Important Notice

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

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

Customers are responsible for their applications using TI components.

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

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

Copyright © 2003 – 2004 Texas Instruments Incorporated

Revision History

February 2003 – Revision 1.0
March 2003 – Revision 1.1
July 2004 – Revision 2.0

Mailing Address

Texas Instruments
Training Technical Organization
7839 Churchill Way
M/S 3984
Dallas, Texas 75251-1903
TMS320C28x 1-Day Workshop

TMS320C28x Workshop Agenda

- M1: C28x 1-Day Workshop Overview
- M2: Introduction to the TMS320F281x
- M3: Development Tools
- M4: Control Peripherals
- Lunch
- M5: DSP/BIOS Real-Time OS
- M6: Flash Programming
- M7: The Next Step...
This page is intentionally blank.
**Introduction**

This module will provide a brief introduction to the TMS320C281x 1-day workshop, and explain the general lab exercise flow. The TMS320 family members will be discussed, and various members of the C2000 family will be highlighted. Additionally, the broad application base of the C28x will be explored.

**Learning Objectives**

- Introductions and Workshop Lab Overview
- TMS320 Platforms and the C2000
- Review Broad Application Base
- TMS320C2000 Family Members
## Module Topics

<table>
<thead>
<tr>
<th>Workshop Topics</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C28x 1-Day Workshop Overview</td>
<td>1-1</td>
</tr>
<tr>
<td>Module Topics</td>
<td>1-2</td>
</tr>
<tr>
<td>Workshop Introduction</td>
<td>1-3</td>
</tr>
<tr>
<td>TMS320C28x Device Family</td>
<td>1-4</td>
</tr>
</tbody>
</table>
Workshop Introduction

Introductions

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

Workshop Lab Overview

- M3: Set up eZdsp™ F2812 Starter Kit
- M4: Use the Event Manager to Generate and View PWM Waveforms
- M5: DSP/BIOS and Filtering PWM Waveform using IQmath
- M6: Programming an Application into Flash Memory
TMS320C28x Device Family

Multiple Platforms Provide the Right Solutions for Your Application

C2000
Power Efficient Performance
Efficient Integration for Control

C5000
High Performance ‘C’ Efficiency

C6000

C2000 Portfolio Expanding with Price/Performance Optimized Derivatives

High-Precision Control

High-end Derivatives

150 MIPS!

F2810 150 MIPS
F2811 150 MIPS
R2810 150 MIPS
R2811 150 MIPS
F2812 150 MIPS
C280x 100 MIPS

24x™ up to 40 MIPS

Multi-Function, Appliance & Consumer Control

Application specific versions

Cost optimized versions
Workshop Introduction

Broad C28x™ Application Base

Optical Networking
Control of laser diode

Digital Power Supply
Provides control, sensing, PFC, and other functions

Printers
Print head control
Paper path motor control

Evaluating Other Segments
eg. Musical Instruments

Non-traditional Motor Control
Many new cool applications to come

Automotive

TI C2000: Industry’s Broadest Product Portfolio for Embedded Applications

<table>
<thead>
<tr>
<th>CPU</th>
<th>F2837</th>
<th>F2833</th>
<th>F2832</th>
<th>F2831</th>
<th>F2830</th>
<th>F2828S</th>
<th>( \frac{32}{16} )</th>
<th>( \frac{20}{12} )</th>
<th>( \frac{16}{12} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIPS</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>RAM (words)</td>
<td>16K</td>
<td>16K</td>
<td>16K</td>
<td>16K</td>
<td>16K</td>
<td>16K</td>
<td>16K</td>
<td>16K</td>
<td>16K</td>
</tr>
<tr>
<td>ROM (words)</td>
<td>32K</td>
<td>32K</td>
<td>32K</td>
<td>32K</td>
<td>32K</td>
<td>32K</td>
<td>32K</td>
<td>32K</td>
<td>32K</td>
</tr>
<tr>
<td>Event Manager</td>
<td>4K</td>
<td>4K</td>
<td>4K</td>
<td>4K</td>
<td>4K</td>
<td>4K</td>
<td>4K</td>
<td>4K</td>
<td>4K</td>
</tr>
<tr>
<td>ADC Resolution</td>
<td>12-bit</td>
<td>10-bit</td>
<td>10-bit</td>
<td>10-bit</td>
<td>10-bit</td>
<td>10-bit</td>
<td>10-bit</td>
<td>10-bit</td>
<td>10-bit</td>
</tr>
<tr>
<td># of Chan</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Conv. Time</td>
<td>15ns</td>
<td>15ns</td>
<td>15ns</td>
<td>15ns</td>
<td>15ns</td>
<td>15ns</td>
<td>15ns</td>
<td>15ns</td>
<td>15ns</td>
</tr>
<tr>
<td>McBSP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>EXMIF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Watch Dog</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SPI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SCI (UART)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>CAN</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Volts (V)</td>
<td>3.3</td>
<td>2.75</td>
<td>3.3</td>
<td>2.75</td>
<td>3.3</td>
<td>2.75</td>
<td>3.3</td>
<td>2.75</td>
<td>3.3</td>
</tr>
<tr>
<td># I/O</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>Package</td>
<td>176LQFP</td>
<td>128LQFP</td>
<td>144QFP</td>
<td>100LQFP</td>
<td>256QFP</td>
<td>160QFP</td>
<td>192QFP</td>
<td>256QFP</td>
<td>320QFP</td>
</tr>
</tbody>
</table>
This page is intentionally blank.
Introduction

This module provides an overview of the TMS320F281x architecture. Features of the core, memory, control peripherals, and communication ports will be presented.

Learning Objectives

- TMS320F281x Architecture
  - C28x DSP Core
  - Memory Sub-System
  - Control Peripherals
  - Communication Peripherals
Module Topics

Introduction to the TMS320F281x

Module Topics

TMS320F2812/11/10
High Performance DSP Core
Memory Sub-System
Control Peripherals
Communication Ports
TMS320F2812 Memory Map
Fast Interrupt Manager
High Performance DSP Core

C/C++ Efficient 32-bit DSP Core

Interrupt Management

C28x™ 32-bit DSP

- 32x32 bit Multiplier
- 32-bit Timers (3)
- Real-Time JTAG

C28x™ DSP Core

- Single-cycle 32-bit multiplier makes computationally intensive control algorithms more efficient
- Three 32-bit timers support multiple control loops / time bases
- Single cycle read-modified-write in any memory location and 32-bit registers improve control algorithm efficiency
- Real-time JTAG debug shortens development cycle
- Fast & flexible interrupt management significantly reduce interrupt latency
- Code compatible with the C24x family
Memory Sub-System

Flash Memory

Quarter of a Megabyte of on-chip Flash Memory

Fast program execution out of both RAM and Flash memory
- 100-120 MIPS with Flash Acceleration Technology
- 150 MIPS out of RAM for time-critical code

Up to 128K x 16 Flash
- (8 x 4K and 6 x 16K Sectors)

128-bit security protects software investment

External memory interface (XINTF) supports systems with larger memory models (up to 1MW address reach)

Control Peripherals

12-bit Analog-to-Digital Converter

Fast & Flexible 12-bit 16-Channel ADC
- 12.5 MSPS throughput (80ns pipeline conversion, 200ns single conversion)
- Dual sample/hold enable simultaneous sampling or sequencing modes
- Analog input: 0V to 3V
- 16 channel, multiplexed inputs
- Auto Sequencer supports up to 16 conversions without CPU intervention
- Sequencer can be operated as two independent 8-state sequencers or as one large 16-state sequencer
- Sixteen result registers (individually addressable) to store conversion values
Two Event Managers

- GP Timer 1 (Counter & Period)
- GP Timer 2 (Counter & Period)
- GP Compare 1
- GP Compare 2
- Full Compare
- Dead Band Logic
- Output Logic
- Clock Select
- PWM Trip Enable Controller
- Input Logic
- MUX
- Capture

Control Peripherals (EVA + EVB)

- 16 PWMs
- 6 Complimentary Pairs + 4 Independent + Trip Inputs + Programmable Dead-Bands
- 10 16-bit Compares, 4 General Purpose Timers
- 6.67ns max PWM Resolution
- 2 Quadrature Encoder Interface with Index Input
- 6 Capture Inputs With Programmable Resolution

Communication Ports

Multi-Channel Buffered Serial Port

- McBSP
- FIFO Interface
- McBSP Module
- 32-bit Read FIFO
- Mode Select
- 32-bit Write FIFO
- Transmit/Receive FIFO Control Regs
- Interrupt & Events
- Interrupt Select Logic

Communications Ports

- Full-duplex communication
- 16-level, 32-bit transmit/receive FIFOs
- 8, 12, 16, 20, 24, and 32-bit data
- 8-bit data, option of LSB or MSB transmitted first
- 128 channels for reception and transmission
- Direct interface to SPI & IIS devices
Enhanced Controller Area Network

- Communications Ports
  - 2.0B compliant
  - 32 Mailboxes
  - 32 Local receive masks
  - 32 Interrupt masks
  - Low-power mode
  - Programmable wake-up on bus activity
  - Automatic reply to a remote message request
  - Time-stamping of messages

SCI (UART) and SPI

- Communication Ports
  - SCI-UART (2 modules)
    - Two wake up multiprocessor modes idle-line and address bit
    - Half or full duplex operation
    - 16-level transmit / receive FIFO buffer
  - SPI
    - Auto-baud-detect Hardware Logic
    - Delayed transmit control
    - 16 level transmit / receive FIFO
TMS320F2812 Memory Map

![TMS320F2812 Memory Map Diagram]

Fast Interrupt Manager

**C28x Fast Interrupt Response Manager**

- 96 dedicated PIE vectors
- No software decision making required
- Direct access to RAM vectors
- Auto flags update
- Concurrent auto context save
- Auto Context Save
- Multiplier (T, PH, PL)
- Accumulator (AH, AL)
- Status Reg (ST0, ST1)
- Program Counter
- Aux. Register 0 & 1
- Interrupt Enable Reg
- Data Page Pointer

Minimum interrupt latency of 14 – 16 cycles (approximately 100ns at 150MHz)
This page is intentionally blank.
Development Tools

Introduction

This module will discuss the various development tools that are available for the design process. These development tools will include the Code Composer Studio (CCS) integrated development environment (IDE), the F28xx Peripheral Register Header Files, and the IQmath Library. The Peripheral Register Header files simplify the programming of the many peripherals on the C28x family of devices. The IQmath library can seamlessly port floating-point algorithms into fixed-point code. Additionally, the setup and operation of the TMS320F2812 eZdsp™ kit will be explained.

Learning Objectives

- Explore Code Composer Studio Development Environment
- Understanding the usage of the F28xx Peripheral Register Header Files
- Describe the use of the IQmath Library
- Explain eZdsp™ F2812 Features and Hardware
- Review eZdsp™ and PC Connections
Module Topics

Development Tools ................................................................................................................................. 3-1

Module Topics..................................................................................................................................... 3-2

Code Composer Studio Development Environment .................................................................................. 3-3
  CCS Project........................................................................................................................................ 3-4
  Build Options................................................................................................................................... 3-5

F28xx Peripheral Register Header Files ................................................................................................. 3-6

IQmath Library..................................................................................................................................... 3-8

eZdsp™ F2812 ...................................................................................................................................... 3-10
  eZdsp™ F2812 Hardware.................................................................................................................. 3-10
  Bootloader Options.......................................................................................................................... 3-11
  Connecting the eZdsp™ to PC ......................................................................................................... 3-11

Lab 3: eZdsp™ F2812 ........................................................................................................................... 3-12
Code Composer Studio Development Environment

Code Composer Studio

- Code Composer Studio includes:
  - Integrated Edit/Debug GUI
  - Code Generation Tools
  - DSP/BIOS

Code Composer Studio

Project Manager:
- Source & object files
- File dependencies
- Compiler, Assembler & Linker build options

Full C/C++ & Assembly Debugging:
- C & ASM Source
- Mixed mode
- Disassembly (patch)
- Set Break Points
- Set probe Points

Editor:
- Structure Expansion

Menus or Icons

Help

CPU Window

Status window

Watch window

Graph window

Memory window
Code Composer Studio Development Environment

**Code Composer Studio: IDE**

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

**CCS Project**

**The CCS Project**

Project (.pj) files contain:

- Source files (by reference)
  - Source (C, assembly)
  - Libraries
  - DSP/BIOS configuration
  - Linker command files
- Project settings:
  - Build Options (compiler and assembler)
  - Build configurations
  - DSP/BIOS
  - Linker
Build Options

Build Options GUI - Compiler

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

Build Options GUI - Linker

- GUI has 2 categories for linking
- Specifies various link options
- "\Debug\" indicates on subfolder level below project (.pj) location
F28xx Peripheral Register Header Files

DSP281x Header File Package
(http://www.ti.com, literature # SPRC097)

- Simplifies program of peripherals and other functions
- Takes care of register definitions and addresses
- Header file package consists of:
  - \DSP281x_headers\include .h files
  - \DSP281x_common\src .c source files
  - \DSP281x_headers\cmd linker command files
  - \DSP281x_headers\gel .gel files for CCS
  - \DSP281x_examples example programs
  - \doc documentation

- TI has done all of the work for you!

Structure Naming Conventions

- The DSP281x header files define:
  - All of the peripheral structures
  - All of the register names
  - All of the bit field names
  - All of the register addresses

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

Notes:
[1] "PeripheralName" are assigned by TI and found in the DSP281x header files. They are a combination of capital and small letters (i.e. CpuTimer0Regs).
[2] "RegisterName" are the same names as used in the data sheet. They are always in capital letters (i.e. TCR, TIM, TPR,..).
[3] "FieldName" are the same names as used in the data sheet. They are always in capital letters (i.e. POL, TOG, TSS,..).
The CCS Watch Window using Structures

Code Maestro to the Rescue!
IQmath Library

IQmath Library: Floating Point “Ease of Use” on a Fixed Point Machine

- Control algorithms typically start in a Floating-Point format.
- The conversion of such algorithms, to run on a fixed-point machine, is a laborious and time-consuming task.
- The 32-bit math capabilities of the C28x™ core enable a new C/C++ approach, which makes this task easier and much faster—optimized to take advantage of C28x™ architecture.

IQ Fractional Representation

\[ S \underbrace{\ldots}_{32 \text{ bit mantissa}} \cdot \underbrace{\ldots}_{0} \]

\[ -2^1 + 2^{-1} + \ldots + 2^1 + 2^0 \cdot 2^{-1} + 2^{-2} + \ldots + 2^{-Q} \]
**IQmath**

A mathematical approach and a set of supporting math libraries that enable the following:

- Reduced time to implement/port/debug math algorithms in C/C++
- Increased numerical resolution of algorithms from 16-bits to 32/64-bits

**Typical**

Users typically start with a floating point algorithm...

```c
float Y, M, X, B;
Y = M * X + B;
```

and then spend many hours converting to a fixed point algorithm which is not easy to read

```c
int Y, M, X, B;  //Q1 to Q15
Y = ((M * X) + B << Q) >> Q);
```

**IQmath**

IQmath reduces this effort dramatically and the code is easier to read (looks "natural")

(Using IQmath in C):

```c
_iq Y, M, X, B;  //Q1 to Q30
Y = _IQmpy(M, X) + B;
```

(Using IQmath in C++):

```c
iq Y, M, X, B;  //Q1 to Q30
Y = M * X + B;
```

**IQmath Approach Summary**

"IQmath" + fixed-point processor with 32-bit capabilities =

- Seamless portability of code between fixed and floating-point devices
  - User selects target math type in "IQmathLib.h" file
    - `#if MATH_TYPE == IQ_MATH`
    - `#if MATH_TYPE == FLOAT_MATH`
- One source code set for simulation vs. target device
- Numerical resolution adjustability based on application requirement
  - Set in "IQmathLib.h" file
    - `#define GLOBAL_Q 18`
    - Explicitly specify Q value
      - `_iq20 X, Y, Z;`
- Numerical accuracy without sacrificing time and cycles
- Rapid conversion/porting and implementation of algorithms

IQmath library is freeware - available from TI DSP website

http://www.dspvillage.ti.com (follow C2000 DSP links)
eZdsp™ F2812 Features

- TMS320F2812 Digital Signal Processor
- 150 MIPS Operating Speed
- 18K Words On-chip RAM
- 128K Words On-chip Flash Memory
- 64K Words Off-chip SRAM Memory
- 30 MHz Clock
- Expansion Connectors
- On-board IEEE 1149.1 JTAG Controller
- TI F28xx CCS Tools Driver

eZdsp™ F2812 Hardware

- JTAG Interface (P1)
- EXPANSION Data & Address (P2)
- SRAM 64K x 16
- Parallel Port/ JTAG Controller Interface (P3)
- Power Connector (P6) +5V
- I/O Interface (P4/P8/P7)
- Bootloader GPIO Pins
- TMS320F2812 - DSP
- ANALOG Interface (P5/P9)
## Bootloader Options

### Bootloader Options

- **Execution Entry Point**
- **Bootloading Routines**
  - FLASH
  - H0 SARAM
  - OTP
  - SPI
  - SCI-A
  - Parallel load

<table>
<thead>
<tr>
<th>GPIO pins</th>
<th>F4</th>
<th>F12</th>
<th>F3</th>
<th>F2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- Jump to FLASH address 0x3F 7FF6
- Jump to H0 SARAM address 0x3F 8000
- Jump to OTP address 0x3D 7800
- Bootload external EEPROM to on-chip memory via SPI port
- Bootload code to on-chip memory via SCI-A port
- Bootload code to on-chip memory via GPIO port B (parallel)

### Connecting the eZdsp™ to PC

#### Connecting the eZdsp™ to your PC

- 25 pin male D-sub connector (Plugs into parallel port on PC)
- 25 pin female D-sub connector

---

**Code Composer Studio – eZdsp™ F2812 Configuration**
Lab 3: eZdsp™ F2812

Lab 3: eZdsp™ F2812 Kit

- Setup the eZdsp™ F2812 kit using the quick start guide
  - Install Code Composer Studio with drivers
  - Connect eZdsp™ hardware to PC
  - Configure and setup CCS software
  - Start CCS by clicking on icon
- Review Appendix A and on-line help/documentation
Introduction

This module explains the operation of the analog-to-digital converter (ADC) and Event Manager. The ADC system consists of a 12-bit analog-to-digital converter with 16 analog input channels. The analog input channels have a range from 0 to 3 volts. Two input analog multiplexers are used, each supporting 8 analog input channels. Each multiplexer has its own dedicated sample and hold circuit. Therefore, sequential, as well as simultaneous sampling is supported. Also, the ADC system features programmable auto sequence conversions with 16 results registers. Start of conversion (SOC) can be performed by an external trigger, software, or an Event Manager event. The Event Manager system generates PWM waveforms using the timers and compare units. Also, the capture units and the quadrature encoder pulse circuit will be discussed. All devices of the F281x family have two event managers, EVA and EVB. These two event managers are identical to each other in terms of functionality. Register mapping and bit definitions are also identical, with the exception of naming conventions and register addresses. Therefore, for simplicity, only the functionality of EVA will be explained.

Learning Objectives

- Explain the operation of the Analog-to-Digital Converter
- Pulse Width Modulation (PWM) Review
- Generate PWM with the Event Manager:
  - General-Purpose Timer
  - Compare Units
- Explain other Event Manager functions:
  - Capture Units
  - Quadrature Encoder Pulse (QEP) Circuit

Note: Two identical Event Manager (EVA and EVB) modules are available. For simplicity, only EVA will be explained.
Module Topics

Control Peripherals............................................................................................................ 4-1

Analog-to-Digital Converter.............................................................................................. 4-3

Event Manager....................................................................................................................... 4-5
  PWM Waveforms............................................................................................................... 4-5
  General Purpose Timers.................................................................................................... 4-7
  Compare Units................................................................................................................... 4-8
  Capture Units.................................................................................................................... 4-11
  Quadrature Encoder Pulse............................................................................................... 4-13

Lab 4: Control Peripherals................................................................................................... 4-15
Analog-to-Digital Converter

ADC Module

- 12-bit resolution ADC core
- Sixteen analog inputs (range of 0 to 3V)
- Two analog input multiplexers
  - Up to 8 analog input channels each
- Two sample/hold units (for each input mux)
- Sequential and simultaneous sampling modes
- Autosequencing capability - up to 16 autoconversions
  - Two independent 8-state sequencers
    - “Dual-sequencer mode”
    - “Cascaded mode”
- Sixteen individually addressable result registers
- Multiple trigger sources for start-of-conversion
  - External trigger, S/W, and Event Manager events

ADC Module Block Diagram (Cascaded Mode)
**ADC Module Block Diagram (Dual-Sequencer mode)**

- Analog MUX
  - ADCINA0
  - ADCINA1
  - ADCINA7
  - ADCINB0
  - ADCINB1
  - ADCINB7
- S/H MUX
  - MUX A
  - S/H A
  - MUX B
  - S/H B
- 12-bit A/D Converter
- Sequencer Arbiter
- Result MUX
  - RESULT0
  - RESULT1
  - RESULT7
  - RESULT8
  - RESULT9
  - RESULT15

- Software EVA
- Ext Pin (ADC SOC)
- Start Sequence
- Trigger
- Autosequencer
  - MAX_CONV1
  - CHSEL.00 (state 0)
  - CHSEL.01 (state 1)
  - CHSEL.02 (state 2)
  - CHSEL.07 (state 7)
  - Start Sequence
  - Trigger
- Autosequencer
  - MAX_CONV2
  - CHSEL.08 (state 8)
  - CHSEL.09 (state 9)
  - CHSEL.10 (state 10)
  - CHSEL.15 (state 15)
  - Start Sequence
  - Trigger

**ADC Control Registers (file: Adc.c)**

- **ADCTRL1**
  - Module Reset
  - Continuous Run / Stop EOS
  - Sequencer Mode (cascaded / dual)
  - Acquisition Time Prescale (S/H)
- **ADCTRL2**
  - EV SOC; Start Conversion (s/w trigger); EV SOC mask bit
  - Reset SEQ
  - Interrupt Enable; Interrupt Mode: every EOS / every other EOS
- **ADCTRL3**
  - ADC Clock Prescale
  - Sampling Mode (sequential / simultaneous)
- **ADC_MAXCONV**
  - Maximum number of autoconversions
- **ADCCSELSEQx (x=1-4)**
  - Channel select sequencing
- **ADC_RESULTx (x=0-15)**
Event Manager

Event Manager Block Diagram (EVA)

PWM Waveforms

PWM Signal Representation

Original Signal

same areas (energy)

PWM representation PAM representation
Event Manager

Asymmetric PWM Waveform

- Period
- Compare
- Counter
- \( T_{\text{PWM}} \)
- \( T_{\text{PWM}} / T_{\text{cmp}} \) Pin (active high)
- Caused by Period match (toggle output in Asym mode only)
- Caused by Compare match

Symmetric PWM Waveform

- Period
- Compare
- Counter
- \( T_{\text{PWM}} \)
- \( T_{\text{PWM}} / T_{\text{CMP}} \) Pin (active high)
- Interrupts

General Purpose Timer

- Period
- Compare
- Counter
- \( T_{\text{PWM}} / T_{\text{CMP}} \) Pin

Full Compare Units

- PWM1
- PWM2
- PWM3
- PWM4
- PWM5
- PWM6
General Purpose Timers

General-Purpose Timers (EVA)

General-Purpose Timer Block Diagram (EVA)
GP Timer Control Registers (EVA) (file: Ev.c)

- **GPTCONA**
  - GP Timer Status (counting up / down)
  - ADC Start by Timer (underflow / period / compare)
  - Output Pin Condition (forced low / high, active low / high)
  - Output Enable

- **T1CON / T2CON**
  - Count Mode (stop/hold, continuous-up/down, continuous-up, directional-up/down)
  - Timer Clock Prescale; Timer Clock Source; Timer Enable
  - Compare Register Reload Condition (underflow, period, immediately)
  - Timer Compare Enable

- **EXTCONA**
  - Independent Compare Output Enable Mode (C24x/C28x)

### Compare Units

**Compare Units (EVA)**

```
EV Control Registers / Logic → Reset → PIE → 2 TCLKINA / TDIRA → ADC Start
GP Timer 1 Compare → Output Logic → T1CMP/T1PWM
           GP Timer 1
Compare Unit 1 → PWM Circuits → Output Logic → PWM1
Compare Unit 2 → PWM Circuits → Output Logic → PWM2
Compare Unit 3 → PWM Circuits → Output Logic → PWM3
           PWM4
 mux
GP Timer 2 Compare → Output Logic → T2CMP/T2PWM
           GP Timer 2
MUX
CAP1/QEP1 → CLK → QEP Circuit → DIR → CAP2/QEP2
           CAP3/QEPI1
```

C28x 1-Day Workshop - Control Peripherals
**Compare Units Block Diagram (EVA)**

![Block Diagram](compare_units_diagram.png)

**Motivation for Dead-Band**

- Gate Signals are Complementary PWM
  - Transistor gates turn on faster than they shut off
  - Short circuit if both gates are on at the same time!
**Dead-Band Functionality (EVA)**

Clock

PH$_x$

DT

DTPH$_x$

DTPH$_x$

dead time

Asymmetric PWM Example

HSPCLK

Prescaler

DBTCNA . 4 - 2

edge detect

ENA

reset

comparator

DT

4-bit period

DBTCNA . 11 - 8

DTPH$_x$

DTPH$_x$

**Compare Control Registers (EVA) (file: Ev.c)**

- **COMCONA**
  - Compare Enable; Compare Output Enable
  - Compare Reload Condition (underflow, period, immediately)
  - Action Control Register Reload Condition (underflow, period, immediately)
    - PDPINT Status
- **EXTCONA**
  - Independent Compare Output Enable Mode (C24x/C28x)
- **ACTRA**
  - Pin Action on Compare (forced low / high, active low / high)
- **DBTCONA**
  - Dead-Band Timer Enable
  - Dead-Band Timer Prescaler; Dead-Band Timer Period
Capture Units

Capture Units (EVA)

- Capture units timestamp transitions on capture input pins
- Three capture units (per event manager) - each associated with a capture input pin
Event Manager

Capture Units Block Diagram (EVA)

Can latch on:
- rising edge
- falling edge
- both

TTL Signal
min. valid width:
2 CPUCLK lo
2 CPUCLK hi

Capture Control Registers (EVA) (file: Ev.c)

- CAPCONA
  - Capture Reset
  - Capture Unit Enable for Compare; Enable for QEP
  - Edge Detection (rising, falling, both edges)
  - Timer Select (Timer 1 / 2)
  - ADC Start on CAP3INT flag

- CAPFIFOA
  - FIFO Status (empty, one entry, two entries, three entries attempted and first entry lost)

- EXTCONA
  - QEP Index Enable, Qualification Mode (C24x/C28x)

Event Manager
Quadrature Encoder Pulse

What is an Incremental Quadrature Encoder?

A digital (angular) position sensor

photo sensors spaced $\theta/4$ deg. apart

slots spaced $\theta$ deg. apart

light source (LED)

Incremental Optical Encoder

Quadrature Output from Photo Sensors
How is Position Determined from Quadrature Signals?

Position resolution is $\theta/4$ degrees.

$$(A,B) = \begin{cases} (00) & \text{increment counter} \\ (10) & \text{decrement counter} \end{cases}$$

Incremental Encoder Connections (EVA)

- Glueless interface between DSP and QEP sensor
- Index connection resets counter to zero
Lab 4: Control Peripherals

- **Objective**

The objective of this lab is to demonstrate the techniques discussed in this module and become familiar with the operation of the on-chip analog-to-digital converter and Event Manager. General-Purpose Timer 1 and Compare 1 will be setup to generate a 2 kHz, 25% duty cycle symmetric PWM waveform. The waveform will then be sampled with the on-chip analog-to-digital converter and displayed using the graphing feature of Code Composer Studio. The ADC has been setup to sample a single input channel at a 50 kHz sampling rate and store the conversion result in a buffer in the DSP memory. This buffer operates in a circular fashion, such that new conversion data continuously overwrites older results in the buffer.

Recall that there are two timebases associated with Event Manager “EVA”. The timer and its associated control registers are configured to produce a PWM waveform on the cZdsp™:

**General Purpose Timer #1 – PWM Generation**

- Used as a timebase for Compare 1 generation of PWM waveform

**General Purpose Timer #2 – ADC Conversion Trigger**

- Used as a timebase for triggering ADC samples (period match trigger SOC)

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

**Notes**
- General Purpose Timer 1 is used to generate a 2 kHz PWM waveform
- Program performs conversion on ADC channel A0 (ADCINA0 pin)
- General Purpose Timer 2 is used to auto-trigger the conversions at a 50 kHz sampling rate
- Data is continuously stored in a circular buffer
- Data is displayed using the graphing feature of Code Composer Studio
- ADC ISR will also toggle the eZdsp™ LED DS2 as a visual indication that it is running

➢ **Procedure**

**Project File**

1. A project named Lab4.pjt has been created for this lab. Open the project by clicking on Project ➔ Open… and look in C:\C28x1DAY\LABS\LAB4. All Build Options have been configured. The files used in this lab are:

   - Main_4.c
   - Lab.cdb
   - User_4.cmd
   - SysCtrl.c
   - DSP281x_GlobalVariableDefs.c
   - DefaultIsr_4.c
   - Gpio.c
   - Ev.c
   - Labcfg.cmd
   - DSP281x_Headers_BIOS.cmd
   - CodeStartBranch.asm

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

**Note:** _DO NOT_ make any changes to Gpio.c and Ev.c — _ONLY INSPECT_

2. Open and inspect Gpio.c by double clicking on the filename in the project window. Notice that the shared I/O pins in Group A have been set for the PWM1 function. Next, open and inspect Ev.c and see that the General-Purpose Timer 1 and Compare 1 has been setup to implement the PWM waveform as described in the objective for this lab. Notice the values used in the following registers: T1CON, T1CNT, T1PR, DBTCONA (set deadband units off), CMPR1, ACTRA, COMCONA. (Note that the last step for the timer setup enables Timer 1). See the global variable names and values that have been set using #define in the beginning of Ev.c file. Notice that GPTCONA has been initialized earlier in the code during Timer 2 setup. Close the inspected files.

**Build and Load**

3. Code Composer Studio can automatically load the output file after a successful build. On the menu bar click: Option ➔ Customize… and select the “Program Load Options” tab, check “Load Program After Build”, then click OK.
4. Click the “Build” button and watch the tools run in the build window. The output file should automatically load.

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

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

Run the Code – PWM Waveform

7. Open Main_4.c by double clicking on the filename in the project window. In Main_4.c place the cursor in the “main loop” section, right click on the mouse key and select Run To Cursor.

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

Note: For the next set of steps “VREFLO” must be connected to “GND”. Using a connector wire provided, connect “VREFLO” (pin # P9-18) to “GND” (pin # P9-17). Exercise care when connecting any wires, as the power to the eZdsp™ is on, and we do not want to damage the eZdsp™! Details of pin assignments can be found in Appendix A.

9. Using another connector wire provided, connect the PWM1 (pin # P8-9) to ADCIN0 (pin # P9-2) on the eZdsp™. Again, exercise care when connecting any wires, as the power to the eZdsp™ is on, and we do not want to damage the eZdsp™!

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

11. Open and setup a graph to plot a 50-point window of the ADC results buffer. Click: View ➔ Graph ➔ Time/Frequency... and set the following values:

<table>
<thead>
<tr>
<th>Start Address</th>
<th>AdcBuf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acquisition Buffer Size</td>
<td>50</td>
</tr>
<tr>
<td>Display Data Size</td>
<td>50</td>
</tr>
<tr>
<td>DSP Data Type</td>
<td>16-bit unsigned integer</td>
</tr>
<tr>
<td>Sampling Rate (Hz)</td>
<td>50000</td>
</tr>
<tr>
<td>Time Display Unit</td>
<td>μs</td>
</tr>
</tbody>
</table>

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

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

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

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

<table>
<thead>
<tr>
<th>Display Type</th>
<th>FFT Magnitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Address</td>
<td>AdcBuf</td>
</tr>
<tr>
<td>Acquisition Buffer Size</td>
<td>50</td>
</tr>
<tr>
<td>FFT Framesize</td>
<td>50</td>
</tr>
<tr>
<td>DSP Data Type</td>
<td>16-bit unsigned integer</td>
</tr>
<tr>
<td>Sampling Rate (Hz)</td>
<td>50000</td>
</tr>
</tbody>
</table>

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

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

**Using Real-time Emulation**

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

15. Reset the DSP, and enable real-time mode by selecting:

**Debug ➔ Real-time Mode**
16. A message box will appear. Select **YES** to enable debug events. This will set bit 1 (DGBM bit) of status register 1 (ST1) to a “0”. The DGBM is the debug enable mask bit. When the DGBM bit is set to “0”, memory and register values can be passed to the host processor for updating the debugger windows.

17. The memory and graph windows displaying *AdcBuf* should still be open. The connector wire between PWM1 (pin # P8-9) and ADCINA0 (pin # P9-2) should still be connected. In real-time mode, we would like to have our window continuously refresh. Click:

   View → Real-time Refresh Options...

   and check “Global Continuous Refresh”. Alternately, we could have right clicked on each window individually and selected “Continuous Refresh”.

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

18. Run the code and watch the windows update in real-time mode. Are the values updating as expected?

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

**Optional Exercise**

You might want to experiment with this code by changing some of the values or just modify the code. Try generating another waveform of a different frequency and duty cycle. Also, try to generate complementary pair PWM outputs. Next, try to generate additional simultaneous waveforms by using Compare 1, Compare 2, and Compare 3. Hint: don’t forget to setup the proper shared I/O pins, etc. (This optional exercise requires some further working knowledge of the event manager. Additionally, it may require more time than is allocated for this lab. Therefore, the student may want to try this after the class).

**End of Exercise**
This page is intentionally blank.
Introduction

This module discusses the basic features of using the DSP/BIOS real-time operating system. The configuration tool, scheduling threads, and real-time analysis tools will be explained.

Learning Objectives

- Introduction to DSP/BIOS
- DSP/BIOS Configuration Tool
- Scheduling DSP/BIOS Threads
- Real-Time Analysis Tools
- DSP/BIOS API Modules and Summary
# Module Topics

<table>
<thead>
<tr>
<th>DSP/BIOS Real-Time OS</th>
<th>5-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module Topics</td>
<td>5-2</td>
</tr>
<tr>
<td>Introduction to DSP/BIOS</td>
<td>5-3</td>
</tr>
<tr>
<td>DSP/BIOS Configuration Tool</td>
<td>5-4</td>
</tr>
<tr>
<td>Scheduling DSP/BIOS Threads</td>
<td>5-6</td>
</tr>
<tr>
<td>Real-Time Analysis Tools</td>
<td>5-10</td>
</tr>
<tr>
<td>DSP/BIOS API Modules and Summary</td>
<td>5-11</td>
</tr>
<tr>
<td>Lab 5: DSP/BIOS</td>
<td>5-12</td>
</tr>
</tbody>
</table>
Introduction to DSP/BIOS

What is DSP/BIOS?
- A full-featured, scalable real-time kernel
  - System configuration tools
  - Preemptive multi-threading scheduler
  - Real-time analysis tools

Why use DSP/BIOS?
- Helps manage complex system resources
- Integrated with Code Composer Studio IDE
  - Requires no runtime license fees
  - Fully supported by TI and is a key component of TI's eXpressDSP™ real-time software technology
- Uses minimal MIPS and memory (2-8Kw)

DSP/BIOS Terminology

- **thread**: a path of program execution
- **scheduler**: a program that manages threads
- **preemption**: the act of a higher priority thread interrupting a lower priority thread
- **post**: an event signal that often makes a thread “ready”
- **pend**: When a thread waits for an event post
- **semaphore**: a data object that tracks event occurrences (used by post and pend)
DSP/BIOS Configuration Tool

DSP/BIOS Configuration Tool (file .cdb)

- **System Setup Tools**
  Handles memory configuration (builds .cmd file), run-time support libraries, interrupt vectors, system setup and reset, etc.

- **Real-Time Analysis Tools**
  Allows application to run uninterrupted while displaying debug data

- **Real-Time Scheduler**
  Preemptive thread manager kernel

- **Real-Time I/O**
  Allows two way communication between threads or between target and PC host

**Memory Section Manager**

- **MEM manager** allows you to create memory area and place sections
- **To create a new memory area:**
  - Right-click on MEM and select Insert memory
  - Fill in base, length, space
Memory Section Manager Properties

- To place a section into a memory area:
  - Right-click on MEM and select Properties
  - Select the appropriate tab (e.g. Compiler)
  - Select the memory for each section

Hardware Interrupts
Scheduling DSP/BIOS Threads

Learning Objectives

- Introduction to DSP/BIOS
- DSP/BIOS Configuration Tool
- Scheduling DSP/BIOS Threads
- Real-Time Analysis Tools
- DSP/BIOS API Modules and Summary

DSP/BIOS Thread Types

<table>
<thead>
<tr>
<th>Priority</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HWI</td>
<td>Used to implement 'urgent' part of real-time event</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Triggered by hardware interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HWI priorities set by hardware</td>
</tr>
<tr>
<td></td>
<td>SWI</td>
<td>Use SWI to perform HWI 'follow-up' activity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SWI's are 'posted' by software</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Multiple SWIs at each of 15 priority levels</td>
</tr>
<tr>
<td></td>
<td>TSK</td>
<td>Use TSK to run different programs concurrently under separate contexts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TSK’s enabled by posting 'semaphore' (a signal)</td>
</tr>
<tr>
<td></td>
<td>IDL</td>
<td>Multiple IDL functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Runs as an infinite loop, like traditional while loop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All BIOS data transfers to host occur here</td>
</tr>
</tbody>
</table>

5 - 6
Enabling BIOS – Return from main()

```c
main
{
  ...
  // return to BIOS
}
```

- Must delete the endless while() loop
- `main()` returns to BIOS IDLE thread, allowing BIOS to schedule events, transfer info to host, etc.
- An endless while() loop in `main()` will not allow BIOS to activate

HWI Dispatcher Performs Save/Restore

- For non-BIOS code, we use the `interrupt` keyword to declare an ISR
  - tells the compiler to perform context save/restore

```c
interrupt void MyHwi(void)
{
}
```

- For DSP/BIOS code, the dispatcher will perform the save/restore
  - Remove the `interrupt` keyword from the `MyHwi()`
  - Check the “Use Dispatcher” box when you configure the interrupt vector in the DSP/BIOS config tools
Using Software Interrupts - SWI

- Make each algorithm an independent software interrupt
- SWI scheduling is handled by DSP/BIOS
  - HWI function triggered by hardware
  - SWI function triggered by software
    e.g. a call to SWI_post()
- Why use a SWI?
  - No limitation on number of SWIs, and priorities for SWIs are user-defined
  - SWI can be scheduled by hardware or software event(s)
  - Defer processing from HWI to SWI

DSP/BIOS

Function 1
Function 2

SWI Properties
Managing SWI Priority

- Drag and Drop SWIs to change priority
- Equal priority SWIs run in the order that they are posted

Priority Based Thread Scheduling

User sets the priority...BIOS does the scheduling

User sets the priority...BIOS does the scheduling
Real-Time Analysis Tools

**Built-in Real-Time Analysis Tools**

- Gather data on target (3-10 CPU cycles)
- Send data during BIOS IDL (100s of cycles)
- Format data on host (1000s of cycles)
- Data gathering does NOT stop target CPU

**Execution Graph**
- Software logic analyzer
- Debug event timing and priority

**CPU Load Graph**
- Shows amount of CPU horsepower being consumed

**Statistics View**
- Profile routines w/o halting the CPU

**Message LOG**
- Send debug msgs to host
- Doesn’t halt the DSP
- Deterministic, low DSP cycle count
- More efficient than traditional printf()

```c
LOG_printf(&trace, "AdcSwi_count = %u", AdcSwi_count++);
```
DSP/BIOS API Modules and Summary

DSP/BIOS - API Modules

Instrumentation/Real-Time Analysis
- LOG: Message log manager
- STS: Statistics accumulator manager
- TRC: Trace manager
- RTDX: Real-Time Data eXchange manager

Thread Types
- HWI: Hardware interrupt manager
- SWI: Software interrupt manager
- TSK: Multi-tasking manager
- IDL: Idle function & process loop manager

Clock and Periodic Functions
- CLK: System clock manager
- PRD: Periodic function manager

TSK Communication/Synchronization
- SEM: Semaphores manager
- MBX: Mailboxes manager
- LCK: Resource lock manager

Device-Independent Input/Output
- PIP: Data pipe manager
- HST: Host input/output manager
- SIO: Stream I/O manager
- DEV: Device driver interface

Memory and Low-Level Primitives
- MEM: Memory manager
- SYS: System services manager
- QUE: Queue manager
- ATM: Atomic functions
- GBL: Global setting manager

Summary and Benefits of DSP/BIOS
- Fast time to market
  - no need to develop or maintain a “home-brew” kernel
- Efficient debugging of real-time applications
  - Real-Time Analysis
- Create robust applications
  - industry proven kernel technology
- Reduce cost of software maintenance
  - code reuse and standardized software
- Standardized APIs
  - enable rapid migration across C28x TMS320 DSPs
- Small footprint (2-8Kw)
  - easily fits in limited memory space
- Set of library functions (scalable)
  - use only what is needed to minimize code and data size
- Full featured kernel (extensible)
  - allows additional OS functions in future
Lab 5: DSP/BIOS

Objective

The objective of this lab is to demonstrate the techniques discussed in this module and to become familiar with DSP/BIOS. In the previous lab, General-Purpose Timer 1 and Compare 1 from Event Manager A (EVA) were setup to generate a 2 kHz, 25% duty cycle symmetric PWM waveform. The waveform was then sampled with the on-chip analog-to-digital converter. In this lab the sampled waveform will be passed through an IQmath FIR filter and displayed using the graphing feature of Code Composer Studio. In this lab exercise, we are going to change the ADCINT_ISR HWI to a SWI. Then, we will replace the LED blink routine with a Periodic Function. Also, some features of the real-time analysis tools will be demonstrated.

Lab 5: DSP/BIOS

- ADCINT_ISR posts a SWI
- LED blink routine performed with a Periodic SWI

Procedure

Project File

1. A project named Lab5.pjt has been created for this lab. Open the project by clicking on Project → Open... and look in C:\C28x1DAY\LABS\LAB5. All Build Options have been configured. The files used in this lab are:

Note: Due to a bug in CCS and DSP/BIOS, the configuration file name (.cdb) and the project output (.out) and map (.map) names must be the same. Therefore, in this module the Build Options output and map file names will be lab.out and map.out, respectively.
Main_5.c
Lab.cfg.cmd
Lab.cdb
DSP281x_Headers_BIOS.cmd
User_5.cmd
CodeStartBranch.asm
SysCtrl.c
Gpio.c
DSP281x_GlobalVariableDefs.c
PieCtrl_4_5.c
DefaultIsr_5_6.c
Adc.c
Ev.c
Filter.c

Include IQmathLib.h

2. Open Lab.h which is located in the include folder in the project window. Uncomment the line that includes the IQmathLib.h header file. Next, in the Function Prototypes section, uncomment the function prototype for IQssfir(), the IQ math single-sample FIR filter function. Save the changes.

Add a SWI to main.c

Note: DO NOT make any changes to Main_5.c and DefaultISR_5_6.c – ONLY INSPECT

3. Open Main_5.c and notice that at the end of main() two new functions will be used in this module – AdcSwi() and LedBlink().

4. Open DefaultIsr_5_6.c, and notice that the interrupt key word for the ADCINT_ISR is not used. The interrupt keyword is not used when a HWI is under DSP/BIOS control. A HWI is under DSP/BIOS control when it uses any DSP/BIOS functionality, such as posting a SWI, or calling any DSP/BIOS function or macro.

5. In Main_5.c notice that the (in-line assembly) code used to enable global interrupts is not used. DSP/BIOS will enable global interrupts after main().

6. Again, in Main_5.c notice that the while() loop (the endless loop) is not used. When using DSP/BIOS, you must return from main(). In all DSP/BIOS programs, the main() function should contain all one-time user-defined initialization functions. DSP/BIOS will then take-over control of the software execution.

Post a SWI

7. In DefaultIsr_5_6.c the following SWI_post has been added to the ADCINT_ISR(), just after the structure used to acknowledge the PIE group:

   SWI_post(&ADC_swj);  // post a SWI

This post a SWI that will execute the ADC_swj() code you examined a few steps back in the lab. In other words, the ADC interrupt still executes the same code as before. However, some of that code is now in a posted SWI that DSP/BIOS will execute according to the specified scheduling priorities.
Add the SWI to the CDB File

8. Open the Lab.cdb file by left clicking on the plus sign (+) to the left of DSP/BIOS Config and double clicking on the Lab.cdb file. In the configuration file Lab.cdb we need to add and setup the AdcSwi() SWI. Click on the plus sign (+) to the left of Scheduling and again on the plus sign (+) to the left of SWI – Software Interrupt Manager.

9. Right click on SWI – Software Interrupt Manager and select Insert SWI. SWI0 will be added. Right-click on it, and rename it to ADC_swi. This is just an arbitrary name. We want to differentiate the AdcSwi() function itself (which is nothing but an ordinary C function) from the DSP/BIOS SWI object which we are calling ADC_swi.

10. Select the Properties for ADC_swi and type _AdcSwi (with a leading underscore) in the function field. Click OK. This tells DSP/BIOS that it should run the function AdcSwi() when it executes the ADC_swi SWI.

11. We need to have the PIE for the ADC interrupt use the dispatcher. The dispatcher will automatically perform the context save and restore, and allow the DSP/BIOS scheduler to have insight into the ISR. The ADC interrupt is located at PIE_INT1_6.

   Click on the plus sign (+) to the left of HWI – Hardware Interrupt Service Routine Manager. Click the plus sign (+) to the left of PIE Interrupts. Locate the interrupt location for the ADC: PIE_INT1_6. Right click, select Properties, and select the Dispatcher tab.

   Now check the “Use Dispatcher” box and select OK. Close the configuration file and click YES to save changes.

Build, Load, and Reset

12. Click the “Build” button to rebuild and load the project.

13. Reset the DSP.

Run the Code – AdcSwi()

14. We will be running our code in real-time mode, and need to have our window continuously refresh. Enable Real-time Mode and be sure that the Global Continuous Refresh option has been checked.

**Note:** For the next step, check to be sure that the jumper wire connecting “VREFLO” (pin # P9-18) to “GND” (pin # P9-17) and the wire connecting PWM1 (pin # P8-9) to ADCIN0 (pin # P9-2) are still in place on the eZdsp™.

15. Run the code in real-time mode watch the window update. Verify that the ADC result buffer contains updated values.
16. Open and setup a dual time graph to plot a 50-point window of the filtered and unfiltered ADC results buffer. Click: View ➔ Graph ➔ Time/Frequency... and set the following values:

<table>
<thead>
<tr>
<th>Display Type</th>
<th>Dual Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Address – upper display</td>
<td>AdcBufFiltered</td>
</tr>
<tr>
<td>Start Address – lower display</td>
<td>AdcBuf</td>
</tr>
<tr>
<td>Acquisition Buffer Size</td>
<td>50</td>
</tr>
<tr>
<td>Display Data Size</td>
<td>50</td>
</tr>
<tr>
<td>DSP Data Type</td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td>Q-value</td>
<td>24</td>
</tr>
<tr>
<td>Sampling Rate (Hz)</td>
<td>50000</td>
</tr>
<tr>
<td>Time Display Unit</td>
<td>µs</td>
</tr>
</tbody>
</table>

Select OK to save the graph options.

17. The graphical display should show the generated IQmath FIR filtered 2 kHz, 25% duty cycle symmetric PWM waveform in the upper display and the unfiltered waveform generated in the previous lab exercise in the lower display. Notice the shape and phase differences between the waveform plots (the filtered curve has rounded edges, and lags the unfiltered plot by several samples). The amplitudes of both plots should run from 0 to 3.0. The results should be the same as the previous lab, but we are using a SWI rather than a HWI.

18. Open and setup two (2) frequency domain plots – one for the filtered and another for the unfiltered ADC results buffer. Click: View ➔ Graph ➔ Time/Frequency... and set the following values:
<table>
<thead>
<tr>
<th></th>
<th>GRAPH #1</th>
<th>GRAPH #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Type</td>
<td>FFT Magnitude</td>
<td>FFT Magnitude</td>
</tr>
<tr>
<td>Start Address</td>
<td>AdcBuf</td>
<td>AdcBufFiltered</td>
</tr>
<tr>
<td>Acquisition Buffer Size</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Display Data Size</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>DSP Data Type</td>
<td>32-bit signed integer</td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td>Q-value</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Sampling Rate (Hz)</td>
<td>50000</td>
<td>50000</td>
</tr>
<tr>
<td>Time Display Unit</td>
<td>µs</td>
<td>µs</td>
</tr>
</tbody>
</table>

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

19. The graphical displays should show the frequency components of the filtered and unfiltered 2 kHz, 25% duty cycle symmetric PWM waveforms. Notice that the higher frequency components are reduced using the Low-Pass IQmath FIR filter in the filtered graph as compared to the unfiltered graph.

20. Fully halt the DSP (real-time mode) by using **Debug → Halt** and then unchecking the “Real-time mode”.

**Add a Periodic Function**

Recall that an instruction was used in the ADCINT_ISR to toggle the LED on the eZdsp™. This instruction has been moved into a periodic function that will toggle the LED at the same rate.

21. Open **Main_5.c** and notice the instruction used to toggle the LED to the LedBlink() function:

   ```
   GpioDataRegs.GPFTOGGLE.bit.GPIOF14 = 1; // Toggle the pin
   ```

22. In the configuration file **Lab.cdb** we need to add and setup the LedBlink_PRD. Open Lab.cdb and click on the plus sign (+) to the left of Scheduling. Right click on PRD - Periodic Function Manger and select Insert PRD. PRD0 will be added. Right-click on it and rename it to **LedBlink_PRD**.

23. Select the **Properties** for LedBlink_PRD and type _LedBlink (with a leading underscore) in the function field. This tells DSP/BIOS to run the LedBlink() function when it executed the LedBlink_PRD periodic function object.

   Next, in the period (ticks) field type **500**. The default DSP/BIOS system timer increments every 1 miliseconds, so what we are doing is telling the DSP/BIOS scheduler to schedule the LedBlink() function to execute every 500 milliseconds. A PRD object is
just a special type of SWI which gets scheduled periodically and runs in the context of the SWI level at a specified SWI priority. Click OK. Close the configuration file and click YES to save changes.

Build and Load

24. Click the “Build” button to rebuild and load the project.

25. Reset the DSP.

Run the Code – LedBlink

26. Run the code and check to see if the LED on the eZdsp™ is blinking. When done, halt the code. If you would like, experiment with different period (tick) values and notice that the blink rate changes.

DSP/BIOS – Real-time Analysis

The DSP/BIOS analysis tools complement the CCS environment by enabling real-time program analysis of a DSP/BIOS application. You can visually monitor a DSP application as it runs with essentially no impact on the application’s real-time performance. In CCS, the DSP/BIOS analysis tools are found on the DSP/BIOS menu. Unlike traditional debugging, which is external to the executing program, DSP/BIOS program analysis requires that the target program be instrumented with analysis code. By using DSP/BIOS APIs and objects, developers automatically instrument the target for capturing and uploading real-time information to CCS using these tools.

27. Open the CPU load graph. On the menu bar click:

DSP/BIOS ➔ CPU Load Graph

The CPU load graph displays the percentage of available CPU computing horsepower that the application is consuming. The CPU may be running ISRs, software interrupts, periodic functions, performing I/O with the host, or running any user routine. When the CPU is not executing user code, it will be idle (in the DSP/BIOS idle thread). You will notice that we are consuming about 50% of the CPU with our lab code.

28. Next, open the execution graph. On the menu bar click:

DSP/BIOS ➔ Execution Graph

The execution graph is a special graph used to display information about different threads in the system and when they occur relative to the other events. This graph is not based on time, but the activity of events (i.e. when an event happens, such as a SWI or periodic function begins execution). You can enable or disable logging for each of the object types using the RTA Control Panel (DSP/BIOS ➔ RTA Control Panel). Note that the execution graph simply records DSP/BIOS CLK events along with other system events (the DSP/BIOS clock periodically triggers the DSP/BIOS scheduler). As a result, the time scale on the execution graph is not linear.
29. Halt the DSP.

30. In the next few steps the Log Event Manager will be setup to capture an event in real-time while the program executes. We will be using `Log_printf()` to write to a log buffer. The `LOG_printf()` function is a very efficient means of sending a message from the code to the CCS display. Unlike an ordinary C-language printf(), which can consume several hundred DSP cycles to format the data on the DSP before transmission to the CCS host PC, a log_printf() transmits the raw data to the host. The host then formats the data and displays it in CCS. This consumes only 10’s of cycles rather than 100’s of cycles.

In `Main_5.c` just after the static local variable declaration in `AdcSwi()`, `uncomment` the following code used with `LOG_printf`:

```c
static Uint32 AdcSwi_count=0;          // used for LOG_printf
/*** Using LOG_printf() to write to a log buffer ***/
    LOG_printf(&trace, "AdcSwi_count = %u", AdcSwi_count++);
```

31. In the configuration file `Lab.cdb` we need to add and setup the trace buffer. Open `Lab.cdb` and click on the plus sign (+) to the left of `Instrumentation` and again on the plus sign (+) to the left of `LOG – Event Log Manager`.

32. Right click on `LOG – Event Log Manager` and select `Insert LOG`. `LOG0` will be added. Right-click on it and rename it to `trace`.

33. Select the `Properties` for `trace` and set the logtype to `circular` and the datatype to `printf`. Click `OK`.

34. Since the configuration file was modified, we need to rebuild the project. Click the “Build” button.

35. Reset the DSP.

36. Open the `Message Log` by clicking:

    DSP/BIOS → Message Log

    The message log dialog box is displaying the number of times (count value) that `AdcSwi()` has executed.

37. Run the DSP.

38. Observe the operation of the various windows that are open, and the information that they are conveying in real-time.
Note: In this module only the basic features of DSP/BIOS and the real-time analysis tools have been used. For more information and details, please refer to the DSP/BIOS user’s manuals and other DSP/BIOS related training.

End of Exercise
This page is intentionally blank.
Flash Programming

Introduction

This module will explain the details to program the flash memory using the Code Composer Studio plug-in. Other programming utilities will be discussed. Additionally, the code security module and CSM passwords will be covered.

Learning Objectives

- Explain concepts and steps required to program the flash memory
- Discuss flash programming utilities
- Show flash programming CCS plug-in
- Explain code security module and CSM passwords
Module Topics

Flash Programming .................................................................................................................. 6-1

Module Topics ........................................................................................................................ 6-2

Flash Programming Basics ................................................................................................. 6-3

Programming Utilities and CCS Plug-in ............................................................................. 6-4

Code Security Module and Passwords .................................................................................. 6-5

Lab 6: Flash Programming .................................................................................................... 6-6
Flash Programming Basics

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

---

Sequence of steps for Flash programming:

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

- Minimum Erase size is a sector (8Kw or 16Kw)
- Minimum Program size is a bit!
- Important not to lose power during erase step: If CSM passwords happen to be all zeros, the CSM will be permanently locked!
- Chance of this happening is quite small! (Erase step is performed sector by sector)
Programming Utilities and CCS Plug-in

Flash Programming Utilities

- Code Composer Studio Plug-in (uses JTAG) *
- Serial Flash loader from TI (uses SCI boot) *
- Gang Programmers (use GPIO boot)
  - BP Micro programmer
  - Data I/O programmer
- Build your own custom utility
  - Use a different ROM bootloader method than SCI
  - Embed flash programming into your application
  - Flash API algorithms provided by TI

* Available from TI web at www.ti.com

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

Code Security Module (CSM)

- Access to the following on-chip memory is restricted:
  
<table>
<thead>
<tr>
<th>Address</th>
<th>Memory Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00 8000</td>
<td>LO SARAM (4K)</td>
</tr>
<tr>
<td>0x00 9000</td>
<td>L1 SARAM (4K)</td>
</tr>
<tr>
<td>0x00 A000</td>
<td>reserved</td>
</tr>
<tr>
<td>0x3D 7800</td>
<td>OTP (1K)</td>
</tr>
<tr>
<td>0x3D 7C00</td>
<td>reserved</td>
</tr>
<tr>
<td>0x3D 8000</td>
<td>FLASH (128K)</td>
</tr>
</tbody>
</table>

- Data reads and writes from restricted memory are only allowed for code running from restricted memory

- All other data read/write accesses are blocked:
  JTAG emulator/debugger, ROM bootloader, code running in external memory or unrestricted internal memory

CSM Password

- 128-bit user defined password is stored in Flash
- 128-bits = $2^{128} = 3.4 \times 10^{38}$ possible passwords
- To try 1 password every 2 cycles at 150 MHz, it would take at least $1.4 \times 10^{23}$ years to try all possible combinations!
Lab 6: Flash Programming

Objective

The objective of this lab is to demonstrate the techniques discussed in this module and program the on-chip flash memory. The TMS320F2812 device has been designed for standalone operation in an embedded system. Using the on-chip flash eliminates the need for external non-volatile memory or a host processor from which to bootload. In this lab, the steps required to properly configure the software for execution from internal flash memory will be covered.

Lab 6: Flash Programming

- System Programmed into Flash Memory
- Using of CCS Flash Plug-in
- **DO NOT PROGRAM PASSWORDS**

Procedure

Project File

1. A project named Lab6.pjt has been created for this lab. Open the project by clicking on Project → Open... and look in C:\C28x1DAY\LABS\LAB6. All Build Options have been configured. The files used in this lab are:

Note: Due to a bug in CCS and DSP/BIOS, the configuration file name (.cdb) and the project output (.out) and map (.map) names must be the same. Therefore, in this module the Build Options output and map file names will be lab.out and map.out, respectively.
Link Initialized Sections to Flash

Initialized sections, such as code and constants, must contain valid values at device power-up. For a stand-alone embedded system with the F2812 device, these initialized sections must be linked to the on-chip flash memory. Note that a stand-alone embedded system must operate without an emulator or debugger in use, and no host processor is used to perform bootloading.

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

2. The RUN address of the following sections needs to run from flash. The memory section manager in the DSP/BIOS configuration tool (Lab.cdb) has been set to link the following sections to on-chip flash memory:

<table>
<thead>
<tr>
<th>DSP/BIOS Data tab</th>
<th>DSP/BIOS Code tab</th>
<th>Compiler Sections tab</th>
</tr>
</thead>
<tbody>
<tr>
<td>.gblinit</td>
<td>.bios</td>
<td>.text</td>
</tr>
<tr>
<td></td>
<td>.sysinit</td>
<td>.switch</td>
</tr>
<tr>
<td></td>
<td>.hwi</td>
<td>.cinit</td>
</tr>
<tr>
<td></td>
<td>.rtdx_text</td>
<td>.pinit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.econst /.const</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.data</td>
</tr>
</tbody>
</table>

3. The LOAD address of the following sections needs to load to flash. Again the memory section manager in the DSP/BIOS configuration tool (Lab.cdb) has been set to specify the load addresses. To view these setting, select the Load Address tab. Notice that the “Specify Separate Load Addresses” box has been checked and all the entries are set to the flash memory block.
Copying .hwi_vec Section from Flash to RAM

The DSP/BIOS .hwi_vec section contains the interrupt vectors. This section must be loaded to flash (load address) but run from RAM (run address). The code that performs this copy is located in InitPieCtrl(). The DSP/BIOS configuration tool generates global symbols that can be accessed by code in order to determine the load address, run address, and length of the .hwi_vec section. The C-compiler runtime support library contains a memory copy function called memcpy() which will be used to perform the copy.

4. Open and inspect InitPieCtrl() (in PieCtrl_6.c file). Notice the memcpy() function and the symbols used to initialize (copy) the .hwi_vec section.

Copying the .trcdata Section from Flash to RAM

The DSP/BIOS .trcdata section is used by CCS and DSP/BIOS for certain real-time debugging features. This section must be loaded to flash (load address) but run from RAM (run address). The DSP/BIOS configuration tool generates global symbols that can be accessed by code in order to determine the load address, run address, and length of the .trcdata section. The memory copy function memcpy() will again be used to perform the copy.

The copying of .trcdata must be performed prior to main(). This is because DSP/BIOS modifies the contents of .trcdata during DSP/BIOS initialization, which also occurs prior to main(). The DSP/BIOS configuration tool provides a user initialization function which will be used to perform the .trcdata section copy prior to both main() and DSP/BIOS initialization.

5. Open the DSP/BIOS configuration file (Lab.cdb) and select the Properties for the Global Settings. Notice the box “Call User Init Function” has been checked and the UserInit() function name with a leading underscore _UserInit is entered. This will cause the function UserInit() to execute prior to main().

6. Open and inspect the file Main_6.c. Notice that the function UserInit() is used to copy the .trcdata section from its load address to its run address before main().

Initializing the Flash Control Registers

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

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

8. Since the DSP/BIOS configuration tool does not know about user defined sections, the “secureRamFuncs” section will be linked using the user linker command file User_6.cmd. Open and inspect User_6.cmd. The “secureRamFuncs” will load to flash (load address) but will run from L1SARAM (run address). Also notice that the linker has been asked to generate symbols for the load start, load end, and run start addresses.
While not a requirement from a DSP hardware perspective (since the C28x DSP has a unified memory architecture), Code Composer Studio generally prefers code to be linked to program space (and data to be linked to data space). Therefore, notice that for the L1SARAM memory we are linking “secureRamFuncs” to, we are specifying “PAGE = 0” (which is program space).

9. Open and inspect Main_6.c. Notice that the memory copy function memcpy() is being used to copy the section “secureRamFuncs, which contains the initialization function for the flash control registers.

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

   ```c
   InitFlash();
   ```

   at the required spot in main().

**Code Security Module and Passwords**

The CSM module provides protection against unwanted copying (i.e. pirating!) of your code from flash, OTP memory, and the L0 and L1 RAM blocks. The CSM uses a 128-bit password made up of 8 individual 16-bit words. They are located in flash at addresses 0x3F7FF8 to 0x3F7FFF. During this lab, dummy passwords of 0xFFFF will be used – therefore only dummy reads of the password locations are needed to unsecure the CSM. **DO NOT PROGRAM ANY REAL PASSWORDS INTO THE DEVICE.** After development, real passwords are typically placed in the password locations to protect your code. We will not be using real passwords in the workshop.

The CSM module also requires programming values of 0x0000 into flash addresses 0x3F7F80 through 0x3F7FF5 in order to properly secure the CSM. Both tasks will be accomplished using a simple assembly language program Passwords.asm.

11. Open and inspect Passwords.asm. This file specifies the desired password values **(DO NOT CHANGE THE VALUES FROM 0xFFFF)** and places them in an initialized section named “passwords”. It also creates an initialized section named “csm_rsbd” which contains all 0x0000 values for locations 0x3F7F80 to 0x3F7FF5 (length of 0x76).

12. Open User_6.cmd and notice that the initialized sections for “passwords” and “csm_rsbd” are linked to memories named PASSWORDS and CSM_RSVD, respectively. The DSP/BIOS configuration tool (Lab.cdb) defines memory blocks for PASSWORDS and CSM_RSVD.

**Executing from Flash after Reset**

The F2812 device contains a ROM bootloader that will transfer code execution to the flash after reset. When the boot mode selection pins are set for “Jump to Flash” mode, the bootloader will branch to the instruction located at address 0x3F7F6 in the flash. An instruction that branches to the beginning of your program needs to be placed at this address. Note that the CSM
passwords begin at address 0x3F7FF8. There are exactly two words available to hold this branch instruction, and not coincidentally, a long branch instruction “LB” in assembly code occupies exactly two words. Generally, the branch instruction will branch to the start of the C-environment initialization routine located in the C-compiler runtime support library. The entry symbol for this routine is _c_int00. Recall that C code cannot be executed until this setup routine is run. Therefore, assembly code must be used for the branch. We are using the assembly code file named CodeStartBranch.asm.

13. Open and inspect CodeStartBranch.asm. This file creates an initialized section named “codestart” that contains a long branch to the C-environment setup routine. This section has been placed in memory using the DSP/BIOS configuration tool and the memory space is named BEGIN_FLASH.

14. In the earlier lab exercises, the section “codestart” was directed to the memory named BEGIN_H0. Open and inspect User_6.cmd and notice that the section “codestart” will now be directed to BEGIN_FLASH.

15. The eZdsp™ board needs to be configured for “Jump to Flash” bootmode. Move jumper JP7 to position 1-2 to accomplish this. This jumper controls the pullup/down resistor on the GPIOF4 pin, which is one of the pins sampled by the bootloader to determine the bootmode. GPIOF4 alone is sufficient to configure “Jump to Flash” bootmode (see the TMS320F28x DSP Boot ROM Reference Guide, and also the eZdsp F2812 Technical Reference, for more information).

Build – Lab.out

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

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

CCS Flash Plug-in

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

Tools → F28xx On-Chip Flash Programmer

19. Notice that the eZdsp™ board uses a 30 MHz oscillator (located on the board near LEDs DS1 and DS2). Confirm the “Clock Configuration” in the upper left corner has the OSCCLK set to 30 MHz and the PLLCR value is set to 10. Recall that the PLL is divided by two, which gives a SYSCLKOUT of 150 MHz.

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

21. We will not be using the plug-in to program the “Code Security Password”. Do not modify the Code Security Password fields.
22. In the “Operation” block, notice that the “COFF file to Program/Verify” field automatically defaults to the current .out file. Check to be sure that “Erase, Program, Verify” is selected. We will be using the default wait states, as shown on the slide in this module.

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

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

Running the Code – Using CCS

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

   File → Load Symbols → Load Symbols Only...

   and select Lab.out in the Debug folder.

26. Reset the DSP. The program counter should now be at 0x3FFC00, which is the start of the bootloader in the Boot ROM.

27. Single-Step through the bootloader code until you arrive at the beginning of the codestart section in the CodeStartBranch.asm file. (Be patient, it will take about 55 single-steps). Notice that we have placed some code in CodeStartBranch.asm to give an option to first disable the watchdog, if selected.

28. Step a few more times until you reach the start of the C-compiler initialization routine at the symbol _c_int00.

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

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

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


32. Disconnect the emulator from the eZdsp™ board.

33. Remove the power from the board.
34. Re-connect the power to the board.

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

**End of Exercise**

**Lab 6 Reference: BIOS Startup Sequence from Flash Memory**

**BIOS Startup Sequence from Flash Memory**

```plaintext
RESET

0x3D 8000
FLASH (128K)

0x3F 7FF6
LB

0x3D 8000
H0 SARAM (8K)

0x3F 8000
Boot ROM (4K)

0x3F F000
Boot Code
0x3F FC00
(SCAN GPIO)

0x3F FFC0
BROM vector (32)

0x3F FC00
_c_int00

BIOS code Sections

_c_int00

BIOS_reset

BIOS_init

main

BIOS_start

IDL_run

“rts2800_ml.lib”

“user” code sections

main

{......

return;

}```
Introduction

This module provides an overview to the recommended next steps for developing your design. Also, a list of where to find more information will be covered.

Learning Objectives

| Comprehensive Tools and Support Set Standard in Ease of Use for Developers |
|-----------------|---------------------------------------------------------------------------|
| **Tools**       | F2812 eZdsp™ developer's kit from Spectrum Digital                       |
|                 | Code Composer Studio™ IDE for C2000™                                     |
| **Software**    | Application specific libraries                                          |
|                 | Math functions                                                           |
|                 | Communications drivers                                                  |
|                 | Pre-bundled system solutions                                             |
| **Training and Support** | Control developers seminar                                         |
|                 | DMC workshop                                                             |
|                 | One-day technical introduction to C28x™                                 |
|                 | Multi-day get started developing C28x™ workshop                          |
| **High-Performance Analog** | Numerous data converter and power management products designed for motor control |
| **Third Party Network** | Development boards and emulation tools                                   |
|                 | Large consultant network                                                 |
|                 | Increasing range of application software                                 |
Module Topics

The Next Step..................................................................................................................... 7-1

Module Topics..................................................................................................................... 7-2

Development Support......................................................................................................... 7-3
Development Support

**Recommended Next Step: F2812 eZdsp™**

You now have the F2812 eZdsp™, so...

Start Today!

The F2812 eZdsp™ integrates the essential key features to allow users to understand the TMS320F2812 characteristics.

**Hardware Features**
- TMS320F2812 Digital Signal Processor
- 18 Kwords RAM
- 128 Kwords on chip Flash ROM
- 64 Kwords on board RAM
- Expansion connectors
- Onboard embedded IEEE 1149.1 JTAG controller
- 5 Volt only operation with supplied adapter
- Onboard IEEE 1149.1 JTAG emulation

**Software**
- Includes TI’s C28xx Code Composer Studio

**Studio Debug Tools**
- Includes Flash programming utilities from Spectrum Digital

**Socketed Version Available**

---

**Recommended Next Step: Multi-day Training Course**

**TMS320C28x Workshop Outline**
- Architectural Overview
- Programming Development Environment
- Peripheral Register Header Files
- Reset and Interrupts
- System Initialization
- Analog-to-Digital Converter
- Event Manager
- Numerical Concepts and IQmath
- Using DSP/BIOS
- System Design
- Communications
- Support Resources

**In-depth TMS320F2812 Design and Peripheral Training**
# C28x Signal Processing Libraries

<table>
<thead>
<tr>
<th>Signal Processing Libraries</th>
<th>Literature #</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACI3_3: Simulated Indirect FOC of ACI Motor</td>
<td>SPRC077</td>
</tr>
<tr>
<td>ACI3_4: Real Direct FOC of ACI Motor</td>
<td>SPRC079</td>
</tr>
<tr>
<td>ACI3_4: Simulated Direct FOC of ACI Motor</td>
<td>SPRC078</td>
</tr>
<tr>
<td>ACI3_1: Three-Phase ACI Control with Constant V/Hz</td>
<td>SPRC130</td>
</tr>
<tr>
<td>PMSM3_1: Three-Phase Sensored FOC</td>
<td>SPRC129</td>
</tr>
<tr>
<td>PMSM3_2: Three-Phase Sensorless FOC</td>
<td>SPRC128</td>
</tr>
<tr>
<td>Digital Motor Control Library</td>
<td>SPRC080</td>
</tr>
<tr>
<td>DSP Fast Fourier Transform (FFT) Library</td>
<td>SPRC081</td>
</tr>
<tr>
<td>DSP Filter Library</td>
<td>SPRC082</td>
</tr>
<tr>
<td>DSP Fixed-Point Math Library</td>
<td>SPRC085</td>
</tr>
<tr>
<td>DSP IQ Math Library</td>
<td>SPRC087</td>
</tr>
<tr>
<td>DSP Signal Generator Library</td>
<td>SPRC083</td>
</tr>
<tr>
<td>DSP Software Test Bench (STB) Library</td>
<td>SPRC084</td>
</tr>
<tr>
<td>C281x C/C++ Header Files and Peripheral Examples</td>
<td>SPRC097</td>
</tr>
</tbody>
</table>

Available from TI DSP Website ⇒ http://www.dspvillage.ti.com

---

# Large Third Party Network

- TECNO SOFT
- FSI
- NFO
- BPM5
- PENSA
- International Rectifier
- Hyperperception
- SOFTRONICS
- Oztek Corp.
- SCHMIDHAUSER AG
- Visual Solutions
- Drivetec, Inc.
- Spectrum Digital
New TMS320F280x Devices!

- **High Performance CPU Core**
  - 100 MIPS Performance
  - Single Cycle 32x32 MAC (or dual 16x16)
  - Single Cycle Read-Modify-Write
  - Fast Interrupt Response
  - F24x/LF240xA Source Code Compatible
- **Memory**
  - Flash: 16Kw ~ 64Kw (execute at 85 MIPS)
  - RAM: 6Kw ~ 18Kw
- **Event Manager**
  - Increased number of timers
  - Flexible PWM channels
  - Full Spec TBD (below is preliminary and may change)
    - Dedicated CAP/QEP – position DMC
  - One Timer for 6 PWM - synchronized
- **ADC**
  - High speed throughput (6.25 MSPS)
  - Simultaneous sampling or sequencing sampling modes
  - 12 bit 16-channel, multiplexed inputs
- **Peripherals**
  - Up to 4-channels SPI
  - Up to 2 SCIs (UART)
  - PC
  - Up to 2 CAN ports

Options to Meet Developer’s Needs...

<table>
<thead>
<tr>
<th></th>
<th>F2801</th>
<th>F2806</th>
<th>F2808</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 MIPS</td>
<td>100 MIPS</td>
<td>100 MIPS</td>
<td></td>
</tr>
<tr>
<td>16Kw Flash</td>
<td>32Kw Flash</td>
<td></td>
<td>64Kw Dual Flash</td>
</tr>
<tr>
<td>6Kw RAM</td>
<td>10Kw RAM</td>
<td>18Kw RAM</td>
<td></td>
</tr>
<tr>
<td>4Kw ROM</td>
<td>4Kw ROM</td>
<td>4Kw ROM</td>
<td></td>
</tr>
<tr>
<td>Code security module</td>
<td>Code security module</td>
<td>Code security module</td>
<td></td>
</tr>
<tr>
<td>100-pin LQFP</td>
<td>100-pin LQFP</td>
<td>100-pin LQFP</td>
<td></td>
</tr>
<tr>
<td>100-pin µ*BGA</td>
<td>100-pin µ*BGA</td>
<td>100-pin µ*BGA</td>
<td></td>
</tr>
<tr>
<td>2-channel SPI</td>
<td>4-channel SPI</td>
<td>4-channel SPI</td>
<td></td>
</tr>
<tr>
<td>1 SCI (UART)</td>
<td>2 SCIs (UART)</td>
<td>2 SCIs (UART)</td>
<td></td>
</tr>
<tr>
<td>I²C</td>
<td>I²C</td>
<td>I²C</td>
<td></td>
</tr>
<tr>
<td>CAN</td>
<td>CAN</td>
<td>Dual CAN</td>
<td></td>
</tr>
<tr>
<td>12-bit ADC, 16 channels</td>
<td>12-bit ADC, 16 channels</td>
<td>12-bit ADC, 16 channels</td>
<td></td>
</tr>
</tbody>
</table>
Need a Micro-Controller, Consider the MSP430!

**Ultra-low Power**
- 0.1uA power down
- 0.8uA standby mode
- 250uA / 1MIPS @ 3V
- < 6us clock start-up
- < 50nA port leakage
- Zero-power BOR

**High-Performance**
- Modern 16-bit RISC
- SoC integration

**Analog Integration**
- SVS, Comparator, BOR, ADC, DAC, Temp Sensor

---

Customers Are Using C2000™ Products For …

<table>
<thead>
<tr>
<th><strong>Motor Control</strong></th>
<th><strong>Digital Power</strong></th>
<th><strong>Optical Networking</strong></th>
<th><strong>Others</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Active suspension</td>
<td>Battery charging</td>
<td>TEC control</td>
<td>Adaptive cruise control</td>
</tr>
<tr>
<td>Air conditioners</td>
<td>Frequency converters</td>
<td>Optical switch control</td>
<td>Airbag control</td>
</tr>
<tr>
<td>Aircraft A/C</td>
<td>Fuel cell control</td>
<td>Tunable laser control</td>
<td>Antitheft systems</td>
</tr>
<tr>
<td>Bonding machines</td>
<td>Server power control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Building automation</td>
<td>Solar energy control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cameras</td>
<td>Uninterr. power supplies</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Car A/C</td>
<td>Radar control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD and DVD drives</td>
<td>Reactor monitoring</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Check readers</td>
<td>RF ID systems</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNC control</td>
<td>Spectrum analyzers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compressors</td>
<td>Telecom switches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Copiers</td>
<td>Tire pressure sensing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Door openers</td>
<td>Ultrasonic scanners</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elevator</td>
<td>Washing machines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>motor control</td>
<td>Washing machines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoders</td>
<td>Windmill control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fan control</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Food mixers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fork lifts</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fuel pumps</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Golf trainers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hair removers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heart/lung machines</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Human transporters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Industrial drives</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inverters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Magnetic bearings</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mass flow control</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medical pumps</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Missile control</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Potters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Postal sorters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power assisted steering</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power drills</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power generators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Printers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Refrigerators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robot control</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sewing machines</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ship propulsion control</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stepper motor control</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Textile machines</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Toy trains</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Treadmills</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vacuum cleaners</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vibration control</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wafer testers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Washing machines</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windmill control</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For More Information...

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

USA - Product Information Center (PIC)
Phone: 972-644-5580
Email: sc-infomaster@ti.com
   » Information and support for all TI Semiconductor products/tools
   » Submit suggestions and errata for tools, silicon and documents

Other Resources
Software Registration/Upgrades: 972-293-5050
Hardware Repair/Upgrades: 281-274-2285
Enroll in Technical Training: www.ti.com/sc/training
   (choose Multi-Day Workshops)

European Product Information Center (EPIC)
Web: http://www-k.ext.ti.com/sc/technical_support/pic/euro.htm

<table>
<thead>
<tr>
<th>Phone: Language</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belgium (English)</td>
<td>+32 (0) 27 45 55 32</td>
</tr>
<tr>
<td>France</td>
<td>+33 (0) 1 30 70 11 64</td>
</tr>
<tr>
<td>Germany</td>
<td>+49 (0) 8161 80 33 11</td>
</tr>
<tr>
<td>Israel (English)</td>
<td>1800 949 0107 (free phone)</td>
</tr>
<tr>
<td>Italy</td>
<td>800 79 11 37 (free phone)</td>
</tr>
<tr>
<td>Netherlands (English)</td>
<td>+31 (0) 546 87 95 45</td>
</tr>
<tr>
<td>Spain</td>
<td>+34 902 35 40 28</td>
</tr>
<tr>
<td>Sweden (English)</td>
<td>+46 (0) 8587 555 22</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>+44 (0) 1604 66 33 99</td>
</tr>
<tr>
<td>Finland (English)</td>
<td>+358 (0) 9 25 17 39 48</td>
</tr>
</tbody>
</table>
Fax: All Languages        | +49 (0) 8161 80 2045 |

Email: epic@ti.com
   » Literature Requests
   » SW registration/upgrades, HW repair/upgrades
   » Information and support for all TI Semiconductor products/tools
   » Submit suggestions and errata for tools, silicon and documents
This page is intentionally blank.
Module Topics

Module Topics........................................................................................................................................ A-1

eZdsp™ F2812 ........................................................................................................................................ A-3  
  eZdsp™ F2812 Connector / Header and Pin Diagram ................................................................. A-3  
P2 – Expansion Interface ............................................................................................................. A-4  
P4 / P8 / P7 – I/O Interface ........................................................................................................ A-5  
P5 / P9 – Analog Interface ........................................................................................................... A-7  
eZdsp™ F2812 Jumper Diagram .................................................................................................... A-8  
JP1 – XMP/MCn Select ................................................................................................................... A-8  
JP2 – Flash Programming Voltage Select .................................................................................. A-8  
JP9 – PLL Disable ...................................................................................................................... A-9  
DS1 / DS2 - LEDs ....................................................................................................................... A-9  
TP1 / TP2 – Test Points ................................................................................................................ A-10
eZdsp™ F2812

eZdsp™ F2812 Connector / Header and Pin Diagram

Table 1: eZdsp™ F2812 Connectors

<table>
<thead>
<tr>
<th>Connector</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>JTAG Interface</td>
</tr>
<tr>
<td>P2</td>
<td>Expansion</td>
</tr>
<tr>
<td>P3</td>
<td>Parallel Port/JTAG Controller Interface</td>
</tr>
<tr>
<td>P4/P8/P7</td>
<td>I/O Interface</td>
</tr>
<tr>
<td>P5/P9</td>
<td>Analog Interface</td>
</tr>
<tr>
<td>P6</td>
<td>Power Connector</td>
</tr>
</tbody>
</table>
# Appendix

## P2 – Expansion Interface

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal</th>
<th>Pin #</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+5V</td>
<td>2</td>
<td>+5V</td>
</tr>
<tr>
<td>3</td>
<td>XD0</td>
<td>4</td>
<td>XD1</td>
</tr>
<tr>
<td>5</td>
<td>XD2</td>
<td>6</td>
<td>XD3</td>
</tr>
<tr>
<td>7</td>
<td>XD4</td>
<td>8</td>
<td>XD5</td>
</tr>
<tr>
<td>9</td>
<td>XD6</td>
<td>10</td>
<td>XD7</td>
</tr>
<tr>
<td>11</td>
<td>XD8</td>
<td>12</td>
<td>XD9</td>
</tr>
<tr>
<td>13</td>
<td>XD10</td>
<td>14</td>
<td>XD11</td>
</tr>
<tr>
<td>15</td>
<td>XD12</td>
<td>16</td>
<td>XD13</td>
</tr>
<tr>
<td>17</td>
<td>XD14</td>
<td>18</td>
<td>XD15</td>
</tr>
<tr>
<td>19</td>
<td>XD0</td>
<td>20</td>
<td>XD1</td>
</tr>
<tr>
<td>21</td>
<td>XA2</td>
<td>22</td>
<td>XA3</td>
</tr>
<tr>
<td>23</td>
<td>XA4</td>
<td>24</td>
<td>XA5</td>
</tr>
<tr>
<td>25</td>
<td>XA6</td>
<td>26</td>
<td>XA7</td>
</tr>
<tr>
<td>27</td>
<td>XA8</td>
<td>28</td>
<td>XA9</td>
</tr>
<tr>
<td>29</td>
<td>XA10</td>
<td>30</td>
<td>XA11</td>
</tr>
<tr>
<td>31</td>
<td>XA12</td>
<td>32</td>
<td>XA13</td>
</tr>
<tr>
<td>33</td>
<td>XA14</td>
<td>34</td>
<td>XA15</td>
</tr>
<tr>
<td>35</td>
<td>GND</td>
<td>36</td>
<td>GND</td>
</tr>
<tr>
<td>37</td>
<td>XZCS0/AND1n</td>
<td>38</td>
<td>XZCS2n</td>
</tr>
<tr>
<td>39</td>
<td>XREADY</td>
<td>40</td>
<td>10K Pullup</td>
</tr>
<tr>
<td>41</td>
<td>XRnW</td>
<td>42</td>
<td>10K Pullup</td>
</tr>
<tr>
<td>43</td>
<td>XRDn</td>
<td>44</td>
<td>XRDn</td>
</tr>
<tr>
<td>45</td>
<td>+3.3V</td>
<td>46</td>
<td>XNMINT13</td>
</tr>
<tr>
<td>47</td>
<td>XRSm/RSm</td>
<td>48</td>
<td>No connect</td>
</tr>
<tr>
<td>49</td>
<td>GND</td>
<td>50</td>
<td>GND</td>
</tr>
<tr>
<td>51</td>
<td>GND</td>
<td>52</td>
<td>GND</td>
</tr>
<tr>
<td>53</td>
<td>XA16</td>
<td>54</td>
<td>XA17</td>
</tr>
<tr>
<td>55</td>
<td>XA18</td>
<td>56</td>
<td>XHOLDn</td>
</tr>
<tr>
<td>57</td>
<td>XHOLDAn</td>
<td>58</td>
<td>No connect</td>
</tr>
<tr>
<td>59</td>
<td>No connect</td>
<td>60</td>
<td>No connect</td>
</tr>
</tbody>
</table>
P4 / P8 / P7 – I/O Interface

|   | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| P4 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 2  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| P8 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 1  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| P7 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Table 3: P4/P8, I/O Connectors

<table>
<thead>
<tr>
<th>P4 Pin #</th>
<th>P4 Signal</th>
<th>P4 Pin #</th>
<th>P8 Signal</th>
<th>P8 Pin #</th>
<th>P8 Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+5 Volts</td>
<td>1</td>
<td>+5 Volts</td>
<td>2</td>
<td>+5 Volts</td>
</tr>
<tr>
<td>2</td>
<td>XINT2/ADCSOC</td>
<td>3</td>
<td>SCITXDA</td>
<td>4</td>
<td>SCIRXDA</td>
</tr>
<tr>
<td>3</td>
<td>MCLKXA</td>
<td>5</td>
<td>XINT1n/XBIC</td>
<td>6</td>
<td>CAP1/QEP1</td>
</tr>
<tr>
<td>4</td>
<td>MCLKRA</td>
<td>7</td>
<td>CAP2/QEP2</td>
<td>8</td>
<td>CAP3/QEP11</td>
</tr>
<tr>
<td>5</td>
<td>MFSX</td>
<td>9</td>
<td>PWM1</td>
<td>10</td>
<td>PWM2</td>
</tr>
<tr>
<td>6</td>
<td>MFSR</td>
<td>11</td>
<td>PWM3</td>
<td>12</td>
<td>PWM4</td>
</tr>
<tr>
<td>7</td>
<td>MDXA</td>
<td>13</td>
<td>PWM5</td>
<td>14</td>
<td>PWM6</td>
</tr>
<tr>
<td>8</td>
<td>MDRA</td>
<td>15</td>
<td>T1PWM/T1CMP</td>
<td>16</td>
<td>T2PWM/T2CMP</td>
</tr>
<tr>
<td>9</td>
<td>No connect</td>
<td>17</td>
<td>TDIRA</td>
<td>18</td>
<td>TCLKI0A</td>
</tr>
<tr>
<td>10</td>
<td>GND</td>
<td>19</td>
<td>GND</td>
<td>20</td>
<td>GND</td>
</tr>
<tr>
<td>11</td>
<td>CAP5/QEP4</td>
<td>21</td>
<td>No connect</td>
<td>22</td>
<td>XINT1n/XBIC</td>
</tr>
<tr>
<td>12</td>
<td>CAP6/QEP12</td>
<td>23</td>
<td>SPISMDA</td>
<td>24</td>
<td>SPISOMA</td>
</tr>
<tr>
<td>13</td>
<td>T3PWM/T3CMP</td>
<td>25</td>
<td>SPICLKA</td>
<td>26</td>
<td>SPISTE0A</td>
</tr>
<tr>
<td>14</td>
<td>T4PWM/T4CMP</td>
<td>27</td>
<td>CANTXA</td>
<td>28</td>
<td>CANRXA</td>
</tr>
<tr>
<td>15</td>
<td>TDIRB</td>
<td>29</td>
<td>XCLKOUT</td>
<td>30</td>
<td>PWM7</td>
</tr>
<tr>
<td>16</td>
<td>TCLKIN0</td>
<td>31</td>
<td>PWM8</td>
<td>32</td>
<td>PWM9</td>
</tr>
<tr>
<td>17</td>
<td>XF/XPLLDISn</td>
<td>33</td>
<td>PWM10</td>
<td>34</td>
<td>PWM11</td>
</tr>
<tr>
<td>18</td>
<td>SCITXDB</td>
<td>35</td>
<td>PWM12</td>
<td>36</td>
<td>CAP4/QEP3</td>
</tr>
<tr>
<td>19</td>
<td>SCIRXDB</td>
<td>37</td>
<td>T1CTRIP/DPDINT0n</td>
<td>38</td>
<td>T3CTRIP/DPDINT0n</td>
</tr>
</tbody>
</table>

| 20       | GND       | 39       | GND    | 40        | GND      |
### Table 4: P7, I/O Connector

<table>
<thead>
<tr>
<th>P7 Pin #</th>
<th>P7 Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C1TRIPn</td>
</tr>
<tr>
<td>2</td>
<td>C2TRIPn</td>
</tr>
<tr>
<td>3</td>
<td>C3TRIPn</td>
</tr>
<tr>
<td>4</td>
<td>T2CTRIPn/EVASOCn</td>
</tr>
<tr>
<td>5</td>
<td>C4TRIPn</td>
</tr>
<tr>
<td>6</td>
<td>C5TRIPn</td>
</tr>
<tr>
<td>7</td>
<td>C6TRIPn</td>
</tr>
<tr>
<td>8</td>
<td>T4CTRIPn/EVBSOCn</td>
</tr>
<tr>
<td>9</td>
<td>No connect</td>
</tr>
<tr>
<td>10</td>
<td>GND</td>
</tr>
</tbody>
</table>
## P5 / P9 – Analog Interface

<table>
<thead>
<tr>
<th>P5</th>
<th>ANALOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 5: P5/P9, Analog Interface Connector

<table>
<thead>
<tr>
<th>P5 Pin #</th>
<th>Signal</th>
<th>P9 Pin #</th>
<th>Signal</th>
<th>P9 Pin #</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ADCINB0</td>
<td>1</td>
<td>GND</td>
<td>2</td>
<td>ADCINA0</td>
</tr>
<tr>
<td>2</td>
<td>ADCINB1</td>
<td>3</td>
<td>GND</td>
<td>4</td>
<td>ADCINA1</td>
</tr>
<tr>
<td>3</td>
<td>ADCINB2</td>
<td>5</td>
<td>GND</td>
<td>6</td>
<td>ADCINA2</td>
</tr>
<tr>
<td>4</td>
<td>ADCINB3</td>
<td>7</td>
<td>GND</td>
<td>8</td>
<td>ADCINA3</td>
</tr>
<tr>
<td>5</td>
<td>ADCINB4</td>
<td>9</td>
<td>GND</td>
<td>10</td>
<td>ADCINA4</td>
</tr>
<tr>
<td>6</td>
<td>ADCINB5</td>
<td>11</td>
<td>GND</td>
<td>12</td>
<td>ADCINA5</td>
</tr>
<tr>
<td>7</td>
<td>ADCINB6</td>
<td>13</td>
<td>GND</td>
<td>14</td>
<td>ADCINA6</td>
</tr>
<tr>
<td>8</td>
<td>ADCINB7</td>
<td>15</td>
<td>GND</td>
<td>16</td>
<td>ADCINA7</td>
</tr>
<tr>
<td>9</td>
<td>ADCREFM</td>
<td>17</td>
<td>GND</td>
<td>18</td>
<td>VREFLO</td>
</tr>
<tr>
<td>10</td>
<td>ADCREFP</td>
<td>19</td>
<td>GND</td>
<td>20</td>
<td>No connect</td>
</tr>
</tbody>
</table>
eZdsp™ F2812 Jumper Diagram

**JP1 – XMP/MCn Select**

Table 6: JP1, XMP/MCn Select

<table>
<thead>
<tr>
<th>Position</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>Microprocessor mode</td>
</tr>
<tr>
<td>2-3 *</td>
<td>Microcomputer mode</td>
</tr>
</tbody>
</table>

* as shipped from factory

**JP2 – Flash Programming Voltage Select**

Table 7: JP2, Flash Programming Voltage Select

<table>
<thead>
<tr>
<th>Position</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2 *</td>
<td>Voltage supplied to DSP</td>
</tr>
<tr>
<td>2-3</td>
<td>Voltage not supplied to DSP</td>
</tr>
</tbody>
</table>

* always in this position
Appendix


*Table 8: JP7, JP8, JP11, JP12, Boot Mode Select*

<table>
<thead>
<tr>
<th>JP7, BOOT3 SCIXDA</th>
<th>JP8, BOOT2 MDXA</th>
<th>JP11, BOOT1 SPISTEA</th>
<th>JP12, BOOT0 SPICLKA</th>
<th>MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>FLASH</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>SPI</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>SCI</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>H0 *</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>OTP</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>PARALLEL</td>
</tr>
</tbody>
</table>

* factory default

**JP9 – PLL Disable**

*Table 9: JP9, PLL Disable*

<table>
<thead>
<tr>
<th>Position</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2 *</td>
<td>PLL Enabled</td>
</tr>
<tr>
<td>2-3</td>
<td>PLL disabled</td>
</tr>
</tbody>
</table>

* as shipped from the factory

**DS1 / DS2 - LEDs**

*Table 10: LEDs*

<table>
<thead>
<tr>
<th>LED #</th>
<th>Color</th>
<th>Controlling Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS1</td>
<td>Green</td>
<td>+5 Volts</td>
</tr>
<tr>
<td>DS2</td>
<td>Green</td>
<td>XF bit (XF high = on)</td>
</tr>
</tbody>
</table>
# TP1 / TP2 – Test Points

## Table 11: Test Points

<table>
<thead>
<tr>
<th>Test Point</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP1</td>
<td>Ground</td>
</tr>
<tr>
<td>TP2</td>
<td>Analog Ground</td>
</tr>
</tbody>
</table>