurable)
- ◆ Can use multiple IPC ISRs at a time
- ◆ Requires additional setup in application code prior to use

There are three options to use the IPC on the device.

Basic option: A very simple option that does not require any drivers. This option only requires IPC registers to implement very simple flagging of messages passed between processors.

Driver options: If the application code needs a set of basic IPC driver functions for reading or writing data, setting/clearing bits, and function calls, then there are 2 IPC software driver solutions provided by TI.

IPC-Lite:

- Only uses the IPC registers. No additional memory such as message RAM or shared RAM is needed.
- Only one IPC ISR can be used at a time.
- Can only process one message at a time.
- M3 can use IPC lite to communicate with the C28 boot ROM. The C28 boot ROM processes basic IPC read, write, bit manipulation, function call, and branch commands.

Main IPC Software API Driver: (This is a more feature filled IPC solution)

- Utilizes circular buffers in CtoM and MtoC message RAM's.
- Allows application to queue up to 4 messages prior to processing (configurable).
- Allows application to use multiple IPC ISR's at a time.
- Requires additional setup in application code prior to use.

In addition to the above, SYS/BIOS 6 will provide a new transport module to work with the shared memory and IPC resources on the F28M35x.

Lab 5: Data Transfer using IPC

➤ Objective

The objective of this lab exercise is to demonstrate and become familiar with the operation of the Inter Processor Communication. As in the previous lab exercise, the C28 Control Subsystem ADC has been setup to sample a single input channel at a 50 kHz sampling rate. Except this time the ADC result will be written to the C-to-M Message RAM. The M3 Master subsystem will then read the C-to-M Message RAM and store the results in a circular buffer. The samples will be viewed and displayed using the graphing feature of Code Composer Studio.

Lab 5: Inter-Processor Communications

- ◆ **C28 subsystem:**
 - ◆ Captures PWM waveform with ADC
 - ◆ ADC results written to C-to-M Message RAM
- ◆ **M3 subsystem:**
 - ◆ Reads C-to-M Message RAM
 - ◆ Results stored in circular buffer
- ◆ Results displayed using CCS
- ◆ LEDs LD2 and LD3 blinks unchanged from Lab4

➤ Procedure

Create a New Project – C28 Control Subsystem

1. The Lab4-2_C28 project will be used as a starting point for this lab exercise. Right click on project Lab4-2_C28 and select “Copy” followed by “Paste”. A “Copy Project” window will open.

In the Project name field rename the project to **Lab5_C28**. Uncheck the “Use default location” box. Click the Browse... button and navigate to:

C:\F28M35x\Labs\Lab5\C28

Click OK, and then OK again to copy the project.

2. Close the Lab4-2_C28 project by right-clicking on the project in the Project Explorer window and select Close Project.
3. Next, in the Lab5_C28 project delete the “Debug” folder.
4. Then rename file Lab4-2_C28.c to **Lab5_C28.c**.

Edit Source File Lab5_C28.c

5. In the global variable area at the beginning of the file Lab5_C28.c add a new “pragma” and the definition of a new variable “LatestADCResult”:

```
#pragma DATA_SECTION(LatestADCResult, "CtoM")
Uint16 LatestADCResult;
```

The #pragma directive will place variable “LatestADCResult” in a new data section “CtoM”. Later we will use a dedicated linker command file to connect “CtoM” to the physical address of the mailbox memory.

6. At the end of file Lab5_C28.c edit the interrupt service routine function “adc1_isr”. After the first code line add the following two lines:

```
LatestADCResult = Voltage_C28[ConversionCount]; // load value into CtoM
CtoMIPCRegs.CTOMIPCSET.bit.IPC1 = 1;           // signal IPC1 to M3
```

These two lines will load the latest ADC sample into a single Uint16 variable of the “CtoM” Message buffer and set the IPC1 flag, which can be sampled by the M3 Master subsystem.

Create a New Linker Command File – Lab5_C28.cmd

7. In the Lab5_C28 project create a new linker command file by clicking:

File → New → Source File

and name the file **Lab5_C28.cmd** then click Finish.

8. In the new file enter the following:

```
SECTIONS
{
    CtoM: > CTOMRAM, PAGE = 1
}
```

Save and close the file.

Build the C28 Control Subsystem Project

9. Click the “Build” button and watch the tools run in the “Console” window. Check for errors in the “Problems” window.

Create a New Project – M3 Master Subsystem

10. The Lab4-2_M3 project will be used as a starting point for this lab exercise. Right click on project Lab4-2_M3 and select “Copy” followed by “Paste”. A “Copy Project” window will open.

In the Project name field rename the project to **Lab5_M3**. Uncheck the “Use default location” box. Click the Browse... button and navigate to:

C:\F28M35x\Labs\Lab5\M3

Click OK, and then OK again to copy the project.

11. Close the Lab4-2_M3 project by right-clicking on the project in the Project Explorer window and select `Close Project`.
12. Next, in the Lab5_M3 project delete the “Debug” folder.
13. Then rename file Lab4-2_M3.c to **Lab5_M3.c**.

Edit Source File Lab5_M3.c

14. At the beginning of the Lab5_M3.c file add a new `#include` directive:

```
#include "inc/hw_ipc.h"           // new in Lab5_M3
```

15. In the global variable area at the beginning of the file add three new variables for the M3 Master subsystem:

```
unsigned int Voltage_M3[100];  
unsigned int i = 0;  
#pragma DATA_SECTION(AdcValue, "CtoM")  
unsigned int AdcValue;
```

16. At the end of function “main()”, replace the “while(1);” instruction with:

```
while(1)  
{  
    while(IPCCtoMFlagBusy(IPC_CTOMIPSTS_IPC1)==0); // wait for IPC1  
    Voltage_M3[i]= AdcValue; // read and store result from CtoM - Ram  
    IPCCtoMFlagAcknowledge (IPC_CTOMIPACK_IPC1); // clear IPC1-flag  
    if(i++ >= 100)i=0;  
}
```

This code is a simple poll of the IPC1 flag. If set, the value from the C28 Control subsystem is read and stored in a circular buffer “Voltage_M3[i]”.

Create a New Linker Command File – Lab5_M3.cmd

17. In the Lab5_M3 project create a new linker command file by clicking:

File → New → Source File

and name the file **Lab5_M3.cmd** then click `Finish`.

18. In the new file enter the following:

```
SECTIONS  
{  
    CtoM: > CTOMRAM, TYPE = DSECT  
}
```

Save and close the file.

Build the M3 Master Subsystem Project and Program Flash

19. Click the “Build” button and watch the tools run in the “Console” window. Check for errors in the “Problems” window.

20. Program the flash memory by clicking the “Debug” button (green bug) or click Run→Debug. After successfully programming the flash memory the “Progress Information” box will close.

Dual Subsystem Debugging

21. The “Debug” window should reflect the current status of the two subsystems showing the Cortex_M3_0 as connected (suspended), and C28xx_0 as disconnected. Now in the “Debug” window highlight the C28xx_0 (Disconnected : Unknown) line and right click and select “Connect Target”. With the line still highlighted, load the program by clicking:

Run → Load → Load Program...

and Browse to C:\F28M35x\Labs\Lab5\C28\Debug\Lab5_C28.out

Select Open and then OK to program the flash memory.

22. In the “Debug” window left click on the line Texas Instruments XDSv2 USB Emulator_0/C28xx_0 and then click the “Reset CPU” icon.
23. Next in the “Debug” window left click on the line Texas Instruments XDSv2 USB Emulator_0/Cortex_M3_0 and then click the “Reset CPU” icon.

Running the Dual Subsystem

24. In the “Debug” window highlight the C28 subsystem and run the code by clicking the Resume icon (or F8). The C28 runs into an “Idle” instruction in the boot ROM code and waits for an IPC command from the M3 subsystem.
25. Next in the “Debug” window highlight the M3 subsystem and run the code (Resume icon or F8). Both the C28 and M3 subsystems should now be running.

Inspect ADC Samples

26. Using a connector wire provided, connect the PWM1A (pin # GPIO-00) to ADCINA0 (pin # ADC-A0) on the Docking Station.
27. After a few seconds halt the C28 Control subsystem using the “Suspend” button on the toolbar (or alt+F8).

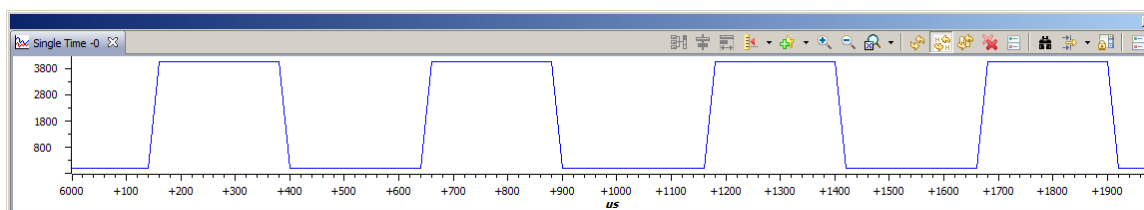
Graph PWM Waveform

28. Open and setup a graph to plot a 100-point window of the ADC results buffer. Click: Tools → Graph → Single Time and set the following values:

Acquisition Buffer Size	100
DSP Data Type	16-bit unsigned integer
Sampling Rate (Hz)	50000
Start Address	Voltage_C28
Display Data Size	100
Time Display Unit	μs

Select OK to save the graph options. The graphical display should show the 2 kHz, 50% duty cycle symmetric PWM waveform.

Graph view of variable "Voltage_C28[100]" in the C28 Control subsystem:

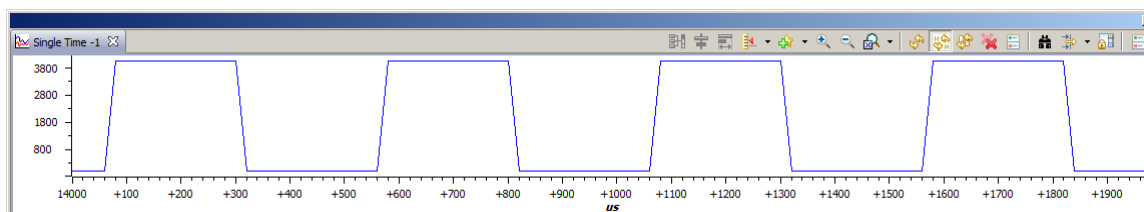


29. Run the C28 Control subsystem code using the "Resume" button on the toolbar (or F8 key).
30. Halt the M3 Master subsystem using the "Suspend" button on the toolbar (or alt+F8).
31. Open and setup a graph to plot a 100-point window of the ADC results buffer.
Click: Tools → Graph → Single Time and set the following values:

Acquisition Buffer Size	100
DSP Data Type	32-bit unsigned integer
Sampling Rate (Hz)	50000
Start Address	Voltage_M3
Display Data Size	100
Time Display Unit	μs

Select OK to save the graph options. The graphical display should show the 2 kHz, 50% duty cycle symmetric PWM waveform.

Graph view of variable "Voltage_M3[100]" in the M3 Master subsystem:



Note: Unsigned integer is 16-bits on the C28 Control subsystem and 32-bits on the M3 Master subsystem.

Terminate Debug Session and Close the Projects

32. Terminate the active debug session using the Terminate button.
33. Next, close the Lab5_C28 and Lab5_M3 projects by right-clicking on each project in the Project Explorer window and select `Close Project`.

End of Exercise

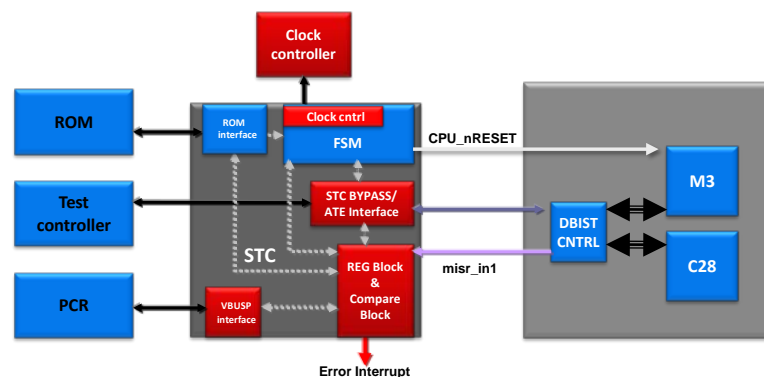
Safety Features

F28M35x Safety Features

- ◆ F28M35x offers Built-In Self Test resources
- ◆ Dual subsystems architecture:
 - ◆ Ability to cross check and monitor each other
 - ◆ Both have access to ADC results (C28 has priority)
 - ◆ Dedicated RAM and peripherals
 - ◆ Communications and controls are separated
- ◆ Enables safety compliance metrics in end applications
- ◆ Three major areas of self test configuration:
 - ◆ CPU Verification
 - ◆ Memory Verification
 - ◆ Peripheral Verification

Self Test Controller (STC) / Memory BIST

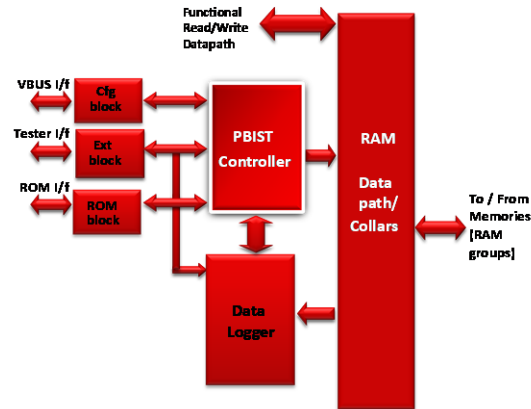
CPU Self Test Controller (STC/LBIST)



- ◆ Provides high diagnostic coverage
- ◆ Significantly lowers software and runtime overhead
- ◆ No software BIST (Built In Self Test) code overhead in flash
- ◆ Simple to configure and start BIST via register

Programmable Memory BIST (PBIST)

- ◆ All on-chip RAM memory can be tested
- ◆ Simple register setup and configuration
- ◆ Typically run at startup, but can be executed during the application
- ◆ Multiple Memory Test Algorithms
- ◆ Detects multiple failure modes

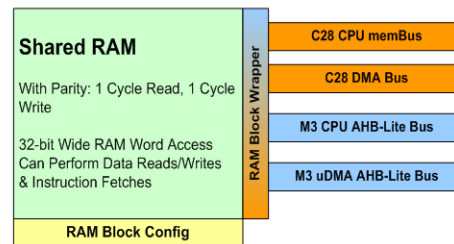


Provides a *mechanism* to determine if *runtime faults* were caused by hard or soft error. This capability can be used to *improve availability* through inline recovery from soft error.

Access Protection - Shared / Dedicated Memory

Access Protection - Shared Memory

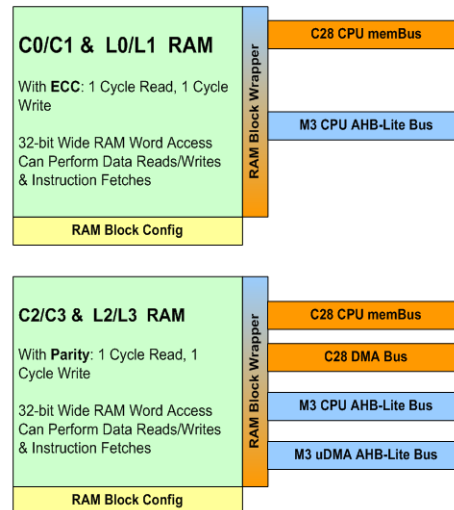
- ◆ M3 decides ownership
- ◆ M3 access can be blocked via memory protection unit (MPU)
- ◆ Owner can further disable accesses:
 - ◆ CPU write protection: enable or disable writes from CPU
 - ◆ Fetch protection: enable or disable fetches by owning CPU
 - ◆ DMA write protection



Ownership	Master Subsystem		Control Subsystem	
	M3 CPU	uDMA	C28	DMA
Master Subsystem*	R/W/Exe	R/W	R	R
Control Subsystem	R	R	R/W/Exe	R/W

Access Protection - Dedicated Memory

- ◆ **M3 Access (Cx)**
 - ◆ C0, C1 = secure memory
 - ◆ M3 access can be blocked via memory protection unit (MPU)
- ◆ **C28 Access (Lx)**
 - ◆ L0, L1 = secure memory
- ◆ **CPU Write Protection**
 - ◆ Prevents Master CPU writes
 - ◆ Can only be disabled by reset
- ◆ **Access Violation Reporting Mechanism**
 - ◆ M3 CPU violation: bus fault
 - ◆ C28 CPU violation: PIE interrupt
 - ◆ Debug information stored in an access violation status register



Error Detection and Correction

Parity Error Detection

- ◆ **Parity detection on:**
 - ◆ Shared RAM (S0-S7)
 - ◆ Message RAM
 - ◆ M3 – C2 & C3 RAM
 - ◆ C28 – L2 & L3 RAM

Bit Positions	34	33	32	31 - 16	15 - 0
Bit Content	Parity for upper 16 bits of data	Parity for lower 16 bits of data	Parity for address	Upper 16 bits of data	Lower 16 bits of data

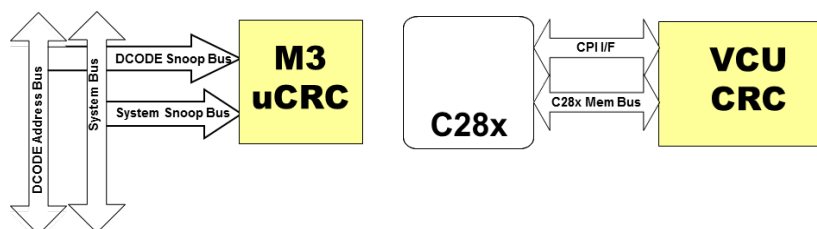
- ◆ **Parity bits**
 - ◆ 1 bit for upper 16-bits
 - ◆ 1 bit for lower 16-bits
 - ◆ 1 bit for address
- ◆ **No performance lost**
 - ◆ 1 cycle 32-bit read or write
 - ◆ 1 cycle even for interleaved CPU & DMA block accesses (M3 – C2/C3 & C28 – L2/L3)
- ◆ **RAM INIT**
 - ◆ Boot-ROM code to protect against ECC or parity errors at power-up
- ◆ **Failure response**
 - ◆ NMI to respective CPU
 - ◆ Error detected is logged

Error Correction Code (ECC)

SECDED – Single Error Correction Double Error Detection

- | | |
|---|--|
| <ul style="list-style-type: none"> ◆ Single-bit error detection & correction <ul style="list-style-type: none"> ◆ Ignore (optional) ◆ Or increment counter and interrupt respective CPU after error threshold is reached ◆ Error detected is logged | <ul style="list-style-type: none"> ◆ Double-bit error detection <ul style="list-style-type: none"> ◆ NMI to respective CPU ◆ Error detected is logged |
|---|--|
-
- | | |
|--|---|
| <ul style="list-style-type: none"> ◆ RAM <ul style="list-style-type: none"> ◆ M3 – C0 and C1 ◆ C28 – L0, L1, M0, and M1 ◆ ECC for data and address <ul style="list-style-type: none"> ◆ 7 bits for upper 16-bits of data ◆ 7 bits for lower 16-bits of data ◆ 7 bits for address ◆ Performance <ul style="list-style-type: none"> ◆ Single-cycle access with either 16 or 32-bit data ◆ Boot INIT <ul style="list-style-type: none"> ◆ Boot-ROM code to protect against ECC or parity errors at power-up | <ul style="list-style-type: none"> ◆ FLASH <ul style="list-style-type: none"> ◆ ECC for data and address <ul style="list-style-type: none"> ◆ 8 bits for upper 64-bits of data ◆ 8 bits for lower 64-bits of data ◆ Performance with ECC enabled <ul style="list-style-type: none"> ◆ 25ns Flash access (40MHz) ◆ 128-bit wide 2-level pre-fetch <ul style="list-style-type: none"> ◆ (8) 16-bit or (4) 32-bit instructions ◆ 8-byte data cache ◆ Same ECC design for M3 and C28 |
|--|---|

CRC - Memory Test



- ◆ Both M3 and C28 have dedicated CRC32 modules
- ◆ Each module requires their respective CPU to point to and read memory
- ◆ Calculated CRC can be compared with expected value for memory verification
- ◆ Secured memory can be verified by running the CRC memory read code from secured memory

Register Protection

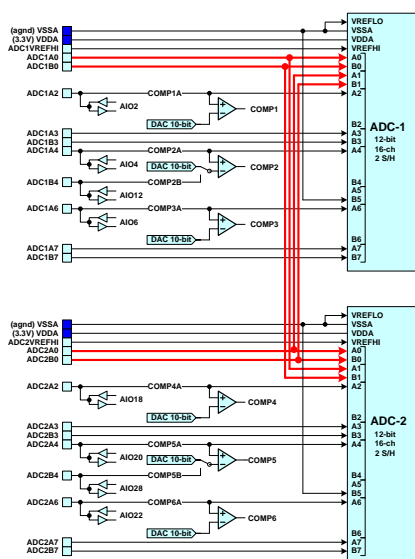
Register Protection

Protection Against Corrupting Critical System Registers

- ◆ **Lock**
 - ◆ Lock pin ownership (Master or Control)
 - ◆ Lock shared SARAM ownership (write once lock register)
- ◆ **M3 “MWRALLOW”**
 - ◆ Enable/disable M3 from writing to critical system registers
 - ◆ Value “a5a5a5a5” is written to MWRALLOW register
 - ◆ MWRALLOW is only available in privilege mode
 - ◆ Debugger access ignore MWRALLOW
- ◆ **C28 “EALLOW”**
 - ◆ Enable/disable C28 from writing to critical registers
 - ◆ Includes flash control registers
 - ◆ Debugger accesses ignore EALLOW

Peripheral and Communication Verification

Peripheral Verification



- ◆ **ADC verification**
 - ◆ Both converters share 4 inputs
 - ◆ ADC1A0 = ADC1A0 & ADC2A1
 - ◆ ADC1B0 = ADC1B0 & ADC2B1
 - ◆ ADC2A0 = ADC1A1 & ADC2A0
 - ◆ ADC2B0 = ADC1B1 & ADC2B0
 - ◆ Redundant channels do not waste pins
 - ◆ Results available to both M3 and C28
- ◆ **PWM verification**
 - ◆ Connect PWM output to ADC input via RC filter
 - ◆ Connect PWM output to input captures
- ◆ **Inverter verification**
 - ◆ Connect PWM signal from the inverter phases to input captures
- ◆ **Fault-safe**
 - ◆ PWM trip zones disable PWMs in 20ns

Communication Verification

◆ DCAN

- ◆ Mailbox RAM Parity
- ◆ Internal Loopback
 - ◆ TX signal internally connected to RX
 - ◆ RX input buffer disconnected
 - ◆ TX buffer is active
 - ◆ Acknowledge bits are ignored
- ◆ External Loopback
 - ◆ TX needs to be connected externally to RX
 - ◆ Acknowledge bits are ignored
 - ◆ Test in addition the RX and TX path
- ◆ Internal Loopback with Silent Mode
 - ◆ Internal loopback with TX buffer disabled
 - ◆ Allow test of DCAN controller without affecting connected CAN network

◆ SCI

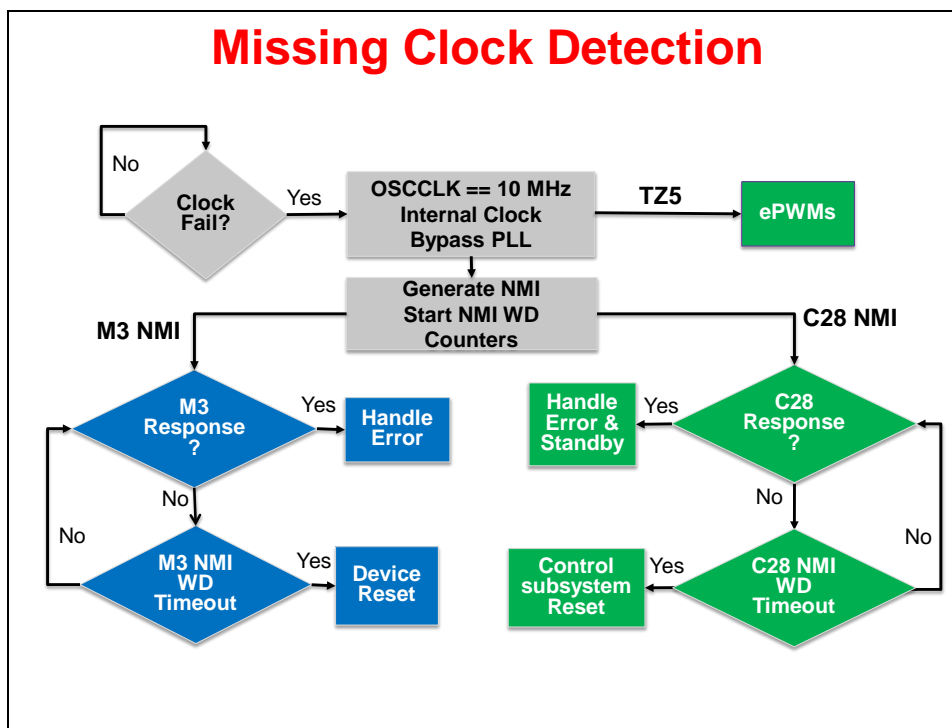
- ◆ Internal Loopback
 - ◆ TX signal internally connected to RX
 - ◆ TX and RX FIFO active
- ◆ M3 can “listen” to C28 SCIA TX
 - ◆ C28.SCIA connected to M3.UART4

◆ SPI

- ◆ Internal Loopback
 - ◆ TX signal internally connected to RX
 - ◆ TX and RX FIFO active
- ◆ M3 can “listen” to C28 SPIA TX
 - ◆ C28.SPIA connected to M3.SSI3

Missing Clock Detection

Missing Clock Detection



Other Resources – The Next Step...

C2000 MCU Multi-day Training Course

C2000 MCU Multi-day Training Course



**In-depth hands-on
TMS320F28069 Design
and Peripheral
Training**

TMS320F2806x Workshop Outline

- Architectural Overview
- Programming Development Environment
- Peripheral Register Header Files
- Reset and Interrupts
- System Initialization
- Analog-to-Digital Converter
- Control Peripherals
- Numerical Concepts
- Direct Memory Access (DMA)
- Control Law Accelerator (CLA)
- Viterbi, Complex Math, CRC Unit (VCU)
- System Design
- Communications
- DSP/BIOS
- Support Resources

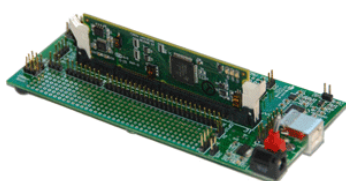
controlSUITE™



Experimenter's Kits

C2000 Experimenter's Kits

F28069, F28035, F28027, F28335, F2808, C28343, C28346, F28M35, F28377D



- ◆ **Part Number:**
 - ◆ TMDSDOCK28069
 - ◆ TMDSDOCK28035
 - ◆ TMDSDOCK28027
 - ◆ TMDSDOCK28335
 - ◆ TMDSDOCK2808
 - ◆ TMDSDOCKKH52C1
 - ◆ TMDSDOCK28377D
- JTAG emulator required for:*
 - ◆ TMDSDOCK28343
 - ◆ TMDSDOCK28346-168
- ◆ **Experimenter Kits include**
 - ◆ controlCARD
 - ◆ USB docking station
 - ◆ C2000 Applications Software CD with example code and full hardware details
 - ◆ Code Composer Studio
- ◆ **Docking station features**
 - ◆ Access to controlCARD signals
 - ◆ Breadboard areas
 - ◆ Onboard USB JTAG Emulation
 - ◆ *JTAG emulator not required*
- ◆ **Available through TI authorized distributors and the TI eStore**

Peripheral Explorer Kit

F28335 Peripheral Explorer Kit



TMDSPREX28335

- ◆ **Experimenter Kit includes**
 - ◆ F28335 controlCARD
 - ◆ Peripheral Explorer baseboard
 - ◆ C2000 Applications Software CD with example code and full hardware details
 - ◆ Code Composer Studio
- ◆ **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)
- ◆ **Onboard USB JTAG Emulation**
 - ◆ *JTAG emulator not required*
- ◆ **Available through TI authorized distributors and the TI eStore**

controlSTICK Evaluation Tool

C2000 controlSTICK Evaluation Tool

F28069, F28027



◆ Part Number:

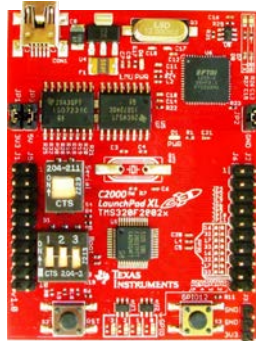
- ◆ TMDX28069USB
- ◆ TMDS28027USB

- ◆ Low-cost USB evaluation tool
- ◆ Onboard JTAG Emulation
 - ◆ JTAG emulator not required
- ◆ Access to controlSTICK signals
- ◆ C2000 Applications Software CD with example code and full hardware details
- ◆ Code Composer Studio
- ◆ Available through TI authorized distributors and the TI eStore

LaunchPad Evaluation Kit

C2000 LaunchPad Evaluation Kit

F28027, F28027F




◆ Part Number:

- ◆ LAUNCHXL-F28027
- ◆ LAUNCHXL-F28027F

- ◆ Low-cost evaluation kit
 - ◆ F28027 standard version
 - ◆ F26027F version with InstaSPIN-FOC
- ◆ Various BoosterPacks available
- ◆ Onboard JTAG Emulation
 - ◆ JTAG emulator not required
- ◆ Access to LaunchPad signals
- ◆ C2000 Applications Software with example code and full hardware details in available in ControlSUITE
- ◆ Code Composer Studio
- ◆ Available through TI authorized distributors and the TI eStore

Application Kits

C2000 controlCARD Application Kits



- ◆ Developer's Kit for – *Motor Control, PFC, High Voltage, Digital Power, Renewable Energy, LED Lighting, etc.*
- ◆ Kits includes
 - ◆ controlCARD and application specific baseboard
 - ◆ Code Composer Studio
- ◆ Software download includes
 - ◆ Complete schematics, BOM, gerber files, and source code for board and all software
 - ◆ Quick-start demonstration GUI for quick and easy access to all board features
 - ◆ Fully documented software specific to each kit and application
- ◆ See www.ti.com/c2000 for other kits and more details
- ◆ Available through TI authorized distributors and the TI eStore

C2000 Workshop Download Wiki

C2000 Workshop Download Wiki



Navigation

- Main Page
- All pages
- All categories
- Popular pages
- Popular authors
- Popular categories
- Category stats
- Recent changes
- Random page
- Help
- Google Search

Page [Discussion](#) [Read](#) [View source](#) [View history](#)

Hands-On Training for TI Embedded Processors

Hands-On Training for TI Embedded Processors

Translate this page to [zh-CN - 中文\(中国大陆\)](#) [Translate](#)

TI's Technical Training Organization (TTO) conducts hands-on training for TI embedded processors at various worldwide locations. You can access the workshop materials from this site, organized by specific processor families. You can also enroll in a live workshop using the links below.

Workshop Descriptions and Materials

C2000™ 32-bit Real-Time MCU Training

- C2000™ One-Day Workshop - online videos provided
- C2000™ Multi-Day Workshop
- F28M35x™ Workshop
- F2837xD™ Workshop
- C2000™ Archived Workshops (F2407 / F2812 / F2808 / F28335 / Delfino / Piccolo)

<http://www.ti.com/hands-on-training>

For More Information...

For More Information . . .

- ◆ **USA – Product Information Center (PIC)**
 - ◆ Phone: 800-477-8924 or 512-434-1560
 - ◆ E-mail: support@ti.com
- ◆ **TI E2E Community (videos, forums, blogs)**
 - ◆ <http://e2e.ti.com>
- ◆ **Embedded Processor Wiki**
 - ◆ <http://processors.wiki.ti.com>
- ◆ **TI Training**
 - ◆ <http://www.ti.com/training>
- ◆ **TI eStore**
 - ◆ <http://estore.ti.com>
- ◆ **TI website**
 - ◆ <http://www.ti.com>

Appendix – F28M35xx Experimenter's Kit

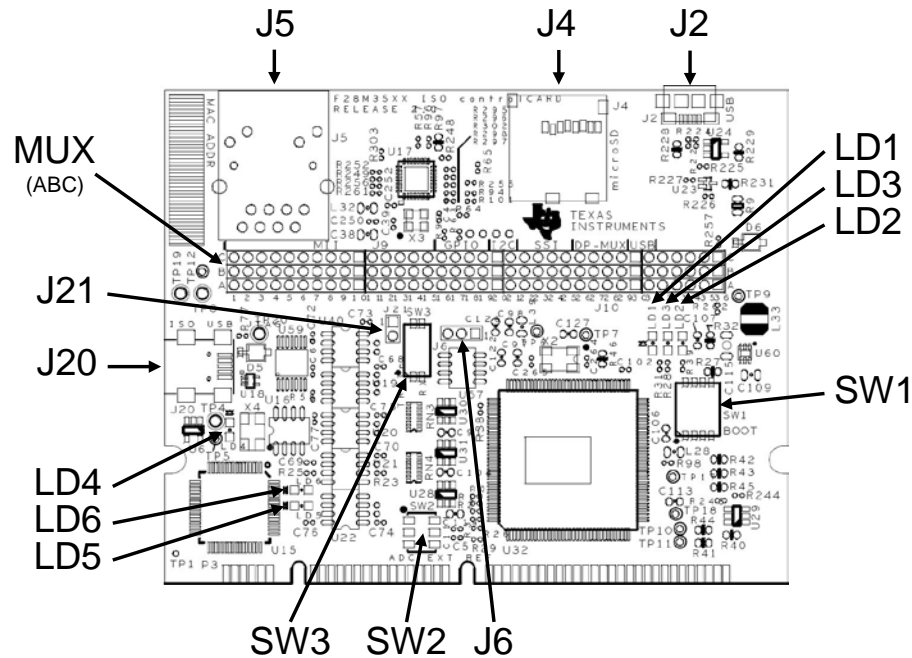
F28M35xx controlCARD

- Isolated JTAG – ISO JTAG
- Connectivity Ports/Jumpers
- LEDs
- SW1
- SW2
- SW3
- Connectivity Mux Connector (ABC)
- MAC Address
- Connectivity Mux Control/Connectivity Signal Mapping

Docking Station

- SW1 / LD1
- JP1 / JP2
- J1 / J2 / J3 / J8 / J9
- F2833x Boot Mode Selection
- F280xx Boot Mode Selection
- J3 – DB-9 to 4-Pin Header Cable

F28M35xx controlCARD



Isolated JTAG – ISO JTAG

J20 – USB_A connector is intended for XDS100V2 JTAG emulation and SCI communication through dedicated FTDI logic

Connectivity Ports/Jumpers

J2 – USB micro AB connector supports USB 2.0 host/device

J4 – SD Micro card adaptor through SPI port

J5 – Ethernet port

J6 – EEPROM write protect

Jumpers on pins 1-2 connects WP pin to 3.3V (protected)

Jumpers on pins 2-3 connects WP pin to GND (not protected)

J21 – Default unconnected

Connected if isolated EMU0 signal is needed for JTAG

LEDs

LD1 – Turns on when controlCARD is powered on (**Green**)

LD2 – Controlled by PC6_GPIO71 (**Red**)

LD3 – Controlled by PC7_GPIO70 (**Red**)

LD4 – Power for ISO JTAG logic (**Green**)

LD5 – UART/SCI Rx activity indicator

LD6 – UART/SCI Tx activity indicator

SW1 – Controls the Boot Mode Options of the F28M35xx Device

M3 boot Mode	C28x boot Mode	PG2_GPIO34	PG3_GPIO35	PG7_GPIO47	PF3_GPIO43
Boot from Parallel IOs	Boot from Master	X	0	0	0
Boot from M3 RAM	Boot from M3	X	0	0	1
Boot from M3 serial peripherals (UART0/SSI0/I2C0)	Boot from M3	X	0	1	0
Boot from M3 CAN	Boot from M3	X	0	1	1
Boot from M3 Ethernet	Boot from M3	X	1	0	0
Boot from M3 Flash	Boot from Master	X	1	1	1

SW2 – ADC VREF Control

The ADC reference will range from 0 to 3.3V by default. However, if the ADC in the ADC registers is configured to use external limits, the ADC will convert its full range of resolution from VREF-LO to VREF-HI.

Position1 – Controls VREF-HI, the value that the ratio-metric ADC will convert as the maximum 12-bit value, which is 0x0FFF. In the downward position, VREF-HI will be connected to 3.3V. In the upward position, VREF-HI will be connected to pin 66 of the DIMM100-socket. This would allow a connecting board to control the ADC VREF-HI value. This extends VREF-HI connections to both the ADCs on the F28M35xx device.

Position 2 – Controls VREF-LO, the value that the ratio-metric ADC will convert as the minimum 12-bit value, which is 0x0000. In the downward position, VREF-LO will be connected to 0V. In the upward position, VREF-LO will be connected to pin 16 of the DIMM100-socket. This would allow a connecting board to control the ADC-VREFLO value. This extends VREF-LO connections to both the ADCs on the F28M35xx device.

SW3 – TRST/ ISO SCI Communication Signal Enables

Position 1:

ON - TRST signal from ISO JTAG circuit will be connected to F28M35xx. Needed during JTAG debug using ISO JTAG.

OFF - TRST signal from ISO JTAG circuit will NOT be connected to F28M35xx. Needed when the application is running from flash at power up without the JTAG connections.

Position 2:

ON - RS-232 transceiver will be enabled and allow communication through a serial cable via pins 2 and 42 of the DIMM-100 socket. Putting SW3 in the “ON” position will allow the F28M35xx controlCARD to be DIMM signal compatible with the F2808, F28044, F28335, F28035 and F28027 controlCARDS. GPIO-28 will be stuck as logic high in this position.

OFF - The default option. SW5 in the “OFF” position allows GPIO-28 to be used as a GPIO. Serial communication is still possible, through the FTDI – FT2232 chip.

Connectivity Mux Connector (ABC)

F28M35xx MCU GPIO functions have been partitioned as “connectivity ports” and “control ports”. DIMM 100 connector supports all the control signals/standard communication functions and maintains compatibility across all C2000 28x device families.

Connectivity mux – The mux uses physical jumpers to allow easy access to the connectivity ports on the top edge of the card (i.e. USB, Ethernet, uSDCARD) or redirects the GPIOs to the DIMM100 connector to maintain signal compatibility with the C2000 legacy applications and devices.

Unmuxed signals (pins 32 -36) should not be populated with jumpers. These are signals are intended to be used with boards that can be connected to Connectivity Mux.

MAC Address

F28M35xx device Ethernet examples use a fixed TI's MAC address: A8-63-F2-00-00-80. Refer to the board label for a unique TI's MAC ID assigned for each F28M35xx controlCARD. User applications can program a fixed MAC address in the non-volatile memory reserved for MAC address. Refer to device documentation for details.

Connectivity Mux Control/Connectivity Signal Mapping

		F28M35xx -cCard	Revision 1.1
A_Row	B_Row	C_Row	
To DIMM 100 connector	F28M35xx or DIMM	To Connectivity ports	
C28_GPIO49	PH1_GPIO49	M3_MII_RXD0	Ethernet
C28_GPIO39	PE6_GPIO36	M3_MII_MDIO	
C28_GPIO40	PG0_GPIO40	M3_MII_RXD2	
C28_GPIO41	PG1_GPIO41	M3_MII_RXD1	
C28_GPIO43	PG3_GPIO43	M3_MII_RXDV	
C28_GPIO51	PH3_GPIO51	M3_MII_TXD2	
C28_GPIO54	PH6_GPIO54	M3_MII_TXEN	
C28_GPIO55	PH7_GPIO55	M3_MII_TXCK	
C28_GPIO56	PJ0_GPIO56	M3_MII_RXER	
C28_GPIO58	PJ2_GPIO58	M3_MII_RXCK	
C28_GPIO59	PJ3_GPIO59	M3_MII_MDC	
C28_GPIO60	PJ4_GPIO60	M3_MII_COL	
C28_GPIO61	PJ5_GPIO61	M3_MII_CRS	
C28_GPIO62	PJ6_GPIO62	M3_MII_PHYINTRn	
C28_GPIO63	PJ7_GPIO63	M3_MII_PHYRSTn	
C28_GPIO32	PF0_GPIO32	M3_PF0/CAN1RX	CAN0/1
C28_GPIO33	PF1_GPIO33	M3_PF1/CAN1TX	
C28_GPIO6	PA6_GPIO6	M3_PA6/CAN0RX	
C28_GPIO31	PE7_GPIO31	M3_PE7/CAN0TX	
C28_GPIO14	PB6_GPIO14	M3_I2C0SDA	I2C
C28_GPIO15	PB7_GPIO15	M3_I2C0SCL	
C28_GPIO16	PD0_GPIO16	M3_SSI0TX	SSI
C28_GPIO17	PD1_GPIO17	M3_SSI0RX	
C28_GPIO18	PD2_GPIO18	M3_SSI0Clk	
C28_GPIO19	PD3_GPIO19	M3_SSI0Fss	
C28_GPIO54	DIMM_pin 20	C28_GPIO60	DIMM IO options
C28_GPIO55	DIMM_pin 70	C28_GPIO61	
C28_GPIO56	DIMM_pin 22	C28_GPIO62	
C28_GPIO57	DIMM_pin 72	C28_GPIO63	
C28_GPIO57	PJ1_GPIO57	M3_USB0FLT	USB
C28_GPIO42	PG2_GPIO42	M3_USB0DM	
PC4_GPIO68/MII_RXD3	PG5_GPIO45/USB0DP	PF5_GPIO37/MII_RXD3	Un_muxed signals
PH4_GPIO52/MII_TXD1	PF6_GPIO38/USB0VBUS	PH5_GPIO53/MII_TXD0	
PG6_GPIO46/USB0ID	PC5_GPIO46/USB0EPEN	PG7_GPIO47/MII_TXER	
NC	GND	NC	
3.3V	NC	5V	

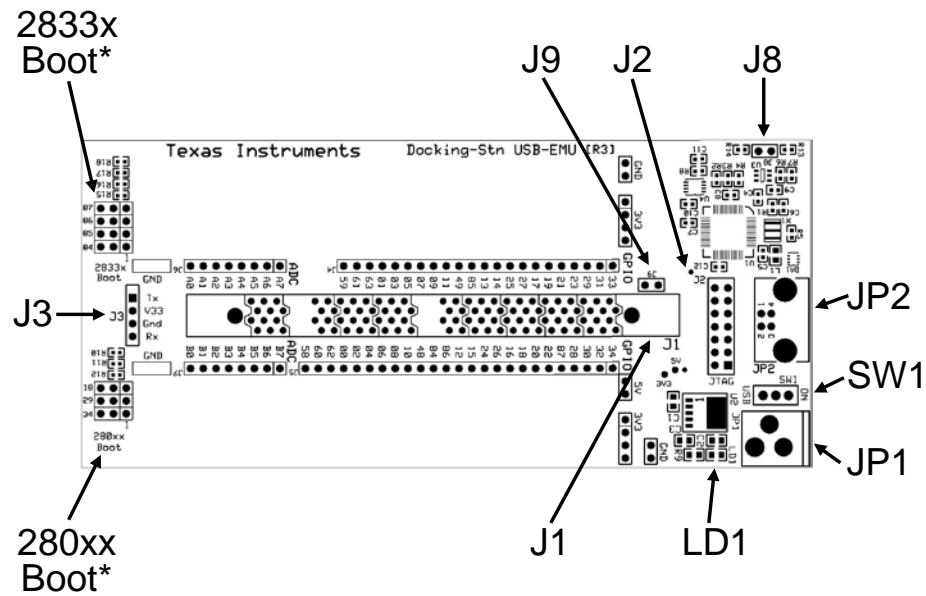
DIMM signal / Peripheral functions are active if Jumper selections are placed per the color code

Connects to DIMM 100 compatible signal map

Do not populate

Do not populate

Docking Station



***Note:** Jumper Left = 1; Jumper Right = 0

SW1 / LD1

SW1 – USB: Power from USB; ON – Power from JP1

LD1 – Power-On indicator

JP1 / JP2

JP1 – 5.0 V power supply input

JP2 – USB JTAG emulation port

J1 / J2 / J3 / J8 / J9

J1 – ControlCARD 100-pin DIMM socket

J2 – JTAG header connector

J3 – UART communications header connector

J8 – Internal emulation enable/disable jumper (NO jumper for internal emulation)

J9 – User virtual COM port to C2000 device (Note: ControlCARD would need to be modified to disconnect the C2000 UART connection from header J3)

Note: The internal emulation logic on the Docking Station routes through the FT2232 USB device. By default this device enables the USB connection to perform JTAG communication and in parallel create a virtual serial port (SCI/UART). As shipped, the C2000 device is not connected to the virtual COM port and is instead connected to J3.

F2833x Boot Mode Selection

MODE	GPIO87/XA15	GPIO86/XA14	GPIO85/XA13	GPIO84/XA12	MODE ⁽¹⁾
F	1	1	1	1	Jump to Flash
E	1	1	1	0	SCI-A boot
D	1	1	0	1	SPI-A boot
C	1	1	0	0	I2C-A boot
B	1	0	1	1	eCAN-A boot
A	1	0	1	0	McBSP-A boot
9	1	0	0	1	Jump to XINTF x16
8	1	0	0	0	Jump to XINTF x32
7	0	1	1	1	Jump to OTP
6	0	1	1	0	Parallel GPIO I/O boot
5	0	1	0	1	Parallel XINTF boot
4	0	1	0	0	Jump to SARAM
3	0	0	1	1	Branch to check boot mode
2	0	0	1	0	Branch to Flash, skip ADC calibration
1	0	0	0	1	Branch to SARAM, skip ADC calibration
0	0	0	0	0	Branch to SCI, skip ADC calibration

⁽¹⁾ All four GPIO pins have an internal pullup.

F280xx Boot Mode Selection

Mode	Description	GPIO18 SPICLKA ⁽¹⁾ SCITXDB	GPIO29 SCITXDA	GPIO34
Boot to Flash ⁽²⁾	Jump to flash address 0x3F 7FF6. You must have programmed a branch instruction here prior to reset to redirect code execution as desired.	1	1	1
SCI-A Boot	Load a data stream from SCI-A.	1	1	0
SPI-A Boot	Load from an external serial SPI EEPROM on SPI-A.	1	0	1
I2C Boot	Load data from an external EEPROM at address 0x50 on the I2C bus.	1	0	0
eCAN-A Boot ⁽³⁾	Call CAN_Boot to load from eCAN-A mailbox 1.	0	1	1
Boot to M0 SARAM ⁽⁴⁾	Jump to M0 SARAM address 0x00 0000.	0	1	0
Boot to OTP ⁽⁴⁾	Jump to OTP address 0x3D 7800.	0	0	1
Parallel I/O Boot	Load data from GPIO0 - GPIO15.	0	0	0

⁽¹⁾ You must take extra care because of any effect toggling SPICLKA to select a boot mode may have on external logic.

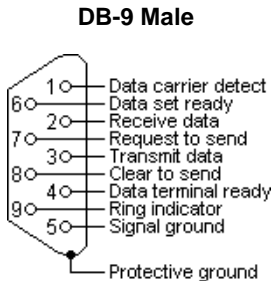
⁽²⁾ When booting directly to flash, it is assumed that you have previously programmed a branch statement at 0x3F 7FF6 to redirect program flow as desired.

⁽³⁾ On devices that do not have an eCAN-A module this configuration is reserved. If it is selected, then the eCAN-A bootloader will run and will loop forever waiting for an incoming message.

⁽⁴⁾ When booting directly to OTP or M0 SARAM, it is assumed that you have previously programmed or loaded code starting at the entry point location.

J3 – DB-9 to 4-Pin Header Cable

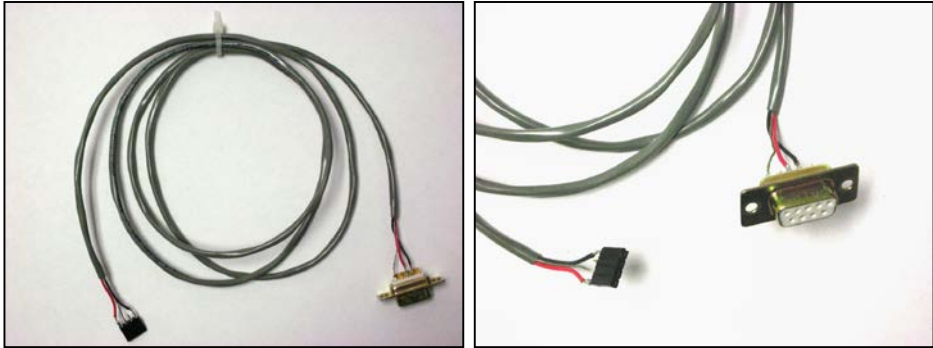
Note: This cable is NOT included with the Experimenter's Kit and is only shown for reference.



Pin-Out Table for Both Ends of the Cable:

DB-9 female Pin#	SIL 0.1" female Pin#
2 (black)	1 (TX)
3 (red)	4 (RX)
5 (bare wire)	3 (GND)

Note: pin 2 on SIL is a no-connect



Notes:

Notes: