TMS320LF/LC240x DSP Controllers Reference Guide

System and Peripherals

Literature Number: SPRU357
January 2000

PRINTED WITH SOY INK™

TEXAS INSTRUMENTS

Printed on Recycled Paper

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 acknowledgement, 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.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

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.

Preface

Read This First

About This Manual

This reference guide describes the architecture, system hardware, peripherals, and general operation of the TMS320x2407/x2406/x2404/x2402 digital signal processor (DSP) controllers. For a description of the CPU, assembly language instructions, and XDS510 emulator, refer to *TMS320C24x DSP Controllers CPU and Instruction Set Reference Guide* (SPRU160). This book should be used in conjunction with SPRU160.

Notational Conventions

This document uses the following conventions:

- ❑ Program listings, program examples, and interactive displays are shown in a `special typeface` similar to a typewriter's. Examples use a **`bold version`** of the special typeface for emphasis; interactive displays use a **`bold version`** of the special typeface to distinguish commands that you enter from items that the system displays (such as prompts, command output, error messages, etc.).

  Here is a sample program listing:

  ```
  0011  0005  0001         .field    1, 2
  0012  0005  0003         .field    3, 4
  0013  0005  0006         .field    6, 3
  0014  0006               .even
  ```

  Here is an example of a system prompt and a command that you might enter:

  ```
  C:  csr –a /user/ti/simuboard/utilities
  ```

- ❑ In syntax descriptions, the instruction, command, or directive is in a **bold typeface** and parameters are in an *italic typeface*. Portions of a syntax that are in **bold** should be entered as shown; portions of a syntax that are in *italics* describe the type of information that should be entered. Here is an example of a directive syntax:

  .asect **"***section name***",** *address*

  .asect is the directive. This directive has two parameters, indicated by *section name* and *address*. When you use .asect, the first parameter must be an actual section name, enclosed in double quotes; the second parameter must be an address.

- ❑ Square brackets [ ] identify an optional parameter. If you use an optional parameter, you specify the information within the brackets; you don't enter the brackets themselves. Here's an example of an instruction that has an optional parameter:

  LACC *16-bit constant [, shift]*

  The LACC instruction has two parameters. The first parameter, *16-bit constant*, is required. The second parameter, *shift*, is optional. As this syntax shows, if you use the optional second parameter, you must precede it with a comma.

  Square brackets are also used as part of the pathname specification for VMS pathnames; in this case, the brackets are actually part of the pathname (they are not optional).

- ❑ Braces { } indicate a list. The symbol **|** (read as *or*) separates items within the list. Here's an example of a list:

  ```
  { * | *+ | *– }
  ```

  This provides three choices: `*`, `*+`, or `*–`.

  Unless the list is enclosed in square brackets, you must choose one item from the list.

- ❑ Some directives can have a varying number of parameters. For example, the .byte directive can have up to 100 parameters. The syntax for this directive is:

  .byte *value*$_1$ *[, ... , value*$_n$*]*

  This syntax shows that .byte must have at least one value parameter, but you have the option of supplying additional value parameters, separated by commas.

Information About Cautions and Warnings

This book may contain cautions.

> CAUTION
>
> **This is an example of a caution statement.**
>
> **A caution statement describes a situation that could potentially damage your software or equipment.**

Related Documentation From Texas Instruments

The following books describe the 'C24x and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477-8924. When ordering, please identify the book by its title and literature number. Many of these documents are located on the Internet at http://www.ti.com.

TMS320C24x DSP Controllers CPU and Instruction Set Reference Guide (literature number SPRU160) describes the TMS320C24x 16-bit fixed-point digital signal processor controller. Covered are its architecture, internal register structure, data and program addressing, and instruction set. Also includes instruction set comparisons and design considerations for using the XDS510 emulator.

TMS320LF2407, TMS320LF2406, TMS320LF2402, TMS320LC2406, TMS320LC2404, TMS320LC2402 DSP Controllers (literature number SPRS094) data sheet contains the electrical and timing specifications for these devices, as well as signal descriptions and pinouts for all of the available packages.

TMS320C1x/C2x/C2xx/C5x Code Generation Tools Getting Started Guide (literature number SPRU121) describes how to install the TMS320C1x, TMS320C2x, TMS320C2xx, and TMS320C5x assembly language tools and the C compiler for the 'C1x, 'C2x, 'C2xx, and 'C5x devices. The installations for MS-DOS™, OS/2™, SunOS™, and Solaris™ systems are covered.

TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User's Guide (literature number SPRU018) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the 'C1x, 'C2x, 'C2xx, and 'C5x generations of devices.

TMS320C2x/C2xx/C5x Optimizing C Compiler User's Guide (literature number SPRU024) describes the 'C2x/C2xx/C5x C compiler. This C compiler accepts ANSI standard C source code and produces TMS320 assembly language source code for the 'C2x, 'C2xx, and 'C5x generations of devices.

TMS320C2xx C Source Debugger User's Guide (literature number SPRU151) tells you how to invoke the 'C2xx emulator and simulator versions of the C source debugger interface. This book discusses various aspects of the debugger interface, including window management, command entry, code execution, data management, and breakpoints. It also includes a tutorial that introduces basic debugger functionality.

TMS320C2xx Simulator Getting Started (literature number SPRU137) describes how to install the TMS320C2xx simulator and the C source debugger for the 'C2xx. The installation for MS-DOS™, PC-DOS™, SunOS™, Solaris™, and HP-UX™ systems is covered.

TMS320C2xx Emulator Getting Started Guide (literature number SPRU209) tells you how to install the Windows™ 3.1 and Windows™ 95 versions of the 'C2xx emulator and C source debugger interface.

XDS51x Emulator Installation Guide (literature number SPNU070) describes the installation of the XDS510™, XDS510PP™, and XDS510WS™ emulator controllers. The installation of the XDS511™ emulator is also described.

JTAG/MPSD Emulation Technical Reference (literature number SPDU079) provides the design requirements of the XDS510™ emulator controller, discusses JTAG designs (based on the IEEE 1149.1 standard), and modular port scan device (MPSD) designs.

TMS320 DSP Development Support Reference Guide (literature number SPRU011) describes the TMS320 family of digital signal processors and the tools that support these devices. Included are code-generation tools (compilers, assemblers, linkers, etc.) and system integration and debug tools (simulators, emulators, evaluation modules, etc.). Also covered are available documentation, seminars, the university program, and factory repair and exchange.

TMS320 DSP Designer's Notebook: Volume 1 (literature number SPRT125) presents solutions to common design problems using 'C2x, 'C3x, 'C4x, 'C5x, and other TI DSPs.

TMS320 Third-Party Support Reference Guide (literature number SPRU052) alphabetically lists over 100 third parties that provide various products that serve the family of TMS320 digital signal processors. A myriad of products and applications are offered—software and hardware development tools, speech recognition, image processing, noise cancellation, modems, etc.

Trademarks

320 Hotline On-line is a trademark of Texas Instruments Incorporated.

cDSP is a trademark of Texas Instruments Incorporated.

HP-UX is a trademark of Hewlett-Packard Company.

MS-DOS is a registered trademark of Microsoft Corporation.

OS/2 is a trademark of International Business Machines Corporation.

PC is a trademark of International Business Machines Corporation.

PC-DOS is a trademark of International Business Machines Corporation.

Solaris is a trademark of Sun Microsystems, Inc.

SunOS is a trademark of Sun Microsystems, Inc.

TI is a trademark of Texas Instruments Incorporated.

Windows is a registered trademark of Microsoft Corporation.

XDS510 is a trademark of Texas Instruments Incorporated.

XDS510PP is a trademark of Texas Instruments Incorporated.

XDS510WS is a trademark of Texas Instruments Incorporated.

XDS511 is a trademark of Texas Instruments Incorporated.

Contents

Figures

Tables

Examples

Chapter 1

Introduction

The TMS320x240x series of devices are members of the TMS320 family of digital signal processors (DSPs) designed to meet a wide range of digital motor control (DMC) and other embedded control applications. This series is based on the ’C2xLP 16-bit, fixed-point, low-power DSP CPU, and is complemented with a wide range of on-chip peripherals and on-chip ROM or flash program memory, plus on-chip dual-access RAM (DARAM).

This reference guide describes the following four ’240x devices: ’2407, ’2406, ’2404, and ’2402. These low-cost DSPs are intended to enable multiple applications for a nominal price.

This chapter provides an overview of the current TMS320 family, describes the background and benefits of the ’240x DSP controller products, and introduces the ’2407, ’2406, ’2404, and ’2402 devices.

Throughout this book, all four devices are collectively referred to as ’240x devices.

Topic **Page**

1.1 TMS320 Family Overview

The TMS320 family consists of fixed-point, floating-point, multiprocessor digital signal processors (DSPs), and fixed-point DSP controllers. TMS320 DSPs have an architecture designed specifically for real-time signal processing. The ’240x series of DSP controllers combines this real-time processing capability with controller peripherals to create an ideal solution for control system applications. The following characteristics make the TMS320 family the right choice for a wide range of processing applications:

- ❑ Very flexible instruction set
- ❑ Inherent operational flexibility
- ❑ High-speed performance
- ❑ Innovative parallel architecture
- ❑ Cost effectiveness

In 1982, Texas Instruments introduced the TMS32010, the first fixed-point DSP in the TMS320 family. Before the end of the year, *Electronic Products* magazine awarded the TMS32010 the title “Product of the Year”. Today, the TMS320 family consists of these generations: ’C1x, ’C2x, ’C20x, ’C24x, ’C5x, ’C54x, and ’C6x fixed-point DSPs; ’C3x and ’C4x floating-point DSPs; and ’C8x multiprocessor DSPs. The ’240x devices are considered part of the ’24x generation of fixed-point DSPs, and members of the ’C2000 platform.

Devices within a generation of a TMS320 platform have the same CPU structure but different on-chip memory and peripheral configurations. Spin-off devices use new combinations of on-chip memory and peripherals to satisfy a wide range of needs in the worldwide electronics market. By integrating memory and peripherals onto a single chip, TMS320 devices reduce system costs and save circuit board space.

1.2 TMS320C240x Series of DSP Controllers

Designers have recognized the opportunity to redesign existing digital motor control (DMC) systems to use advanced algorithms that yield better performance and reduce system component count. DSPs enable:

- ❑ Design of robust controllers for a new generation of inexpensive motors, such as AC induction, DC permanent magnet, and switched-reluctance motors
- ❑ Full variable-speed control of brushless motor types that have lower manufacturing cost and higher reliability
- ❑ Energy savings through variable-speed control, saving up to 25% of the energy used by fixed-speed controllers
- ❑ Increased fuel economy, improved performance, and elimination of hydraulic fluid in automotive electronic power steering (EPS) systems
- ❑ Reduced manufacturing and maintenance costs by eliminating hydraulic fluid in automotive electronic braking systems
- ❑ More efficient and quieter operation due to diminished torque ripple, resulting in less loss of power, lower vibration, and longer life
- ❑ Elimination or reduction of memory lookup tables through real-time polynomial calculation, thereby reducing system cost
- ❑ Use of advanced algorithms that can reduce the number of sensors required in a system
- ❑ Control of power switching inverters, along with control algorithm processing
- ❑ Single-processor control of multimotor systems

The ’240x DSP controllers are designed to meet the needs of control-based applications. By integrating the high performance of a DSP core and the on-chip peripherals of a microcontroller into a single-chip solution, the ’240x series yields a device that is an affordable alternative to traditional microcontroller units (MCUs) and expensive multichip designs. At 30 million instructions per second (MIPS), the ’240x DSP controllers offer significant performance over traditional 16-bit microcontrollers and microprocessors.

The 16-bit, fixed-point DSP core of the ’240x devices provides analog designers a digital solution that does not sacrifice the precision and performance of their systems. In fact, system performance can be enhanced through the use of advanced control algorithms for techniques such as adaptive control, Kalman filtering, and state control. The ’240x DSP controllers offer reliability and programmability. Analog control systems, on the other hand, are hardwired solutions and can experience performance degradation due to aging, component tolerance, and drift.

The high-speed central processing unit (CPU) allows the digital designer to process algorithms in real time rather than approximate results with look-up tables. The instruction set of these DSP controllers, which incorporates both signal processing instructions and general-purpose control functions, coupled with the extensive development support available for the ’240x devices, reduces development time and provides the same ease of use as traditional 8- and 16-bit microcontrollers. The instruction set also allows you to retain your software investment when moving from other general-purpose TMS320 fixed-point DSPs. It is source- and object-code compatible with the other members of the ’24x generation, source-code compatible with the ’C2x generation, and upwardly source-code compatible with the ’C5x generation of DSPs from Texas Instruments.

The ’240x architecture is also well-suited for processing control signals. It uses a 16-bit word length along with 32-bit registers for storing intermediate results, and has two hardware shifters available to scale numbers independently of the CPU. This combination minimizes quantization and truncation errors, and increases processing power for additional functions. Such functions might include a notch filter that could cancel mechanical resonances in a system or an estimation technique that could eliminate state sensors in a system.

The ’240x DSP controllers take advantage of an existing set of peripheral functions that allow Texas Instruments to quickly configure various series members for different price/performance points or for application optimization. This library of both digital- and mixed-signal peripherals includes:

- Timers
- Serial communications ports (SCI, SPI)
- Analog-to-digital converters (ADC)
- Event manager
- Safety features such as watchdog timer and power drive protection

The DSP controller peripheral library is continually growing and changing to suit the needs of tomorrow’s embedded control marketplace.

1.3 Peripheral Overview

The peripheral set for the ’240x devices includes:

- ❑ Event Manager: Timers and PWM generators for digital motor control
- ❑ CAN Interface: Controller Area Network (CAN) 2.0b compatible, with six mailboxes (not available in ’2402 and ’2404)
- ❑ A/D: 10-bit ±1, 500 nS conversion, 16/8 channel, analog-to-digital converter
- ❑ SPI: Serial Peripheral Interface – synchronous serial port (not available in ’2402)
- ❑ SCI: Serial Communications Interface – asynchronous serial port (universal asynchronous receiver and transmitter – UART)
- ❑ Watchdog timer
- ❑ General-purpose bidirectional digital I/O (GPIO) pins

Note:

For device pinouts, electrical characteristics, and timing specifications of ’240x devices, refer to the following data sheet: *TMS320LF2407, TMS320LF2406, TMS320LF2402 TMS320LC2406, TMS320LC2404, TMS320LC2402 DSP Controllers* (literature number SPRS094).

1.4 ’240x Highlights

- The ’2407 has an external memory interface and is intended primarily for emulation tools.
- The ’2406 is similar to the ’240x but lacks external memory interface.
- The ’2402 is a minimum-cost motor control device.

The device configurations available and their features are shown in Table 1–1.

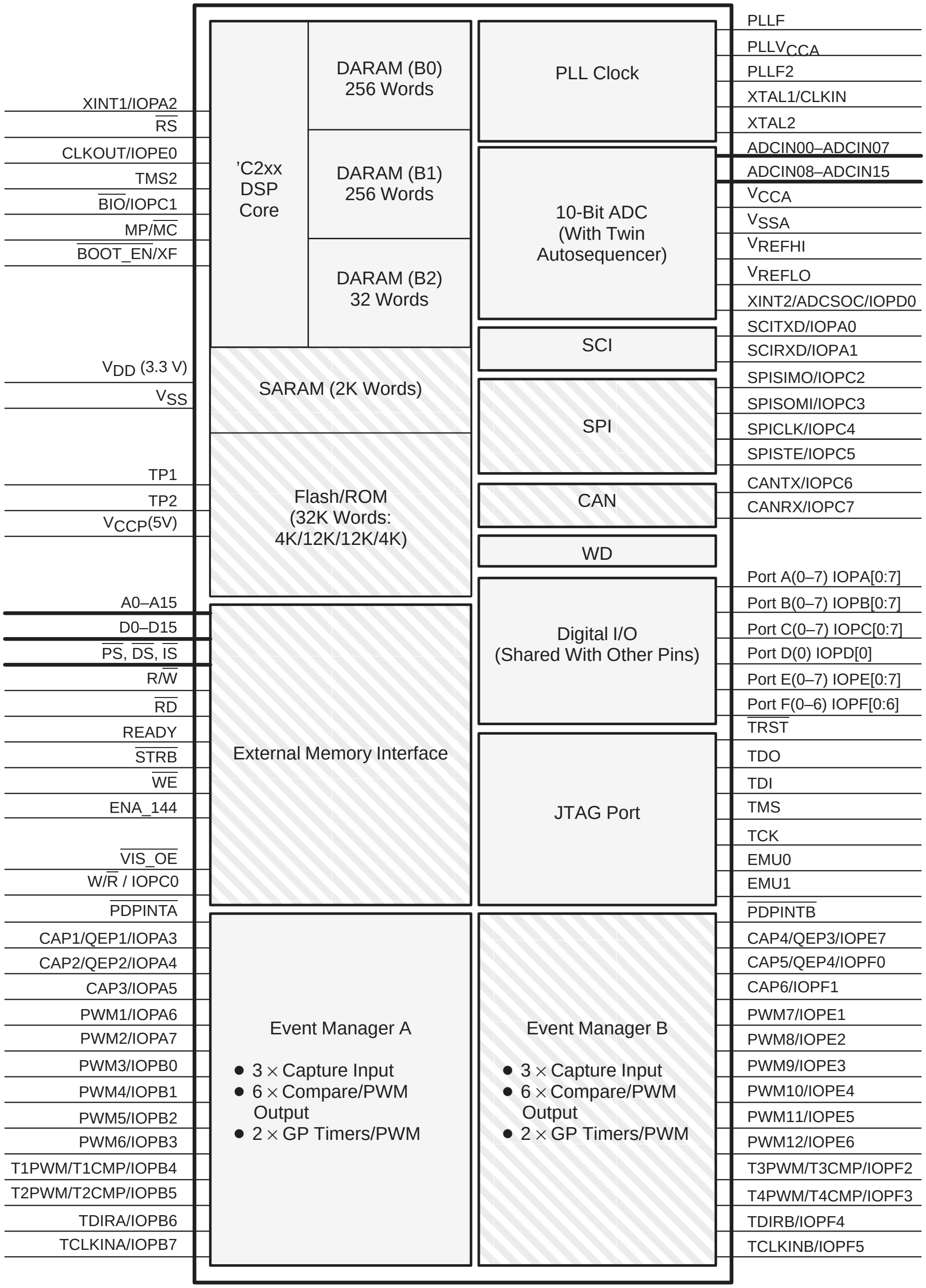
Table 1–1. Hardware Features of ’240x Devices

Feature		**’LF2407[†]**	**’LF2406**	**’LF2402**	**’LC2406**	**’LC2404**	**’LC2402**
’C2xx DSP Core		Yes	Yes	Yes	Yes	Yes	Yes
Instruction Cycle		33 ns	33 ns	33 ns	33 ns	33 ns	33 ns
MIPS (30 MHz)		30 MIPS	30 MIPS	30 MIPS	30 MIPS	30 MIPS	30 MIPS
RAM (16-bit word)	DARAM	544	544	544	544	544	544
	SARAM	2K	2K	—	2K	1K	—
On-chip Flash (16-bit word) (4 sectors: 4K, 12K, 12K, 4K)		32K	32K	8K	—	—	—
On-chip ROM (16-bit word)		—	—	—	32K	16K	4K
Boot ROM (16-bit word)		256	256	256	—	—	—
External Memory Interface		Yes	—	—	—	—	—
Event Managers A and B (EVA and EVB)		EVA, EVB	EVA, EVB	EVA	EVA, EVB	EVA, EVB	EVA
• General-Purpose (GP) Timers		4	4	2	4	4	2
• Compare (CMP)/PWM		10/16	10/16	5/8	10/16	10/16	5/8
• Capture (CAP)/QEP		6/4	6/4	3/2	6/4	6/4	3/2
Watchdog Timer		Yes	Yes	Yes	Yes	Yes	Yes
10-Bit ADC		Yes	Yes	Yes	Yes	Yes	Yes
• Channels		16	16	8	16	16	8
• Conversion Time (minimum)		500 ns	500 ns	500 ns	500 ns	500 ns	500 ns
SPI		Yes	Yes	—	Yes	Yes	—
SCI		Yes	Yes	Yes	Yes	Yes	Yes
CAN		Yes	Yes	—	Yes	—	—
Digital I/O Pins		41	41	21	41	41	21
External Interrupts		5	5	3	5	5	3
Supply Voltage		3.3 V	3.3 V	3.3 V	3.3 V	3.3 V	3.3 V
Packaging		144 TQFP	100 TQFP	64 PQFP	100 TQFP	100 TQFP	64 PQFP

[†] ’LF2407, the full-featured device of the ’LF240x family of DSP controllers, is useful for emulation and code development.

Figure 1–1 provides a graphical overview of the devices.

Figure 1–1. ’240x Device Overview



▨ Indicates optional modules. The memory size and peripheral selection of these modules change for different ’240x devices. See Table 1–1 for device-specific details.

Chapter 2

System Configuration and Interrupts

This chapter describes the system configuration registers and interrupts. It also explains how the peripheral interrupt expansion (PIE) is used to increase interrupt request capacity.

Topic **Page**

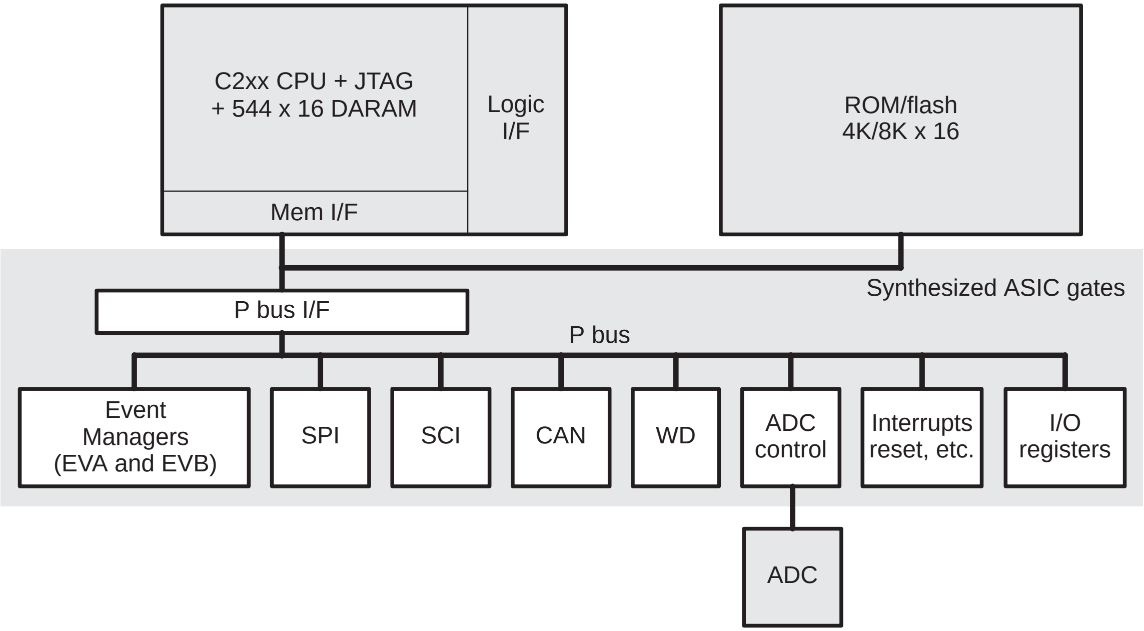
2.1 Architecture Summary

The ’240x devices are implemented as ASIC customizable digital signal processors (cDSPs™). The CPU, program ROM/FLASH is implemented as ASIC hard macros as shown in the shaded blocks in Figure 2–1. The CPU uses the LP256 hard macro which consists of the TMS320C2xx DSP CPU core, 544 x 16 words of dual-access RAM (DARAM), the analysis/JTAG logic, the internal memory interface, and the logic interface. The logic interface, however, is not used in the ’240x.

The peripherals interface to the internal memory interface of the CPU through the PBUS interface. All on-chip peripherals are accessed through the peripheral bus, PBUS. At lower frequencies, all peripheral accesses (reads and writes) are zero-wait-state, single-cycle accesses. All peripherals, excluding the watchdog timer counter, are clocked by the CPU clock. A third ASIC module is the 10-bit 500-ns A/D converter.

These devices have up to 41 bit-selectable digital I/O ports. Most or all of these I/O ports are multiplexed with other functions, such as event manager signals, serial communication port signals, or interrupts. Most of these multiplexed digital I/O pins come up in their digital I/O pin mode as an input following a device reset. For a detailed description of the architecture and instruction set, refer to the *TMS320C24x DSP Controllers CPU and Instruction Set Reference Guide* (SPRU160).

Figure 2–1. ’240x Device Architecture



2.2 Configuration Registers

2.2.1 System Control and Status Registers 1 and 2 (SCSR1, SCSR2)

Figure 2–2. System Control and Status Register 1 (SCSR1) — Address 07018h

15	14	13	12	11	10	9	8
OSC FAIL FLAG	CLKSRC	LPM1	LPM0	CLK_PS2	CLK_PS1	CLK_PS0	OSC FAIL RESET
RW-0	RW-0	RW-0	RW-0	RW-1	RW-1	RW-1	RW-0

7	6	5	4	3	2	1	0
ADC CLKEN	SCI CLKEN	SPI CLKEN	CAN CLKEN	EVB CLKEN	EVA CLKEN	Reserved	ILLADR
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	R-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 15 **Oscillator Fail Flag**

When set to 1, indicates that the external reference (oscillator or crystal) is operating at a frequency too slow to be functional when compared to an internal reference oscillator. Power on reset value is 0

0 Operating at too slow frequency

1 Operating normally.

Bit 14 **CLKSRC.** CLKOUT pin source select

0 CLKOUT pin has CPU Clock (30 MHz on a 30-MHz device) as the output.

1 CLKOUT pin has Watchdog clock as the output

Bits 13–12 **LPM(1:0).** Low-power mode select

These bits indicate which low-power mode is entered when the CPU executes the IDLE instruction. See Table 2–1 for description of the low-power modes.

Table 2–1. Description of Low-Power Modes

LPM(1:0)	Low-Power mode selected
00	IDLE1 (LPM0)
01	IDLE2. (LPM1)
1x	HALT (LPM2)

Bits 11–9 **PLL Clock prescale select.** These bits select the PLL multiplication factor for the input clock.

CLK PS2	CLK PS1	CLK PS0	System Clock Frequency
0	0	0	4 x F_{in}
0	0	1	2 x F_{in}
0	1	0	1.33 x F_{in}
0	1	1	1 x F_{in}
1	0	0	0.8 x F_{in}
1	0	1	0.66 x F_{in}
1	1	0	0.57 x F_{in}
1	1	1	0.5 x F_{in}

Note: F_{in} is the input clock frequency.

Bit 8 **Reset OSC Fail.** Reset if Oscillator fails

0 System reset is NOT initiated if the clock monitor detects a bad oscillator input.

1 System reset is initiated if the clock monitor detects a bad oscillator input.

Bit 7 **ADC CLKEN.** ADC module clock enable control bit

0 Clock to module is disabled (i.e. shut down to conserve power)

1 Clock to module is enabled and running normally

Bit 6 **SCI CLKEN.** SCI module clock enable control bit

0 Clock to module is disabled (i.e. shut down to conserve power)

1 Clock to module is enabled and running normally

Bit 5 **SPI CLKEN.** SPI module clock enable control bit

0 Clock to module is disabled (i.e. shut down to conserve power)

1 Clock to module is enabled and running normally

Bit 4 **CAN CLKEN.** CAN module clock enable control bit

0 Clock to module is disabled (i.e. shut down to conserve power)

1 Clock to module is enabled and running normally

Bit 3 **EVB CLKEN.** EVB module clock enable control bit

0 Clock to module is disabled (i.e. shut down to conserve power)

1 Clock to module is enabled and running normally

Bit 2 **EVA CLKEN.** EVA module clock enable control bit

0 Clock to module is disabled (i.e. shut down to conserve power)

1 Clock to module is enabled and running normally

Bit 1 **Reserved**

Bit 0 **ILLADR.** Illegal Address detect bit

If an illegal address has occurred this bit will be set. It is up to software to clear this bit following an illegal address detect. Note: An illegal address will cause an NMI.

Figure 2–3. System Control and Status Register 2 (SCSR2) — Address 07019h

15–8
Reserved
RW-0

7–6	5	4	3	2	1	0
Reserved	WD OVERRIDE	XMIF_HI-Z	$\overline{\text{BOOT\_EN}}$	MP/$\overline{\text{MC}}$	DON	PON
RW-0	RW-1	RW-0	RW-$\overline{\text{BOOT\_EN}}$ pin	RW-MP/$\overline{\text{MC}}$ pin	RW-1	RW-1

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–6 **Reserved.** Writes have no effect, read are undefined

Bit 5 **Watchdog Override**. (WD protect bit)

After RESET, this bit gives the user the ability to disable the WD function through software (by setting the WDDIS bit = 1 in the WDCR). This bit is a clear only bit and defaults to a 1 after reset, *Note: this bit is cleared by writing a 1 to it.*

0 Protects the WD from being disabled by software. This bit cannot be set to 1 by software. It is a clear-only bit, cleared by writing a 1.

1 This is the default reset value and allows the user to disable the WD through the WDDIS bit in the WDCR.
Once cleared however, this bit can no longer be set to 1 by software, thereby protecting the integrity of the WD timer.

Bit 4 **XMIF_Hi-Z Control**

This bit controls the state of the external memory interface (XMIF) signals.

0 XMIF signals in normal driven mode, i.e., not Hi-Z (high impedance)

1 All XMIF signal are forced to Hi-Z state

Bit 3 $\overline{\text{Boot Enable}}$

This bit reflects the state of the $\overline{\text{BOOT\_EN}}$ / XF pin at the time of reset. After reset and device has “Booted up”, this bit can be changed in software to re-enable Flash memory visibility, or return to active Boot ROM.

0 Enable Boot ROM — Address space 0000 — 00FF is now occupied by the on-chip Boot ROM Block. Flash memory is totally disabled in this mode.
Note: There is no on-chip boot ROM in ROM (i.e., “LC”240x) devices.

1 Disable Boot ROM — Program address space 0000 — 7FFF is mapped to on-chip Flash Memory, in case of ’LF2407 and ’LF2406. In case of ’LF2402, addresses 0000 – 1FFF are mapped.

Bit 2 Microprocessor / Microcontroller Select

This bit reflects the state of the MP/$\overline{\text{MC}}$ pin at time of reset. After reset this bit can be changed in software to allow dynamic mapping of memory on and off chip.

0 Set to Microcontroller mode — Program Address range 0000 — 7FFF is mapped internally (i.e. Flash or ROM)

1 Set to Microprocessor mode — Program Address range 0000 — 7FFF is mapped externally (i.e. Customer provides external memory device.)
Note: MP/$\overline{\text{MC}}$ pin is available only in ’LF2407.

Bits 1–0 SARAM Program / Data Space Select

DON	PON	SARAM status
0	0	SARAM not mapped (disabled), address space allocated to external memory.
0	1	SARAM mapped internally to Program space
1	0	SARAM mapped internally to Data space
1	1	SARAM block mapped internally to both Data and Program spaces. This is the default or reset value.

Note: See memory map for location of SARAM addresses

2.2.2 Device Identification Number Register (DINR)

Figure 2–4. Device Identification Number Register (DINR) — Address 701Ch

15	14	13	12	11	10	9	8
DIN15	DIN14	DIN13	DIN12	DIN11	DIN10	DIN9	DIN8
R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x

7	6	5	4	3	2	1	0
DIN7	DIN6	DIN5	DIN4	DIN3	DIN2	DIN1	DIN0
R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x

Note: R = Read access, -x = hardwired device-specific DIN value

Bits 15–4 **DIN15–DIN4.** These bits contain the hard wired device specific Device Identification Number.

Bits 3–0 **DIN3–DIN0.** These bits contain the hard wired device revision specific value.

Device	Rev #	DIN #
F2407	1	0510h
F2407	2	0520h
F2407	3	0530h
etc.		

2.3 Interrupt Priority and Vectors

A centralized interrupt expansion scheme is implemented in order to accommodate the large number of peripheral interrupts with the six maskable interrupts supported by the CPU. Table 2–2 provides the interrupt source priority and vectors for the ’240x devices. The details of the ’240x interrupt expansion scheme are explained in Chapter 2.

Table 2–2. ’240x Interrupt Source Priority and Vectors

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
1	Reset	RSN 0000h	N/A	N	$\overline{\text{RS}}$ Pin, Watchdog	Reset from pin, watchdog time out
2	Reserved	– 0026h	N/A	N	CPU	Emulator trap
3	$\overline{\text{NMI}}$	NMI 0024h	N/A	N	Nonmaskable interrupt	Nonmaskable interrupt

(a) **INT1** *(level 1)*

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
4	PDPINTA	INT1 0002h	0020h	Y	EVA	Power drive protection interrupt pin
5	PDPINTB	INT1 0002h	0019h	Y	EVB	Power drive protection interrupt pin
6	ADCINT	INT1 0002h	0004h	Y	ADC	ADC interrupt in high-priority mode
7	XINT1	INT1 0002h	0001h	Y	External interrupt logic	External interrupt pin in high-priority mode
8	XINT2	INT1 0002h	0011h	Y	External interrupt logic	External interrupt pin in high-priority mode
9	SPIINT	INT1 0002h	0005h	Y	SPI	SPI interrupt in high-priority mode
10	RXINT	INT1 0002h	0006h	Y	SCI	SCI receiver interrupt in high-priority mode
11	TXINT	INT1 0002h	0007h	Y	SCI	SCI transmitter interrupt in high-priority mode
12	CANMBINT	INT1 0002h	0040h	Y	CAN	CAN mailbox interrupt (high-priority mode)
13	CANERINT	INT1 0002h	0041h	Y	CAN	CAN error interrupt (high-priority mode)

Table 2–2. ’240x Interrupt Source Priority and Vectors (Continued)

(b) **INT2** *(level 2)*

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
14	CMP1INT	INT2 0004h	0021h	Y	EVA	Compare 1 interrupt
15	CMP2INT	INT2 0004h	0022h	Y	EVA	Compare 2 interrupt
16	CMP3INT	INT2 0004h	0023h	Y	EVA	Compare 3 interrupt
17	T1PINT	INT2 0004h	0027h	Y	EVA	Timer 1 period interrupt
18	T1CINT	INT2 0004h	0028h	Y	EVA	Timer 1 compare interrupt
19	T1UFINT	INT2 0004h	0029h	Y	EVA	Timer 1 underflow interrupt
20	T1OFINT	INT2 0004h	002Ah	Y	EVA	Timer 1 overflow interrupt
21	CMP4INT	INT2 0004h	0024h	Y	EVB	Compare 4 interrupt
22	CMP5INT	INT2 0004h	0025h	Y	EVB	Compare 5 interrupt
23	CMP6INT	INT2 0004h	0026h	Y	EVB	Compare 6 interrupt
24	T3PINT	INT2 0004h	002Fh	Y	EVB	Timer 3 period interrupt
25	T3CINT	INT2 0004h	0030h	Y	EVB	Timer 3 compare interrupt
26	T3UFINT	INT2 0004h	0031h	Y	EVB	Timer 3 underflow interrupt
27	T3OFINT	INT2 0004h	0032h	Y	EVB	Timer 3 overflow interrupt

Table 2–2. ’240x Interrupt Source Priority and Vectors (Continued)

(c) **INT3** *(level 3)*

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
28	T2PINT	INT3 0006h	002Bh	Y	EVA	Timer 2 period interrupt
29	T2CINT	INT3 0006h	002Ch	Y	EVA	Timer 2 compare interrupt
30	T2UFINT	INT3 0006h	002Dh	Y	EVA	Timer 2 underflow interrupt
31	T2OFINT	INT3 0006h	002Eh	Y	EVA	Timer 2 overflow interrupt
32	T4PINT	INT3 0006h	0039h	Y	EVB	Timer 4 period interrupt
33	T4CINT	INT3 0006h	003Ah	Y	EVB	Timer 4 compare interrupt
34	T4UFINT	INT3 0006h	003Bh	Y	EVB	Timer 4 undeflow interrupt
35	T4OFINT	INT3 0006h	003Ch	Y	EVB	Timer 4 overflow interrupt

(d) **INT4** *(level 4)*

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
36	CAP1INT	INT4 0008h	0033h	Y	EVA	Capture 1 interrupt
37	CAP2INT	INT4 0008h	0034h	Y	EVA	Capture 2 interrupt
38	CAP3INT	INT4 0008h	0035h	Y	EVA	Capture 3 interrupt
39	CAP4INT	INT4 0008h	0036h	Y	EVB	Capture 4 interrupt
40	CAP5INT	INT4 0008h	0037h	Y	EVB	Capture 5 interrupt
41	CAP6INT	INT4 0008h	0038h	Y	EVB	Capture 6 interrupt

Table 2–2. ’240x Interrupt Source Priority and Vectors (Continued)

(e) **INT5** *(level 5)*

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
27	SPIINT	INT5 000Ah	0005h	Y	SPI	SPI interrupt (low priority)
28	RXINT	INT5 000Ah	0006h	Y	SCI	SCI receiver interrupt (low-priority mode)
29	TXINT	INT5 000Ah	0007h	Y	SCI	SCI transmitter interrupt (low-priority mode)
30	CANMBINT	INT5 000Ah	0040h	Y	CAN	CAN mailbox interrupt (low-priority mode)
31	CANERINT	INT5 000Ah	0041h	Y	CAN	CAN error interrupt (low-priority mode)

(f) **INT6** *(level 6)*

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
32	ADCINT	INT6 000Ch	0004h	Y	ADC	ADC interrupt (low priority)
33	XINT1	INT6 000Ch	0001h	Y	External interrupt logic	External interrupt pins (low-priority mode)
34	XINT2	INT6 000Ch	0011h	Y	External interrupt logic	External interrupt pins (low-priority mode)
	Reserved	000Eh	N/A	Y	CPU	Analysis interrupt
N/A	TRAP	0022h	N/A	N/A	CPU	TRAP instruction
N/A	Phantom Interrupt Vector	N/A	0000h	N/A	CPU	Phantom interrupt vector

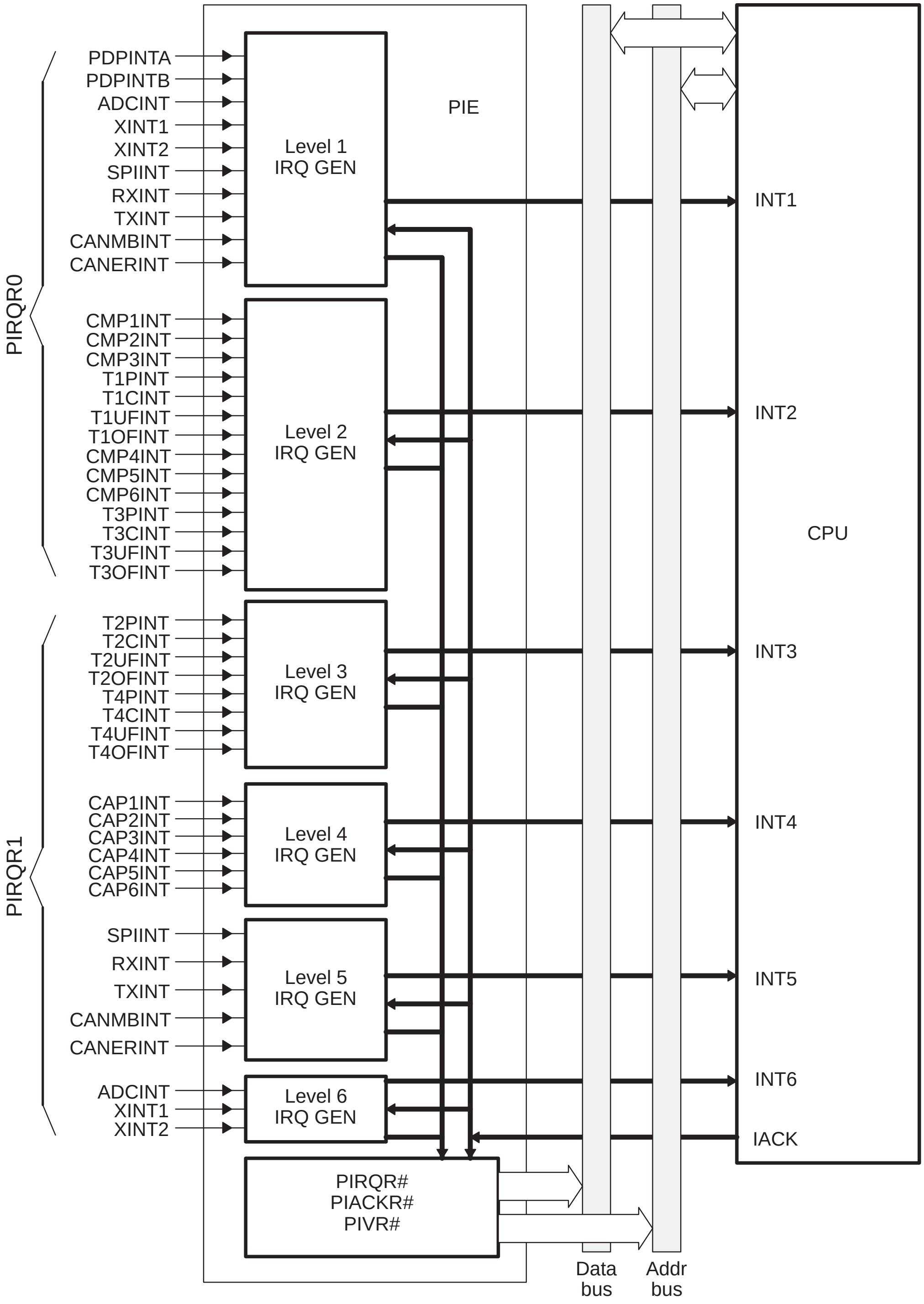
Note: Shaded interrupts are new interrupts added to ’240x by virtue of EVB.

2.4 Peripheral Interrupt Expansion (PIE)

The ’240x CPU supports one nonmaskable interrupt (NMI) and six maskable prioritized interrupt requests INT1–INT6 at the core level. The ’240x devices have many peripherals, and each peripheral is capable of generating one or more interrupts in response to many events at the peripheral level.

Because the ’C240x CPU does not have sufficient capacity to handle all peripheral interrupt requests at the core level, a centralized interrupt controller (PIE) is required to arbitrate the interrupt requests from various sources such as peripherals and other external pins (see Figure 2–5).

Figure 2–5. Peripheral Interrupt Expansion Block Diagram



2.4.1 Interrupt Hierarchy

The number of interrupt slots available is expanded by having two levels of hierarchy in the interrupt request system. Both the interrupt request/acknowledge hardware and the interrupt service routine software have two levels of hierarchy.

2.4.2 Interrupt Request Structure

At the lower level of the hierarchy, the peripheral interrupt requests (PIRQ) from several peripherals to the interrupt controller are ORed together to generate an interrupt request (INTn) to the CPU. This is the core level interrupt request. There is an interrupt flag bit and an interrupt enable bit located in the peripheral configuration registers for each event that can cause a PIRQ. There is also one PIRQ for each event. If an interrupt causing event occurs in a peripheral and the corresponding interrupt enable bit is set, the interrupt request from the peripheral to the interrupt controller will be asserted. This interrupt request simply reflects the status of the peripheral's interrupt flag, gated with the interrupt enable bit. When the interrupt flag is cleared, the interrupt request is cleared.

Some peripherals may have the capability to make either a high-priority or a low-priority interrupt request. If a peripheral has this capability, the value of its interrupt priority bit is also transmitted to the interrupt controller. The interrupt request (PIRQ) continues to be asserted until it is either automatically cleared by an interrupt acknowledge or cleared by the software.

At the upper level of the hierarchy, the ORed PIRQs generate interrupt (INTn) requests to the CPU. The request to the 'C240x CPU is a low-going pulse of two CPU clock cycles. The PIE controller generates an INTn pulse when any of the PIRQ's controlling the INTn become active. If any of the PIRQ's capable of asserting the CPU interrupt request are still active in the cycle following an interrupt acknowledge for the INTn, another INTn pulse is generated. An interrupt acknowledge clears the highest priority pending PIRQ. Note that the interrupts are automatically cleared only at the core level and not at the peripheral level. The interrupt controller (not the peripherals) defines the following:

❑ Which CPU interrupt request gets asserted by which peripheral

❑ Relative priority of each peripheral interrupt requests

This is shown in Table 2–2, *'240x Interrupt Source Priority and Vectors*, on page 2-8.

2.4.3 Interrupt Acknowledge

The hierarchical interrupt expansion scheme requires one interrupt acknowledge signal for each peripheral interrupt request to the interrupt controller. When the CPU asserts its interrupt acknowledge, it simultaneously puts a value on the program address bus, which corresponds to the CPU interrupt being acknowledged. (It does this to fetch the interrupt vector from program memory: each INTn has a vector stored in a dedicated program memory address.) This value is shown in Table 2–2, *’240x Interrupt Source Priority and Vectors*, on page 2-8. The PIE controller decodes this value to determine which of the CPU interrupt requests is being acknowledged. It then generates a peripheral interrupt acknowledge in response to the highest priority, currently asserted PIRQ associated with that CPU interrupt.

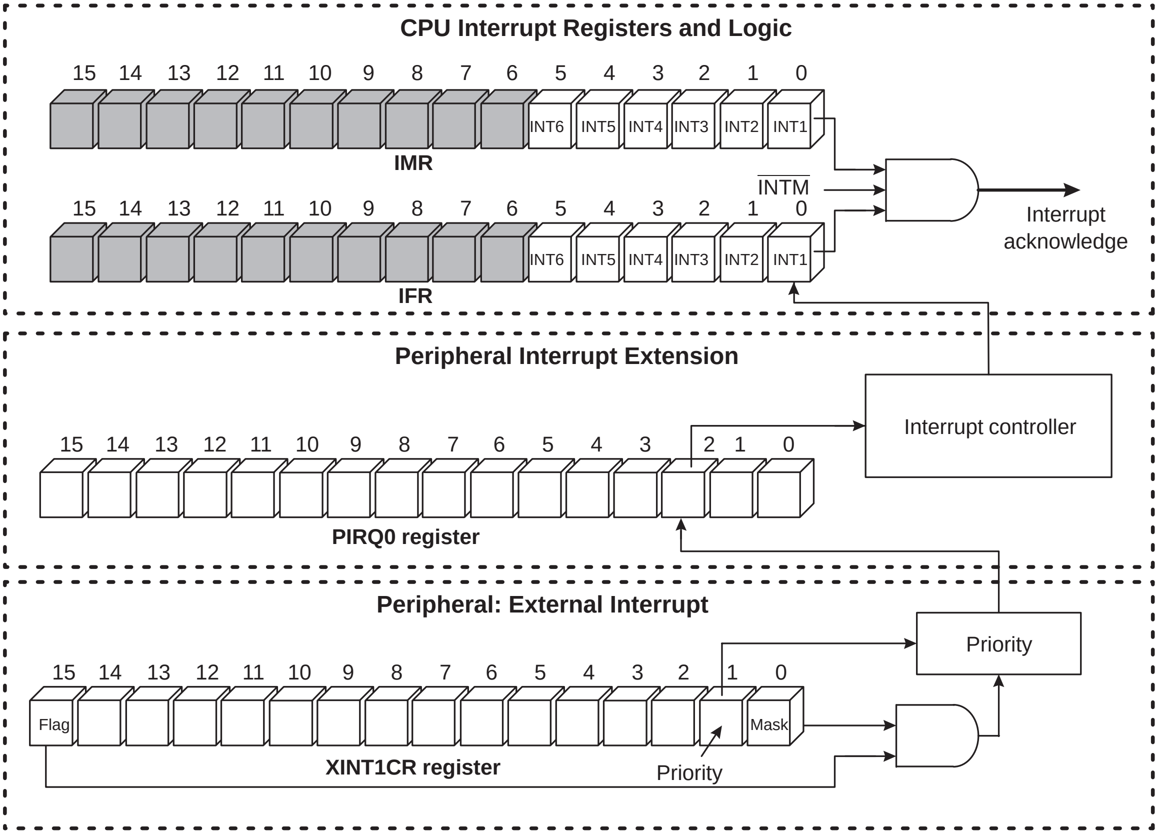
2.5 Interrupt Vectors

When the CPU receives an interrupt request, it does not know which peripheral event caused the request. To enable the CPU to distinguish between all of these events, a unique peripheral interrupt vector is generated in response to an active peripheral interrupt request. This vector is loaded into the peripheral interrupt vector register (PIVR) in the PIE controller. It can then be read by the CPU and used to generate a vector to branch to the interrupt service routine (ISR) which corresponds to the event being acknowledged.

In effect there are two vector tables: The CPU's vector table which is used to get to the first, general interrupt service routine (GISR) in response to a CPU interrupt request; and the peripheral vector table which is used to get to the event specific interrupt service routine (SISR) corresponding to the event which caused the PIRQ. The code in the GISR should read the PIVR, and after saving any necessary context, use this value to generate a vector to the SISR.

Figure 2–6 shows an example of how XINT1 (external interrupt in high-priority mode) generates an interrupt. For XINT1 in high-priority mode, a value of 0001h is loaded in the PIVR register. The CPU ascertains what value was loaded in the PIVR register and uses this value to determine which peripheral caused the interrupt and then branches to the appropriate SISR. Such a branch to the SISR could be a conditional branch (BCND) which is executed on the condition that the PIVR register holds a particular value. An alternative scheme would be to left-shift the PIVR register by 1 bit while loading it in the accumulator and adding a fixed offset value. Program control could then branch to the address value stored in the accumulator (using BACC instruction). This address would point to the SISR.

Figure 2–6. Interrupt Requests



2.5.1 Phantom Interrupt Vector

The phantom interrupt vector is an interrupt system integrity feature. If the CPU's interrupt acknowledge is asserted, but there is no associated peripheral interrupt request asserted, the phantom vector is used so that this fault is handled in a controlled manner. The phantom interrupt vector is required when, for example, the CPU executes a software interrupt instruction with an argument corresponding to a peripheral interrupt (usually INT1–INT6). Another example is when a peripheral makes an interrupt request but its INTn flag was cleared by software before the CPU acknowledged the request. In this case, there may be no peripheral interrupt request asserted to the interrupt controller; and therefore, the controller does not know which peripheral interrupt vector to load into the PIVR. In these two situations, the phantom interrupt vector is loaded into the PIVR in lieu of a peripheral interrupt vector.

2.5.2 Software Hierarchy

There are two levels of interrupt service routine hierarchy: the general interrupt service routine (GISR), and the specific interrupt service routine (SISR). There is one GISR for each maskable prioritized request (INT1–INT6) to the CPU which performs all necessary context saves before it fetches the peripheral interrupt vector from the PIVR. This vector is used to generate a branch to the SISR. There is one SISR for every interrupt request (IRQn) from a peripheral to the interrupt controller and this SISR performs the required actions in response to the peripheral interrupt request.

2.5.3 Nonmaskable Interrupts

Nonmaskable interrupts such as reset and NMI are not part of PIE. The PIE controller does not support expansion of nonmaskable interrupts. This is because an ISR must read the peripheral interrupt vector from the PIVR before interrupts are reenabled. (All interrupts are automatically disabled when an interrupt is taken.) If the PIVR is not read before interrupts are reenabled, another interrupt is acknowledged and a new peripheral interrupt vector is loaded into the PIVR, causing permanent loss of the original peripheral interrupt vector.

2.6 Interrupt Operation Sequence

An interrupt generating event occurs in a peripheral. Refer to Figure 2–7 for ’240x interrupt response and flow in each module of the ’240x.The interrupt flag (IF) bit corresponding to that event is set in a register in the peripheral. If the corresponding interrupt enable (IE) bit is set, the peripheral generates an interrupt request to the PIE controller by asserting its PIRQ. If the interrupt is not enabled, the IF remains set until cleared by software. If the interrupt is enabled at a later time, and the interrupt flag is still set, the PIRQ will immediately be asserted.

If no unacknowledged CPU interrupt request of the same priority level (INTn) has previously been sent, the PIRQ causes the PIE controller to generate a CPU interrupt request (INTn). This pulse is active low for two CPU clock cycles.

The interrupt request to the CPU sets the corresponding flag in the CPU’s interrupt flag register (IFR). If the CPU interrupt has been enabled by setting the corresponding bit in the CPU’s interrupt mask register (IMR), the CPU stops what it is doing, masks all other maskable interrupts by setting the INTM bit, saves some context, and starts executing the general interrupt service routine (GISR) for that interrupt priority level (INTn). The CPU generates an interrupt acknowledge automatically which is accompanied by a value on the program address bus (PAB) corresponding to the interrupt priority level being responded to. For example, if INT3 is asserted, its vector 0006h is loaded in the PAB. This is the interrupt vector corresponding to INTn (refer to Table 2–2 *’240x Interrupt Source Priority and Vectors,* on page 2-8).

The PIE controller decodes the PAB value and generates a peripheral interrupt acknowledge to clear the PIRQ bit associated with the CPU interrupt being acknowledged. The PIE controller then loads the peripheral interrupt vector register (PIVR) with the appropriate peripheral interrupt vector (or the phantom interrupt vector), from the table stored in the PIE controller.

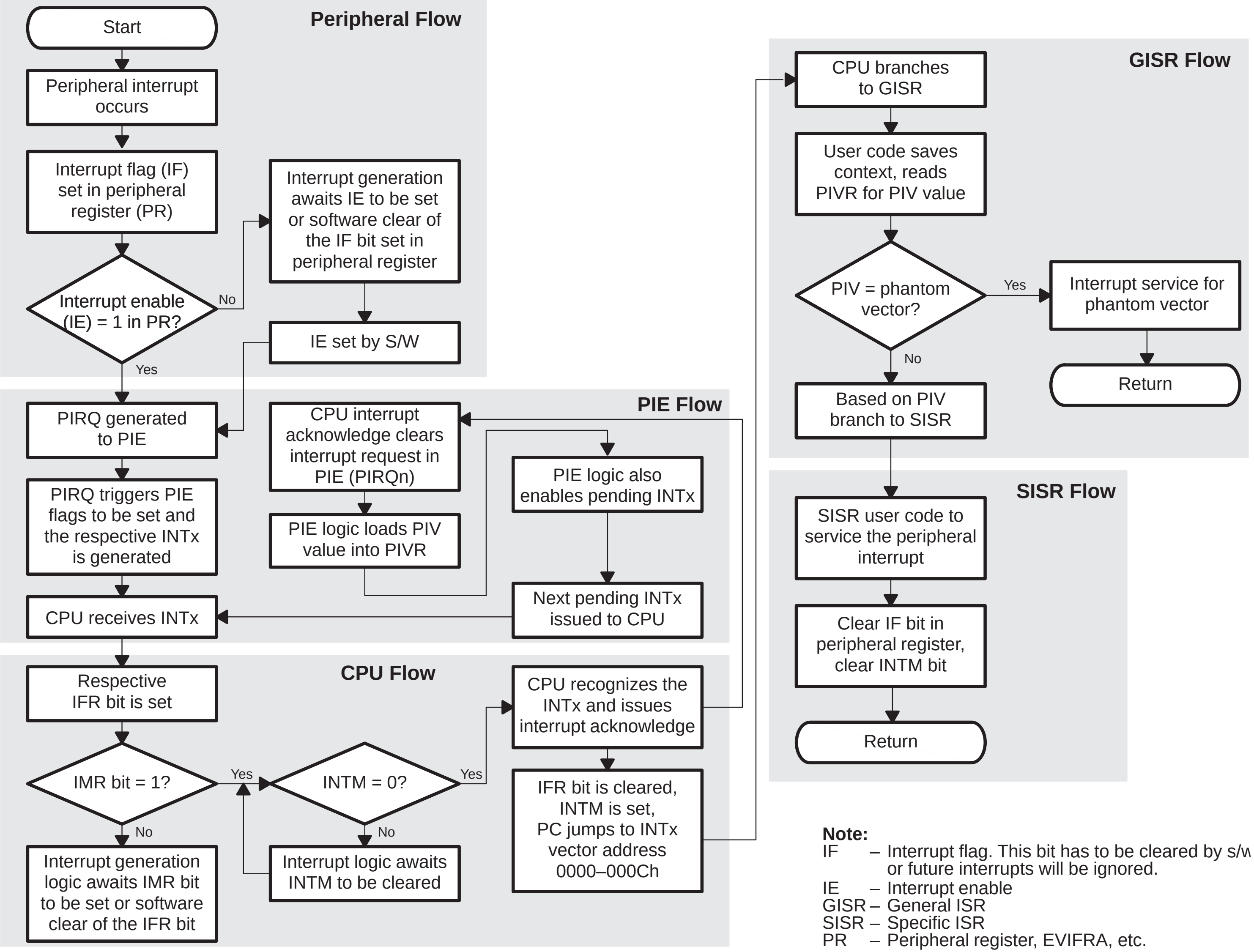
When the GISR has completed any necessary context saves, it reads the PIVR and uses that interrupt vector to branch to the specific interrupt service routine (SISR) for the interrupt event which occurred in the peripheral.

CAUTION

Re-enabling interrupts

Interrupts *must not* be reenabled until the PIVR has been read; otherwise, it’s contents can get overwritten by a subsequent interrupt.

Figure 2–7. ’240x Interrupt Response and Flow


Peripheral Flow
Start
Peripheral interrupt occurs
Interrupt flag (IF) set in peripheral register (PR)
Interrupt enable (IE) = 1 in PR?
No
Yes
Interrupt generation awaits IE to be set or software clear of the IF bit set in peripheral register
IE set by S/W
PIE Flow
PIRQ generated to PIE
PIRQ triggers PIE flags to be set and the respective INTx is generated
CPU receives INTx
CPU interrupt acknowledge clears interrupt request in PIE (PIRQn)
PIE logic loads PIV value into PIVR
PIE logic also enables pending INTx
Next pending INTx issued to CPU
CPU Flow
Respective IFR bit is set
IMR bit = 1?
Yes
No
Interrupt generation logic awaits IMR bit to be set or software clear of the IFR bit
INTM = 0?
Yes
No
Interrupt logic awaits INTM to be cleared
CPU recognizes the INTx and issues interrupt acknowledge
IFR bit is cleared, INTM is set, PC jumps to INTx vector address 0000–000Ch
GISR Flow
CPU branches to GISR
User code saves context, reads PIVR for PIV value
PIV = phantom vector?
Yes
No
Interrupt service for phantom vector
Return
Based on PIV branch to SISR
SISR Flow
SISR user code to service the peripheral interrupt
Clear IF bit in peripheral register, clear INTM bit
Return

Note:

IF – Interrupt flag. This bit has to be cleared by s/w or future interrupts will be ignored.
IE – Interrupt enable
GISR – General ISR
SISR – Specific ISR
PR – Peripheral register, EVIFRA, etc.

2.7 Interrupt Latency

There are three components to interrupt latency:

1) *Synchronization* is the time it takes for the request generated in response to the occurrence of an interrupt generating event to be recognized by the PIE controller and converted into a request to the CPU.

2) *Core Latency* is the time it takes for the CPU to recognize the enabled interrupt request, clear it's pipeline, and begin fetching the first instruction from the CPU's interrupt vector table. There is a minimum core latency of four CPU cycles. If a higher priority maskable interrupt is requested during this minimum latency period, it is masked until the ISR for the interrupt being serviced reenables interrupt. The latency can be longer than the minimum if the interrupt request occurs during an uninterruptible operation, e.g., a repeat loop, a multi-cycle instruction, or during a wait-stated access. If a higher priority interrupt occurs during this additional latency period, it gets serviced *before* the original lower priority interrupt, assuming both are enabled.

3) *ISR Latency* is the time it takes to get to the specific interrupt service routine (ISR) code for the event that caused the acknowledged interrupt. ISR latency can vary depending on how much context saving is required.

2.8 Sample ISR Code

```
; This sample ISR code illustrates how to branch to a SISR corresponding
; to a peripheral interrupt. No context save is done.
; Timer 1 period interrupt is assumed

                main code
                    .
        B       GISR2           ; This instruction resides at 0004h of PM
                    .
                    .
;========================================================================
; ISRs
;========================================================================
GISR2:  LDP     #PIVR >> 7h     ; Load the data page containing PIVR
        LACL    PIVR            ; Load PIVR value in the accumulator
        XOR     #0027h          ; Timer 1 period interrupt ?
        BCND    SISR27,eq       ; Branch to T1PINT if Accumulator = 0
                                ; Else reload PIVR in the accumulator and continue
                                ; checking for other peripheral interrupts

SISR27: ...........             ; Execute the ISR specific to T1PINT
       ...........              ; After executing the SISR, clear the flag bit
        LDP     #0E8h           ; that asserted the interrupt, so that future
        SPLK    #0080h, EVIFRA  ; interrupts may be recognized

EXIT_ISR
        CLRC    INTM            ; Before exiting the SISR, clear the interrupt
        RET                     ; mode bit
```

2.9 CPU Interrupt Registers

The CPU interrupt registers in the upper level of heirarchy include the following:

- The interrupt flag register (IFR)
- The interrupt mask register (IMR)

2.9.1 Interrupt Flag Register (IFR)

The interrupt flag register (IFR), a 16-bit, memory-mapped register at address 0006h in data-memory space, is used to identify and clear pending interrupts. The IFR contains flag bits for all the maskable interrupts (INT1–INT6).

When a maskable interrupt is requested, the flag bit in the corresponding peripheral control register is set to 1. If the corresponding mask bit is also 1, the interrupt request is sent to the CPU, setting the corresponding flag in the IFR. This indicates that the interrupt is pending or waiting for acknowledgement.

You can read the IFR to identify pending interrupts and write to the IFR to clear pending interrupts. To clear a single interrupt, write a 1 to the corresponding IFR bit. All pending interrupts can be cleared by writing the current contents of the IFR back into the IFR.

The following events also clear an IFR flag:

- The CPU acknowledges the interrupt.
- The ’240x is reset.

Notes:

1) To clear an IFR bit, you must write a 1 to it, not a 0.

2) When a maskable interrupt is acknowledged, *only* the IFR bit is cleared automatically. The flag bit in the corresponding peripheral control register is *not* cleared. If an application requires that the control register flag be cleared, the bit must be cleared by software.

3) When an interrupt is requested by an INTR instruction and the corresponding IFR bit is set, the CPU does not clear the bit automatically. If an application requires that the IFR bit be cleared, the bit must be cleared by software.

4) IMR and IFR registers pertain to core-level interrupts. All peripherals have their own interrupt mask and flag bits in their respective control/configuration registers. Note that several peripheral interrupts are grouped under one core-level interrupt.

Figure 2–8. Interrupt Flag Register (IFR) — Address 0006h

15–6	5	4	3	2	1	0
Reserved	INT6 flag	INT5 flag	INT4 flag	INT3 flag	INT2 flag	INT1 flag
0	RW1C-0	RW1C-0	RW1C-0	RW1C-0	RW1C-0	RW1C-0

Note: 0 = Always read as zeros, R = Read access, W1C = Write 1 to this bit to clear it, -0 = value after reset

Bits 15–6 **Reserved.** These bits are always read as 0s.

Bit 5 **INT6.** Interrupt 6 flag. This bit is the flag for interrupts connected to interrupt level INT6.

0 No INT6 interrupt is pending.
1 At least one INT6 interrupt is pending. Write a 1 to this bit to clear it to 0 and clear the interrupt request.

Bit 4 **INT5.** Interrupt 5 flag. This bit is the flag for interrupts connected to interrupt level INT5.

0 No INT5 interrupt is pending.
1 At least one INT5 interrupt is pending. Write a 1 to this bit to clear it to 0 and clear the interrupt request.

Bit 3 **INT4.** Interrupt 4 flag. This bit is the flag for interrupts connected to interrupt level INT4.

0 No INT4 interrupt is pending.
1 At least one INT4 interrupt is pending. Write a 1 to this bit to clear it to 0 and clear the interrupt request.

Bit 2 **INT3.** Interrupt 3 flag. This bit is the flag for interrupts connected to interrupt level INT3.

0 No INT3 interrupt is pending.
1 At least one INT3 interrupt is pending. Write a 1 to this bit to clear it to 0 and clear the interrupt request.

Bit 1 **INT2.** Interrupt 2 flag. This bit is the flag for interrupts connected to interrupt level INT2.

0 No INT2 interrupt is pending.
1 At least one INT2 interrupt is pending. Write a 1 to this bit to clear it to 0 and clear the interrupt request.

Bit 0 **INT1.** Interrupt 1 flag. This bit is the flag for interrupts connected to interrupt level INT1.

0 No INT1 interrupt is pending.
1 At least one INT1 interrupt is pending. Write a 1 to this bit to clear it to 0 and clear the interrupt request.

2.9.2 Interrupt Mask Register (IMR)

The IMR is a 16-bit, memory-mapped register located at address 0004h in data memory space. The IMR contains mask bits for all the maskable interrupt levels (INT1–INT6). Neither $\overline{\text{NMI}}$ nor $\overline{\text{RS}}$ is included in the IMR; thus, IMR has no effect on these interrupts.

You can read the IMR to identify masked or unmasked interrupt levels, and you can write to the IMR to mask or unmask interrupt levels. To unmask an interrupt level, set its corresponding IMR bit to 1. To mask an interrupt level, set its corresponding IMR bit to 0. When an interrupt is masked, it is not acknowledged, regardless of the value of the INTM bit. When an interrupt is unmasked, it is acknowledged if the corresponding IFR bit is 1 and the INTM bit is 0.

The IMR is shown in Figure 2–9, and descriptions of the bits follow the figure.

Figure 2–9. Interrupt Mask Register (IMR) — Address 0004h

15–6	5	4	3	2	1	0
Reserved	INT6 mask	INT5 mask	INT4 mask	INT3 mask	INT2 mask	INT1 mask
0	RW	RW	RW	RW	RW	RW

Note: 0 = Always read as zeros, R = Read access, W = Write access, bit values are not affected by a device reset

Bits 15–6 **Reserved**. These bits are always read as 0s.

Bit 5 **INT6.** Interrupt 6 mask. This bit masks or unmasks interrupt level INT6.

0 Level INT6 is masked.
1 Level INT6 is unmasked.

Bit 4 **INT5.** Interrupt 5 mask. This bit masks or unmasks interrupt level INT5.

0 Level INT5 is masked.
1 Level INT5 is unmasked.

Bit 3 **INT4.** Interrupt 4 mask. This bit masks or unmasks interrupt level INT4.

0 Level INT4 is masked.
1 Level INT4 is unmasked.

Bit 2 **INT3.** Interrupt 3 mask. This bit masks or unmasks interrupt level INT3.

0 Level INT3 is masked.
1 Level INT3 is unmasked.

Bit 1 **INT2.** Interrupt 2 mask. This bit masks or unmasks interrupt level INT2.

0 Level INT2 is masked.
1 Level INT2 is unmasked.

Bit 0 **INT1.** Interrupt 1 mask. This bit masks or unmasks interrupt level INT1.

0 Level INT1 is masked.
1 Level INT1 is unmasked.

Note: The IMR bits are not affected by a device reset.

2.10 Peripheral Interrupt Registers

The peripheral interrupt registers include the following:

- The peripheral interrupt vector register (PIVR)
- The peripheral interrupt request register 0 (PIRQR0)
- The peripheral interrupt request register 1 (PIRQR1)
- The peripheral interrupt request register 2 (PIRQR2)
- The peripheral interrupt acknowledge register 0 (PIACKR0)
- The peripheral interrupt acknowledge register 1 (PIACKR1)
- The peripheral interrupt acknowledge register 2 (PIACKR2)

> **PIRQR0/1/2 and PIACKR0/1/2 are control registers internal to PIE module for generating interrupts (INT1 – INT6) to the CPU. While programming, these registers can be ignored as they monitor the internal operation of the PIE. These registers are generally used for test purposes.**

2.10.1 Peripheral Interrupt Vector Register (PIVR)

The peripheral interrupt vector register (PIVR) is a 16-bit read-only register. It is located at address 701Eh (in data space).

During the peripheral interrupt acknowledge cycle, the PIVR is loaded with the interrupt vector of the highest-priority pending interrupt associated with the CPU interrupt (INTn) being acknowledged (or the phantom interrupt vector). The PIVR is shown in Figure 2–10.

Figure 2–10. Peripheral Interrupt Vector Register (PIVR)— Address 701Eh

15	14	13	12	11	10	9	8
V15	V14	V13	V12	V11	V10	V9	V8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
V7	V6	V5	V4	V3	V2	V1	V0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Note: R = Read access; -0 = value after reset

Bits 15–0 **V15–V0**. Interrupt vector. This register contains the peripheral interrupt vector of the most recently acknowledged peripheral interrupt.

2.10.2 Peripheral Interrupt Request Registers (PIRQR0, 1, 2)

The peripheral interrupt request registers (PIRQRx) enable:

❑ The state of the peripheral interrupt requests to be read

❑ A simulated assertion of a particular peripheral interrupt request

PIRQR0 is shown in Figure 2–11, PIRQR1 is shown in Figure 2–12, and PIRQR2 is shown in Figure 2–13.

Figure 2–11. Peripheral Interrupt Request Register 0 (PIRQR0) — Address 7010h

15	14	13	12	11	10	9	8
IRQ0.15	IRQ0.14	IRQ0.13	IRQ0.12	IRQ0.11	IRQ0.10	IRQ0.9	IRQ0.8
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
IRQ0.7	IRQ0.6	IRQ0.5	IRQ0.4	IRQ0.3	IRQ0.2	IRQ0.1	IRQ0.0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–0 IRQ0.15–IRQ0.0

0 Corresponding peripheral interrupt is not pending.

1 Peripheral Interrupt is pending.

Note: Writing a 1 sends IRQ to core; writing a 0 has no effect.

Table 2–3. Peripheral Interrupt Request Descriptions (PIRQR0)

Bit position	Interrupt	Interrupt Description	Interrupt Level
IRQ 0.0	PDPINTA	Power device protection interrupt pin	INT1
IRQ 0.1	ADCINT	ADC Interrupt. High priority	INT1
IRQ 0.2	XINT1	External Interrupt pin 1. High priority	INT1
IRQ 0.3	XINT2	External Interrupt pin 2. High priority	INT1
IRQ 0.4	SPIINT	SPI interrupt. High priority	INT1
IRQ 0.5	RXINT	SCI receiver interrupt. High priority	INT1
IRQ 0.6	TXINT	SCI transmitter interrupt. High priority	INT1
IRQ 0.7	CANMBINT	CAN mailbox interrupt. High priority	INT1
IRQ 0.8	CANERINT	CAN error interrupt. High priority	INT1
IRQ 0.9	CMP1INT	Compare 1 interrupt	INT2

Table 2–3. Peripheral Interrupt Request Descriptions (PIRQR0) (Continued)

Bit position	Interrupt	Interrupt Description	Interrupt Level
IRQ 0.10	CMP2INT	Compare 2 interrupt	INT2
IRQ 0.11	CMP3INT	Compare 3 interrupt	INT2
IRQ 0.12	T1PINT	Timer 1 period interrupt	INT2
IRQ 0.13	T1CINT	Timer 1 compare interrupt	INT2
IRQ 0.14	T1UFINT	Timer 1 underflow interrupt	INT2
IRQ 0.15	T1OFINT	Timer 1 overflow interrupt	INT2

Figure 2–12. Peripheral Interrupt Request Register 1 (PIRQR1) — Address 7011h

15	14	13	12	11	10	9	8
Reserved	IRQ1.14	IRQ1.13	IRQ1.12	IRQ1.11	IRQ1.10	IRQ1.9	IRQ1.8
R-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
IRQ1.7	IRQ1.6	IRQ1.5	IRQ1.4	IRQ1.3	IRQ1.2	IRQ1.1	IRQ1.0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 15 **Reserved**. Reads return zero, writes have no effect.

Bits 14–0 **IRQ1.14–IRQ1.0**

0 Corresponding peripheral interrupt is not pending.

1 Peripheral Interrupt is pending.

Note: Writing a 1 sends IRQ to core; writing a 0 has no effect.

Table 2–4. Peripheral Interrupt Request Descriptions (PIRQR1)

Bit position	Interrupt	Interrupt Description	Interrupt Level
IRQ 1.0	T2PINT	Timer 2 period interrupt	INT3
IRQ 1.1	T2CINT	Timer 2 compare interrupt	INT3
IRQ 1.2	T2UFINT	Timer 2 underflow interrupt	INT3
IRQ 1.3	T2OFINT	Timer 2 overflow interrupt	INT3
IRQ 1.4	CAP1INT	Capture 1 interrupt	INT4
IRQ 1.5	CAP2INT	Capture 2 interrupt	INT4
IRQ 1.6	CAP3INT	Capture 3 interrupt	INT4
IRQ 1.7	SPIINT	SPI interrupt. Low priority	INT5
IRQ 1.8	RXINT	SCI receiver interrupt. Low priority	INT5
IRQ 1.9	TXINT	SCI transmitter interrupt. Low priority	INT5
IRQ 1.10	CANMBINT	CAN mailbox interrupt. Low priority	INT5
IRQ 1.11	CANERINT	CAN error interrupt. Low priority	INT5
IRQ 1.12	ADCINT	ADC Interrupt. Low priority	INT6
IRQ 1.13	XINT1	External Interrupt pin 1. Low priority	INT6
IRQ 1.14	XINT2	External Interrupt pin 2. Low priority	INT6

Figure 2–13. Peripheral Interrupt Request Register 2 (PIRQR2) — Address 7012h

15	14	13	12	11	10	9	8
IRQ2.15	IRQ2.14	IRQ2.13	IRQ2.12	IRQ2.11	IRQ2.10	IRQ2.9	IRQ2.8
R-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
IRQ2.7	IRQ2.6	IRQ2.5	IRQ2.4	IRQ2.3	IRQ2.2	IRQ2.1	IRQ2.0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, -0 = value after reset

Bits 15–0 IRQ2.15–IRQ2.0

0 Corresponding peripheral interrupt is not pending.

1 Peripheral Interrupt is pending.

Note: Writing a 1 sends IRQ to core; writing a 0 has no effect.

Table 2–5. Peripheral Interrupt Request Descriptions (PIRQR2)

Bit position	Interrupt	Interrupt Description	Interrupt Level
IRQ 2.0	PDPINTB	Power drive protection interrupt pin	INT1
IRQ 2.1	CMP4INT	Compare 4 interrupt	INT2
IRQ 2.2	CMP5INT	Compare 5 interrupt	INT2
IRQ 2.3	CMP6INT	Compare 6 interrupt	INT2
IRQ 2.4	T3PINT	Timer 3 period interrupt	INT2
IRQ 2.5	T3CINT	Timer 3 compare interrupt	INT2
IRQ 2.6	T3UFINT	Timer 3 underflow interrupt	INT2
IRQ 2.7	T3OFINT	Timer 3 overflow interrupt	INT2
IRQ 2.8	T4PINT	Timer 4 period interrupt	INT3
IRQ 2.9	T4CINT	Timer 4 compare interrupt	INT3
IRQ 2.10	T4UFINT	Timer 4 underflow interrupt	INT3
IRQ 2.11	T4OFINT	Timer 4 overflow interrupt	INT3
IRQ 2.12	CAP4INT	Capture 4 interrupt	INT4
IRQ 2.13	CAP5INT	Capture 5 interrupt	INT4
IRQ 2.14	CAP6INT	Capture 6 interrupt	INT4

2.10.3 Peripheral Interrupt Acknowledge Registers (PIACKR0 ,1, 2)

The peripheral interrupt acknowledge registers (PIACKRx) are memory mapped to enable an easy test of the peripheral interrupt acknowledges. There are three of these 16-bit registers; and therefore, the PIE controller can support up to 48 peripheral interrupts. These registers are generally used for test purposes only and are not for user applications. PIACKR0 is shown in Figure 2–14, PIACKR1 is shown in Figure 2–15, and PIACKR2 is shown in Figure 2–16.

Figure 2–14. Peripheral Interrupt Acknowledge Register 0 (PIACKR0) — Address 7014h

15	14	13	12	11	10	9	8
IAK0.15	IAK0.14	IAK0.13	IAK0.12	IAK0.11	IAK0.10	IAK0.9	IAK0.8
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
IAK0.7	IAK0.6	IAK0.5	IAK0.4	IAK0.3	IAK0.2	IAK0.1	IAK0.0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–0 **IACK0.15–IACK0.0**. Peripheral interrupt acknowledge bits. Writing a 1 causes the corresponding peripheral interrupt acknowledge to be asserted, which clears the corresponding peripheral interrupt request. Note that asserting the interrupt acknowledge by writing to this register does not update the PIVR. Reading the register always returns zeros.

Table 2–6. Peripheral Interrupt Acknowledge Descriptions (PIACKR0)

Bit position	Interrupt	Interrupt Description	Interrupt Level
IAK 0.0	PDPINT	Power Device Protection interrupt pin	INT1
IAK 0.1	ADCINT	ADC Interrupt. High priority	INT1
IAK 0.2	XINT1	External Interrupt pin 1. High priority	INT1
IAK 0.3	XINT2	External Interrupt pin 2. High priority	INT1
IAK 0.4	SPIINT	SPI interrupt. High priority	INT1
IAK 0.5	RXINT	SCI receiver interrupt. High priority	INT1
IAK 0.6	TXINT	SCI transmitter interrupt. High priority	INT1
IAK 0.7	CANMBINT	CAN mailbox interrupt. High priority	INT1
IAK 0.8	CANERINT	CAN error interrupt. High priority	INT1
IAK 0.9	CMP1INT	Compare 1 interrupt	INT2
IAK 0.10	CMP2INT	Compare 2 interrupt	INT2
IAK 0.11	CMP3INT	Compare 3 interrupt	INT2
IAK 0.12	T1PINT	Timer 1 period interrupt	INT2
IAK 0.13	T1CINT	Timer 1 compare interrupt	INT2
IAK 0.14	T1UFINT	Timer 1 underflow interrupt	INT2
IAK 0.15	T1OFINT	Timer 1 overflow interrupt	INT2

Figure 2–15. Peripheral Interrupt Acknowledge Register 1 (PIACKR1) — Address 7015h

15	14	13	12	11	10	9	8
Reserved	IAK1.14	IAK1.13	IAK1.12	IAK1.11	IAK1.10	IAK1.9	IAK1.8
R-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
IAK1.7	IAK1.6	IAK1.5	IAK1.4	IAK1.3	IAK1.2	IAK1.1	IAK1.0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 15 **Reserved**. Reads return zero; writes have no effect.

Bits 14–0 **IACK1.14–IACK1.0**. Bit behavior is the same as that of PIACKR0.

Table 2–7. Peripheral Interrupt Acknowledge Descriptions (PIACKR1)

Bit position	Interrupt	Interrupt Description	Interrupt Level
IAK 1.0	T2PINT	Timer 2 period interrupt	INT3
IAK 1.1	T2CINT	Timer 2 compare interrupt	INT3
IAK 1.2	T2UFINT	Timer 2 underflow interrupt	INT3
IAK 1.3	T2OFINT	Timer 2 overflow interrupt	INT3
IAK 1.4	CAPINT1	Capture 1 interrupt	INT4
IAK 1.5	CAPINT2	Capture 2 interrupt	INT4
IAK 1.6	CAPINT3	Capture 3 interrupt	INT4
IAK 1.7	SPIINT	SPI interrupt. Low priority	INT5
IAK 1.8	RXINT	SCI receiver interrupt. Low priority	INT5
IAK 1.9	TXINT	SCI transmitter interrupt. Low priority	INT5
IAK 1.10	CANMBINT	CAN mailbox interrupt. Low priority	INT5
IAK 1.11	CANERINT	CAN error interrupt. Low priority	INT5
IAK 1.12	ADCINT	ADC Interrupt. Low priority	INT6
IAK 1.13	XINT1	External Interrupt pin 1. Low priority	INT6
IAK 1.14	XINT2	External Interrupt pin 2. Low priority	INT6

Figure 2–16. Peripheral Interrupt Acknowledge Register 2 (PIACKR2) — Address 7016h

15	14	13	12	11	10	9	8
IAK2.15	IAK2.14	IAK2.13	IAK2.12	IAK2.11	IAK2.10	IAK2.9	IAK2.8
R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
IAK2.7	IAK2.6	IAK2.5	IAK2.4	IAK2.3	IAK2.2	IAK2.1	IAK2.0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Note: R = read access; W = write access; value following dash (–) is value after reset

Bits 15–0 **IACK2.15–IACK2.0.** Bit behavior is the same as that of PIACKR0.

Table 2–8. Peripheral Interrupt Acknowledge Descriptions (PIACKR2)

Bit position	Interrupt	Interrupt Description	Interrupt Level
IAK 2.0	PDPINTB	Power drive protection interrupt pin	INT1
IAK 2.1	CMP4INT	Compare 4 interrupt	INT2
IAK 2.2	CMP5INT	Compare 5 interrupt	INT2
IAK 2.3	CMP6INT	Compare 6 interrupt	INT2
IAK 2.4	T3PINT	Timer 3 period interrupt	INT2
IAK 2.5	T3CINT	Timer 3 compare interrupt	INT2
IAK 2.6	T3UFINT	Timer 3 underflow interrupt	INT2
IAK 2.7	T3OFINT	Timer 3 overflow interrupt	INT2
IAK 2.8	T4PINT	Timer 4 period interrupt	INT3
IAK 2.9	T4CINT	Timer 4 compare interrupt	INT3
IAK 2.10	T4UFINT	Timer 4 underflow interrupt	INT3
IAK 2.11	T4OFINT	Timer 4 overflow interrupt	INT3
IAK 2.12	CAP4INT	Capture 4 interrupt	INT4
IAK 2.13	CAP5INT	Capture 5 interrupt	INT4
IAK 2.14	CAP6INT	Capture 6 interrupt	INT4

2.11 Reset

The ’240x devices have two sources of reset:

❑ An external reset pin

❑ A watchdog timer timeout

The reset pin is an I/O pin. If there is an internal reset event (watchdog timer), the reset pin is put into output mode and driven low to indicate to external circuits that the ’240x device is resetting itself.

The external reset pin and watchdog timer reset are ORed together to drive the reset input to the CPU.

2.12 Illegal Address Detect

The decode logic has the capability to detect accesses to illegal addresses (all unimplemented addresses including *reserved* registers in each peripheral’s memory map). The occurrence of an illegal access sets the illegal address flag (ILLADR) in the “System Control and Status Register 1” (SCSR1). See section 2.2.1, *System Control and Status Registers 1 and 2 (SCSR1, SCSR2)*, on page 2-3. The detection of an illegal address generates a nonmaskable interrupt ($\overline{\text{NMI}}$). The illegal address condition is asserted whenever illegal addresses are accessed. The illegal address flag (ILLADR) remains set following an illegal address condition until it is cleared by software. A check for illegal address access is made for data memory space only.

2.13 External Interrupt Control Registers

The two external interrupt control registers that control and monitor XINT1 and XINT2 pin activities are XINT1CR and XINT2CR.

2.13.1 External Interrupt 1 Control Register (XINT1CR)

Figure 2–17. External Interrupt 1 Control Register (XINT1CR) — Address 7070h

15	14–3	2	1	0
XINT1 flag	Reserved	XINT1 polarity	XINT1 priority	XINT1 enable
RC-0	R-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, C = Clear by writing a 1, -0 = value after reset

Bit 15 **XINT1 Flag**

This bit indicates whether the selected transition has been detected on the XINT1 pin and is set whether or not the interrupt is enabled. This bit is cleared by the appropriate interrupt acknowledge, by software writing a 1 (writing a 0 has no effect), or by a device reset.

0 No transition detected

1 Transition detected

Bits 14–3 **Reserved.** Reads return zero; writes have no effect.

Bit 2 **XINT1 Polarity**

This read/write bit determines whether interrupts are generated on the rising edge or the falling edge of a signal on the pin.

0 Interrupt generated on a falling edge (high-to-low transition).

1 Interrupt generated on a rising edge (low-to-high transition).

Bit 1 **XINT1 Priority**

This read/write bit determines which interrupt priority is requested. The CPU interrupt priority levels corresponding to low and high priority are coded into the peripheral interrupt expansion controller. These priority levels are shown in Table 2–2, *’240x Interrupt Source Priority and Vectors,* in Chapter 2, on page 2-8.

0 High priority

1 Low priority

Bit 0 **XINT1 Enable**

This read/write bit enables or disables external interrupt XINT1.

0 Disable Interrupt

1 Enable interrupt

2.13.2 External Interrupt 2 Control Register (XINT2CR)

Figure 2–18. External Interrupt 2 Control Register (XINT2CR) — Address 7071h

15	14–3	2	1	0
XINT2 flag	Reserved	XINT2 polarity	XINT2 priority	XINT2 enable
RC-0	R-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, C = Clear by writing a 1, -0 = value after reset

Bit 15 **XINT2 Flag**

This bit indicates whether the selected transition has been detected on the XINT2 pin, and is set whether or not the interrupt is enabled. This bit is cleared by the appropriate interrupt acknowledge, by software writing a 1 (writing a 0 has no effect), or by a device reset.

0	No transition detected
1	Transition detected

Bits 14–3 **Reserved.** Reads return zero; writes have no effect.

Bit 2 **XINT2 Polarity**

This read/write bit determines whether interrupts are generated on the rising edge or the falling edge of a signal on the pin.

0	Interrupt generated on a falling edge (high-to-low transition).
1	Interrupt generated on a rising edge (low-to-high transition).

Bit 1 **XINT2 Priority**

This read/write bit determines which interrupt priority is requested. The CPU interrupt priority levels corresponding to low and high priority are coded into the peripheral interrupt expansion controller. These priority levels are shown in Table 2–2, *’240x Interrupt Source Priority and Vectors,* in Chapter 2, on page 2-8.

0	High priority
1	Low priority

Bit 0 **XINT2 Enable**

This read/write bit enables or disables the external interrupt XINT2.

0	Disable Interrupt
1	Enable interrupt

Chapter 3

Memory

This chapter describes the RAM, ROM, and Flash availability on the ’240x.

In addition to dual-access RAM (DARAM – B0, B1, B2), which is part of the CPU core, the ’240x devices include flash EPROM or ROM for additional on-chip program memory. Devices with the “LF” prefix are flash devices and devices with the “LC” prefix are ROM devices.

The ’2407 has a 16-bit address bus that can access three individually selectable spaces (192K words total):

❑ A 64K-word program space
❑ A 64K-word data space
❑ A 64K-word I/O space

This chapter shows memory maps for program, data, and I/O spaces. It also describes available ’240x memory configuration options.

Topic **Page**

3.1 Factory Masked On-Chip ROM

The on-chip ROM in ROM devices is mapped in program memory space. This ROM is always enabled since these devices lack an external memory interface. This ROM is programmed with customer-specific code.

3.2 Flash

The on-chip flash in flash devices is mapped in program memory space. This flash memory is always enabled in devices that lack an external memory interface. For the ’2407 which has an external memory interface, the MP/$\overline{\text{MC}}$ pin determines whether the on-chip program memory (flash) or the off-chip program memory (customer design specific) is accessed.

3.2.1 Flash Control Register Access

In addition to the flash memory array, the flash module has four registers that control operations on the flash array. At any given time, you can access the memory array in the flash module (array-access mode) or you can access the control registers (register-access mode) but you cannot access both simultaneously. The flash module has a flash-access control register that selects between the two access modes. This register is the flash control mode register (FCMR) and is mapped at FF0Fh in I/O space. This is a special type of I/O register that cannot be read. The register functions as follows:

- An OUT instruction, using the register address as an I/O port, places the flash module in register-access mode. The data operand used is insignificant. For example:

  ```
  OUT   dummy, 0FF0Fh; Selects register-access mode
  ```

- An IN instruction, using the register address as an I/O port, places the flash module in array-access mode. The data operand used is insignificant. For example:

  ```
  IN   dummy, 0FF0Fh; Selects array-access mode
  ```

The flash array is not directly accessible as memory in register-access mode, and the control registers are not directly accessible in array-access mode. When operating as a program memory to store code, the flash module operates in array-access mode. For details on programming the flash array in ’240x, refer to the *TMS320F20x/F24x DSP Embedded Flash Memory Technical Reference* (SPRU282).

3.3 Overview of Memory and I/O Spaces

The ’240x design is based on an enhanced Harvard architecture. These devices have multiple memory spaces accessible on three parallel buses: a program address bus (PAB), a data-read address bus (DRAB), and a data-write address bus (DWAB). Each of the three buses access different memory spaces for different phases of the device’s operation. Because the bus operations are independent, it is possible to access both the program and data spaces simultaneously. Within a given machine cycle, the CALU can execute as many as three concurrent memory operations.

The ’240x address map is organized into three individually selectable spaces:

- **Program memory** (64K words) contains the instructions to be executed, as well as immediate data used during program execution.
- **Data memory** (64K words) holds data used by the instructions.
- **Input/output (I/O) space** (64K words) interfaces to external peripherals and may contain on-chip registers.

These spaces provide a total address space of 192K words. The ’240x devices include on-chip memory to aid in system performance and integration.

The advantages of operating from on-chip memory are:

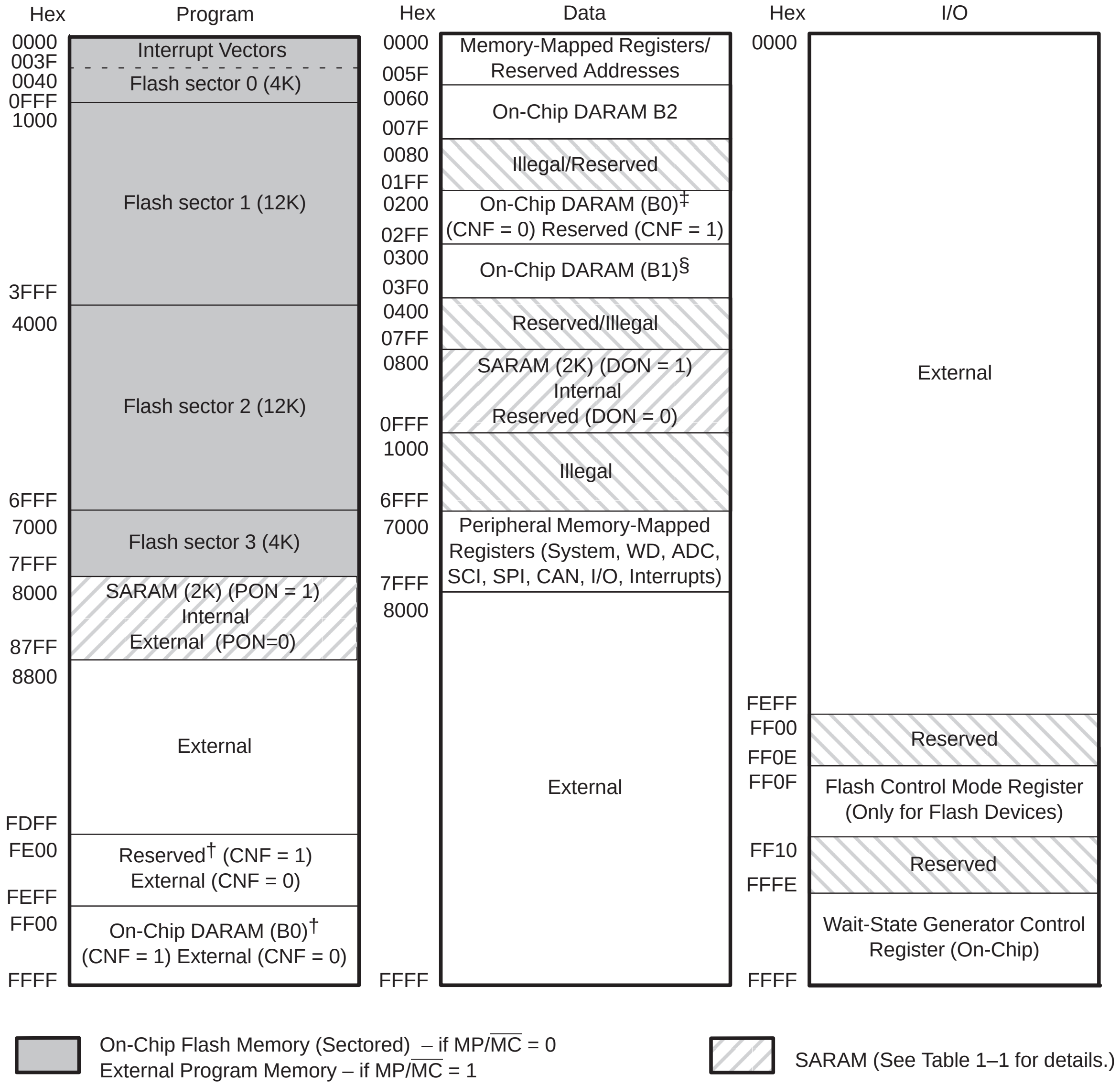
- Higher performance than external memory (because the wait states required for slower external memories are avoided)
- Lower cost than external memory
- Lower power consumption than external memory

The advantage of operating from external memory is the ability to access a larger address space. Only the ’2407 has an external memory interface. Other devices have only on-chip memory. Figure 3–1 through Figure 3–6 depict the memory map of ’240x devices.

CAUTION

Access to an illegal address will generate an NMI.

Figure 3–1. Memory Map for 'LF2407



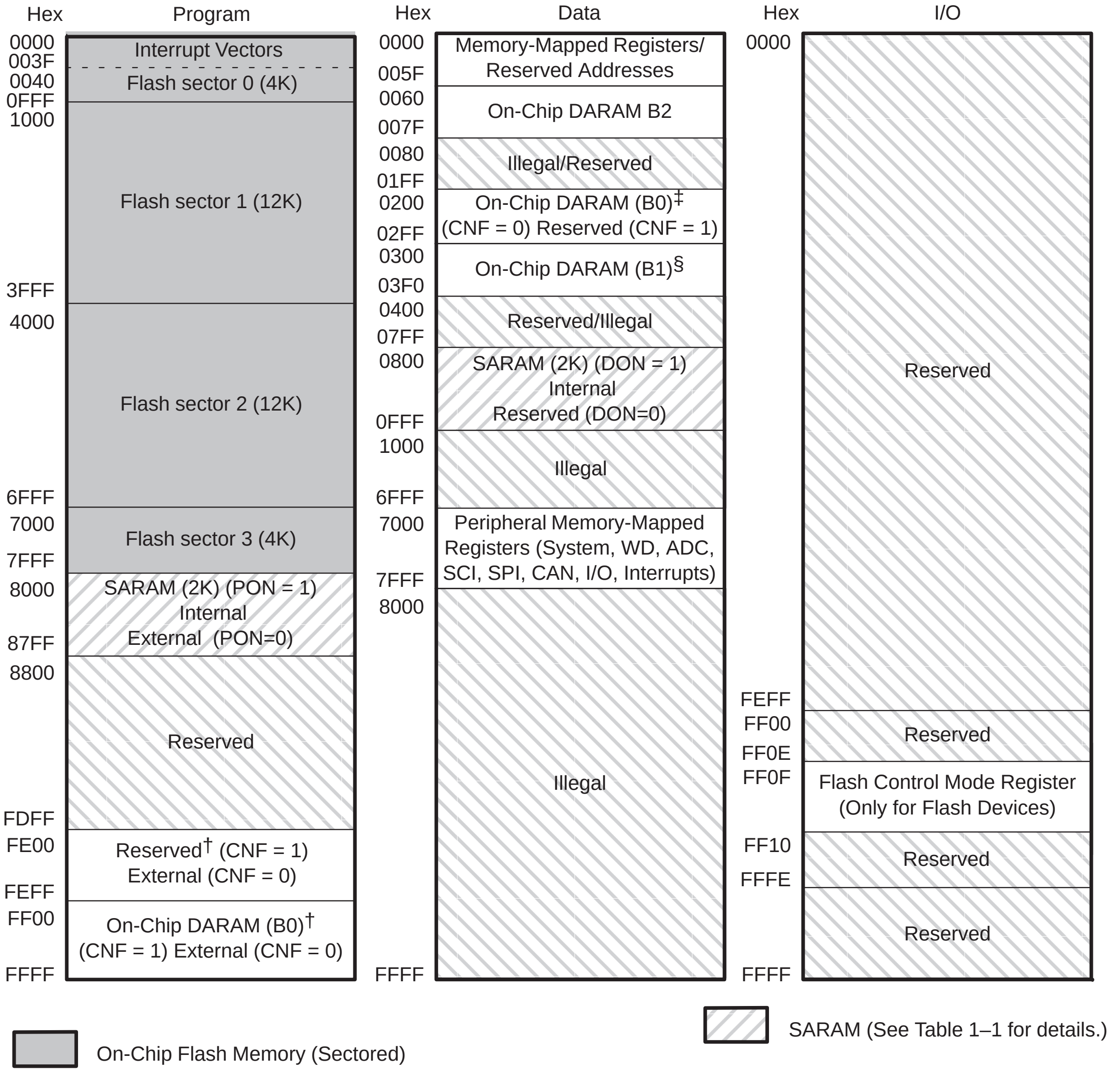
NOTE A: Boot ROM: If the boot ROM is enabled, then address 0000–00FF in the program space will be occupied by boot ROM.

[†] When CNF = 1, addresses FE00h–FEFFh and FF00h–FFFFh are mapped to the same physical block (B0) in program-memory space. For example, a write to FE00h has the same effect as a write to FF00h. For simplicity, addresses FE00h–FEFFh are referred to as reserved when CNF = 1.

[‡] When CNF = 0, addresses 0100h–01FFh and 0200h–02FFh are mapped to the same physical block (B0) in data-memory space. For example, a write to 0100h has the same effect as a write to 0200h. For simplicity, addresses 0100h–01FFh are referred to as reserved.

[§] Addresses 0300h–03FFh and 0400h–04FFh are mapped to the same physical block (B1) in data-memory space. For example, a write to 0400h has the same effect as a write to 0300h. For simplicity, addresses 0400h–04FFh are referred to as reserved.

Figure 3–2. Memory Map for ’LF2406


Hex
Program
0000
003F
0040
0FFF
1000
3FFF
4000
6FFF
7000
7FFF
8000
87FF
8800
FDFF
FE00
FEFF
FF00
FFFF
Interrupt Vectors
Flash sector 0 (4K)
Flash sector 1 (12K)
Flash sector 2 (12K)
Flash sector 3 (4K)
SARAM (2K) (PON = 1) Internal External (PON=0)
Reserved
Reserved† (CNF = 1) External (CNF = 0)
On-Chip DARAM (B0)† (CNF = 1) External (CNF = 0)
Hex
Data
0000
005F
0060
007F
0080
01FF
0200
02FF
0300
03F0
0400
07FF
0800
0FFF
1000
6FFF
7000
7FFF
8000
FFFF
Memory-Mapped Registers/ Reserved Addresses
On-Chip DARAM B2
Illegal/Reserved
On-Chip DARAM (B0)‡ (CNF = 0) Reserved (CNF = 1)
On-Chip DARAM (B1)§
Reserved/Illegal
SARAM (2K) (DON = 1) Internal Reserved (DON=0)
Illegal
Peripheral Memory-Mapped Registers (System, WD, ADC, SCI, SPI, CAN, I/O, Interrupts)
Illegal
Hex
I/O
0000
FEFF
FF00
FF0E
FF0F
FF10
FFFE
FFFF
Reserved
Reserved
Flash Control Mode Register (Only for Flash Devices)
Reserved
Reserved
SARAM (See Table 1–1 for details.)
On-Chip Flash Memory (Sectored)

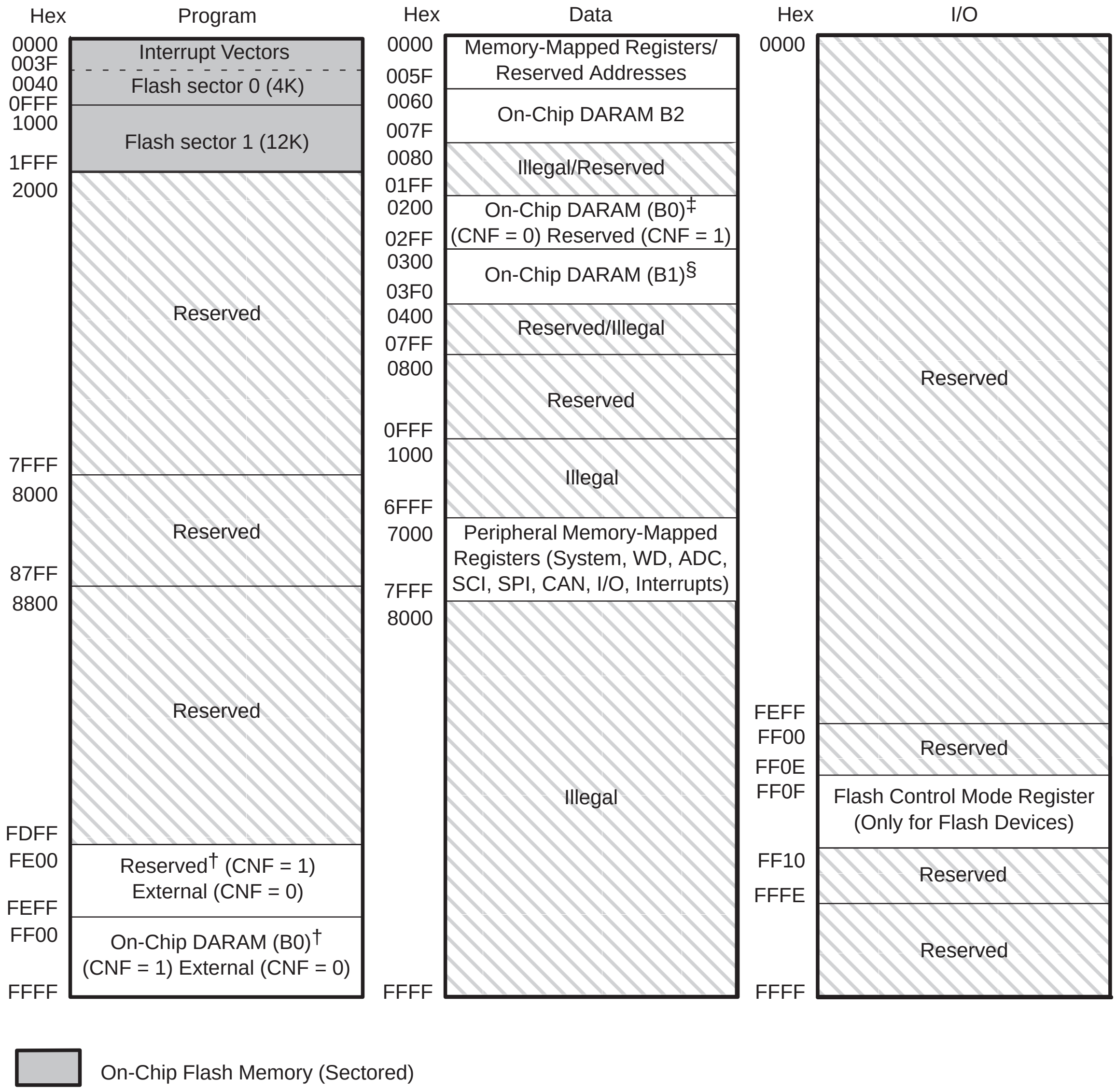
NOTE A: Boot ROM: If the boot ROM is enabled, then address 0000–00FF in the program space will be occupied by boot ROM.

† When CNF = 1, addresses FE00h–FEFFh and FF00h–FFFFh are mapped to the same physical block (B0) in program-memory space. For example, a write to FE00h has the same effect as a write to FF00h. For simplicity, addresses FE00h–FEFFh are referred to as reserved when CNF = 1.

‡ When CNF = 0, addresses 0100h–01FFh and 0200h–02FFh are mapped to the same physical block (B0) in data-memory space. For example, a write to 0100h has the same effect as a write to 0200h. For simplicity, addresses 0100h–01FFh are referred to as reserved.

§ Addresses 0300h–03FFh and 0400h–04FFh are mapped to the same physical block (B1) in data-memory space. For example, a write to 0400h has the same effect as a write to 0300h. For simplicity, addresses 0400h–04FFh are referred to as reserved.

Figure 3–3. Memory Map for ’LF2402



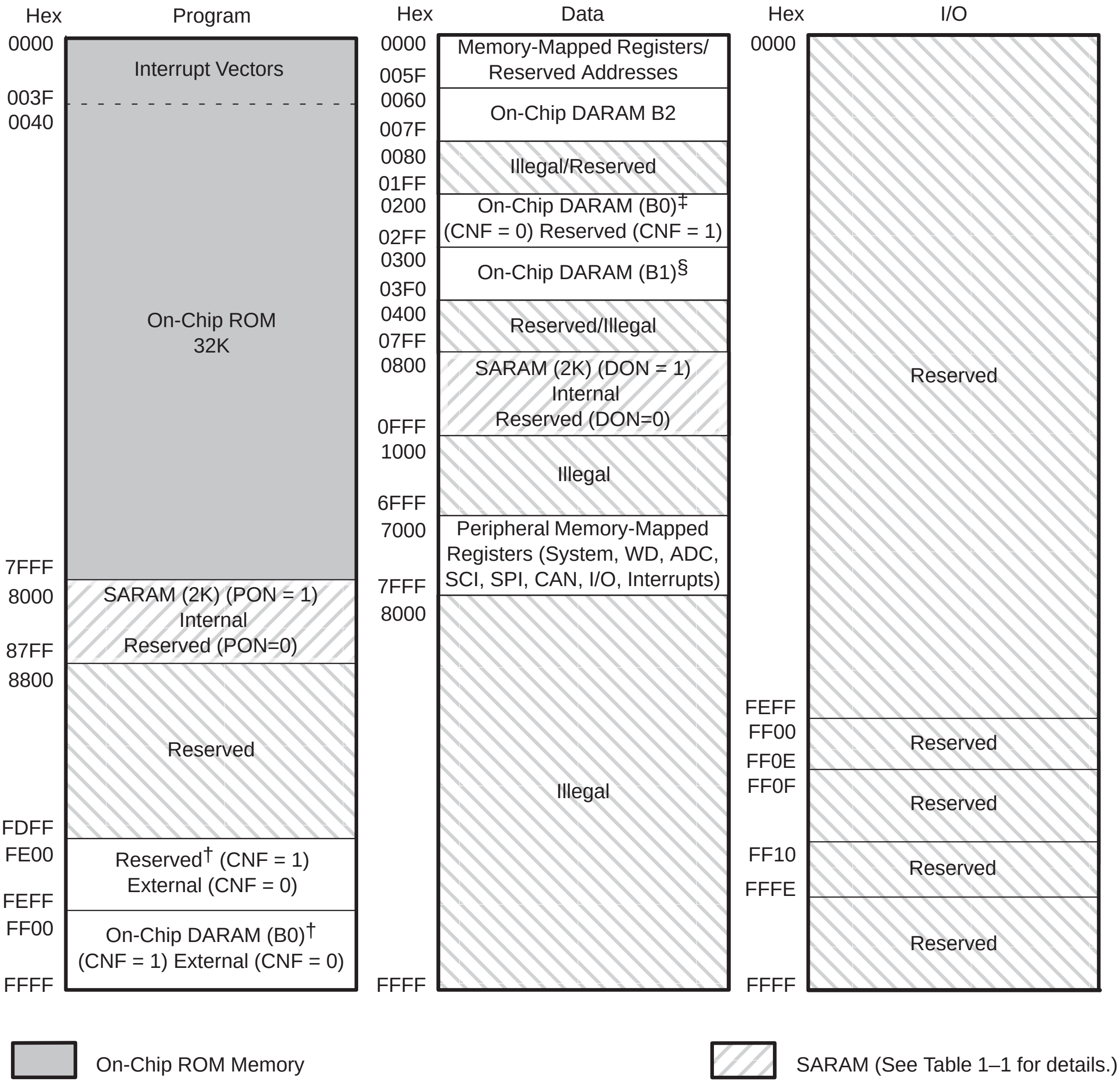
NOTE A: Boot ROM: If the boot ROM is enabled, then address 0000–00FF in the program space will be occupied by boot ROM.

[†] When CNF = 1, addresses FE00h–FEFFh and FF00h–FFFFh are mapped to the same physical block (B0) in program-memory space. For example, a write to FE00h has the same effect as a write to FF00h. For simplicity, addresses FE00h–FEFFh are referred to as reserved when CNF = 1.

[‡] When CNF = 0, addresses 0100h–01FFh and 0200h–02FFh are mapped to the same physical block (B0) in data-memory space. For example, a write to 0100h has the same effect as a write to 0200h. For simplicity, addresses 0100h–01FFh are referred to as reserved.

[§] Addresses 0300h–03FFh and 0400h–04FFh are mapped to the same physical block (B1) in data-memory space. For example, a write to 0400h has the same effect as a write to 0300h. For simplicity, addresses 0400h–04FFh are referred to as reserved.

Figure 3–4. Memory Map for 'LC2406

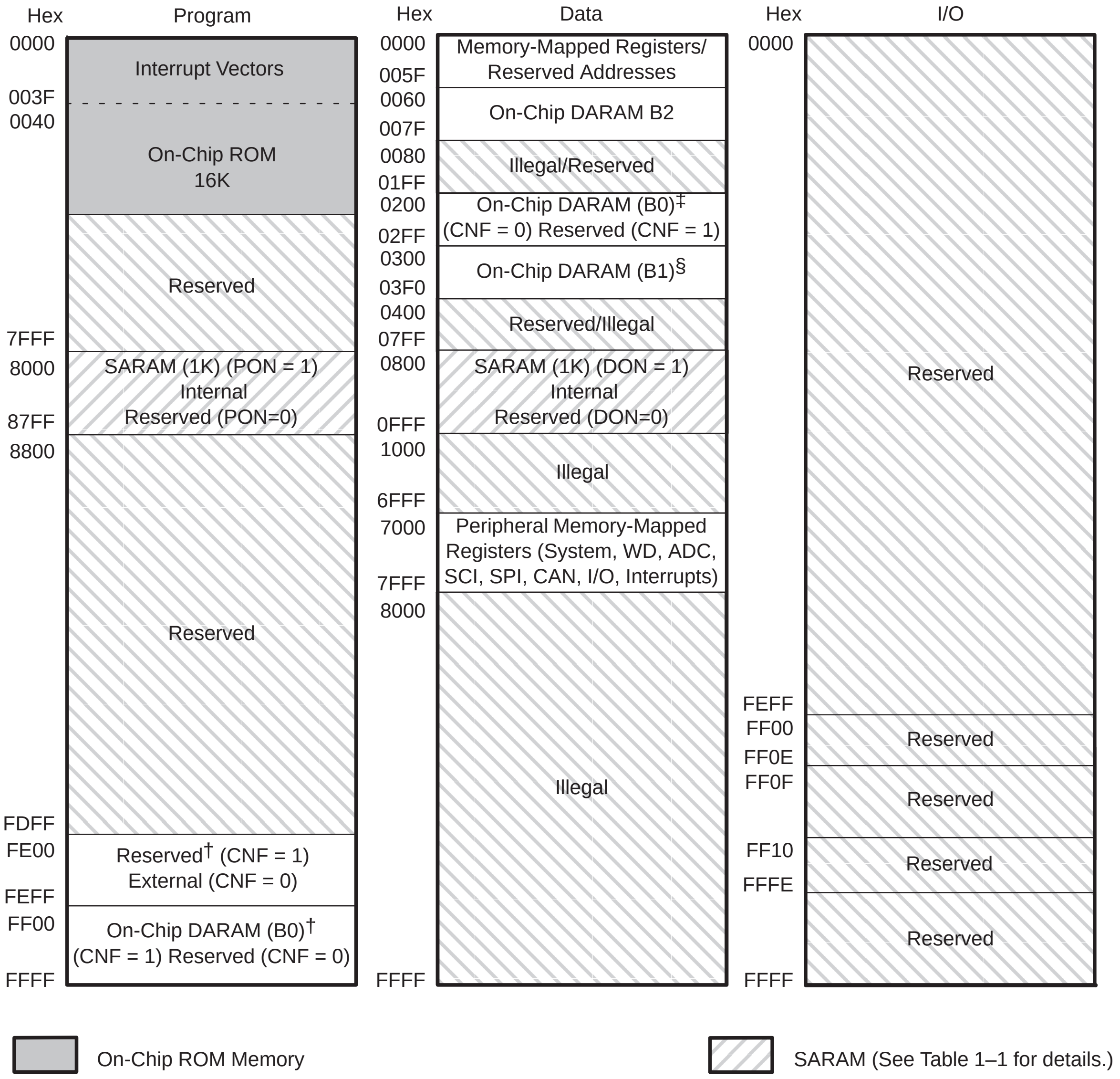


† When CNF = 1, addresses FE00h–FEFFh and FF00h–FFFFh are mapped to the same physical block (B0) in program-memory space. For example, a write to FE00h has the same effect as a write to FF00h. For simplicity, addresses FE00h–FEFFh are referred to as reserved when CNF = 1.

‡ When CNF = 0, addresses 0100h–01FFh and 0200h–02FFh are mapped to the same physical block (B0) in data-memory space. For example, a write to 0100h has the same effect as a write to 0200h. For simplicity, addresses 0100h–01FFh are referred to as reserved.

§ Addresses 0300h–03FFh and 0400h–04FFh are mapped to the same physical block (B1) in data-memory space. For example, a write to 0400h has the same effect as a write to 0300h. For simplicity, addresses 0400h–04FFh are referred to as reserved.

Figure 3–5. Memory Map for ’LC2404

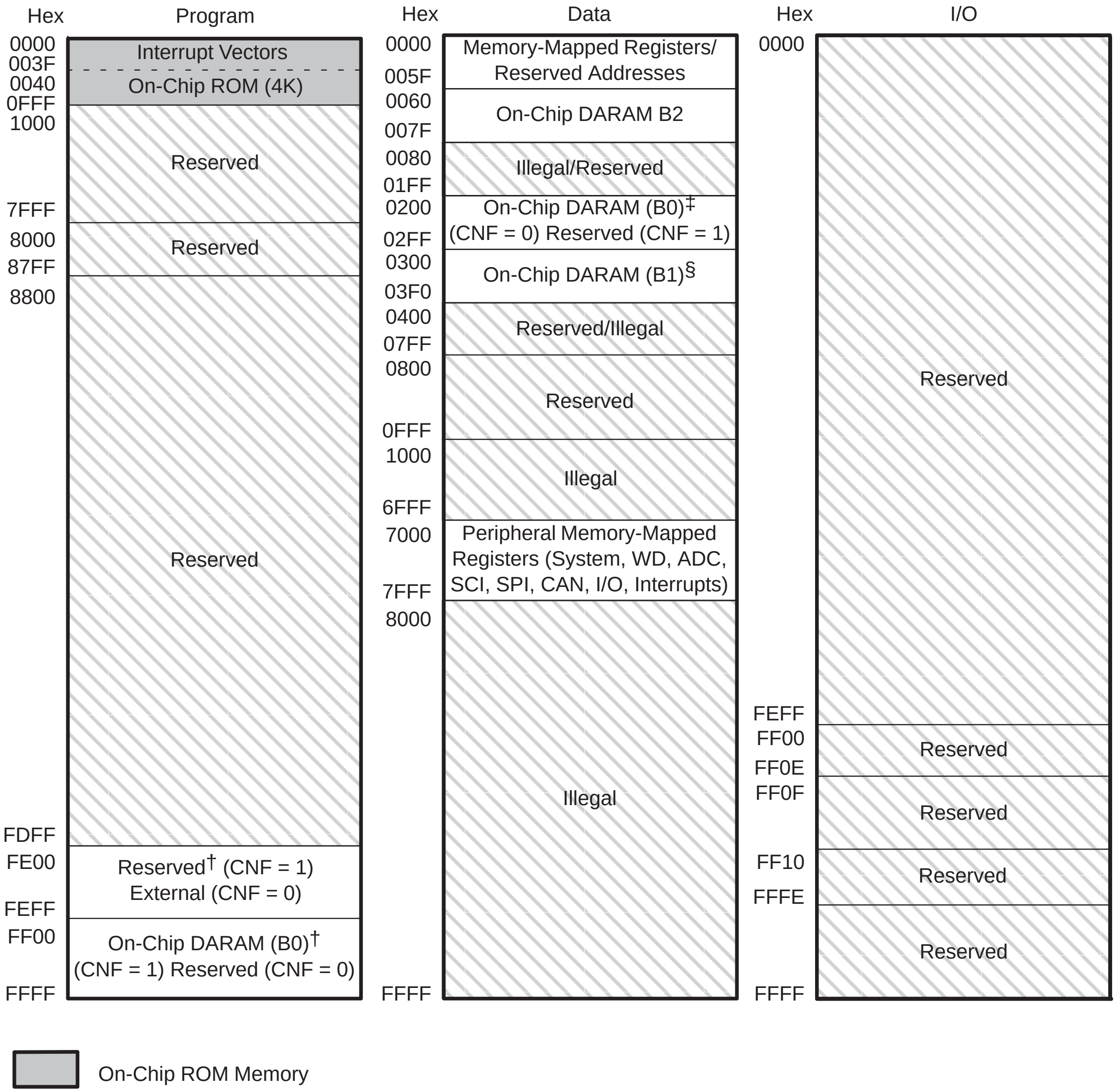

Hex
Program
0000
Interrupt Vectors
003F
0040
On-Chip ROM 16K
Reserved
7FFF
8000
SARAM (1K) (PON = 1) Internal Reserved (PON=0)
87FF
8800
Reserved
FDFF
FE00
Reserved† (CNF = 1) External (CNF = 0)
FEFF
FF00
On-Chip DARAM (B0)† (CNF = 1) Reserved (CNF = 0)
FFFF
Hex
Data
0000
Memory-Mapped Registers/ Reserved Addresses
005F
0060
On-Chip DARAM B2
007F
0080
Illegal/Reserved
01FF
0200
On-Chip DARAM (B0)‡ (CNF = 0) Reserved (CNF = 1)
02FF
0300
On-Chip DARAM (B1)§
03F0
0400
Reserved/Illegal
07FF
0800
SARAM (1K) (DON = 1) Internal Reserved (DON=0)
0FFF
1000
Illegal
6FFF
7000
Peripheral Memory-Mapped Registers (System, WD, ADC, SCI, SPI, CAN, I/O, Interrupts)
7FFF
8000
Illegal
FFFF
Hex
I/O
0000
Reserved
FEFF
FF00
Reserved
FF0E
FF0F
Reserved
FF10
Reserved
FFFE
Reserved
FFFF
On-Chip ROM Memory
SARAM (See Table 1–1 for details.)

[†] When CNF = 1, addresses FE00h–FEFFh and FF00h–FFFFh are mapped to the same physical block (B0) in program-memory space. For example, a write to FE00h has the same effect as a write to FF00h. For simplicity, addresses FE00h–FEFFh are referred to as reserved when CNF = 1.

[‡] When CNF = 0, addresses 0100h–01FFh and 0200h–02FFh are mapped to the same physical block (B0) in data-memory space. For example, a write to 0100h has the same effect as a write to 0200h. For simplicity, addresses 0100h–01FFh are referred to as reserved.

[§] Addresses 0300h–03FFh and 0400h–04FFh are mapped to the same physical block (B1) in data-memory space. For example, a write to 0400h has the same effect as a write to 0300h. For simplicity, addresses 0400h–04FFh are referred to as reserved.

Figure 3–6. Memory Map for ’LC2402



[†] When CNF = 1, addresses FE00h–FEFFh and FF00h–FFFFh are mapped to the same physical block (B0) in program-memory space. For example, a write to FE00h has the same effect as a write to FF00h. For simplicity, addresses FE00h–FEFFh are referred to as reserved when CNF = 1.

[‡] When CNF = 0, addresses 0100h–01FFh and 0200h–02FFh are mapped to the same physical block (B0) in data-memory space. For example, a write to 0100h has the same effect as a write to 0200h. For simplicity, addresses 0100h–01FFh are referred to as reserved.

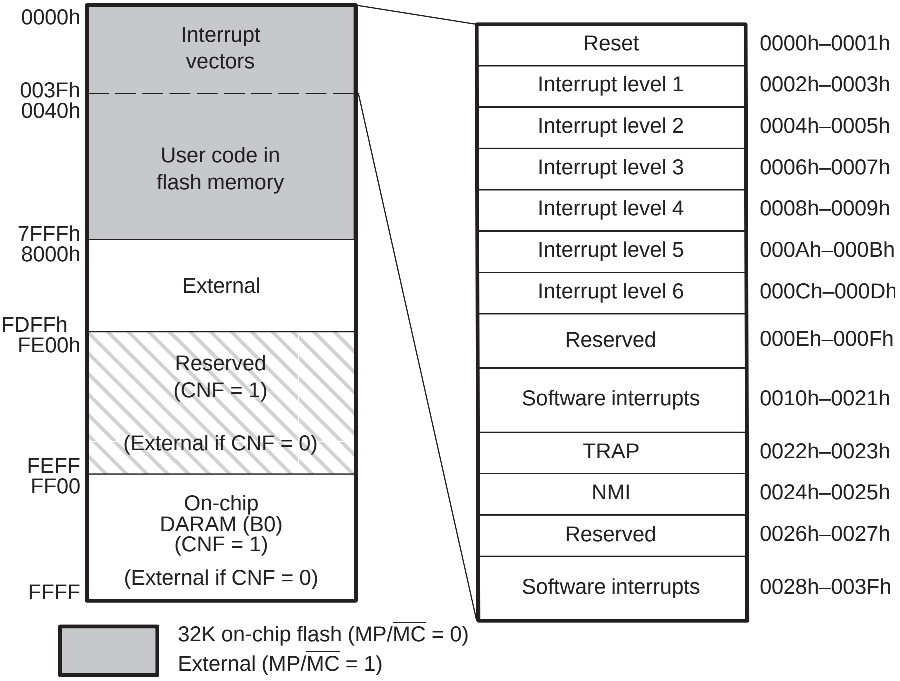
[§] Addresses 0300h–03FFh and 0400h–04FFh are mapped to the same physical block (B1) in data-memory space. For example, a write to 0400h has the same effect as a write to 0300h. For simplicity, addresses 0400h–04FFh are referred to as reserved.

3.4 Program Memory

In addition to storing the user code, the program memory also stores immediate operands and table information. A maximum of 64K 16-bit words can be addressed in the program memory for ’240x. This number includes on-chip DARAM and flash EEPROM/ROM. Whenever an off-chip memory location needs to be accessed, the appropriate control signals for external access ($\overline{\text{PS}}$, $\overline{\text{DS}}$, $\overline{\text{STRB}}$, etc.) are automatically generated.

Figure 3–7 shows the ’LF2407 program memory map.

Figure 3–7. Program Memory Map for ’LF2407


0000h
Interrupt vectors
003Fh
0040h
User code in flash memory
7FFFh
8000h
External
FDFFh
FE00h
Reserved (CNF = 1)
(External if CNF = 0)
FEFF
FF00
On-chip DARAM (B0) (CNF = 1)
(External if CNF = 0)
FFFF
32K on-chip flash (MP/$\overline{\text{MC}}$ = 0)
External (MP/$\overline{\text{MC}}$ = 1)

Reset	0000h–0001h
Interrupt level 1	0002h–0003h
Interrupt level 2	0004h–0005h
Interrupt level 3	0006h–0007h
Interrupt level 4	0008h–0009h
Interrupt level 5	000Ah–000Bh
Interrupt level 6	000Ch–000Dh
Reserved	000Eh–000Fh
Software interrupts	0010h–0021h
TRAP	0022h–0023h
NMI	0024h–0025h
Reserved	0026h–0027h
Software interrupts	0028h–003Fh

3.4.1 Program Memory Configuration

Two factors determine the configuration of program memory:

❑ **CNF bit.** The CNF bit (bit 12) of status register ST1 determines whether DARAM B0 is in on-chip program space:

- **CNF = 0.** The 256 words are mapped as external memory.
- **CNF = 1.** The 256 words of DARAM B0 are configured for program use. At reset, B0 is mapped to data space (CNF = 0).

- **MP/$\overline{\textbf{MC}}$ pin.** The level on the MP/$\overline{\text{MC}}$ pin determines whether program instructions are read from on-chip flash/ROM or external memory:

  - **MP/$\overline{\textbf{MC}}$ = 0.** The device is configured in microcontroller mode. The on-chip ROM/flash EEPROM is accessible. The device fetches the reset vector from on-chip memory. Accesses to program memory addresses 0000h–7FFFh will be made to on-chip memory in the case of ’2407.

  - **MP/$\overline{\textbf{MC}}$ = 1.** The device is configured in microprocessor mode. The device fetches the reset vector from external memory. Accesses to program memory addresses 0000h–7FFFh will be made to off-chip memory.

  Regardless of the value of MP/$\overline{\text{MC}}$, the ’240x fetches its reset vector at location 0000h in program memory. Note that there is no MP/$\overline{\text{MC}}$ pin available on devices that lack an external memory interface.

3.5 Data Memory

Data memory space addresses up to 64K of 16-bit words. 32K words are internal memory (0000h to 7FFFh). Internal data memory includes memory-mapped registers, DARAM, and peripheral memory-mapped registers. The remaining 32K words of memory (8000h to FFFFh) form part of the external data memory. Note that addresses 8000h–FFFFh are not accessible in ’2406, ’2404, and ’2402.

Figure 3–8 shows the data memory map for the ’2407. Each device has three on-chip DARAM blocks: B0, B1, and B2. B0 is configurable as data memory or program memory. It is the same memory block accessible either as data memory or program memory, depending on the CNF bit. Blocks B1 and B2 are available for data memory only. External data memory is available only on the ’2407.

Data memory can be addressed with either of two addressing modes: direct-addressing or indirect-addressing.

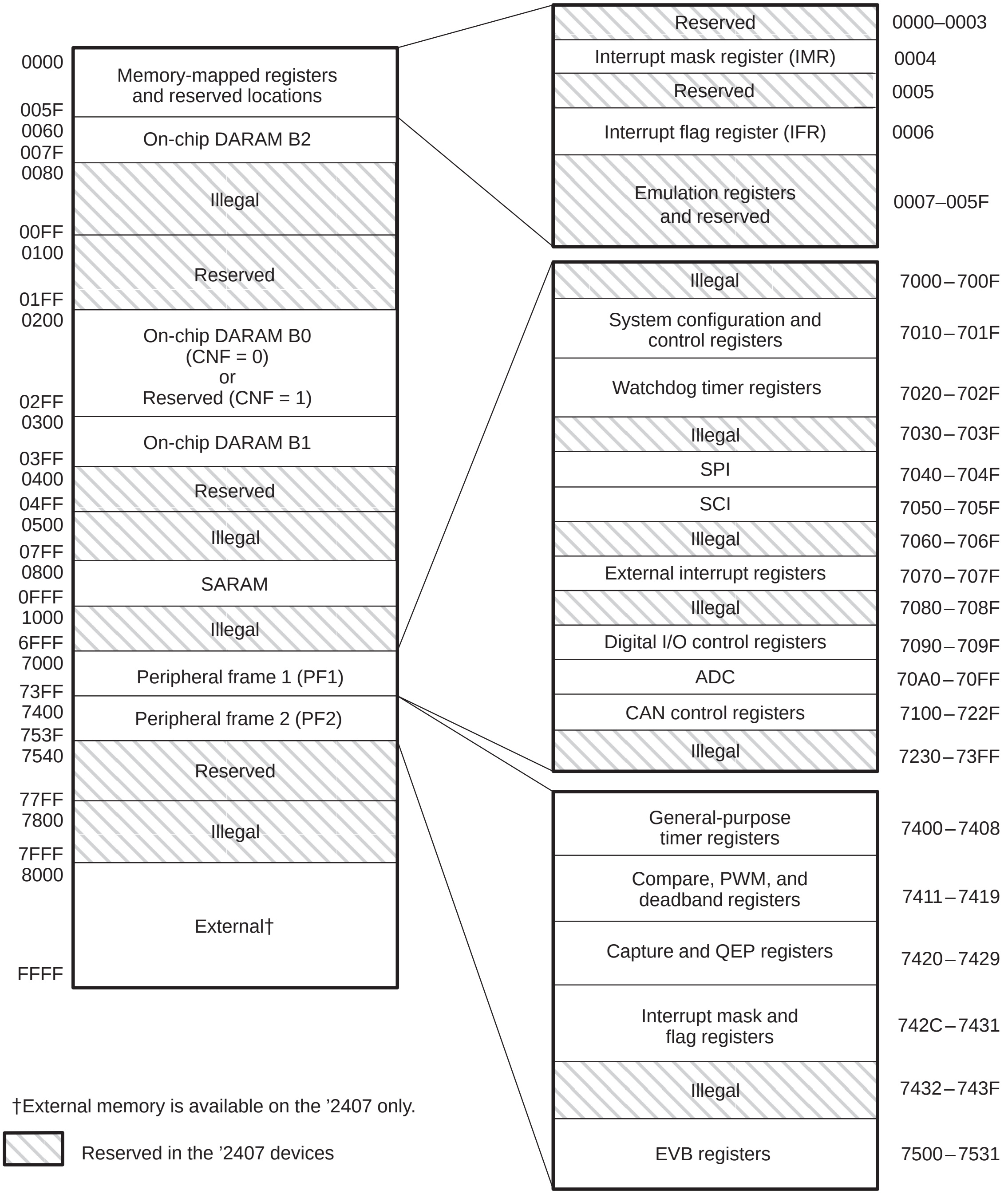
When direct addressing is used, data memory is addressed in blocks of 128 words called data pages. Figure 3–9 shows how these blocks are addressed. The entire 64K of data memory consists of 512 data pages labeled 0 through 511. The current data page is determined by the value in the 9-bit data page pointer (DP) in status register ST0. Each of the 128 words on the current page is referenced by a 7-bit offset taken from the instruction that is using direct addressing. Therefore, when an instruction uses direct addressing, you must specify both the data page (with a preceding instruction) and the offset (in the instruction that accesses data memory).

An access to the following address spaces in the data memory is illegal and generates a NMI. In addition to these addresses, an access to any of the reserved addresses within the peripheral register maps is also illegal.

0080h–00FFh
0500h–07FFh
1000h–700Fh
7030h–703Fh
7060h–706Fh
7080h–708Fh
70C0h–70FFh

710Fh–71FFh (inside CAN)
7230h–73FFh (partly inside CAN)
7440h–74FFh
7540h–75FFh
7600h–7FFFh
8000h–FFFFh (on ’2406, ’2404, and ’2402 only)

Figure 3–8. ’2407 Peripheral Memory Map



0000	005F	Memory-mapped registers and reserved locations
0060	007F	On-chip DARAM B2
0080	00FF	Illegal
0100	01FF	Reserved
0200	02FF	On-chip DARAM B0 (CNF = 0) or Reserved (CNF = 1)
0300	03FF	On-chip DARAM B1
0400	04FF	Reserved
0500	07FF	Illegal
0800	0FFF	SARAM
1000	6FFF	Illegal
7000	73FF	Peripheral frame 1 (PF1)
7400	753F	Peripheral frame 2 (PF2)
7540	77FF	Reserved
7800	7FFF	Illegal
8000	FFFF	External†

Reserved	0000–0003
Interrupt mask register (IMR)	0004
Reserved	0005
Interrupt flag register (IFR)	0006
Emulation registers and reserved	0007–005F

Illegal	7000–700F
System configuration and control registers	7010–701F
Watchdog timer registers	7020–702F
Illegal	7030–703F
SPI	7040–704F
SCI	7050–705F
Illegal	7060–706F
External interrupt registers	7070–707F
Illegal	7080–708F
Digital I/O control registers	7090–709F
ADC	70A0–70FF
CAN control registers	7100–722F
Illegal	7230–73FF

General-purpose timer registers	7400–7408
Compare, PWM, and deadband registers	7411–7419
Capture and QEP registers	7420–7429
Interrupt mask and flag registers	742C–7431
Illegal	7432–743F
EVB registers	7500–7531

†External memory is available on the ’2407 only.

▨ Reserved in the ’2407 devices

Figure 3–9. Data Memory Pages

DP Value	Offset	Data Memory
0000 0000 0	000 0000	Page 0: 0000h–007Fh
:	:	
0000 0000 0	111 1111	
0000 0000 1	000 0000	Page 1: 0080h–00FFh
:	:	
0000 0000 1	111 1111	
0000 0001 0	000 0000	Page 2: 0100h–017Fh
:	:	
0000 0001 0	111 1111	
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
1111 1111 1	000 0000	Page 511: FF80h–FFFFh
:	:	
1111 1111 1	111 1111	

Data Page 0 Address Map

The data memory also includes the device’s memory-mapped registers (MMR), which reside at the top of data page 0 (addresses 0000h–007Fh). Note the following:

- ❑ The two registers that can be accessed with zero wait states are:
  - ■ Interrupt mask register (IMR)
  - ■ Interrupt flag register (IFR)
- ❑ The test/emulation reserved area is used by the test and emulation systems for special information transfers.

CAUTION

Do Not Write to Test/Emulation Addresses

Writing to the test/emulation addresses can cause the device to change its operating mode, and therefore, affect the operation of an application.

- ❑ The scratch-pad RAM block (B2) includes 32 words of DARAM that provide for variable storage without fragmenting the larger RAM blocks, whether internal or external. This RAM block supports dual-access operations and can be addressed via any data-memory addressing mode.

Table 3–1 shows the address map of data page 0.

Table 3–1. Data Page 0 Address Map

Address	Name	Description
0000h–0003h	–	Reserved
0004h	IMR	Interrupt mask register
0005h	–	Reserved
0006h	IFR	Interrupt flag register
0023h–0027h	–	Reserved
002Bh–002Fh	–	Reserved for test/emulation
0060h–007Fh	B2	Scratch-pad RAM (DARAM B2)

Data Memory Configuration

Two factors that contribute to the configuration of data memory are:

- ❑ **CNF bit.** The CNF bit (bit 12) of status register ST1 determines whether the on-chip DARAM B0 is mapped to data space or program space.
  - ■ **CNF = 1.** B0 is used for program space.
  - ■ **CNF = 0.** B0 is used for data space.

  At reset, B0 is mapped into data space (CNF = 0).

3.5.1 Global Data Memory

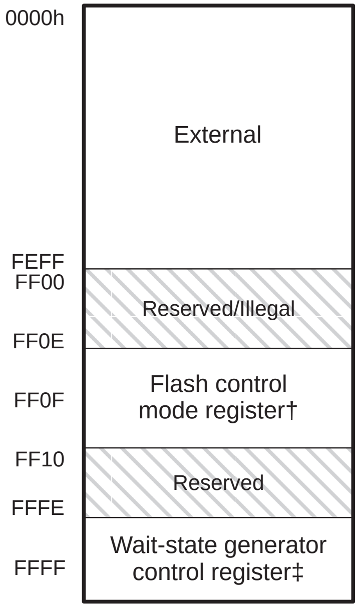
Note:

Global Data Memory is not available in ’240x. Hence, the global memory allocation register (GREG) is a reserved location and should not be accessed.

3.6 I/O Space

The I/O-space memory addresses up to 64K 16-bit words. Figure 3–10 shows the I/O space address map for the ’240x.

Figure 3–10. I/O Space Address Map for ’240x


0000h
External
FEFF
FF00
Reserved/Illegal
FF0E
Flash control mode register†
FF0F
FF10
Reserved
FFFE
Wait-state generator control register‡
FFFF

† Available only on Flash devices.
‡ Available only on devices with external memory interface.

3.7 XMIF Qualifier Signal Description

The ’240x can address the following memory sizes in each of the external memory spaces:

Ext. Memory Space	Size (in words)	Qualifier signal (strobe)
Program space	64K	$\overline{\text{PS}}$
Data space	64K	$\overline{\text{DS}}$
I/O space	64K	$\overline{\text{IS}}$

The signals that define the XMIF are given in Table 3–2.

Table 3–2. XMIF Signal Descriptions

Signal/s name	Signal description
A(0:15)	External 16-bit unidirectional address bus.
D(0:15)	External 16-bit bidirectional data bus.
$\overline{\text{PS}}$	Program space strobe
$\overline{\text{DS}}$	Data space strobe
$\overline{\text{IS}}$	I/O space strobe
$\overline{\text{STRB}}$	External memory access strobe
$\overline{\text{WE}}$	Write strobe
$\overline{\text{RD}}$	Read strobe
R/$\overline{\text{W}}$	Read / Write qualifier
MP/$\overline{\text{MC}}$	Microprocessor/microcontroller selection pin
$\overline{\text{VIS\_OE}}$	Is active low whenever the external data bus is driving as an output during visibility mode. Can be used by external decode logic to prevent data bus contention while running in visibility mode
ENA_144	If pulled low, the ’2407 device behaves like a ’2402/2404/2406; that is, has no external memory and generates an Illegal address if any of the 3 external spaces are accessed. This pin has an internal pull-down resistor, so when left disconnected, device behaves appropriately.

Note: These signals allow external memory such as SRAM to be interfaced to the ’240x in the conventional way.

Figure 3–11 and Figure 3–12 show *Visibility* mode timing diagrams.

Figure 3–11. Program Address/Data — Visibility Functional Timing

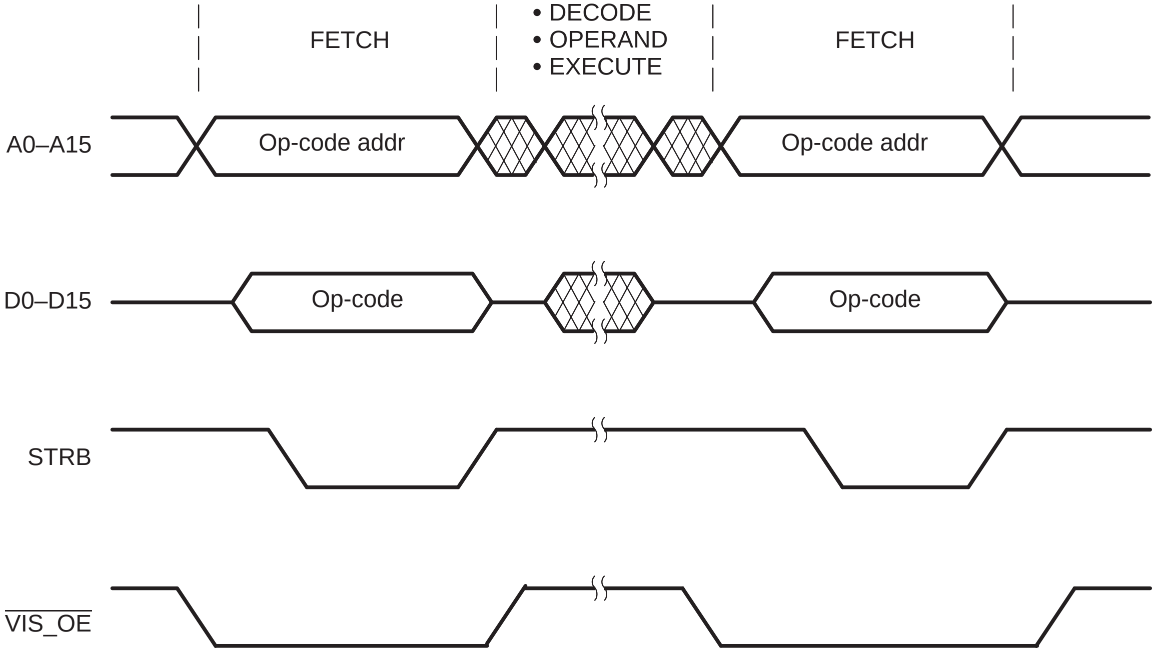
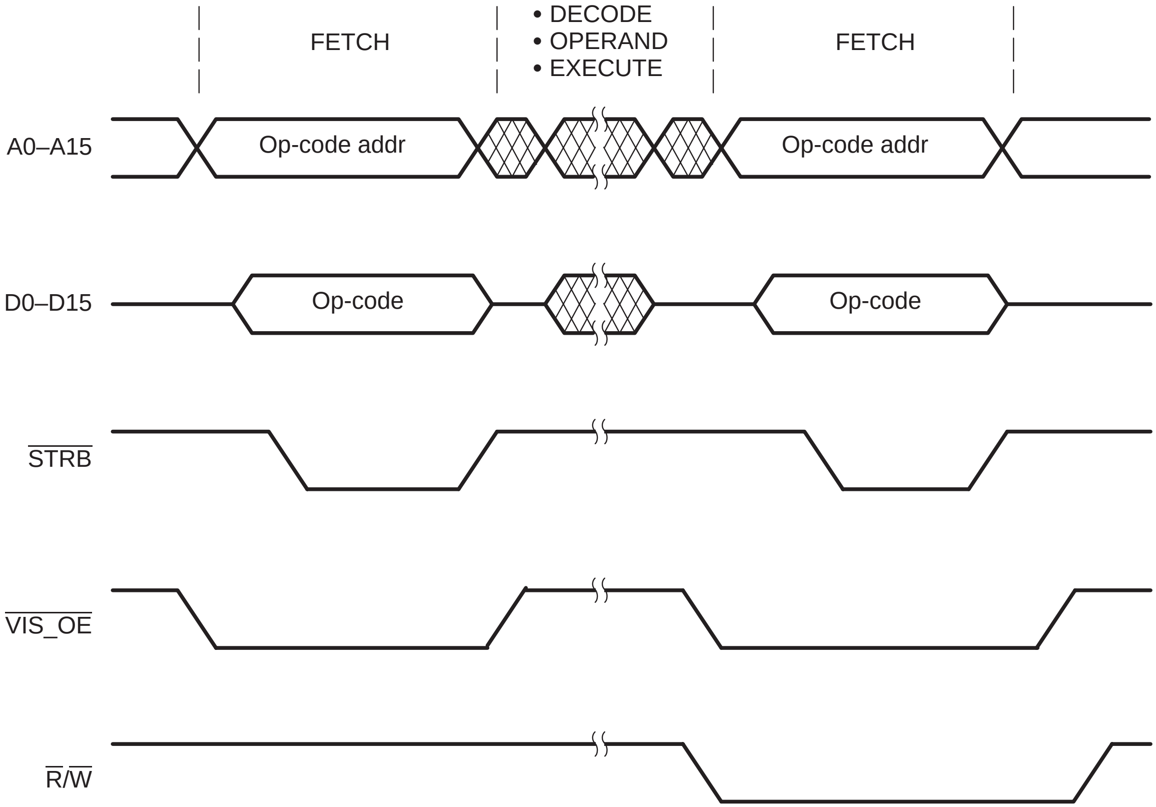


Figure 3–12. Data Address/ Data — Visibility Functional Timing



3.8 Program and Data Spaces

$\overline{\text{PS}}$ and $\overline{\text{STRB}}$ are inactive (high) for accesses to on-chip program memory and data memory. The external data and address busses are active only when accesses are made to external memory locations, except when in bus visibility (BVIS) mode (see section 3.10, *Wait-State Generation*).

Two cycles are required on all external writes, including a half-cycle before $\overline{\text{WE}}$ goes low and a half-cycle after $\overline{\text{WE}}$ goes high. This prevents data contention on the external buses.

3.9 I/O Space

I/O space accesses are distinguished from program and data memory accesses by $\overline{\text{IS}}$ going low. All 64K I/O words (external I/O port and on-chip I/O registers) are accessed via the IN and OUT instructions.

While accesses are made to the on-chip I/O mapped registers, signals $\overline{\text{IS}}$ and $\overline{\text{STRB}}$ are made inactive, that is, driven to the high state. The external address and data bus is only active when accesses are made to external I/O memory locations.

Two cycles are required on all external writes, including a half-cycle before $\overline{\text{WE}}$ goes low and a half-cycle after $\overline{\text{WE}}$ goes high. This prevents data contention on the external busses.

3.10 Wait-State Generation

Wait states are necessary when you want to interface the ’240x with slower external logic and memory. By adding wait states, you lengthen the time the CPU waits for external memory or an external I/O port to respond when the CPU reads from or writes to that memory or port. Specifically, the CPU waits one extra cycle (one CLKOUT cycle) for every wait state. The wait states operate on CLKOUT cycle boundaries.

To avoid bus conflicts, writes from the ’240x always take at least two CLKOUT cycles. The ’240x offers two options for generating wait states:

- **The READY signal**. With the READY signal, you can externally generate any number of wait states.
- **The on-chip wait-state generator**. With this generator, you can generate zero to seven wait states.

3.10.1 Generating Wait States With the READY Signal

When READY is low, the ’240x waits one CLKOUT cycle and checks READY again. The ’240x will not continue executing until READY is driven high; therefore, if the READY signal is not used, it should be pulled high during external accesses.

The READY pin can be used to generate any number of wait states. However, when the ’240x operates at full speed, it cannot respond fast enough to provide a READY-based wait state for the first cycle. For extended wait states using external READY logic, the on-chip wait-state generator must be programmed to generate at least one wait state.

Note: The READY pin has no effect on accesses to *internal* memory.

3.10.2 Generating Wait States With the ’240x Wait-State Generator

The software wait-state generator can be programmed to generate zero to seven wait states for a given off-chip memory space (program, data, or I/O), regardless of the state of the READY signal. This wait-state generator has the bit fields shown in Figure 3–13 and described after the figure.

Figure 3–13. ’240x Wait-State Generator Control Register (WSGR) — I/O-Space Address FFFFh (’240x)

15–11	10–9	8–6	5–3	2–0
Reserved	BVIS	ISWS	DSWS	PSWS
0	W-11	W-111	W-111	W-111

Note: 0 = Always read as zeros: W = Write access: -n = value after reset

Bits 15–11 **Reserved**. Bits 15–11 are reserved and always read as 0s.

Bits 10–9 **Bus visibility modes.** Bits 10–9 allow selection of various bus visibility modes while running from internal program and/or data memory. These modes provide a method of tracing internal bus activity.

Bit 10	Bit 9	Visibility mode
0	0	Bus visibility OFF (reduces power and noise)
0	1	Bus visibility OFF (reduces power and noise)
1	0	Data-address bus output to external address bus Data-data bus output to external data bus
1	1	Program-address bus o/p to external address bus Program-data bus output to external data bus

Bits 8–6 **ISWS — I/O-space wait-state bits**. Bits 8-6 determine the number of wait states (0–7) that are applied to reads from and writes to off-chip I/O space. At reset, the three ISWS bits become 111, setting seven wait states for reads from and writes to off-chip I/O space.

Bits 5–3 **DSWS — Data-space wait-state bits**. Bits 5–3 determine the number of wait states (0–7) that are applied to reads from and writes to off-chip data space. At reset, the three DSWS bits become 111, setting seven wait states for reads from and writes to off-chip data space.

Bits 2–0 **PSWS — Program-space wait-state bits**. Bits 2-0 determine the number of wait states (0–7) that are applied to reads from and writes to off-chip program space. At reset, the three PSWS bits become 111, setting seven wait states for reads from and writes to off-chip program space.

Table 3–3 shows how to set the number of wait states you want for each type of off-chip memory.

Table 3–3. Setting the Number of Wait States With the ’240x WSGR Bits

ISWS Bits				DSWS				PSWS			
8	**7**	**6**	**I/O WS**	**5**	**4**	**3**	**Data WS**	**2**	**1**	**0**	**Prog WS**
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	1	0	0	1	1
0	1	0	2	0	1	0	2	0	1	0	2
0	1	1	3	0	1	1	3	0	1	1	3
1	0	0	4	1	0	0	4	1	0	0	4
1	0	1	5	1	0	1	5	1	0	1	5
1	1	0	6	1	1	0	6	1	1	0	6
1	1	1	7	1	1	1	7	1	1	1	7

In summary, while the READY signal remains high,the wait-state generator inserts from zero to seven wait states to a given memory space, depending on the values of PSWS, DSWS, and ISWS. The READY signal may then be driven low to generate additional wait states. If m is the number of CLKOUT cycles required for a particular read or write operation and w is the number of wait states added, the operation will take (m + w) cycles. At reset, all WSGR bits are set to 1, making seven wait states the default for every memory space.

Chapter 4

Clocks

The ’240x devices use the phase-locked loop (PLL) circuit embedded in the ’240x CPU core to synthesize the on-chip clocks from a lower frequency external clock. There is no means of bypassing the PLL.

Topic **Page**

4.1 Pins

There are three device pins associated with clocks:

❏ XTAL1/CLKIN – This is the clock input from the external crystal to the on-chip oscillator. If an external oscillator is used, its output must be connected to this pin.

❏ XTAL2 – This is the clock output from the on-chip oscillator to drive the external crystal.

❏ CLKOUT/IOPE0 – This is the clock output pin. It is multiplexed with GPIO pin IOPE0. This pin can be used to output the device (CPU) clock or the watchdog timer clock. The clock select control bits are in the “System Control and Status Register 1 (SCSR1)”, described in section 2.2.1 on page 2-3. This pin is configured to output CLKOUT from the CPU following a device reset.

4.2 Phase-Locked Loop

The PLL used in the ’240x devices is different than the one used in the ’24x devices. The ’240x PLL supports multiplication factors ranging from 0.5 to 4 times the input clock frequency. The ’240x PLL also needs external R, C components. (See device data sheet for more details.) A bypass capacitor (0.1 µF to 0.01 µF, ceramic) should be connected between the PLLV$_{CCA}$ and V$_{SS}$ pins. All PCB traces pertaining to the PLL circuit must be kept as short as possible. In addition, the loop area formed by the loop filter components, PCB traces, and DSP chip should be as small as possible. The lead lengths of the loop filter components must be kept as short as possible.

4.3 Watchdog Timer Clock

A low frequency clock, WDCLK, is used to clock the watchdog timer. WDCLK has a nominal frequency of 58593.8 Hz when CPUCLK = 30 MHz. WDCLK is derived from the CLKOUT of the CPU. This ensures that the watchdog timer continues to count when the CPU is in IDLE1 or IDLE 2 mode (see section 4.4, *Low-Power Modes*, on page 4-3).

The WDCLK is generated in the watchdog timer peripheral.

$$\text{WDCLK} = \frac{\text{CLKOUT}}{512}$$

4.3.1 Watchdog Suspend

WDCLK is stopped when the CPU's suspend signal goes active. This is achieved by stopping the clock input to the clock divider which generates WDCLK from CLKIN.

4.4 Low-Power Modes

The ’240x has an IDLE instruction. When executed, the IDLE instruction stops the clocks to all circuits in the CPU; however, the clock output from the CPU continues to run. With this instruction, the CPU clocks can be shut down to save power. The CPU exits the IDLE state if reset, or if it receives an interrupt request.

4.4.1 Clock Domains

All ’240x-based devices have two clock domains:

❑ The CPU clock domain consists of the clock for most of the CPU logic.

❑ The System clock domain consists of the peripheral clock (which is derived from CLKOUT of the CPU) and the clock for the interrupt logic in the CPU.

When the CPU goes into IDLE mode, the CPU clock domain is stopped while the system clock domain continues to run. This mode is also known as IDLE1 mode. The ’240x CPU also contains support for a second IDLE mode, IDLE2, implemented in external logic. By asserting the IDLE2 input to the ’240x CPU, both the CPU clock domain and the system clock domain are stopped, allowing further power savings. A third low-power mode, HALT mode, which is the deepest mode, is possible if the oscillator and WDCLK are also shut down. In HALT mode, the input clock to the PLL is shut off.

There are two control bits, LPM (1:0) that specify which of the three possible low-power modes is entered when the IDLE instruction is executed. This is described in Table 4–1. These bits are located in the “System Control and Status Register 1 (SCSR1)”, which is described in section 2.2.1, on page 2-3.

Table 4–1. Low-Power Modes Summary

Low-Power Mode	LPMx Bits SCSR[12:13]	CPU Clock Domain	System Clock Domain	WDCLK Status	PLL Status	OSC Status	Exit Condition
CPU running normally	XX	On	On	On	On	On	—
IDLE1 – (LPM0)	00	Off	On	On	On	On	Peripheral interrupts, XINT1/2, Reset, PDPINTA/B
IDLE2 – (LPM1)	01	Off	Off	On	On	On	Wakeup interrupts, Watchdog, XINT1/2, Reset, PDPINTA/B
HALT – (LPM2) {PLL/OSC power down}	1X	Off	Off	Off	Off	Off	Watchdog, Reset, PDPINTA/B

4.4.2 Wake Up from Low-Power Modes

4.4.2.1 Reset

A reset (from any source) causes the device to exit any of the Idle modes. If the device is halted, the reset initially starts the oscillator; however, initiation of the CPU reset sequence may be delayed while the oscillator powers up before clocks are generated.

4.4.2.2 External interrupts

The external interrupts, XINTx, can cause the device to exit any of the low-power modes, except HALT. If the device is in IDLE2 mode, the synchronous logic connected to the external interrupt pins is bypassed with combinatorial logic that recognizes the interrupt on the pin, starts the clocks, and then allows the clocked logic to generate an interrupt request to the PIE controller.

4.4.2.3 Wake Up Interrupts

Some peripherals have the capability to start the device clocks and then generate an interrupt in response to certain external events, such as activity on a communication line. As an example, the CAN wake up interrupt can assert the CAN error interrupt request even when there are no clocks running.

4.4.2.4 Peripheral interrupts

All peripheral interrupts, if enabled locally and globally, can cause the device to exit IDLE1 mode. INTM must be enabled for LPM operation. If the IMR bits are not enabled, the device “wakes up” from LPM mode and executes the next instruction. Since no ISRs are executed, the peripheral flags must be cleared.

For example, when XINT1 is used to “wake up” the device from LPM0, two things can happen based on how the XINT1 interrupt is configured. If the XINT1 interrupt is enabled (by setting the appropriate bit in the XINT1CR register and bit 0 in IMR is set to 1) and INTM bit is zero, a valid XINT1 signal will first take the device out of LPM0 and will also force the device to the appropriate interrupt vector. However, if the XINT1 interrupt is not enabled (by virtue of INTM bit or bit 0 of IMR), upon a XINT1 interrupt, the DSP would “wake up” and continue executing the instruction following the IDLE instruction.

Chapter 5

Digital Input/Output (I/O)

The digital I/O ports module provides a flexible method for controlling both dedicated I/O and shared pin functions. All I/O and shared pin functions are controlled using nine 16-bit registers. These registers are divided into two types:

- ❑ I/O MUX Control registers (MCRx) – Used to control the multiplexor selection that chooses between the primary function of a pin or the general-purpose I/O function.
- ❑ Data and Direction Control registers (PxDATDIR) – Used to control the data and data direction of bidirectional I/O pins.

The GPIO pins are controlled through “data-memory mapped registers.” Note that there is no relationship between the GPIO pins and the I/O space of the device.

Topic **Page**

5.1 Digital I/O Ports Register Implementation on ’240x Devices

Table 5–1 lists the registers available to the digital I/O module as implemented on the ’240x devices. These registers are memory-mapped to data space from 7090h through 709Fh. All reserved registers and bits are unimplemented: reads return zero and writes have no effect.

Note that when multiplexed I/O pins are configured for peripheral functions or as GPIO outputs, the pin status can be monitored by reading the I/O data register.

Figure 5–1. Shared Pin Configuration

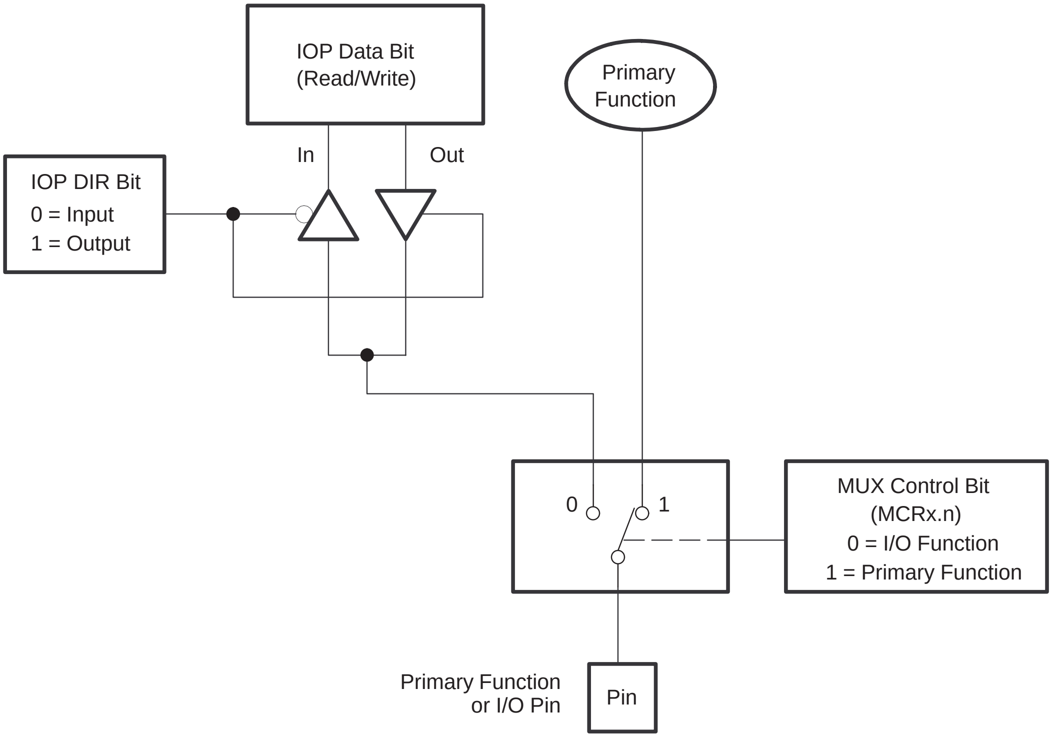


Table 5–1. ’240x Digital I/O Port Control Registers Implementation

Address	Register Mnemonic	Description
7090h	MCRA	I/O MUX Control Register A
7092h	MCRB	I/O MUX Control Register B
7094h	MCRC	I/O MUX Control Register C
7098h	PADATDIR	I/O port A Data and Direction Register
709Ah	PBDATDIR	I/O port B Data and Direction Register
709Ch	PCDATDIR	I/O port C Data and Direction Register
709Eh	PDDATDIR	I/O port D Data and Direction Register
7095h	PEDATDIR	I/O port E Data and Direction Register
7096h	PFDATDIR	I/O port F Data and Direction Register

5.2 Differences in GPIO Implementation in ’240x

There are several differences in the GPIO implementation in ’240x when compared with ’241/’242/’243. When the bit value in an MCRx register (OCRx register in ’24x) is 1, the primary function is always chosen. By the same token, when the bit value is 0, the GPIO function is always chosen. There are no exceptions, as in the case of ’24x, where XF, $\overline{\text{BIO}}$, and CLKOUT pins have a different configuration.

Also, some pins (such as XF and CLKOUT) are paired with different GPIO pins compared to ’24x (i.e., the “primary function/GPIO pin” mapping/pairing is not exactly identical to ’24x. Due to the addition of two GPIO ports, “E” and “F”, a new MUX Control register (MCRC) has been added.

5.3 I/O MUX Control Registers

There are three I/O mux control registers: I/O mux control register A (MCRA), I/O mux control register B (MCRB), and I/O mux control register C (MCRC).

5.3.1 I/O Mux Control Register A

Figure 5–2. I/O Mux Control Register A (MCRA) — Address 7090h

15	14	13	12	11	10	9	8
MCRA.15	MCRA.14	MCRA.13	MCRA.12	MCRA.11	MCRA.10	MCRA.9	MCRA.8
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
MCRA.7	MCRA.6	MCRA.5	MCRA.4	MCRA.3	MCRA.2	MCRA.1	MCRA.0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Table 5–2. I/O Mux Control Register A (MCRA) Configuration

Bit #	Name.bit #	Pin Function Selected	
		(MCA.n = 1) **(Primary)**	**(MCA.n = 0)** **(Secondary)**
0	MCRA.0	SCITXD	IOPA0
1	MCRA.1	SCIRXD	IOPA1
2	MCRA.2	XINT1	IOPA2
3	MCRA.3	CAP1/QEP1	IOPA3
4	MCRA.4	CAP2/QEP2	IOPA4
5	MCRA.5	CAP3	IOPA5
6	MCRA.6	CMP1	IOPA6
7	MCRA.7	CMP2	IOPA7
8	MCRA.8	CMP3	IOPB0
9	MCRA.9	CMP4	IOPB1
10	MCRA.10	CMP5	IOPB2
11	MCRA.11	CMP6	IOPB3
12	MCRA.12	T1CMP	IOPB4
13	MCRA.13	T2CMP	IOPB5
14	MCRA.14	TDIRA	IOPB6
15	MCRA.15	TCLKINA	IOPB7

Note: Due to the absence of XINT1/IOPA2 and TDIRA/IOPB6 pins, bits 2 and 14 of MCRA must be treated as “reserved” for ’2402 devices.

5.3.2 I/O Mux Output Control Register B

Figure 5–3. I/O Mux Control Register B (MCRB) — Address 7092h

15	14	13	12	11	10	9	8
MCRB.15	MCRB.14	MCRB.13	MCRB.12	MCRB.11	MCRB.10	MCRB.9	MCRB.8
RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-0

7	6	5	4	3	2	1	0
MCRB.7	MCRB.6	MCRB.5	MCRB.4	MCRB.3	MCRB.2	MCRB.1	MCRB.0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-1	RW-1

Note: R = Read access, W = Write access, -0 = value after reset

Table 5–3. I/O Mux Control Register B (MCRB)

		Pin Function Selected	
Bit #	**Name.bit #**	**(MCB.n = 1) (Primary)**	**(MCB.n = 0) (Secondary)**
0	MCRB.0	W/$\overline{\text{R}}$	IOPC0
1	MCRB.1	$\overline{\text{BIO}}$	IOPC1
2	MCRB.2	SPISIMO	IOPC2
3	MCRB.3	SPISOMI	IOPC3
4	MCRB.4	SPICLK	IOPC4
5	MCRB.5	$\overline{\text{SPISTE}}$	IOPC5
6	MCRB.6	CANTX	IOPC6
7	MCRB.7	CANRX	IOPC7
8	MCRB.8	XINT2/ADCSOC	IOPD0
9	MCRB.9	EMU0	Reserved
10	MCRB.10	EMU1	Reserved
11	MCRB.11	TCK	Reserved
12	MCRB.12	TDI	Reserved
13	MCRB.13	TDO	Reserved
14	MCRB.14	TMS	Reserved
15	MCRB.15	TMS2	Reserved

Notes:

1) Due to the absence of the W/$\overline{\text{R}}$/IOPC0, $\overline{\text{BIO}}$/IOPC1, and $\overline{\text{SPISTE}}$/IOPC5 pins, bits 0, 1, and 5 of MCRB must be treated as reserved in the ’2402.

2) Due to the absence of SPI and CAN modules (in ’2402), bits 2, 3, 4, 6, and 7 of MCRB should always be written with 0. The corresponding pins work as GPIO pins only.

3) Due to the absence of the CAN module and W/$\overline{\text{R}}$ function in ’2404, bits 0, 6, and 7 of MCRB should always be written with 0. The corresponding pins work as GPIO pins only.

4) Due to the absence of the W/$\overline{\text{R}}$ function in ’2406, bit 0 of MCRB should always be written with 0. The corresponding pin works as a GPIO pin only.

5.3.3 I/O Mux Output Control Register C

Figure 5–4. I/O Mux Control Register C (MCRC) — Address 7094h

15	14	13	12	11	10	9	8
Reserved	Reserved	MCRC.13	MCRC.12	MCRC.11	MCRC.10	MCRC.9	MCRC.8
		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
MCRC.7	MCRC.6	MCRC.5	MCRC.4	MCRC.3	MCRC.2	MCRC.1	MCRC.0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-1

Note: R = Read access, W = Write access, -0 = value after reset

Table 5–4. I/O Mux Control Register C (MCRC)

		Pin Function Selected	
Bit #	**Name.bit #**	**(MCC.n = 1) (Primary)**	**(MCC.n = 0) (Secondary)**
0	MCRC.0	CLKOUT	IOPE0
1	MCRC.1	PWM7	IOPE1
2	MCRC.2	PWM8	IOPE2
3	MCRC.3	PWM9	IOPE3
4	MCRC.4	PWM10	IOPE4
5	MCRC.5	PWM11	IOPE5
6	MCRC.6	PWM12	IOPE6
7	MCRC.7	CAP4/QEP3	IOPE7
8	MCRC.8	CAP5/QEP4	IOPF0
9	MCRC.9	CAP6	IOPF1
10	MCRC.10	T3PWM/T3CMP	IOPF2
11	MCRC.11	T4PWM/T4CMP	IOPF3
12	MCRC.12	TDIRB	IOPF4
13	MCRC.13	TCLKINB	IOPF5
14	MCRC.14	Reserved	Reserved
15	MCRC.15	Reserved	Reserved

Note: Due to the absence of the EVB, bits 1 through 13 must be treated as reserved in the ’2402.

5.4 Data and Direction Control Registers

There are six data and direction control registers. Refer to Table 5–1, *’240x Digital I/O Port Control Registers Implementation*, on page 5-3 for the address locations of each register.

Figure 5–5. Port A Data and Direction Control Register (PADATDIR)

15	14	13	12	11	10	9	8
A7DIR	A6DIR	A5DIR	A4DIR	A3DIR	A2DIR	A1DIR	A0DIR
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
IOPA7	IOPA6	IOPA5	IOPA4	IOPA3	IOPA2	IOPA1	IOPA0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–8 AnDIR

0 Configure corresponding pin as an input.

1 Configure corresponding pin as an output.

Bits 7–0 IOPAn

If AnDIR = 0, then:

0 Corresponding I/O pin is read as a *low*.

1 Corresponding I/O pin is read as a *high*.

If AnDIR = 1, then:

0 Set corresponding I/O pin *low*.

1 Set corresponding I/O pin *high*.

Table 5–5. PADATDIR I/O Pin Designation (Assuming Pins Have Been Selected as I/O; i.e., Secondary Function)

I/O Port Data Bit	Pin Name
IOPA0	SCITXD/IOPA0
IOPA1	SCIRXD/IOPA1
IOPA2	XINT1/IOPA2†
IOPA3	CAP1/QEP1/IOPA3
IOPA4	CAP2/QEP2/IOPA4
IOPA5	CAP3/IOPA5
IOPA6	CMP1/IOPA6
IOPA7	CMP2/IOPA7

† There is no IOPA2 pin in ’2402 devices.

Figure 5–6. Port B Data and Direction Control Register (PBDATDIR)

15	14	13	12	11	10	9	8
B7DIR	B6DIR	B5DIR	B4DIR	B3DIR	B2DIR	B1DIR	B0DIR
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
IOPB7	IOPB6	IOPB5	IOPB4	IOPB3	IOPB2	IOPB1	IOPB0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–8 **BnDIR**

0 Configure corresponding pin as an input.

1 Configure corresponding pin as an output.

Bits 7–0 **IOPBn**

If BnDIR = 0, then:

0 Corresponding I/O pin is read as a *low*.

1 Corresponding I/O pin is read as a *high*.

If BnDIR = 1, then:

0 Set corresponding I/O pin *low*.

1 Set corresponding I/O pin *high*.

Table 5–6. PBDATDIR I/O Pin Designation (Assuming Pins Have Been Selected as I/O; i.e., Secondary Function)

I/O Port Data Bit	Pin Name
IOPB0	CMP3/IOPB0
IOPB1	CMP4/IOPB1
IOPB2	CMP5/IOPB2
IOPB3	CMP6/IOPB3
IOPB4	T1CMP/IOPB4
IOPB5	T2CMP/IOPB5
IOPB6	TDIR/IOPB6†
IOPB7	TCLKIN/IOPB7

† There is no IOPB6 pin in ’2402 devices.

Figure 5–7. Port C Data and Direction Control Register (PCDATDIR)

15	14	13	12	11	10	9	8
C7DIR	C6DIR	C5DIR	C4DIR	C3DIR	C2DIR	C1DIR	C0DIR
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
IOPC7	IOPC6	IOPC5	IOPC4	IOPC3	IOPC2	IOPC1	IOPC0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–8 **CnDIR**

0 Configure corresponding pin as an input.

1 Configure corresponding pin as an output.

Bits 7–0 **IOPCn**

If CnDIR = 0, then:

0 Corresponding I/O pin is read as a *low*.

1 Corresponding I/O pin is read as a *high*.

If CnDIR = 1, then:

0 Set corresponding I/O pin *low*.

1 Set corresponding I/O pin *high*.

Table 5–7. PCDATDIR I/O Pin Designation (Assuming Pins Have Been Selected as I/O; i.e., Secondary Function)

I/O Port Data Bit	Pin Name
IOPC0	W/$\overline{\text{R}}$/IOPC0†
IOPC1	$\overline{\text{BIO}}$/IOPC1†
IOPC2	SPISIMO/IOPC2
IOPC3	SPISOMI/IOPC3
IOPC4	SPICLK/IOPC4
IOPC5	$\overline{\text{SPISTE}}$/IOPC5†
IOPC6	CANTX/IOPC6
IOPC7	CANRX/IOPC7

† These pins are not available in ’2402 devices.

Figure 5–8. Port D Data and Direction Control Register (PDDATDIR)

15–9	8
Reserved	D0DIR
	RW-0

7–1	0
Reserved	IOPD0
	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–9 **Reserved**

Bit 8 **D0DIR**

0 Configure corresponding pin as an input.

1 Configure corresponding pin as an output.

Bits 7–1 **Reserved**

Bit 0 **IOPD0**

If D0DIR = 0, then:

0 Corresponding I/O pin is read as a *low*.

1 Corresponding I/O pin is read as a *high*.

If D0DIR = 1, then:

0 Set corresponding I/O pin *low*.

1 Set corresponding I/O pin *high*.

Table 5–8. PDDATDIR I/O Pin Designation (Assuming Pins Have Been Selected as I/O; i.e., Secondary Function)

I/O Port Data Bit	Pin Name
IOPD0	XINT2/ADCSOC/IOPD0
IOPD1	Reserved
IOPD2	Reserved
IOPD3	Reserved
IOPD4	Reserved
IOPD5	Reserved
IOPD6	Reserved
IOPD7	Reserved

Figure 5–9. Port E Data and Direction Control Register (PEDATDIR)

15	14	13	12	11	10	9	8
E7DIR	E6DIR	E5DIR	E4DIR	E3DIR	E2DIR	E1DIR	E0DIR
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
IOPE7	IOPE6	IOPE5	IOPE4	IOPE3	IOPE2	IOPE1	IOPE0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–8 **EnDIR**

0 Configure corresponding pin as an input.

1 Configure corresponding pin as an output.

Bits 7–0 **IOPEn**

If EnDIR = 0, then:

0 Corresponding I/O pin is read as a *low*.

1 Corresponding I/O pin is read as a *high*.

If EnDIR = 1, then:

0 Set corresponding I/O pin *low*.

1 Set corresponding I/O pin *high*.

Table 5–9. PEDATDIR I/O Pin Designation (Assuming Pins Have Been Selected as I/O; i.e., Secondary Function)

I/O Port Data Bit	Pin Name
IOPE0	CLKOUT/IOPE0
IOPE1	PWM7/IOPE1†
IOPE2	PWM8/IOPE2†
IOPE3	PWM9/IOPE3†
IOPE4	PWM10/IOPE4†
IOPE5	PWM11/IOPE5†
IOPE6	PWM12/IOPE6†
IOPE7	CAP4/QEP3/IOPE7†

† These pins are not available in ’2402 devices.

Figure 5–10. Port F Data and Direction Control Register (PFDATDIR)

15	14	13	12	11	10	9	8
Reserved	F6DIR	F5DIR	F4DIR	F3DIR	F2DIR	F1DIR	F0DIR
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
Reserved	IOPF6	IOPF5	IOPF4	IOPF3	IOPF2	IOPF1	IOPF0
	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 15 **Reserved**

Bits 14–8 **FnDIR**

0 Configure corresponding pin as an input.

1 Configure corresponding pin as an output.

Bit 7 **Reserved**

Bits 6–0 **IOPFn**

If FnDIR = 0, then:

0 Corresponding I/O pin is read as a *low*.

1 Corresponding I/O pin is read as a *high*.

If FnDIR = 1, then:

0 Set corresponding I/O pin *low*.

1 Set corresponding I/O pin *high*.

Table 5–10. PFDATDIR I/O Pin Designation (Assuming Pins Have Been Selected as I/O; i.e., Secondary Function)

I/O Port Data Bit	Pin Name
IOPF0	CAP5/QEP4/IOPF0†
IOPF1	CAP6/IOPF1†
IOPF2	T3PWM/T3CMP/IOPF2†
IOPF3	T4PWM/T4CMP/IOPF3†
IOPF4	TDIR2/IOPF4†
IOPF5	TCLKIN2/IOPF5†
IOPF6	IOPF6†
Reserved	Reserved

† These pins are not available in ’2402 devices.

Chapter 6

Event Manager (EV)

This chapter describes the ’240x Event Manager (EV) module. Most of the EV pins are shared with general-purpose digital I/O signals. This pin sharing and how it is controlled is described in Chapter 5, *Digital Input/Output (I/O)*.

The EV module provides a broad range of functions and features that are particularly useful in motion control and motor control applications. There are differences in terms of the functionality between the EV module of ’240x devices and the EV module of ’240 devices. (However, the EV modules in the ’24x and ’240x families of DSPs are exactly identical in terms of functionality.) Note that all devices of the ’240x family (with the exception of ’2402) have two EV modules as opposed to one EV module in ’241/’242/’243.

Topic **Page**

6.1 Event Manager (EV) Functional Blocks

All devices of the ’240x family, with the exception of the ’2402, have two event managers, EVA and EVB. These two event managers are exactly identical to each other in terms of functionality and register mapping/bit definition. For the sake of brevity, only the functionality of EVA is explained. Minor differences (such as naming conventions and register addresses) are highlighted as appropriate.

Each EV module in the ’240x device contains the following functional blocks:

- Two general-purpose (GP) timers (described in section 6.3 on page 6-14).
- Three compare units (described in section 6.4 on page 6-37).
- Pulse-width modulation (PWM) circuits that include space vector PWM circuits, dead-band generation units, and output logic (described in section 6.5 on page 6-46, section 6.6 on page 6-55, and section 6.7 on page 6-60).
- Three capture units (described in section 6.8 on page 6-66).
- Quadrature encoder pulse (QEP) circuit (described in section 6.9, page 6-78).
- Interrupt logic (described in section 6.10 on page 6-82).

Figure 6–1 shows a block diagram of the EVA module and Figure 6–2 shows a block diagram of the EVB module.

Figure 6–1. Event Manager (EVA) Block Diagram

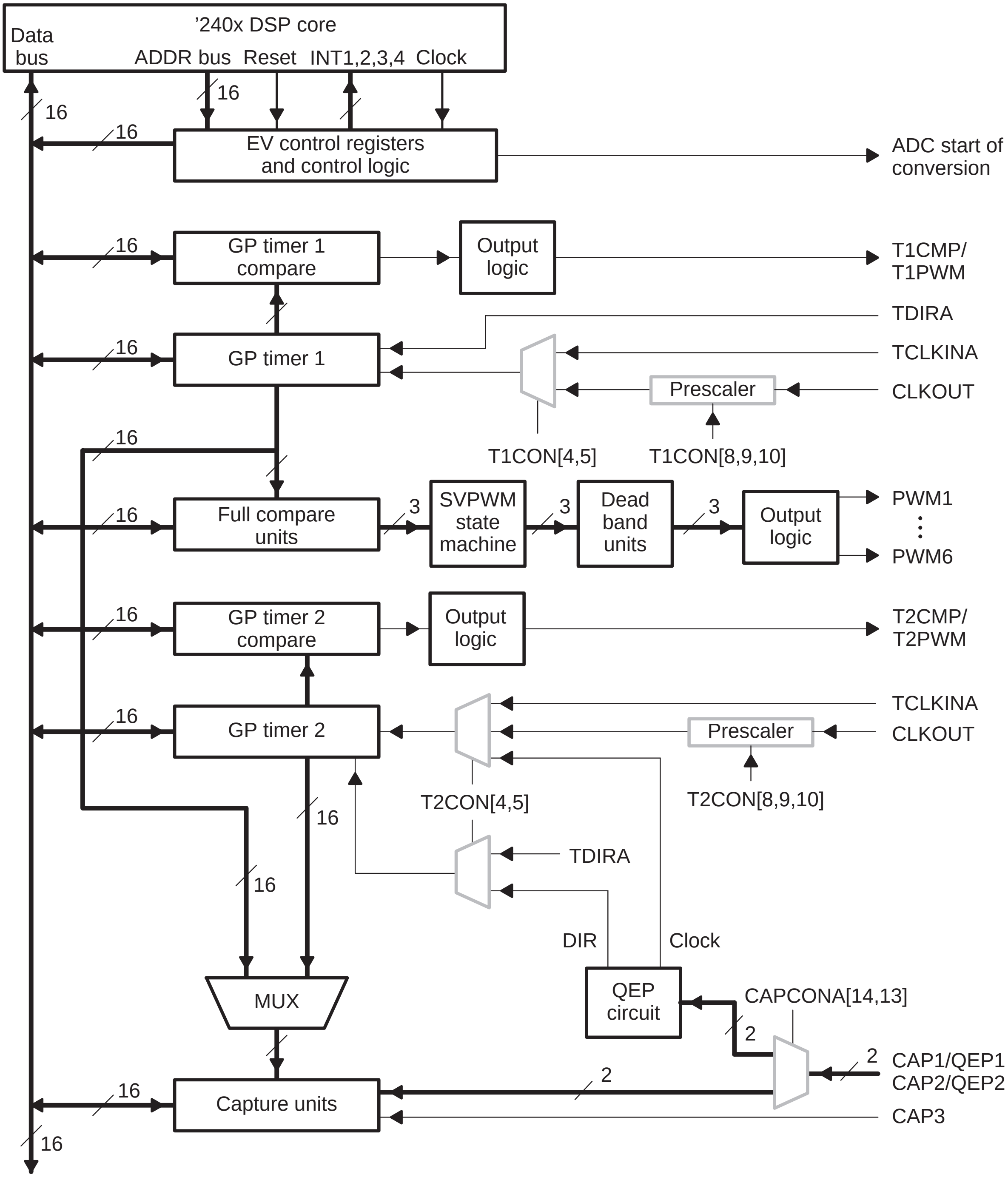
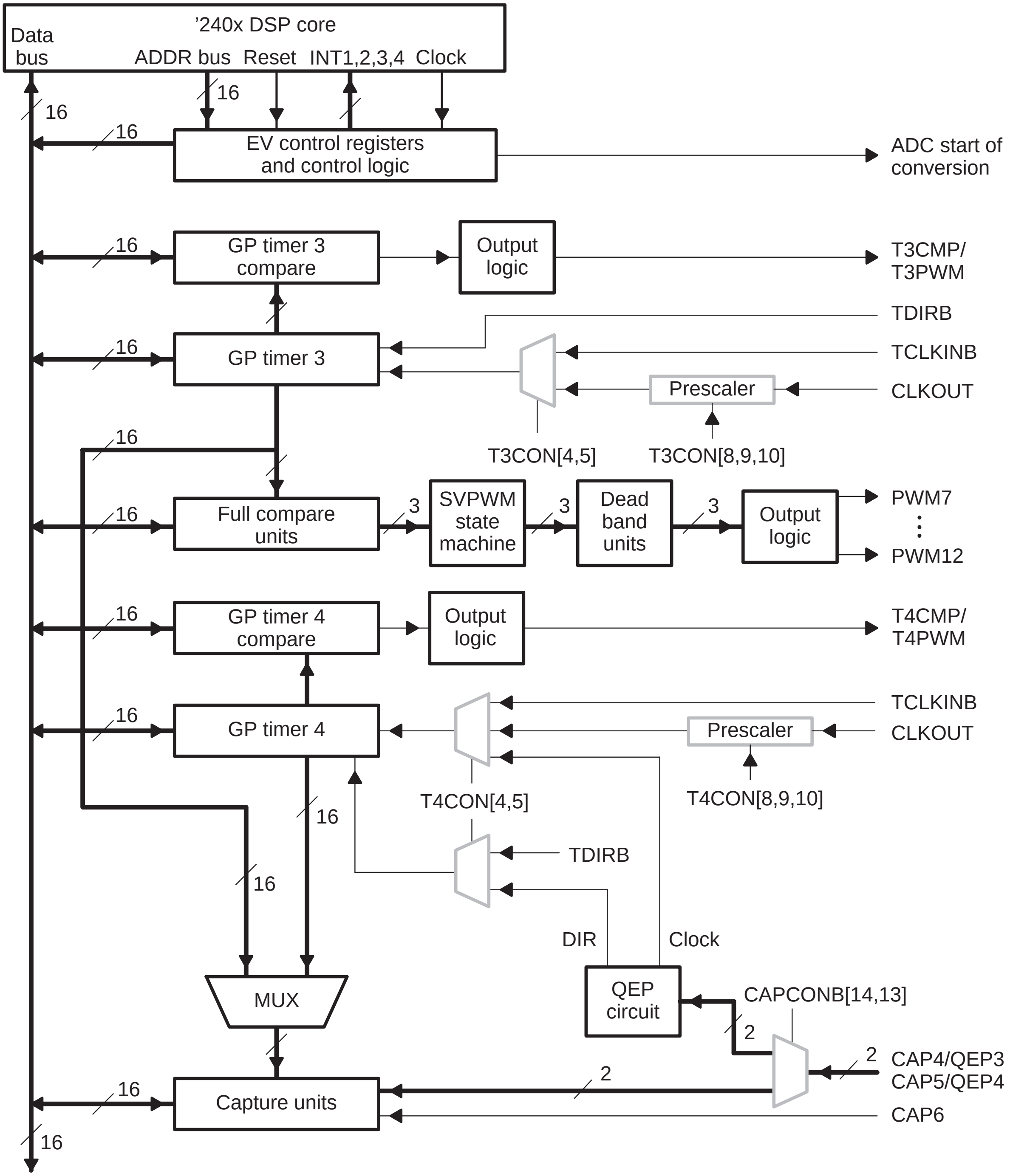


Figure 6–2. Event Manager (EVB) Block Diagram



6.1.1 Differences Between ’C240 EV and ’240x EV

- ❑ The single-up count and single-up/down count modes have been removed from the remaining GP timers. *Software change:* The four timer modes are now decoded with TMODE1–0. This decoding is different from the ’C240 EV. TMODE2 is now a reserved bit.
- ❑ There is no 32-bit timer mode.
- ❑ The GP Timers do not stay at the period register value, FFFFh or 0000h when operating in directional-up/down count mode (including QEP mode). They now reverse direction when one of these end points is reached.
- ❑ A capture 3 event is now able to start the ADC.
- ❑ The capture units of a particular EV can now use any timer associated with that EV as a time base.
- ❑ The capture interrupt flag gets set when a capture event occurs only if there are one or more capture events stored in the FIFO already.
- ❑ The Capture FIFO status bits are now RW. Bits 5–0 of CAPFIFO are now unnecessary and are reserved.
- ❑ Both locations in the capture FIFO can be read individually, not just the top location.
- ❑ The QEP logic can only clock GP timer 2 for EVA and GP timer 4 for EVB.
- ❑ The three simple compare units have been removed.
- ❑ The compare mode of the (full) compare units has been removed. They now only operate in PWM mode.
- ❑ The dead band counters have been reduced from 8 bits to 4 bits. The dead band prescaler has been increased from 3 bits to 5 bits, adding two more prescale values: x/16 and x/32. *Software change:* There are now three DBTPSx bits. DBTPS0 moves to bit 2 of DBTCON, DBTPS1 moves to bit 3 and bit 4 becomes DBTPS2.
- ❑ Any register bits associated with the removed functions are now reserved (not implemented).
- ❑ Most interrupt control logic has been removed from each peripheral. Each peripheral now simply has one interrupt request signal and associated enable for each interrupt flag. The peripheral interrupt vector table (containing the peripheral interrupt vectors) is now located in the peripheral interrupt expansion (PIE) controller.

- Software writing a 1 to the interrupt flag, which has been identified by the interrupt vector ID, is required to clear the flag. Reading the interrupt vector ID no longer automatically clears the associated flag.
- $\overline{\text{PDPINTA/B}}$ is now enabled following reset.
- Only one write is required to initialize COMCONA/B, not two as on the ’C240.

6.1.2 EV Pins

Each EV module has eight device pins available for compare/PWM outputs:

- Two GP timer compare/PWM output pins:

EVA	EVB
T1CMP/T1PWM	T3CMP/T3PWM
T2CMP/T2PWM	T4CMP/T4PWM

- Six (full) compare/PWM output pins:

EVA	EVB
PWM1	PWM7
PWM2	PWM8
PWM3	PWM9
PWM4	PWM10
PWM5	PWM11
PWM6	PWM12

The EVA module uses three device pins, CAP1/QEP1, CAP2/QEP2, and CAP3, as capture or quadrature encoder pulse inputs.

The EVB module uses three device pins, CAP4/QEP3, CAP5/QEP4, and CAP6, as capture or quadrature encoder pulse inputs.

The timers in the EV module can be programmed to operate based on an external clock or the internal device clock. The device pin TCLKINA/B supplies the external clock input.

The device pin TDIRA/B is used to specify the counting direction when a GP timer is in directional up-/down-counting mode.

The device pins are summarized in Table 6–1 and Table 6–2.

Table 6–1. Event Manager A Pins

Pin Name	Description
CAP1/QEP1	Capture Unit 1 input, QEP circuit input 1
CAP2/QEP2	Capture Unit 2 input, QEP circuit input 2
CAP3	Capture Unit 3 input
PWM1	Compare Unit 1 output 1
PWM2	Compare Unit 1 output 2
PWM3	Compare Unit 2 output 1
PWM4	Compare Unit 2 output 2
PWM5	Compare Unit 3 output 1
PWM6	Compare Unit 3 output 2
T1CMP	Timer 1 compare/PWM output
T2CMP	Timer 2 compare/PWM output
TCLKINA	External clock input for Timers in EVA
TDIRA	External timer direction input in EVA

Table 6–2. Event Manager B Pins

Pin Name	Description
CAP4/QEP3	Capture Unit 4 input, QEP circuit input 3
CAP5/QEP4	Capture Unit 5 input, QEP circuit input 4
CAP6	Capture Unit 6 input
PWM7	Compare Unit 4 output 1
PWM8	Compare Unit 4 output 2
PWM9	Compare Unit 5 output 1
PWM10	Compare Unit 5 output 2
PWM11	Compare Unit 6 output 1
PWM12	Compare Unit 6 output 2
T3CMP	Timer 3 compare/PWM output
T4CMP	Timer 4 compare/PWM output
TCLKINB	External clock input for Timers in EVB
TDIRB	External timer direction input in EVB

6.1.3 Power Drive Protection Interrupt ($\overline{\text{PDPINTx}}$, x = A or B)

An interrupt is generated when the device pin power drive protection interrupt ($\overline{\text{PDPINTx}}$) is pulled low. This interrupt is provided for the safe operation of systems such as power converters and motor drives. If $\overline{\text{PDPINTx}}$ is unmasked, all EV output pins will be put in the high-impedance state by hardware immediately after the $\overline{\text{PDPINTx}}$ pin is pulled low. The interrupt flag associated with $\overline{\text{PDPINTx}}$ is also set when such an event occurs; however, it must wait until the transition on $\overline{\text{PDPINTx}}$ has been qualified and synchronized with the internal clock. The qualification and synchronization causes a delay of *2* clock cycles. If $\overline{\text{PDPINTx}}$ is unmasked, the flag keeps the EV outputs in the high-impedance state and generates a peripheral interrupt request. The setting of the flag does not depend on whether $\overline{\text{PDPINTx}}$ is masked: it happens when a qualified transition occurs on the $\overline{\text{PDPINTx}}$ pin. $\overline{\text{PDPINTx}}$ can be used to inform the monitoring program of motor drive abnormalities such as over-voltage, over-current, and excessive temperature rise.

This interrupt is enabled following reset.

6.1.4 EV Registers

The Event Manager registers occupy two 64-word (16-bit) frames of address space. The Event Manager module decodes the lower 6-bits of the address; while the upper 10 bits of the address are decoded by the peripheral address decode logic, which provides a module select to the Event Manager when the peripheral address bus carries an address within the range designated for the EV on that device.

On the ’240x devices (as with the ’C240 device), EVA registers are located in the range 7400h to 7431h. EVB registers are located in the range of 7500h to 7531h.

The undefined registers and undefined bits of the EV registers all return zero when read by user software. Writes have no effect. See Section 6.2, on page 6-11.

6.1.5 EV Interrupts

The event manager interrupts are arranged into three groups. Each group is assigned one CPU interrupt (INT2, 3 or 4). Since each group has multiple interrupt sources, the CPU interrupt requests are processed by the Peripheral Interrupt Expansion module. The ’240x interrupt requests have the following stages of response:

❑ *Interrupt source*. If peripheral interrupt conditions occur, the respective flag bits in registers EVxIFRA, EVxIFRB, or EVxIFRC (x = A or B) are set. Once set, these flags remain set until explicity cleared by the software. It is mandatory to clear these flags in the software or future interrupts will not be recognized.

❑ *Interrupt enable*. The event manager interrupts can be individually enable-d or disabled by interrupt mask registers EVxIMRA, EVxIMRB, and EVxIMRC (x = A or B). Each bit is set to 1 to enable/unmask the interrupt or cleared to 0 to disable/mask the interrupt.

❑ *PIE request*. If both interrupt flag bits and interrupt mask bits are set, then the peripheral issues a peripheral interrupt request to the PIE module. The PIE module can receive more than one interrupt from the peripheral. The PIE logic records all the interrupt requests and generates the respective CPU interrupt (INT1, 2, 3, or 4) based on the preassigned priority of the received interrupts. See Table 2–2, *’240x Interrupt Source Priority and Vectors*, on page 2-8 for priority and vector values.

- ❑ *CPU response*. On receipt of INT1, 2, 3, or 4 interrupt request, the respective bit in the CPU interrupt flag register (IFR) will be set. If the corresponding interrupt mask register (IMR) bit is set and INTM bit is cleared, then the CPU recognizes the interrupt and issues an acknowledgement to the PIE. Following this, the CPU finishes executing the current instruction and branches to the interrupt vector corresponding to INT1, 2, 3, or 4. At this time, the respective IFR bit will be cleared and the INTM bit will be set disabling further interrupt recognition. The interrupt vector contains a branch instruction for the interrupt service routine. From here, the interrupt response is controlled by the software.

- ❑ *PIE response*. The PIE logic uses the acknowledge signal from the core to clear the PIRQ bit that issued the CPU interrupt. Along with this, the PIE updates its PIVR register with the interrupt vector, unique to the peripheral interrupt, that was just acknowledged. After this, the PIE hardware works in parallel to the current interrupt software to generate aCPU interrupt and other pending interrupts, if any.

- ❑ *Interrupt software*. The interrupt software has two levels of response.

  - ■ Level 1 (GISR). In the first level the software should do any context save and read the PIVR register from PIE module to decide which interrupt group caused the interrupt.Since the PIVR value is unique, it can be used to branch to the interrupt service routine specific to this interrupt condition.

  - ■ Level 2 (SISR). This level is optional and could reside as a part of level 1. However, at this stage the interrupt software has explicit responsibility to avoid improper interrupt response. After executing the interrupt specific code, the routine should clear the interrupt flag in the EVxIFRA EVxIFRB, or EVxIFRC that caused the serviced interrupt. Code will return after enabling the CPU’s global interrupt bit INTM (clear INTM bit).

6.2 Event Manager (EV) Register Addresses

Table 6–3 through Table 6–10 display the addresses of the Event Manager registers.

Table 6–3. Addresses of EVA Timer Registers

Address	Register	Name	
7400h	GPTCONA	Timer control register	
7401h	T1CNT	Timer 1 counter register	Timer 1
7402h	T1CMPR	Timer 1 compare register	
7403h	T1PR	Timer 1 period register	
7404h	T1CON	Timer 1 control register	
7405h	T2CNT	Timer 2 counter register	Timer 2
7406h	T2CMPR	Timer 2 compare register	
7407h	T2PR	Timer 2 period register	
7408h	T2CON	Timer 2 control register	

Table 6–4. Addresses of EVB Timer Registers

Address	Register	Name	
7500h	GPTCONB	Timer control register	
7501h	T3CNT	Timer 3 counter register	Timer 3
7502h	T3CMPR	Timer 3 compare register	
7503h	T3PR	Timer 3 period register	
7504h	T3CON	Timer 3 control register	
7505h	T4CNT	Timer 4 counter register	Timer 4
7506h	T4CMPR	Timer 4 compare register	
7507h	T4PR	Timer 4 period register	
7508h	T4CON	Timer 4 control register	

Table 6–5. Addresses of EVA Compare Control Registers

Address	Register	Name
7411h	COMCONA	Compare control register
7413h	ACTRA	Compare action control register
7415h	DBTCONA	Dead-band timer control register
7417h	CMPR1	Compare register 1
7418h	CMPR2	Compare register 2
7419h	CMPR3	Compare register 3

Table 6–6. Addresses of EVB Compare Control Registers

Address	Register	Name
7511h	COMCONB	Compare control register
7513h	ACTRB	Compare action control register
7515h	DBTCONB	Dead-band timer control register
7517h	CMPR4	Compare register 4
7518h	CMPR5	Compare register 5
7519h	CMPR6	Compare register 6

Table 6–7. Addresses of EVA Capture Registers

Address	Register	Name
7420h	CAPCONA	Capture control register
7422h	CAPFIFOA	Capture FIFO status register
7423h	CAP1FIFO	Two-level-deep capture FIFO stack 1
7424h	CAP2FIFO	Two-level-deep capture FIFO stack 2
7425h	CAP3FIFO	Two-level-deep capture FIFO stack 3
7427h	CAP1FBOT	Bottom registers of FIFO stacks, allows most recent CAPTURE value to be read.
7428h	CAP2FBOT	
7429h	CAP3FBOT	

Table 6–8. Addresses of EVB Capture Registers

Address	Register	Name
7520h	CAPCONB	Capture control register
7522h	CAPFIFOB	Capture FIFO status register
7523h	CAP4FIFO	Two-level-deep capture FIFO stack 4
7524h	CAP5FIFO	Two-level-deep capture FIFO stack 5
7525h	CAP6FIFO	Two-level-deep capture FIFO stack 6
7527h	CAP4FBOT	Bottom registers of FIFO stacks, allows most recent CAPTURE value to be read.
7528h	CAP5FBOT	
7529h	CAP6FBOT	

Table 6–9. Addresses of EVA Interrupt Registers

Address	Register	Name
742Ch	EVAIMRA	Interrupt mask register A
742Dh	EVAIMRB	Interrupt mask register B
742Eh	EVAIMRC	Interrupt mask register C
742Fh	EVAIFRA	Interrupt flag register A
7430h	EVAIFRB	Interrupt flag register B
7431h	EVAIFRC	Interrupt flag register C

Table 6–10. Addresses of EVB Interrupt Registers

Address	Register	Name
752Ch	EVBIMRA	Interrupt mask register A
752Dh	EVBIMRB	Interrupt mask register B
752Eh	EVBIMRC	Interrupt mask register C
752Fh	EVBIFRA	Interrupt flag register A
7530h	EVBIFRB	Interrupt flag register B
7531h	EVBIFRC	Interrupt flag register C

6.3 General-Purpose (GP) Timers

There are two general-purpose (GP) timers in each module. These timers can be used as independent time bases in applications such as:

- ❑ The generation of a sampling period in a control system
- ❑ Providing a time base for the operation of the quadrature encoder pulse (QEP) circuit (GP timer 2/4 only) and the capture units
- ❑ Providing a time base for the operation of the compare units and associated PWM circuits to generate PWM outputs

Timer Functional Blocks

Figure 6–3 shows a block diagram of a GP timer. Each GP timer includes:

- ❑ One readable and writeable (RW) 16-bit up and up/down counter register TxCNT (x = 1, 2, 3, 4). This register stores the current value of the counter and keeps incrementing or decrementing depending on the direction of counting.
- ❑ One RW 16-bit timer compare register (shadowed), TxCMPR (x = 1, 2, 3, 4)
- ❑ One RW 16-bit timer period register (shadowed), TxPR (x = 1, 2, 3, 4)
- ❑ RW 16-bit individual timer control register, TxCON (x = 1, 2, 3, 4)
- ❑ Programmable prescaler applicable to both internal and external clock inputs
- ❑ Control and interrupt logic
- ❑ One GP timer compare output pin, TxCMP (x = 1, 2, 3, 4)
- ❑ Output conditioning logic

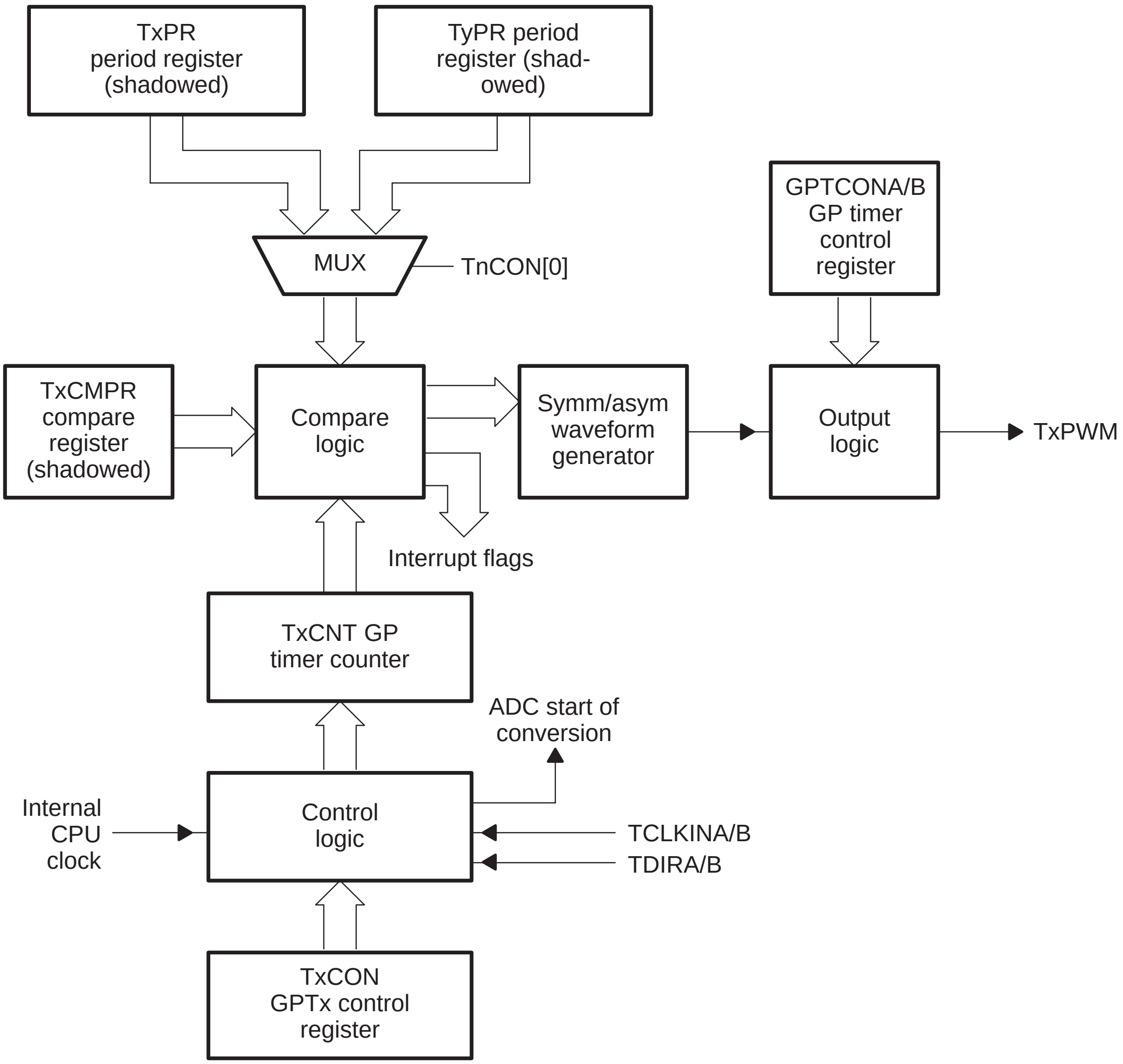
Another overall control register, GPTCONA/B, specifies the action to be taken by the timers on different timer events, and indicates the counting directions of the GP timers. GPTCONA/B is readable and writeable, although writing to the status bits has no effect.

Note:

Timer 2 can select the period register of timer 1 as its period register. In Figure 6–3, the mux is applicable only when the figure represents timer 2.

Timer 4 can select the period register of timer 3 as its period register. In Figure 6–3, the mux is applicable only when the figure represents timer 4.

Figure 6–3. General-Purpose Timer Block Diagram (x = 2 or 4)
[when x = 2: y = 1 and n = 2; when x = 4: y = 3 and n = 4]



GP Timer Inputs

The inputs to the GP timers are:

❑ The internal device (CPU) clock.

❑ An external clock, TCLKINA/B, that has a maximum frequency of one-fourth that of the device clock.

- ❑ Direction input, TDIRA/B, for use by the GP timers in directional up-/down-counting mode.
- ❑ Reset signal, RESET.

When a timer is used with the QEP circuit, the QEP circuit generates both the timer's clock and the counting direction.

GP Timer Outputs

The outputs of the timers are:

- ❑ GP timer compare outputs TxCMP, x = 1, 2, 3, 4
- ❑ ADC start-of-conversion signal to ADC module
- ❑ Underflow, overflow, compare match, and period match signals to its own compare logic and to the compare units
- ❑ Counting direction indication bits

Individual GP Timer Control Register (TxCON)

The operational mode of a timer is controlled by its individual control register TxCON. Bits in the TxCON register determine:

- ❑ Which of the four counting modes the timer is in
- ❑ Whether an internal or external clock is to be used by the GP Timer
- ❑ Which of the eight input clock prescale factors (ranging from 1 to 1/128) is used
- ❑ On which condition the timer compare register is reloaded
- ❑ Whether the timer is enabled or disabled
- ❑ Whether the timer compare operation is enabled or disabled
- ❑ Which period register is used by timer 2, its own, or timer 1's period register (EVA)
  Which period register is used by timer 4, its own, or timer 3's period register (EVB)

Overall GP Timer Control Register (GPTCONA/B)

The control register GPTCONA/B specifies the action to be taken by the timers on different timer events and indicates their counting directions.

GP Timer Compare Registers

The compare register associated with a GP timer stores the value to be constantly compared with the counter of the GP timer. When a match happens, the following events occur:

- ❑ A transition occurs on the associated compare output according to the bit pattern in GPTCONA/B.
- ❑ The corresponding interrupt flag is set.
- ❑ A peripheral interrupt request is generated if the interrupt is unmasked.

The compare operation of a GP timer can be enabled or disabled by the appropriate bit in TxCON.

The compare operation and outputs can be enabled in any of the timer modes, including QEP mode.

GP Timer Period Register

The value in the period register of a GP timer determines the period of the timer. A GP timer resets to 0, or starts counting downward when a match occurs between the period register and the timer counter, depending on which counting mode the timer is in.

Double Buffering of GP Timer Compare and Period Registers

The compare and period registers, TxCMPR and TxPR, of a GP timer are shadowed. A new value can be written to any of these registers at any time during a period. However, the new value is written to the associated shadow register. For the compare register, the content in the shadow register is loaded into the working (active) register only when a certain timer event specified by TxCON occurs. For the period register, the working register is reloaded with the value in its shadow register only when the value of the counter register TxCNT is 0. The condition on which a compare register is reloaded can be one of the following:

- ❑ Immediately after the shadow register is written
- ❑ On underflow; that is, when the GP timer counter value is 0
- ❑ On underflow or period match; that is, when the counter value is 0 or when the counter value equals the value of the period register

The double buffering feature of the period and compare registers allows the application code to update the period and compare registers at any time during a period in order to change the timer period and the width of the PWM pulse for the period that follows. On-the-fly change of the timer period value, in the case of PWM generation, means on-the-fly change of PWM carrier frequency.

Caution :

The period register of a GP timer should be initialized before its counter is initialized to a non-zero value. Otherwise, the value of the period register will remain unchanged until the next underflow.

Note that a compare register is transparent (the newly loaded value goes directly into the active register) when the associated compare operation is disabled. This applies to all Event Manager compare registers.

GP Timer Compare Output

The compare output of a GP timer can be specified active high, active low, forced high, or forced low, depending on how the GPTCONA/B bits are configured. It goes from low to high (high to low) on the first compare match when it is active high (low). It then goes from high to low (low to high) on the second compare match if the GP timer is in an up-/down-counting mode, or on period match if the GP timer is in up-counting mode. The timer compare output becomes high (low) right away when it is specified to be forced high (low).

Timer Counting Direction

The counting directions of the GP timers are reflected by their respective bits in GPTCONA/B during all timer operations as follows:

- 1 represents the up-counting direction.
- 0 represents the down-counting direction.

The input pin TDIRA/B determines the direction of counting when a GP timer is in directional up-/down-counting mode. When TDIRA/B is high, upward counting is specified; when TDIRA/B is low, downward counting is specified.

Timer Clock

The source of the GP timer clock can be the internal device clock or the external clock input, TCLKINA/B. The frequency of the external clock must be less than or equal to one-fourth of that of the device clock. GP timer 2 (EVA) and

GP timer 4 (EVB) can be used with the QEP circuits, in directional up-/down-counting mode. In this case, the QEP circuits provide both the clock and direction inputs to the timer.

A wide range of prescale factors are provided for the clock input to each GP timer.

QEP-Based Clock Input

The quadrature encoder pulse (QEP) circuit, when selected, can generate the input clock and counting direction for GP timer 2/4 in the directional up-/down-counting mode. This input clock cannot be scaled by GP timer prescaler circuits (that is, the prescaler of the selected GP timer is always 1 if the QEP circuit is selected as the clock source). Furthermore, the frequency of the clock generated by the QEP circuits is four times that of the frequency of each QEP input channel because both the rising and falling edges of both QEP input channels are counted by the selected timer. The frequency of the QEP input must be less than or equal to one-fourth of that of the device clock.

GP Timer Synchronization

GP timer 2 can be synchronized with GP timer 1 (for EVA) and GP timer 4 can be synchronized with GP timer 3 (for EVB) by proper configuration of T2CON and T4CON, respectively, in the following ways:

- EVA:
  Set the T2SWT1 bit in T2CON to start GP timer 2 counting with the TENABLE bit in T1CON (thus, both timer counters start simultaneously).
- EVA:
  Initialize the timer counters in GP timers 1 and 2 with different values before starting synchronized operation.
- EVA:
  Specify that GP timer 2 uses the period register of GP timer 1 as its period register (ignoring its own period register) by setting SELT1PR in T2CON.
- EVB:
  Set the T4SWT3 bit in T4CON to start GP timer 4 counting with the TENABLE bit in T3CON (thus, both timer counters start simultaneously).
- EVB:
  Initialize the timer counters in GP timers 3 and 4 with different values before starting synchronized operation.
- EVB:
  Specify that GP timer 4 uses the period register of GP timer 3 as its period register (ignoring its own period register) by setting SELT3PR in T4CON.

This allows the desired synchronization between GP timer events. Since each GP timer starts the counting operation from its current value in the counter register, one GP timer can be programmed to start with a known delay after the other GP timer.

Starting the A/D Converter with a Timer Event

The bits in GPTCONA/B can specify that an ADC start signal be generated on a GP timer event such as underflow, compare match, or period match. This feature provides synchronization between the GP timer event and the ADC start without any CPU intervention.

GP Timer in Emulation Suspend

The GP timer control register bits also define the operation of the GP timers during emulation suspend. These bits can be set to allow the operation of GP timers to continue when an emulation interrupt occurs making in-circuit emulation possible. They can also be set to specify that the operation of GP timers stops immediately, or after completion of the current counting period, when emulation interrupt occurs.

Emulation suspend occurs when the device clock is stopped by the emulator, for example, when the emulator encounters a break point.

GP Timer Interrupts

There are sixteen interrupt flags in EVAIFRA, EVAIFRB, EVBIFRA, and EVBIFRB registers for the GP timers. Each of the four GP timers can generate four interrupts upon the following events:

❑ Overflow: TxOFINT (x = 1, 2, 3, or 4)

❑ Underflow: TxUFINT (x = 1, 2, 3, or 4)

❑ Compare match: TxCINT (x = 1, 2, 3, or 4)

❑ Period match: TxPINT (x = 1, 2, 3, or 4)

A timer compare event (match) happens when the content of a GP timer counter is the same as that of the compare register. The corresponding compare interrupt flag is set one clock cycle after the match if the compare operation is enabled.

An overflow event occurs when the value of the timer counter reaches FFFFh. An underflow event occurs when the timer counter reaches 0000h. Similarly, a period event happens when the value of the timer counter is the same as that of the period register. The overflow, underflow, and period interrupt flags of the timer are set one clock cycle after the occurrence of each individual event. Note that the definition of overflow and underflow is different from their conventional definitions.

6.3.1 GP Timer Counting Operation

Each GP timer has four possible modes of operation:

- ❑ Stop/Hold mode
- ❑ Continuous Up-Counting mode
- ❑ Directional Up-/Down-Counting mode
- ❑ Continuous Up-/Down-Counting mode

The bit pattern in the corresponding timer control register TxCON determines the counting mode of a GP timer. The timer enabling bit, TxCON[6], enables or disables the counting operation of a timer. When the timer is disabled, the counting operation of the timer stops and the prescaler of the timer is reset to x/1. When the timer is enabled, the timer starts counting according to the counting mode specified by other bits of TxCON.

Stop/Hold Mode

In this mode the GP timer stops and holds at its current state. The timer counter, the compare output, and the prescale counter all remain unchanged in this mode.

Continuous Up-Counting Mode

The GP timer in this mode counts up according to the scaled input clock until the value of the timer counter matches that of the period register. On the next rising edge of the input clock after the match, the GP timer resets to 0 and starts counting up again.

The period interrupt flag of the timer is set one clock cycle after the match between the timer counter and period register. A peripheral interrupt request is generated if the flag is not masked. An ADC start is sent to the ADC module at the same time the flag is set, if the period interrupt of this timer has been selected by the appropriate bits in GPTCONA/B to start the ADC.

One clock cycle after the GP timer becomes 0, the underflow interrupt flag of the timer is set. A peripheral interrupt request is generated by the flag if it is unmasked. An ADC start is sent to the ADC module at the same time if the underflow interrupt flag of this timer has been selected by appropriate bits in GPTCONA/B to start ADC.

The overflow interrupt flag is set one clock cycle after the value in TxCNT matches FFFFh. A peripheral interrupt request is generated by the flag if it is unmasked.

The duration of the timer period is (TxPR) + 1 cycles of the scaled clock input except for the first period. The duration of the first period is the same if the timer counter is 0 when counting starts.

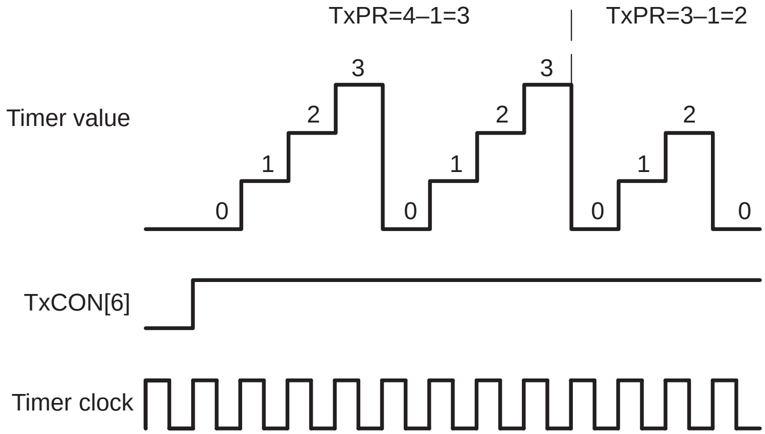
The initial value of the GP timer can be any value between 0h and FFFFh inclusive. When the initial value is greater than the value in the period register, the timer counts up to FFFFh, resets to 0, and continues the operation as if the initial value was 0. When the initial value in the timer counter is the same as that of the period register, the timer sets the period interrupt flag, resets to 0, sets the underflow interrupt flag, and then continues the operation again as if the initial value was 0. If the initial value of the timer is between 0 and the contents of the period register, the timer counts up to the period value and continue to finish the period as if the initial counter value was the same as that of the period register.

The counting direction indication bit in GPTCONA/B is 1 for the timer in this mode. Either the external or internal device clock can be selected as the input clock to the timer. TDIRA/B input is ignored by the GP timer in this counting mode.

The continuous up-counting mode of the GP timer is particularly useful for the generation of edge-triggered or asynchronous PWM waveforms and sampling periods in many motor and motion control systems.

Figure 6–4 shows the continuous up-counting mode of the GP timer.

Figure 6–4. GP Timer Continuous Up-Counting Mode (TxPR = 3 or 2)



As shown in Figure 6–4, *GP Timer Continuous Up-Counting Mode (TxPR = 3 or 2)*, no clock cycle is missed from the time the counter reaches the period register value to the time it starts another counting cycle.

Directional Up-/Down-Counting Mode

The GP timer in directional up-/down-counting mode counts up or down according to the scaled clock and TDIRA/B inputs. The GP timer starts counting up until its value reaches that of the period register (or FFFFh if the initial count is greater than the period) when the TDIRA/B pin is held high. When the timer value equals that of its period register (or FFFFh) the timer resets to zero and continues counting up to the period again. When TDIRA/B is held low, the GP timer counts down until its value becomes 0. When the value of the timer has counted down to 0, the timer reloads its counter with the value in the period register and starts counting down again.

The initial value of the timer can be any value between 0000h to FFFFh. When the initial value of the timer counter is greater than that of the period register, the timer counts up to FFFFh before resetting itself to 0 and counting up to the period. If TDIRA/B is low when the timer starts with a value greater than the period register, it counts down to the value of the period register and continues counting down to 0, at which point the timer counter gets reloaded with the value from the period register as normal.

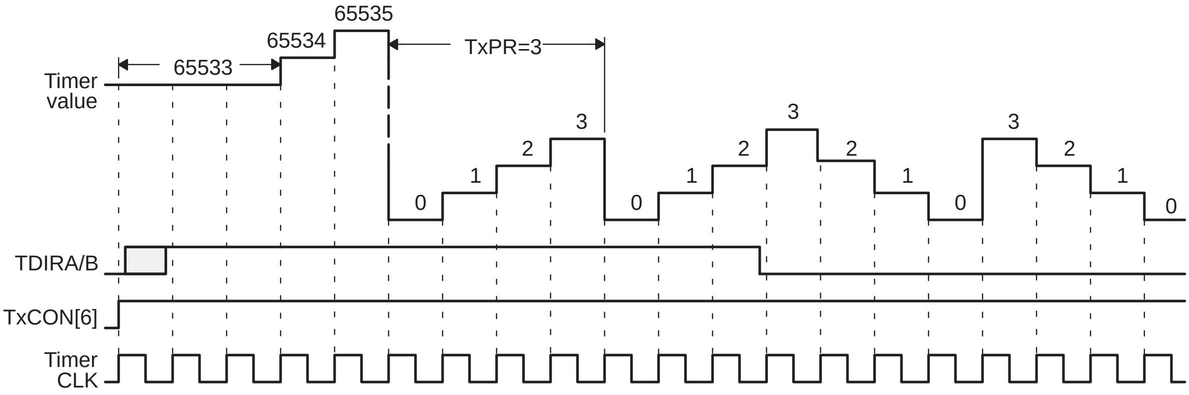
The period, underflow, and overflow interrupt flags, interrupts, and associated actions are generated on respective events in the same manner as they are generated in the continuous up-counting mode.

The latency from a change of TDIRA/B to a change of counting direction is one clock cycle after the end of the current count (that is, after the end of the current prescale counter period).

The direction of counting is indicated for the timer in this mode by the corresponding direction indication bit in GPTCONA/B: 1 means counting up; 0 means counting down. Either the external clock from the TCLKINA/B pin or the Internal device clock can be used as the input clock for the timer in this mode.

Figure 6–5 shows the directional up-/down-counting mode of the GP timers.

Figure 6–5. GP Timer Directional Up-/Down-Counting Mode: Prescale Factor 1 and TxPR = 3



The directional up-/down-counting mode of GP timer 2/4 can be used with the quadrature encoder pulse (QEP) circuits in the EV module. The QEP circuits provide both the counting clock and direction for GP timer 2/4 in this case. This mode of operation can also be used to time the occurrence of external events in motion/motor control and power electronics applications.

Continuous Up-/Down-Counting Mode

This mode of operation is the same as the directional up-/down-counting mode, but the TDIRA/B pin has no effect on the counting direction. The counting direction only changes from up to down when the timer reaches the period value (or FFFFh if the initial timer value is greater than the period). The timer direction only changes from down to up when the timer reaches 0.

The period of the timer in this mode is 2*(TxPR) cycles of the scaled clock input except for the first period. The duration of the first counting period is the same if the timer counter is 0 when counting starts.

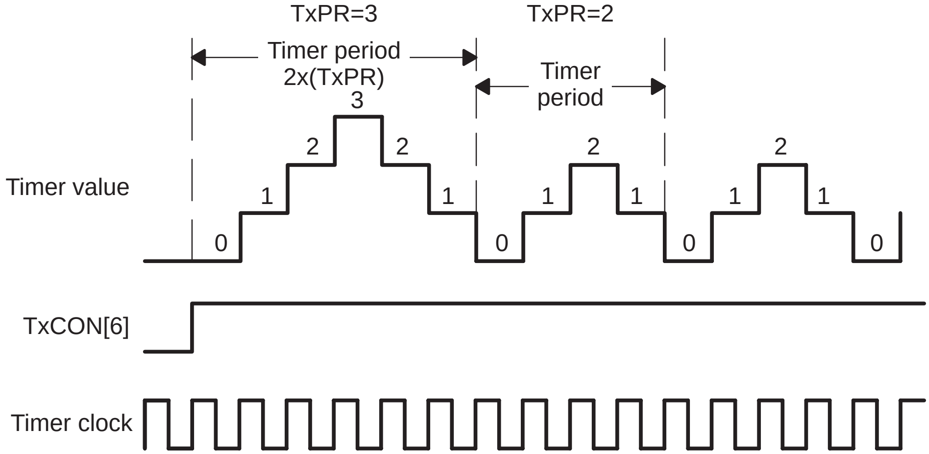
The initial value of the GP timer counter can be any value between 0h and FFFFh inclusive. When the initial value is greater than that of the period register, the timer counts up to FFFFh, resets to 0, and continues the operation as if the initial value was 0. When the initial value in the timer counter is the same as that of the period register, the timer counts down to 0 and continues again as if the initial value was 0. If the initial value of the timer is between 0 and the contents of the period register, the timer counts up to the period value and continues to finish the period as if the initial counter value was the same as that of the period register.

The period, underflow, and overflow interrupt flags, interrupts, and associated actions are generated on respective events in the same manner as they are generated in continuous up-counting mode.

The counting direction indication bit for this timer in GPTCONA/B is 1 when the timer counts upward and 0 when the timer counts downward. Either the external clock from the TCLKINA/B pin or the internal device clock can be selected as the input clock. TDIRA/B input is ignored by the timer in this mode.

Figure 6–6 shows the continuous up-/down-counting mode of the GP timer.

Figure 6–6. GP Timer Continuous Up-/Down-Counting Mode (TxPR = 3 or 2)



Continuous up-/down-counting mode is particularly useful in generating centered or symmetric PWM waveforms found in a broad range of motor/motion control and power electronics applications.

6.3.2 GP Timer Compare Operation

Each GP timer has an associated compare register TxCMPR and a PWM output pin TxPWM. The value of a GP timer counter is constantly compared to that of its associated compare register. A compare match occurs when the value of the timer counter is the same as that of the compare register. Compare operation is enabled by setting TxCON[1] to 1. If it is enabled, the following happens on a compare match:

- The compare interrupt flag of the timer is set one clock cycle after the match.
- A transition occurs on the associated PWM output according to the bit configuration in GPTCONA/B, one device clock cycle after the match.
- If the compare interrupt flag has been selected by the appropriate GPTCONA/B bits to start ADC, an ADC start signal is generated at the same time the compare interrupt flag is set.

A peripheral interrupt request is generated by the compare interrupt flag if it is unmasked.

PWM Transition

The transition on the PWM output is controlled by an asymmetric and symmetric waveform generator and the associated output logic, and depends on the following:

- Bit definition in GPTCONA/B
- Counting mode the timer is in
- Counting direction when the counting mode is continuous-up/-down mode

Asymmetric/Symmetric Waveform Generator

The asymmetric/symmetric waveform generator generates an asymmetric or symmetric PWM waveform based on the counting mode the GP timer is in.

Asymmetric Waveform Generation

An asymmetric waveform (Figure 6–7) is generated when the GP timer is in continuous up-counting mode. When the GP timer is in this mode, the output of the waveform generator changes according to the following sequence:

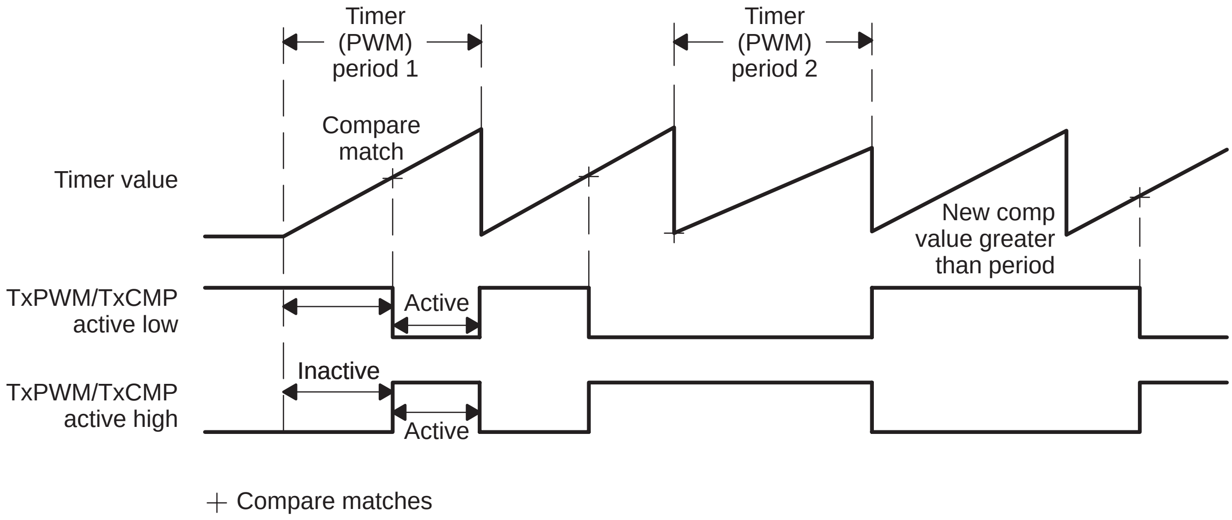
- 0 before the counting operation starts
- remains unchanged until the compare match happens

- ❑ toggles on compare match
- ❑ remains unchanged until the end of the period
- ❑ resets to 0 at the end of a period on period match, if the new compare value for the following period is not 0

The output is 1 for the whole period, if the compare value is 0 at the beginning of a period. The output does not reset to 0 if the new compare value for the following period is 0. This is important because it allows the generation of PWM pulses of 0% to 100% duty cycle without glitches. The output is 0 for the whole period if the compare value is greater than the value in the period register. The output is 1 for one cycle of the scaled clock input if the compare value is the same as that of the period register.

One characteristic of asymmetric PWM waveforms is that a change in the value of the compare register only affects one side of the PWM pulse.

Figure 6–7. GP Timer Compare/PWM Output in Up-Counting Mode



Symmetric Waveform Generation

A symmetric waveform (Figure 6–8) is generated when the GP timer is in continuous up-/down-counting modes. When the GP timer is in this mode, the state of the output of the waveform generator is determined by the following:

- ❑ 0 before the counting operation starts
- ❑ Remains unchanged until first compare match

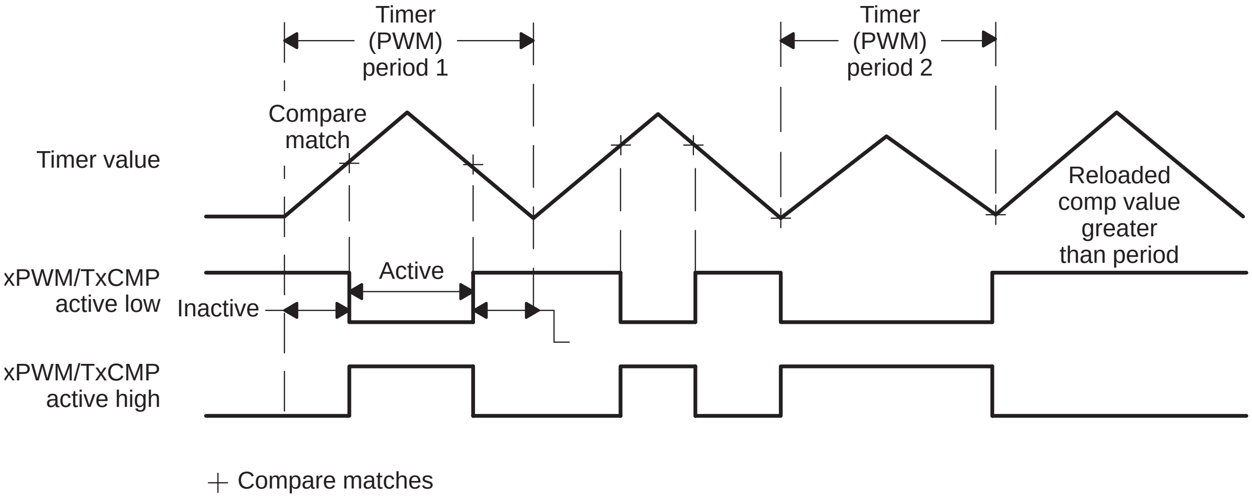
- ❑ Toggles on the first compare match
- ❑ Remains unchanged until the second compare match
- ❑ Toggles on the second compare match
- ❑ Remains unchanged until the end of the period
- ❑ Resets to 0 at the end of the period if there is no second compare match, and the new compare value for the following period is not 0

The output is set to 1 at the beginning of a period and remains 1 until the second compare match if the compare value is 0 at the beginning of a period. After the first transition, the output remains 1 until the end of the period if the compare value is 0 for the second half of the period. When this happens, the output does not reset to 0 if the new compare value for the following period is still 0. This is done again to assure the generation of PWM pulses of 0% to 100% duty cycle without any glitches. The first transition does not happen if the compare value is greater than or equal to that of the period register for the first half of the period. However, the output still toggles when a compare match happens in the second half of the period. This error in output transition, often as a result of calculation error in the application routine, is corrected at the end of the period because the output resets to 0, unless the new compare value for the following period is 0. In this case, the output remains 1, which again puts the output of the waveform generator in the correct state.

Note:

The output logic determines what the active state is for all output pins.

Figure 6–8. GP Timer Compare/PWM Output in Up-/Down-Counting Modes



Output Logic

The output logic further conditions the output of the waveform generator to form the ultimate PWM output that controls different kinds of power devices. The PWM output can be specified active high, active low, forced low, and forced high by proper configuration of the appropriate GPTCONA/B bits.

The polarity of the PWM output is the same as that of the output of the associated asymmetric/symmetric waveform generator when the PWM output is specified active high.

The polarity of the PWM output is the opposite of that of the output of the associated asymmetric/symmetric waveform generator when the PWM output is specified active low.

The PWM output is set to 1 (or 0) immediately after the corresponding bits in GPTCONA/B are set, and the bit pattern specifies that the state of PWM output is forced high (or low).

In summary, during a normal counting mode, transitions on the GP timer PWM outputs happen according to Table 6–11 for the continuous up-counting mode and according to Table 6–12 for the continuous up-/down-counting mode, assuming compare is enabled.

Setting active means setting high for active high and setting low for active low. Setting inactive means the opposite.

The asymmetric/symmetric waveform generation, based on the timer counting mode and the output logic, is also applicable to the compare units.

Table 6–11. GP Timer Compare Output in Continuous Up-Counting Modes

Time in a period	**State of Compare Output**
Before compare match	Inactive
On compare match	Set active
On period match	Set inactive

Table 6–12. GP Timer Compare Output in Continuous Up-/Down-Counting Modes

Time in a period	**State of Compare Output**
Before 1st compare match	Inactive
On 1st compare match	Set active
On 2nd compare match	Set inactive
After 2nd compare match	Inactive

All GP timer PWM outputs are put in the high-impedance state when any of the following events occurs:

- ❑ GPTCONA/B[6] is set to 0 by software
- ❑ $\overline{\text{PDPINTx}}$ is pulled low and is not masked
- ❑ Any reset event occurs
- ❑ TxCON[1] is set to 0 by software

Active/Inactive Time Calculation

For the continuous up-counting mode, the value in the compare register represents the elapsed time between the beginning of a period and the occurrence of the first compare match, that is, the length of the inactive phase. This elapsed time is equal to the period of the scaled input clock multiplied by the value of TxCMPR. Therefore, the length of the active phase (the output pulse width) is given by (TxPR) – (TxCMPR) + 1 cycle of the scaled input clock.

For the continuous up-/down-counting mode, the compare register can have a different value while counting down from the value while counting up. The length of the active phase, that is, the output pulse width, for up-/down-counting modes is given by (TxPR) – $(\text{TxCMPR})_{\text{up}}$ + (TxPR) – $(\text{TxCMPR})_{\text{dn}}$ cycles of the scaled input clock, where $(\text{TxCMPR})_{\text{up}}$ is the compare value on the way up and $(\text{TxCMPR})_{\text{dn}}$ is the compare value on the way down.

When the value in TxCMPR is 0, the GP timer compare output is active for the whole period if the timer is in the up-counting mode. For the up-/down-counting mode, the compare output is active at the beginning of the period if $(\text{TxCMPR})_{\text{up}}$ is 0. The output remains active until the end of the period if $(\text{TxCMPR})_{\text{dn}}$ is also 0.

The length of the active phase (the output pulse width) is 0 when the value of TxCMPR is greater than that of TxPR for up-counting modes. For the up-/down-counting mode, the first transition is lost when $(\text{TxCMPR})_{\text{up}}$ is greater than or equal to (TxPR). Similarly, the second transition is lost when $(\text{TxCMPR})_{\text{dn}}$ is greater than or equal to (TxPR). The GP timer compare output is inactive for the entire period if both $(\text{TxCMPR})_{\text{up}}$ and $\text{TxCMPR})_{\text{dn}}$ are greater than or equal to (TxPR) for the up-/down-counting mode.

Figure 6–7, *GP Timer Compare/PWM Output in Up-Counting Mode* (page 6-27) shows the compare operation of a GP timer in the up counting mode. Figure 6–8, *GP Timer Compare/PWM Output in Up-/Down-Counting Modes* (page 6-28) shows the compare operation of a GP timer in the up-/down-counting mode.

6.3.3 Timer Control Registers (TxCON and GPTCONA/B)

The addresses of the GP timer registers are given in Table 6–3 and Table 6–4 on page 6-11. The bit definition of the individual GP timer control registers, TxCON, is shown in Figure 6–9. The bit definition of the overall GP timer control registers, GPTCONA and GPTCONB, are shown in Figure 6–10 (on page 6-33) and Figure 6–11 (on page 6-34), respectively.

Individual GP Timer Control Register (TxCON; x = 1, 2, 3, or 4)

Figure 6–9. Timer Control Register (TxCON; x = 1, 2, 3, or 4) — Addresses 7404h (T1), 7408h (T2), 7504h (T3), and 7508h (T4)

15	14	13	12	11	10	9	8
Free	Soft	Reserved	TMODE1	TMODE0	TPS2	TPS1	TPS0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
T2SWT1/ T4SWT3	TENABLE	TCLKS1	TCLKS0	TCLD1	TCLD0	TECMPR	SELT1PR/ SELT3PR
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–14 **Free, Soft.** Emulation control bits.

00 Stop immediately on emulation suspend.

01 Stop after current timer period is complete on emulation suspend.

10 Operation is not affected by emulation suspend.

11 Operation is not affected by emulation suspend.

Bit 13 **Reserved**. Reads return zero, writes have no effect.

Bits 12–11 **TMODE1–TMODE0.** Count Mode Selection.

00 Stop/Hold

01 Continuous-Up/-Down Count Mode

10 Continuous-Up Count Mode

11 Directional-Up/-Down Count Mode

Bits 10–8 **TPS2–TPS0.** Input Clock Prescaler.

000	x/1	100	x/16
001	x/2	101	x/32
010	x/4	110	x/64
011	x/8	111	x/128

x = device (CPU) clock frequency

Bit 7 **T2SWT1.** In case of EVA, this bit is T2SWT1. (GP timer 2 start with GP timer 1.) Start GP timer 2 with GP timer 1’s timer enable bit. This bit is reserved in T1CON.

T4SWT3. In case of EVB, this bit is T4SWT3. (GP timer 4 start with GP timer 3.) Start GP timer 4 with GP timer 3’s timer enable bit. This bit is reserved in T3CON.

0 Use own TENABLE bit.

1 Use TENABLE bit of T1CON to enable and disable operation ignoring own TENABLE bit.

Bit 6 **TENABLE**. Timer enable.

0 Disable timer operation (the timer is put in hold and the prescaler counter is reset).

1 Enable timer operations.

Bits 5–4 **TCLKS1, TCLKS0.** Clock Source Select.

5	4	Timer 1	Timer 2
0	0	Internal	Internal
0	1	External	External
1	0	Reserved	Reserved
1	1	Reserved	QEP Circuit[†]

[†] This option is valid only if SELT1PR = 0

Bits 3–2 **TCLD1, TCLD0**. Timer Compare Register Reload Condition.

00 When counter is 0.

01 When counter value is 0 or equals period register value.

10 Immediately

11 Reserved

Bit 1 **TECMPR**. Timer compare enable.

0 Disable timer compare operation.

1 Enable timer compare operation.

Bit 0 **SELT1PR**. In case of EVA, this bit is SELT1PR. (Period register select.) This bit is a reserved bit in T1CON.
SELT3PR. In case of EVB, this bit is SELT3PR. (Period register select.) This bit is a reserved bit in T3CON.

0 Use own period register.

1 Use T1PR (in case of EVA) or T3PR (in case of EVB) as period register ignoring own period register.

Overall GP Timer Control Register (GPTCONA)

Figure 6–10. GP Timer Control Register A (GPTCONA) — Address 7400h

15	14	13	12–11	10–9	8–7
Reserved	T2STAT	T1STAT	Reserved	T2TOADC	T1TOADC
RW-0	R-1	R-1	RW-0	RW-0	RW-0

6	5–4	3–2	1–0
TCOMPOE	Reserved	T2PIN	T1PIN
RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -n = value after reset

Bit 15 **Reserved**. Reads return zero; writes have no effect.

Bit 14 **T2STAT**. GP timer 2 Status. Read only.

0 Counting downward

1 Counting upward

Bit 13 **T1STAT**. GP timer 1 Status. Read only.

0 Counting downward

1 Counting upward

Bits 12–11 **Reserved**. Reads return zero; writes have no effect.

Bits 10–9 **T2TOADC**. Start ADC with timer 2 event.

00 No event starts ADC.

01 Setting of underflow interrupt flag starts ADC.

10 Setting of period interrupt flag starts ADC.

11 Setting of compare interrupt flag starts ADC.

Bits 8–7 **T1TOADC**. Start ADC with timer 1 event.

00 No event starts ADC.

01 Setting of underflow interrupt flag starts ADC.

10 Setting of period interrupt flag starts ADC.

11 Setting of compare interrupt flag starts ADC.

Bit 6 **TCOMPOE**. Compare output enable. If $\overline{\text{PDPINTx}}$ is active this bit is set to zero.

0 Disable all GP timer compare outputs (all compare outputs are put in the high-impedance state).

1 Enable all GP timer compare outputs.

Bits 5–4 **Reserved**. Reads return zero; writes have no effect.

Bits 3–2 **T2PIN**. Polarity of GP timer 2 compare output.

00 Forced low

01 Active low

10 Active high

11 Forced high

Bits 1–0 **T1PIN**. Polarity of GP timer 1 compare output.

00 Forced low

01 Active low

10 Active high

11 Forced high

Overall GP Timer Control Register (GPTCONB)

Figure 6–11. GP Timer Control Register B (GPTCONB) — Address 7500h

15	14	13	12–11	10–9	8–7
Reserved	T4STAT	T3STAT	Reserved	T4TOADC	T3TOADC
RW-0	R-1	R-1	RW-0	RW-0	RW-0

6	5–4	3–2	1–0
TCOMPOE	Reserved	T4PIN	T3PIN
RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -n = value after reset

Bit 15 **Reserved**. Reads return zero; writes have no effect.

Bit 14 **T4STAT**. GP timer 4 Status. Read only.

0 Counting downward

1 Counting upward

Bit 13 **T3STAT**. GP timer 3 Status. Read only.

0 Counting downward

1 Counting upward

Bits 12–11 **Reserved**. Reads return zero; writes have no effect.

Bits 10–9 **T4TOADC**. Start ADC with timer 4 event.

00 No event starts ADC.

01 Setting of underflow interrupt flag starts ADC.

10 Setting of period interrupt flag starts ADC.

11 Setting of compare interrupt flag starts ADC.

Bits 8–7 **T3TOADC**. Start ADC with timer 3 event.

00 No event starts ADC.

01 Setting of underflow interrupt flag starts ADC.

10 Setting of period interrupt flag starts ADC.

11 Setting of compare interrupt flag starts ADC.

Bit 6 **TCOMPOE**. Compare output enable. If $\overline{\text{PDPINTx}}$ is active this bit is set to zero.

0 Disable all GP timer compare outputs (all compare outputs are put in the high-impedance state).

1 Enable all GP timer compare outputs.

Bits 5–4 **Reserved**. Reads return zero; writes have no effect.

Bits 3–2 **T4PIN**. Polarity of GP timer 4 compare output.

00 Forced low

01 Active low

10 Active high

11 Forced high

Bits 1–0 **T3PIN**. Polarity of GP timer 3 compare output.

00 Forced low

01 Active low

10 Active high

11 Forced high

6.3.4 Generation of PWM Outputs Using the GP Timers

Each GP timer can independently be used to provide a PWM output channel. Thus, up to two PWM outputs may be generated by the GP timers.

PWM Operation

To generate a PWM output with a GP timer, a continuous up- or up-/down-counting mode can be selected. Edge-triggered or asymmetric PWM waveforms are generated when a continuous-up count mode is selected. Centered or symmetric PWM waveforms are generated when a continuous-up/-down mode is selected. To set up the GP timer for the PWM operation, do the following:

- Set up TxPR according to the desired PWM (carrier) period.
- Set up TxCON to specify the counting mode and clock source, and start the operation.
- Load TxCMPR with values corresponding to the on-line calculated widths (duty cycles) of PWM pulses.

The period value is obtained by dividing the desired PWM period by the period of the GP timer input clock, and subtracting one from the resulting number when the continuous up-counting mode is selected to generate asymmetric PWM waveforms. When the continuous up-/down-counting mode is selected to generate symmetric PWM waveforms, this value is obtained by dividing the desired PWM period by two times the period of the GP timer input clock.

The GP timer can be initialized the same way as in the previous example. During run time, the GP timer compare register is constantly updated with newly determined compare values corresponding to the newly determined duty cycles.

6.3.5 GP Timer Reset

When any RESET event occurs, the following happens:

- All GP timer register bits, except for the counting direction indication bits in GPTCONA/B, are reset to 0; thus, the operation of all GP timers is disabled. The counting direction indication bits are all set to 1.
- All timer interrupt flags are reset to 0.
- All timer interrupt mask bits are reset to 0, except for $\overline{\text{PDPINTx}}$; thus, all GP timer interrupts are masked, except for $\overline{\text{PDPINTx}}$.
- All GP timer compare outputs are put in the high-impedance state.

6.4 Compare Units

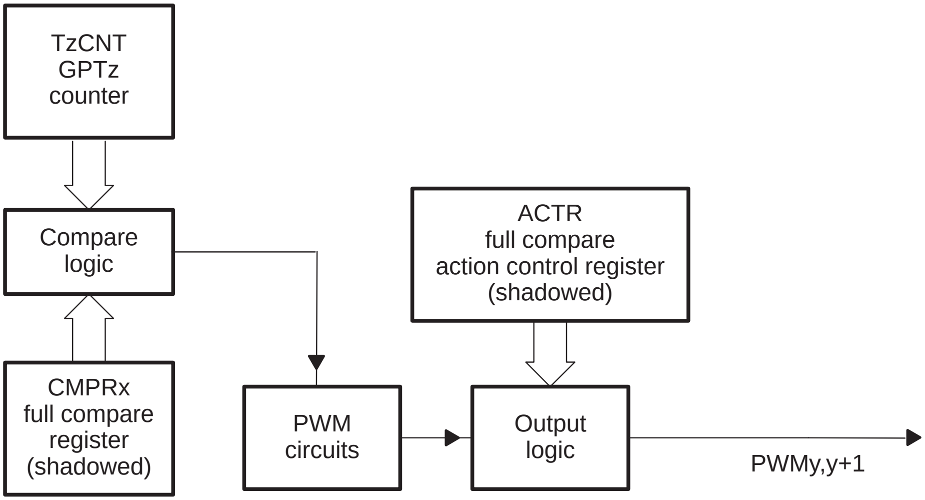
There are three (full) compare units (compare units 1, 2, and 3) in the EVA module and three (full) compare units (compare units 4, 5, and 6) in the EVB module. Each compare unit has two associated PWM outputs. The time base for the compare units is provided by GP timer 1 (for EVA) and by GP timer 2 (for EVB).

The compare units in each EV module include:

- ❑ Three 16-bit compare registers (CMPR1, CMPR2, and CMPR3 for EVA; and CMPR4, CMPR5, and CMPR6 for EVB), all with an associated shadow register, (RW)
- ❑ One 16-bit compare control register (COMCONA for EVA, and COMCONB for EVB), (RW)
- ❑ One 16-bit action control register (ACTRA for EVA, and ACTRB for EVB), with associated shadow register, (RW)
- ❑ Six PWM (3-state) output (compare output) pins (PWMy, y = 1, 2, 3, 4, 5, 6 for EVA and PWMz, z = 7, 8, 9, 10, 11, 12 for EVB)
- ❑ Control and interrupt logic

The functional block diagram of a compare unit is shown in Figure 6–12.

Figure 6–12. Compare Unit Block Diagram
(For EVA: x = 1, 2, 3; y = 1, 3, 5; z = 1.
For EVB: x = 4, 5, 6; y = 7, 9, 11; z = 2)



The time base for the compare units and the associated PWM circuits is provided by GP timer 1 (for EVA) or GP timer 2 (for EVB), which can be in any of its counting modes when the compare operation is enabled. Transitions occur on the compare outputs.

Compare Inputs/Outputs

The inputs to a compare unit include:

- ❑ Control signals from control registers
- ❑ GP timer 1/3 (T1CNT/T3CNT) and its underflow and period match signals
- ❑ RESET

The output of a compare unit is a compare match signal. If the compare operation is enabled, this match signal sets the interrupt flag and causes transitions on the two output pins associated with the compare unit.

Compare Operation Modes

The operation mode of the compare units is determined by the bits in COMCONx. These bits determine:

- ❑ Whether the compare operation is enabled
- ❑ Whether the compare outputs are enabled
- ❑ The condition on which the compare registers are updated with the values in their shadow registers
- ❑ Whether space vector PWM mode is enabled

Operation

The following paragraph describes the operation of the EVA compare unit. The operation of the EVB compare unit is identical. For EVB, GP timer 3 and ACTRB are used.

The value of the GP timer 1 counter is continuously compared with that of the compare register. When a match is made, a transition appears on the two outputs of the compare unit according to the bits in the action control register (ACTRA). The bits in ACTRA can individually specify each output to be toggle active high or toggle active low (if not forced high or low) on a compare match. The compare interrupt flag associated with a compare unit is set when a compare match is made between GP timer 1 and the compare register of this compare unit, if compare is enabled. A peripheral interrupt request is generated by the flag if the interrupt is unmasked. The timing of output transitions, setting of interrupt flags, and generation of interrupt requests are the same as that of the GP timer compare operation. The outputs of the compare units in compare mode are subject to modification by the output logic, dead band units, and the space vector PWM logic.

Register Setup for Compare Unit Operation

The register setup sequence for compare unit operation requires:

For EVA	For EVB
Setting up T1PR	Setting up T3PR
Setting up ACTRA	Setting up ACTRB
Initializing CMPRx	Initializing CMPRx
Setting up COMCONA	Setting up COMCONB
Setting up T1CON	Setting up T3CON

6.4.1 Compare Units Registers

The addresses of registers associated with compare units and associated PWM circuits are shown in Table 6–5, *Addresses of EVA Compare Control Registers* (page 6-11) and Table 6–6, *Addresses of EVB Compare Control Registers* (page 6-12), and are further discussed in the following subsections.

Compare Control Registers (COMCONA and COMCONB)

The operation of the compare units is controlled by the compare control registers (COMCONA and COMCONB). The bit definition of COMCONA is summarized in Figure 6–13 and that of COMCONB is summarized in Figure 6–14. COMCONA and COMCONB are readable and writeable.

Figure 6–13. Compare Control Register A (COMCONA) — Address 7411h

15	14	13	12	11	10	9	8
CENABLE	CLD1	CLD0	SVENABLE	ACTRLD1	ACTRLD0	FCOMPOE	Reserved
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7–0
Reserved
R-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 15 **CENABLE**. Compare enable.

0 Disable compare operation. All shadowed registers (CMPRx, ACTRA) become transparent.

1 Enable compare operation.

Bits14–13 **CLD1, CLD0**. Compare register CMPRx reload condition.

00 When T1CNT = 0 (that is, on underflow)

01 When T1CNT = 0 or T1CNT = T1PR (that is, on underflow or period match)

10 Immediately

11 Reserved; result is unpredictable.

Bit 12 **SVENABLE**. Space vector PWM mode enable.

0 Disable space vector PWM mode.

1 Enable space vector PWM mode.

Bits 11–10 **ACTRLD1, ACTRLD0**. Action control register reload condition.

00 When T1CNT = 0 (on underflow)

01 When T1CNT = 0 or T1CNT = T1PR (on underflow or period match)

10 Immediately

11 Reserved

Bit 9 **FCOMPOE**. Compare output enable. Active $\overline{\text{PDPINTx}}$ clears this bit to zero.

0 PWM output pins are in high-impedance state; that is, they are disabled.

1 PWM output pins are not in high-impedance state; that is, they are enabled.

Bits 8–0 **Reserved**. Read returns zero; writes have no effect.

Figure 6–14. Compare Control Register B (COMCONB) — Address 7511h

15	14	13	12	11	10	9	8
CENABLE	CLD1	CLD0	SVENABLE	ACTRLD1	ACTRLD0	FCOMPOE	Reserved
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7–0
Reserved
R-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 15 **CENABLE**. Compare enable.

0 Disable compare operation. All shadowed registers (CMPRx, ACTRB) become transparent.

1 Enable compare operation.

Bits14–13 **CLD1, CLD0**. Compare register CMPRx reload condition.

00 When T3CNT = 0 (that is, on underflow)

01 When T3CNT = 0 or T3CNT = T3PR (that is, on underflow or period match)

10 Immediately

11 Reserved; result is unpredictable.

Bit 12 **SVENABLE**. Space vector PWM mode enable.

0 Disable space vector PWM mode.

1 Enable space vector PWM mode.

Bits 11–10 **ACTRLD1, ACTRLD0**. Action control register reload condition.

00 When T3CNT = 0 (on underflow)

01 When T3CNT = 0 or T3CNT = T3PR (on underflow or period match)

10 Immediately

11 Reserved

Bit 9 **FCOMPOE**. Compare output enable. Active $\overline{\text{PDPINTx}}$ clears this bit to zero.

0 PWM output pins are in high-impedance state; that is, they are disabled.

1 PWM output pins are not in high-impedance state; that is, they are enabled.

Bits 8–0 **Reserved**. Read returns zero; writes have no effect.

Compare Action Control Registers (ACTRA and ACTRB)

The compare action control registers (ACTRA and ACTRB) control the action that takes place on each of the six compare output pins (PWMx, where x = 1–6 for ACTRA, and x = 7–12 for ACTRB) on a compare event, if the compare operation is enabled by COMCONx[15]. ACTRA and ACTRB are double-buffered. The condition on which ACTRA and ACTRB is reloaded is defined by bits in COMCONx. ACTRA and ACTRB also contain the SVRDIR, D2, D1, and D0 bits needed for space vector PWM operation. The bit configuration of ACTRA is described in Figure 6–15 and that of ACTRB is described in Figure 6–16.

Figure 6–15. Compare Action Control Register A (ACTRA) — Address 7413h

15	14	13	12	11	10	9	8
SVRDIR	D2	D1	D0	CMP6ACT1	CMP6ACT0	CMP5ACT1	CMP5ACT0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
CMP4ACT1	CMP4ACT0	CMP3ACT1	CMP3ACT0	CMP2ACT1	CMP2ACT0	CMP1ACT1	CMP1ACT0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 15 **SVRDIR**. Space vector PWM rotation direction. Used only in space vector PWM output generation.

0 Positive (CCW)

1 Negative (CW)

Bits 14–12 **D2–D0**. Basic space vector bits. Used only in space vector PWM output generation.

Bits 11–10 **CMP6ACT1–0**. Action on compare output pin 6, CMP6.

00	Forced low
01	Active low
10	Active high
11	Forced high

Bits 9–8 **CMP5ACT1–0**. Action on compare output pin 5, CMP5.

00	Forced low
01	Active low
10	Active high
11	Forced high

Bits 7–6 **CMP4ACT1–0**. Action on compare output pin 4, CMP4.

00	Forced low
01	Active low
10	Active high
11	Forced high

Bits 5–4 **CMP3ACT1–0**. Action on compare output pin 3, CMP3.

00	Forced low
01	Active low
10	Active high
11	Forced high

Bits 3–2 **CMP2ACT1–0**. Action on compare output pin 2, CMP2.

00	Forced low
01	Active low
10	Active high
11	Forced high

Bits 1–0 **CMP1ACT1–0**. Action on compare output pin 1, CMP1.

00	Forced low
01	Active low
10	Active high
11	Forced high

Figure 6–16. Compare Action Control Register B (ACTRB) — Address 7513h

15	14	13	12	11	10	9	8
SVRDIR	D2	D1	D0	CMP12ACT1	CMP12ACT0	CMP11ACT1	CMP11ACT0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
CMP10ACT1	CMP10ACT0	CMP9ACT1	CMP9ACT0	CMP8ACT1	CMP8ACT0	CMP7ACT1	CMP7ACT0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 15 **SVRDIR**. Space vector PWM rotation direction. Used only in space vector PWM output generation.

0 Positive (CCW)

1 Negative (CW)

Bits 14–12 **D2–D0**. Basic space vector bits. Used only in space vector PWM output generation.

Bits 11–10 **CMP12ACT1–0**. Action on compare output pin 12, CMP12.

00	Forced low
01	Active low
10	Active high
11	Forced high

Bits 9–8 **CMP11ACT1–0**. Action on compare output pin 11, CMP11.

00	Forced low
01	Active low
10	Active high
11	Forced high

Bits 7–6 **CMP10ACT1–0**. Action on compare output pin 10, CMP10.

00	Forced low
01	Active low
10	Active high
11	Forced high

Bits 5–4 **CMP9ACT1–0**. Action on compare output pin 9, CMP9.

00	Forced low
01	Active low
10	Active high
11	Forced high

Bits 3–2 **CMP8ACT1–0**. Action on compare output pin 8, CMP8.

00	Forced low
01	Active low
10	Active high
11	Forced high

Bits 1–0 **CMP7ACT1–0**. Action on compare output pin 7, CMP7.

00	Forced low
01	Active low
10	Active high
11	Forced high

6.4.2 Compare Unit Interrupts

There is a maskable interrupt flag in EVIFRA and EVIFRC for each compare unit. The interrupt flag of a compare unit is set one clock cycle after a compare match, if compare operation is enabled. A peripheral interrupt request is generated by the flag if it is unmasked.

6.4.3 Compare Unit Reset

When any reset event occurs, all register bits associated with the compare units are reset to 0 and all compare output pins are put in the high-impedance state.

6.5 PWM Circuits Associated With Compare Units

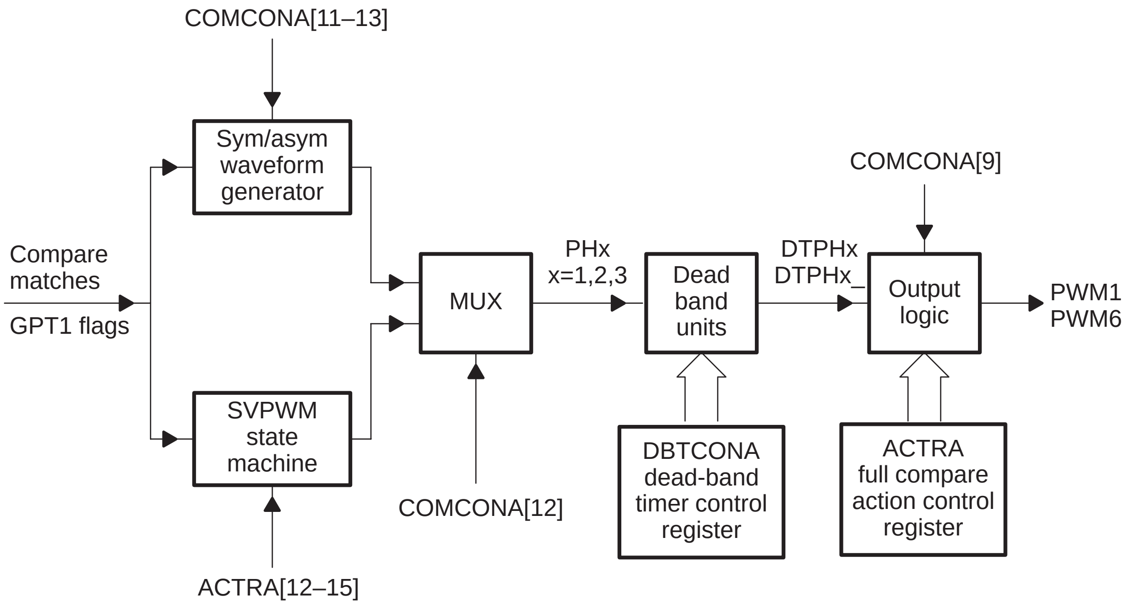
The PWM circuits associated with compare units make it possible to generate six PWM output channels (per EV) with programmable dead-band and output polarity. The EVA PWM circuits functional block diagram is shown in Figure 6–17. It includes the following functional units:

- ❑ Asymmetric/Symmetric Waveform Generators
- ❑ Programmable Dead-Band Unit (DBU)
- ❑ Output Logic
- ❑ Space Vector (SV) PWM State Machine

The EVB PWM circuits functional block diagram is identical to that of the EVA's, with the corresponding change of configuration registers.

The asymmetric/symmetric waveform generators are the same as those of the GP timers. The dead-band units and output logic are discussed in sections 6.5.2 and 6.5.3, respectively. The space vector PWM state machine and the space vector PWM technique are described later in this chapter.

Figure 6–17. PWM Circuits Block Diagram



The PWM circuits are designed to minimize CPU overhead and user intervention when generating pulse width modulated waveforms used in motor control and motion control applications. PWM generation with compare units and associated PWM circuits are controlled by the following control registers: T1CON, COMCONA, ACTRA, and DBTCONA (in case of EVA); and T3CON, COMCONB, ACTRB, and DBTCONB (in case of EVB).

6.5.1 PWM Generation Capability of Event Manager

The PWM waveform generation capability of each event manager module (A and B) is summarized as follows:

- Five independent PWM outputs, three of which are generated by the compare units; the other two are generated by the GP timer compares, plus three additional PWM outputs dependent on the three compare unit PWM outputs
- Programmable dead-band for the PWM output pairs associated with the compare units
- Minimum dead-band duration of one device clock cycle
- Minimum PWM pulsewidth and pulsewidth increment/decrement of one clock cycle
- 16-bit maximum PWM resolution
- On-the-fly change of PWM carrier frequency (double buffered period registers)
- On-the-fly change of PWM pulsewidths (double buffered compare registers)
- Power Drive Protection Interrupt
- Programmable generation of asymmetric, symmetric, and space vector PWM waveforms
- Minimum CPU overhead because of the auto-reloading of the compare and period registers

6.5.2 Programmable Dead-Band (Dead-Time) Unit

EVA and EVB have their own programmable dead-band units (DBTCONA and DBTCONB, respectively)The programmable dead-band unit features:

- One 16-bit dead-band control register, DBTCONx (RW)
- One input clock prescaler: x/1, x/2, x/4, etc., to x/32
- Device (CPU) clock input
- Three 4-bit down counting timers
- Control logic

Dead-Band Timer Control Registers A and B (DBTCONA and DBTCONB)

The operation of the dead-band unit is controlled by the dead-band timer control registers (DBTCONA and DBTCONB). The bit description of DBTCONA is given in Figure 6–18 and that of DBTCONB is given in Figure 6–19.

Figure 6–18. Dead-Band Timer Control Register A (DBTCONA) — Address xx15h

15–12	11	10	9	8
Reserved	DBT3	DBT2	DBT1	DBT0
R-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1–0
EDBT3	EDBT2	EDBT1	DBTPS2	DBTPS1	DBTPS0	Reserved
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	R-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–12 **Reserved**. Reads return zero; writes have no effect.

Bits 11–8 **DBT3 (MSB)–DBT0 (LSB)**. Dead-band timer period. These bits define the period value of the three 4-bit dead-band timers.

Bit 7 **EDBT3**. Dead-band timer 3 enable (for pins PWM5 and PWM6 of Compare Unit 3).

0	Disable
1	Enable

Bit 6 **EDBT2**. Dead-band timer 2 enable (for pins PWM3 and PWM4 of Compare Unit 2).

0	Disable
1	Enable

Bit 5 **EDBT1**. Dead-band timer 1 enable (for pins PWM1 and PWM2 of Compare Unit 1).

0	Disable
1	Enable

Bits 4–2 **DBTPS2 to DBTPS0**. Dead-band timer prescaler.

000	x/1
001	x/2
010	x/4
011	x/8
100	x/16
101	x/32
110	x/32
111	x/32
	x = Device (CPU) clock frequency

Bits 1–0 **Reserved**. Reads return zero; writes have no effect.

Figure 6–19. Dead-Band Timer Control Register B (DBTCONB) — Address xx15h

15–12	11	10	9	8
Reserved	DBT3	DBT2	DBT1	DBT0
R-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1–0
EDBT3	EDBT2	EDBT1	DBTPS2	DBTPS1	DBTPS0	Reserved
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	R-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–12 **Reserved**. Reads return zero; writes have no effect.

Bits 11–8 **DBT3 (MSB)–DBT0 (LSB)**. Dead-band timer period. These bits define the period value of the three 4-bit dead-band timers.

Bit 7 **EDBT3**. Dead-band timer 3 enable (for pins PWM11 and PWM12 of Compare Unit 6).

0	Disable
1	Enable

Bit 6 **EDBT2**. Dead-band timer 2 enable (for pins PWM9 and PWM10 of Compare Unit 5).

0	Disable
1	Enable

Bit 5 **EDBT1**. Dead-band timer 1 enable (for pins PWM7 and PWM8 of Compare Unit 4).

0	Disable
1	Enable

Bits 4–2 **DBTPS2 to DBTPS0**. Dead-band timer prescaler.

000	x/1
001	x/2
010	x/4
011	x/8
100	x/16
101	x/32
110	x/32
111	x/32
	x = Device (CPU) clock frequency

Bits 1–0 **Reserved**. Reads return zero; writes have no effect.

Inputs and Outputs of Dead-Band Unit

The inputs to the dead-band unit are PH1, PH2, and PH3 from the asymmetric/symmetric waveform generators of compare units 1, 2, and 3, respectively.

The outputs of the dead-band unit are DTPH1, DTPH1_, DTPH2, DTPH2_, DTPH3, and DTPH3_, corresponding to PH1, PH2, and PH3, respectively.

Dead Band Generation

For each input signal PHx, two output signals, DTPHx and DTPHx_, are generated. When dead-band is not enabled for the compare unit and its associated outputs, the two signals are exactly the same. When the dead-band unit is enabled for the compare unit, the transition edges of the two signals are separated by a time interval called dead-band. This time interval is determined by the DBTCONx bits. If you assume the value in DBTCONx[11–8] is *m,* and the value in DBTCONx[4–2] corresponds to prescaler *x/p,* then the dead-band value is (*p***m*) device clock cycles.

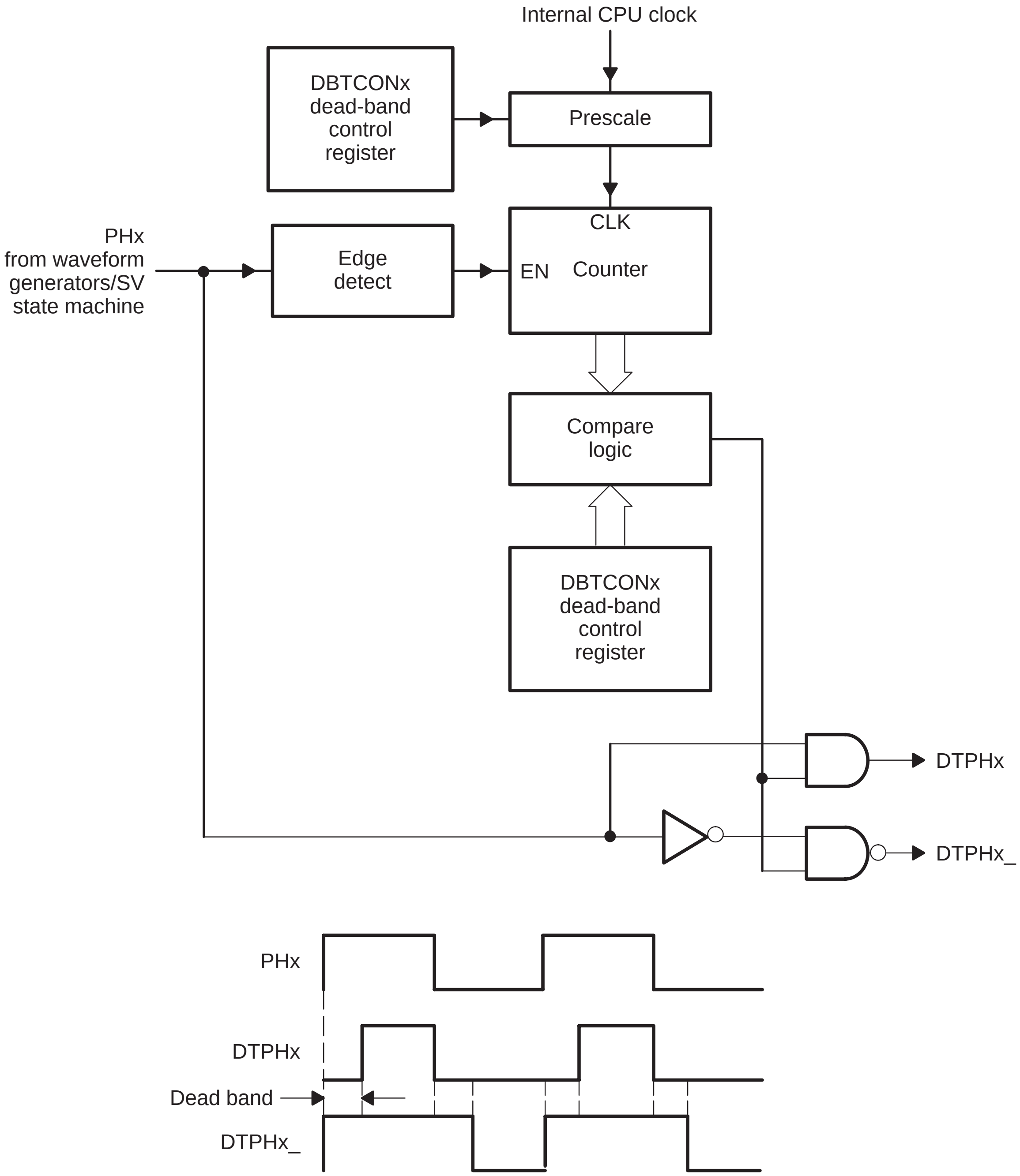
Table 6–13, on page 6-51, shows the dead-band generated by typical bit combinations in DBTCONx. The values are based on a 50 ns device clock. Figure 6–20, on page 6-52, shows the block diagram of the dead-band logic for one compare unit.

Table 6–13. Dead-Band Generation Examples

	DBTPS2–DBTPS0 (*p*) (DBTCONx[4–2])					
DBT3–DBT0 (*m*) (DBTCONx[11–8])	**110 and 1x1 (P=32)**	**100 (P=16)**	**011 (P=8)**	**010 (P=4)**	**001 (P=2)**	**000 (P=1)**
0	0	0	0	0	0	0
1	1.6	0.8	0.4	0.2	0.1	0.05
2	3.2	1.6	0.8	0.4	0.2	0.1
3	4.8	2.4	1.2	0.6	0.3	0.15
4	6.4	3.2	1.6	0.8	0.4	0.2
5	8	4	2	1	0.5	0.25
6	9.6	4.8	2.4	1.2	0.6	0.3
7	11.2	5.6	2.8	1.4	0.7	0.35
8	12.8	6.4	3.2	1.6	0.8	0.4
9	14.4	7.2	3.6	1.8	0.9	0.45
A	16	8	4	2	1	0.5
B	17.6	8.8	4.4	2.2	1.1	0.55
C	19.2	9.6	4.8	2.4	1.2	0.6
D	20.8	10.4	5.2	2.6	1.3	0.65
E	22.4	11.2	5.6	2.8	1.4	0.7
F	24	12	6	3	1.5	0.75

Note: Table values are given in µs.

Figure 6–20. Dead-Band Unit Block Diagram (x = 1, 2, or 3)


Internal CPU clock
DBTCONx dead-band control register
Prescale
CLK
PHx from waveform generators/SV state machine
Edge detect
EN
Counter
Compare logic
DBTCONx dead-band control register
DTPHx
DTPHx_
PHx
DTPHx
Dead band
DTPHx_

Other Important Features of Dead-Band Units

The dead-band unit is designed to prevent an overlap under any operating situation between the turn-on period of the upper and lower devices controlled by the two PWM outputs associated with each compare unit. This includes situations when the user has loaded a dead-band value greater than that of the duty cycle, and when the duty cycle is 100% or 0%. As a result, the PWM outputs associated with a compare unit do not reset to an inactive state at the end of a period when dead band is enabled for the compare unit.

6.5.3 Output Logic

The output logic circuit determines the polarity and/or the action that must be taken on a compare match for outputs PWMx, for x = 1–12. The outputs associated with each compare unit can be specified active low, active high, forced low, or forced high. The polarity and/or the action of the PWM outputs can be programmed by proper configuration of bits in the ACTR register. The PWM output pins can all be put in the high-impedance state by any of the following:

- Software clearing the COMCONx[9] and COMCONx[8] bits, respectively
- Hardware pulling $\overline{\text{PDPINTx}}$ low when $\overline{\text{PDPINTx}}$ is unmasked
- The occurrence of any reset event

Active $\overline{\text{PDPINTx}}$ (when enabled) and system reset override the bits in COMCONx and ACTRx

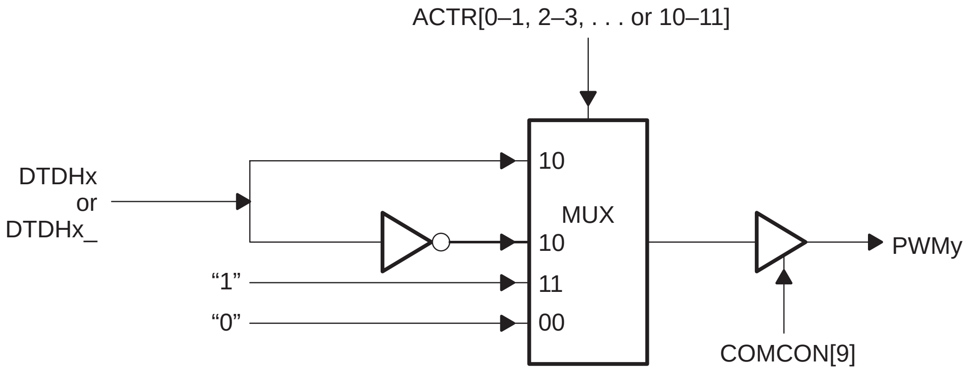
Figure 6–21, on page 6-54, shows a block diagram of the output logic circuit (OLC). The inputs of Output Logic for the compare units are:

- DTPH1, DTPH1_, DTPH2, DTPH2_, DTPH3, and DTPH3_ from the dead-band unit and compare match signals
- The control bits of ACTRx
- $\overline{\text{PDPINTx}}$ and RESET

The outputs of the Output Logic for the compare units are:

- PWMx, x = 1–6 (for EVA)
- PWMy, y = 7–12 (for EVB)

Figure 6–21. Output Logic Block Diagram (x = 1, 2, or 3; y = 1, 2, 3, 4, 5, or 6)



Output logic for PWM mode

6.6 PWM Waveform Generation With Compare Units and PWM Circuits

A pulse width modulated (PWM) signal is a sequence of pulses with changing pulse widths. The pulses are spread over a number of fixed-length periods so that there is one pulse in each period. The fixed period is called the PWM (carrier) period and its inverse is called the PWM (carrier) frequency. The widths of the PWM pulses are determined, or modulated, from pulse to pulse according to another sequence of desired values, the modulating signal.

In a motor control system, PWM signals are used to control the on and off time of switching power devices that deliver the desired current and energy to the motor windings (see Figure 6–24 on page 6-60). The shape and frequency of the phase currents and the amount of energy delivered to the motor windings control the required speed and torque of the motor. In this case, the command voltage or current to be applied to the motor is the modulating signal. The frequency of the modulating signal is typically much lower than the PWM carrier frequency.

PWM Signal Generation

To generate a PWM signal, an appropriate timer is needed to repeat a counting period that is the same as the PWM period. A compare register is used to hold the modulating values. The value of the compare register is constantly compared with the value of the timer counter. When the values match, a transition (from low to high, or high to low) happens on the associated output. When a second match is made between the values, or when the end of a timer period is reached, another transition (from high to low, or low to high) happens on the associated output. In this way, an output pulse is generated whose on (or off) duration is proportional to the value in the compare register. This process is repeated for each timer period with different (modulating) values in the compare register. As a result, a PWM signal is generated at the associated output.

Dead Band

In many motion/motor and power electronics applications, two power devices, an upper and a lower, are placed in series on one power converter leg. The turn-on periods of the two devices must not overlap with each other in order to avoid a shoot-through fault. Thus, a pair of non-overlapping PWM outputs is often required to properly turn on and off the two devices. A dead time (deadband) is often inserted between the turning-off of one transistor and the turning-on of the other transistor. This delay allows complete turning-off of one transistor before the turning-on of the other transistor. The required time delay is specified by the turning-on and turning-off characteristics of the power transistors and the load characteristics in a specific application.

6.6.1 Generation of PWM Outputs With Event Manager

Each of the three compare units, together with GP timer 1 (in case of EVA) or GP timer 3 (in case of EVB), the dead-band unit, and the output logic in the event manager module, can be used to generate a pair of PWM outputs with programmable dead-band and output polarity on two dedicated device pins. There are six such dedicated PWM output pins associated with the three compare units in each EV module. These six dedicated output pins can be used to conveniently control 3-phase AC induction or brushless DC motors. The flexibility of output behavior control by the compare action control register (ACTRx) also makes it easy to control switched reluctance and synchronous reluctance motors in a wide range of applications. The PWM circuits can also be used to conveniently control other types of motors such as DC brush and stepper motors in single or multi-axis control applications. Each GP timer compare unit, if desired, can also generate a PWM output based on its own timer.

Asymmetric and Symmetric PWM Generation

Both asymmetric and symmetric PWM waveforms can be generated by every compare unit on the EV module. In addition, the three compare units together can be used to generate 3-phase symmetric space vector PWM outputs. PWM generation with GP timer compare units has been described in the GP timer sections. Generation of PWM outputs with the compare units is discussed in this section.

6.6.2 Register Setup for PWM Generation

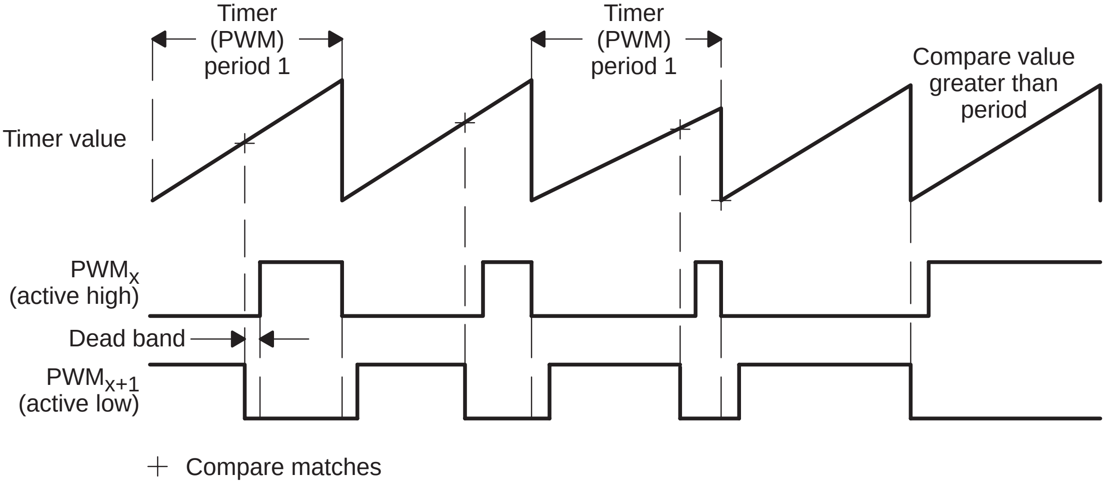
All three kinds of PWM waveform generations with compare units and associated circuits require configuration of the same Event Manager registers. The setup process for PWM generation includes the following steps:

- Setup and load ACTRx.
- Setup and load DBTCONx, if dead-band is to be used.
- Initialize CMPRx.
- Setup and load COMCONx.
- Setup and load T1CON (for EVA) or T3CON (for EVB) to start the operation.
- Rewrite CMPRx with newly determined values.

6.6.3 Asymmetric PWM Waveform Generation

The edge-triggered or asymmetric PWM signal is characterized by modulated pulses which are not centered with respect to the PWM period, as shown in Figure 6–22. The width of each pulse can only be changed from one side of the pulse.

Figure 6–22. Asymmetric PWM Waveform Generation With Compare Unit and PWM Circuits (x = 1, 3, or 5)



To generate an Asymmetric PWM signal, GP timer 1 is put in the continuous up-counting mode and its period register is loaded with a value corresponding to the desired PWM carrier period. The COMCONx is configured to enable the compare operation, set the selected output pins to be PWM outputs, and enable the outputs. If dead-band is enabled, the value corresponding to the required dead-band time should be written by software into the DBT(3:0) bits in DBTCONx(11:8). This is the period for the 4-bit dead-band timers. One dead-band value is used for all PWM output channels.

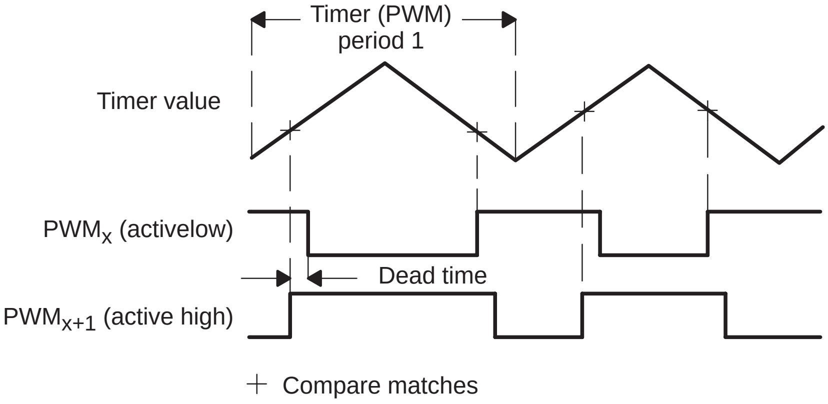
By proper configuration of ACTRx with software, a normal PWM signal can be generated on one output associated with a compare unit while the other is held low (or off) or high (or on), at the beginning, middle, or end of a PWM period. Such software controlled flexibility of PWM outputs is particularly useful in switched reluctance motor control applications.

After GP timer 1 (or GP timer 3) is started, the compare registers are rewritten every PWM period with newly determined compare values to adjust the width (the duty cycle) of PWM outputs that control the switch-on and off duration of the power devices. Since the compare registers are shadowed, a new value can be written to them at any time during a period. For the same reason, new values can be written to the action and period registers at any time during a period to change the PWM period or to force changes in PWM output definition.

6.6.4 Symmetric PWM Waveform Generation

A centered or symmetric PWM signal is characterized by modulated pulses which are centered with respect to each PWM period. The advantage of a symmetric PWM signal over an asymmetric PWM signal is that it has two inactive zones of the same duration: at the beginning and at the end of each PWM period. This symmetry has been shown to cause less harmonics than an asymmetric PWM signal in the phase currents of an AC motor such as induction and DC brushless motors when sinusoidal modulation is used. Figure 6–23 shows two examples of symmetric PWM waveforms.

Figure 6–23. Symmetric PWM Waveform Generation With Compare Units and PWM Circuits (x = 1, 3, or 5)



The generation of a symmetric PWM waveform with a compare unit is similar to the generation of an asymmetric PWM waveform. The only exception is that GP timer 1 (or GP timer 3) now needs to be put in continuous up-/down-counting mode.

There are usually two compare matches in a PWM period in symmetric PWM waveform generation, one during the upward counting before period match, and another during downward counting after period match. A new compare value becomes effective after the period match (reload on period) because it-makes it possible to advance or delay the second edge of a PWM pulse. An application of this feature is when a PWM waveform modification compensates for current errors caused by the dead-band in AC motor control.

Because the compare registers are shadowed, a new value can be written to them at any time during a period. For the same reason, new values can be written to the action and period registers at any time during a period to change the PWM period or to force changes in the PWM output definition.

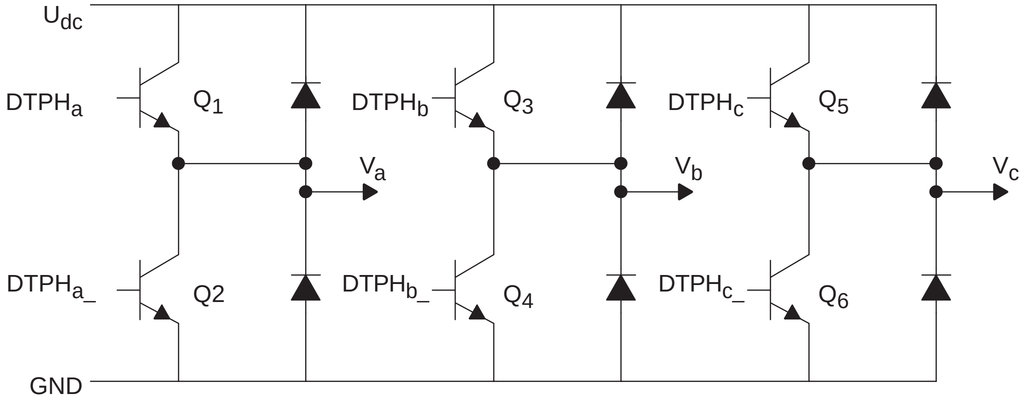
6.7 Space Vector PWM

Space vector PWM refers to a special switching scheme of the six power transistors of a 3-phase power converter. It generates minimum harmonic distortion to the currents in the windings of a 3-phase AC motor. It also provides more efficient use of supply voltage in comparison with the sinusoidal modulation method.

6.7.1 3-Phase Power Inverter

The structure of a typical 3-phase power inverter is shown in Figure 6–24, where V_a, V_b, and V_c are the voltages applied to the motor windings. The six power transistors are controlled by $DTPH_x$ and $DTPH_{x\_}$ (x = a, b, and c). When an upper transistor is switched on ($DTPH_x = 1$), the lower transistor is switched off ($DTPH_{x\_} = 0$). Thus, the on and off states of the upper transistors (Q1, Q3, and Q5) or, equivalently, the state of DTPHx (x = a, b, and c) are sufficient to evaluate the applied motor voltage U_{out}.

Figure 6–24. 3-Phase Power Inverter Schematic Diagram



Power Inverter Switching Patterns and the Basic Space Vectors

When an upper transistor of a leg is on, the voltage V_x (x = a, b, or c) applied by the leg to the corresponding motor winding is equal to the voltage supply U_{dc}. When it is off, the voltage applied is zero. The on and off switching of the upper transistors ($DTPH_x$, x = a, b, or c) have eight possible combinations. The eight combinations and the derived motor line-to-line and phase voltage in terms of DC supply voltage U_{dc} are shown in Table 6–14, on page 6-61, where a, b, and c represent the values of $DTPH_a$, $DTPH_b$, and $DTPH_c$, respectively.

Table 6–14. Switching Patterns of a 3-Phase Power Inverter

a	b	c	$V_{a0}(U_{dc})$	$V_{b0}(U_{dc})$	$V_{c0}(U_{dc})$	$V_{ab}(U_{dc})$	$V_{bc}(U_{dc})$	$V_{ca}(U_{dc})$
0	0	0	0	0	0	0	0	0
0	0	1	–1/3	–1/3	2/3	0	–1	1
0	1	0	–1/3	2/3	–1/3	–1	1	0
0	1	1	–2/3	1/3	1/3	–1	0	1
1	0	0	2/3	–1/3	–1/3	1	0	–1
1	0	1	1/3	–2/3	1/3	1	–1	0
1	1	0	1/3	1/3	–2/3	0	1	–1
1	1	1	0	0	0	0	0	0

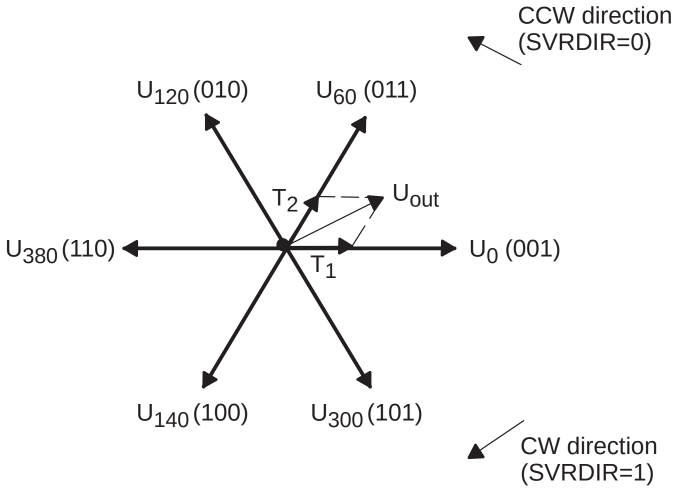
Note: 0 = off, 1 = on

Mapping the phase voltages corresponding to the eight combinations onto the d-q plane by performing a d-q transformation (which is equivalent to an orthogonal projection of the 3-vectors (a b c) onto the two dimensional plane perpendicular to the vector (1,1,1), the d-q plane), results in six nonzero vectors and two zero vectors. The nonzero vectors form the axes of a hexagonal. The angle between two adjacent vectors is 60 degrees. The two zero vectors are at the origin. These eight vectors are called the basic space vectors and are denoted by U_0, U_{60}, U_{120}, U_{180}, U_{240}, U_{300}, O_{000}, and O_{111}. The same transformation can be applied to the demanded voltage vector U_{out} to be applied to a motor. Figure 6–25 shows the projected vectors and the projected desired motor voltage vector U_{out}.

The d axis and q axis of a d-q plane correspond here to the horizontal and vertical geometrical axes of the stator of an AC machine.

The objective of the space vector PWM method is to approximate the motor voltage vector U_{out} by a combination of these eight switching patterns of the six power transistors.

Figure 6–25. Basic Space Vectors and Switching Patterns



The binary representations of two adjacent basic vectors are different in only one bit. That is, only one of the upper transistors switches when the switching pattern switches from U_x to U_{x+60} or from U_{x+60} to U_x. Also, the zero vectors O_{000} and O_{111} apply no voltage to the motor.

Approximation of Motor Voltage with Basic Space Vectors

The projected motor voltage vector U_{out}, at any given time, falls in one of the six sectors. Thus, for any PWM period, it can be approximated by the vector sum of two vector components lying on the two adjacent basic vectors:

$$U_{out} = T_1 U_x + T_2 U_{x+60} + T_0 (O_{000} \text{ or } O_{111})$$

where T_0 is given by $T_p–T_1–T_2$ and T_p is the PWM carrier period. The third term on the right side of the equation above doesn’t affect the vector sum U_{out}. The generation of U_{out} is beyond the scope of this context. For more details on space vector PWM and motor control theory, see *The Field Orientation Principle in Control of Induction Motors* by Andrzej M. Trzynadlowski.

The above approximation means that the upper transistors must have the on and off pattern corresponding to U_x and U_{x+60} for the time duration of T_1 and T_2, respectively, in order to apply voltage U_{out} to the motor. The inclusion of zero basic vectors helps to balance the turn on and off periods of the transistors, and thus their power dissipation.

6.7.2 Space Vector PWM Waveform Generation with Event Manager

The EV module has built-in hardware to greatly simplify the generation of symmetric space vector PWM waveforms. Software is used to generate space vector PWM outputs.

Software

To generate space vector PWM outputs, the user software must:

- ❑ Configure ACTRx to define the polarity of the compare output pins
- ❑ Configure COMCONx to enable compare operation and space vector PWM mode, and set the reload condition for CMPRx to be underflow
- ❑ Put GP timer 1 (or GP timer 3) in continuous up-/down-counting mode to start the operation

The user software then needs to determine the voltage U_{out} to be applied to the motor phases in the two dimensional d-q plane, decompose $U_{out,}$ and perform the following for each PWM period:

- ❑ Determine the two adjacent vectors, U_x and U_{x+60}
- ❑ Determine the parameters T_1, T_2, and T_0
- ❑ Write the switching pattern corresponding to U_x in ACTRx[14–12] and 1 in ACTRx[15], or the switching pattern of U_{x+60} in ACTRx[14–12] and 0 in ACTRx[15]
- ❑ Put (1/2 T1) in CMPR1 and (1/2 T1 + 1/2 T2) in CMPR2

Space Vector PWM Hardware

The space vector PWM hardware in the EV module does the following to complete a space vector PWM period:

- ❑ At the beginning of each period, sets the PWM outputs to the (new) pattern U_y defined by ACTRx[14–12]
- ❑ On the first compare match during up-counting between CMPR1 and GP timer 1 at (1/2 T1), switches the PWM outputs to the pattern of U_{y+60} if ACTRx[15] is 1, or to the pattern of U_y if ACTRx[15] is 0 ($U_{0-60} = U_{300}$, $U_{360+60} = U_{60}$)
- ❑ On the second compare match during up-counting between CMPR2 and GP timer 1 at (1/2 T1 + 1/2 T2), switches the PWM outputs to the pattern (000) or (111), whichever differs from the second pattern by one bit
- ❑ On the first compare match during down-counting between CMPR2 and GP timer 1 at (1/2 T1 + 1/2 T2), switches the PWM outputs back to the second output pattern
- ❑ On the second compare match during down-counting between CMPR1 and GP timer 1 at (1/2 T1), switches the PWM outputs back to the first pattern

Space Vector PWM Waveforms

The space vector PWM waveforms generated are symmetric with respect to the middle of each PWM period, and for this reason, it is called the symmetric space vector PWM generation method. Figure 6–26 shows examples of the symmetric space vector PWM waveforms.

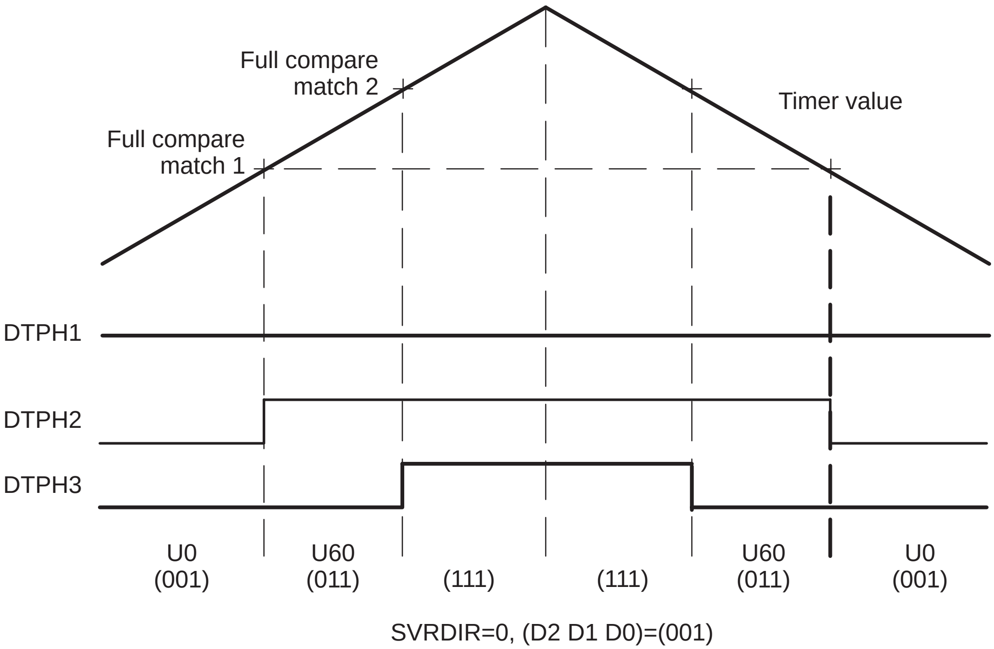
The Unused Compare Register

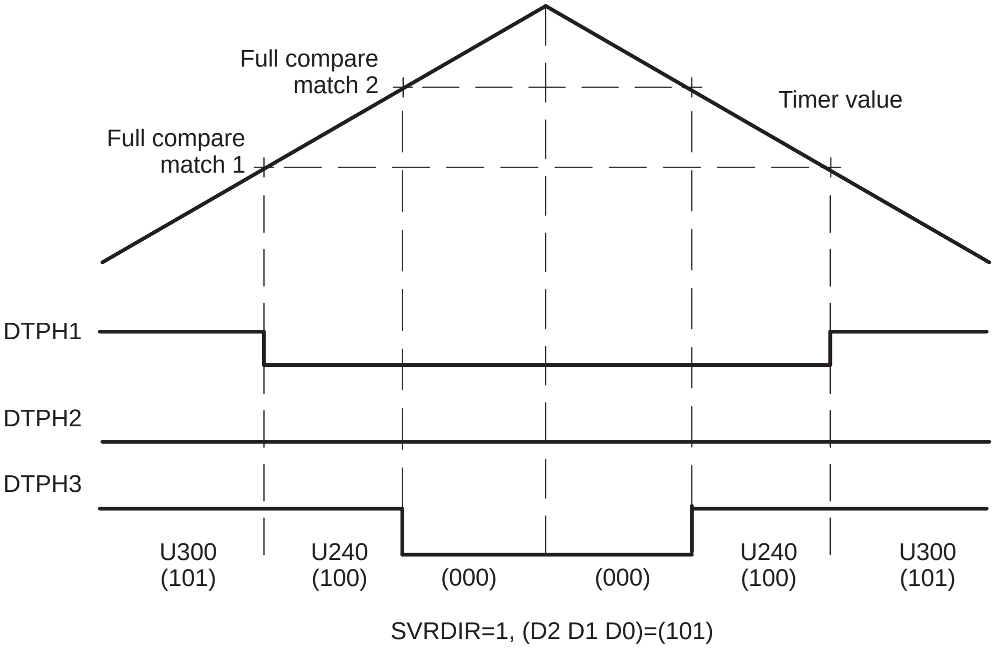
Only two compare registers are used in space vector PWM output generation. The third compare register, however, is still constantly compared with GP timer 1. When a compare match happens, the corresponding compare interrupt flag remains set and a peripheral interrupt request is generated, if the flag is unmasked. Therefore, the compare register that is not used in the space vector PWM outputs generation can still be used to time events happening in a specific application. Also, because of the extra delay introduced by the state machine, the compare output transitions are delayed by one clock cycle in space vector PWM mode.

6.7.3 Space Vector PWM Boundary Conditions

All three compare outputs become inactive when both compare registers (CMPR1 and CMPR2) are loaded with a zero value in space vector PWM mode. In general, it is the user’s responsibility to assure that (CMPR1) ≤ (CMPR2) ≤ (T1PR) in the space vector PWM mode. Otherwise, unpredictable behavior may result.

Figure 6–26. Symmetric Space Vector PWM Waveforms





6.8 Capture Units

Capture units enable logging of transitions on capture input pins. There are six capture units, three is each EV module. Capture Units 1, 2, and 3 are associated with EVA and Capture Units 4, 5, and 6 are associated with EVB. Each capture unit is associated with a capture input pin.

Each EVA capture unit can choose GP timer 2 or 1 as its time base; however, CAP1 and CAP2 cannot choose a different timer between themselves as their timebase. Each EVB capture unit can choose GP timer 4 or 3 as its time base; however, CAP4 and CAP5 cannot choose a different timer between themselves as their timebase.

The value of the GP timer is captured and stored in the corresponding 2-level-deep FIFO stack when a specified transition is detected on a capture input pin (CAPx). Figure 6–27 shows a block diagram of an EVA capture unit and Figure 6–28 shows a block diagram of an EVB capture unit.

Figure 6–27. Capture Units Block Diagram (EVA)

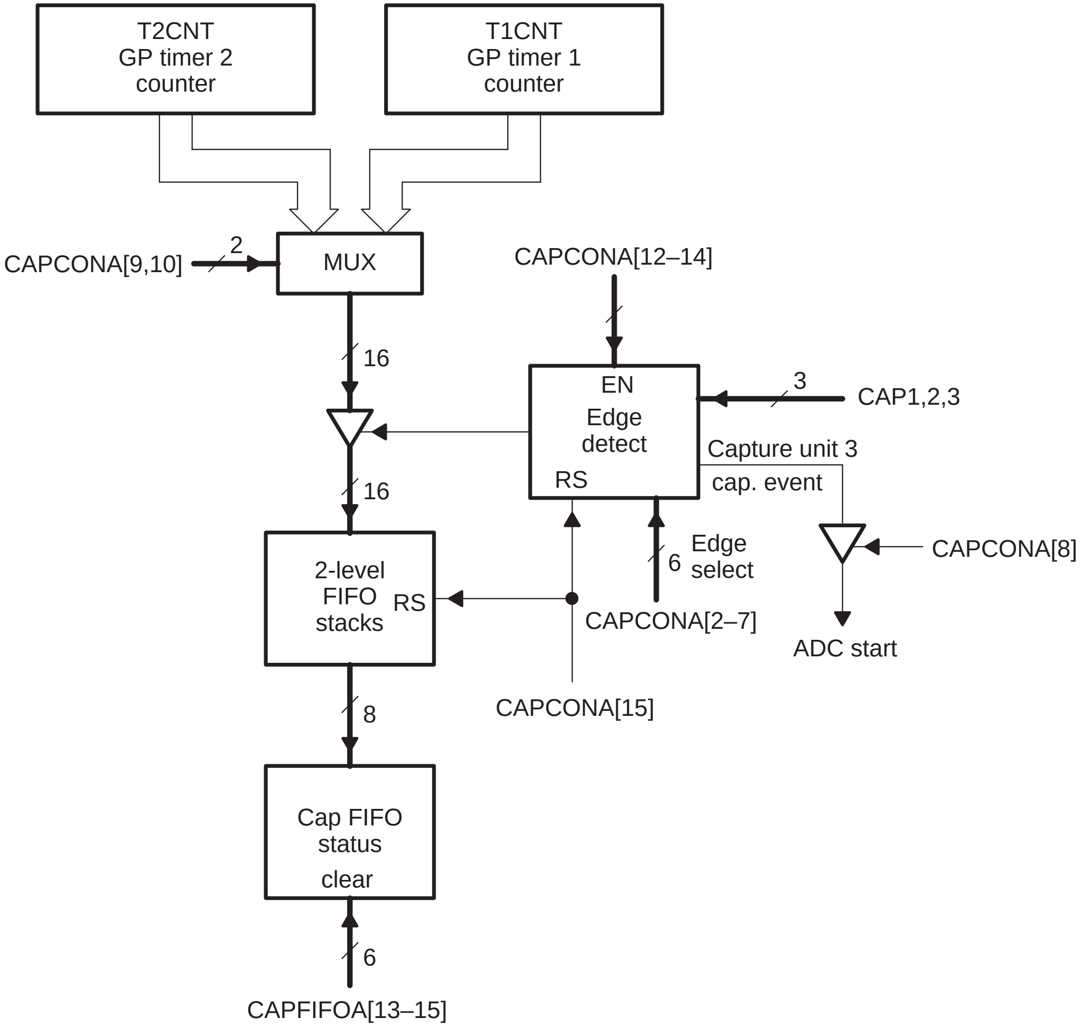
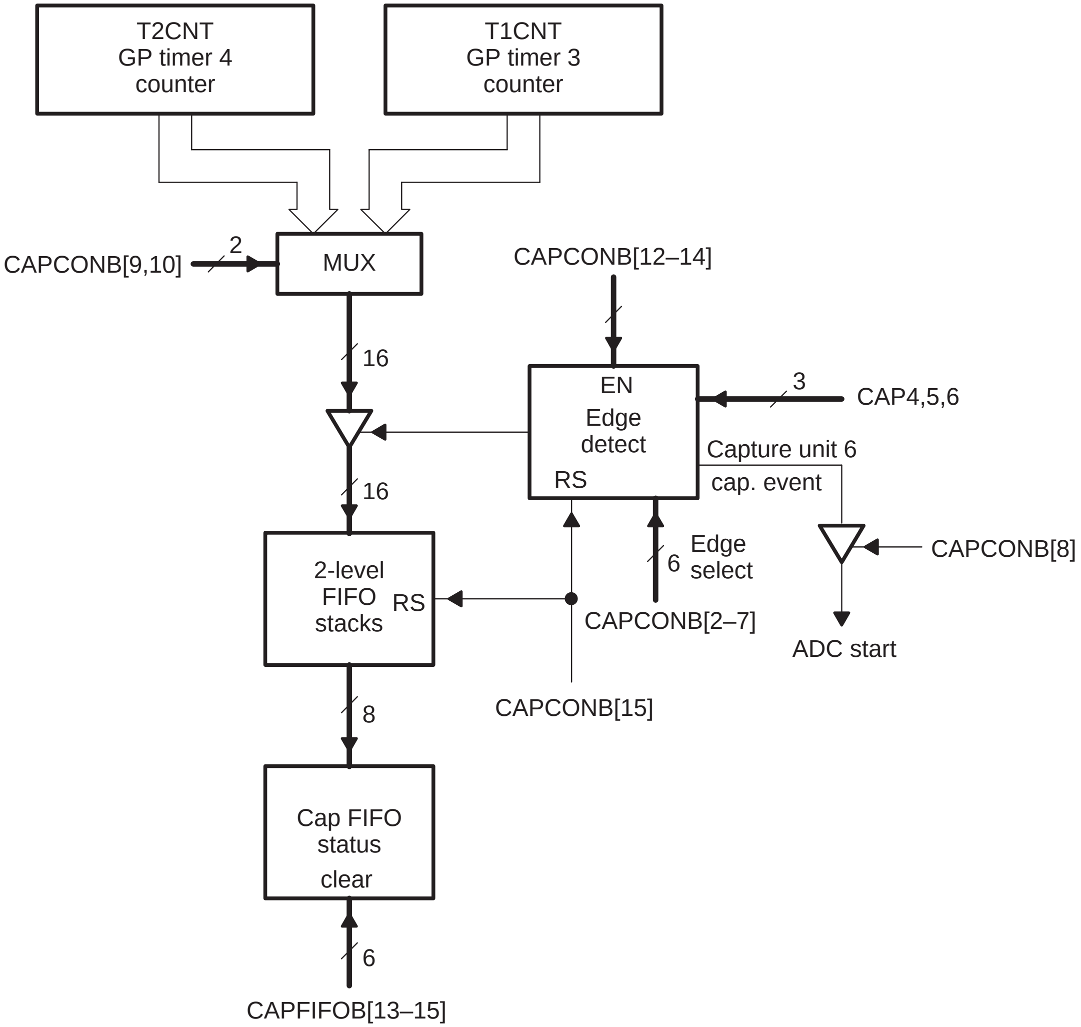


Figure 6–28. Capture Units Block Diagram (EVB)



6.8.1 Capture Unit Features

Capture units have the following features:

- ❑ One 16-bit capture control register (CAPCONA for EVA, CAPCONB for EVB), (RW)
- ❑ One 16-bit capture FIFO status register (CAPFIFOA for EVA, CAPFIFOB for EVB)
- ❑ Selection of GP timer 1 or 2 (for EVA) and GP timer 3 or 4 (for EVB) as the time base

- ❑ Three 16-bit 2-level-deep FIFO stacks, one for each capture unit.
- ❑ Six Schmitt-triggered capture input pins, CAP1 through CAP6, one input pin for each capture unit. (All inputs are synchronized with the device/CPU clock: in order for a transition to be captured, the input must hold at its current level to meet the two rising edges of the device clock. Input pins CAP1 and CAP2 (CAP4 and CAP5 in case of EVB) can also be used as QEP inputs to QEP circuit)
- ❑ User-specified transition (rising edge, falling edge, or both edges) detection
- ❑ Six maskable interrupt flags, one for each capture unit

6.8.2 Operation of Capture Units

After a capture unit is enabled, a specified transition on the associated input pin causes the counter value of the selected GP timer to be loaded into the corresponding FIFO stack. At the same time, if there are already one or more valid capture values stored in the FIFO stack (CAPxFIFO bits not equal to zero) the corresponding interrupt flag is set. If the flag is unmasked, a peripheral interrupt request is generated. The corresponding status bits in CAPFIFOx are adjusted to reflect the new status of the FIFO stack each time a new counter value is captured in a FIFO stack. The latency from the time a transition happens in a capture input to the time the counter value of selected GP timer is locked is two clock cycles.

All capture unit registers are cleared to 0 by a RESET condition.

Capture Unit Time Base Selection

For EVA, Capture Unit 3 has a separate time base selection bit from Capture Units 1 and 2. This allows the two GP timers to be used at the same time, one for Capture Units 1 and 2, and the other for Capture Unit 3. For EVB, Capture Unit 6 has a separate time base selection bit.

Capture operation does not affect the operation of any GP timer or the compare/PWM operations associated with any GP timer.

Capture Unit Setup

For a capture unit to function properly, the following register setup must be performed:

1) Initialize the CAPFIFOx and clear the appropriate status bits.

2) Set the selected GP timer in one of its operating modes.

3) Set the associated GP timer compare register or GP timer period register, if necessary.

4) Set up CAPCONA or CAPCONB as appropriate.

6.8.3 Capture Unit Registers

The operation of the capture units is controlled by four 16-bit control registers, CAPCONA/B and CAPFIFOA/B. TxCON (x = 1, 2, 3, or 4) registers are also used to control the operation of the capture units since the time base for capture circuits can be provided by any of these timers. Additionally, CAPCONA/B is also used to control the operation of the QEP circuit. Table 6–7 and Table 6–8 on page 6-12 shows the addresses of these registers.

Capture Control Register A (CAPCONA)

Figure 6–29. Capture Control Register A (CAPCONA) — Address 7420h

15	14–13	12	11	10	9	8
CAPRES	CAPQEPN	CAP3EN	Reserved	CAP3TSEL	CAP12TSEL	CAP3TOADC
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7–6	5–4	3–2	1–0
CAP1EDGE	CAP2EDGE	CAP3EDGE	Reserved
RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 15 **CAPRES**. Capture reset. Always reads zero.

Note: This bit is not implemented as a register bit. Writing a 0 simply clears the capture registers.

0 Clear all registers of capture units and QEP circuit to 0.

1 No action

Bits 14–13 **CAPQEPN**. Capture Units 1 and 2 and QEP circuit control.

00 Disable Capture Units 1 and 2 and QEP circuit. FIFO stacks retain their contents.

01 Enable Capture Units 1 and 2, disable QEP circuit.

10 Reserved

11 Enable QEP circuit. Disable Capture Units 1 and 2; bits 4–7 and 9 are ignored.

Bit 12 **CAP3EN**. Capture Unit 3 control.

0 Disable Capture Unit 3; FIFO stack of Capture Unit 3 retains its contents.

1 Enable Capture Unit 3

Bit 11 **Reserved**. Reads return zero; writes have no effect.

Bit 10 **CAP3TSEL**. GP timer selection for Capture Unit 3.

0 Select GP timer 2

1 Select GP timer 1

Bit 9 **CAP12TSEL**. GP timer selection for Capture Units 1 and 2.

0 Select GP timer 2

1 Select GP timer 1

Bit 8 **CAP3TOADC**. Capture Unit 3 event starts ADC.

0 No action

1 Start ADC when the CAP3INT flag is set

Bits 7–6 **CAP1EDGE**. Edge detection control for Capture Unit 1.

00 No detection

01 Detect rising edge

10 Detect falling edge

11 Detect both edges

Bits 5–4 **CAP2EDGE**. Edge detection control for Capture Unit 2.

00 No detection

01 Detect rising edge

10 Detect falling edge

11 Detect both edges

Bits 3–2 **CAP3EDGE**. Edge detection control for Capture Unit 3.

00 No detection

01 Detect rising edge

10 Detect falling edge

11 Detect both edges

Bits 1–0 **Reserved**. Reads return zero; writes have no effect.

Capture Control Register B (CAPCONB)

Figure 6–30. Capture Control Register B (CAPCONB) — Address 7520h

15	14–13	12	11	10	9	8
CAPRES	CAPQEPN	CAP6EN	Reserved	CAP6TSEL	CAP45TSEL	CAP6TOADC
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7–6	5–4	3–2	1–0
CAP4EDGE	CAP5EDGE	CAP6EDGE	Reserved
RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 15 **CAPRES**. Capture reset. Always reads zero.

Note: This bit is not implemented as a register bit. Writing a 0 simply clears the capture registers.

0 Clear all registers of capture units and QEP circuit to 0.

1 No action

Bits 14–13 **CAPQEPN**. Capture Units 4 and 5 and QEP circuit control.

00 Disable Capture Units 4 and 5 and QEP circuit. FIFO stacks retain their contents.

01 Enable Capture Units 4 and 5, disable QEP circuit.

10 Reserved

11 Enable QEP circuit. Disable Capture Units 4 and 5; bits 4–7 and 9 are ignored.

Bit 12 **CAP6EN**. Capture Unit 6 control.

0 Disable Capture Unit 6; FIFO stack of Capture Unit 6 retains its contents.

1 Enable Capture Unit 6

Bit 11 **Reserved**. Reads return zero; writes have no effect.

Bit 10 **CAP6TSEL**. GP timer selection for Capture Unit 6.

0 Select GP timer 5

1 Select GP timer 4

Bit 9 **CAP45TSEL**. GP timer selection for Capture Units 4 and 5.

0 Select GP timer 5

1 Select GP timer 4

Bit 8 **CAP6TOADC**. Capture Unit 6 event starts ADC.

0 No action

1 Start ADC when the CAP6INT flag is set

Bits 7–6 **CAP4EDGE**. Edge detection control for Capture Unit 4.

00 No detection

01 Detect rising edge

10 Detect falling edge

11 Detect both edges

Bits 5–4 **CAP5EDGE**. Edge detection control for Capture Unit 5.

00 No detection

01 Detect rising edge

10 Detect falling edge

11 Detect both edges

Bits 3–2 **CAP6EDGE**. Edge detection control for Capture Unit 6.

00 No detection

01 Detect rising edge

10 Detect falling edge

11 Detect both edges

Bits 1–0 **Reserved**. Reads return zero; writes have no effect.

Capture FIFO Status Register A (CAPFIFOA)

CAPFIFOA contains the status bits for each of the three FIFO stacks of the capture units. The bit description of CAPFIFOA is given in Figure 6–31. If a write occurs to the CAPnFIFOA status bits at the same time as they are being updated (because of a capture event), the write data takes precedence.

Figure 6–31. Capture FIFO Status Register A (CAPFIFOA) — Address 7422h

15–14	13–12	11–10	9–8
Reserved	CAP3FIFO	CAP2FIFO	CAP1FIFO
R-0	RW-0	RW-0	RW-0

7–0
Reserved
R-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–14 **Reserved**. Reads return zero; writes have no effect.

Bits 13–12 **CAP3FIFO**. CAP3FIFO Status.

00	Empty
01	Has one entry
10	Has two entries
11	Had two entries and captured another one; first entry has been lost.

Bits 11–10 **CAP2FIFO**. CAP2FIFO Status

00	Empty
01	Has one entry
10	Has two entries
11	Had two entries and captured another one; first entry has been lost.

Bits 9–8 **CAP1FIFO**. CAP1FIFO Status.

00	Empty
01	Has one entry
10	Has two entries
11	Had two entries and captured another one; first entry has been lost.

Bits 7–0 **Reserved**. Reads return zero; writes have no effect.

Capture FIFO Status Register B (CAPFIFOB)

CAPFIFOB contains the status bits for each of the three FIFO stacks of the capture units. The bit description of CAPFIFOB is given in Figure 6–32. If a write occurs to the CAPnFIFOB status bits at the same time as they are being updated (because of a capture event), the write data takes precedence.

Figure 6–32. Capture FIFO Status Register B (CAPFIFOB) — Address 7522h

15–14	13–12	11–10	9–8
Reserved	CAP6FIFO	CAP5FIFO	CAP4FIFO
R-0	RW-0	RW-0	RW-0

7–0
Reserved
R-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–14 **Reserved**. Reads return zero; writes have no effect.

Bits 13–12 **CAP6FIFO**. CAP6FIFO Status.

00	Empty
01	Has one entry
10	Has two entries
11	Had two entries and captured another one; first entry has been lost.

Bits 11–10 **CAP5FIFO**. CAP5FIFO Status

00	Empty
01	Has one entry
10	Has two entries
11	Had two entries and captured another one; first entry has been lost.

Bits 9–8 **CAP4FIFO**. CAP4FIFO Status.

00	Empty
01	Has one entry
10	Has two entries
11	Had two entries and captured another one; first entry has been lost.

Bits 7–0 **Reserved**. Reads return zero; writes have no effect.

6.8.4 Capture Unit FIFO Stacks

Each capture unit has a dedicated 2-level-deep FIFO stack. The top stack consists of CAP1FIFO, CAP2FIFO, and CAP3FIFO (in case of EVA) or CAP4FIFO, CAP5FIFO, and CAP6FIFO (in case of EVB). The bottom stack consists of CAP1FBOT, CAP2FBOT, and CAP3FBOT (in case of EVA) or

CAP4FBOT, CAP5FBOT, and CAP6FBOT (in case of EVB). The top-level register of any of the FIFO stacks is a read-only register that always contains the oldest counter value captured by the corresponding capture unit. Therefore, a read access to the FIFO stack of a capture unit always returns the oldest counter value stored in the stack. When the oldest counter value in the top register of the FIFO stack is read, the newer counter value in the bottom register of the stack, if any, is pushed into the top register.

If desired, the bottom register of the FIFO stack can be read. Reading the bottom register of the FIFO stack causes the FIFO status bits to change to 01 (has one entry), if they were previously 10 or 11. If the FIFO status bits were previously 01 when the bottom FIFO register is read, they will change to 00 (empty).

First Capture

The counter value of the selected GP timer (captured by a capture unit when a specified transition happens on its input pin) is written into the top register of the FIFO stack, if the stack is empty. At the same time, the corresponding status bits are set to 01. The status bits are reset to 00 if a read access is made to the FIFO stack before another capture is made.

Second Capture

If another capture occurs before the previously captured counter value is read, the newly captured counter value goes to the bottom register. In the mean time, the corresponding status bits are set to (10). When the FIFO stack is read before another capture happens, the older counter value in the top register is read out, the newer counter value in the bottom register is pushed up into the top register, and the corresponding status bits are set to 01.

The appropriate capture interrupt flag is set by the second capture. A peripheral interrupt request is generated if the interrupt is not masked.

Third Capture

If a capture happens when there are already two counter values captured in the FIFO stack, the oldest counter value in the top register of the stack is pushed out and lost, the counter value in the bottom register of the stack is pushed up into the top register, the newly captured counter value is written into the bottom register, and the status bits are set to 11 to indicate one or more older captured counter values have been lost.

The appropriate capture interrupt flag is also set by the third capture. A peripheral interrupt request is generated if the interrupt is not masked.

6.8.5 Capture Interrupt

When a capture is made by a capture unit and there is already at least one valid value in the FIFO (indicated by CAPxFIFO bits not equal to zero), the corresponding interrupt flag is set, and if unmasked, a peripheral interrupt request is generated. Thus, a pair of captured counter values can be read by an interrupt service routine if the interrupt is used. If an interrupt is not desired, either the interrupt flag or the status bits can be polled to determine if two captures have occurred allowing the captured counter values to be read.

6.9 Quadrature Encoder Pulse (QEP) Circuit

Each Event Manager module has a quadrature encoder pulse (QEP) circuit. The QEP circuit, when enabled, decodes and counts the quadrature encoded input pulses on pins CAP1/QEP1 and CAP2/QEP2 (in case of EVA) or CAP4/QEP3 and CAP5/QEP4 (in case of EVB). The QEP circuit can be used to interface with an optical encoder to get position and speed information from a rotating machine. When the QEP circuit is enabled, the capture function on CAP1/CAP2 and CAP4/CAP5 pins is disabled.

6.9.1 QEP Pins

The two QEP input pins are shared between capture units 1 and 2 (or 3 and 4, for EVB), and the QEP circuit. Proper configuration of CAPCONx bits is required to enable the QEP circuit and disable capture units, thus assigning the associated input pins for use by the QEP circuit.

6.9.2 QEP Circuit Time Base

The time base for the QEP circuit is provided by GP timer 2 (GP timer 4, in case of EVB). The GP timer must be put in directional-up/down count mode with the QEP circuit as the clock source. Figure 6–33 shows the block diagram of the QEP circuit for EVA and Figure 6–34 shows the block diagram of the QEP circuit for EVB.

Figure 6–33. Quadrature Encoder Pulse (QEP) Circuit Block Diagram for EVA

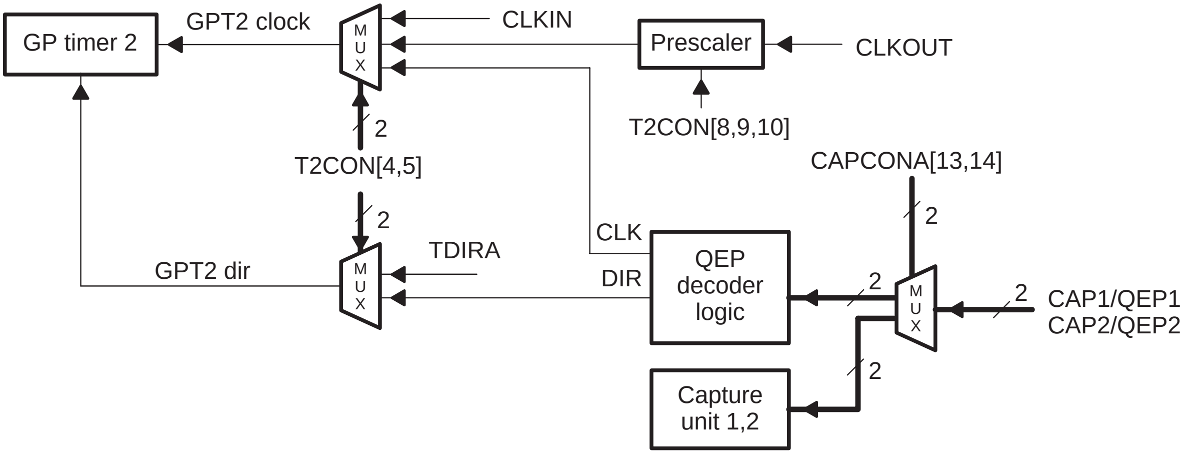
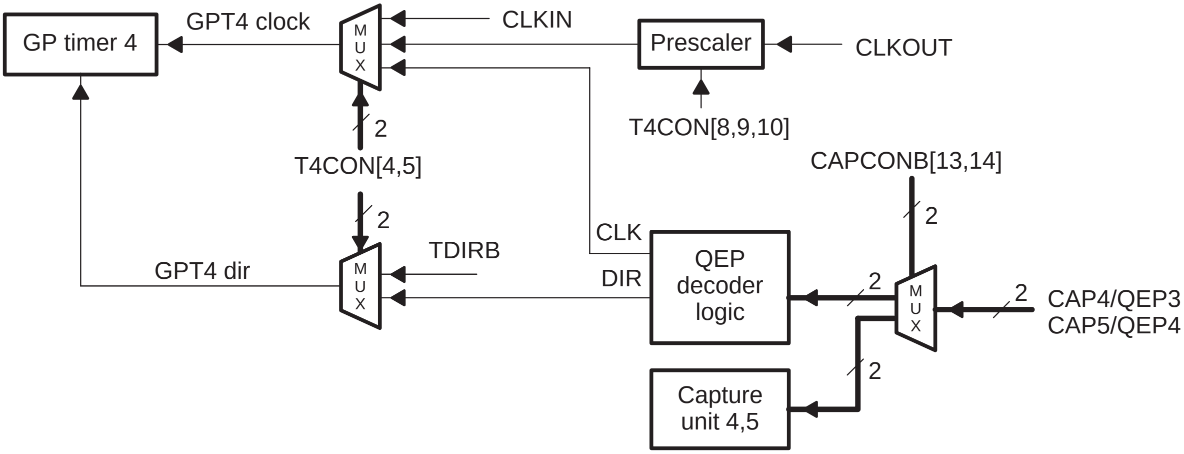


Figure 6–34. Quadrature Encoder Pulse (QEP) Circuit Block Diagram for EVB



6.9.3 Decoding

Quadrature encoded pulses are two sequences of pulses with a variable frequency and a fixed phase shift of a quarter of a period (90 degrees). When generated by an optical encoder on a motor shaft, the direction of rotation of the motor can be determined by detecting which of the two sequences is the leading sequence. The angular position and speed can be determined by the pulse count and pulse frequency.

QEP Circuit

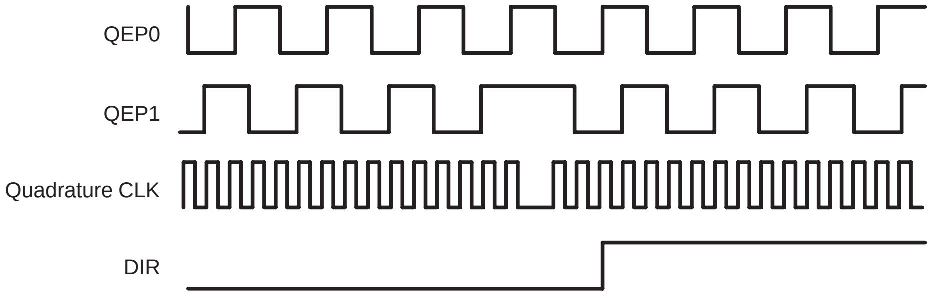
The direction detection logic of the QEP circuit in the EV module determines which one of the sequences is the leading sequence. It then generates a direction signal as the direction input to GP timer 2 (or 4). The timer counts up if CAP1/QEP1 (CAP4/QEP3 for EVB) input is the leading sequence, and counts down if CAP2/QEP2 (CAP5/QEP4 for EVB) is the leading sequence.

Both edges of the pulses of the two quadrature encoded inputs are counted by the QEP circuit. Therefore, the frequency of the clock generated by the QEP logic to GP timer 2 (or 4) is four times that of each input sequence. This quadratureclock is connected to the clock input of GP timer 2 (or 4).

Quadrature Encoded Pulse Decoding Example

Figure 6–35 shows an example of quadrature encoded pulses and the derived clock and counting direction.

Figure 6–35. Quadrature Encoded Pulses and Decoded Timer Clock and Direction



6.9.4 QEP Counting

GP timer 2 (or 4) always starts counting from its current value. A desired value can be loaded to the GP timer's counter prior to enabling the QEP mode. When the QEP circuit is selected as the clock source, the timer ignores the TDIRA/B and TCLKINA/B input pins.

GP Timer Interrupt and Associated Compare Outputs in QEP Operation

Period, underflow, overflow, and compare interrupt flags for a GP timer with a QEP circuit clock are generated on respective matches. A peripheral interrupt request can be generated by an interrupt flag, if the interrupt is unmasked.

6.9.5 Register Setup for the QEP Circuit

To start the operation of the QEP circuit in EVA:

1) Load GP timer 2's counter, period, and compare registers with desired values, if necessary.

2) Configure T2CON to set GP timer 2 in directional-up/down mode with the QEP circuits as clock source, and enable the selected timer.

3) Configure CAPCONA to enable the QEP circuit.

To start the operation of the QEP circuit in EVB:

1) Load GP timer 4’s counter, period, and compare registers with desired values, if necessary.

2) Configure T4CON to set GP timer 4 in directional-up/down mode with the QEP circuits as clock source, and enable the selected timer.

3) Configure CAPCONB to enable the QEP circuit.

6.10 Event Manager (EV) Interrupts

EV interrupt events are organized into 3 groups: A, B, and C. Each group is associated with a different interrupt flag and interrupt enable register. There are several event manager peripheral interrupt requests in each EV interrupt group. Table 6–16 shows all EVA interrupts, their priority, and grouping; and Table 6–17 shows all EVB interrupts, their priority, and grouping. There is an interrupt flag register and a corresponding interrupt mask register for each EV interrupt group, as shown in Table 6–15. A flag in EVAIFRx (x = A, B, or C) is masked (will not generate a peripheral interrupt request) if the corresponding bit in EVAIMRx is 0.

Table 6–15. Interrupt Flag Register and Corresponding Interrupt Mask Register

Flag Register	Mask Register	EV Module
EVAIFRA	EVAIMRA	EVA
EVAIFRB	EVAIMRB	
EVAIFRC	EVAIMRC	
EVBIFRA	EVBIMRA	EVB
EVBIFRB	EVBIMRB	
EVBIFRC	EVBIMRC	

6.10.1 EV Interrupt Request and Service

When a peripheral interrupt request is acknowledged, the appropriate peripheral interrupt vector is loaded into the peripheral interrupt vector register (PIVR) by the PIE controller. The vector loaded into the PIVR is the vector for the highest priority pending enabled event. The vector register can be read by the interrupt service routine (ISR).

Table 6–16. Event Manager A (EVA) Interrupts

Group	Interrupt	Priority within group	Vector (ID)	Description/Source	INT
	PDPINTA	1 (highest)	0020h	Power Drive Protection Interrupt A	1
A	CMP1INT	2	0021h	Compare Unit 1 compare interrupt	2
	CMP2INT	3	0022h	Compare Unit 2 compare interrupt	
	CMP3INT	4	0023h	Compare Unit 3 compare interrupt	
	T1PINT	5	0027h	GP timer 1 period interrupt	
	T1CINT	6	0028h	GP timer 1 compare interrupt	
	T1UFINT	7	0029h	GP timer 1 underflow interrupt	
	T1OFINT	8 (lowest)	002Ah	GP timer 1 overflow interrupt	
B	T2PINT	1 (highest)	002Bh	GP timer 2 period interrupt	3
	T2CINT	2	002Ch	GP timer 2 compare interrupt	
	T2UFINT	3	002Dh	GP timer 2 underflow interrupt	
	T2OFINT	4	002Eh	GP timer 2 overflow interrupt	
C	CAP1INT	1 (highest)	0033h	Capture Unit 1 interrupt	4
	CAP2INT	2	0034h	Capture Unit 2 interrupt	
	CAP3INT	3	0035h	Capture Unit 3 interrupt	

Table 6–17. Event Manager B (EVB) Interrupts

Group	Interrupt	Priority within group	Vector (ID)	Description/Source	INT
	PDPINTB	1 (highest)	0019h	Power Drive Protection Interrupt B	1
A	CMP4INT	2	0024h	Compare Unit 4 compare interrupt	2
	CMP5INT	3	0025h	Compare Unit 5 compare interrupt	
	CMP6INT	4	0026h	Compare Unit 6 compare interrupt	
	T3PINT	5	002Fh	GP timer 3 period interrupt	
	T3CINT	6	0030h	GP timer 3 compare interrupt	
	T3UFINT	7	0031h	GP timer 3 underflow interrupt	
	T3OFINT	8 (lowest)	0032h	GP timer 3 overflow interrupt	
B	T4PINT	1 (highest)	0039h	GP timer 4 period interrupt	3
	T4CINT	2	003Ah	GP timer 4 compare interrupt	
	T4UFINT	3	003Bh	GP timer 4 underflow interrupt	
	T4OFINT	4	003Ch	GP timer 4 overflow interrupt	
C	CAP4INT	1 (highest)	0036h	Capture Unit 4 interrupt	4
	CAP5INT	2	0037h	Capture Unit 5 interrupt	
	CAP6INT	3	0038h	Capture Unit 6 interrupt	

Table 6–18. Conditions for Interrupt Generation

Interrupt	Condition For Generation
Underflow	When the counter reaches 0000h
Overflow	When the counter reaches FFFFh
Compare	When the counter register contents match that of the compare register
Period	When the counter register contents match that of the period register

Interrupt Generation

When an interrupt event occurs in the EV module, the corresponding interrupt flag in one of the EV interrupt flag registers is set to 1. A peripheral interrupt request is generated to the Peripheral Interrupt Expansion controller, if the flag is locally unmasked (the corresponding bit in EVAIMRx is set to 1).

Interrupt Vector

The peripheral interrupt vector corresponding to the interrupt flag that has the highest priority among the flags that are set and enabled is loaded into the PIVR when an interrupt request is acknowledged (this is all done in the peripheral interrupt controller, external to the event manager peripheral).

> CAUTION
>
> **Failure to Clear the Interrupt Flag Bit**
>
> **The interrupt flag bit in the peripheral register must be cleared by software writing a 1 to the bit in the ISR. Failure to clear this bit will prevent future interrupt requests by that source.**

6.10.2 EV Interrupt Flag Registers

Addresses of the EVA interrupt registers and the EVB interrupt registers are shown in Table 6–9 and Table 6–10, respectively, on page 6-13. The registers are all treated as 16-bit memory mapped registers. The unused bits all return zero when read by software. Writing to unused bits has no effect. Since EVxIFRx are readable registers, occurrence of an interrupt event can be monitored by software polling the appropriate bit in EVxIFRx when the interrupt is masked.

EVA Interrupt Flag Register A (EVAIFRA)

Figure 6–36. EVA Interrupt Flag Register A (EVAIFRA) — Address 742Fh

15–11	10	9	8
Reserved	T1OFINT FLAG	T1UFINT FLAG	T1CINT FLAG
R-0	RW-0	RW-0	RW-0

7	6–4	3	2	1	0
T1PINT FLAG	Reserved	CMP3INT FLAG	CMP2INT FLAG	CMP1INT FLAG	PDPINTA FLAG
RW-0	R-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–11 **Reserved**. Reads return zero; writes have no effect.

Bit 10 **T1OFINT FLAG**. GP timer 1 overflow interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 9 **T1UFINT FLAG**. GP timer 1 underflow interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 8 **T1CINT FLAG**. GP timer 1 compare interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 7 **T1PINT FLAG.** GP timer 1 period interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bits 6–4 **Reserved**. Reads return zero; writes have no effect.

Bit 3 **CMP3INT FLAG**. Compare 3 interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 2 **CMP2INT FLAG**. Compare 2 interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 1 **CMP1INT FLAG**. Compare 1 interrupt.

Read: 0 Flag is reset
1 Flag is set
Write: 0 No effect
1 Reset flag

Bit 0 **PDPINTA FLAG**. Power drive protection interrupt.

Read: 0 Flag is reset
1 Flag is set
Write: 0 No effect
1 Reset flag

EVA Interrupt Flag Register B (EVAIFRB)

Figure 6–37. EVA Interrupt Flag Register B (EVAIFRB) — Address 7430h

15–4	3	2	1	0
Reserved	T2OFINT FLAG	T2UFINT FLAG	T2CINT FLAG	T2PINT FLAG
R-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–4 **Reserved**. Reads return zero; writes have no effect.

Bit 3 **T2OFINT FLAG**. GP timer 2 overflow interrupt.

Read: 0 Flag is reset
1 Flag is set
Write: 0 No effect
1 Reset flag

Bit 2 **T2UFINT FLAG**. GP timer 2 underflow interrupt.

Read: 0 Flag is reset
1 Flag is set
Write: 0 No effect
1 Reset flag

Bit 1 **T2CINT FLAG**. GP timer 2 compare interrupt.

Read: 0 Flag is reset
1 Flag is set
Write: 0 No effect
1 Reset flag

Bit 0 **T2PINT FLAG**. GP timer 2 period interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

EVA Interrupt Flag Register C (EVAIFRC)

Figure 6–38. EVA Interrupt Flag Register C (EVAIFRC) — Address 7431h

15–3	2	1	0
Reserved	CAP3INT FLAG	CAP2INT FLAG	CAP1INT FLAG
R-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–3 **Reserved**. Reads return zero; writes have no effect.

Bit 2 **CAP3INT FLAG**. Capture 3 interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 1 **CAP2INT FLAG**. Capture 2 interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 0 **CAP1INT FLAG**. Capture 1 interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

EVA Interrupt Mask Register A (EVAIMRA)

Figure 6–39. EVA Interrupt Mask Register A (EVAIMRA) — Address 742Ch

15–11	10	9	8
Reserved	T1OFINT ENABLE	T1UFINT ENABLE	T1CINT ENABLE
R-0	RW-0	RW-0	RW-0

7	6–4	3	2	1	0
T1PINT ENABLE	Reserved	CMP3INT ENABLE	CMP2INT ENABLE	CMP1INT ENABLE	PDPINTA ENABLE
RW-0	R-0	RW-0	RW-0	RW-0	RW-1

Note: R = Read access, W = Write access, -n = value after reset

Bits 15–11 **Reserved**. Reads return zero; writes have no effect.

Bit 10 **T1OFINT ENABLE**

0	Disable
1	Enable

Bit 9 **T1UFINT ENABLE**

0	Disable
1	Enable

Bit 8 **T1CINT ENABLE**

0	Disable
1	Enable

Bit 7 **T1PINT ENABLE**

0	Disable
1	Enable

Bits 6–4 **Reserved**. Reads return zero; writes have no effect.

Bit 3 **CMP3INT ENABLE**

0	Disable
1	Enable

Bit 2 **CMP2INT ENABLE**

0	Disable
1	Enable

Bit 1 **CMP1INT ENABLE**

0	Disable
1	Enable

Bit 0 **PDPINTA ENABLE.** This is enabled (set to 1) following reset.

0 Disable

1 Enable

EVA Interrupt Mask Register B (EVAIMRB)

Figure 6–40. EVA Interrupt Mask Register B (EVAIMRB) — Address 742Dh

15–4	3	2	1	0
Reserved	T2OFINT ENABLE	T2UFINT ENABLE	T2CINT ENABLE	T2PINT ENABLE
R-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–4 **Reserved**. Reads return zero; writes have no effect.

Bit 3 **T2OFINT ENABLE**

0 Disable

1 Enable

Bit 2 **T2UFINT ENABLE**

0 Disable

1 Enable

Bit 1 **T2CINT ENABLE**

0 Disable

1 Enable

Bit 0 **T2PINT ENABLE**

0 Disable

1 Enable

EVA Interrupt Mask Register C (EVAIMRC)

Figure 6–41. EVA Interrupt Mask Register C (EVAIMRC) — Address 742Eh

15–3	2	1	0
Reserved	CAP3INT ENABLE	CAP2INT ENABLE	CAP1INT ENABLE
R-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–3 **Reserved**. Reads return zero; writes have no effect.

Bit 2 **CAP3INT ENABLE**

0 Disable

1 Enable

Bit 1 **CAP2INT ENABLE**

0 Disable

1 Enable

Bit 0 **CAP1INT ENABLE**

0 Disable

1 Enable

EVB Interrupt Flag Register A (EVBIFRA)

Figure 6–42. EVB Interrupt Flag Register A (EVBIFRA) — Address 752Fh

15–11	10	9	8
Reserved	T3OFINT FLAG	T3UFINT FLAG	T3CINT FLAG
R-0	RW-0	RW-0	RW-0

7	6–4	3	2	1	0
T3PINT FLAG	Reserved	CMP6INT FLAG	CMP5INT FLAG	CMP4INT FLAG	PDPINTB FLAG
RW-0	R-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–11 **Reserved**. Reads return zero; writes have no effect.

Bit 10 **T3OFINT FLAG**. GP timer 3 overflow interrupt.

Read: 0 Flag is reset

1 Flag is set

Write: 0 No effect

1 Reset flag

Bit 9 **T3UFINT FLAG**. GP timer 3 underflow interrupt.

Read: 0 Flag is reset

1 Flag is set

Write: 0 No effect

1 Reset flag

Bit 8 **T3CINT FLAG**. GP timer 3 compare interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 7 **T3PINT FLAG.** GP timer 3 period interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bits 6–4 **Reserved**. Reads return zero; writes have no effect.

Bit 3 **CMP6INT FLAG**. Compare 6 interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 2 **CMP5INT FLAG**. Compare 5 interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 1 **CMP4INT FLAG**. Compare 4 interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 0 **PDPINTB FLAG**. Power drive protection interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

EVB Interrupt Flag Register B (EVBIFRB)

Figure 6–43. EVB Interrupt Flag Register B (EVBIFRB) — Address 7530h

15–4	3	2	1	0
Reserved	T4OFINT FLAG	T4UFINT FLAG	T4CINT FLAG	T4PINT FLAG
R-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–4 **Reserved**. Reads return zero; writes have no effect.

Bit 3 **T4OFINT FLAG**. GP timer 4 overflow interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 2 **T4UFINT FLAG**. GP timer 4 underflow interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 1 **T4CINT FLAG**. GP timer 4 compare interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 0 **T4PINT FLAG**. GP timer 4 period interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

EVB Interrupt Flag Register C (EVBIFRC)

Figure 6–44. EVB Interrupt Flag Register C (EVBIFRC) — Address 7531h

15–3	2	1	0
Reserved	CAP6INT FLAG	CAP5INT FLAG	CAP4INT FLAG
R-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–3 **Reserved**. Reads return zero; writes have no effect.

Bit 2 **CAP6INT FLAG**. Capture 6 interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 1 **CAP5INT FLAG**. Capture 5 interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 0 **CAP4INT FLAG**. Capture 4 interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

EVB Interrupt Mask Register A (EVBIMRA)

Figure 6–45. EVB Interrupt Mask Register A (EVBIMRA) — Address 752Ch

15–11	10	9	8
Reserved	T3OFINT ENABLE	T3UFINT ENABLE	T3CINT ENABLE
R-0	RW-0	RW-0	RW-0

7	6–4	3	2	1	0
T3PINT ENABLE	Reserved	CMP6INT ENABLE	CMP5INT ENABLE	CMP4INT ENABLE	PDPINTB ENABLE
RW-0	R-0	RW-0	RW-0	RW-0	RW-1

Note: R = Read access, W = Write access, -n = value after reset

Bits 15–11 **Reserved**. Reads return zero; writes have no effect.

Bit 10 **T3OFINT ENABLE**

0 Disable

1 Enable

Bit 9 **T3UFINT ENABLE**

0 Disable

1 Enable

Bit 8 **T3CINT ENABLE**

0 Disable

1 Enable

Bit 7 **T3PINT ENABLE**

0 Disable

1 Enable

Bits 6–4 **Reserved**. Reads return zero; writes have no effect.

Bit 3 **CMP6INT ENABLE**

0 Disable

1 Enable

Bit 2 **CMP5INT ENABLE**

0 Disable

1 Enable

Bit 1 **CMP4INT ENABLE**

0 Disable

1 Enable

Bit 0 **PDPINTB ENABLE.** This is enabled (set to 1) following reset.

0 Disable

1 Enable

EVB Interrupt Mask Register B (EVBIMRB)

Figure 6–46. EVB Interrupt Mask Register B (EVBIMRB) — Address 752Dh

15–4	3	2	1	0
Reserved	T4OFINT ENABLE	T4UFINT ENABLE	T4CINT ENABLE	T4PINT ENABLE
R-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–4 **Reserved**. Reads return zero; writes have no effect.

Bit 3 **T4OFINT ENABLE**

0 Disable

1 Enable

Bit 2 **T4UFINT ENABLE**

0 Disable

1 Enable

Bit 1 **T4CINT ENABLE**

0 Disable

1 Enable

Bit 0 **T4PINT ENABLE**

0 Disable

1 Enable

EVB Interrupt Mask Register C (EVBIMRC)

Figure 6–47. EVB Interrupt Mask Register C (EVBIMRC) — Address 752Eh

15–3	2	1	0
Reserved	CAP6INT ENABLE	CAP5INT ENABLE	CAP4INT ENABLE
R-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 15–3 **Reserved**. Reads return zero; writes have no effect.

Bit 2 **CAP6INT ENABLE**

0	Disable
1	Enable

Bit 1 **CAP5INT ENABLE**

0	Disable
1	Enable

Bit 0 **CAP4INT ENABLE**

0	Disable
1	Enable

Chapter 7

Analog-to-Digital Converter (ADC)

Topic Page

7.1 Features

- 10-bit ADC core with built-in Sample and Hold (S/H)
- Fast conversion time (S/H + Conversion) of 500 ns
- Sixteen multiplexed analog inputs (ADCIN0 – ADCIN15). Eight in ’2402
- Autosequencing capability – up to 16 “autoconversions” in a single session. Each conversion session can be programmed to select any one of the 16 input channels.
- Two independent 8-state sequencers (SEQ1 and SEQ2) that can be operated individually in “dual-sequencer mode” or cascaded into one large 16-state sequencer (SEQ) in “cascaded mode”
- Four Sequencing Control Registers (CHSELSEQn) that determine the sequence of analog channels that are taken up for conversion in a given sequencing mode
- Sixteen (individually addressable) result registers to store the converted values (RESULT0 – RESULT15)
- Multiple trigger sources for start-of-conversion (SOC) sequence

Software:	Software immediate start (using SOC_SEQn bit)
EVA:	Event manager A (multiple event sources within EVA)
EVB:	Event manager B (multiple event sources within EVB)
External:	ADCSOC pin

- Flexible interrupt control allows interrupt request on every End of Sequence (EOS) or every other EOS
- Sequencer can operate in “start/stop” mode, allowing multiple “time-sequenced triggers” to synchronize conversions
- EVA and EVB can independently trigger SEQ1 and SEQ2, respectively. (This is applicable for dual-sequencer mode only.)
- Sample-and-hold acquisition time window has separate prescale control
- Built-in calibration mode
- Built-in “self-test” mode

Table 7–1. Addresses of ADC Registers

Address	Register	Name
70A0h	ADCTRL1	ADC control register 1
70A1h	ADCTRL2	ADC control register 2
70A2h	MAX_CONV	Maximum conversion channels register
70A3h	CHSELSEQ1	Channel select sequencing control register 1
70A4h	CHSELSEQ2	Channel select sequencing control register 2
70A5h	CHSELSEQ3	Channel select sequencing control register 3
70A6h	CHSELSEQ4	Channel select sequencing control register 4
70A7h	AUTO_SEQ_SR	Autosequence status register
70A8h	RESULT0	Conversion result buffer register 0
70A9h	RESULT1	Conversion result buffer register 1
70AAh	RESULT2	Conversion result buffer register 2
70ABh	RESULT3	Conversion result buffer register 3
70ACh	RESULT4	Conversion result buffer register 4
70ADh	RESULT5	Conversion result buffer register 5
70AEh	RESULT6	Conversion result buffer register 6
70AFh	RESULT7	Conversion result buffer register 7
70B0h	RESULT8	Conversion result buffer register 8
70B1h	RESULT9	Conversion result buffer register 9
70B2h	RESULT10	Conversion result buffer register 10
70B3h	RESULT11	Conversion result buffer register 11
70B4h	RESULT12	Conversion result buffer register 12
70B5h	RESULT13	Conversion result buffer register 13
70B6h	RESULT14	Conversion result buffer register 14
70B7h	RESULT15	Conversion result buffer register 15
70B8h	CALIBRATION	Calibration result, used to correct subsequent conversions

7.2 ADC Overview

7.2.1 Autoconversion Sequencer: Principle of Operation

The ADC sequencer consists of two independent 8-state sequencers (SEQ1 and SEQ2) that can also be cascaded together to form one 16-state sequencer (SEQ). The word “state” represents the number of autoconversions that can be performed with the sequencer. Block diagrams of the single (16-state, cascaded) and dual (two 8-state, separated) sequencer modes are shown in Figure 7–1 and Figure 7–2, respectively.

In both cases, the ADC has the ability to autosequence a series of conversions. For every conversion, any one of the available 16 input channels can be selected through the analog mux. After conversion, the digital value of the selected channel is stored in the appropriate result register (RESULTn). (The first result is stored in RESULT0, the second result in RESULT1, and so on). It is also possible to sample the same channel multiple times, allowing the user to perform “over-sampling”, which gives increased resolution over traditional single sampled conversion results.

Figure 7–1. Block Diagram of Autosequenced ADC in Cascaded Mode

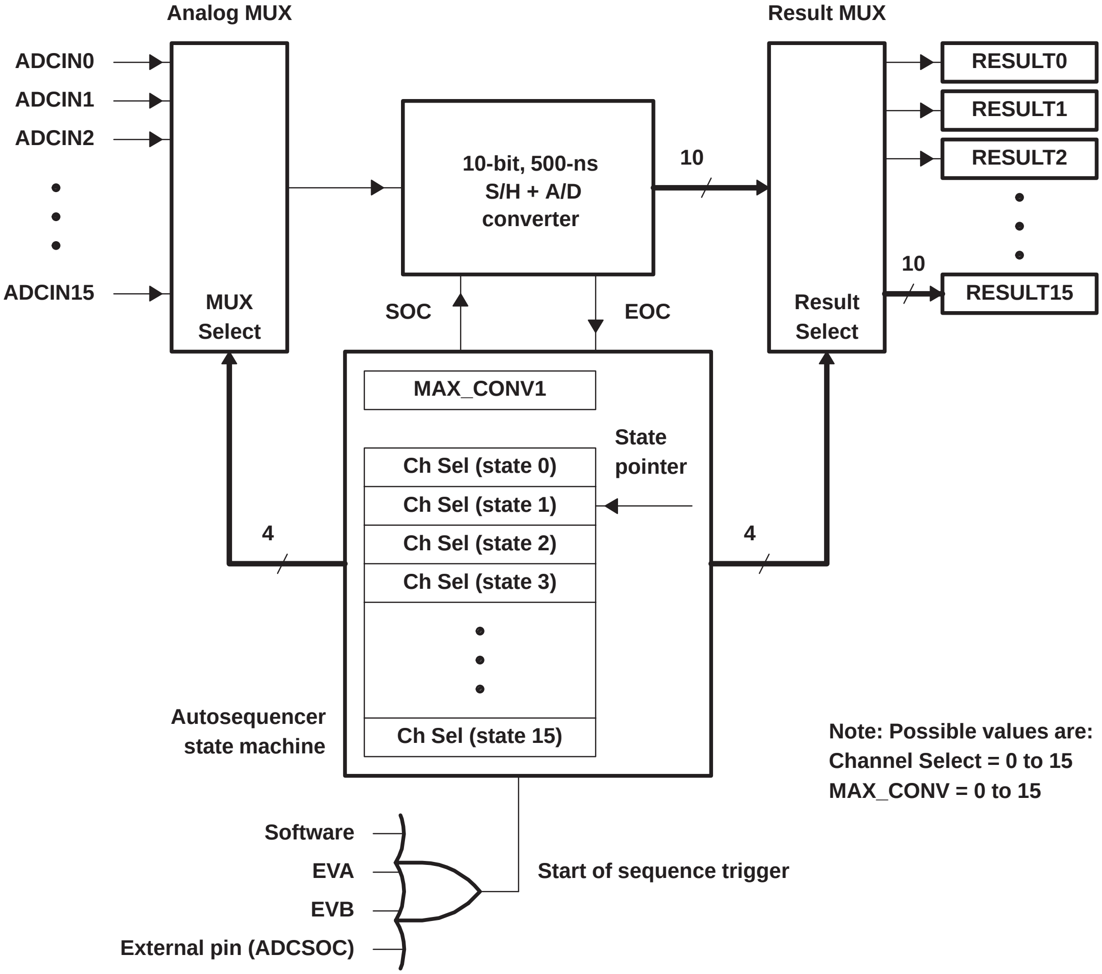
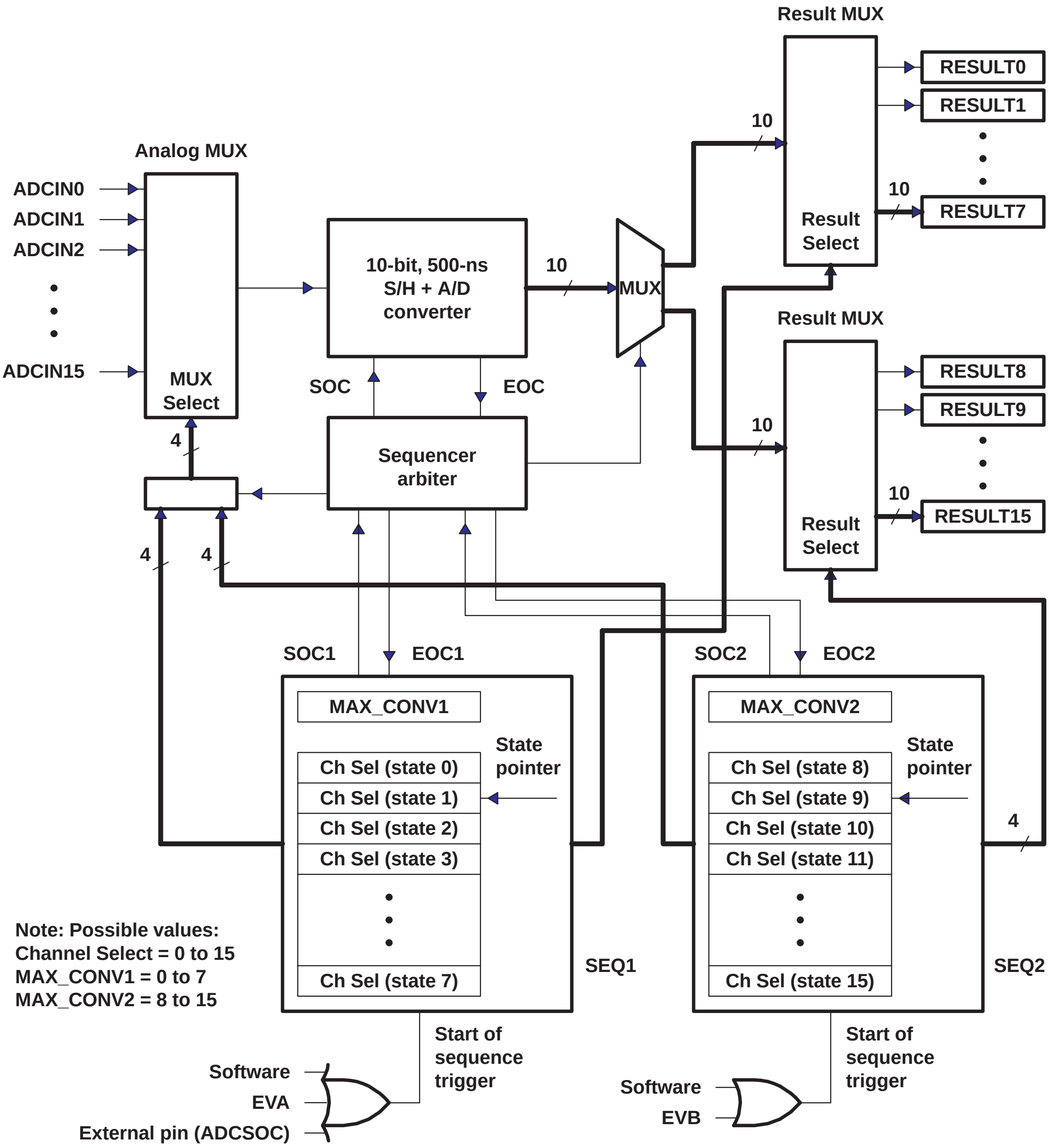


Figure 7–2. Block Diagram of Autosequenced ADC With Dual Sequencers



Note: Possible values:
Channel Select = 0 to 15
MAX_CONV1 = 0 to 7
MAX_CONV2 = 8 to 15

The sequencer operation for both 8-state and 16-state modes is almost identical; the few differences are highlighted in Table 7–2.

Table 7–2. Comparison of Single and Cascaded Operating Modes

Feature	Single 8-state sequencer #1 (SEQ1)	Single 8-state sequencer #2 (SEQ2)	Cascaded 16-state sequencer (SEQ)
Start of conversion triggers	EVA, software, external pin	EVB, software	EVA, EVB, software, external pin
Maximum number of autoconversions (i.e., sequence length)	8	8	16
Autostop at end of sequence (EOS)	Yes	Yes	Yes
Arbitration priority	High	Low	Not applicable
ADC conversion result register locations	0 to 7	8 to 15	0 to 15
CHSELSEQn bit field assignment	CONV00 to CONV07	CONV08 to CONV15	CONV00 to CONV15

For convenience, the sequencer states will be subsequently referred to as:

❑ For SEQ1:	CONV00 to CONV07
❑ For SEQ2:	CONV08 to CONV15
❑ For Cascaded SEQ:	CONV00 to CONV15

The analog input channel selected for each sequenced conversion is defined by CONVnn bit fields in the ADC Input Channel Select Sequencing Control Registers (CHSELSEQn). (See section 7.6.5.) CONVnn is a 4-bit field that specifies any one of the 16 channels for conversion. Since a maximum of 16 conversions in a sequence is possible when using the sequencers in cascaded mode, 16 such 4-bit fields (CONV00 – CONV15) are available and are spread across four 16-bit registers (CHSELSEQ1 – CHSELSEQ4). The CONVnn bits can have any value from 0 – 15. The analog channels can be chosen in any desired order and the same channel may be selected multiple times.

7.2.2 Basic Operation

The following description applies to the 8-state sequencers (SEQ1 or SEQ2). In this mode, SEQ1/SEQ2 can “autosequence” up to eight conversions of any channel in a single sequencing session. The result of each conversion is stored in one of the eight result registers (RESULT0 – RESULT7 for SEQ1 and RESULT8 – RESULT15 for SEQ2). These registers are filled from the lowest address to the highest address.

The number of conversions in a sequence is controlled by MAX_CONVn (a 3-bit or 4-bit field in the MAX_CONV register), which is automatically loaded into the Sequencing Counter Status bits (SEQ_CNTR3 – 0) in the Autosequence Status Register (AUTO_SEQ_SR) at the start of an autosequenced conversion session. The MAX_CONVn field can have a value ranging from 0 to 7. SEQ_CNTRn bits count down from their loaded value as the sequencer starts from state CONV00 and continues sequentially (CONV01, CONV02, and so on) until SEQ_CNTRn has reached zero. The number of conversions completed during an autosequencing session is equal to (MAX_CONVn + 1).

Example 7–1. Conversion in Dual-Sequencer Mode Using SEQ1

Suppose seven conversions are desired from SEQ1 (i.e., Channels 2, 3, 2, 3, 6, 7, and 12 need to be converted as part of the autosequenced session), then MAX_CONV1 should be set to 6 and the CHSELSEQn registers should be set to the values shown in the table below:

	Bits 15–12	**Bits 11–8**	**Bits 7–4**	**Bits 3–0**	
70A3h	3	2	3	2	CHSELSEQ1
70A4h	x	12	7	6	CHSELSEQ2
70A5h	x	x	x	x	CHSELSEQ3
70A6h	x	x	x	x	CHSELSEQ4

Note: Values are in decimal, and x = don’t care

Conversion begins once the start-of-conversion (SOC) trigger is received by the sequencer. The SOC trigger also loads the SEQ_CNTR_n bits. Those channels that are specified in the CHSELSEQn registers are taken up for conversion, in the predetermined sequence. The SEQ_CNTR_n bits are decremented by one automatically after every conversion. Once SEQ_CNTR_n reaches zero, two things can happen depending on the status of the Continuous Run bit (CONT_RUN) in the ADCTRL1 register.

❑ If CONT_RUN is set, the conversion sequence starts all over again automatically (i.e., SEQ_CNTR_n gets reloaded with the original value in MAX_CONV1 and SEQ1 state is set to CONV00). In this case, the user must ensure that the result registers are read before the next conversion sequence begins. The arbitration logic designed into the ADC ensures that the result registers are not corrupted should a contention arise (ADC module trying to write into the result registers while the user tries to read from them at the same time).

❑ If CONT_RUN is not set, the sequencer stays in the last state (CONV06, in this example) and SEQ_CNTR_n continues to hold a value of zero.

Since the interrupt flag is set every time SEQ_CNTR_n reaches zero, the user can (if needed) manually reset the sequencer (using the RST_SEQn bit in the ADCTRL2 register) in the Interrupt Service Routine (ISR), so that SEQ_CNTR_n gets reloaded with the original value in MAX_CONV1 and SEQ1 state is set to CONV00. This feature is useful in the “Start/Stop” operation of the sequencer. Example 7–1 also applies to SEQ2 and the cascaded 16-state sequencer (SEQ) with differences outlined in Table 7–2.

7.2.3 Sequencer “Start/Stop” Operation With Multiple “Time-Sequenced Triggers”

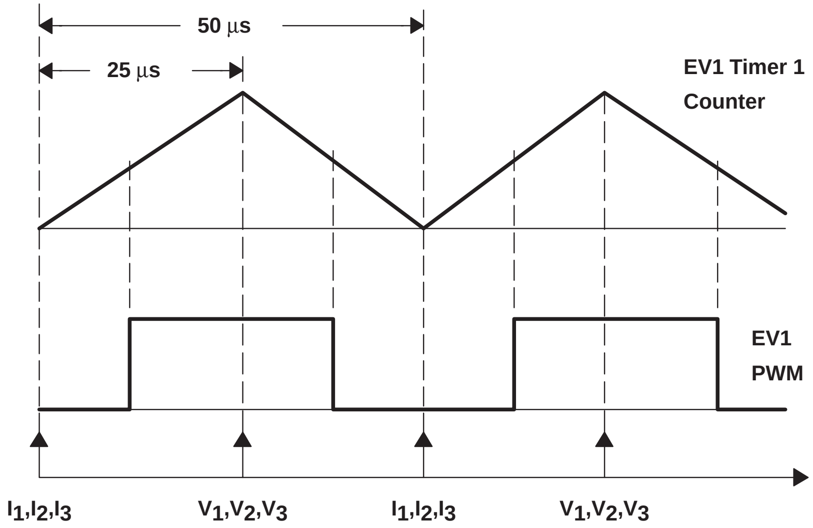
In addition to the mode described above, any sequencer (SEQ1, SEQ2, or SEQ) can be operated in a “stop/start” mode which is synchronized to multiple start-of-conversion (SOC) triggers, separated in time. This mode is identical to Example 7–1, but the sequencer is allowed to be retriggered without being reset to the initial state CONV00, once it has finished its first sequence (i.e., the sequencer is not reset in the interrupt service routine). Therefore, when one conversion sequence ends, the sequencer stays in the current conversion state. The Continuous Run bit (CONT_RUN) in the ADCTRL1 register must be set to 0 (i.e., disabled) for this mode.

Example 7–2. Sequencer “Start/Stop” Operation

Requirement: To start three autoconversions (e.g., I_1,I_2,I_3) off trigger 1 (underflow) and three autoconversions (e.g., V_1,V_2,V_3) off trigger 2 (period). Triggers 1 and 2 are separated in time by, say, 25 μs and are provided by Event Manager A (EVA). See Figure 7–3. Only SEQ1 is used in this case.

Note: Triggers 1 and 2 may be a SOC signal from EVA, external pin, or software. The same trigger source may occur twice to satisfy the dual-trigger requirement of this example.

Figure 7–3. Example of Event Manager Triggers to Start the Sequencer



Here MAX_CONV1 is set to 2 and the ADC Input Channel Select Sequencing Control Registers (CHSELSEQn) are set to:

	Bits 15–12	Bits 11–8	Bits 7–4	Bits 3–0	
70A3h	V_1	I_3	I_2	I_1	CHSELSEQ1
70A4h	x	x	V_3	V_2	CHSELSEQ2
70A5h	x	x	x	x	CHSELSEQ3
70A6h	x	x	x	x	CHSELSEQ4

Once reset and initialized, SEQ1 waits for a trigger. With the first trigger, three conversions with channel-select values of: CONV00 (I_1), CONV01 (I_2), and CONV02 (I_3) are performed. SEQ1 then waits at current state for another trigger. Twenty-five microseconds later when the second trigger arrives, another three conversions occur, with channel-select values of CONV03 (V_1), CONV04 (V_2), and CONV05 (V_3).

The value of MAX_CONV1 is automatically loaded into SEQ_CNTR_n for both trigger cases. If a different number of conversions are required at the second trigger point, the user must (at some appropriate time before the second trigger) change the value of MAX_CONV1 through software, otherwise, the current (originally loaded) value will be reused. This can be done by an ISR that changes the value of MAX_CONV1 at the appropriate time. The interrupt operation modes are described in section 7.2.5, “Interrupt Operation During Sequenced Conversions”.

At the end of the second autoconversion session, the ADC result registers will have following values:

Buffer Register	ADC conversion result buffer
RESULT0	I_1
RESULT1	I_2
RESULT2	I_3
RESULT3	V_1
RESULT4	V_2
RESULT5	V_3
RESULT6	x
RESULT7	x
RESULT8	x
RESULT9	x
RESULT10	x
RESULT11	x
RESULT12	x
RESULT13	x
RESULT14	x
RESULT15	x

At this point, SEQ1 keeps “waiting” at the current state for another trigger. Now, the user can reset SEQ1 (by software) to state CONV00 and repeat the same trigger1,2 sessions.

7.2.4 Input Trigger Description

Each sequencer has a set of trigger inputs that can be enabled/disabled. The valid input triggers for SEQ1, SEQ2, and cascaded SEQ is as follows:

SEQ1 (sequencer 1)	SEQ2 (sequencer 2)	Cascaded SEQ
Software trigger (software SOC)	Software trigger (software SOC)	Software trigger (software SOC)
Event manager A (EVA SOC)	Event manager B (EVB SOC)	Event manager A (EVA SOC)
External SOC pin (ADC_SOC)		Event manager B (EVB SOC)
		External SOC pin (ADC_SOC)

Note that:

- ❏ A SOC trigger can initiate an autoconversion sequence whenever a sequencer is in “idle” state. An idle state is either CONV00 prior to receiving a trigger, or any state which the sequencer lands on at the completion of a conversion sequence, i.e., when SEQ_CNTR_n has reached a count of zero.

- ❏ If a SOC trigger occurs while a current conversion sequence is underway, it sets the SOC_SEQn bit (which would have been cleared on the commencement of a previous conversion sequence) in the ADCTRL2 register. If yet another SOC trigger occurs, it is lost (i.e., when the SOC_SEQn bit is already set (SOC pending), subsequent triggers will be ignored).

- ❏ Once triggered, the sequencer cannot be stopped/halted in mid sequence. The program must either wait until an End of Sequence (EOS) or initiate a sequencer reset, which brings the sequencer immediately back to the idle start state (CONV00 for SEQ1 and cascaded cases; CONV08 for SEQ2).

- ❏ When SEQ1/2 are used in cascaded mode, triggers going to SEQ2 are ignored, while SEQ1 triggers are active. Cascaded mode can be viewed as SEQ1 with 16 states instead of 8.

7.2.5 Interrupt Operation During Sequenced Conversions

The sequencer can generate interrupts under two operating modes. These modes are determined by the Interrupt-Mode-Enable Control bits in ADCTRL2.

A variation of Example 7–2 can be used to show how interrupt mode 1 and mode 2 are useful under different operating conditions.

Case 1: Number of samples in the first and second sequences are not equal

- ❑ Mode 1 Interrupt operation (i.e., Interrupt request occurs at every EOS)

1) Sequencer is initialized with MAX_CONVn = 1 for converting I_1 and I_2

2) At ISR “a”, MAX_CONVn is changed to 2 (by software) for converting V_1, V_2, and V_3

3) At ISR “b”, the following events take place :

    1) MAX_CONVn is changed to 1 again for converting I_1 and I_2.

    2) Values I_1, I_2, V_1, V_2, and V_3 are read from ADC result registers.

    3) The sequencer is reset.

4) Steps 2 and 3 are repeated. Note that the interrupt flag is set every time SEQ_CNTR_n reaches zero and both interrupts are recognized.

Case 2: Number of samples in the first and second sequences are equal

- ❑ Mode 2 Interrupt operation (i.e., Interrupt request occurs at every other EOS)

1) Sequencer is initialized with MAX_CONVn = 2 for converting I_1, I_2, and I_3 (or V_1, V_2, and V_3).

2) At ISR “b” and “d”, the following events take place :

    1) Values I_1, I_2, I_3,V_1, V_2, and V_3 are read from ADC result registers.

    2) The sequencer is reset.

3) Step 2 is repeated. Note that the interrupt flag is set every time SEQ_CNTR_n reaches zero. This would happen after the ADC has finished converting I_1, I_2, and I_3 and also after converting V_1, V_2, and V_3. But, only the EOS generated after the conversion of V_1, V_2, and V_3 triggers the interrupt.

Case 3: Number of samples in the first and second sequences are equal (with dummy read)

- ❑ Mode 2 Interrupt operation (i.e., Interrupt request occurs at every other EOS)

1) Sequencer is initialized with MAX_CONVn = 2 for I_1, I_2, x sampling

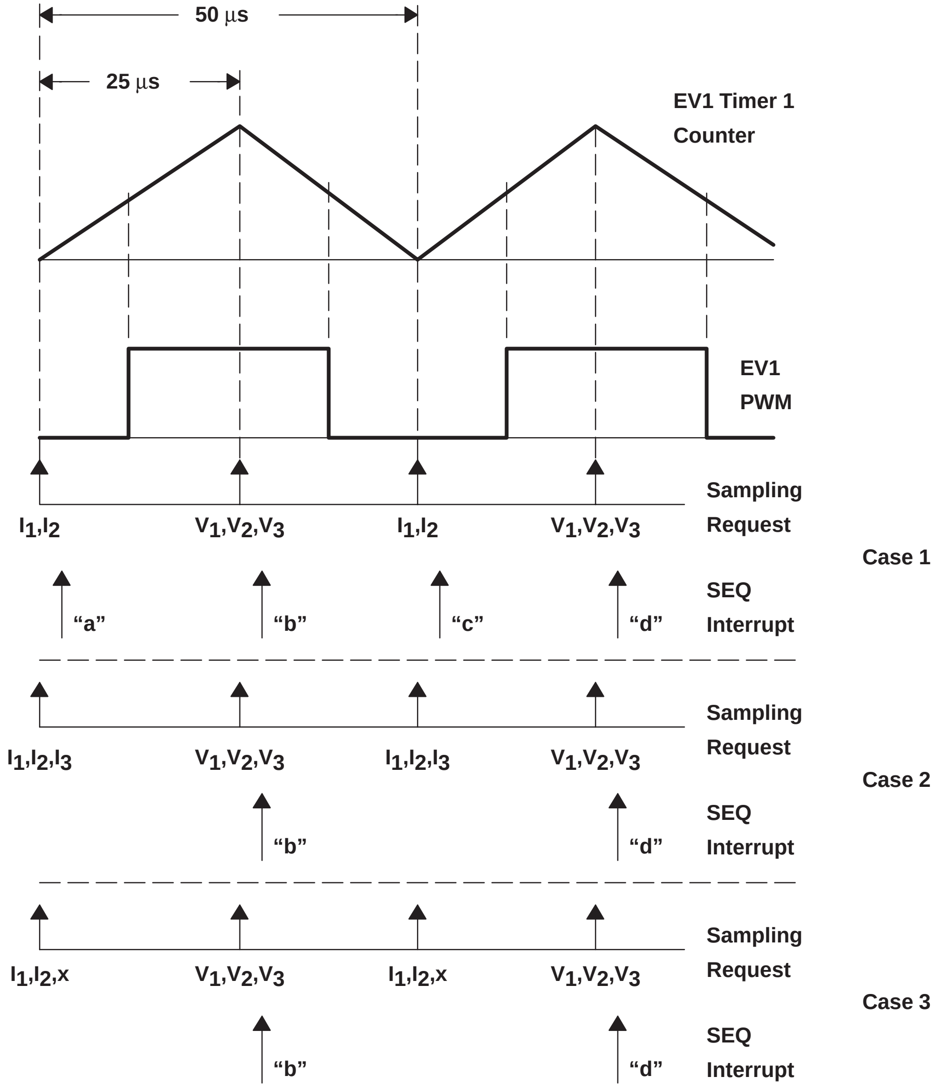
2) At ISR “b” and “d”, the following events take place :

    1) Values I_1, I_2, x,V_1, V_2, and V_3 are read from ADC result registers.

    2) The sequencer is reset.

3) Step 2 is repeated. Note that the third I-sample (x) is a dummy sample, and is not really required. However, to minimize ISR overhead and CPU intervention, advantage is taken of the “every other” Interrupt request feature of Mode 2.

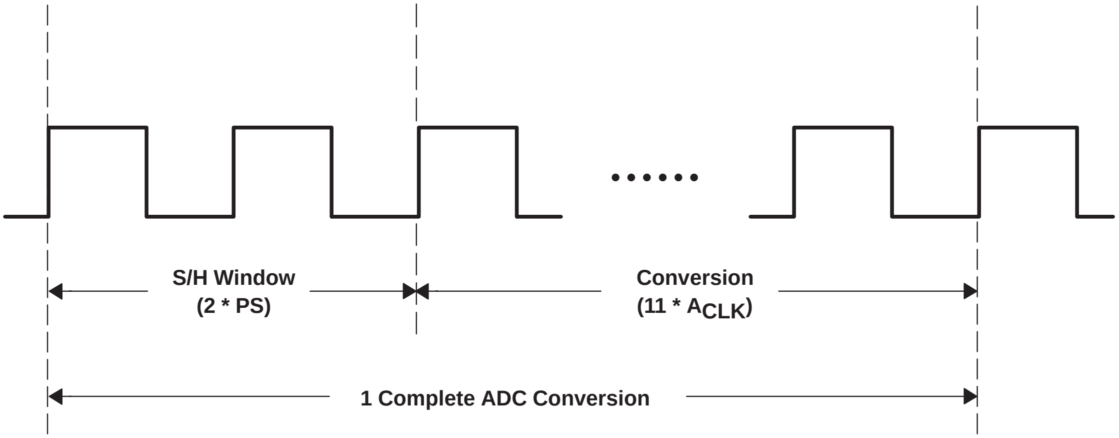
Figure 7–4. Interrupt Operation During Sequenced Conversions



7.3 ADC Clock Prescaler

The S/H block in the ’240x ADC can be tailored to accomodate the variation in source impedances. This is achieved by the ACQ_PS3–ACQ_PS0 bits and the CPS bit in the ADCTR1 register. The analog-to-digital conversion process can be divided into two time segments, as shown in Figure 7–5.

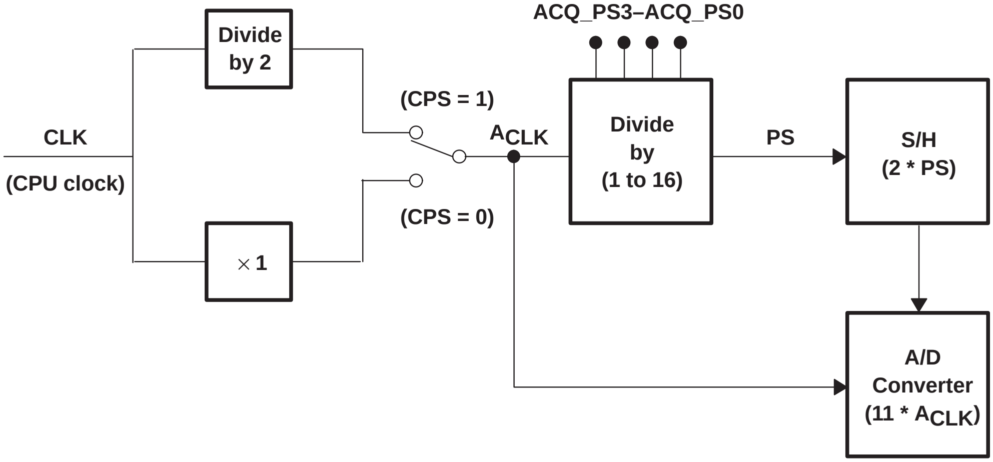
Figure 7–5. ADC Conversion Time



PS = a prescaled CPU clock

PS will be the same as the CPU clock if the prescaler = 1 (i.e., ACQ_PS3–ACQ_PS0 bits are all zero) and if CPS = 0. For any other value of the prescaler, the magnitude of PS will be magnified (effectively increasing the S/H window time) as described by the “Acquisition Time Window” column in the bit description for ACQ_PS3–ACQ_PS0. If the CPS bit is made 1, the S/H window is doubled. This doubling of the S/H window is in addition to the “stretching” provided by the prescaler. Figure 7–6 shows the role played by the various prescaler bits in the ADC module. Note that PS and A_{CLK} will be equal to CPU clock if CPS = 0.

Figure 7–6. Clock Prescalers in '240x ADC



7.4 Calibration

In the calibration mode, the sequencers are not operational and the ADCINn pins are not connected to the A/D converter. The signal that gets connected to the A/D converter input is determined by BRG_ENA (Bridge Enable) and HI/LO (V_{REFHI}/V_{REFLO} selection) bits. These two signals connect either V_{REFLO} or V_{REFHI} or their midpoint to the A/D converter input and a single conversion is then done. The calibration mode can calculate the zero, midpoint or full-scale offset errors of the ADC. The 2's complement of the offset error should then be loaded in the CALIBRATION register. From that point on, the ADC hardware automatically adds the offset error to the converted value.

To summarize, the CALIBRATION register stores the end result of calibration in the calibration mode. In the normal mode of the ADC, the value in the CALIBRATION register is automatically added to the output of the ADC before the result is stored in the RESULTn register.

In order to obtain the best results for midpoint offset calcualtions, both $|(V_{REFHI} - V_{REFLO})/2|$ and $|(V_{REFLO} - V_{REFHI})/2|$ must be used as the reference voltage and the calibration done. The two results should then be averaged.

7.5 Self-Test

The self-test mode is a way to detect shorts/opens on an ADC pin. The sampling period is doubled in this mode. In the first half of the sampling period, either V_{REFHI} or V_{REFLO} (as determined by the HI/LO bit) is connected to the input of the A/D converter, in addition to the analog input signal provided by the user. In the second half of the sampling period, only user signal is connected to the ADC. Assuming there is an open (and V_{REFHI} is used), the result register would contain the digital representation of V_{REFHI} only.

The self-test mode should be used only to check for shorts and opens. It should not be used during normal operation. In addition, the calibration mode and the self-test mode should not be enabled at the same time.

7.6 Register Bit Descriptions

7.6.1 ADC Control Register 1

Figure 7–7. ADC Control Register 1 (ADCTRL1) — Address 70A0h

15	14	13	12	11	10	9	8
Reserved	RESET	SOFT	FREE	ACQ_PS3	ACQ_PS2	ACQ_PS1	ACQ_PS0
	RS-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
CPS	CONT_RUN	INT_PRI	SEQ_CASC	CAL_ENA	BRG_ENA	HI/LO	STEST_ ENA
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, S = Set only, -0 = value after reset

Bit 15 **Reserved**

Bit 14 **RESET**. ADC module software reset

This bit causes a master reset on the entire ADC module. All register bits and sequencer state machines are reset to the initial state as occurs when the device reset pin is pulled low (or after a power-on reset).

0 No effect

1 Resets entire ADC module (bit is then set back to 0 by ADC logic)

Bits 13, 12 **SOFT and FREE**. Soft and Free bits

These bits determine what occurs when an emulation-suspend occurs (due to the debugger hitting a breakpoint, for example). In free-run mode, the peripheral can continue with whatever it is doing. In stop mode, the peripheral can either stop immediately or stop when the current operation (i.e., the current conversion) is complete.

Soft	Free	
0	0	Immediate stop on suspend
1	0	Complete current conversion before stopping
X	1	Free run, continue operation regardless of suspend

Bits 11–8 **ACQ_PS3 – ACQ_PS0**. Acquisition time window – prescale bits 3–0

These bits define the ADC clock prescale factor applied to the acquisition portion of the conversion. The prescale values are defined below:

#	ACQ_ PS3	ACQ_ PS2	ACQ_ PS1	ACQ_ PS0	Pre- scaler (div by)	Acquisition Time Window	Source Z (CPS=0) (Ω)	Source Z (CPS=1) (Ω)
0	0	0	0	0	1	2 x T_{clk}	67	385
1	0	0	0	1	2	4 x T_{clk}	385	1020
2	0	0	1	0	3	6 x T_{clk}	702	1655
3	0	0	1	1	4	8 x T_{clk}	1020	2290
4	0	1	0	0	5	10 x T_{clk}	1337	2925
5	0	1	0	1	6	12 x T_{clk}	1655	3560
6	0	1	1	0	7	14 x T_{clk}	1972	4194
7	0	1	1	1	8	16 x T_{clk}	2290	4829
8	1	0	0	0	9	18 x T_{clk}	2607	5464
9	1	0	0	1	10	20 x T_{clk}	2925	6099
A	1	0	1	0	11	22 x T_{clk}	3242	6734
B	1	0	1	1	12	24 x T_{clk}	3560	7369
C	1	1	0	0	13	26 x T_{clk}	3877	8004
D	1	1	0	1	14	28 x T_{clk}	4194	8639
E	1	1	1	0	15	30 x T_{clk}	4512	9274
F	1	1	1	1	16	32 x T_{clk}	4829	9909

Notes: 1) Period of T_{clk} is dependent on the “Conversion Clock Prescale” bit (Bit 7); i.e.,
CPS = 0: T_{clk} = 1/CLK (for example, for CLK = 30 MHz, T_{clk} = 33 ns)
CPS = 1: T_{clk} = 2 × (1/CLK) (for example, for CLK = 30 MHz, T_{clk} = 66 ns)

2) Source impedance Z is a design estimate only.

Bit 7 **CPS**. Conversion clock prescale

This bit defines the ADC conversion logic clock prescale

0 F_{clk} = CLK/1

1 F_{clk} = CLK/2

CLK = CPU clock frequency

Bit 6 **CONT_RUN**. Continuous run

This bit determines whether the sequencer operates in continuous conversion mode or start-stop mode. This bit can be written while a current conversion sequence is active. This bit will take effect at the end of the current conversion sequence; i.e., software can set/clear this bit until EOS has occurred, for valid action to be taken. In the continuous conversion mode, there is no need to reset the sequencer; however, the sequencer must be reset in the start-stop mode to put the converter in state CONV00.

0 Start-stop mode. Sequencer stops after reaching EOS. This is used for multiple time-sequenced triggers.

1 Continuous conversion mode. After reaching EOS, the sequencer starts all over again from state CONV00 (for SEQ1 and cascaded) or CONV08 (for SEQ2).

Bit 5 **INT_PRI**. ADC interrupt request priority

0 High priority

1 Low priority

Bit 4 **SEQ_CASC**. Cascaded sequencer operation

This bit determines whether SEQ1 and SEQ2 operate as two 8-state sequencers or as a single 16-state sequencer (SEQ).

0 Dual-sequencer mode. SEQ1 and SEQ2 operate as two 8-state sequencers.

1 Cascaded mode. SEQ1 and SEQ2 operate as a single 16-state sequencer (SEQ).

Bit 3 **CAL_ENA**. Offset calibration enable

When set to 1, CAL_ENA disables the input channel multiplexer, and connects the calibration reference selected by the bits HI/LO and BRG_ENA to the ADC core inputs. The calibration conversion can then be started by setting bit 14 of ADCTRL2 register (STRT_CAL) to 1. Note that CAL_ENA should be set to 1 first before the STRT_CAL bit can be used.

Note: This bit should not be set to 1 if STEST_ENA = 1

0 Calibration mode disabled

1 Calibration mode enabled

Bit 2 **BRG_ENA**. Bridge enable

Together with the HI/LO bit, BRG_ENA allows a reference voltage to be converted in calibration mode. See the description of the HI/LO bit for reference voltage selections during calibration.

0 Full reference voltage is applied to the ADC input

1 A reference midpoint voltage is applied to the ADC input

Bit 1 **HI/LO**. V_{REFHI}/V_{REFLO} selection

When the fail self-test mode is enabled (STEST_ENA = 1), HI/LO defines the test voltage to be connected. In calibration mode, HI/LO defines the reference source polarity; see Table 7–3. In normal operating mode, HI/LO has no effect.

0 V_{REFLO} is used as precharge value at ADC input

1 V_{REFHI} is used as precharge value at ADC input

Table 7–3. Reference Voltage Bit Selection

BRG_ENA	HI/LO	CAL_ENA = 1 Reference voltage (V)	STEST_ENA = 1 Reference voltage (V)
0	0	V_{REFLO}	V_{REFLO}
0	1	V_{REFHI}	V_{REFHI}
1	0	$|(V_{REFHI} – V_{REFLO}) / 2|$	V_{REFLO}
1	1	$|(V_{REFLO} – V_{REFHI}) / 2|$	V_{REFHI}

Bit 0 **STEST_ENA**. Self-test function enable

0 Self-test mode disabled

1 Self-test mode enabled

7.6.2 ADC Control Register 2

Figure 7–8. ADC Control Register 2 (ADCTRL2) — Address 70A1h

15	14	13	12	11	10	9	8
EVB_SOC_ SEQ	RST_SEQ1/ STRT_CAL	SOC_SEQ1	SEQ1_BSY	INT_ENA_ SEQ1 (Mode 1)	INT_ENA_ SEQ1 (Mode 0)	INT_FLAG_ SEQ1	EVA_SOC_ SEQ1
RW-0	RS-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
EXT_SOC_ SEQ1	RST_SEQ2	SOC_SEQ2	SEQ2_BSY	INT_ENA_ SEQ2 (Mode 1)	INT_ENA_ SEQ2 (Mode 0)	INT_FLAG_ SEQ2	EVB_SOC_ SEQ2
RW-0	RS-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, S = Set only, -0 = value after reset

Bit 15 **EVB_SOC_SEQ**. EVB SOC enable for cascaded sequencer *(Note: This bit is active only in cascaded mode.)*

0 No action

1 Setting this bit allows the cascaded sequencer to be started by an Event Manager B signal. The Event Manager can be programmed to start a conversion on various events. See chapter 6, Event Manager (EV), for details.

Bit 14 **RST_SEQ1 / STRT_CAL**. Reset Sequencer/Start Calibration

Case: Calibration Disabled (Bit 3 of ADCTRL1) = 0

Writing a 1 to this bit will reset the sequencer immediately to an initial “pretriggered” state, i.e., waiting for a trigger at CONV00. A currently active conversion sequence will be aborted.

0	No action
1	Immediately reset sequencer to state CONV00

Case: Calibration Enabled (Bit 3 of ADCTRL1) = 1

Writing a 1 to this bit will begin the converter calibration process.

0	No action
1	Immediately start calibration process

Bit 13 **SOC_SEQ1**. Start-of-conversion (SOC) trigger for Sequencer 1 (SEQ1). This bit can be set by the following triggers:

❑ S/W – Software writing a 1 to this bit

❑ EVA – Event Manager A

❑ EVB – Event Manager B (only in cascaded mode)

❑ EXT – External pin (i.e., the ADCSOC pin)

When a trigger occurs, there are 3 possibilities:

Case 1: SEQ1 idle and SOC bit clear
SEQ1 starts immediately (under arbiter control). This bit is set and cleared, allowing for any “pending” trigger requests.

Case 2: SEQ1 busy and SOC bit clear
Bit is set signifying a trigger request is pending. When SEQ1 finally starts after completing current conversion, this bit will be cleared.

Case 3: SEQ1 busy and SOC bit set
Any trigger occurring in this case will be ignored (lost).

0	Clears a pending SOC trigger. Note: If the sequencer has already started, this bit will automatically be cleared, and hence, writing a zero will have no effect; i.e., an already started sequencer cannot be stopped by clearing this bit.
1	Software trigger – Start SEQ1 from currently stopped position (i.e., Idle mode)

Bit 12 **SEQ1_BSY**. SEQ1 Busy

This bit is set to a 1 while the ADC autoconversion sequence is in progress. It is cleared when the conversion sequence is complete.

0 Sequencer is Idle (i.e., waiting for trigger).

1 Conversion sequence is in progress.

Bits 11–10 **INT_ENA_SEQ1**. Interrupt-mode-enable control for SEQ1

Bit 11	Bit 10	Operation Description
0	0	Interrupt is Disabled
0	1	Interrupt **Mode 1** Interrupt requested immediately when INT_FLAG_SEQ1 flag is set
1	0	Interrupt **Mode 2** Interrupt requested only if INT_FLAG_SEQ1 flag is already set. If clear, INT_FLAG_SEQ1 flag is set and INT request is suppressed. (This mode allows Interrupt requests to be generated for every other EOS.)
1	1	Reserved

Bit 9 **INT_FLAG_SEQ1.** ADC interrupt flag bit for SEQ1

This bit indicates whether an interrupt event has occurred or not. This bit must be cleared by the user writing a 1 to it.

0 No interrupt event

1 An interrupt event has occurred.

Bit 8 **EVA_SOC_SEQ1.** Event Manager A SOC mask bit for SEQ1

0 SEQ1 cannot be started by EVA trigger.

1 Allows SEQ1 to be started by Event Manager A trigger. The Event Manager can be programmed to start a conversion on various events. See chapter 6, Event Manager (EV), for details.

Bit 7 **EXT_SOC_SEQ1.** External signal start-of-conversion bit for SEQ1

0 No action

1 Setting this bit enables an ADC autoconversion sequence to be started by a signal from the ADCSOC device pin.

Bit 6 **RST_SEQ2.** Reset SEQ2

0 No action

1 Immediately resets SEQ2 to an initial “pretriggered” state, i.e., waiting for a trigger at CONV08. A currently active conversion sequence will be aborted.

Bit 5 **SOC_SEQ2.** Start-of-conversion trigger for Sequencer 2 (SEQ2)
(Only applicable in dual-sequencer mode, ignored in cascaded mode.)

This bit can be set by the following triggers:

- ❑ S/W – Software writing of 1 to this bit
- ❑ EVB – Event Manager B

When a trigger occurs, there are 3 possibilities:

Case 1: SEQ2 idle and SOC bit clear
SEQ2 starts immediately (under arbiter control) and the bit is cleared, allowing for any "pending" trigger requests.

Case 2: SEQ2 busy and SOC bit clear
Bit is set signifying a trigger request is pending. When SEQ2 finally starts after completing current conversion, this bit will be cleared.

Case 3: SEQ2 busy and SOC bit set
Any trigger occurring in this case will be ignored (lost).

0 Clears a Pending SOC trigger.
Note: If the sequencer has already started, this bit will automatically be cleared, and hence, writing a zero will have no effect; i.e., an already started sequencer cannot be stopped by clearing this bit.

1 Software trigger – Start SEQ2 from currently stopped position (i.e., Idle mode)

Bit 4 **SEQ2_BSY.** SEQ2 Busy

This bit is set to a 1 while the ADC autoconversion sequence is in progress. It is cleared when the conversion sequence is complete.

0 Sequencer is idle (i.e., waiting for trigger).

1 Conversion sequence is in progress.

Bits 3–2 **INT_ENA_SEQ2.** Interrupt-mode-enable control for SEQ2

Bit 3	Bit 2	Operation Description
0	0	Interrupt is Disabled
0	1	Interrupt **Mode 1** Interrupt requested immediate on INT_FLAG_SEQ2 flag set
1	0	Interrupt **Mode 2** Interrupt requested only if INT_FLAG_SEQ2 flag is already set. If clear, INT_FLAG_SEQ2 flag is set and INT request is suppressed. (This mode allows Interrupt requests to be generated for every other EOS)
1	1	Reserved

Bit 1 **INT_FLAG_SEQ2.** ADC interrupt flag bit for SEQ2

This bit indicates whether an interrupt event has occurred or not. This bit must be cleared by the user writing a 1 to it.

0 No interrupt event.

1 An interrupt event has occurred.

Bit 0 **EVB_SOC_SEQ2.** Event Manager B SOC mask bit for SEQ2

0 SEQ2 cannot be started by EVB trigger.

1 Allows SEQ2 to be started by Event Manager B trigger. The Event Manager can be programmed to start a conversion on various events. See chapter 6, Event Manager (EV), for details.

7.6.3 Maximum Conversion Channels Register

Figure 7–9. Maximum Conversion Channels Register (MAX_CONV) — Address 70A2h

15–8
Reserved
R-x

7	6	5	4	3	2	1	0
Reserved	MAX_ CONV2_2	MAX_ CONV2_1	MAX_ CONV2_0	MAX_ CONV1_3	MAX_ CONV1_2	MAX_ CONV1_1	MAX_ CONV1_0
R-x	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, x = undefined, -0 = value after reset

Bits 15–7 **Reserved**

Bits 6–0 **MAX_CONVn.** MAX_CONVn bit field defines the maximum number of autoconversions. The bit fields and their operation vary according to the sequencer modes (dual/cascaded). The following table summarizes the differences.

Value	SEQ1	SEQ2	Cascaded
Initial state	CONV00	CONV08	CONV00
End state	CONV07	CONV15	CONV15
Max value	7	7	15
Max value + 1	8	8	16

This register contains the number of conversions executed during an autoconversion session. An autoconversion session always starts with the “initial state” and continues sequentially until the “end state” if allowed. The result buffer is filled in a sequential order. Any number of conversions between 1 and (MAX_CONVn +1) can be programmed for a session,

❑ For SEQ1 operation, bits MAX_CONV1_2 – 0 are used.

❑ For SEQ2 operation, bits MAX_CONV2_2 – 0 are used.

❑ For SEQ operation, bits MAX_CONV1_3 – 0 are used.

Example 7–3. MAX_CONV Register Bits Programming

If only five conversions are required, then MAX_CONVn is set to four.

Case 1: Dual mode SEQ1 and cascaded mode
Sequencer goes from CONV00 to CONV04, and the five conversion results are stored in the registers Result 00 to Result 04 of the Conversion Result Buffer.

Case 2: Dual mode SEQ2
Sequencer goes from CONV08 to CONV12, and the five conversion results are stored in the registers Result 08 to Result 12 of the Conversion Result Buffer.

MAX_CONV1 Value >7 for Dual-Sequencer Mode

If a value for MAX_CONV1, which is greater than 7, is chosen for the dual-sequencer mode (i.e., two separate 8-state sequencers), then SEQ_CNTR_n will continue counting past seven, causing the sequencer to “wrap around” to CONV00 and continue counting.

Table 7–4. Bit Selections for MAX_CONV1 for Various Number of Conversions

MAX_CONV1.3–0	Number of conversions
0000	1
0001	2
0010	3
0011	4
0100	5
0101	6
0110	7
0111	8
1000	9
1001	10
1010	11
1011	12
1100	13
1101	14
1110	15
1111	16

7.6.4 Autosequence Status Register

Figure 7–10. Autosequence Status Register (AUTO_SEQ_SR) — Address 70A7h

15–12	11	10	9	8
Reserved	SEQ_ CNTR_3	SEQ_ CNTR_2	SEQ_ CNTR_1	SEQ_ CNTR_0
R-x	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
SEQ2- State3	SEQ2- State2	SEQ2- State1	SEQ2- State0	SEQ1- State3	SEQ1- State2	SEQ1- State1	SEQ1- State0
R-1	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Note: R = Read access, x = undefined, -0 = value after reset

Bits 15–12 **Reserved**

Bits 11–8 **SEQ_CNTR_3 – SEQ_CNTR_0.** Sequencing counter status bits

The SEQ_CNTR_n 4-bit status field is used by SEQ1, SEQ2, and the cascaded sequencer.

SEQ2 is irrelevant in cascaded mode.

At the start of an autosequenced session, SEQ_CNTR_n is loaded with the value from MAX_CONVn. The SEQ_CNTR_n bits can be read at any time during the countdown process to check status of the sequencer. This value, together with the SEQ1 and SEQ2 Busy bits, uniquely identifies the progress or state of the active sequencer at any point in time.

Table 7–5. Status Bit Values for SEQ_CNTR_n

SEQ_CNTR_n (read only)	**Number of conversions remaining**
0000	1
0001	2
0010	3
0011	4
0100	5
0101	6
0110	7
0111	8
1000	9
1001	10
1010	11
1011	12
1100	13
1101	14
1110	15
1111	16

Bits 7–4 SEQ2-State3 – SEQ2-State0

Reflects the state of SEQ2 sequencer at any point of time. If need be, user can poll these bits to read “interim” results before an EOS. SEQ2 is irrelevant in cascaded mode.

Bits 3–0 SEQ1-State3 – SEQ1-State0

Reflects the state of SEQ1 sequencer at any point of time. If need be, user can poll these bits to read “interim” results before an EOS.

7.6.5 ADC Input Channel Select Sequencing Control Registers

Figure 7–11. ADC Input Channel Select Sequencing Control Registers (CHSELSEQn)

	Bits 15–12	**Bits 11–8**	**Bits 7–4**	**Bits 3–0**	
70A3h	CONV03	CONV02	CONV01	CONV00	CHSELSEQ1
	RW-0	RW-0	RW-0	RW-0	

Note: R = Read access, W = Write access, -0 = value after reset

	Bits 15–12	**Bits 11–8**	**Bits 7–4**	**Bits 3–0**	
70A4h	CONV07	CONV06	CONV05	CONV04	CHSELSEQ2
	RW-0	RW-0	RW-0	RW-0	

Note: R = Read access, W = Write access, -0 = value after reset

	Bits 15–12	**Bits 11–8**	**Bits 7–4**	**Bits 3–0**	
70A5h	CONV11	CONV10	CONV09	CONV08	CHSELSEQ3
	RW-0	RW-0	RW-0	RW-0	

Note: R = Read access, W = Write access, -0 = value after reset

	Bits 15–12	**Bits 11–8**	**Bits 7–4**	**Bits 3–0**	
70A6h	CONV15	CONV14	CONV13	CONV12	CHSELSEQ4
	RW-0	RW-0	RW-0	RW-0	

Note: R = Read access, W = Write access, -0 = value after reset

Each of the 4-bit fields, CONVnn, selects one of the sixteen muxed analog input ADC channels for an autosequenced conversion.

Table 7–6. CONVnn Bit Values and the ADC Input Channels Selected

CONVnn Value	ADC Input Channel Selected
0000	Channel 0
0001	Channel 1
0010	Channel 2
0011	Channel 3
0100	Channel 4
0101	Channel 5
0110	Channel 6
0111	Channel 7
1000	Channel 8
1001	Channel 9
1010	Channel 10
1011	Channel 11
1100	Channel 12
1101	Channel 13
1110	Channel 14
1111	Channel 15

7.6.6 ADC Conversion Result Buffer Registers (for Dual-Sequencer Mode)

RESULT15	8th Conv (Seq 2)	16th Conv

Note: In the cascaded sequencer mode, registers RESULT8 through RESULT15 will hold the results of the ninth through fifteenth conversions.

Figure 7–12. ADC Conversion Result Buffer Registers

15	14	13	12	11	10	9	8
D9	D8	D7	D6	D5	D4	D3	D2

7	6	5	4	3	2	1	0
D1	D0	0	0	0	0	0	0

Notes: 1) Buffer addresses = 70A8h to 70B7h (i.e., 16 registers)

2) The 10-bit conversion result (D9–D0) is left-justified.

7.7 ADC Conversion Clock Cycles

The conversion time is a function of the number of conversions performed in a given sequence. The conversion cycle can be divided into five phases:

❑ Start of Sequence Sync-up (SOS_synch)

The SOS_synch applies only to the first conversion in a sequence.

❑ Acquisition time (ACQ)

❑ Conversion time (CONV)

❑ End of Conversion cycle (EOC)

The ACQ, CONV, and EOC apply to all conversions in a sequence

❑ End of Sequence flag-setting cycle (EOS)

The EOS applies only to the last conversion in a sequence.

Each category is listed in Table 7–7 with the number of CLKOUT cycles it takes to complete.

Table 7–7. ADC Conversion Phases vs CLKOUT cycles

Conversion phase	**CLKOUT cycles (CPS = 0)**	**CLKOUT cycles (CPS = 1)**
SOS_synch	2	2 or 3†
ACQ	2‡	4‡
CONV	10	20
EOC	1	2
EOS	1	1

† When CPS = 1, a “Start of Sequence” can take an extra CLKOUT cycle to sync up with the ADC clock (ADCCLK) depending on which cycle the SOC bit is set in software.

‡ The ACQ value is dependent on the ACQ_PSn bits. Values shown in Table 7–7 are applicable when ACQ_PS = 0. As an example, values for ACQ when ACQ_PS = 1, 2, and 3 are shown in Table 7–8. This table can be extrapolated for all ACQ_PS values.

Table 7–8. ACQ Values When ACQ_PS = 1, 2, and 3

ACQ_PS	**(CPS = 0)**	**(CPS = 1)**
1	ACQ = 4	ACQ = 8
2	ACQ = 6	ACQ = 12
3	ACQ = 8	ACQ = 16

Example 7–4. Calculating the Conversion Time for a Multiple Conversion Sequence With CPS = 0 and ACQ = 0:

1st conversion – 15 CLKOUT cycles

2nd conversion – 13 CLKOUT cycles

3rd conversion – 13 CLKOUT cycles

Last conversion – 14 CLKOUT cycles.

Example 7–5. Calculating the Conversion Time for a Single Conversion Sequence With CPS = 1 and ACQ = 1:

1st and only conversion – 33 or 34 CLKOUT cycles.

Chapter 8

Serial Communications Interface (SCI)

This chapter describes the architecture, functions, and programming of the serial communications interface (SCI) module. All registers in this peripheral are eight bits wide.

The programmable SCI supports asynchronous serial (UART) digital communications between the CPU and other asynchronous peripherals that use the standard NRZ (non-return-to-zero) format. The SCI's receiver and transmitter are double buffered, and each has its own separate enable and interrupt bits. Both may be operated independently or simultaneously in the full-duplex mode.

To ensure data integrity, the SCI checks received data for break detection, parity, overrun, and framing errors. The bit rate (baud) is programmable to over 65,000 different speeds through a 16-bit baud-select register.

For convenience, references to a bit in a register are abbreviated using the register name followed by a period and the number of the bit. For example, the notation for bit 6 of SCI priority control register (SCIPRI) is SCIPRI.6.

Topic **Page**

8.1 Differences vs. ’C240 SCI

Multiplexing the SCI pins with general-purpose I/O is controlled by bits in the digital I/O peripheral. As a consequence, the register SCIPC2 (705Eh) has been removed.

The CLKENA bit in SCICTL1 (7051h) has been removed, since it served no purpose in 2-pin SCI implementations.

The function of the SCIENA bit in SCICCR (7050h) has changed, and is now a LOOP BACK ENA test mode bit. The enable function is no longer required for correct operation of the SCI.

There is no difference with respect to ’241/’242/’243 SCI.

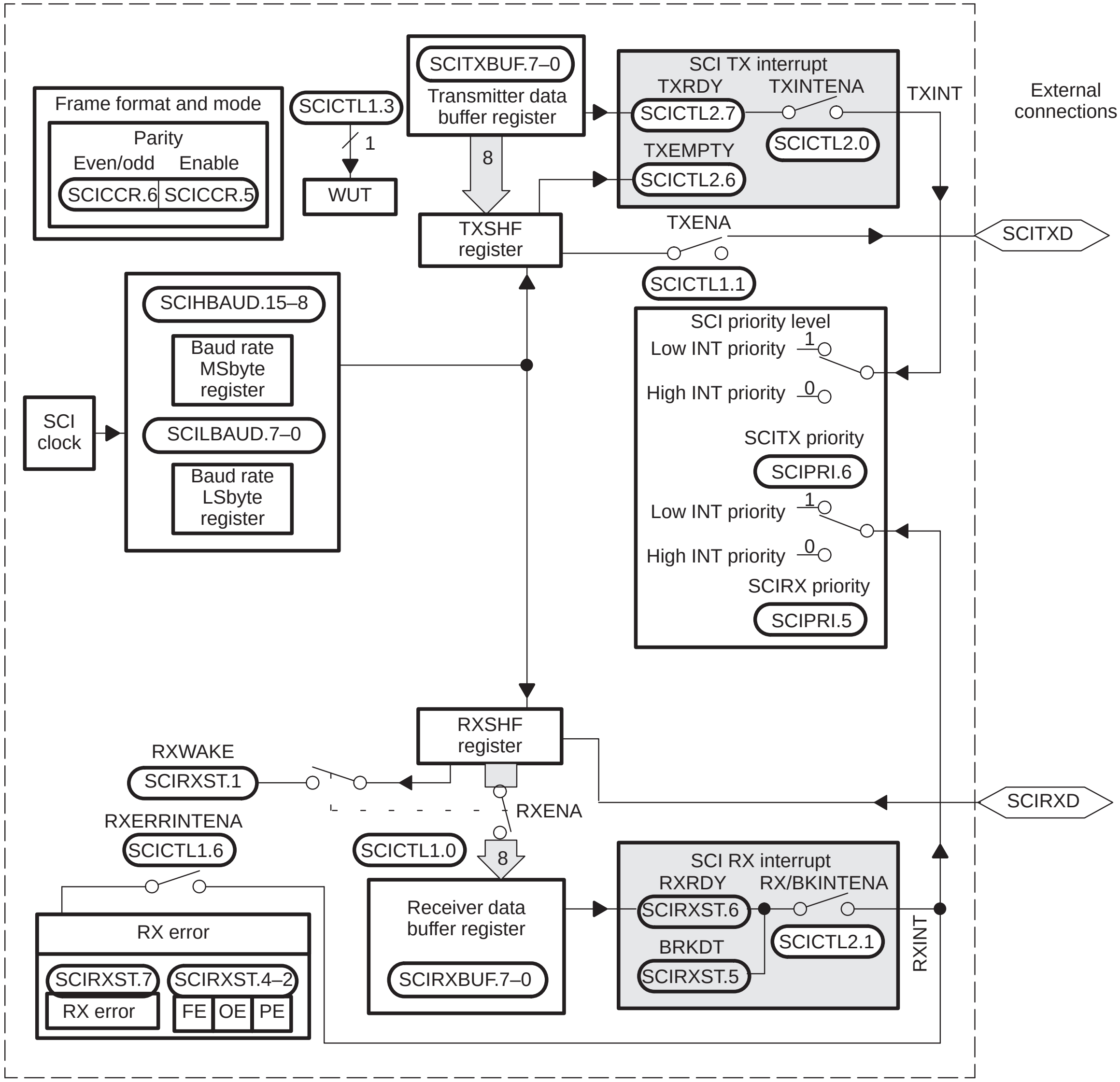
8.1.1 SCI Physical Description

The SCI module, shown in Figure 8–1, has the following key features:

- Two I/O pins
  - SCIRXD (SCI receive data input)
  - SCITXD (SCI transmit data output)
- Programmable bit rates to over 65,000 different speeds through a 16-bit baud select register
  - Range with 30-MHz CLKOUT: 57.2 bps to 1875 kbps
  - Number of bit rates: 64K
- Programmable data word length from one to eight bits
- Programmable stop bits of either one or two bits
- Internally generated serial clock
- Four error detection flags
  - Parity error
  - Overrun error
  - Framing error
  - Break detect

- Two wake-up multiprocessor modes that can be used with either communications format
  - Idle-line wake up
  - Address-bit wake up
- Half- or full-duplex operation
- Double-buffered receive and transmit functions
- Transmitter and receiver can be operated by interrupts or by polling using status flags:
  - Transmitter: TXRDY flag (transmitter buffer register is ready to receive another character from the CPU core) and TX EMPTY flag (transmit shift register is empty)
  - Receiver: RXRDY flag (receive buffer register ready to receive another character from the external world), BRKDT flag (break condition occurred), and RX ERROR monitoring four interrupt conditions
- Separate enable bits for transmitter and receiver interrupts (except break)
- NRZ (non-return-to-zero) format

Figure 8–1. SCI Block Diagram



8.1.2 Architecture

The major elements used in full duplex are shown in Figure 8–1, *SCI Block Diagram* and include:

- ❑ A transmitter (TX) and its major registers (upper half of Figure 8–1)
  - ■ SCITXBUF — transmitter data buffer register. Contains data (loaded by the CPU) to be transmitted
  - ■ TXSHF register — transmitter shift register. Loads data from register SCITXBUF and shifts data onto the SCITXD pin, one bit at a time
- ❑ A receiver (RX) and its major registers (lower half of Figure 8–1)
  - ■ RXSHF register — receiver shift register. Shifts data in from SCIRXD pin, one bit at a time
  - ■ SCIRXBUF — receiver data buffer register. Contains data to be read by the CPU. Data from a remote processor is loaded into register RXSHF and then into registers SCIRXBUF and SCIRXEMU
- ❑ A programmable baud generator
- ❑ Data-memory-mapped control and status registers

The SCI receiver and transmitter can operate either independently or simultaneously.

8.1.3 SCI Module Registers

Table 8–1. Addresses of SCI Registers

				Described In	
Address	**Symbol**	**Name**	**Description**	**Section**	**Page**
7050h	SCICCR	SCI communication control register	Defines the character format, protocol, and communications mode used by the SCI.	8.6.1	8-20
7051h	SCICTL1	SCI control register 1	Controls the RX/TX and receiver error interrupt enable, TXWAKE and SLEEP functions, and the SCI software reset.	8.6.2	8-22
7052h	SCIHBAUD	SCI baud register, high bits	Stores the data (MSbyte) required to generate the bit rate.	8.6.3	8-25
7053h	SCILBAUD	SCI baud register, low bits	Stores the data (LSbyte) required to generate the bit rate.	8.6.3	8-25
7054h	SCICTL2	SCI control register 2	Contains the transmitter interrupt enable, the receiver-buffer/break interrupt enable, the transmitter ready flag, and the transmitter empty flag.	8.6.4	8-26
7055h	SCIRXST	SCI receiver status register	Contains seven receiver status flags.	8.6.5	8-27
7056h	SCIRXEMU	SCI emulation data buffer register	Contains data received for screen updates, principally used by the emulator. (Not a real register – just an alternate address for reading SCIRXEMU without clearing RXRDY)	8.6.6.1	8-29
7057h	SCIRXBUF	SCI receiver data buffer register	Contains the current data from the receiver shift register.	8.6.6.2	8-30
7058h	—	Reserved	Reserved		
7059h	SCITXBUF	SCI transmit data buffer register	Stores data bits to be transmitted by the SCITX.	8.6.7	8-30
705Ah	—	Reserved	Reserved		
705Bh	—	Reserved	Reserved		
705Ch	—	Reserved	Reserved		
705Dh	—	Reserved	Reserved		
705Eh	—	Reserved	Reserved		
705Fh	SCIPRI	SCI priority control register	Contains the receiver and transmitter interrupt priority select bits and the emulator suspend enable bit.	8.6.8	8-31

8.1.4 Multiprocessor and Asynchronous Communication Modes

The SCI has two multiprocessor protocols, the *idle-line* multiprocessor mode (see section 8.3.1 on page 8-10) and the *address-bit* multiprocessor mode (see section 8.3.2 on page 8-12). These protocols allow efficient data transfer between multiple processors.

The SCI offers the universal asynchronous receiver/transmitter (UART) communications mode for interfacing with many popular peripherals. The asynchronous mode (see section 8.4, on page 8-14) requires two lines to interface with many standard devices such as terminals and printers that use RS-232-C formats. Data transmission characteristics include:

- one start bit
- one to eight data bits
- an even/odd parity bit or no parity bit
- one or two stop bits

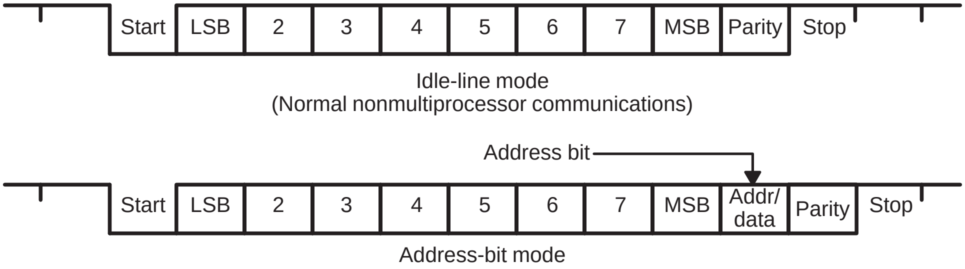
8.2 SCI Programmable Data Format

SCI data, both receive and transmit, is in NRZ (nonreturn-to-zero) format. The NRZ data format, shown in Figure 8–2, consists of:

- ❏ one start bit
- ❏ one to eight data bits
- ❏ an even/odd parity bit (optional)
- ❏ one or two stop bits
- ❏ an extra bit to distinguish addresses from data (address-bit mode only)

The basic unit of data is called a character and is one to eight bits in length. Each character of data is formatted with a start bit, one or two stop bits, and optional parity and address bits. A character of data with its formatting information is called a frame and is shown in Figure 8–2.

Figure 8–2. Typical SCI Data Frame Formats



To program the data format, use the SCICCR register. The bits used to program the data format are shown in Table 8–2.

Table 8–2. Programming the Data Format Using SCICCR

Bit Name	Designation	Functions
SCI CHAR2–0	SCICCR.2–0	Select the character (data) length (one to eight bits). Bit values are shown in Table 8–4 (page 8-21).
PARITY ENABLE	SCICCR.5	Enables the parity function if set to 1, or disables the parity function if cleared to 0.
EVEN/ODD PARITY	SCICCR.6	If parity is enabled, selects odd parity if cleared to 0 or even parity if set to 1.
STOP BITS	SCICCR.7	Determines the number of stop bits transmitted—one stop bit if cleared to 0 or two stop bits if set to 1.

8.3 SCI Multiprocessor Communication

The multiprocessor communication format allows one processor to efficiently send blocks of data to other processors on the same serial link. On one serial line, there should be only one transfer at a time. In other words, there can be only one talker on a serial line at a time.

The *first byte* of a block of information that the talker sends contains an *address byte* that is read by all listeners. Only listeners with the correct address can be interrupted by the data bytes that follow the address byte. The listeners with an incorrect address remain uninterrupted until the next address byte.

All processors on the serial link set their SCI's SLEEP bit (SCICTL1.2) to 1 so that they are interrupted only when the address byte is detected. When a processor reads a block address that corresponds to the CPU's device address as set by your application software, your program must clear the SLEEP bit to enable the SCI to generate an interrupt on receipt of each data byte.

Although the receiver still operates when the SLEEP bit is 1, it does not set RXRDY, RXINT, or any of the receive error status bits to 1 unless the address byte is detected and the address bit in the received frame is a 1 (applicable to addressed-bit mode). The SCI does not alter the SLEEP bit; your software must alter the SLEEP bit.

A processor recognizes an address byte differently, depending on the multiprocessor mode used. For example:

❑ The *idle-line mode* (section 8.3.1 on page 8-10) leaves a quiet space before the address byte. This mode does not have an extra address/data bit and is more efficient than the address-bit mode for handling blocks that contain more than ten bytes of data. The idle-line mode should be used for typical non-multiprocessor SCI communication.

❑ The *address-bit mode* (section 8.3.2 on page 8-12) adds an extra bit (that is, an address bit) into every byte to distinguish addresses from data. This mode is more efficient in handling many small blocks of data because, unlike the idle mode, it does not have to wait between blocks of data. However, at high transmit speed, the program is not fast enough to avoid a 10-bit idle in the transmission stream.

The multiprocessor mode is software selectable via the ADDR/IDLE MODE bit (SCICCR.3). Both modes use the TXWAKE flag bit (SCICTL1.3), RXWAKE flag bit (SCIRXST.1), and the SLEEP flag bits (SCICTL1.2) to control the SCI transmitter and receiver features of these modes.

In both multiprocessor modes, the receipt sequence is:

1) At the receipt of an address block, the SCI port wakes up and requests an interrupt (bit RX/BK INT ENA-SCICTL2.1 must be enabled to request an interrupt). It reads the first frame of the block, which contains the destination address.

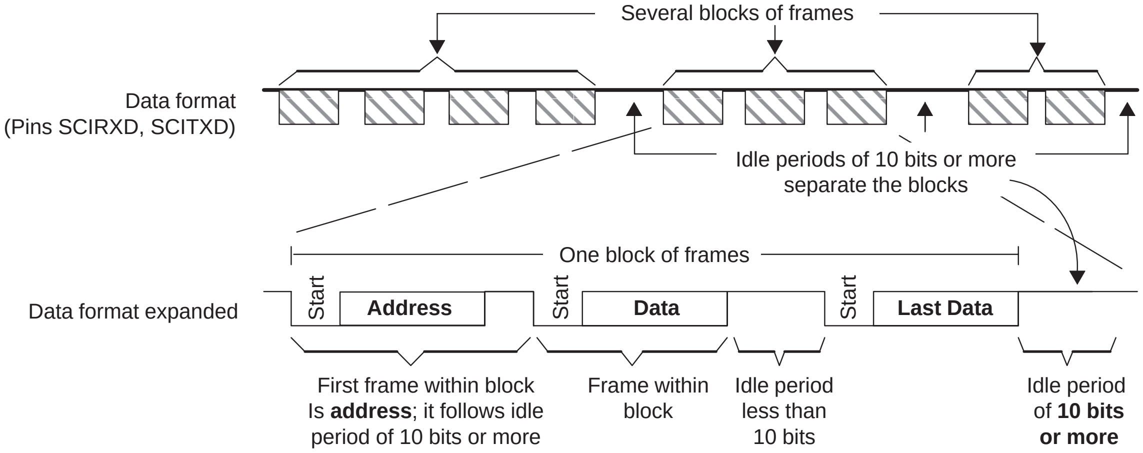
2) A software routine is entered through the interrupt and checks the incoming address. This address byte is checked against its device address byte stored in memory.

3) If the check shows that the block is addressed to the device CPU, the CPU clears the SLEEP bit and reads the rest of the block; if not, the software routine exits with the SLEEP bit still set and does not receive interrupts until the next block start.

8.3.1 Idle-Line Multiprocessor Mode

In the Idle-line multiprocessor protocol (ADDR/IDLE MODE bit=0), blocks are separated by having a longer idle time between the blocks than between frames in the blocks. An idle time of ten or more high-level bits after a frame indicates the start of a new block. The time of a single bit is calculated directly from the baud value (bits per second). The idle-line multiprocessor communication format is shown in Figure 8–3 (ADDR/IDLE MODE bit is SCICCR.3).

Figure 8–3. Idle-Line Multiprocessor Communication Format



The steps followed by the idle-line mode:

1) SCI wakes up after receipt of the block-start signal.

2) The processor now recognizes the next SCI interrupt.

3) The service routine compares the received address (sent by a remote transmitter) to its own.

4) If the CPU *is being addressed*, the service routine clears the SLEEP bit and receives the rest of the data block.

5) If the CPU *is not being addressed*, the SLEEP bit remains set. This lets the CPU continue to execute its main program without being interrupted by the SCI port until the next detection of a block start.

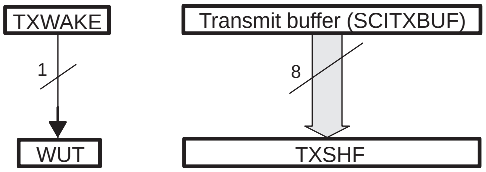
There are two ways to send a block start signal:

❑ Method 1: Deliberately leave an idle time of ten bits or more by delaying the time between the transmission of the last frame of data in the previous block and the transmission of the address frame of the new block.

❑ Method 2: The SCI port first sets the TXWAKE bit (SCICTL1.3) to 1 before writing to the SCITXBUF register. This sends an idle time of exactly 11 bits. In this method, the serial communications line is not idle any longer than necessary. (A don't care byte has to be written to SCITXBUF after setting TXWAKE, and before sending the address, so as to transmit the idle time.)

Associated with the TXWAKE bit is the wake-up temporary (WUT) flag. WUT is an internal flag, double-buffered with TXWAKE. When TXSHF is loaded from SCITXBUF, WUT is loaded from TXWAKE, and the TXWAKE bit is cleared to 0. This arrangement is shown in Figure 8–4. (Figure 8–1, *SCI Block Diagram* on page 8-4 shows this in additional detail.)

Figure 8–4. Double-Buffered WUT and TXSHF



Note: WUT = wake up temporary

To send out a block start signal of exactly one frame time during a sequence of block transmissions:

1) Write a 1 to the TXWAKE bit.

2) Write a data word (content not important: a *don't care*) to the SCITXBUF register (transmit data buffer) to send a block-start signal. (The first data word written is suppressed while the block-start signal is sent out and ignored after that.) When the TXSHF (transmit shift register) is free again, SCITXBUF's contents are shifted to TXSHF, the TXWAKE value is shifted to WUT, and then TXWAKE is cleared.

   Because TXWAKE was set to a 1, the start, data, and parity bits are replaced by an idle period of 11 bits transmitted following the last stop bit of the previous frame.

3) Write a new address value to SCITXBUF.

A *don't-care* data word must first be written to register SCITXBUF so that the TXWAKE bit value can be shifted to WUT. After the don't-care data word is shifted to the TXSHF register, the SCITXBUF (and TXWAKE if necessary) can be written to again because TXSHF and WUT are both double-buffered.

The receiver operates, regardless of the SLEEP bit. However, the receiver neither sets RXRDY nor the error status bits, nor does it request a receive interrupt *until an address frame is detected*.

8.3.2 Address-Bit Multiprocessor Mode

In the address-bit protocol (ADDR/IDLE MODE bit=1), frames have an extra bit, called an address bit, that immediately follows the last data bit. The address bit is set to 1 in the first frame of the block and to 0 in all other frames. The idle period timing is irrelevant (see Figure 8–5, ADDR/IDLE MODE bit in SCICCR.3).

The TXWAKE bit value is placed in the address bit. During transmission, when the SCITXBUF register and TXWAKE are loaded into the TXSHF register and WUT respectively, TXWAKE is reset to 0 and WUT becomes the value of the address bit of the current frame. Thus, to send an address:

1) Set the TXWAKE bit to 1 and write the appropriate address value to the SCITXBUF register.

2) When this address value is transferred to the TXSHF register and shifted out, its address bit is sent as a 1, which flags the other processors on the serial link to read the address.

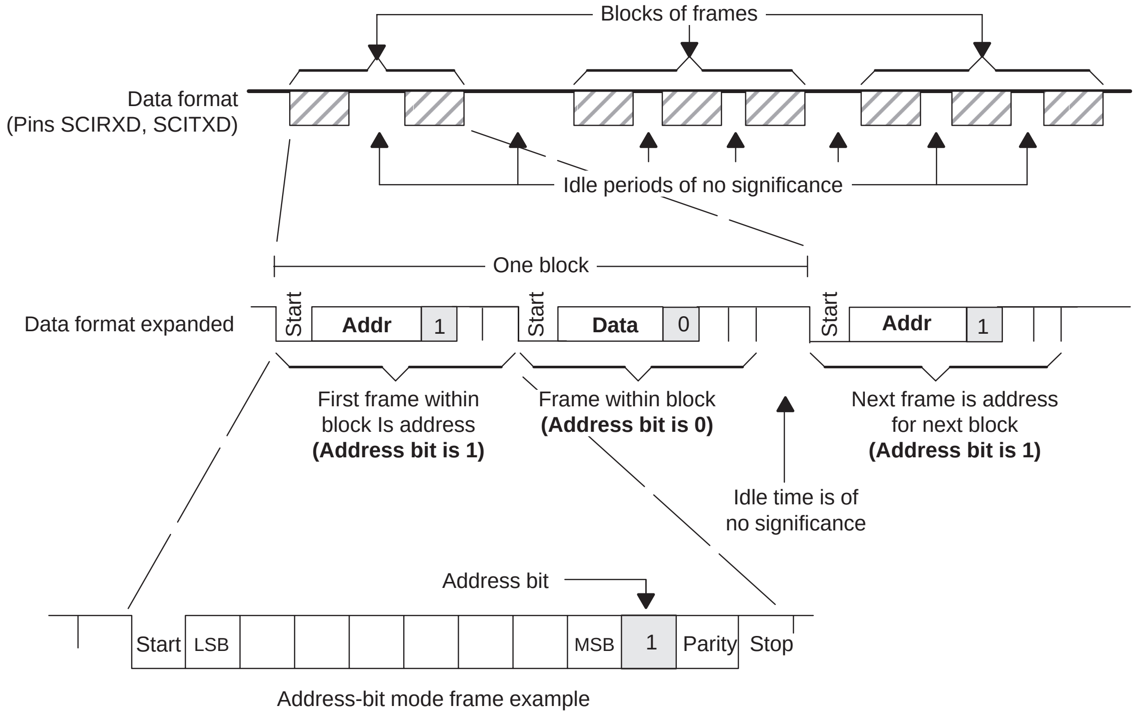
3) Since TXSHF and WUT are both double-buffered, SCITXBUF and TXWAKE can be written to immediately after TXSHF and WUT are loaded.

4) To transmit non-address frames in the block, leave the TXWAKE bit set to 0.

Note: Address-bit format for transfers of 11 bytes or less

As a general rule, the address-bit format is typically used for data frames of 11 bytes or less. This format adds one bit value (1 for an address frame, 0 for a data frame) to all data bytes transmitted. The idle-line format is typically used for data frames of 12 bytes or more.

Figure 8–5. Address-Bit Multiprocessor Communication Format



8.4 SCI Communication Format

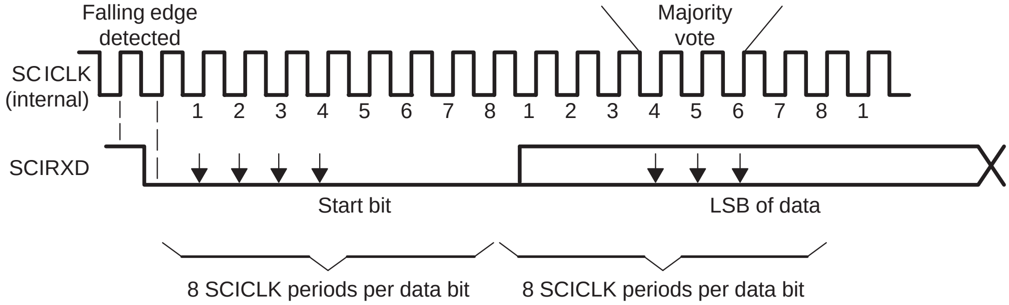
The SCI asynchronous communication format uses either single line (one way) or two line (two way) communications. In this mode, the frame consists of a start bit, one to eight data bits, an optional even/odd parity bit, and one or two stop bits (shown in Figure 8–6). There are *eight SCICLK periods* per data bit.

The receiver begins operation on receipt of a valid start bit. A valid start bit is identified by four consecutive internal SCICLK periods of zero bits as shown in Figure 8–6. If any bit is not zero, then the processor starts over and begins looking for another start bit.

For the bits following the start bit, the processor determines the bit value by making three samples in the middle of the bits. These samples occur on the fourth, fifth, and sixth SCICLK periods, and bit-value determination is on a majority (two out of three) basis. Figure 8–6 illustrates the asynchronous communication format for this with a start bit showing how edges are found and where a majority vote is taken.

Since the receiver synchronizes itself to frames, the external transmitting and receiving devices do not have to use a synchronized serial clock. The clock can be generated locally.

Figure 8–6. SCI Asynchronous Communications Format

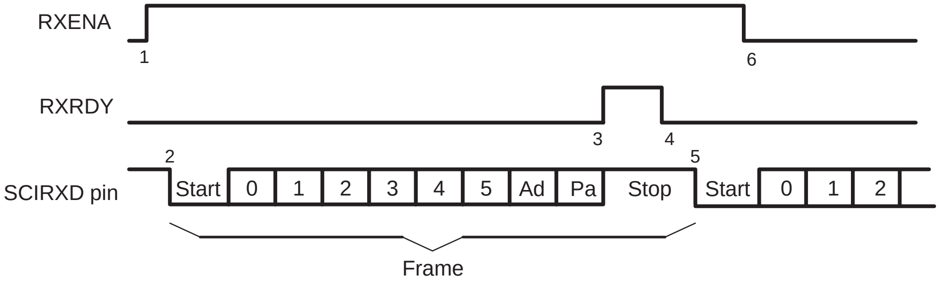


8.4.1 Receiver Signals in Communication Modes

Figure 8–7 illustrates an example of receiver signal timing that assumes the following conditions:

- ❑ Address-bit wake-up mode (address bit does not appear in idle-line mode)
- ❑ Six bits per character

Figure 8–7. SCI RX Signals in Communication Modes



Notes:
1) Flag bit RXENA (SCICTL1.0) goes high to enable the receiver.
2) Data arrives on the SCIRXD pin, start bit detected.
3) Data is shifted from RXSHF to the receive buffer register (SCIRXBUF); an interrupt is requested. Flag bit RXRDY (SCIRXST.6) goes high to signal that a new character has been received.
4) The program reads SCIRXBUF; flag RXRDY is automatically cleared.
5) The next byte of data arrives on the SCIRXD pin; the start bit is detected, then cleared.
6) Bit RXENA is brought low to disable the receiver. Data continues to be assembled in RXSHF but is not transferred to the receive buffer register.

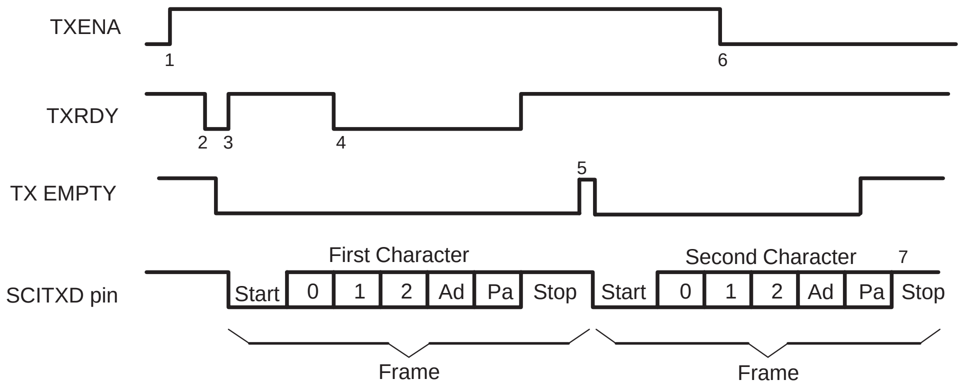
8.4.2 Transmitter Signals in Communication Modes

Figure 8–8 illustrates an example of transmitter signal timing that assumes the following conditions:

❑ Address-bit wake-up mode (address bit does not appear in idle-line mode)

❑ Three bits per character

Figure 8–8. SCI TX Signals in Communications Modes



Notes:

1) Bit TXENA (SCICTL1.1) goes high, enabling the transmitter to send data.
2) SCITXBUF is written to; thus, (1) the transmitter is no longer empty, and (2) TXRDY goes low.
3) The SCI transfers data to the shift register (TXSHF). The transmitter is ready for a second character (TXRDY goes high), and it requests an interrupt (to enable an interrupt, bit TX INT ENA — SCICTL2.0 — must be set).
4) The program writes a second character to SCITXBUF after TXRDY goes high (item 3). (TXRDY goes low again after the second character is written to SCITXBUF.)
5) Transmission of the first character is complete. TX EMPTY goes high temporarily. Transfer of the second character to shift register TXSHF begins.
6) Bit TXENA goes low to disable the transmitter; the SCI finishes transmitting the current character.
7) Transmission of the second character is complete; transmitter is empty and ready for new character.

8.5 SCI Port Interrupts

The internally-generated serial clock is determined by the device clock frequency and the baud-select registers. The SCI uses the 16-bit value of the baud-select registers to select one of 64k different serial clock rates.

The SCI's receiver and transmitter can be interrupt controlled. The SCICTL2 register has one flag bit (TXRDY) that indicates active interrupt conditions, and the SCIRXST register has two interrupt flag bits (RXRDY and BRKDT), plus the RX ERROR interrupt flag which is a logical OR of the FE, OE & PE conditions. The transmitter and receiver have separate interrupt-enable bits. When not enabled, the interrupts are not asserted; however, the condition flags remain active, reflecting transmission and receipt status.

The SCI has independent peripheral interrupt vectors for the receiver and transmitter. Peripheral interrupt requests can be either high priority or low priority. This is indicated by the priority bits which are output from the peripheral to the PIE controller. SCI interrupts can be programmed to assert the high- or low-priority levels by the SCIRX PRIORITY (SCIPRI.5) and SCITX PRIORITY (SCIPRI.6) control bits. When both RX and TX interrupt requests are made at the same priority level, the receiver always has higher priority than the transmitter, reducing the possibility of receiver overrun.

The operation of peripheral interrupts is described in the Peripheral Interrupt Expansion controller chapter of the device specification of which this SCI chapter is a part.

❑ If the RX/BK INT ENA bit (SCICTL2.1) is set, the receiver peripheral interrupt request is asserted when one of the following events occurs:

  ■ The SCI receives a complete frame and transfers the data in the RXSHF register to the SCIRXBUF register. This action sets the RXRDY flag (SCIRXST.6) and initiates an interrupt.

  ■ A break detect condition occurs (the SCIRXD is low for ten bit periods following a missing stop bit). This action sets the BRKDT flag bit (SCIRXST.5) and initiates an interrupt.

❑ If the TX INT ENA bit (SCICTL2.0) is set, the transmitter peripheral interrupt request is asserted whenever the data in the SCITXBUF register is transferred to the TXSHF register, indicating that the CPU can write to the TXBUF; this action sets the TXRDY flag bit (SCICTL2.7) and initiates an interrupt.

Note:

Interrupt generation due to RXRDY and BRKDT bits are controlled by RX/BK_INT_ENA bit (SCICTL2.1). Interrupt generation due to RX_ERROR bit is controlled by RX_ERR_INT_ENA bit (SCICTL1.6).

8.5.1 SCI Baud Rate Calculation

The internally generated serial clock is determined by the device clock frequency (CLKOUT) and the baud rate select registers. The SCI uses the 16-bit value of the baud select registers to select one of the 64K different serial clock rates possible for a given device clock.

See the bit descriptions in section 8.6.3, *Baud-Select Registers*, for the formula to use to calculate the SCI Asynchronous Baud.

Table 8–3. Asynchronous Baud Register Values for Common SCI Bit Rates

	Device Clock Frequency, 30 MHz		
Ideal Baud	**BRR**	**Actual Baud**	**% Error**
2400	1562	2399	–0.03
4800	780	4802	0.03
8192	457	8188	–0.05
9600	390	9591	– 0.10
19200	194	19230	0.16
38400	97	38265	–0.35

Note: The maximum CLKOUT frequency for ’240x devices is 30 MHz.

8.6 SCI Module Registers

The functions of the SCI are software configurable. Sets of control bits, organized into dedicated bytes, are programmed to initialize the desired SCI communications format. This includes operating mode and protocol, baud value, character length, even/odd parity or no parity, number of stop bits, and interrupt priorities and enables. The SCI is controlled and accessed through registers listed in Figure 8–9, and described in the sections that follow.

Figure 8–9. SCI Control Registers

Address	**Register mnemonic**	**Bit Number**								**Register name**
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
7050h	SCICCR	STOP BITS	EVEN/ ODD PARITY	PARITY ENABLE	LOOP BACK ENA	ADDR/ IDLE MODE	SCI CHAR2	SCI CHAR1	SCI CHAR0	Communication control
7051h	SCICTL1	Reserved	RX ERR INT ENA	SW RESET	Reserved	TXWAKE	SLEEP	TXENA	RXENA	SCI control reg.1
7052h	SCIHBAUD	BAUD15 (MSB)	BAUD14	BAUD13	BAUD12	BAUD11	BAUD10	BAUD9	BAUD8	Baud rate (MSbyte)
7053h	SCILBAUD	BAUD7	BAUD6	BAUD5	BAUD4	BAUD3	BAUD2	BAUD1	BAUD0 (LSB)	Baud rate (LSbyte)
7054h	SCICTL2	TXRDY	TX EMPTY	Reserved				RX/BK INTENA	TX INTENA	SCI control reg.2
7055h	SCIRXST	RX ERROR	RXRDY	BRKDT	FE	OE	PE	RXWAKE	Reserved	Receiver status
7056h	SCIRXEMU	ERXDT7	ERXDT6	ERXDT5	ERXDT4	ERXDT3	ERXDT2	ERXDT1	ERXDT0	EMU data buffer
7057h	SCIRXBUF	RXDT7	RXDT6	RXDT5	RXDT4	RXDT3	RXDT2	RXDT1	RXDT0	Receiver data buffer
7058h	——	Reserved								——
7059h	SCITXBUF	TXDT7	TXDT7	TXDT5	TXDT4	TXDT3	TXDT2	TXDT1	TXDT0	Transmit data buffer
705Ah	——	Reserved								——
705Bh	——	Reserved								——
705Ch	——	Reserved								——
705Dh	——	Reserved								——
705Eh	——	Reserved								——
705Fh	SCIPRI	Reserved	SCITX PRIORITY	SCIRX PRIORITY	SCI SUSP SOFT	SCI SUSP FREE	Reserved			Priority/ emulation control

8.6.1 SCI Communication Control Register

The SCI communication control (SCICCR) register defines the character format, protocol, and communications mode used by the SCI.

Figure 8–10. SCI Communication Control Register (SCICCR) — Address 7050h

7	6	5	4	3	2	1	0
STOP BITS	EVEN/ODD PARITY	PARITY ENABLE	LOOPBACK ENA	ADDR/IDLE MODE	SCICHAR2	SCICHAR1	SCICHAR0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 7 **STOP BITS**. SCI number of stop bits.

This bit specifies the number of stop bits transmitted. The receiver checks for only one stop bit.

0	One stop bit
1	Two stop bits

Bit 6 **PARITY**. SCI parity odd/even selection.

If the PARITY ENABLE bit (SCICCR.5) is set, PARITY (bit 6) designates odd or even parity (odd or even number of bits with the value of 1 in both transmitted and received characters).

0	Odd parity
1	Even parity

Bit 5 **PARITY ENABLE**. SCI parity enable.

This bit enables or disables the parity function. If the SCI is in the address-bit multiprocessor mode (set using bit 3 of this register), the address bit is included in the parity calculation (if parity is enabled). For characters of less than eight bits, the remaining unused bits should be masked out of the parity calculation.

0	Parity disabled; no parity bit is generated during transmission or is expected during reception
1	Parity is enabled

Bit 4 **LOOP BACK ENA**. Loop Back test mode enable.

This bit enables the Loop Back test mode where the Tx pin is internally connected to the Rx pin.

0	Loop Back test mode disabled
1	Loop Back test mode enabled

Bit 3 **ADDR/IDLE MODE**. SCI multiprocessor mode control bit.

This bit selects one of the multiprocessor protocols

0	Idle-line mode protocol selected
1	Address-bit mode protocol selected

Multiprocessor communication is different from the other communication modes because it uses SLEEP and TXWAKE functions (bits SCICTL1.2 and SCICTL1.3, respectively). The idle-line mode is usually used for normal communications because the address-bit mode adds an extra bit to the frame. The Idle-line mode does not add this extra bit and is compatible with RS-232 type communications.

Bits 2–0 **SCI CHAR2–0**. Character-length control bits 2 - 0.

These bits select the SCI character length from one to eight bits. Characters of less than eight bits are right-justified in SCIRXBUF and SCIRXEMU and are padded with leading zeros in SCIRXBUF. SCITXBUF *doesn't need to be padded with leading zeros.* Table 8–4 lists the bit values and character lengths for SCI CHAR2-0 bits.

Table 8–4. SCI CHAR2–0 Bit Values and Character Lengths

SCI CHAR2–0 Bit Values (Binary)			
SCI CHAR2	**SCI CHAR1**	**SCI CHAR0**	**Character Length (Bits)**
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

8.6.2 SCI Control Register 1

The SCI control register 1 controls the receiver/transmitter enable, TXWAKE and SLEEP functions, and the SCI software reset.

Figure 8–11. SCI Control Register 1 (SCICTL1) — Address 7051h

7	6	5	4	3	2	1	0
Reserved	RX ERR INT ENA	SW RESET	Reserved	TXWAKE	SLEEP	TXENA	RXENA
R-0	RW-0	RW-0	R-0	RS-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, S = Set only, -0 = value after reset

Bit 7 **Reserved**. Reads return zero; writes have no effect.

Bit 6 **RX ERR INT ENA**. SCI receiver enable.

Setting this bit enables an interrupt if the RX ERROR bit (SCIRXST.7) becomes set because of errors occurring.

0	Receive error interrupt disabled
1	Receive error interrupt enabled

Bit 5 **SW RESET**. SCI software reset (active low).

Writing a 0 to this bit initializes the SCI state machines and operating flags (registers SCICTL2 and SCIRXST) to the reset condition.

The SW RESET bit does not affect any of the configuration bits.

All affected logic is held in the specified reset state until a 1 is written to SW RESET (the bit values following a reset are shown beneath each register diagram in this section). Thus, after a system reset, re-enable the SCI by writing a 1 to this bit.

Clear this bit after a receiver break detect (BRKDT flag, bit SCIRXST.5).

SW RESET affects the operating flags of the SCI, but it neither affects the configuration bits nor restores the reset values. Table 8–5 lists the affected flags.

Table 8–5. SW RESET-Affected Flags

SCI Flag	Register.Bit	Value After SW RESET
TXRDY	SCICTL2.7	1
TX EMPTY	SCICTL2.6	1
RXWAKE	SCIRXST.1	0
PE	SCIRXST.2	0
OE	SCIRXST.3	0
FE	SCIRXST.4	0
BRKDT	SCIRXST.5	0
RXRDY	SCIRXST.6	0
RX ERROR	SCIRXST.7	0

Once SW RESET is asserted, the flags are frozen until the bit is de-asserted.

Bit 4 **Reserved**. Reads return zero; writes have no effect.

Bit 3 **TXWAKE**. SCI transmitter wakeup method select.

The TXWAKE bit controls selection of the data-transmit feature, depending on which transmit mode (idle line or address bit) is specified at the ADDR/IDLE MODE bit (SCICCR.3)

0 Transmit feature is not selected.

1 Transmit feature selected is dependent on the mode: idle-line or address-bit:

In *idle-line* mode**:** write a 1 to TXWAKE, then write data to register SCITXBUF to generate an idle period of 11 data bits.

In *address-bit* mode**:** write a 1 to TXWAKE, then write data to SCITXBUF to set the address bit for that frame to 1.

TXWAKE is not cleared by the SW RESET bit (SCICTL1.5); it is cleared by a system reset or the transfer of TXWAKE to the WUT flag.

Bit 2 **SLEEP**. SCI sleep.

In a multiprocessor configuration, this bit controls the receive sleep function. Clearing this bit brings the SCI out of the sleep mode.

0 Sleep mode disabled

1 Sleep mode enabled

The receiver still operates when the SLEEP bit is set; however, operation does not update the receive buffer ready bit (SCIRXST.6, RXRDY) or the error status bits (SCIRXST.5–2: BRKDT, FE, OE, and PE) unless the address byte is detected. This bit is *not* cleared when the address byte is detected.

Bit 1 **TXENA**. SCI transmitter enable.

Data is transmitted through the SCITXD pin only when TXENA is set. If reset, transmission is halted but only after all data previously written to SCITXBUF has been sent.

0	Transmitter disabled
1	Transmitter enabled

Bit 0 **RXENA**. SCI receiver enable.

Data is received on the SCIRXD pin and is sent to the receive shift register and then the receive buffers. This bit enables or disables the receiver (transfer to the buffers).

0	Prevent received characters from transfer into the SCIRXEMU and SCIRXBUF receive buffers.
1	Send receive characters into the SCIRXEMU and SCIRXBUF buffers.

Clearing RXENA stops received characters from being transferred into the two receive buffers and also stops the generation of receiver interrupts. However, the receiver shift register can continue to assemble characters. Thus, if RXENA is set during the reception of a character, the complete character will be transferred into the receive buffer registers, SCIRXEMU and SCIRXBUF.

8.6.3 Baud-Select Registers

The values in the baud-select registers (SCIHBAUD and SCILBAUD) specify the baud rate for the SCI.

Figure 8–12. Baud-Select MSbyte Register (SCIHBAUD) — Address 7052h

15	14	13	12	11	10	9	8
BAUD15 (MSB)	BAUD14	BAUD13	BAUD12	BAUD11	BAUD10	BAUD9	BAUD8
RW-0	RW-0	RW-0	RW-0	RS-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, S = Set only, -0 = value after reset

Figure 8–13. Baud-Select LSbyte Register (SCILBAUD) — Address 7053h

7	6	5	4	3	2	1	0
BAUD7	BAUD6	BAUD5	BAUD4	BAUD3	BAUD2	BAUD1	BAUD0 (LSB)
RW-0	RW-0	RW-0	RW-0	RS-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, S = Set only, -0 = value after reset

Bits 15–0 **BAUD15–BAUD0**. SCI 16-bit baud selection.

Registers SCIHBAUD (MSbyte) and SCILBAUD (LSbyte) concatenate to form a 16-bit baud value, BRR.

The internally-generated serial clock is determined by the CLKOUT and the two baud select registers. The SCI uses the 16-bit value of these registers to select one of 64K serial clock rates for the communication modes.

The SCI baud rate is calculated using the following equation:

$$\text{SCI Asynchronous Baud} = \frac{\text{CLKOUT}}{(\text{BRR} + 1) \times 8}$$

Alternatively,

$$\text{BRR} = \frac{\text{CLKOUT}}{\text{SCI Asynchronous Baud} \times 8} - 1$$

Note that the above formulas are applicable only when 1 ≤ BRR ≤ 65535. If BRR = 0, then

$$\text{SCI Asynchronous Baud} = \frac{\text{CLKOUT}}{16}$$

Where: BRR = The 16-bit value (in decimal) in the baud-select registers.

8.6.4 SCI Control Register 2

SCI control register 2 enables the receive-ready, break-detect, and transmit-ready interrupts as well as transmitter-ready and -empty flags.

Figure 8–14. SCI Control Register 2 (SCICTL2) — Address 7054h

7	6	5–2	1	0
TXRDY	TX EMPTY	Reserved	RX/BK INT ENA	TX INT ENA
R-1	R-1	R-0	RW-0	RW-0

Note: R = Read access, W = Write access, -n = value after reset

Bit 7 **TXRDY**. Transmitter buffer-register ready flag.

When set, this bit indicates that the transmit buffer register, SCITXBUF, is ready to receive another character. Writing data to the SCITXBUF automatically clears this bit. When set, this flag asserts a transmitter interrupt request if the interrupt-enable bit TX INT ENA (SCICTL2.0) is also set. TXRDY is set to 1 by enabling the SW RESET bit (SCICTL.2) or by a system reset.

0 SCITXBUF is full.

1 SCITXBUF is ready to receive the next character.

Bit 6 **TX EMPTY**. Transmitter empty flag.

This flag’s value indicates the contents of the transmitter’s buffer register (SCITXBUF) and shift register (TXSHF). An active SW RESET (SCICTL1.2), or a system reset, sets this bit. This bit *does not* cause an interrupt request.

0 Transmitter buffer or shift register or both are loaded with data.

1 Transmitter buffer and shift registers are both empty.

Bits 5–2 **Reserved.**

Reads return zero; writes have no effect.

Bit 1 **RX/BK INT ENA**. Receiver-buffer/break interrupt enable.

This bit controls the interrupt request caused by *either* the RXRDY flag *or* the BRKDT flag (bits SCIRXST.6 and .5) being set. However, RX/BRK INT ENA does not prevent the setting of these flags.

0 Disable RXRDY/BRKDT interrupt.

1 Enable RXRDY/BRKDT interrupt.

Bit 0 **TX INT ENA**. SCITXBUF-register interrupt enable.

This bit controls issuing an interrupt request caused by setting the TXRDY flag bit (SCICTL2.7). However, it doesn’t prevent the TXRDY flag from being set (being set indicates that register SCITXBUF is ready to receive another character).

0 Disable TXRDY interrupt.

1 Enable TXRDY interrupt.

8.6.5 Receiver Status Register

The receiver status (SCIRXST) register contains seven bits that are receiver status flags (two of which can generate interrupt requests). Each time a complete character is transferred to the receive buffers (SCIRXEMU and SCIRXBUF), the status flags are updated. Each time the buffers are read, the flags are cleared. Figure 8–16 on page 8-29 shows the relationships between several of the register's bits.

Figure 8–15. Receiver Status Register (SCIRXST) — Address 7055h

7	6	5	4	3	2	1	0
RX ERROR	RXRDY	BRKDT	FE	OE	PE	RXWAKE	Reserved
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 7 **RX ERROR**. SCI receiver-error flag.

The RX ERROR flag indicates that one of the error flags in the receiver status register is set. RX ERROR is a logical OR of the break detect, framing error, overrun, and parity error enable flags (bits 5–2: BRKDT, FE, OE, and PE).

0 No error flags set.

1 Error flag(s) set.

A 1 on this bit will cause an interrupt if the RX ERR INT ENA bit (SCICTL1.6) is set. This bit can be used for fast error-condition checking during the interrupt service routine. This error flag cannot be cleared directly; it is cleared by an active SW RESET or by a system reset.

Bit 6 **RXRDY**. SCI receiver-ready flag.

When a new character is ready to be read into the SCIRXBUF register, the receiver sets this bit, and a receiver interrupt is generated if the RX/BK INT ENA bit (SCICTL2.1) is a 1. RXRDY is cleared by reading the SCIRXBUF register, by an active SW RESET, or by a system reset.

0 No new character in SCIRXBUF.

1 Character ready to be read from SCIRXBUF.

Bit 5 **BRKDT**. SCI break-detect flag.

The SCI sets this bit when a break condition occurs. A break condition occurs when the SCI receive data line (SCIRXD) remains continuously low for at least ten bits, beginning after a missing first stop bit. The occurrence of a break causes a receiver interrupt to be generated if the RX/BK INT ENA bit is a 1, but it does not cause the receiver buffer to be loaded. A BRKDT interrupt can occur, even if the receiver SLEEP bit is set to 1. BRKDT is cleared by an active SW RESET or by a system reset. It is not cleared by receipt of a character after the break is detected. In order to receive more characters, the SCI must be reset by toggling the SW RESET bit or by a system reset.

0 No break condition.

1 Break condition occurred.

Bit 4 **FE**. SCI framing-error flag.

The SCI sets this bit when an expected stop bit is not found. Only the first stop bit is checked. The missing stop bit indicates that synchronization with the start bit has been lost and that the character is incorrectly framed. It is reset by clearing the SW RESET bit or by a system reset.

0 No framing error detected.

1 Framing error detected.

Bit 3 **OE**. SCI overrun-error flag.

The SCI sets this bit when a character is transferred into registers SCIRXEMU and SCIRXBUF before the previous character is fully read by the CPU or DMAC. The previous character is overwritten and lost. The OE flag is reset by an active SW RESET or by a system reset.

0 No overrun error detected.

1 Overrun error detected.

Bit 2 **PE**. SCI parity-error flag.

This flag bit is set when a character is received with a mismatch between the number of 1s and its parity bit. The address bit is included in the calculation. If parity generation and detection is not enabled, the PE flag is disabled and read as 0. The PE bit is reset by an active SW RESET or a system reset.

0 No parity error **or** parity is disabled.

1 Parity error is detected.

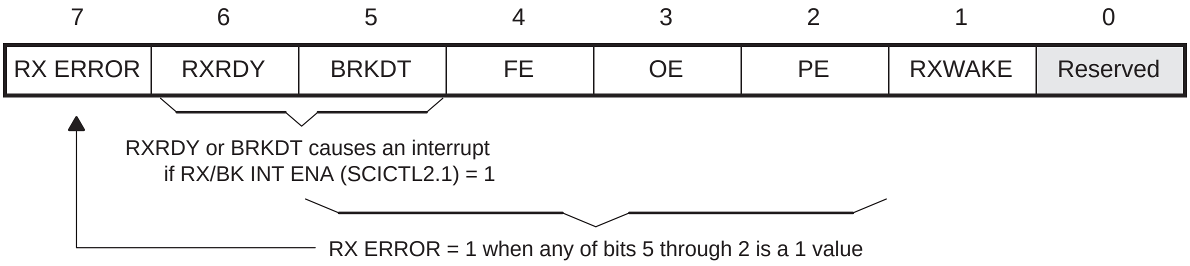
Bit 1 **RXWAKE**. Receiver wakeup-detect flag.

A value of 1 in this bit indicates detection of a receiver wakeup condition. In the address bit multiprocessor mode (SCICCR.3 = 1), RXWAKE reflects the value of the address bit for the character contained in SCIRXBUF. In the idle-line multiprocessor mode, RXWAKE is set if the SCIRXD data line is detected as idle. RXWAKE is a read-only flag, cleared by one of the following:

- ❑ the transfer of the first byte after the address byte to SCIRXBUF
- ❑ the reading of SCIRXBUF
- ❑ an active SW RESET
- ❑ a system reset

Bit 0 **Reserved**. Reads return zero; writes have no effect.

Figure 8–16. Register SCIRXST Bit Associations — Address 7055h



7	6	5	4	3	2	1	0
RX ERROR	RXRDY	BRKDT	FE	OE	PE	RXWAKE	Reserved

RXRDY or BRKDT causes an interrupt
if RX/BK INT ENA (SCICTL2.1) = 1

RX ERROR = 1 when any of bits 5 through 2 is a 1 value

8.6.6 Receiver Data Buffer Registers

Received data is transferred from RXSHF to the SCIRXEMU and SCIRXBUF registers. When the transfer is complete, the RXRDY flag (bit SCIRXST.6) is set, indicating that the received data is ready to be read. Both registers contain the same data; they have separate addresses but are not physically separate buffers. The only difference is that reading SCIRXEMU *does not* clear the RXRDY flag; however, reading SCIRXBUF clears the flag.

8.6.6.1 Emulation Data Buffer

Normal SCI data receive operations read the data received from the SCIRXBUF register (described below). The SCIRXEMU register is used principally by the emulator (EMU) because it can continuously read the data received for screen updates without clearing the RXRDY flag. SCIRXEMU is cleared by a system reset.

This is the register which should be used in an Emulator watch window to view the contents of SCIRXBUF register.

SCIRXEMU is not physically implemented, it is just a different address location to access the SCIRXBUF register without clearing the RXRDY flag.

Figure 8–17. Emulation Data Buffer Register (SCIRXEMU) — Address 7056h

7	6	5	4	3	2	1	0
ERXDT7	ERXDT6	ERXDT5	ERXDT4	ERXDT3	ERXDT2	ERXDT1	ERXDT0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Note: R = Read access, -0 = value after reset

8.6.6.2 Receiver Data Buffer

When the current data received is shifted from RXSHF to the receive buffer, flag bit RXRDY is set and the data is ready to be read. If the RX/BK INT ENA bit (SCICTL2.1) is set, this shift also causes an interrupt. When SCIRXBUF is read, the RXRDY flag is reset. SCIRXBUF is cleared by a system reset.

Figure 8–18. Receiver Data Buffer (SCIRXBUF) — Address 7057h

7	6	5	4	3	2	1	0
RXDT7	RXDT6	RXDT5	RXDT4	RXDT3	RXDT2	RXDT1	RXDT0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Note: R = Read access, -0 = value after reset

8.6.7 Transmit Data Buffer Register

Data bits to be transmitted are written to the transmit data buffer (SCITXBUF) register. The transfer of data from this register to the TXSHF transmitter shift register sets the TXRDY flag (SCICTL2.7), indicating that SCITXBUF is ready to receive another set of data. If bit TX INT ENA (SCICTL2.0) is set, this data transfer also causes an interrupt. These bits must be right-justified because the leftmost bits are ignored for characters less than eight bits long.

Figure 8–19. Transmit Data Buffer Register (SCITXBUF) — Address 7059h

7	6	5	4	3	2	1	0
TXDT7	TXDT6	TXDT5	TXDT4	TXDT3	TXDT2	TXDT1	TXDT0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

8.6.8 Priority Control Register

The Priority Control Register contains the receiver and transmitter interrupt priority select bits and controls the SCT operation on the XDS emulator during program suspends such as hitting a breakpoint.

Figure 8–20. SCI Priority Control Register (SCIPRI) — Address 705Fh

7	6	5	4	3	2–0
Reserved	SCITX PRIORITY	SCIRX PRIORITY	SCI SOFT	SCI FREE	Reserved
R-0	RW-0	RW-0	RW-0	RW-0	R-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 7 **Reserved.** Reads return zero; writes have no effect.

Bit 6 **SCITX PRIORITY**. SCI transmitter interrupt priority select. This bit specifies priority level of the SCI transmitter interrupts.

0	Interrupts are high-priority requests.
1	Interrupts are low-priority requests.

Bit 5 **SCIRX PRIORITY**. SCI receiver interrupt priority select. This bit specifies the priority level to the SCI receiver interrupts.

0	Interrupts are high-priority requests.
1	Interrupts are low-priority requests.

Bits 4,3 **SCI SUSP SOFT & FREE bits.** These bits determine what occurs when an emulation suspend occurs (for example, when the debugger hits a breakpoint). The peripheral can continue whatever it is doing (free-run mode), or if in stop mode, it can either stop immediately or stop when the current operation (the current receive/transmit sequence) is complete.

Bit 4	**Bit 3**	
Soft	**Free**	
0	0	Immediate stop on suspend.
1	0	Complete current receive/transmit sequence before stopping.
X	1	Free run, continue SCI operation regardless of suspend.

Bits 2–0 **Reserved.** Reads return zero; writes have no effect.

Chapter 9

Serial Peripheral Interface (SPI)

The serial peripheral interface (SPI) is a high-speed synchronous serial input/output (I/O) port that allows a serial bit stream of programmed length (one to sixteen bits) to be shifted into and out of the device at a programmed bit-transfer rate. The SPI is normally used for communications between the DSP controller and external peripherals or another controller. Typical applications include external I/O or peripheral expansion via devices such as shift registers, display drivers, and analog-to-digital converters (ADCs).

Most SPI registers are eight bits in width (except for the data registers), a carry-over from the 8-bit version of the SPI on the TMS320C240 device. The upper 8 bits return 0s when read.

Topic **Page**

9.1 Differences vs. ’C240 SPI

This SPI has 16-bit transmit and receive capability, with double-buffered transmit and double-buffered receive. All data registers are 16-bits wide.

The SPI is no longer limited to a maximum transmission rate of CLKOUT / 8 in slave mode. The maximum transmission rate in *both* slave mode *and* master mode is now CLKOUT / 4.

Note that there is a software change required since writes of transmit data to the serial data register, SPIDAT (and the new transmit buffer, SPITXBUF), must be left-justified. On the ’C240, these writes had to be left-justified within an 8-bit register. Now they must be left justified within a 16-bit register.

The control and data bits for general-purpose bit I/O multiplexing have been removed from this peripheral, along with the associated registers, SPIPC1 (704Dh) and SPIPC2 (704Eh). These bits are now in the General-Purpose I/O registers.

The polarity of the SPI_SW_RESET bit in ’240x is the opposite of the ’240 SPI.

9.1.1 SPI Physical Description

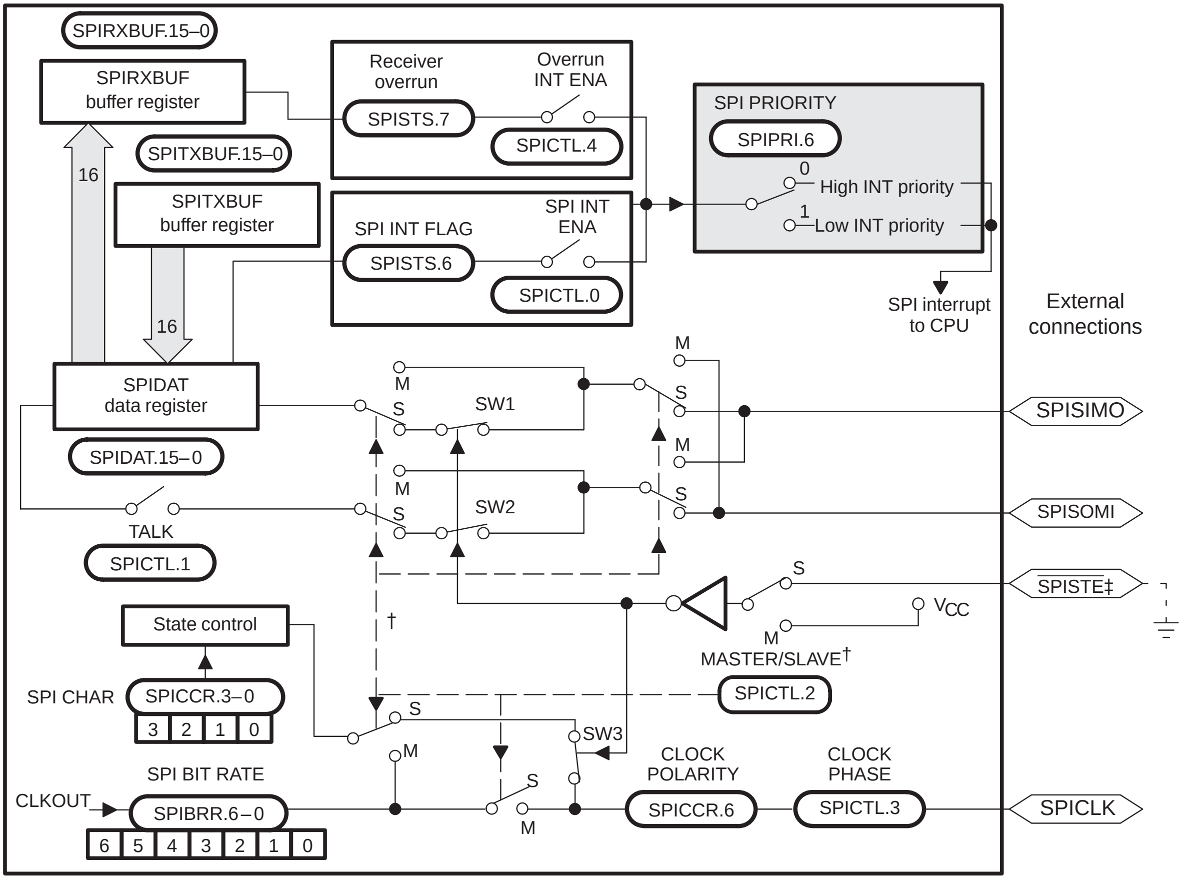
The SPI module, as shown in Figure 9–1, consists of:

- Four I/O pins:
  - SPISIMO (SPI slave in, master out)
  - SPISOMI (SPI slave out, master in)
  - SPICLK (SPI clock)
  - $\overline{\text{SPISTE}}$ (SPI slave transmit enable)
- Master and slave mode operations
- SPI serial receive buffer register (SPIRXBUF)
  This buffer register contains the data that is received from the network and that is ready for the CPU to read
- SPI serial transmit buffer register (SPITXBUF)
  This buffer register contains the next character to be transmitted when the current transmit has completed
- SPI serial data register (SPIDAT).
  This data shift register serves as the transmit/receive shift register
- SPICLK phase and polarity control

- ❏ State control logic
- ❏ Memory-mapped control and status registers

The basic function of the strobe ($\overline{\text{SPISTE}}$) pin is to act as a transmit enable input for the SPI module in slave mode. It stops the shift register so it cannot receive data and puts the SPISOMI pin in the high-impedance state.

Figure 9–1. SPI Module Block Diagram



† The diagram is shown in slave mode.

‡ The $\overline{\text{SPISTE}}$ pin is shown as being enabled, meaning the data can be transmitted or received in this mode. Note that switches SW1, SW2, and SW3 are closed in this configuration. The "switches" are assumed to close when their "control signal" is high.

9.2 SPI Control Registers

Nine registers inside the SPI module (listed in Table 9–1) control the SPI operations:

- ❑ SPICCR (SPI configuration control register). Contains control bits used for SPI configuration
  - ■ SPI module software reset
  - ■ SPICLK polarity selection
  - ■ Four SPI character-length control bits
- ❑ SPICTL (SPI operation control register). Contains control bits for data transmission
  - ■ Two SPI interrupt enable bits
  - ■ SPICLK phase selection
  - ■ Operational mode (master/slave)
  - ■ Data transmission enable
- ❑ SPISTS (SPI status register). Contains two receiver buffer status bits
  - ■ RECEIVER OVERRUN
  - ■ SPI INT FLAG
- ❑ SPIBRR (SPI baud rate register). Contains seven bits that determine the bit transfer rate
- ❑ SPIRXEMU (SPI receive emulation buffer register). Contains the received data. This register is used for emulation purposes only. The SPIRXBUF should be used for normal operation
- ❑ SPIRXBUF (SPI receive buffer — the serial receive buffer register). Contains the received data
- ❑ SPITXBUF (SPI transmit buffer — the serial transmit buffer register). Contains the next character to be transmitted
- ❑ SPIDAT (SPI data register). Contains data to be transmitted by the SPI, acting as the transmit/receive shift register. Data written to SPIDAT is shifted out on subsequent SPICLK cycles. For every bit shifted out of the SPI, a bit from the receive bit stream is shifted into the other end of the shift register
- ❑ SPIPRI (SPI priority register). Contains bits that specify interrupt priority and determine SPI operation on the XDS emulator during program suspensions

Table 9–1. Addresses of SPI Control Registers

Address	Register	Name
7040h	SPICCR	SPI configuration control register
7041h	SPICTL	SPI operation control register
7042h	SPISTS	SPI status register
7043h		Reserved
7044h	SPIBRR	SPI baud rate register
7045h		Reserved
7046h	SPIRXEMU	SPI receive emulation buffer register
7047h	SPIRXBUF	SPI serial receive buffer register
7048h	SPITXBUF	SPI serial transmit buffer register
7049h	SPIDAT	SPI serial data register
704Ah		Reserved
704Bh		Reserved
704Ch		Reserved
704Dh		Reserved
704Eh		Reserved
704Fh	SPIPRI	SPI priority control register

9.3 SPI Operation

This section describes the operation of the SPI. Included are explanations of the operation modes, interrupts, data format, clock sources, and initialization. Typical timing diagrams for data transfers are given.

9.3.1 Introduction to Operation

Figure 9–2 shows typical connections of the SPI for communications between two controllers: a master and a slave.

The master initiates data transfer by sending the SPICLK signal. For both the slave and the master, data is shifted out of the shift registers on one edge of the SPICLK and latched into the shift register on the opposite SPICLK clock edge. If the CLOCK PHASE bit (SPICTL.3) is high, data is transmitted and received a half-cycle before the SPICLK transition (see section 9.3.2, *SPI Module Slave and Master Operation Modes*, on page 9-7). As a result, both controllers send and receive data simultaneously. The application software determines whether the data is meaningful or dummy data. There are three possible methods for data transmission:

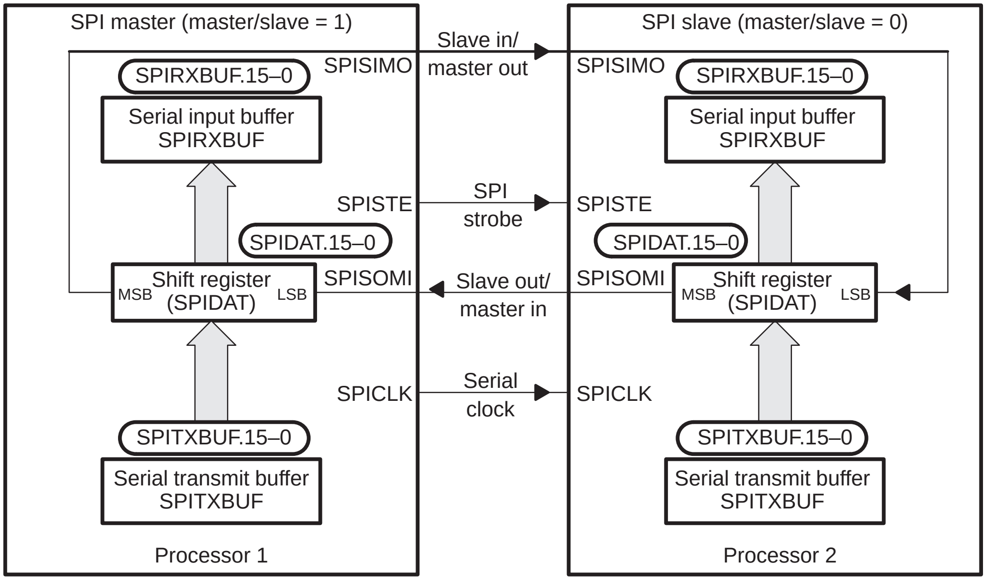
❑ Master sends data; slave sends dummy data.

❑ Master sends data; slave sends data.

❑ Master sends dummy data; slave sends data.

The master can initiate data transfer at any time because it controls the SPICLK signal. The software, however, determines how the master detects when the slave is ready to broadcast data.

Figure 9–2. SPI Master/Slave Connection



9.3.2 SPI Module Slave and Master Operation Modes

The SPI can operate in master or slave mode. The MASTER/SLAVE bit (SPICTL.2) selects the operating mode and the source of the SPICLK signal.

Master Mode

In the master mode (MASTER/SLAVE = 1), the SPI provides the serial clock on the SPICLK pin for the entire serial communications network. Data is output on the SPISIMO pin and latched from the SPISOMI pin.

The SPIBRR determines both the transmit and receive bit transfer rate for the network. The SPIBRR can select 126 different data transfer rates

Data written to SPIDAT or SPITXBUF initiates data transmission on the SPISIMO pin, MSB (most significant bit) first. Simultaneously, received data is shifted through the SPISOMI pin into the LSB (least significant bit) of SPIDAT. When the selected number of bits has been transmitted, the received data is transferred to the SPIRXBUF (buffered receiver) for the CPU to read. Data is stored right-justified in SPIRXBUF.

When the specified number of data bits has been shifted through SPIDAT, the following events occur:

❑ SPIDAT contents are transferred to SPIRXBUF.

❑ SPI INT FLAG bit (SPISTS.6) is set to 1.

- If there is valid data in the transmit buffer SPITXBUF, as indicated by the TXBUF FULL bit in SPISTS, this data is transferred to SPIDAT and is transmitted; otherwise, SPICLK stops after all bits have been shifted out of SPIDAT.
- If the SPI_INT_ENA bit (SPICTL.0) is set to 1, an interrupt is asserted.

In a typical application, the $\overline{\text{SPISTE}}$ pin could serve as a chip enable pin for slave SPI devices. (Drive this slave select pin low before transmitting master data to the slave device, and drive this pin high again after transmitting the master data.)

Slave Mode

In the slave mode (MASTER/SLAVE = 0), data shifts out on the SPISOMI pin and in on the SPISIMO pin. The SPICLK pin is used as the input for the serial shift clock, which is supplied from the external network master. The transfer rate is defined by this clock. The SPICLK input frequency should be no greater than the CLKOUT frequency divided by 4.

Data written to SPIDAT or SPITXBUF is transmitted to the network when appropriate edges of the SPICLK signal are received from the network master. Data written to the SPITXBUF register will be transferred to the SPIDAT register when all bits of the character to be transmitted have been shifted out of SPIDAT. If no character is currently being transmitted when SPITXBUF is written to, the data will be transferred immediately to SPIDAT. To receive data, the SPI waits for the network master to send the SPICLK signal and then shifts the data on the SPISIMO pin into SPIDAT. If data is to be transmitted by the slave simultaneously, and SPITXBUF has not been previously loaded, the data must be written to SPITXBUF or SPIDAT before the beginning of the SPICLK signal.

When the TALK bit (SPICTL.1) is cleared, data transmission is disabled, and the output line (SPISOMI) is put into the high-impedance state. If this occurs while a transmission is active the current character is completely transmitted even though SPISOMI is forced into the high-impedance state. This ensures that the SPI is still able to receive incomming data correctly. ThisTALK bit allows many slave devices to be tied together on the network, but only one slave at a time is allowed to drive the SPISOMI line.

The $\overline{\text{SPISTE}}$ pin operates as the slave select pin. An active-low signal on the $\overline{\text{SPISTE}}$ pin allows the slave SPI to transfer data to the serial data line; an inactive high signal causes the slave SPI's serial shift register to stop and its serial output pin to be put into the high-impedance state. This allows many slave devices to be tied together on the network, although only one slave device is selected at a time.

9.4 SPI Interrupts

Five control bits are used to initialize the SPI’s interrupts:

❑ SPI_INT_ENA bit (SPICTL.0)

❑ SPI_INT_FLAG bit (SPISTS.6)

❑ OVERRUN_INT_ENA bit (SPICTL.4)

❑ RECEIVER_OVERRUN flag bit (SPISTS.7)

❑ SPI_PRIORITY bit (SPIPRI.6)

9.4.1 SPI_INT_ENA Bit (SPICTL.0)

When the SPI interrupt enable bit is set and an interrupt condition occurs, the corresponding interrupt is asserted.

0	Disable SPI interrupts
1	Enable SPI interrupts

9.4.2 SPI_INT_FLAG Bit (SPISTS.6)

This status flag indicates that a character has been placed in the SPI receiver buffer and is ready to be read.

When a complete character has been shifted into or out of SPIDAT, the SPI_INT_FLAG bit (SPISTS.6) is set, and an interrupt is generated if enabled by the SPI_INT_ENA bit. The interrupt flag remains set until it is cleared by one of the following events:

❑ The interrupt is acknowledged (this is different from the ’C240).

❑ The CPU reads the SPIRXBUF (reading the SPIRXEMU does not clear the SPI_INT_FLAG).

❑ The device enters IDLE2 or HALT mode with an IDLE instruction.

❑ Software sets the SPI_SW_RESET bit (SPICCR.7).

❑ A system reset occurs.

When the SPI_INT_FLAG bit is set, a character has been placed into the SPIRXBUF and is ready to be read. If the CPU does not read the character by the time the next complete character has been received, the new character is written into SPIRXBUF, and the RECEIVER OVERRUN flag bit (SPISTS.7) is set.

9.4.3 OVERRUN_INT_ENA Bit (SPICTL.4)

Setting the overrun interrupt enable bit allows the assertion of an interrupt whenever the RECEIVER_OVERRUN flag bit (SPISTS.7) is set by hardware. Interrupts generated by SPISTS.7 and by the SPI_INT_FLAG (SPISTS.6) bit share the same interrupt vector.

0 Disable RECEIVER_OVERRUN flag bit interrupts.

1 Enable RECEIVER_OVERRUN flag bit interrupts.

9.4.4 RECEIVER_OVERRUN_FLAG Bit (SPISTS.7)

The RECEIVER_OVERRUN flag bit is set whenever a new character is received and loaded into the SPIRXBUF before the previously received character has been read from the SPIRXBUF. The RECEIVER_OVERRUN flag bit must be cleared by software.

9.4.5 SPI PRIORITY Bit (SPIPRI.6)

The value of the SPI_PRIORITY bit determines the priority of the interrupt request from the SPI.

0 Interrupts are high-priority requests.

1 Interrupts are low-priority requests.

9.4.6 Data Format

Four bits (SPICCR.3–0) specify the number of bits (1 to 16) in the data character. This information directs the state control logic to count the number of bits received or transmitted to determine when a complete character has been processed. The following statements apply to characters with fewer than 16 bits:

- Data must be left justified when written to SPIDAT and SPITXBUF.
- Data read back from SPIRXBUF is right justified.
- SPIRXBUF contains the most recently received character, right justified, plus any bits that remain from previous transmission(s) that have been shifted to the left (shown in Example 9–1).

Example 9–1. Transmission of Bit from SPIRXBUF

Conditions:

1) Transmission character length = 1 bit (specified in bits SPICCR.3–0)
2) The current value of SPIDAT = 737Bh

SPIDAT (before transmission)

0	1	1	1	0	0	1	1	0	1	1	1	1	0	1	1

SPIDAT (after transmission)

(TXed) 0 ←	1	1	1	0	0	1	1	0	1	1	1	1	0	1	1	x	← (RXed)

SPIRXBUF (after transmission)

1	1	1	0	0	1	1	0	1	1	1	1	0	1	1	x

Note: x = 1 if SPISOMI data is high; x = 0 if SPISOMI data is low; master mode is assumed.

9.4.7 Baud Rate and Clocking Schemes

The SPI module supports 125 different baud rates and four different clock schemes. Depending on whether the SPI clock is in slave or master mode, the SPICLK pin can receive an external SPI clock signal or provide the SPI clock signal, respectively.

❑ In the slave mode, the SPI clock is received on the SPICLK pin from the external source, and can be no greater than the CLKOUT frequency divided by 4.

❑ In the master mode, the SPI clock is generated by the SPI and is output on the SPICLK pin, and can be no greater than the CLKOUT frequency divided by 4.

Baud Rate Determination

Equation 9–1 shows how to determine the SPI baud rates.

Equation 9–1. SPI Baud-Rate Calculations

❑ For SPIBRR = 3 to 127:

$$\text{SPI Baud Rate} = \frac{\text{CLKOUT}}{(\text{SPIBRR} + 1)}$$

- ❑ For SPIBRR = 0, 1, or 2:

$$\text{SPI Baud Rate} = \frac{\text{CLKOUT}}{4}$$

where:

CLKOUT = CPU clock frequency of the device

SPIBRR = Contents of the SPIBRR in the master SPI device

To determine what value to load into SPIBRR, you must know the device system clock (CLKOUT) frequency (which is device-specific) and the baud rate at which you will be operating.

Example 9–2 shows how to determine the maximum baud rate at which a ’C24x can communicate. Assume that CLKOUT = 30 MHz.

Example 9–2. Maximum Baud-Rate Calculation

$$\begin{aligned}\text{SPI Baud Rate} &= \frac{\text{CLKOUT}}{(\text{SPIBRR} + 1)}\\ &= \frac{30 \times 10^6}{(3 + 1)}\\ &= 7.5 \times 10^6 \text{ bps}\end{aligned}$$

The maximum master baud rate would be 5.0 Mbps.

9.4.8 SPI Clocking Schemes

The CLOCK_POLARITY (SPICCR.6) and CLOCK_PHASE (SPICTL.3) bits control four different clocking schemes on the SPICLK pin. The CLOCK POLARITY bit selects the active edge of the clock, either rising or falling. The CLOCK_PHASE bit selects a half-cycle delay of the clock. The four different clocking schemes are as follows:

- ❑ Falling Edge Without Delay. The SPI transmits data on the falling edge of the SPICLK and receives data on the rising edge of the SPICLK.
- ❑ Falling Edge With Delay. The SPI transmits data one half-cycle ahead of the falling edge of the SPICLK signal and receive data on the falling edge of the SPICLK signal.
- ❑ Rising Edge Without Delay. The SPI transmits data on the rising edge of the SPICLK signal and receive data on the falling edge of the SPICLK signal.

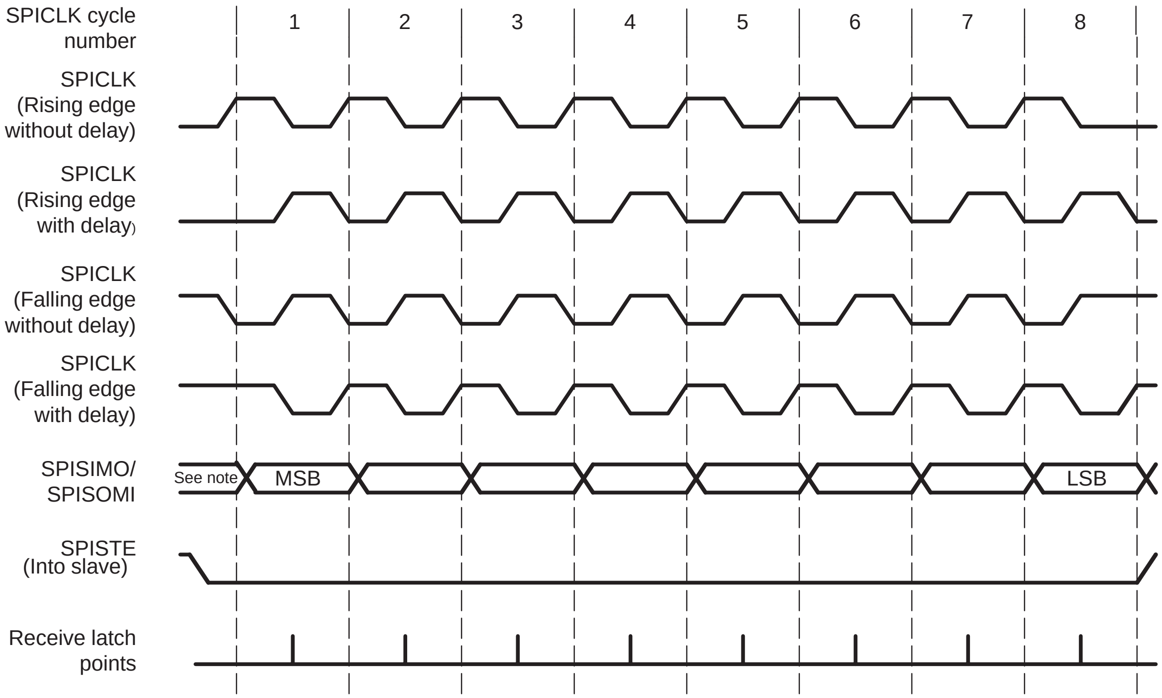
❑ Rising Edge With Delay. The SPI transmits data one half-cycle ahead of the rising edge of the SPICLK signal and receives data on the rising edge of the SPICLK signal.

The selection procedure for the SPI clocking scheme is shown in Table 9–2. Examples of these four clocking schemes relative to transmitted and received data are shown in Figure 9–3.

Table 9–2. SPI Clocking Scheme Selection Guide

SPICLK Scheme	CLOCK POLARITY (SPICCR.6)	CLOCK PHASE (SPICTL.3)
Rising edge without delay	0	0
Rising edge with delay	0	1
Falling edge without delay	1	0
Falling edge with delay	1	1

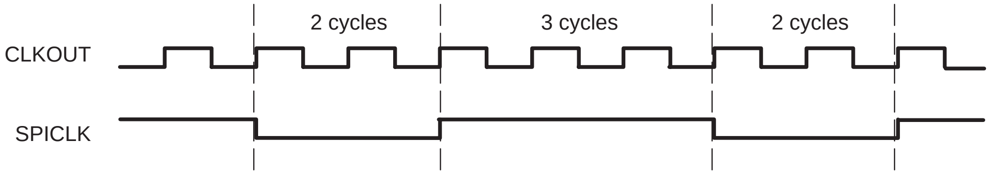
Figure 9–3. SPICLK Signal Options


SPICLK cycle number
1
2
3
4
5
6
7
8
SPICLK (Rising edge without delay)
SPICLK (Rising edge with delay))
SPICLK (Falling edge without delay)
SPICLK (Falling edge with delay)
SPISIMO/ SPISOMI
See note
MSB
LSB
SPISTE (Into slave)
Receive latch points

Note: Previous data bit

For the SPI, the SPICLK symmetry is retained only when the result of (SPIBRR + 1) is an even value. When (SPIBRR + 1) is an odd value and SPIBRR is greater than 3, the SPICLK becomes asymmetrical. The low pulse of the SPICLK is one CLKOUT longer than the high pulse when the CLOCK_POLARITY bit is clear (0). When the CLOCK_POLARITY bit is set to 1, the high pulse of the SPICLK is one CLKOUT longer than the low pulse, as shown in Figure 9–4.

Figure 9–4. SPI: SPICLK-CLKOUT Characteristic when (BRR + 1) is Odd, BRR > 3, and CLOCK POLARITY = 1



9.4.9 Initialization Upon Reset

A system reset forces the SPI peripheral module into the following default configuration:

- ❑ The unit is configured as a slave module (MASTER/SLAVE = 0).
- ❑ The transmit capability is disabled (TALK = 0).
- ❑ Data is latched at the input on the falling edge of the SPICLK signal.
- ❑ Character length is assumed to be one bit.
- ❑ The SPI interrupts are disabled.
- ❑ Data in SPIDAT is reset to 0000h.
- ❑ SPI module pin functions are selected as general-purpose inputs (this is done in I/O Mux control regsiter B [MCRB]).

To change this SPI configuration:

1) Clear the SPI_SW_RESET bit (SPICCR.7) to 0 to force the SPI to the reset state.

2) Initialize the SPI configuration, format, baud rate, and pin functions as desired.

3) Set the SPI_SW_RESET bit to 1 to release the SPI from the reset state.

4) Write to SPIDAT or SPITXBUF (this initiates the communication process in the master).

5) Read SPIRXBUF after the data transmission has completed (SPISTS.6 = 1) to determine what data was received.

9.4.10 Proper SPI Initialization Using the SPI SW RESET Bit

To prevent unwanted and unforeseen events from occurring during or as a result of initialization changes, clear the SPI SW RESET bit (SPICCR.7) before making initialization changes, and then set this bit after initialization is complete.

CAUTION

Do not change SPI configuration when communication is in progress.

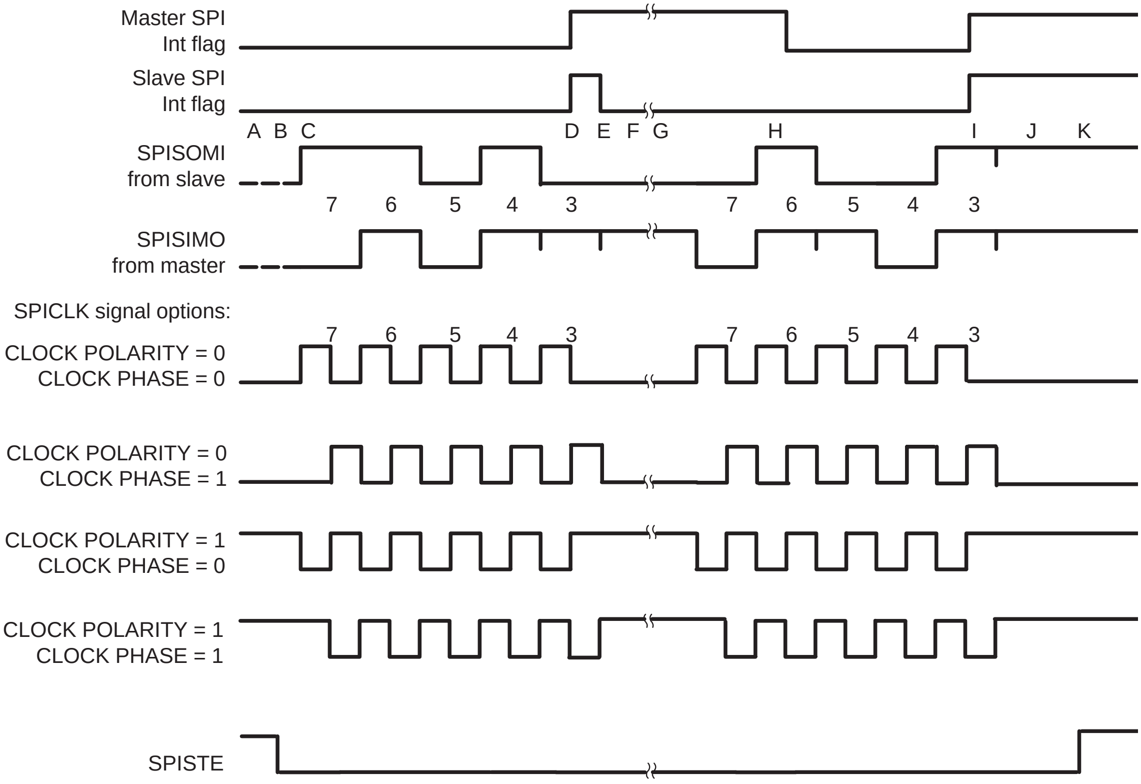
9.4.11 Data Transfer Example

The timing diagram, shown in Figure 9–5, illustrates an SPI data transfer between two devices using a character length of five bits with the SPICLK being symmetrical.

The timing diagram with SPICLK unsymmetrical (Figure 9–4) shares similar characterizations with Figure 9–5 except that the data transfer is one CLKOUT cycle longer per bit during the low pulse (CLOCK POLARITY = 0) or during the high pulse (CLOCK POLARITY = 1) of the SPICLK.

Figure 9–5, *Five Bits per Character,* is applicable for 8-bit SPI only and is not for ‘24x devices that are capable of working with 16-bit data. The figure is shown for illustrative purposes only.

Figure 9–5. Five Bits per Character



A. Slave writes 0D0h to SPIDAT and waits for the master to shift out the data.
B. Master sets the slave SPISTE signal low (active).
C. Master writes 058h to SPIDAT, which starts the transmission procedure.
D. First byte is finished and sets the interrupt flags.
E. Slave reads 0Bh from its SPIRXBUF (right justified).
F Slave writes 04Ch to SPIDAT and waits for the master to shift out the data.
G. Master writes 06Ch to SPIDAT, which starts the transmission procedure.
H. Master reads 01Ah from the SPIRXBUF (right justified).
I. Second byte is finished and sets the interrupt flags.
J. Master reads 89h and the slave reads 8Dh from their respective SPIRXBUF. After the user’s software masks off the unused bits, the master receives 09h and the slave receives 0Dh.
K. Master clears the slave SPISTE signal high (inactive).

9.5 SPI Control Registers

The SPI is controlled and accessed through registers in the control register file. Figure 9–6 lists the SPI control registers and bit numbers.

Figure 9–6. SPI Control Registers

Addr.	Register Name	Bit number								
		15–8	7	6	5	4	3	2	1	0
7040h	SPICCR	–	SPI SW RESET	CLOCK POLARITY	–		SPI CHAR3	SPI CHAR2	SPI CHAR1	SPI CHAR0
7041h	SPICTL	–	–			OVER-RUN INT ENA	CLOCK PHASE	MASTER/ SLAVE	TALK	SPI INT ENA
7042h	SPISTS	–	RECEIVER OVERRUN	SPI INT FLAG	TX BUF FULL	–				
7043h	–	–	–							
7044h	SPIBRR	–	–	SPI BIT RATE 6	SPI BIT RATE 5	SPI BIT RATE 4	SPI BIT RATE 3	SPI BIT RATE 2	SPI BIT RATE 1	SPI BIT RATE 0
7045h	–	–	–							
7046h	SPIRXEMU	ERXB 15–8	ERXB7	ERXB6	ERXB5	ERXB4	ERXB3	ERXB2	ERXB1	ERXB0
7047h	SPIRXBUF	RXB 15–8	RXB7	RXB6	RXB5	RXB4	RXB3	RXB2	RXB1	RXB0
7048h	SPITXBUF	TXB 15–8	TXB7	TXB6	TXB5	TXB4	TXB3	TXB2	TXB1	TXB0
7049h	SPIDAT	SDAT 15–8	SDAT7	SDAT6	SDAT5	SDAT4	SDAT3	SDAT2	SDAT1	SDAT0
704Ah	–	–	–							
704Bh	–	–	–							
704Ch	–	–	–							
704Dh	–	–	–							
704Eh	–	–	–							
704Fh	SPIPRI	–	–	SPI PRIORITY	SPI SUSP SOFT	SPI SUSP FREE	–			

– Reserved

9.5.1 SPI Configuration Control Register (SPICCR)

The SPI Configuration Control Register (SPICCR) controls the setup of the SPI for operation.

Figure 9–7. SPI Configuration Control Register (SPICCR) — Address 7040h

7	6	5–4	3	2	1	0
SPI_SW_ RESET	CLOCK_ POLARITY	Reserved	SPICHAR3	SPICHAR2	SPICHAR1	SPICHAR0
RW-0	RW-0	R-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 7 **SPI_SW_RESET**. SPI Software Reset. When changing configuration, you should clear this bit before the changes and set this bit before resuming operation. (See Section 9.4.10 on page 9-15.)

0 Initializes the SPI operating flags to the reset condition.

Specifically, the RECEIVER_OVERRUN flag bit (SPISTS.7), the SPI_INT_FLAG bit (SPISTS.6), and the TXBUF_FULL flag (SPISTS.5) are cleared. The SPI configuration remains unchanged. If the module is operating as a master, the SPICLK signal output returns to its inactive level.

1 SPI is ready to transmit or receive the next character.

When the SPI SW RESET bit is a 1, a character written to the transmitter will not be shifted out when this bit clears. A new character must be written to the serial data register.

Bit 6 **CLOCK_POLARITY**. Shift Clock Polarity. This bit controls the polarity of the SPICLK signal. CLOCK_POLARITY and CLOCK_PHASE (SPICTL.3) control four clocking schemes on the SPICLK pin. See Section 9.4.8, *SPI Clocking Schemes*, on page 9-12.

0 Data is output on rising edge and input on falling edge. When no SPI data is sent, SPICLK is at low level.

The data input and output edges depend on the value of the CLOCK_PHASE (SPICTL.3) bit as follows:

- CLOCK_PHASE = 0: Data is output on the rising edge of the SPICLK signal; input data is latched on the falling edge of the SPICLK signal.
- CLOCK_PHASE = 1: Data is output one half-cycle before the first rising edge of the SPICLK signal and on subsequent falling edges of the SPICLK signal; input data is latched on the rising edge of the SPICLK signal.

1 Data is output on falling edge and input on rising edge. When no SPI data is used, SPICLK is at high level.

The data input and output edges depend on the value of the CLOCK_PHASE bit (SPICTL.3) as follows:

- ❑ CLOCK_PHASE = 0: Data is output on the falling edge of the SPICLK signal; input data is latched on the rising edge of the SPICLK signal.
- ❑ CLOCK_PHASE = 1: Data is output one half-cycle before the first falling edge of the SPICLK signal and on subsequent rising edges of the SPICLK signal; input data is latched on the falling edge of the SPICLK signal.

Bits 5–4 **Reserved**. Reads return zero; writes have no effect.

Bits 3–0 **SPI CHAR3–SPI CHAR0**. Character Length Control Bits 3–0. These four bits determine the number of bits to be shifted in or out as a single character during one shift sequence.

Table 9–3 lists the character length selected by the bit values.

Table 9–3. Character Length Control Bit Values

SPI CHAR3	SPI CHAR2	SPI CHAR1	SPI CHAR0	Character Length
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
0	0	1	1	4
0	1	0	0	5
0	1	0	1	6
0	1	1	0	7
0	1	1	1	8
1	0	0	0	9
1	0	0	1	10
1	0	1	0	11
1	0	1	1	12
1	1	0	0	13
1	1	0	1	14
1	1	1	0	15
1	1	1	1	16

9.5.2 SPI Operation Control Register (SPICTL)

The SPICTL operation control register controls data transmission, the SPI's ability to generate interrupts, the SPICLK phase, and the operational mode (slave or master).

Figure 9–8. SPI Operation Control Register (SPICTL) — Address 7041h

7–5	4	3	2	1	0
Reserved	OVERRUN_ INT_ENA	CLOCK_ PHASE	MASTER/ SLAVE	TALK	SPI_INT_ ENA
R-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 7–5 **Reserved**. Reads return zero; writes have no effect.

Bit 4 **OVERRUN_INT_ENA**. Overrun Interrupt Enable. Setting this bit causes an interrupt to be generated when the RECEIVER OVERRUN flag bit (SPISTS.7) is set by hardware. Interrupts generated by the RECEIVER_OVERRUN flag bit and the SPI_INT_FLAG bit (SPISTS.6) share the same interrupt vector.

0 Disable RECEIVER_OVERRUN flag bit (SPISTS.7) interrupts.

1 Enable RECEIVER_OVERRUN flag bit (SPISTS.7) interrupts.

Bit 3 **CLOCK_PHASE**. SPI Clock Phase Select. This bit controls the phase of the SPICLK signal.

0 Normal SPI clocking scheme, depending on the CLOCK_POLARITY bit (SPICCR.6).

1 SPICLK signal delayed by one half-cycle; polarity determined by the CLOCK_POLARITY bit.

CLOCK_PHASE and CLOCK_POLARITY (SPICCR.6) bits make four different clocking schemes possible (see Figure 9–3). When operating with CLOCK_PHASE high, the SPI (master or slave) makes the first bit of data available after SPIDAT is written and before the first edge of the SPICLK signal, regardless of which SPI mode is being used.

Bit 2 **MASTER/SLAVE**. SPI Network Mode Control. This bit determines whether the SPI is a network master or slave. During reset initialization, the SPI is automatically configured as a network slave.

0 SPI configured as a slave.

1 SPI configured as a master.

Bit 1 **TALK**. Master/Slave Transmit Enable. The TALK bit can disable data transmission (master or slave) by placing the serial data output in the high-impedance state. If this bit is disabled during a transmission, the transmit shift register continues to operate until the previous character is shifted out. When the TALK bit is disabled, the SPI is still able to receive characters and update the status flags. TALK is cleared (disabled) by a system reset.

0 Disables transmission:

- Slave mode operation: If not previously configured as a general-purpose I/O pin, the SPISOMI pin will be put in the high-impedance state.
- Master mode operation: If not previously configured as a general-purpose I/O pin, the SPISIMO pin will be put in the high-impedance state.

1 Enables transmission

For the 4-pin option, ensure to enable the receiver's SPISTB input pin.

Bit 0 **SPI_INT_ENA**. SPI Interrupt Enable. This bit controls the SPI's ability to generate a transmit/receive interrupt. The SPI INT FLAG bit (SPISTS.6) is unaffected by this bit.

0 Disables interrupt

1 Enables interrupt

9.5.3 SPI Status Register (SPISTS)

The SPISTS register contains the receive buffer status bits.

Figure 9–9. SPI Status Register (SPISTS) — Address 7042h

7	6	5	4–0
RECEIVER_OVERRUN_FLAG†‡	SPI_INT_FLAG†‡	TX_BUF_FULL_FLAG‡	Reserved
RC-0	RC-0	RC-0	R-0

Note: R = Read access, C = Clear, -0 = value after reset

† The RECEIVER_OVERRUN_FLAG bit and the SPI_INT_FLAG bit share the same interrupt vector.

‡ Writing a 0 to bits 5, 6, and 7 has no effect.

Bit 7 **RECEIVER_OVERRUN_FLAG.** SPI Receiver Overrun Flag. This bit is a read/clear only flag. The SPI hardware sets this bit when a receive or transmit operation completes before the previous character has been read from the buffer. The bit indicates that the last received character has been overwritten and therefore lost (when the SPIRXBUF was overwritten by the SPI module before the previous character was read by the user application). The SPI requests one interrupt sequence each time this bit is set if the OVERRUN_INT_ENA bit (SPICTL.4) is set high. The bit is cleared in one of three ways:

- Writing a 1 to this bit
- Writing a 0 to SPI_SW_RESET (SPICCR.7)
- Resetting the system

If the OVERRUN_INT_ENA bit (SPICTL.4) is set, the SPI requests only one interrupt upon the first occurance of setting the RECEIVER_OVERRUN flag bit. Subsequent overruns will not request additional interrupts if this flag bit is already set. This means that in order to allow *new* overrun interrupt requests the user must clear this flag bit by writing a 1 to SPISTS.7 each time an overrun condition occurs. In other words, if the RECEIVER_OVERRUN flag bit is left set (not cleared) by the interrupt service routine, another overrun interrupt will not be immediately re-entered when the interrupt service routine is exited.

However, the RECEIVER_OVERRUN flag bit should be cleared during the interrupt service routine because the RECEIVER_OVERRUN flag bit and SPI_INT_FLAG bit share the same interrupt vector. This will alleviate any possible doubt as to the source of the interrupt when the next byte is received.

Bit 6 **SPI_INT_FLAG**. SPI Interrupt Flag. SPI_INT_FLAG is a read-only flag. The SPI hardware sets this bit to indicate that it has completed sending or receiving the last bit and is ready to be serviced. The received character is placed in the receiver buffer at the same time this bit is set. This flag causes an interrupt to be requested if the SPI INT ENA bit (SPICTL.0) is set. This bit is cleared in one of three ways:

- Reading SPIRXBUF
- Writing a 1 to SPI SW RESET (SPICCR)
- Resetting the system

Bit 5 **TX_BUF_FULL_FLAG**. SPI Transmit Buffer full flag. This read only bit gets set to 1 when a character is written to the SPI Transmit buffer SPITXBUF. It is cleared when the character is automatically loaded into SPIDAT when the shifting out of a previous character is complete. It is cleared at reset.

Bits 4–0 **Reserved**. Reads return zero; writes have no effect.

9.5.4 SPI Baud Rate Register (SPIBRR)

The SPIBRR contains the bits used for baud-rate selection.

Figure 9–10. SPI Baud Rate Register (SPIBRR) — Address 7044h

7	6	5	4	3	2	1	0
Reserved	SPI BIT RATE 6	SPI BIT RATE 5	SPI BIT RATE 4	SPI BIT RATE 3	SPI BIT RATE 2	SPI BIT RATE 1	SPI BIT RATE 0
R-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 7 **Reserved**. Reads return zero; writes have no effect.

Bits 6–0 **SPI BIT RATE 6–SPI BIT RATE 0**. SPI Bit Rate (Baud) Control. These bits determine the bit transfer rate if the SPI is the network master. There are 125 data transfer rates (each a function of the CPU clock, CLKOUT) that can be selected. One data bit is shifted per SPICLK cycle. (SPICLK is the baud rate clock output on the SPICLK pin.)

If the SPI is a network slave, the module receives a clock on the SPICLK pin from the network master; therefore, these bits have no effect on the SPICLK signal. The frequency of the input clock from the master should not exceed the slave SPI's SPICLK signal divided by 4.

In master mode, the SPI clock is generated by the SPI and is output on the SPICLK pin. The SPI baud rates are determined by the formula in Equation 9–2.

Equation 9–2. SPI Baud-Rate Calculations

❑ For SPIBRR = 3 to 127:

$$\text{SPI Baud Rate} = \frac{\text{CLKOUT}}{(\text{SPIBRR} + 1)}$$

❑ For SPIBRR = 0, 1, or 2:

$$\text{SPI Baud Rate} = \frac{\text{CLKOUT}}{4}$$

where: CLKOUT = CPU clock frequency of the device
SPIBRR = Contents of the SPIBRR in the master SPI device

9.5.5 SPI Emulation Buffer Register (SPIRXEMU)

The SPIRXEMU contains the received data. Reading the SPIRXEMU does not clear the SPI INT FLAG bit (SPISTS.6). This is not a real register but a dummy address from which the contents of SPIRXBUF can be read by the emulator without clearing the SPI_INT_FLAG.

Figure 9–11. SPI Emulation Buffer Register (SPIRXEMU) — Address 7046h

15	14	13	12	11	10	9	8
ERXB15	ERXB14	ERXB13	ERXB12	ERXB11	ERXB10	ERXB9	ERXB8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
ERXB7	ERXB6	ERXB5	ERXB4	ERXB3	ERXB2	ERXB1	ERXB0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Note: R = Read access, -0 = value after reset

Bits 15–0 **ERXB15–ERXB0**. Emulation Buffer Received Data. The SPIRXEMU functions almost identically to the SPIRXBUF, except that reading the SPIRXEMU does not clear the SPI INT FLAG bit (SPISTS.6). Once the SPIDAT has received the complete character, the character is transferred to the SPIRXEMU and SPIRXBUF, where it can be read. At the same time, SPI INT FLAG is set.

This mirror register was created to support emulation. Reading the SPIRXBUF clears the SPI INT FLAG bit (SPISTS.6). In the normal operation of the emulator, the control registers are read to continually update the contents of these registers on the display screen. The SPIRXEMU was created so that the emulator can read this register and properly update the contents on the display screen. Reading SPIRXEMU does not clear the SPI INT FLAG, but reading SPIRXBUF clears this flag. In other words, SPIRXEMU enables the emulator to emulate the true operation of the SPI more accurately.

It is recommended that you view SPIRXEMU in the normal emulator run mode.

9.5.6 SPI Serial Receive Buffer Register (SPIRXBUF)

The SPIRXBUF contains the received data. Reading the SPIRXBUF clears the SPI_INT_FLAG bit (SPISTS.6).

Figure 9–12. SPI Serial Receive Buffer Register (SPIRXBUF) — Address 7047h

15	14	13	12	11	10	9	8
RXB15	RXB14	RXB13	RXB12	RXB11	RXB10	RXB9	RXB8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
RXB7	RXB6	RXB5	RXB4	RXB3	RXB2	RXB1	RXB0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Note: R = Read access, -0 = value after reset

Bits 15–0 **RXB15–RXB0**. Received Data. Once SPIDAT has received the complete character, the character is transferred to SPIRXBUF, where it can be read. At the same time, the SPI_INT_FLAG bit (SPISTS.6) is set. Since data is shifted into the SPI's most significant bit first, it is stored right-justified in this register.

9.5.7 SPI Serial Transmit Buffer Register (SPITXBUF)

The SPITXBUF stores the next character to be transmitted. Writing to this register sets the TX_BUF_FULL (SPISTS.5) flag. When transmission of the current character is complete, the contents of this register are automatically loaded in SPIDAT and the TX_BUF_FULL flag is cleared. If no transmission is currently active, data written to this register falls through into the SPIDAT register and the TX_BUF_FULL flag is not set.

In master mode, if no transmission is currently active, writing to this register initiates a transmission in the same manner that writing to SPIDAT does.

Figure 9–13. SPI Serial Transmit Buffer Register (SPITXBUF) — Address 7048h

15	14	13	12	11	10	9	8
TXB15	TXB14	TXB13	TXB12	TXB11	TXB10	TXB9	TXB8
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
TXB7	TXB6	TXB5	TXB4	TXB3	TXB2	TXB1	TXB0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, -0 = value after reset

Bits 15–0 **TXB15–TXB0**. Transmit Data Buffer. This is where the next character to be transmitted is stored. When the transmission of the current character has completed, if the TX_BUF_FULL flag is set, the contents of this register is automatically transferred to SPIDAT, and the TX_BUF_FULL flag is cleared.

Note: Writes to SPITXBUF must be left-justified

9.5.8 SPI Serial Data Register (SPIDAT)

The SPIDAT is the transmit/receive shift register. Data written to the SPIDAT is shifted out (MSB) on subsequent SPICLK cycles. For every bit shifted out (MSB) of the SPI, a bit is shifted into the LSB end of the shift register.

Figure 9–14. SPI Serial Data Register (SPIDAT) — Address 7049h

15	14	13	12	11	10	9	8
SDAT15	SDAT14	SDAT13	SDAT12	SDAT11	SDAT10	SDAT9	SDAT8
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
SDAT7	SDAT6	SDAT5	SDAT4	SDAT3	SDAT2	SDAT1	SDAT0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, -0 = value after reset

Bits 15–0 **SDAT15–SDAT0**. Serial Data. Writing to the SPIDAT performs two functions:

- It provides data to be output on the serial output pin if the TALK bit (SPICTL.1) is set.
- When the SPI is operating as a master, a data transfer is initiated. When initiating a transfer, see the CLOCK_POLARITY bit (SPICCR.6) and the CLOCK_PHASE bit (SPICTL.3) for the requirements.

In master mode, writing dummy data to SPIDAT initiates a receiver sequence. Since the data is not hardware-justified for characters shorter than sixteen bits, transmit data must be written in left-justified form, and received data read in right-justified form.

9.5.9 SPI Priority Control Register (SPIPRI)

The SPIPRI selects the interrupt priority level of the SPI interrupt and controls the SPI operation on the XDS emulator during program suspends, such as hitting a breakpoint.

Figure 9–15. SPI Priority Control Register (SPIPRI) — Address 704Fh

7	6	5	4	3–0
Reserved	SPI_ PRIORITY	SPI_SUSP_ SOFT	SPI_SUSP_ FREE	Reserved
R-0	RW	RW	RW-0	R-0

Note: R = Read access, W = Write access, -0 = value after reset

Bit 7 **Reserved**. Reads return zero; writes have no effect.

Bit 6 **SPI_PRIORITY**. Interrupt Priority Select. This bit specifies the priority level of the SPI interrupt.

0	Interrupts are high-priority requests.
1	Interrupts are low-priority requests.

Bits 5–4 **SPI_SUSP_SOFT and FREE bits.** These bits determine what occurs when an emulation suspend occurs (for example, when the debugger hits a breakpoint). The peripheral can continue whatever it is doing (free run mode) or, if in stop mode, it can either stop immediately or stop when the current operation (the current receive/transmit sequence) is complete.

Bit 5	**Bit 4**	
Soft	**Free**	
0	0	Immediate stop on suspend.
1	0	Complete current receive/transmit sequence before stopping.
X	1	Free run, continue SPI operation regardless of suspend.

Bits 3–0 **Reserved**. Reads return zero; writes have no effect.

9.6 SPI Example Waveforms

Figure 9–16. CLOCK_POLARITY = 0, CLOCK_PHASE = 0 (All data transitions are during the rising edge. Inactive level is low.)

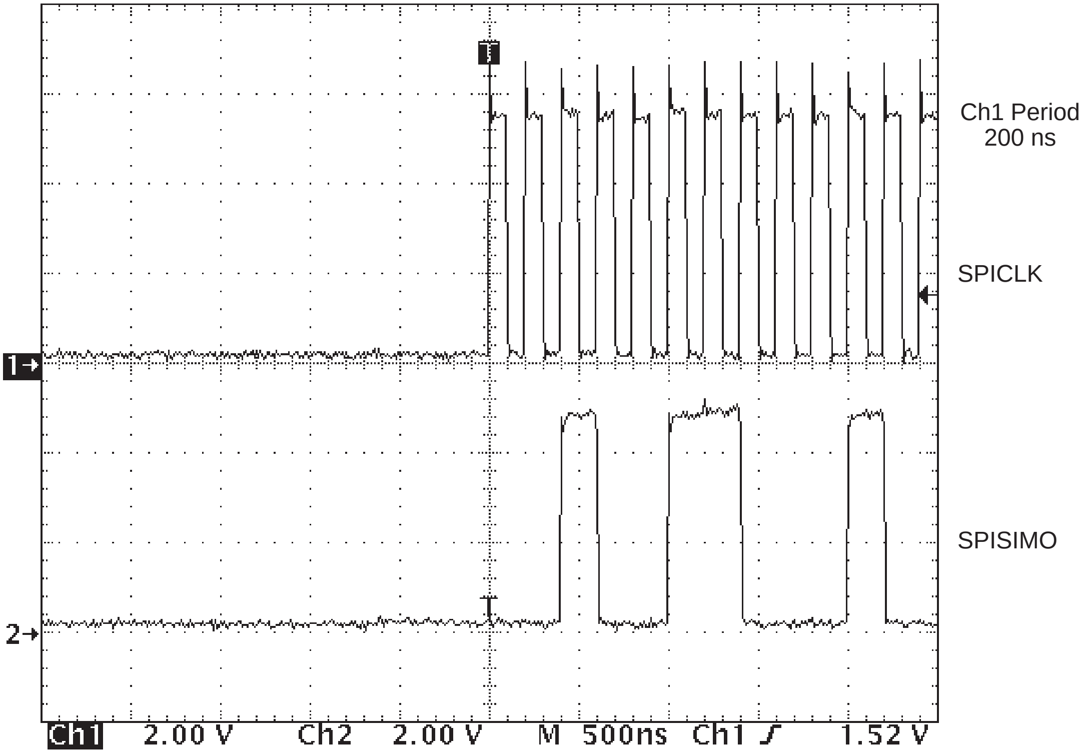


Figure 9–17. CLOCK_POLARITY = 0, CLOCK_PHASE = 1 (Add data transitions are during the rising edge, but delayed by half clock cycle. Inactive level is low.)

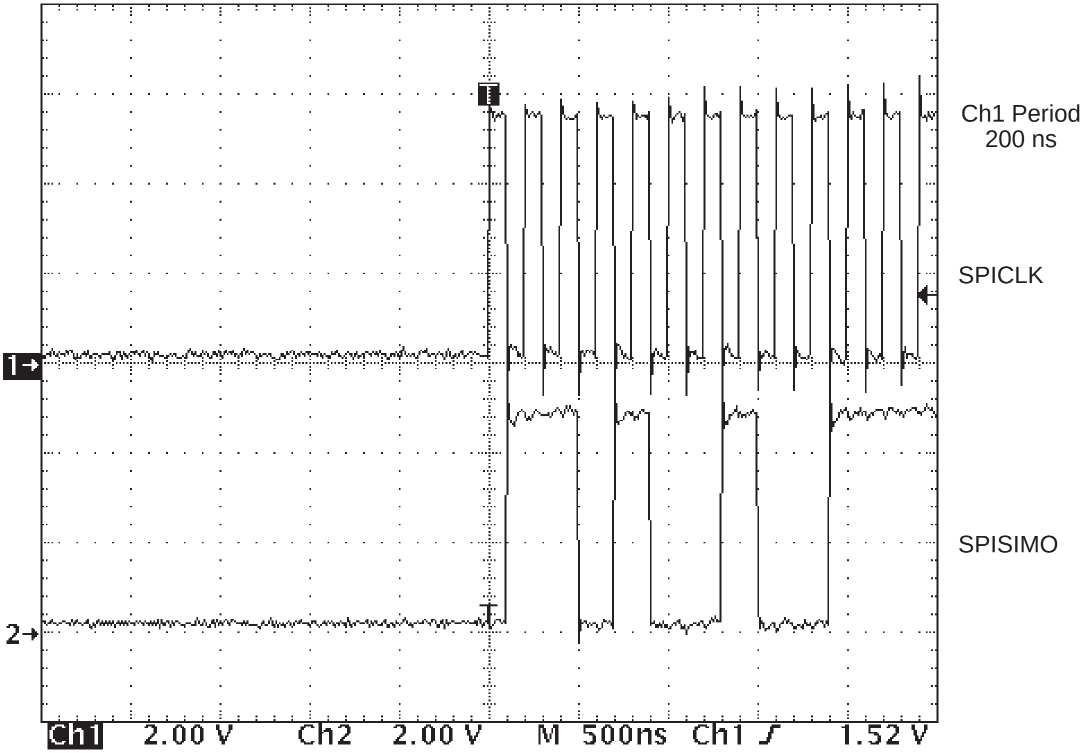


Figure 9–18. CLOCK_POLARITY = 1, CLOCK_PHASE = 0 (All data transitions are during the falling edge. Inactive level is high.)

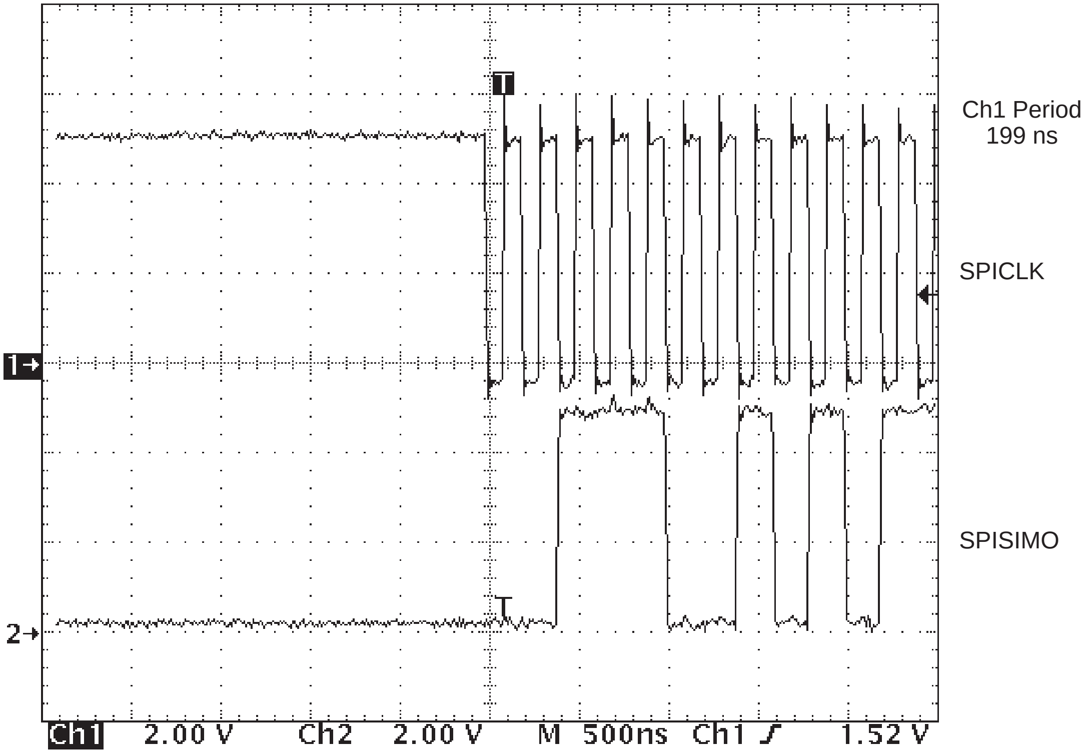
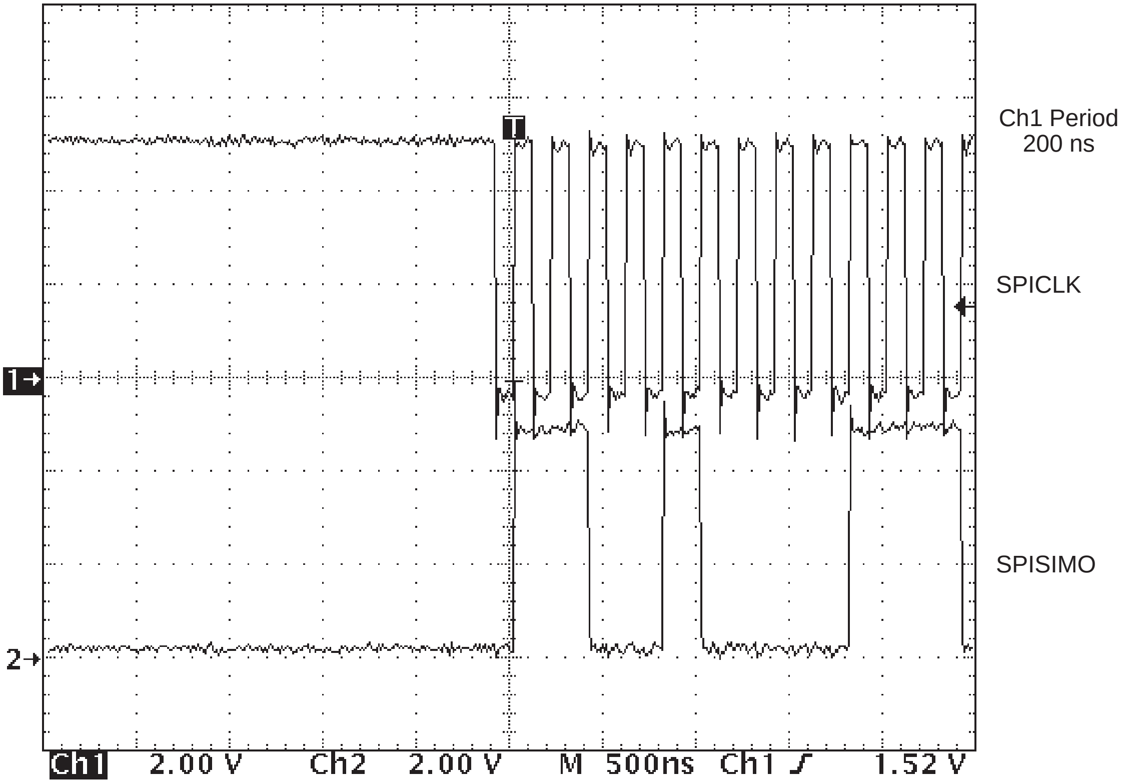


Figure 9–19. CLOCK_POLARITY = 1, CLOCK_PHASE = 1 (Add data transitions are during the falling edge, but delayed by half clock cycle. Inactive level is high.)



Ch1 Period
200 ns

SPICLK

SPISIMO

Figure 9–20. SPISTE Behavior in Master Mode (Master lowers SPISTE during the entire 16 bits of transmission.)

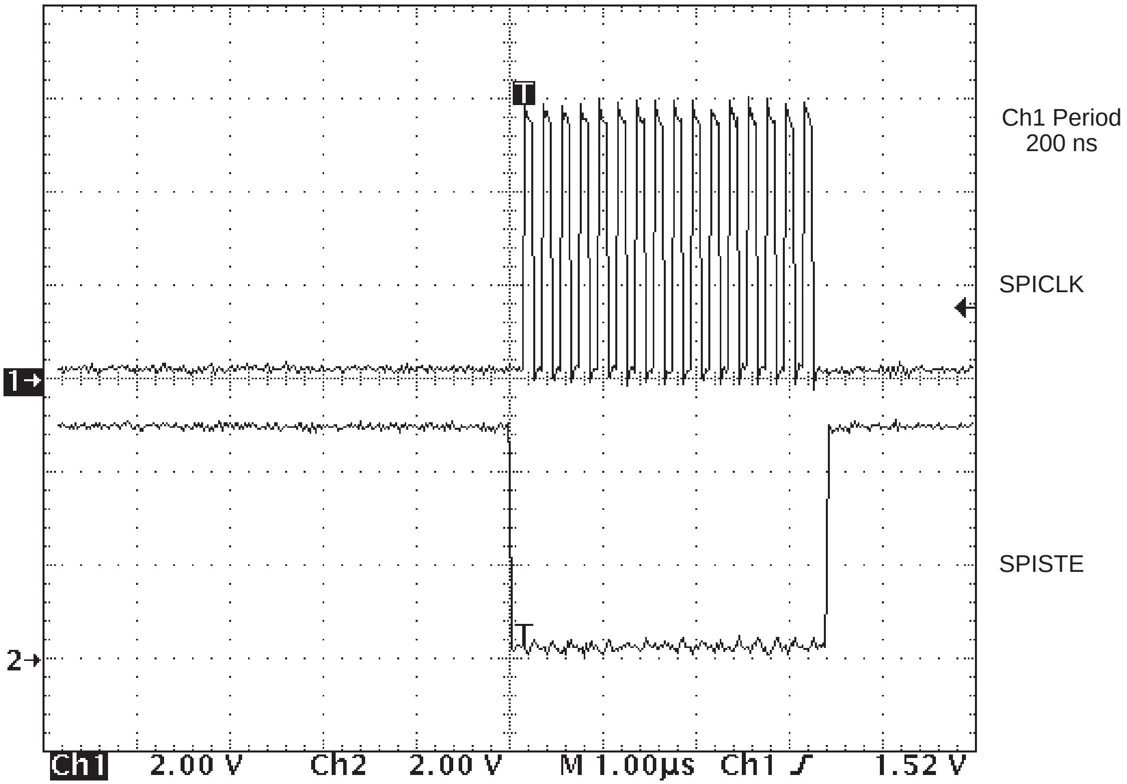
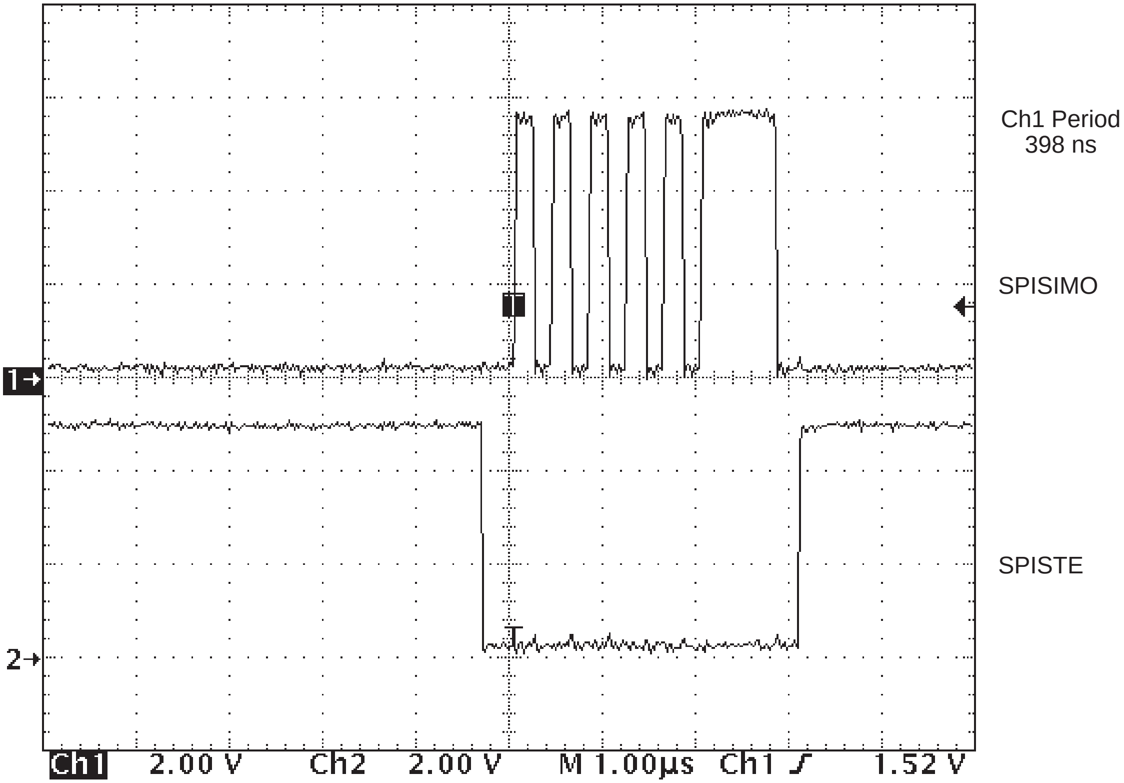


Figure 9–21. SPISTE Behavior in Slave Mode (Slave's SPISTE is lowered during the entire 16 bits of transmission.)



Chapter 10

CAN Controller Module

This chapter describes the controller area network (CAN) module available on some members of the ’24x/’240x family. The interface signals, configuration registers, and mailbox RAM are described in detail; however, the CAN protocol itself is not discussed in depth. For details on the protocol, refer to *CAN Specifications, Version 2.0*, by Robert Bosch GmBH, Germany. The CAN module is a full-CAN controller designed as a 16-bit peripheral and is fully compliant with the CAN protocol, version 2.0B.

Topic **Page**

10.1 Introduction

The CAN peripheral supports the following features:

- Full implementation of CAN protocol, version 2.0B
  - Standard and extended identifiers
  - Data and remote frames
- Six mailboxes for objects of 0- to 8-bytes data length
  - Two receive mailboxes (MBOX0,1), two transmit mailboxes (MBOX4,5)
  - Two configurable transmit/receive mailboxes (MBOX2,3)
- Local acceptance mask registers (LAMn) for mailboxes 0 and 1 and mailboxes 2 and 3
- Programmable bit rate
- Programmable interrupt scheme
- Programmable wake up on bus activity
- Automatic reply to a remote request
- Automatic re-transmission in case of error or loss of arbitration
- Bus failure diagnostic
  - Bus on/off
  - Error passive/active
  - Bus error warning
  - Bus stuck dominant
  - Frame error report
  - Readable error counter
- Self-Test Mode
  - The CAN peripheral operates in a loop back mode
  - Receives its own transmitted message and generates its own acknowledge signal
- Two-Pin Communication
  - The CAN module uses two pins for communication, CANTX and CANRX
  - These two pins are connected to a CAN transceiver chip, which in turn is connected to a CAN bus

10.2 Overview of the CAN Network

The controller area network (CAN) uses a serial multimaster communication protocol that efficiently supports distributed real-time control with a very high level of data integrity, and communication speeds of up to 1 Mbps. The CAN bus is ideal for applications operating in noisy and harsh environments, such as in the automotive and other industrial fields that require reliable communication.

Prioritized messages of up to eight bytes in data length can be sent on a multimaster serial bus using an arbitration protocol and an error-detection mechanism for a high level of data integrity.

10.2.1 CAN Protocol Overview

The CAN protocol supports four different frame types for communication:

- **Data frames** that carry data from a transmitter node to receiver node(s)
- **Remote frames** that are transmitted by a node to request the transmission of a data frame with the same identifier
- **Error frames** that are transmitted by any node on a bus-error detection
- **Overload frames** that provide an extra delay between the preceding and the succeeding data frames or remote frames

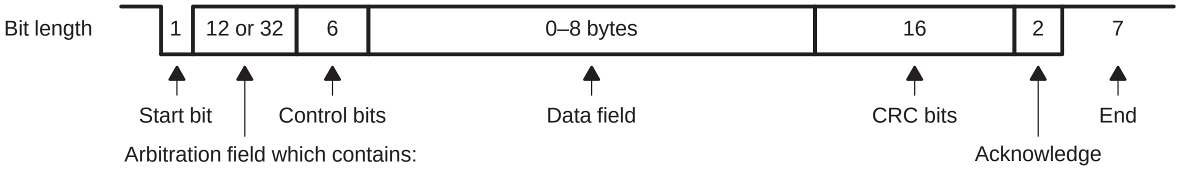
In addition, CAN specification version 2.0B defines two different formats that differ in the length of the identifier field: standard frames with an 11-bit identifier and extended frames with a 29-bit identifier.

CAN standard data frames contain from 44 to 108 bits, and CAN extended data frames contain 64 to 128 bits. Furthermore, up to 23 stuff bits can be inserted in a standard data frame and up to 28 stuff bits in an extended data frame, depending on the data-stream coding. The overall maximum data frame length is 131 bits for a standard frame and 156 bits for an extended frame.

In Figure 10–1, bit fields within the data frame identify:

- Start of the frame
- Arbitration field containing the identifier and the type of message being sent
- Control field containing the number of data
- Up to 8 bytes of data
- Cyclic redundancy check (CRC)
- Acknowledgment
- End-of-frame bits

Figure 10–1. CAN Data Frame



❑ 11-bit identifier + RTR bit for standard frame format

❑ 29-bit identifier + SRR bit + IDE bit + RTR bit for extended frame format

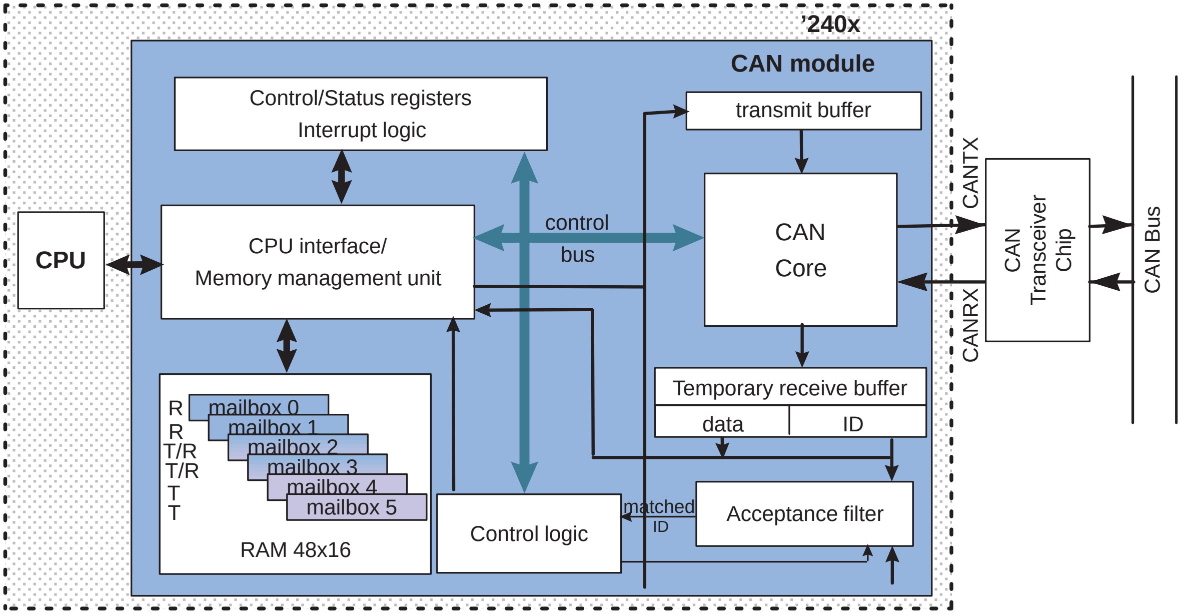
Where: RTR = Remote Transmission Request
SRR = Substitute Remote Request
IDE = Identifier Extension

Note: Unless otherwise noted, numbers are amount of bits in field.

10.2.2 CAN Controller Architecture

Figure 10–2 shows the basic architecture of the CAN controller.

Figure 10–2. TMS320x240x CAN Module Block Diagram



The CAN module is a 16-bit peripheral that accesses the following:

- ❑ Control/status registers
- ❑ Mailbox RAM

Control/Status Registers: The CPU performs 16-bit accesses to the control/status registers. The CAN peripheral always presents full 16-bit data to the CPU bus during read cycles.

Mailbox RAM: Writing/reading from the mailbox RAM is always wordwise (16 bits) and the RAM always presents the full 16-bit word on the bus.

Table 10–1 shows the configuration details of the mailboxes.

Table 10–1. Mailbox Configuration Details

Mailbox	Operating Mode	LAM Used
0	Receive only	LAM0
1	Receive only	LAM0
2	Transmit/Receive (configurable)	LAM1
3	Transmit/Receive (configurable)	LAM1
4	Transmit only	—
5	Transmit only	—

10.2.3 Memory Map

Figure 10–3 shows memory space, and Table 10–2 and Table 10–3 give the register and mailbox locations in the CAN module.

Figure 10–3. TMS320x240x CAN Module Memory Space

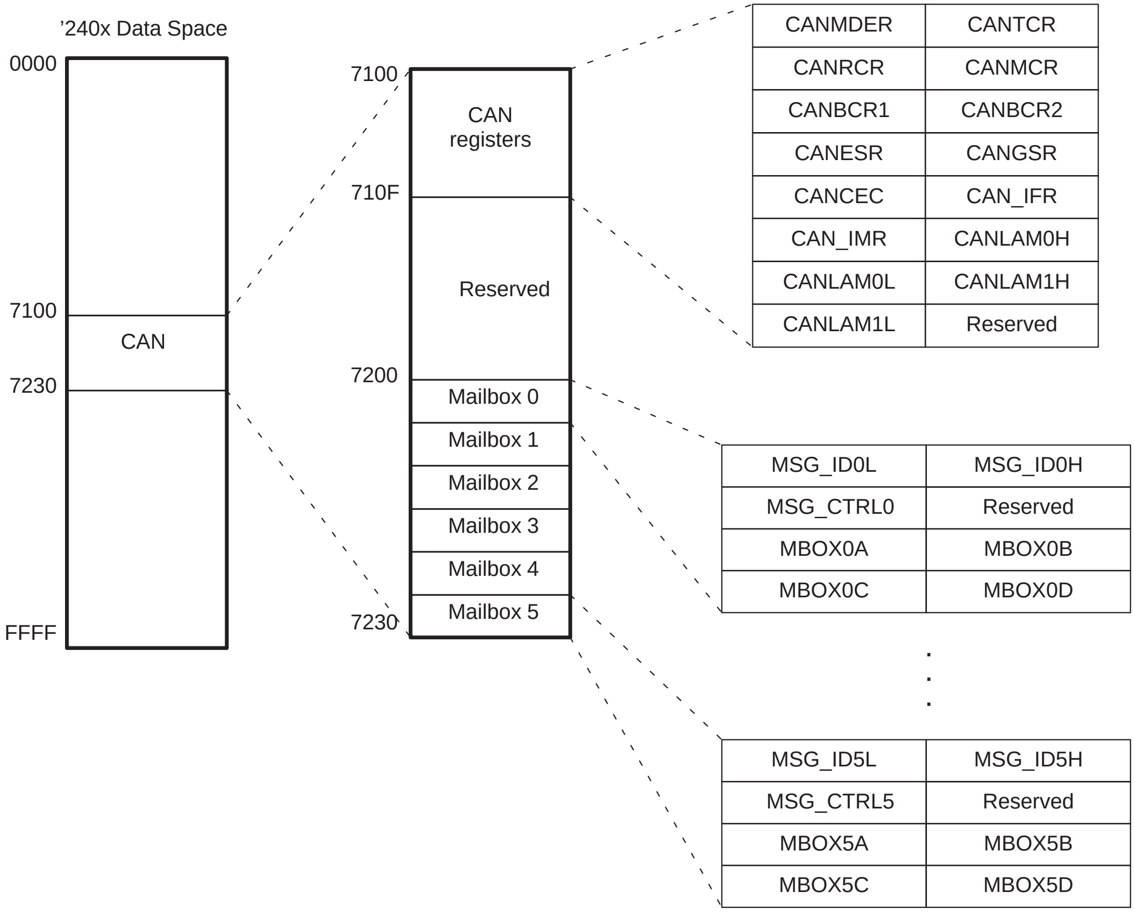


Table 10–2. Register Addresses

Address	Name	Description
7100h	MDER	Mailbox Direction/Enable Register (bits 7 to 0)
7101h	TCR	Transmission Control Register (bits 15 to 0)
7102h	RCR	Receive Control Register (bits 15 to 0)
7103h	MCR	Master Control Register (bits 13 to 6, 1, 0)
7104h	BCR2	Bit Configuration Register 2 (bits 7 to 0)
7105h	BCR1	Bit Configuration Register 1 (bits 10 to 0)
7106h	ESR	Error Status Register (bits 8 to 0)
7107h	GSR	Global Status Register (bits 5 to 0)
7108h	CEC	Transmit and Receive Error Counters (bits 15 to 0)
7109h	CAN_IFR	Interrupt Flag Register (bits 13 to 8, 6 to 0)
710Ah	CAN_IMR	Interrupt Mask Register (bits 15, 13 to 0)
710Bh	LAM0_H	Local Acceptance Mask for MBOX0 and 1 (bits 31, 28 to 16)
710Ch	LAM0_L	Local Acceptance Mask for MBOX0 and 1 (bits 15 to 0)
710Dh	LAM1_H	Local Acceptance Mask for MBOX2 and 3 (bits 31, 28 to 16)
710Eh	LAM1_L	Local Acceptance Mask for MBOX2 and 3 (bits 15 to 0)
710Fh	Reserved	Accesses assert the CAADDRx signal from the CAN peripheral (which will assert an Illegal Address error)

Note: All unimplemented register bits are read as zero; writes have no effect. All register bits are initialized to zero unless otherwise stated in the definition.

The mailboxes are located in one 48 × 16 RAM with 16-bit access and can be written to or read by the CPU (user) or CAN. The CAN write or read access, as well as the CPU read access, needs one clock cycle. The CPU write access needs two clock cycles because the CAN controller performs a read-modify-write cycle; and therefore, inserts one wait state for the CPU.

Table 10–3 shows the mailbox locations in the RAM.

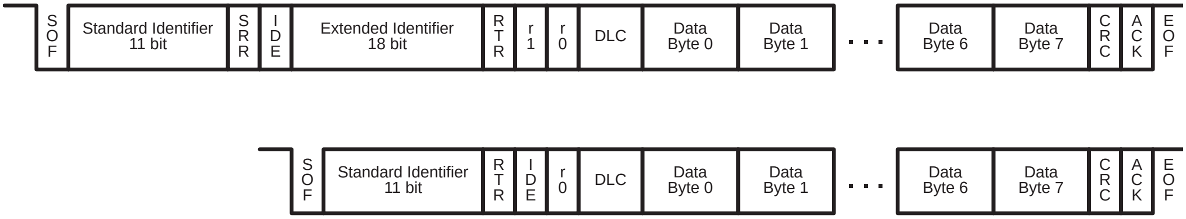
Table 10–3. Mailbox Addresses

	Mailboxes					
Registers	**MBOX_0**	**MBOX_1**	**MBOX_2**	**MBOX_3**	**MBOX_4**	**MBOX_5**
MSG_ID*n*L	7200	7208	7210	7218	7220	7228
MSG_ID*n*H	7201	7209	7211	7219	7221	7229
MSG_CTRL*n*	7202	720A	7212	721A	7222	722A
	Reserved					
MBOX*n*A	7204	720C	7214	721C	7224	722C
MBOX*n*B	7205	720D	7215	721D	7225	722D
MBOX*n*C	7206	720E	7216	721E	7226	722E
MBOX*n*D	7207	720F	7217	721F	7227	722F

10.3 Message Objects

CAN allows messages to be sent, received, and stored by using data frames. Figure 10–4 illustrates the structure of the data frames with extended and standard identifiers.

Figure 10–4. CAN Data Frame



Data frame contains:

- **SOF**: Start of Frame – signifies the start of frame.
- **Identifier**:
  - *Message priority* – determines the priority of the message when two or more nodes are contending for the bus.
  - *Message filtering* – determines if a transmitted message will be received by CAN modules.
- **RTR**: Remote Transmission Request bit – differentiates a *data frame* from a *remote frame*.
- **SRR**: Substitute Remote Request bit – this bit occupies the position as RTR would in a standard frame.
- **IDE**: Identifier Extension bit – differentiates *standard* and *extended* frames.
- **r0, r1**: reserved
- **DLC**: Data Length Code – denotes the number of bytes (0 to 8) in a data frame.
- **Data:** Four 16-bit words are used to store the (maximum) 8-byte data field of a CAN message.

- **CRC**: contains a 16-bit checksum calculated on most parts of the message. This checksum is used for error detection.
- **ACK**: Data Acknowledge
- **EOF**: End of Frame

10.3.1 Mailbox Layout

1) Mailbox RAM:

The mailbox RAM is the area where the CAN frames are stored before they are transmitted, and after they are received. Each mailbox has four 16-bit registers which can store a maximum of 8 bytes (MBOXnA, MBOXnB, MBOXnC, and MBOXnD). Mailboxes that are not used for storing messages may be used as normal memory by the CPU.

2) Message Identifiers:

Each one of the six mailboxes has its own message identifier stored in two 16-bit registers. Figure 10–5 shows the message identifier high word and Figure 10–6 shows the message identifier low word.

Figure 10–5. Message Identifier for High Word Mailboxes 0–5 (MSGIDnH)

15	14	13	12–0
IDE	AME	AAM	IDH[28:16]
RW	RW	RW	RW

Note: R = Read access; W = Write access

Bit 15 **IDE.** Identifier Extension Bit

0 The received message has a standard identifier (11 bits).[†]
The message to be sent has a standard identifier (11 bits).[‡]

1 The received message has an extended identifier (29 bits).[†]
The message to be sent has an extended identifier (29 bits).[‡]

[†] In case of a receive mailbox
[‡] In case of a transmit mailbox

Bit 14 **AME.** Acceptance Mask Enable Bit

0 No acceptance mask will be used. All identifier bits in the received message and the receive MBOX must match in order to store the message.

1 The corresponding acceptance mask is used.

This bit will not be affected by a reception.

This bit is relevant for receive mailboxes only. Hence, it is applicable for MBOX0 and MBOX1 and also for MBOX2 and MBOX3, if they are configured as receive mailboxes. It is a *don't care* for mailboxes 4 and 5.

Bit 13 **AAM.** Auto Answer Mode Bit

0	Transmit mailbox	The mailbox does not reply to remote requests automatically. If a matching identifier is received, it is not stored.
	Receive mailbox	No influence on a receive mailbox.
1	Transmit mailbox	If a matching remote request is received, the CAN Peripheral answers by sending the contents of the mailbox.
	Receive mailbox	No influence on a receive mailbox.

This bit is only used for mailboxes 2 and 3.

Bits 12–0 **IDH[28:16].** Upper 13 Bits of extended identifier. For a standard identifier, the 11-bit identifier will be stored in bits 12 to 2 of the MSGID's upper word.

Figure 10–6. Message Identifier for Low Word Mailboxes 0–5 (MSGIDnL)

15–0
IDL[15:0]
RW

Note: R = Read access; W = Write access

Bits 15–0 **IDL[15:0].** The lower part of the extended identifier is stored in these bits.

3) Message Control Field:

Each one of the six mailboxes has its own "Message Control Field". Figure 10–7 illustrates the layout and default mode of the message control field.

Figure 10–7. Message Control Field (MSGCTRLn)

15–5	4	3–0
Reserved	RTR	DLC[3:0]
	RW	RW

Note: R = Read access; W = Write access

Bits 15–5 **Reserved.**

Bit 4 **RTR.** Remote Transmission Request bit

0	Data frame
1	Remote frame

Bits 3–0 **DLC.** Data Length Code

This value determines how many data bytes are used for transmission or reception.

0000	0 bytes
0010	2 bytes
0100	4 bytes
0110	6 bytes
1000	8 bytes

10.3.2 Message Buffers

Message storage is implemented by RAM. The contents of the storage elements are used to perform the functions of acceptance filtering, transmission, and interrupt handling.

The mailbox module provides six mailboxes, each consisting of 8 bytes of data, 29 identifier bits, and several control bits. Mailboxes 0 and 1 are for reception; mailboxes 2 and 3 are configurable as receive or transmit; and mailboxes 4 and 5 are transmit mailboxes. Mailboxes 0 and 1 share one acceptance mask, while mailboxes 2 and 3 share a different mask.

Note: Unused Message Mailboxes

Unused mailbox RAM may be used as normal memory. Because of this, you must ensure that no CAN function uses the RAM area. This is usually done by disabling the corresponding mailbox or by disabling the CAN function.

10.3.3 Write Access to Mailbox RAM

There are two different types of write accesses to the Mailbox RAM:

1) write access to the identifier of a mailbox

2) write access to the data or control field.

Note:

Write accesses to the identifier can only be accomplished when the mailbox is disabled (MEn = 0 in MDER register).

During accesses to the data field or control field, it is critical that the data does not change while the CAN module is reading it. Therefore, a write access to the data field or control field is disabled for a receive mailbox. For transmit mailboxes, the access is usually denied if the transmit request set (TRS) bit or the transmit request reset (TRR) bit is set. In these cases, a write-denied interrupt flag (WDIF) is asserted. A way to access mailboxes 2 and 3 is to set the change data field request (CDR) bit before accessing the mailbox data.

After the CPU access is finished, the CPU must clear the CDR flag by writing a 0 to it. The CAN module checks for that flag before and after reading the mailbox. If the CDR flag is set during the mailbox checks, the CAN module does not transmit the message but continues to look for other transmit requests. The setting of the CDR flag also stops the write-denied interrupt (WDI) from being asserted.

10.3.4 Transmit Mailbox

Mailboxes 4 and 5 are transmit mailboxes only; whereas, mailboxes 2 and 3 can be configured for reception or transmission.

The CPU stores the data to be transmitted in a mailbox that is configured as a transmit mailbox. After writing the data and the identifier into RAM, and provided the corresponding TRS bit has been set, the message is sent.

If more than one mailbox is configured as a transmit mailbox and more than one corresponding TRS bit is set, the messages are sent one after another, in falling order, beginning with the highest enabled mailbox.

If a transmission fails due to a law of arbitration or an error, the message transmission will be re-attempted.

10.3.5 Receive Mailbox

Mailboxes 0 and 1 are receive-only mailboxes. Mailboxes 2 and 3 can be configured for reception or transmission.

The identifier of each incoming message is compared to the identifiers held in the receive mailboxes by using the appropriate identifier mask. When equality is detected, the received identifier, the control bits, and the data bytes are written into the matching RAM location. At the same time, the corresponding receive message pending (RMPn) bit is set and a mailbox interrupt (MIFx) is generated if enabled. If the current identifier does not match, the message is not stored. The RMPn bit has to be reset by the CPU after reading the data.

If a second message has been received for this mailbox and the RMP bit is already set, the corresponding receive message lost (RML) bit is set. In this case, the stored message is overwritten with the new data if the overwrite protection control (OPC) bit is cleared. Otherwise, the next mailboxes are checked.

Note:

For the mailbox interrupt flag (MIFn) bits in the CAN_IFR register to be set, the corresponding bits in the CAN_IMR register must be enabled. If “polling” is desired to complete transmission or reception of messages (as opposed to interrupts), the following bits must be used:

❑ For transmission: TAn bits in the TCR register

❑ For reception: RMPn bits in the RCR register

10.3.6 Handling of Remote Frames

Remote frame handling can only be done with mailboxes 0 to 3; mailboxes 4 and 5 cannot handle remote frames.

Receiving a Remote Request

If a remote request is received (the incoming message has the remote transmission request bit [RTR] = 1), the CAN module compares the identifier to all identifiers of the mailboxes using the appropriate masks in descending order starting with the highest mailbox number.

In case of a matching identifier with the message object configured as a transmit mailbox and the auto-answer mode bit (AAM) in the message set, the message object is marked to be sent (TRS bit is set). See Figure 10–8 (A).

In case of a matching identifier with the message object configured as a transmit mailbox and the AAM bit not set, the message is not received. See Figure 10–8 (B).

After finding a matching identifier in a send mailbox, no further compare is done.

In case of a matching identifier with the message object configured as a receive mailbox, the message is handled like a data frame and the RMP bit in the receive control register (RCR) is set. The CPU then has to decide how to handle the situation. See Figure 10–8 (E).

If the CPU wants to change the data in a message object that is configured as a remote frame mailbox (AAM bit set), it has to set the mailbox number (MBNR) in the master control register and the CDR in the master control register first.

The CPU may then perform the access and clear the CDR to tell the CAN module that the access is finished. Until the CDR is cleared, the transmission of this mailbox is not performed. Since the TRS bit is not affected by the CDR, a pending transmission is stacked after the CDR is cleared. Thus, the newest data will be sent.

In order to change the identifier in the mailbox, the message object must be disabled first (ME bit in the MDER = 0).

Sending a Remote Request

If the CPU wants to request data from another node, it may configure the message object as a receive mailbox (only mailboxes 2 and 3) and set the TRS bit. See Figure 10–8 (F). In this case, the module sends a remote frame request and receives the data frame in the same mailbox that sent the request. Therefore, only one mailbox is necessary to do a remote request.

To summarize: A mailbox can be configured in four different ways:

❑ Transmit mailbox (mailboxes 4 and 5 or 2 and 3 configured as transmit) can only transmit messages.

❑ Receive mailbox (mailboxes 0 and 1) can only receive messages.

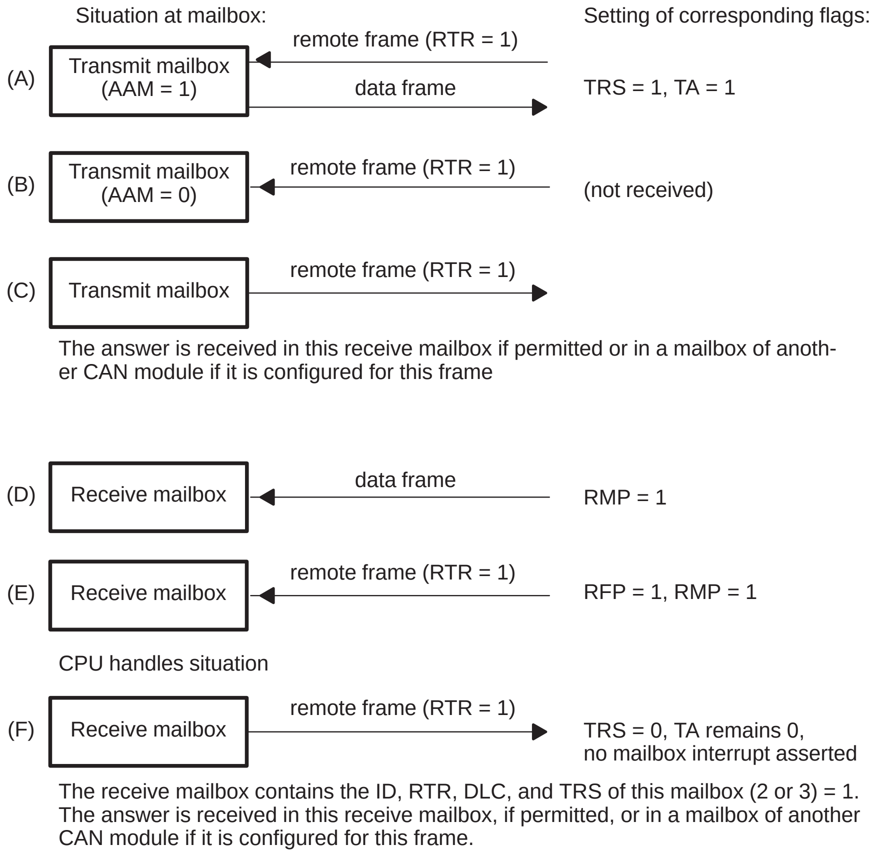
❑ Mailboxes 2 and 3 configured as receive mailboxes can transmit a remote request frame and wait for the corresponding data frame if the TRS bit is set.

❑ Mailboxes 2 and 3 configured as transmit mailboxes can transmit a data frame wherever a remote request frame is received for the corresponding identifier, if the AAM bit is set.

Note:

After successful transmission of a remote frame, the TRS bit is reset but no transmit acknowledge (TA) or mailbox interrupt flag is set.

Figure 10–8. Remote Frame Requests


Situation at mailbox:
Setting of corresponding flags:
(A)
Transmit mailbox (AAM = 1)
remote frame (RTR = 1)
data frame
TRS = 1, TA = 1
(B)
Transmit mailbox (AAM = 0)
remote frame (RTR = 1)
(not received)
(C)
Transmit mailbox
remote frame (RTR = 1)
The answer is received in this receive mailbox if permitted or in a mailbox of another CAN module if it is configured for this frame
(D)
Receive mailbox
data frame
RMP = 1
(E)
Receive mailbox
remote frame (RTR = 1)
RFP = 1, RMP = 1
CPU handles situation
(F)
Receive mailbox
remote frame (RTR = 1)
TRS = 0, TA remains 0,
no mailbox interrupt asserted
The receive mailbox contains the ID, RTR, DLC, and TRS of this mailbox (2 or 3) = 1.
The answer is received in this receive mailbox, if permitted, or in a mailbox of another CAN module if it is configured for this frame.

10.3.7 Acceptance Filter

The identifier of the incoming message is first compared to the message identifier of the receive mailbox (which is stored in the mailbox in MSGIDnH and MSGIDnL registers). Then the appropriate acceptance mask is used to mask out the bits of the identifier that should not be compared. The local acceptance mask can be disabled by setting the acceptance mask enable (AME) bit to 0 in the message identifier high word (MSGIDn) field.

Local Acceptance Mask (LAM)

The local acceptance filtering allows the user to locally mask (that is, treat as a *don't care*) any identifier bit of the incoming message.

Local acceptance mask register LAM1 is used for mailboxes 2 and 3 while local acceptance mask register LAM0 is used for mailboxes 0 and 1. During a reception, mailboxes 3 and 2 are checked before mailboxes 1 and 0. Figure 10–9 illustrates the LAMn_H high word and Figure 10–10 illustrates the LAMn_L low word.

Figure 10–9. Local Acceptance Mask Register n (0, 1) High Word (LAMn_H)

15	14–13	12–0
LAMI	Reserved	LAMn[28:16]
RW-0		RW-0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bit 15 **LAMI.** Local acceptance mask identifier extension bit.

0 The identifier extension bit stored in the mailbox determines which messages are received (standard or extended).

1 Standard and extended frames can be received. In case of an extended frame, all 29 bits of the identifier are stored in the mailbox and all 29 bits of the global acceptance mask register are used for the filter. In case of a standard frame, only the first eleven bits (bits 12–2 of LAMn_H) of the identifier and the local acceptance mask are used.

Bits 14–13 **Reserved.**

Bits 12–0 **LAMn[28:16].** Upper 13 bits of the local acceptance mask.

0 Received identifier bit value must match the identifier bit of the receive mailbox. For example, if bit 27 of LAM is zero, then bit 27 of the transmitted MSGID and bit 27 of the receive mailbox MSGID must be the same.

1 Accept a 0 or a 1 (*don't care*) for the corresponding bit of the receive identifier.

Figure 10–10. Local Acceptance Mask Register n (0, 1) Low Word (LAMn_L)

15–0
LAMn[15:0]
RW-0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 15–0 **LAMn[15:0].** Lower part of the local acceptance mask. These bits enable the masking of any identifier bit of an incoming message.

0 Received identifier bit value must match the identifier bit of the receive mailbox.

1 Accept a 0 or a 1 (don't care) for the corresponding bit of the receive identifier.

10.4 CAN Control Registers

The control register bits allow mailbox functions to be manipulated. Each register performs a specific function, such as enabling or disabling the mailbox, controlling the transmit/receive mail function, and handling interrupts.

10.4.1 Mailbox Direction/Enable Register (MDER)

The Mailbox Direction/Enable register (MDER) consists of the Mailbox Enable (ME) and the Mailbox Direction (MD). In addition to enabling/disabling the mailboxes, MDER is used to select the direction (transmit/receive) for mailboxes 2 and 3. Mailboxes that are disabled may be used as additional memory for the DSP. Figure 10–11 illustrates this register.

Figure 10–11. Mailbox Direction/Enable Register (MDER) — Address 7100h

15–8
Reserved

7	6	5	4	3	2	1	0
MD3	MD2	ME5	ME4	ME3	ME2	ME1	ME0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 15–8 **Reserved.**

Bits 7–6 **MDn.** Mailbox direction for mailbox n. Mailboxes 2 and 3 can be configured as a transmit or receive mailbox.

Mailbox direction bits are defined as follows:

0	Transmit mailbox
1	Receive mailbox

After power-up, all bits are cleared.

Bits 5–0 **MEn.** Mailbox-enable for mailbox n. Each mailbox can be enabled or disabled. If the bit MEn is 0, the corresponding mailbox n is disabled. The mailbox must be disabled before writing to any identifier field.

If the corresponding bit in ME is set, the write access to the identifier of a message object is denied and the mailbox is enabled for the CAN module.

Mailboxes that are disabled may be used as additional memory for the DSP.

Mailbox enable bits are defined as follows:

0	Disable mailbox
1	Enable mailbox

10.4.2 Transmit Control Register (TCR)

The transmit control register (TCR) contains bits that control the transmission of messages (see Figure 10–12).

The control bits to set or reset a transmission request (TRS and TRR, respectively) can be written independently. In this way, a write access to these registers does not set bits that were reset because of a completed transmission.

After power-up, all bits are cleared.

Figure 10–12. Transmission Control Register (TCR) — Address 7101h

15	14	13	12	11	10	9	8
TA5	TA4	TA3	TA2	AA5	AA4	AA3	AA2
RC-0	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0

7	6	5	4	3	2	1	0
TRS5	TRS4	TRS3	TRS2	TRR5	TRR4	TRR3	TRR2
RS-0	RS-0	RS-0	RS-0	RS-0	RS-0	RS-0	RS-0

Note: R = Read access; C = Clear; S = Set only; value following dash (–) = value after reset

TAn: Transmission Acknowledge *(for mailbox n)*

If the message in mailbox n was sent successfully, bit TAn is set.

Bits TAn are reset by writing a 1 from the CPU. This also clears the interrupt if an interrupt was generated. Writing a 0 has no effect. If the CPU tries to reset the bit while the CAN tries to set it, the bit is set.

These bits set a mailbox interrupt flag (MIFx) in the IF register. The MIFx bits initiate a mailbox interrupt if enabled; that is, if the corresponding interrupt mask bit in the IM register is set.

AAn: Abort Acknowledge *(for mailbox n)*

If transmission of the message in mailbox n is aborted, bit AAn is set and the AAIF bit in the IF register is set. The AAIF bit generates an error interrupt if enabled.

Bits AAn are reset by writing a 1 from the CPU. Writing a 0 has no effect. If the CPU tries to reset a bit and the CAN tries to set the bit at the same time, the bit is set.

TRSn: Transmission Request Set *(for mailbox n)*

If TRSn is set, write access to the corresponding mailbox is denied, and the message in mailbox *n* will be transmitted. Several TRS bits can be set simultaneously.

TRS bits can be set by the CPU (user) or the CAN module and reset by internal logic. If the CPU tries to set a bit while the CAN tries to clear it, the bit is set. TRS bits are set by the user writing a 1. Writing a 0 has no effect.

In the event of a remote frame request, the TRS bits are set by the CAN module for mailboxes 2 and 3.

The TRSn bits are reset after a successful or an aborted transmission (if an abort is requested).

A write to a mailbox with TRS set will have no effect and will generate the WDIF interrupt if enabled. A successful transmission initiates a mailbox interrupt, if enabled.

TRS bits are used for mailboxes 4 and 5, and also for 2 and 3 if they are configured for transmission.

TRRn: Transmission Request Reset *(for mailbox n)*

TRR bits can only be set by the CPU (user) and reset by internal logic. In case the CPU tries to set a bit while the CAN module tries to clear it, the bit is set. The TRR bits are set by the user writing a 1. Writing a 0 has no effect.

If TRRn is set, write access to the corresponding mailbox*n* is denied. A write access will initiate a WDIF interrupt, if enabled. If TRRn is set and the transmission which was initiated by TRSn is not currently processed, the corresponding transmission request will be cancelled. If the corresponding message is currently processed, this bit is reset in the event of:

1) A successful transmission
2) An abort due to a lost arbitration
3) An error condition detected on the CAN bus line

If the transmission is successful, the status bit TAn is set. If the transmission is aborted, the corresponding status bit AAn is set. In case of an error condition, an error status bit is set in the ESR.

The status of the TRR bits can be read from the TRS bits. For example, if TRS is set and a transmission is ongoing, TRR can only be reset by the actions described above. If the TRS bit is reset and the TRR bit is set, no effect occurs because the TRR bit will be immediately reset.

10.4.3 Receive Control Register (RCR)

The receive control register (RCR) contains the bits which control the reception of messages and remote frame handling.

Figure 10–13. Receive Control Register (RCR) — Address 7102h

15	14	13	12	11	10	9	8
RFP3	RFP2	RFP1	RFP0	RML3	RML2	RML1	RML0
RC-0	RC-0	RC-0	RC-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
RMP3	RMP2	RMP1	RMP0	OPC3	OPC2	OPC1	OPC0
RC-0	RC-0	RC-0	RC-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access; W = Write access; C = Clear; value following dash (–) = value after reset

RFPn: Remote Frame Pending Register *(for mailbox n)*

Whenever a remote frame request is received by the CAN Peripheral, the corresponding bit RFPn is set.

It may be cleared by the CPU if the TRSn is not set; otherwise, it is reset automatically. If the CPU tries to reset a bit and the CAN Peripheral tries to set the bit at the same time, the bit is cleared.

If the AAM bit in the MSGIDn register is not set (and thus no answer is sent automatically), the CPU must clear bit RFPn after handling the event.

If the message is sent successfully, RFPn is cleared by the CAN Peripheral.

The CPU cannot interrupt an ongoing transfer.

RMLn: Receive Message Lost *(for mailbox n)*

If an old message is overwritten by a new one in mailbox n, bit RMLn is set. RMLn is not set in mailboxes that have the OPCn bit set. Thus, a message may be lost without notification.

These bits can only be reset by the CPU and can be set by the internal logic. They can be cleared by writing a 1 to RMPn. If the CPU tries to reset a bit and the CAN tries to set the bit at the same time, the bit is set.

If one or more RML bits in the RCR register are set, the RMLIF in the IF register is also set. This may initiate an interrupt if the RMLIM bit in the IM register is set.

RMPn: Receive Message Pending *(for mailbox n)*

If a received message is stored in a mailbox n, the bit RMPn is set.

The RMP bits can only be reset by the CPU and are set by the CAN internal logic. The bits RMPn and RMLn are cleared by writing a 1 to the RMPn bit at the corresponding bit location. If the CPU tries to reset a bit and the CAN tries to set the bit at the same time, the bit is set.

A new incoming message will overwrite the stored one if the OPCn bit is cleared. If not, the next mailboxes are checked for a matching identifier. When the old message is overwritten, the corresponding status bit RMLn is set.

The RMP bits in the RCR register set the mailbox interrupt flag (MIFx) bit in the IF register if the corresponding interrupt mask bit in the IM register is set. The MIFx flag initiates a mailbox interrupt if enabled.

OPCn: Overwrite Protection Control *(for mailbox n)*

If there is an overflow condition for mailbox n, the new message is stored/ignored depending on the OPCn value. If the corresponding bit OPCn is set to 1, the old message is protected against being overwritten by the new message. Thus, the next mailboxes are checked for a matching identifier. If no other mailbox is found, the message is lost without further notification. If bit OPCn is not set, the old message is overwritten by the new one.

10.4.4 Master Control Register (MCR)

The Master Control Register is used to control the behavior of the CAN core module.

Figure 10–14. Master Control Register (MCR) — Address 7103h

15–14	13	12	11	10	9	8
Reserved	SUSP	CCR	PDR	DBO	WUBA	CDR
	RW-0	RW-1	RW-0	RW-0	RW-0	RW-0

7	6	5–2	1–0
ABO	STM	Reserved	MBNR[1:0]
RW-0	RW-0		RW-0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 15–14 **Reserved**

Bit 13 **SUSP.** Action on emulator suspend. The value of SUSP bit has no effect on the receive mailboxes.

0 *Soft mode*. The peripheral shuts down during suspend after the current transmission is completed.

1 *Free mode*. The peripheral continues to run in suspend.

Bit 12 **CCR.** Change Configuration Request

0 The CPU requests normal operation. It also exits the bus-off state after the obligatory bus-off recovery sequence.

1 The CPU requests write access to the bit configuration registers (BCRn). Flag CCE in the GSR indicates if the access is granted. CCR must be set while writing to bit timing registers BCR1 and BCR2. This bit will automatically be set to 1 if the bus-off condition is valid and the ABO is not set. Thus, it has to be reset to exit the bus-off mode.

Bit 11 **PDR.** Power-Down Mode Request

Before the CPU enters its IDLE mode (if IDLE shuts off the peripheral clocks), it must request a CAN power down by writing to the PDR bit. The CPU must then poll the PDA bit in the GSR, and enter IDLE only after PDA is set.

0 The power-down mode is not requested (normal operation).

1 The power-down mode is requested.

Bit 10 **DBO.** Data Byte Order

0 The data is received or transmitted in the following order: Databyte 0,1,2,3,4,5,6,7.

1 The data is received or transmitted in the following order: Databyte 3,2,1,0,7,6,5,4.

Note:

The DBO bit is used to define the order in which the data bytes are stored in the mailbox when received and in which the data bytes are transmitted. Byte 0 is the first byte in the message and Byte 7 is the last one as shown in the figure of the CAN message (Figure 10–4).

Bit 9 **WUBA.** Wake Up on Bus Activity

0 The module leaves the power-down mode only after the user writing a 0 to clear PDR.

1 The module leaves the power-down mode when detecting any dominant value on the CAN bus.

Bit 8 **CDR.** Change Data Field Request

The CDR bit is applicable for mailboxes 2 and 3 only *and* in the following situation: 1) either (or both) of these mailboxes are configured for transmission and 2) the corresponding AAM bit is set.

0 The CPU requests normal operation.

1 The CPU requests write access to the data field of the mailbox in MBNR (located also in MCR). The CDR bit must be cleared by the CPU after accessing the mailbox. The CAN module does not transmit the mailbox if the CDR is set. This is checked by the state machine before and after it reads the data from the mailbox to store it in the transmit buffer.

Bit 7 **ABO.** Auto Bus On

0 The bus-off state may only be left after 128 × 11 consecutive recessive bits on the bus and after having reset the CCR bit.

1 After the bus-off state, the module goes back to the bus-on state after 128 × 11 consecutive recessive bits.

Bit 6 **STM.** Self Test Mode

0 The module is in normal mode.

1 The module is in Self Test mode. In this mode, the CAN module generates its own ACK signal. Thus, it enables operation without a bus connected to the module. The message is not sent but read back and stored in the appropriate mailbox. The remote frame handling with Auto Answer mode set is *not* implemented in STM.

Bits 5–2 **Reserved.**

Bits 1–0 **MBNR.** Mailbox Number (for CDR bit assertion)

The CPU requests a write access to the data field for the mailbox having this number and configured for Remote Frame Handling. These are mailboxes 2 (10) or 3 (11), but not 0, 1, 4 or 5.

10.4.5 Bit Configuration Registers (BCR*n*)

The bit configuration registers (BCR1 and BCR2) are used to configure the CAN node with the appropriate network timing parameters. These registers must be programmed before using the CAN module and are writeable only in configuration mode. The CCR bit must be set to put the CAN module in configuration mode.

Note:

To avoid unpredictable behavior, BCR1, 2 should never be programmed with values not allowed by the CAN protocol specification.

Figure 10–15. Bit Configuration Register 2 (BCR2) — Address 7104h

15–8
Reserved

7–0
BRP[7:0]
RW-0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 15–8 **Reserved.**

Bits 7–0 **BRP.** Baud Rate Prescaler

Bits 7:0 of this field specify the duration of a time quantum (TQ) in CAN module system clock units. The length of one TQ is defined by:

$$TQ = \frac{BRP+\ \ 1}{I_{CLK}}$$

where I_{CLK} is the frequency of the CAN module system clock,which is the same as CLKOUT.

Figure 10–16. Bit Configuration Register 1 (BCR1) — Address 7105h

15–11	10	9–8
Reserved	SBG	SJW[1:0]
	RW-0	RW-0

7	6–3	2–0
SAM	TSEG1-[3:0]	TSEG2-[2:0]
RW-0	RW-0	RW-0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 15–11 **Reserved.**

Bit 10 **SBG.** Synchronization on both edges

0 The CAN module resynchronizes on the falling edge only.

1 The CAN module resynchronizes on both rising and falling edges.

Bits 9–8 **SJW.** Synchronization jump width

SJW indicates by how many units of TQ a bit is allowed to be lengthened or shortened when resynchronizing with the receive data stream on the CAN bus. The synchronization is performed either with the falling edge (SBG = 0) or with both edges (SBG = 1) of the bus signal. SJW is programmable from 1 to 4 TQ.

Bit 7 **SAM.** Sample point setting

This parameter sets the number of samples used by the CAN module to determine the actual level of the CAN bus. When the SAM bit is set, the level determined by the CAN bus corresponds to the result from the majority decision of the last three values. The sample points are at the sample point and twice before with a distance of 1/2 TQ.

0 The CAN module samples only once.

1 The CAN module samples three times and makes a majority decision.

Bits 6–3 **TSEG1[3:0].** Time segment 1.

This parameter specifies the length of the TSEG1 segment in TQ units.

TSEG1 combines PROP_SEG and PHASE_SEG1 segments (CAN protocol).

$$\text{TSEG1} = \text{PROP\_SEG} + \text{PHASE SEG1}.$$

The value of TSEG1 is programmable from 3 to 16 TQ and must be greater than or equal to TSEG2.

Bits 2–0 **TSEG2[2:0].** Time segment 2.

TSEG2 defines the length of PHASE_SEG2 in TQ units.

When the resynchronization on falling edge only is used (SBG = 0), the minimum TSEG2 value allowed is calculated as follows:

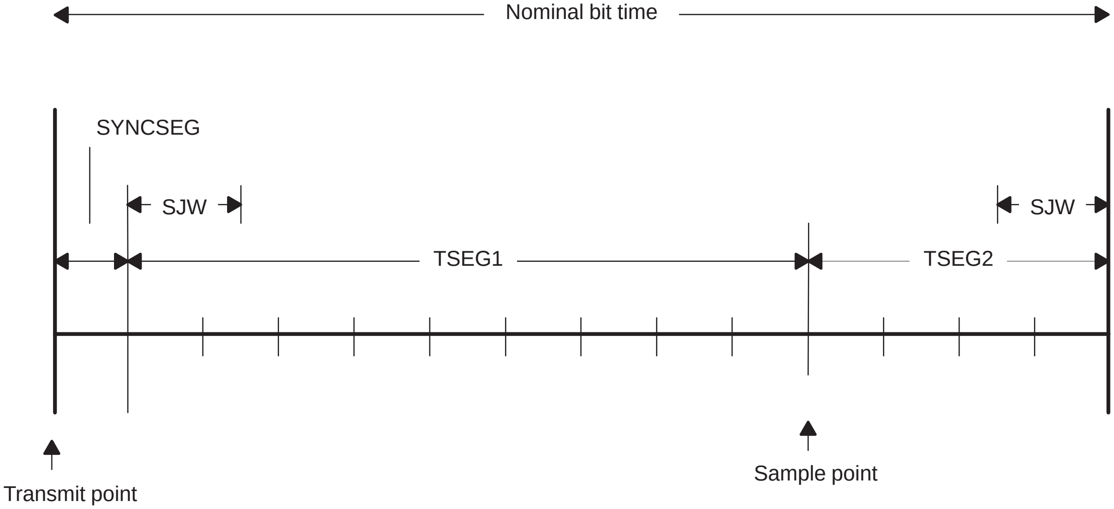
$$\text{TSEG2}_{\text{min}} = 1 + \text{SJW}.$$

The value of TSEG2 is programmable from 2 to 8 TQ in compliance with the formula:

$$(\text{SJW} + \text{SBG} + 1) \leq \text{TSEG2} \leq 8.$$

CAN Bit Timing

Figure 10–17. CAN Bit Timing



Baud rate is calculated as follows (in bits per second):

$$\text{Baud rate} = \frac{I_{CLK}}{(BRP + 1) \times \text{Bit Time}}$$

where, Bit Time = number of TQ per bit
Bit Time = (TSEG1 + 1) + (TSEG2 + 1) + 1
I_{CLK} = CAN module system clock frequency (same as CLKOUT)
BRP = Baud rate prescaler

Table 10–4. CAN Bit Timing Examples for I_{CLK} = 20 MHz

TSEG1	TSEG2	Bit Time	BRP	SJW	SBG	Baud Rate
4	3	10	1	1 or 2	1	1 Mbit/s
14	6	23	8	4	1	0.096 Mbit/s
3	2	8	0	1	0	2.5 Mbit/s

10.5 Status Registers

The two status registers are the global status register (GSR) and the error status register (ESR). As indicated by their names, GSR provides information for all functions of the CAN peripheral and ESR provides information about any type of error encountered.

10.5.1 Global Status Register (GSR)

Figure 10–18. Global Status Register (GSR) — Address 7107h

15–8
Reserved

7–6	5	4	3	2	1	0
Reserved	SMA	CCE	PDA	Rsvd	RM	TM
	R-0	R-1	R-0		R-0	R-0

Note: R = Read access; value following dash (–) = value after reset

Bits 15–6 **Reserved.**

Bit 5 **SMA.** Suspend Mode Acknowledge

0 The CAN peripheral is not in suspend mode.

1 The CAN peripheral has entered suspend mode.

This bit is set after a latency of 1 clock cycle up to the length of one frame after the SUSPEND signal is activated.

Bit 4 **CCE.** Change Configuration Enable

0 Write access to the configuration registers is denied.

1 The CPU has write access to the configuration registers BCR while CCR is set. Access is granted after reset or when the CAN module reaches the idle state.

This bit is set after a latency of 1 clock cycle up to the length of one frame.

Bit 3 **PDA.** Power-Down Mode Acknowledge

Before the CPU enters its IDLE mode (to potentially shut off ALL device clocks), it must request a CAN power down by writing to the PDR bit in MCR. The CPU must then poll the PDA bit and enter IDLE only after PDA is set.

0 Normal operation.

1 The CAN peripheral has entered the power-down mode.

This bit is set after a latency of 1 clock cycle up to the length of one frame.

Bit 2 **Reserved.**

Bit 1 **RM.** The CAN module is in the Receive Mode.

This bit reflects what the CBM is actually doing regardless of mailbox configuration.

0	The CAN core module is not receiving a message.
1	The CAN core module is receiving a message.

Bit 0 **TM.** The CAN module is in the Transmit Mode.

This bit reflects what the CBM is actually doing regardless of mailbox configuration.

0	The CAN core module is not transmitting a message.
1	The CAN core module is transmitting a message.

10.5.2 Error Status Register (ESR)

The error status register (see Figure 10–19) is used to display errors that occurred during operation. Only the first error is stored. Subsequent errors do not change the status of the register. These registers are cleared by writing a 1 to them except for the SA1 flag, which is cleared by any recessive bit on the bus.

Bits 8 to 3 are error bits that can be read and cleared by writing a 1 to them. Bits 2 to 0 are status bits that cannot be cleared, only read.

Figure 10–19. Error Status Register (ESR) — Address 7106h

15–9	8
Reserved	FER
	RC-0

7	6	5	4	3	2	1	0
BEF	SA1	CRCE	SER	ACKE	BO	EP	EW
RC-0	RC-1	RC-0	RC-0	RC-0	R-0	R-0	R-0

Note: R = Read access; C = Clear; value following dash (–) = value after reset

Bits 15–9 **Reserved.**

Bit 8 **FER.** Form Error Flag

0	The CAN module was able to send and receive correctly.
1	A Form Error occurred on the bus. This means that one or more of the fixed-form bit fields had the wrong level on the bus.

Bit 7 **BEF.** Bit Error Flag

0 The CAN module was able to send and receive correctly.

1 The received bit does not match the transmitted bit outside of the arbitration field; or during transmission of the arbitration field, a dominant bit was sent but a recessive bit was received.

Bit 6 **SA1.** Stuck at dominant Error

0 The CAN module detected a recessive bit.

1 The SA1 bit is always 1 after a hardware or a software reset or a bus-off condition. The CAN module did not detect a recessive bit.

Bit 5 **CRCE.** CRC Error

0 The CAN module did not receive a wrong CRC.

1 The CAN module received a wrong CRC.

Bit 4 **SER.** Stuff Error

0 No stuff bit error occurred.

1 The stuff bit rule was violated.

Bit 3 **ACKE.** Acknowledge Error

0 The CAN module received an acknowledge.

1 The CAN module did not receive an acknowledge.

Bit 2 **BO.** Bus Off Status

0 Normal operation.

1 There is an abnormal rate of error occurrences on the CAN bus. This condition occurs when the transmit error counter TEC has reached the limit of 256. While in bus-off status, no messages can be received or transmitted. This state is only exited by clearing the CCR bit in the Master Control Register (MCR) or if the *Auto Bus-On* bit in the Master Control Register is set. After leaving the bus-off state, the error counters are cleared.

Bit 1 **EP.** Error Passive Status

0 The CAN module is not in error-passive mode.

1 The CAN module is in error-passive mode.

Bit 0 **EW.** Warning Status

0 The values of both error counters are less than 96.

1 At least one of the error counters reached the warning level of 96.

10.5.3 CAN Error Counter Register (CEC)

The CAN module contains two error counters: the receive error counter (REC) and the transmit error counter (TEC). The values of both counters can be read from the CEC register via the CPU interface.

Figure 10–20. CAN Error Counter Register (CEC) — Address 7108h

15–8
TEC[7:0]
R-0

7–0
REC[7:0]
R-0

Note: R = Read access; value following dash (–) = value after reset

After exceeding the error passive limit (128), REC is not increased any further. When a message is received correctly, the counter is set again to a value between 119 and 127. After reaching the bus-off status, TEC is undefined, while REC is cleared and its function is changed: It will be incremented after every 11 consecutive recessive bits on the bus. These 11 bits correspond to the gap between two telegrams on the bus. If the receive counter reaches 128, the module changes automatically back to the status bus-on if bit ABO in MCR is set. Otherwise, it changes when the recovery sequence of 11 × 128 bits has finished and the CCR bit in the MCR register is reset by the DSP. All internal flags are reset and the error counters are cleared. The configuration registers keep the programmed values.

After the power-down mode, the error counters stay unchanged. They are cleared when entering the configuration mode.

10.6 Interrupt Logic

There are two interrupt requests from the CAN peripheral to the peripheral interrupt expansion (PIE) controller, the mailbox interrupt and the error interrupt. Both interrupts can assert either a high-priority request or a low-priority request to the CPU. The following events may initiate an interrupt:

- ❑ Mailbox Interrupt
  - ■ A message was transmitted or received successfully. This event asserts the Mailbox interrupt.
- ❑ Abort Acknowledge Interrupt
  - ■ A send transmission was aborted. This event asserts the Error interrupt.
- ❑ Write Denied Interrupt
  - ■ The CPU tried to write to a mailbox but was not allowed to. This event asserts the Error interrupt.
- ❑ Wake-up Interrupt
  - ■ After wake-up, this interrupt is generated. This event asserts the Error interrupt, even when clocks are not running.
- ❑ Receive Message Lost Interrupt
  - ■ An old message was overwritten by a new one. This event asserts the Error interrupt.
- ❑ Bus Off Interrupt
  - ■ The CAN module enters the bus off state. This event asserts the Error interrupt.
- ❑ Error Passive Interrupt
  - ■ The CAN module enters the error passive mode. This event asserts the Error interrupt.
- ❑ Warning Level Interrupt
  - ■ One or both of the error counters is greater than or equal to 96. This event asserts the Error interrupt.

Note: While servicing a CAN interrupt, the user should check all the bits in the CAN_IFR register to ascertain if more than one bit has been set. The corresponding ISRs should be executed for all the set bits. This must be done since the core interrupt will be asserted only once, even if multiple bits are set in the CAN_IFR register.

10.6.1 CAN Interrupt Flag Register (CAN_IFR)

The interrupt flag bits are set if the corresponding interrupt condition occurs. The appropriate mailbox interrupt request is asserted only if the corresponding interrupt mask in CAN_IMR register is set. The peripheral interrupt request stays active until the interrupt flag is cleared by the CPU by writing a 1 to the appropriate bit. An interrupt acknowledge does not clear the interrupt flags. The MIFx flags cannot be cleared by writing to the IF register; instead, they must be cleared by writing a 1 to the appropriate TA bit in the TCR register for a transmit mailbox (mailboxes 2 to 5), or the RMP bit in the RCR register for the receive mailbox (mailboxes 0 to 3). If another interrupt event associated with the same interrupt request occurs before an earlier event has been cleared, the interrupt request will continue to be asserted until after all interrupt flags have been cleared.

Figure 10–21. CAN Interrupt Flag Register (CAN_IFR) — Address 7109h

15–14	13	12	11	10	9	8
Reserved	MIF5	MIF4	MIF3	MIF2	MIF1	MIF0
	R-0	R-0	R-0	R-0	R-0	R-0

7	6	5	4	3	2	1	0
Rsvd	RMLIF	AAIF	WDIF	WUIF	BOIF	EPIF	WLIF
	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0	RC-0

Note: R = Read access; C = Clear; value following dash (–) = value after reset

Bits 15–14 **Reserved.**

Bits 13–8 **MIFx.** Mailbox Interrupt Flag (receive/transmit)

0 No message was transmitted or received.

1 The corresponding mailbox transmitted or received a message successfully.

Each of the six mailboxes may initiate an interrupt. These interrupts can be a receive or a transmit interrupt depending on the mailbox configuration. If one of the configurable mailboxes is configured as Remote Request Mailbox (AAM set) and a remote frame is received, a transmit interrupt is set after sending the corresponding data frame. If a remote frame is sent, a receive interrupt is set after the reception of the desired data frame.

There is one interrupt mask bit for each mailbox. If a message is received, the corresponding bit RMPn in the RCR is set. If a message is sent, the corresponding bit TA in the TCR register is set. The setting of the RMPn bit or the TAn bit also sets the appropriate MIFx flag in the IF register if the corresponding interrupt mask bit is set. The MIFx flag generates an interrupt. The MIMx mask bits determine if an interrupt can be generated by a mailbox.

Bit 7 **Reserved.**

Bit 6 **RMLIF.** Receive Message Lost Interrupt Flag

0 No message was lost.

1 An overflow condition has occurred in at least one of the receive mailboxes.

Bit 5 **AAIF.** Abort Acknowledge Interrupt Flag

0 No transmission was aborted.

1 A send transmission was aborted.

Bit 4 **WDIF.** Write Denied Interrupt Flag

0 The write access to the mailbox was successful.

1 The CPU tried to write to a mailbox but was not allowed to.

Bit 3 **WUIF.** Wake-Up Interrupt Flag

0 The module is still in the sleep mode or in normal operation.

1 The module has left the sleep mode.

Bit 2 **BOIF.** Bus Off Interrupt Flag

0 The CAN module is still in the bus-on mode.

1 The CAN has entered the bus-off mode.

Bit 1 **EPIF.** Error Passive Interrupt Flag

0 The CAN module is not in the error-passive mode.

1 The CAN module has entered the error-passive mode.

Bit 0 **WLIF.** Warning Level Interrupt Flag

0 None of the error counters has reached the warning level.

1 At least one of the error counters has reached the warning level.

10.6.2 CAN Interrupt Mask Register (CAN_IMR)

The setup for the interrupt mask register (see Figure 10–22) is the same as for the interrupt flag register (CAN_IFR) with the addition of the interrupt priority selection bits MIL and EIL. If a mask bit is set, the corresponding interrupt request to the PIE controller is enabled.

Figure 10–22. CAN Interrupt Mask Register (CAN_IMR) — Address 710Ah

15	14	13	12	11	10	9	8
MIL	Reserved	MIM5	MIM4	MIM3	MIM2	MIM1	MIM0
RW-0		RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
EIL	RMLIM	AAIM	WDIM	WUIM	BOIM	EPIM	WLIM
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access; W = Write access; value following dash (–) = value after reset

Two additional control bits are included in this register:

Bit 15 **MIL.** Mailbox Interrupt Priority Level

For the mailbox interrupts MIF5 – MIF0.

0 The mailbox interrupts generate high-priority requests; that is, on line CAMBOXIRQn with CAMBOXPRI set to 1.

1 The mailbox interrupts generate low-priority requests; that is, on line CAMBOXIRQn with CAMBOXPRI set to 0.

Bit 14 **Reserved**

Bit 7 **EIL.** Error Interrupt Priority Level

For the error interrupts RMLIF, AAIF, WDIF, WUIF, BOIF, EPIF, and WLIF.

0 The named interrupts generate high-priority requests; that is, on line CAERRIRQn with CAERRPRI set to 1.

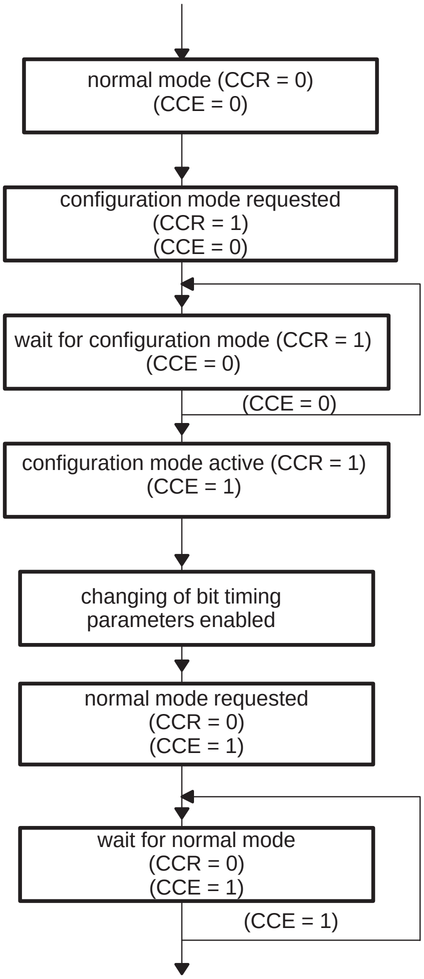
1 The named interrupts generate low-priority requests; that is, on line CAERRIRQn with CAERRPRI set to 0.

NOTE: For description of bits 13–8 and bits 6–0, refer to section 10.6.1.

10.7 Configuration Mode

The CAN module must be initialized before activation. This is only possible when the module is in the configuration mode, which is set by programming CCR with 1. The initialization can be performed only if the status bit CCE confirms the request by getting 1. Afterwards, the bit configuration registers can be written. The module is activated again by programming the control bit CCR with zero. After a hardware reset, the configuration mode is active.

Figure 10–23. CAN Initialization



10.8 Power-Down Mode (PDM)

If the peripheral clocks are to be shut off by the device low-power mode, the CAN peripheral's own low-power mode must be requested before a device low-power mode is entered by executing the IDLE instruction.

Before the CPU enters its IDLE mode prior to the device low-power mode that potentially shuts off *all* device clocks, it must first request a CAN peripheral power down by writing a 1 to the PDR bit in MCR. If the module is transmitting a message when PDR is set, the transmission is continued until a successful transmission, a lost arbitration, or an error condition on the CAN bus line occurs. Then the PDA is asserted. Thus, the module causes no error condition on the CAN bus line. When the module is ready to enter the power-down mode, the status bit PDA is set. The CPU must then poll the PDA bit in GSR, and only enter IDLE after PDA is set.

On exiting the power-down mode, the PDR flag in the MCR must be cleared by software, or automatically, if the WUBA bit in MCR is set and there is bus activity on the CAN bus line. When detecting a dominant signal on the CAN bus, the wake-up interrupt flag (WUIF) is asserted. The power-down mode is exited as soon as the clock is switched on. There is no internal filtering for the CAN bus line.

The automatic wake-up on bus activity can be enabled or disabled by setting the configuration bit WUBA. If there is any activity on the CAN bus line, the module begins its power up sequence. The module waits until detecting 11 consecutive recessive bits on the RX pin and goes to bus active afterwards. The first message, which initiates the bus activity, *cannot* be received.

When WUBA is enabled, the error interrupt WUIF is asserted automatically to the PIE controller, which will handle it as a wake-up interrupt and restart the device clocks if they are stopped.

After leaving the sleep mode with a wake up, the PDR and PDA are cleared. The CAN error counters remain unchanged.

10.9 Suspend Mode

The suspend mode can operate in either *Free mode*, where the CAN peripheral continues to operate regardless of the suspend signal being active, or *Soft mode*, where the CAN peripheral stops operation at the end of the current transmission. Suspend mode is entered when the CPU activates the SUSPEND signal. The SUSP bit in MCR determines which of the two suspend modes (*Free* or *Soft*) is entered.

When the module enters the Soft suspend mode, the status bit SMA is set. If the module is actually transmitting a message when the SUSPEND signal is activated, the transmission is continued until a successful transmission, a lost arbitration, or an error condition on the CAN bus line occurs. Otherwise, it enters suspend mode immediately and sets the SMA bit.

In Free mode, the peripheral ignores the suspend signal and continues to operate, receiving and transmitting messages.

Either way, the module causes no error condition on the CAN bus line.

When suspended (in Soft mode), the module does not send or receive any messages. The module is not active on the CAN bus line. Acknowledge flags and error flags are not sent. The error counters and all other internal registers are frozen. Suspend is only asserted when a system is being debugged with an in-circuit emulator.

In case the module is in bus-off mode when suspend mode is requested, it enters suspend mode immediately. It does, however, still count the 128 × 11 recessive bits needed to return to the bus-on mode. All error counters are undefined in that state. The bus-off flag and the error-passive flag are set.

The module leaves the suspend mode when the SUSPEND signal is deactivated. It waits for the next 11 recessive bits on the bus and goes back to normal operation. This is called the idle mode (different from the CPU's IDLE mode). The module waits for the next message or tries to send one itself. When the module is in bus-off mode, it continues to wait for the bus-on condition. This occurs when 128 × 11 recessive bits are received. It also counts those that occurred during the suspend mode.

Note: The clock is not switched off internally for suspend or low-power mode.

For easy reference, Table 10–5 provides a listing of the notation, definition, and register and bit number.

Table 10–5. CAN Notation

Notation	Signification	Register	Bit No.
AA:	Abort Acknowledge	TCR	11:8
AAIF:	Abort Acknowledge Interrupt Flag	IFR	5
AAIM:	Abort Acknowledge Interrupt Mask	IMR	5
AAM:	Auto Answer Mode	MSGIDn	13
ABO:	Auto Bus On	MCR	7
ACKE:	Acknowledge Error	ESR	3
AME:	Acceptance Mask Enable	MSGIDn	14
BEF:	Bit Error Flag	ESR	7
BO:	Bus Off Status	ESR	2
BOIF:	Bus Off Interrupt Flag	IFR	2
BOIM:	Bus Off Interrupt Mask	IMR	2
BRP:	Baud Rate Prescaler	BCR2	7:0
CCE:	Change Configuration Enable	GSR	4
CCR:	Change Configuration Request	MCR	12
CDR:	Change Data Field Request	MCR	8
CRCE:	CRC Error	ESR	5
DBO:	Data Byte Order	MCR	10
DLC:	Data Length Code	MSGCTRLn	3:0
EIL:	Error Interrupt Priority Level	IMR	7
EP:	Error Passive Status	ESR	1
EPIF:	Error Passive Interrupt Flag	IFR	1
EPIM:	Error Passive Interrupt Mask	IMR	1
EW:	Warning Status	ESR	0
FER:	Form Error Flag	ESR	8
IDE:	Identifier Extension	MSGIDn	15
LAMI:	Local Acceptance Mask Identifier	LAM	15
MBNR:	Mailbox Number	MCR	1:0
ME:	Mailbox Enable	MDER	5:0
MD:	Mailbox Direction	MDER	7:6
MIF:	Mailbox Interrupt Flag	IFR	13:8
MIL:	Mailbox Interrupt Priority Level	IMR	15
MIM:	Mailbox Interrupt Mask	IMR	13:8
OPC:	Overwrite Protection Control	RCR	3:0

Table 10–5. CAN Notation (Continued)

Notation	Signification	Register	Bit No.
PDA:	Power-Down Mode Acknowledge	GSR	3
PDR:	Power-Down Mode Request	MCR	11
REC:	Receive Error Counter	CEC	7:0
RFP:	Remote Frame Pending	RCR	15:12
RM:	Receive Mode	GSR	1
RML:	Receive Message Lost	RCR	11:8
RMLIF:	Receive Message Lost Interrupt Flag	IFR	6
RMLIM:	Receive Message Lost Interrupt Mask	IMR	6
RMP:	Receive Message Pending	RCR	7:4
RTR:	Remote Transmission Request	MSGCTRLn	4
SA1:	Stuck at dominant Error	ESR	6
SAM:	Sample Point Setting	BCR1	7
SBG:	Synchronization on Both Edge	BCR1	10
SER:	Stuff Error	ESR	4
SJW:	Synchronization Jump Width	BCR1	9:8
SMA:	Suspend Mode Acknowledge	GSR	5
STM:	Self Test Mode	MCR	6
SUSP:	Action on Emulator Suspend	MCR	13
TA:	Transmission Acknowledge	TCR	15:12
TEC:	Transmit Error Counter	CEC	15:8
TM:	Transmit Mode	GSR	0
TRS:	Transmission Request Set	TCR	4:7
TRR:	Transmission Request Reset	TCR	0:3
WDIF:	Write Denied Interrupt Flag	IFR	4
WDIM:	Write Denied Interrupt Mask	IMR	4
WLIF:	Warning Level Interrupt Flag	IFR	0
WLIM:	Warning Level Interrupt Mask	IMR	0
WUBA:	Wake Up on Bus Activity	MCR	9
WUIF:	Wake Up Interrupt Flag	IFR	3
WUIM:	Wake Up Interrupt Mask	IMR	3

Chapter 11

Watchdog (WD) Timer

The watchdog (WD) timer peripheral monitors software and hardware operations, and implements system reset functions upon CPU disruption. If the software goes into an improper loop, or if the CPU becomes temporarily disrupted, the WD timer overflows to assert a system reset.

Most conditions that temporarily disrupt chip operation and inhibit proper CPU function can be cleared and reset by the watchdog function. By its consistent performance, the watchdog increases the reliability of the CPU, thus ensuring system integrity.

All registers in this peripheral are eight bits in width and are attached to the lower byte of the peripheral data bus of the 16-bit CPU.

The only difference between the ’240x WD timer and that on the ’C240 is the lack of real-time Interrupt capability.

This implementation of the WD timer generates it’s own watchdog clock locally by dividing down the CLKOUT from CPU.

Topic **Page**

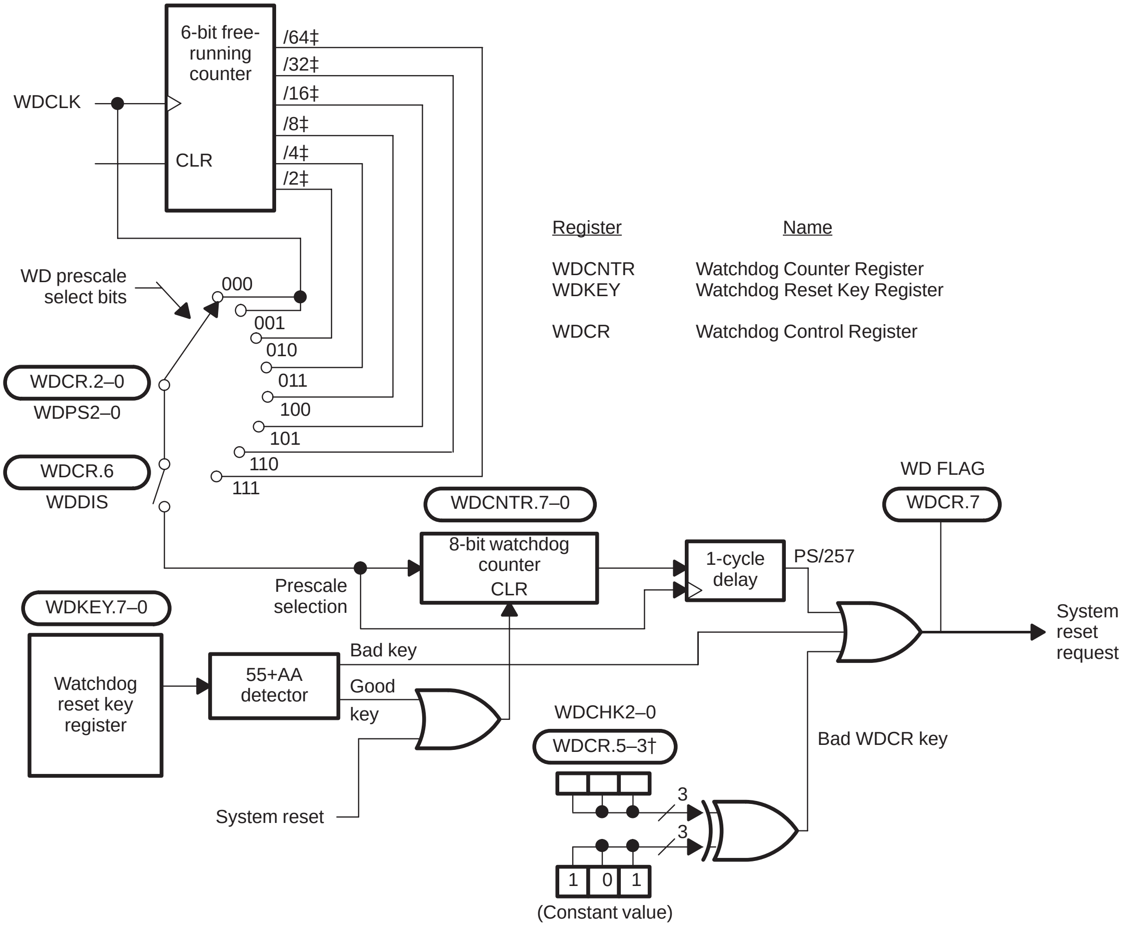
11.1 Watchdog Timer Features

The WD module includes the following features:

- ❑ 8-bit WD counter that generates a system reset upon overflow
- ❑ 6-bit free-running counter that feeds the WD counter via the WD counter prescale
- ❑ A WD reset key (WDKEY) register that clears the WD counter when the correct combination of values are written, and generates a reset if an incorrect value is written to the register
- ❑ WD check bits that initiate a system reset if the WD timer is corrupted
- ❑ Automatic activation of the WD timer, once system reset is released
- ❑ A WD prescale with six selections from the 6-bit free-running counter

Figure 11–1 shows a block diagram of the WD module

Figure 11–1. Block Diagram of the WD Module



Register	Name
WDCNTR	Watchdog Counter Register
WDKEY	Watchdog Reset Key Register
WDCR	Watchdog Control Register

† Writing to bits WDCR.5–3 with anything but the correct pattern (101) generates a system reset.
‡ These prescale values are with respect to the WDCLK signal.

11.2 Control Registers

Three registers control the WD operations:

- ❑ WD Counter Register (WDCNTR) — This register contains the value of the WD counter.
- ❑ WD Key Register (WDKEY) — This register clears the WDCNTR when a 55h value followed by an AAh value is written to WDKEY.
- ❑ WD Control Register (WDCR) — This register contains the following control bits used for watchdog configuration
  - ■ WD disable bit
  - ■ WD flag bit
  - ■ WD check bits (three)
  - ■ WD prescale select bits (three)

The Watchdog Timer Clock is a low-frequency clock (WDCLK) is used to clock the Watchdog Timer. WDCLK has a nominal frequency of 58593.8 Hz when CPUCLK = 30 MHz. WDCLK is derived from the CLKOUT of the CPU. This ensures that the Watchdog continues to count when the CPU is in IDLE1 or IDLE 2 mode (see the section on Low-Power Modes). WDCLK is generated in the Watchdog peripheral. The frequency of WDCLK can be calculated from:

$$\text{WDCLK} = (\text{CLKOUT})/512$$

WDCLK is seen at the CLKOUT pin only when the Watchdog is enabled. If the watchdog is enabled, the watchdog counter should be reset before it overflows; otherwise, the DSP will be reset.

11.2.1 Watchdog Suspend

WDCLK is stopped when the CPU’s suspend signal goes active. This is achieved by stopping the clock input to the clock divider which generates WDCLK from CLKOUT.

Note that the watchdog timer clock does not run when the real-time monitor is running. This is different from the ’F/C240.

11.2.2 Operation of WD Timers

The WD timer is an 8-bit resetable incrementing counter that is clocked by the output of the prescaler. The timer protects against system software failures and CPU disruption by providing a system reset when the WDKEY register is not serviced before a watchdog overflow. This reset returns the system to a known starting point. Software then clears the WDCNTR register by writing a correct data pattern to the WD key logic.

A separate internal clocking signal (WDCLK) is generated by the on-chip clock module and is active in all operational modes except the HALT mode. WDCLK enables the WD timer to function, regardless of the state of any register bit(s) on the chip, except during the HALT low-power mode, which disables the WDCLK signal. The current state of WDCNTR can be read at any time during its operation.

11.2.3 WD Prescale Select

The 8-bit WDCNTR can be clocked directly by the WDCLK signal or through one of six taps from the free-running counter. The 6-bit free-running counter continuously increments at a rate provided by WDCLK. The WD functions are enabled as long as WDCLK is provided to the module. Any one of the six taps (or the direct input from WDCLK) can be selected by the WD prescale select (bits WDPS2–0) as the input to the time base for the WDCNTR. This prescale provides selectable watchdog overflow rates of from 4.36 ms to 279.6 ms for a WDCLK rate of 58593.8 Hz. While the chip is in normal operation mode, the free-running counter cannot be stopped or reset, except by a system reset. Clearing WDCNTR does not clear the free-running counter.

11.2.4 Servicing the WD Timer

The WDCNTR is reset when the proper sequence is written to the WDKEY before the WDCNTR overflows. The WDCNTR is enabled to be reset when a value of 55h is written to the WDKEY. When the next AAh value is written to the WDKEY, then the WDCNTR actually is reset. Any value written to the WDKEY other than 55h or AAh causes a system reset. Any sequence of 55h and AAh values can be written to the WDKEY without causing a system reset; only a write of 55h followed by a write of AAh to the WDKEY resets the WDCNTR.

Table 11–1 shows a typical sequence written to WDKEY after power-up.

Table 11–1. Typical WDKEY Register Power-Up Sequence

Sequential Step	Value Written to WDKEY	Result
1	AAh	No action.
2	AAh	No action.
3	55h	WDCNTR is enabled to be reset by the next AAh.
4	55h	WDCNTR is enabled to be reset by the next AAh.
5	55h	WDCNTR is enabled to be reset by the next AAh.
6	AAh	WDCNTR is reset.
7	AAh	No action.
8	55h	WDCNTR is enabled to be reset by the next AAh.
9	AAh	WDCNTR is reset.
10	55h	WDCNTR is enabled to be reset by the next AAh.
11	23h	System reset due to an improper key value written to WDKEY.

Step 3 above is the first action that enables the WDCNTR to be reset. The WDCNTR is not actually reset until step 6. Step eight re-enables the WDCNTR to be reset, and step 9 resets the WDCNTR. Step 10 again reenables the WDCNTR to be reset. Writing the wrong key value to the WDKEY in step 11 causes a system reset.

A WDCNTR overflow or an incorrect key value written to the WDKEY also sets the WD flag (WDFLAG). After a reset, the program reads this flag to determine the source of the reset. After reset, WDFLAG should be cleared by the software to allow the source of subsequent resets to be determined. WD resets are not prevented when the flag is set.

11.2.4.1 WD Reset

When the WDCNTR overflows, the WD timer asserts a system reset. Reset occurs one WDCNTR clock cycle (either WDCLK or WDCLK divided by a prescale value) later. The reset cannot be disabled in normal operation as long as WDCLK is present. The WD timer is, however, disabled in the oscillator powerdown mode when WDCLK is not active. For software development or flash programming purposes, the WD timer can be disabled by setting the WDDIS bit in the WD control register (WDCR.6). Note that there is no WDDIS pin in the ’240x devices.

11.2.4.2 WD Check Bit Logic

The WD check bits (WDCR.5–3, described in detail in section 11.3.3 on page 11-9) are continuously compared to a constant value (101_2). If writes to the WD check bits do not match this value, a system reset is generated. This functions as a logic check, in case the software improperly writes to the WDCR, or if an external stimulus (such as voltage spikes, EMI, or other disruptive sources) corrupt the contents of the WDCR. Writing to bits WDCR.5-3 with anything but the correct pattern (101_2) generates a system reset.

The check bits are always read as zeros (000_2), regardless of what value has been written to them.

11.2.4.3 WD Setup

The WD timer operates independently of the CPU and is always enabled. It does not need any CPU initialization to function. When a system reset occurs, the WD timer defaults to the fastest WD timer rate available (4.36 ms for a 58593.8-Hz WDCLK signal). As soon as reset is released internally, the CPU starts executing code, and the WD timer begins incrementing. This means that, to avoid a premature reset, WD setup should occur early in the power-up sequence.

11.3 Watchdog Control Registers

The WD module control registers are shown in Table 11–2 and discussed in detail in the following subsections.

Table 11–2. WD Module Control Registers

		Bit Number							
Address	**Register mnemonic**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
7020h	—	Reserved							
7021h	—	Reserved							
7022h	—	Reserved							
7023h	WDCNTR	D7	D6	D5	D4	D3	D2	D1	D0
7024h	—	Reserved							
7025h	WDKEY	D7	D6	D5	D4	S3	S2	D1	D0
7026h	—	Reserved							
7027h	—	Reserved							
7028h	—	Reserved							
7029h	WDCR	Reserved	WDDIS	WDCHK2	WDCHK1	WDCHK0	WDPS2	WDPS1	WDPS0

11.3.1 WD Counter Register

The 8-bit WD counter register (WDCNTR) contains the current value of the WD counter. This register continuously increments at a rate selected through the WD control register. When WDCNTR overflows, an additional single-cycle delay (either WDCLK or WDCLK divided by a prescale value) is incurred before system reset is asserted. Writing the proper sequence to the WD reset key register clears WDCNTR and prevents a system reset. However, it does not clear the free-running counter.

Figure 11–2.WD Counter Register (WDCNTR) — Address 7023h

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Note: R = Read access, -0 = value after reset

Bits 7–0 **D7–D0.** Data Values. These read-only data bits contain the 8-bit WD counter value. Writing to this register has no effect.

11.3.2 WD Reset Key Register

The WD reset key register clears the WDCNTR register when a 55h followed by an AAh is written to WDKEY. Any combination of AAh and 55h is allowed, but only a 55h followed by an AAh resets the counter. Any other value causes a system reset.

Figure 11–3.WD Reset Key Register (WDKEY) — Address 7025h

7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset

Bits 7–0 **D7–D0.** Data Values. These write-only data bits contain the 8-bit WD reset key value. When read, the WDKEY register does **not** return the last key value but rather returns the contents of the WDCR register.

11.3.3 WD Timer Control Register

WDCR contains control bits used for watchdog configuration. These include flag bits that indicate if the WD timer initiated a system reset; check bits that assert a system reset if an incorrect value is written to the WDCR register; and watchdog prescale select bits that select the counter overflow tap which is used to clock the WD counter.

Figure 11–4.WD Timer Control Register (WDCR) — Address 7029h

7	6	5	4	3	2	1	0
Reserved	WDDIS	WDCHK2	WDCHK1	WDCHK0	WDPS2	WDPS1	WDPS0
RC-x	RWc-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, C = Clear by writing 1, W = Write access, Wc = Write access conditional on VCCP or WDDIS pins high, -0 = value after reset, -x = value after reset determined by action/inaction of WD timer

Bit 7 **Reserved**

Bit 6 **WDDIS.** Watchdog Disable. This bit can be written only when the VCCP (on Flash devices) or the WDDIS pin (on ROM devices) is high.

0 Watchdog is enabled.

1 Watchdog is disabled.

Bit 5 **WDCHK2**. Watchdog Check Bit 2. This bit must be written as a 1 when you write to the WDCR register, or else a system reset is asserted. This bit is always read as 0.

0 System reset is asserted.

1 Normal operation continues if all check bits are written correctly.

Bit 4 **WDCHK1**. Watchdog Check Bit 1. This bit must be written as a 0 when you write to the WDCR register, or else a system reset is asserted. This bit is always read as 0.

0 Normal operation continues if all check bits are written correctly.

1 System reset is asserted.

Bit 3 **WDCHK0**. Watchdog Check Bit 0. This bit must be written as a 1 when you write to the WDCR register, or else a system reset is asserted. This bit is always read as 0.

0 System reset is asserted.

1 Normal operation continues if all check bits are written correctly.

Bits 2–0 **WDPS2–WDPS0**. Watchdog Prescale Select Bits. These bits select the counter overflow tap that is used to clock the WD counter. Each selection sets up the maximum time that can elapse before the WD key logic is serviced. Table 11–3 show the overflow times for each prescaler setting when the WDCLK is running at 58593.8 Hz. Because the WD timer counts 257 clocks before overflowing, the times given are the minimum for overflow (reset). The maximum timeout can be up to 1/256 longer than the times listed in Table 11–3 because of the added uncertainty resulting from not clearing the prescaler.

Table 11–3. WD Overflow (Timeout) Selections

WD Prescale Select Bits				58593.8 kHz WDCLK†	
WDPS2	WDPS1	WDPS0	WDCLK Divider	Overflow Frequency (Hz)	Minimum Overflow (ms)
0	0	X	1	228.9	4.36
0	1	0	2	114.4	8.7
0	1	1	4	57.2	17.5
1	0	0	8	28.6	35
1	0	1	16	14.3	69.9
1	1	0	32	7.15	139.8
1	1	1	64	3.6	279.6

X = Don’t care
† Generated by a 30-MHz clock

Chapter 12

’240x–’240 Family Compatibility

This chapter describes the compatibility issues between the ’240x and ’240 family of processors.

The software changes required between ’240 code and ’240x code have been kept to a minimum. A majority of the register addresses, bit positions, and functions are identical between the ’240 and ’240x devices.

Topic **Page**

12.1 General

Low-power mode 2 (HALT) is the lowest power mode on the ’240x. It is similar to the LPM3 (oscillator power down) on the ’240. There is no equivalent to LPM2 (PLL power down) on the ’240. The low-power-mode bits are in a different register (SCSR1) and in different bit positions on the ’240x.

Software reset is not available. However a software reset can be achieved by writing an incorrect key to the watchdog timer after setting a flag in memory to indicate that this was a software reset, and not a true watchdog time-out.

Illegal address detect does not have 100% coverage on the ’240; however, it does on ’240x devices. Furthermore, an illegal address generates a reset on the ’240, and an NMI on the ’240x. The NMI service routine must poll the ILLADDR bit in SCSR1 to determine whether the NMI was caused by an illegal address or the NMI pin.

External interrupts XINT2 and XINT3 on the ’240 are similar to external interrupts XINT1 and XINT2 on the ’240x. The addresses of the registers are different, however, and the general-purpose I/O multiplexing control bits are located in the digital I/O registers, not in the external interrupt control registers. The external interrupt flags are cleared by writing a 1 to the flag bit. This is in order to be consistent with the other peripherals.

The CLOCKOUT control bits are in a different register (SCSR1) and bit position.

12.2 Event Manager

In order to port code from ’240 to ’240x:

- ❑ The GP timer 3 must not be used.
- ❑ The single-up count and single-up/down count modes of the GP timers must not be used. The decoding of the timer modes from the TMODE1–0 bits has changed, and this code will have to be modified when porting code from the ’240 to the ’240x.
- ❑ The 32-bit timer mode cannot be used.
- ❑ Capture 3 on the ’240 cannot be used, when porting code from the ’240 to the ’240x. Capture 4 should be changed into capture 3.
- ❑ The capture units can use either GP Timer 1 or 2 as a time base.
- ❑ When porting code from the the ’240 to the ’240x, the capture interrupt code needs to allow for the fact that an interrupt is usually generated after every second capture and not every capture as on the ’240.
- ❑ The QEP logic can clock GP timer 1 or 2.
- ❑ The three simple compare units cannot be used.
- ❑ The compare mode of the (full) compare units cannot be used; only the PWM mode can be used.
- ❑ Software must change from ’240 to ’240x to comprehend the changes to the dead-band counters and dead-band prescaler.
- ❑ All general interrupt service routines must be changed to get their peripheral interrupt vectors from the PIVR (701Eh) and not one of EVIVRA, EVIVRB or EVIVRC. Reading from PIVR does not clear interrupt flags. Interrupt flags must be cleared manually.

12.3 Analog-to-Digital Converter

When compared to the ’240 ADC, the ’240x ADC has been significantly enhanced. As a result, code written for the ’240 ADC cannot be ported to the ’240x.

12.4 Serial Communications Interface

Some code changes are required. This is code that switches the SCI pins between their SCI functions and their digital I/O functions, and accesses them in digital I/O mode. When porting code from a ’240 to a ’240x device, it must access the relevant bits in the digital I/O peripheral instead of the SCIPC2 register.

The SCI has *free* and *soft* emulation modes.

12.5 Serial Peripheral Interface

This SPI is no longer limited to a maximum transmission rate of CLKOUT / 8 in slave mode. The maximum transmission rate in *both* slave mode *and* master mode is now CLKOUT / 4.

Some code changes are required. This is code that switches the SPI pins between their SPI functions and their digital I/O functions, and accesses them in digital I/O mode. When code is ported from ’240 to ’240x devices, it must access the relevant bits in the digital I/O peripheral instead of the SCIPC1 and SCIPC2 register.

When code is ported from a ’240 to a ’240x device, writes of transmit data to the serial data register, SPIDAT, must be left-justified within a 16-bit register, not within an 8-bit register.

The SPI has *free* and *soft* emulation modes.

12.6 Watchdog Timer

When porting code from ’240 to ’240x devices, all code that uses the RTI peripheral (if any) must be removed.

Chapter 13

’24x–’240x Family Compatibility

Topic Page

13.1 Introduction

This chapter highlights the major differences (in terms of features/peripherals) between the ’240x and the ’24x (TMS320F243/’F241/’C242) family of DSP devices. The ’240x devices share most of the ’24x features; however, ’240x devices have some enhancements. The common features, differences, and enhancements are described in Table 13–1 and Table 13–2.

Table 13–1. ’24x-Compatible Features/Peripherals in ’240x DSPs

	’24x-Compatible Features/Peripherals in ’240x	Reference Document
1	’C2xx CPU, Instruction Set, Interrupt Behavior, and B0, B1, B2 DARAM	TMS320F/C24x DSP Controllers Reference Guide (literature number SPRU160C)
2	XMIF (External Memory Interface)	TMS320F243, TMS320F241 DSP Controllers Data Sheet (literature number SPRS064)
3	Watchdog: (With the minor exception of the absence of the WDDIS pin; this feature is now implemented in software.) All WD registers are identical.	TMS320F243/F241/C242 DSP Controllers Reference Guide (literature number SPRU276C)
4	EVA (Event Manager): EVA is exactly identical to EV2 in ’24x family	TMS320F243/F241/C242 DSP Controllers Reference Guide (literature number SPRU276C)
5	CAN (Controller Area Network)	TMS320F243/F241/C242 DSP Controllers Reference Guide (literature number SPRU276C)
6	SCI (Serial Communications Interface)	TMS320F243/F241/C242 DSP Controllers Reference Guide (literature number SPRU276C)
7	SPI (Serial Peripheral Interface)	TMS320F243/F241/C242 DSP Controllers Reference Guide (literature number SPRU276C)

Table 13–2. New or Modified Features/Peripherals in ’240x DSPs

	New/Modified Features/Peripherals in ’240x	Reference Document/Chapter
1	Flash Memory Map and Flash Wrapper	Chapter 13
2	On-Chip SARAM	TMS320LF2407, TMS320LF2406, TMS320LF2402, TMS320LC2406, TMS320LC2404, TMS320LC2402 DSP Controllers Data Sheet (literature number SPRS094)
3	Peripheral Register Map	Appendix A
4	ADC	Chapter 7
5	Interrupt Vector Table	Chapter 2
6	PIE (Peripheral Interrupt Expansion Unit)	Chapter 2
7	EVB (Event Manager B)	Chapter 6
8	Digital I/O	Chapter 5
9	PLL	Chapter 4

For additional reference materials, see the application report *3.3-V DSP for Digital Motor Control* (literature number SPRA550) or go to http://www.ti.com/sc/docs/apps/digital/appnotes/html for a list of ’24x application notes.

13.2 ’24x–’240x DSP Overview

Table 13–3. Features of ’24x and ’240x DSPs

Device Feature	’LF2407	’LF2406	’LF2402	’24x (’F243)
C2xxLP CPU	Yes	Yes	Yes	Yes
DARAM	544 words	544 words	544 words	544 words
SARAM – Program/Data	2K words	2K words	–	–
Power Supply	3.3V Core 3.3V I/O	3.3V Core 3.3V I/O	3.3V Core 3.3V I/O	5V Core 5V I/O
3V Flash (Program Space)	32K x 16	32K x 16	8K x 16	5V Flash 16/8K x 16
Sectors	4K/12K/12K/4K	4K/12K/12K/4K	2 x 4K	None
Boot ROM – 256 words	Yes	Yes	Yes	–
Event Manager: EVA	Yes	Yes	Yes	EV2
Event Manager: EVB	Yes	Yes	–	–
CAN	Yes	Yes	–	Yes
SPI	Yes	Yes	–	Yes
SCI	Yes	Yes	Yes	Yes
10-bit ADC Channels	16	16	8	8
WD	Yes	Yes	Yes	Yes
General-Purpose Digital I/O	40 – Shared with other functions 1 – Dedicated to I/O	39 – Shared with other functions 2 – Dedicated to I/O	16 – Shared with other functions 5 – Dedicated to I/O	26 – Shared with other functions
External Interrupts	$\overline{\text{PDPINTA}}$, $\overline{\text{PDPINTB}}$, XINT1, XINT2	$\overline{\text{PDPINTA}}$, $\overline{\text{PDPINTB}}$, XINT1, XINT2	$\overline{\text{PDPINTA}}$, XINT2	PDPINT, XINT1, XINT2
External Memory Interface	Yes	No	No	Yes
Package	144 TQFP	100 TQFP	64 PQFP	144 TQFP

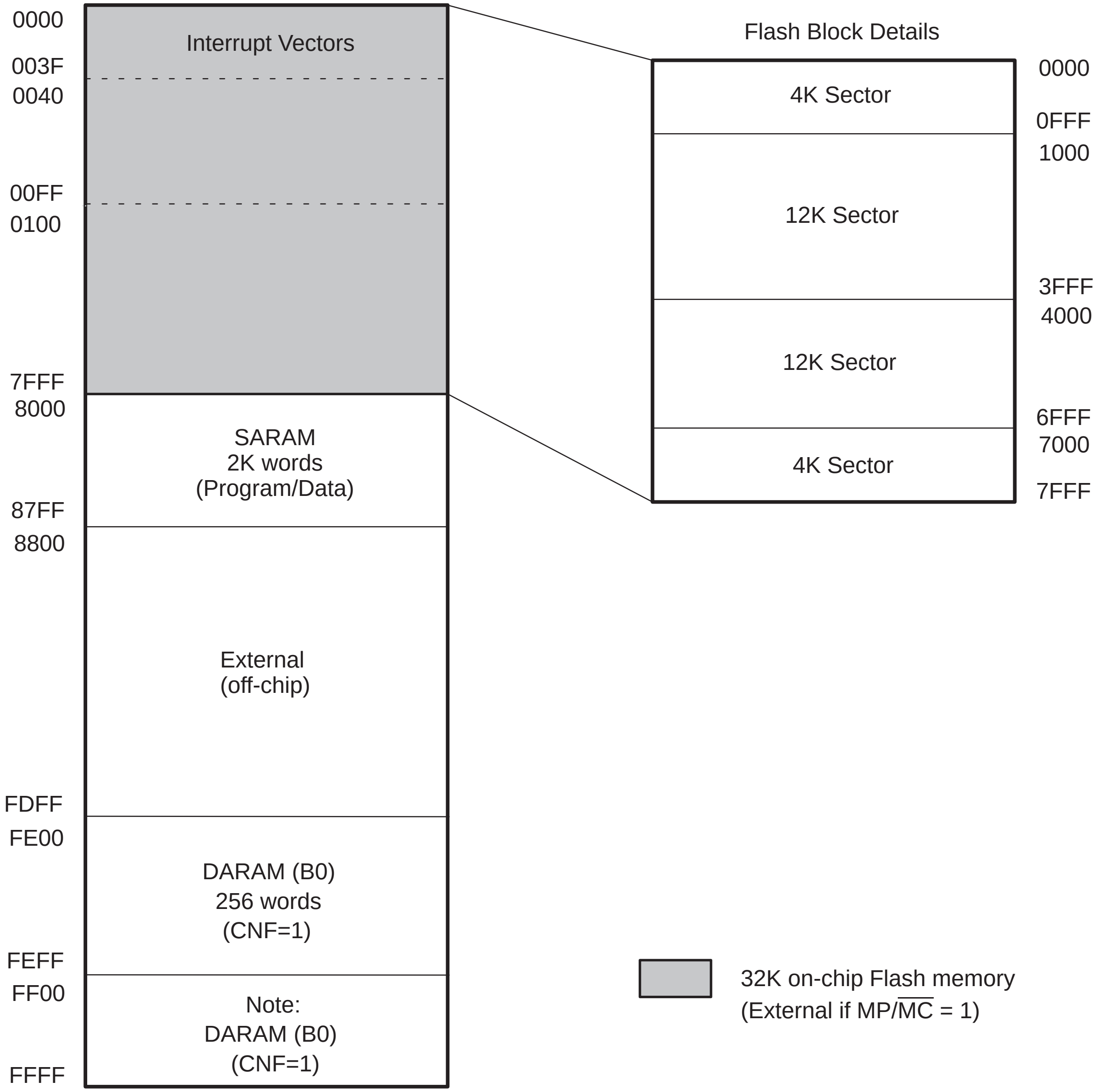
Notes: 1) Refer to the TMS320LF2407, TMS320LF2406, TMS320LF2402, TMS320LC2406, TMS320LC2404, TMS320LC2402 DSP Controllers Data Sheet (literature number SPRS094) for LF (Flash) and LC (ROM) device details.

2) ’F243 column is just for device feature comparison.

13.3 Memory Map

13.3.1 Program Space

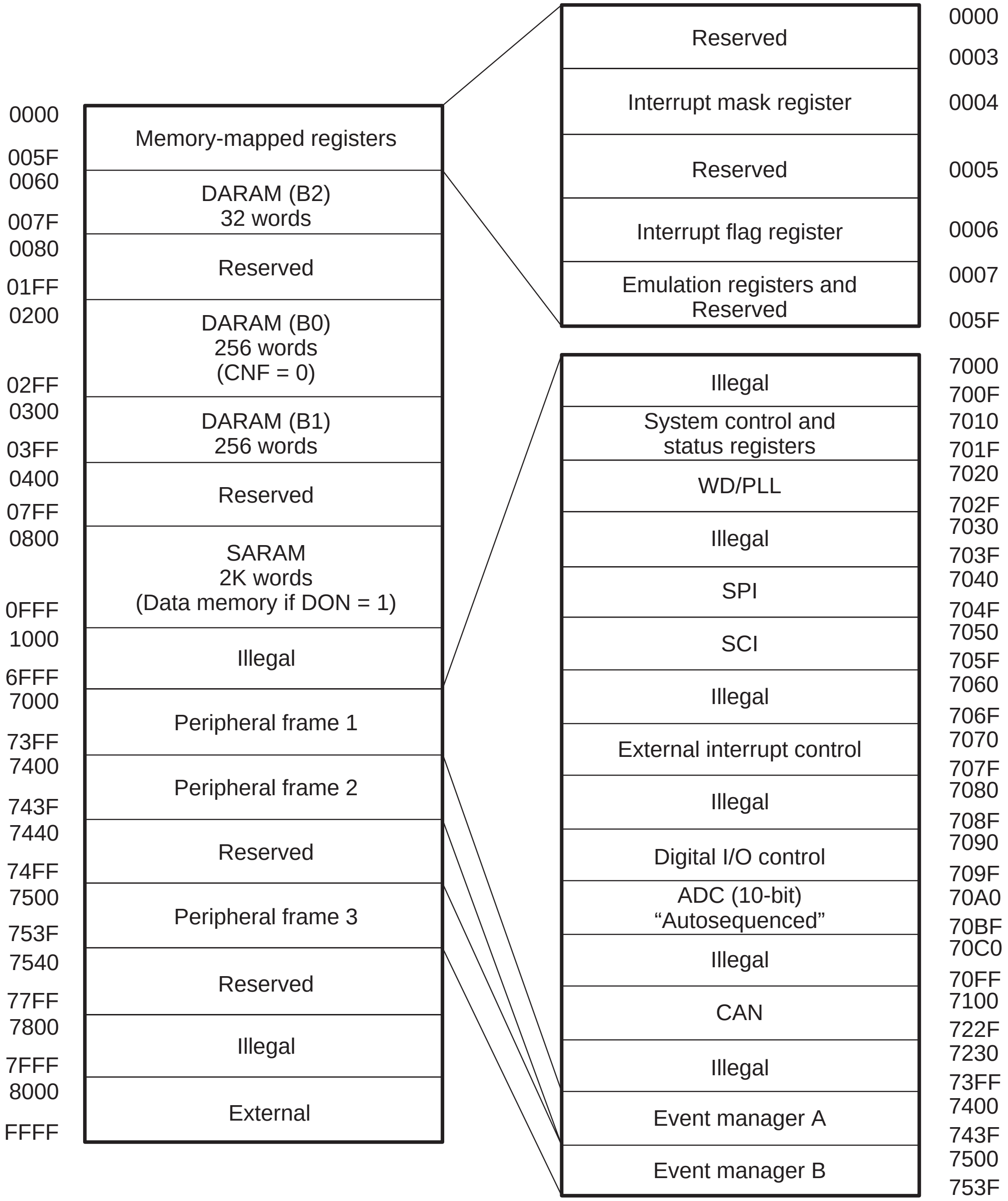
Figure 13–1. ’LF2407 Memory Map for Program Space



Note: When boot ROM is enabled, on-chip locations 0000–00FFh in program memory is mapped to the bootloader. Boot ROM and Flash Memory share the same starting address, and hence, are not visible (active) at the same time. If the $\overline{\text{BOOT\_EN}}$/XF pin = 0 during reset, the BOOT_EN bit in SCSR2 register (bit 3) will be set and enable the Boot ROM at 0000 in program space. While Boot ROM is enabled, the entire Flash memory will be disabled. The SCSR2.3 bit should be disabled (0) to have Flash array enabled instead of Boot ROM. See Appendix C for bootloader details.

13.3.2 Data Space

Figure 13–2. ’LF2407 Memory Map for Data Space



13.3.3 I/O Space

The ’LF2407 is the only device which has all the I/O space enabled and is available through the external memory interface. All ’240x devices have two addresses in the I/O space as internal registers for Flash and wait-state generator. Table 13–4 explains the internal register addresses that are used. The external I/O space is reserved for all ’240x devices except the ’LF2407.

Table 13–4. I/O Mapped Registers

Name	I/O Address	Availability	Description
FCMR	FF0Fh	All ’240x devices	Dummy I/O address space reserved for enabling/ disabling Flash control registers
WSGR	FFFFh	’LF2407 only	Software wait-state generator for the external memory address in Program, Data, and I/O

13.4 Flash Program Memory

The Flash module is generally used to provide permanent program storage. The Flash can emulate standard EEPROM or can be programmed and electrically erased many times to allow code development. The ’240x Flash is similar to that on the ’24x devices, with some key differences and enhancements. ’240x Flash features are as follows:

- Flash run-time execution at 3.3 V
- Flash programming requires a 5-V supply (±5%) at V_{CCP} pin
- Flash has multiple sectors that can be protected while erasing
- Flash programming registers are similar to those on the ’24x devices
- Flash programming is done through CPU
- ’240x devices comes with JTAG interface to aid programming and emulation
- A 256-word Boot ROM is available on ’240x devices to enable programming through SCI or SPI ports.

The following sections explain the Flash programming registers and their bit functions. Flash programming utilities will be provided by Texas Instruments (TI™). Refer to the TI’s web page (www.ti.com, under ’24x Flash tools) for revisions of these utilities.

13.4.1 Flash Control Mode Register (FCMR)

The Flash control mode register is in internal I/O space FF0Fh. This register is a dummy register address to enable the Flash in Flash array mode or in Flash control register mode.

The Flash control registers are used to program the Flash array. These registers are a part of the Flash wrapper and are mapped at the same start address as the Flash array itself. These registers are not visible (disabled) during Flash array mode (i.e., Flash read). During the Flash control register mode, the Flash program control registers are enabled and the Flash array is disabled (i.e., not accessible to CPU). To enter and exit the Flash control register mode, the instructions in Table 13–5 are used.

Table 13–5. Instructions for Entering and Exiting Flash Control Register Mode

Mode	’C2xx Instruction		Description
Flash Array Access Mode	IN	Dummy, 0FF0Fh	Sets the Flash into normal run or operating mode (i.e., Flash read) (Note: Flash Control Registers are disabled in this mode.)
Control Register Access Mode	OUT	Dummy, 0FF0Fh	Sets the Flash into control mode, ready for Erase and Programming operations (i.e., makes the Control registers visible) (Note: Flash Array is disabled in this mode.)

Note: The word “dummy” in the above two instructions indicate that the actual word written into (or read from) I/O address FF0Fh is not important. Rather, it is the mere action of reading or writing from/to FF0Fh that puts the Flash in the appropriate mode.

13.4.2 Flash Control Registers – Summary Table

Table 13–6 lists a set of control register that will be visible during Flash control register mode. The table also explains their individual functions during Flash programming.

Table 13–6. Flash Control Registers in Flash Control Mode

Name	Program Address	Description
PMPC	XXX0h	Pump Control Register. This register controls the charge pump. It contains the EXECUTE bit, two KEY bits that offer protection against inadvertent programming or erasing, and the PWRD bit that is used as an alternative to the POWERDOWN input signal.
CTRL	XXX1h	Flash Control Register. This register controls all modes of operation on the Flash array, such as programming, erasing, and compaction.
WADDR	XXX2h	Write Address Register. This register holds the address for a write operation. It is also used during an erase operation to define which sector of Flash is to be erased.
WDATA	XXX3h	Write Data Register. This register holds the data for a write operation. It is also used during an erase operation as a key to prevent inadvertent erases.
TCR	XXX4h	Test Control Register. This register is used for test operations. It is not accessible in normal operation.
ENAB	XXX5h	Flash Core Enable Register – Reserved
SECT	XXX6h	Sector Enable Register. This register has four bits to enable the four sectors of the array for programming and erasing operations.

13.4.2.1 Pump Control Register

Figure 13–3. Pump Control Register

15–8
Reserved
R-x

7–4	3	2	1	0
Reserved	PWR DWN	KEY1	KEY0	EXEC
R-x	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset, x = undefined

Bits 15–4 **Reserved**

Bit 3 **PWR DWN.** Power down bit

Writing a 1 to this bit puts the Flash pumps into a very low current consumption mode. This register bit is mostly intended for test purposes; however, this bit can be used in normal operating mode also, if needed. Powerdown mode is entered if the PWRD bit is set high.

Bit 2 **KEY1.** Execute key bit 1

This bit must be written as a 1 in the same access as the EXEC bit is set for the EXECUTE operation to start. This bit is used as additional protection against inadvertent programming or erasing of the Flash.

Bit 1 **KEY0.** Execute key bit 0

This bit must be written as a 0 in the same access as the EXEC bit is set for the EXECUTE operation to start. This bit is used as additional protection against inadvertent programming or erasing of the Flash.

Bit 0 **EXEC.** Execute

This bit initiates and ends programming, erasing, and compaction to the Flash array. The EXEC bit and the KEY0/1 bits must be written in the same write access. When the operation time has expired, the EXEC bit should be cleared in the same write as the KEY0/1 bits. The data and address latches are locked whenever the EXEC bit is set and all attempts to read from or write to the array will be ignored (read data will be indeterminate). This bit controls the EXECUTEZ signal on the Charge Pump module.

13.4.2.2 Flash Control Register

Figure 13–4. Flash Control Register

15–10	9	8
Reserved	WSVER En	PRECND Mode1
R-x	R-x	R-x

7	6	5	4	3	2	1	0
PRECND Mode0	ENG/R Mode2	ENG/R Mode1	ENG/R Mode0	FCM3	FCM2	FCM1	FCM0
R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x

Note: R = Read access, x = undefined

Bits 15–10 **Reserved**

Bit 9 **WSVER En.** Verification wait-state enable

When active high, this bit enables the automatic generation of a wait state on all verification reads (ERASEVER, PROGVER, CMPCTVER, RDMRGN1, RDMRGN0) by pulling OREADY low for one cycle. When inactive low, verification reads will have zero wait states.

Bit 8 **PRECND Mode1**

Bit 7 **PRECND Mode0**

Bit 6 **ENG/R Mode2**

Bit 5 **ENG/R Mode1**

Bit 4 **ENG/R Mode0**

Bits 3–0 **FCM3–FCM0.** Flash Control mode bits 3–0

These four bits select which mode of operation the Flash module is in.

0000 **Normal mode** (Read mode)

0001 **ERASE** (Erase mode). Set up charge pump and Flash core for an erase. Set this mode to start the erase operation. The ERASE mode must be set prior to the access that sets the PMPC EXECUTE bit, by the delay defined in the Flash module specification.

0010 **ERASEVER** (Erase verify mode). This is used to verify proper erasure. This sets up the charge pump and Flash core for verification. The sense voltage is lowered to verify the margin of 1’s in the array. The ERASEVER mode must be set prior to the access that sets the PMPC EXECUTE bit, by the delay defined in the Flash module specification.

0011 **PROG** (Program mode). Set up charge pump and Flash core for programming operation. Set this mode to start the write operation. The PROG mode must be set prior to the access that sets the PMPC EXECUTE bit, by the delay defined in the Flash module specification.

0100 **PROGVER** (Program verify mode). This is used to verify proper programming. This sets up the charge pump and Flash core for verification. The sense voltage is raised to verify the margin of 0’s in the array. The PROGVER mode must be set prior to the access that sets the PMPC EXECUTE bit, by the delay defined in the Flash module specification.

0101 **CMPCT** (Compaction mode). Set up charge pump and Flash core for compaction operation. Set this mode to start a compaction operation that pulls over-erased bits out of depletion. The CMPCT mode must be set prior to the access that sets the PMPC EXECUTE bit, by the delay defined in the Flash module specification.

0110 **CMPCTVER** (Compaction verify mode). This is used to test for bits erased into depletion. This sets up the charge pump and Flash core for the depletion test. The CMPCTVER mode must be set prior to the access that sets the PMPC EXECUTE bit, by the delay defined in the Flash module specification.

0111 **RDMRGN0** (Read Margin 0 mode). This is used to verify margin of programmed bits. This sets up the charge pump and Flash core for verification. The sense voltage is raised to verify the margin of 0's in the array. It is similar to program verify, but the voltage is not raised as much. The RDMRGN0 mode must be set prior to the access that sets the PMPC EXECUTE bit, by the delay defined in the Flash module specification.

1000 **RDMRGN1** (Read Margin 1 mode). This is used to verify margin of erased bits. This sets up the charge pump and Flash core for verification. The sense voltage is lowered to verify the margin of 1's in the array. It is similar to erase verify, but the voltage is not lowered as much. The RDMRGN1 mode must be set prior to the access that sets the PMPC EXECUTE bit, by the delay defined in the Flash module specification.

1001 to 1111 **Reserved**.

13.4.2.3 Test Control Register

Figure 13–5. Test Control Register

15–8
Reserved
R-x

7–3	2	1	0
Reserved	TCR Enable	Standby	CNTL Enable
R-x	R-x	R-x	R-x

Note: R = Read access, x = undefined

Bits 15–3 **Reserved**

Bit 2 **TCR Enable.** Test Control Register Enable

This bit enables the Test Control Register (TCR) to the specific core corresponding to this ENAB register. This controls the TEZ input to the Flash core. This bit is only accessible if the TEST input is high.

Bit 1 **Standby.** Standby mode Enable

This bit, when active high, puts the Flash core into standby mode. In this mode, the Flash core goes into a very low current consumption mode, and the Flash core cannot be read. To read the Flash core again, this bit must be cleared first. This controls the SENSE input to the Flash core, which controls whether the sense amp is on or off.

Bit 0 **CNTL Enable.** Control Register Enable

This bit gates the Flash Control Register (CTRL) to the specific core corresponding to this ENAB register. It enables the following Flash operations to the Flash core: PROG, PROGVER, ERASE, ERASEVER, CMPCT, CMPCTVER, RDMRGN0, RDMRGN1, REDU, ENGR0, ENGR1, ENGR2, NOROWRED, and PRECOND. This controls the CTRLENZ input to the Flash core.

13.4.2.4 Sector Enable Register

Figure 13–6. Sector Enable Register

15–8
Reserved
R-x

7–4	3	2	1	0
Reserved	Sect 4 Enable	Sect 3 Enable	Sect 2 Enable	Sect 1 Enable
R-x	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 = value after reset, x = undefined

Bits 15–4 **Reserved**

Bits 3–0 **Sect 4 Enable – Sect 1 Enable.** Sector Enable

Each Sector enable bit is used to protect or enable write and erase operations for each of the defined sectors in the Flash array.

Bit Number	Bit Name	Sector Address Range	Operation
0	Sect En 1	0000 – 0FFFh	0 = Protect sector 1 = Enable prog / erase
1	Sect En 2	1000 – 3FFFh	0 = Protect sector 1 = Enable prog / erase
2	Sect En 3	4000 – 6FFFh	0 = Protect sector 1 = Enable prog / erase
3	Sect En 4	7000 – 7FFFh	0 = Protect sector 1 = Enable prog / erase

13.5 System Features

This section presents some the system features that are new to the ’240x devices. Understanding these key features may help system initialization and ’24x-to-’240x migration.

13.5.1 Oscillator and PLL

Unlike the ’24x device, the ’240x devices have a 5-pin PLL with a 3-bit ratio control to provide eight different CPU clock options. Table 13–7 describes the pins that are used for the PLL/Oscillator module and Table 13–8 lists the oscillator/PLL frequency input specification. PLL is preceded by an on-chip oscillator, which can accept a resonator or a crystal. The PLL accepts the on-chip oscillator or external clock as its input clock. Refer to the TMS320LF2407, TMS320LF2406, TMS320LF2402, TMS320LC2406, TMS320LC2404, TMS320LC2402 DSP Controllers Data Sheet (literature number SPRS094) for clock circuits and recommended values for the external filter components.

Table 13–7. ’240x PLL Pin Names

Pin Names	Description
XTAL1/CLKIN	Oscillator input, crystal and ceramic resonator input, or external clock input
XTAL2	Used only by crystal or ceramic resonators as an output
PLLF	PLL loop filter terminal 1
PLLV$_{CCA}$	PLL supply (3.3 V), to be connected to digital supply
PLLF2	PLL loop filter termianl 2

Table 13–8. Oscillator/PLL Frequency Input Specification

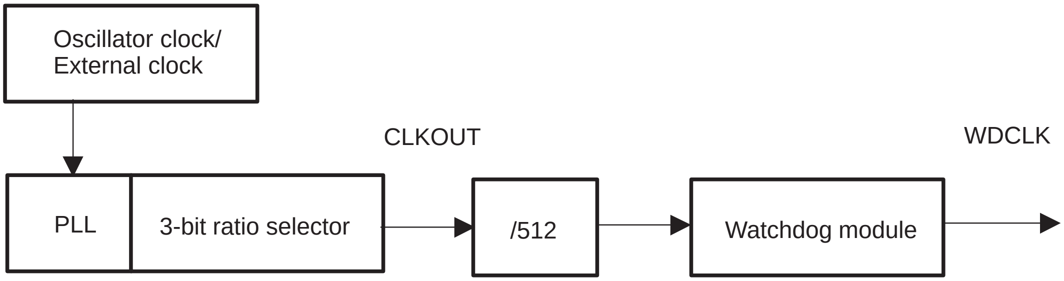
	Value
Input crystal frequency range	4–20 MHz
Input ceramic resonator frequency range	4–13 MHz
Input oscillator/CLOCKIN frequency range	4–20 MHz

13.5.2 Watchdog Clock

Watchdog clock generation logic is different in ’240x devices with respect to ’24x devices. Unlike the fixed PLL (x4) in ’24x, the ’240x devices have a variable clock from the PLL. This changes the input clock options for the watchdogmodule. The clock flow diagram below explains the watchdog clock generation logic.

’240x devices have a watchdog override bit in the SCSR2 register, which is similar to the WDDIS pin available on the ’24x devices. Refer to the description of SCSR2 register bit 5 (section 2.2.1 on page 2-3) for details on this bit function.

Figure 13–7. ’240x Watchdog Clock Generation Logic



13.5.2.1 Other Low-Power Management Features

All ’240x devices have a clock-enable bit in the SCSR1 register to save power and selectively enable peripheral functions. At reset, these peripheral clock-enable bits are disabled, and the applications software should enable the required modules. The peripheral clocks of the following peripherals can be independently enabled/disabled. See section 2.2.1 on page 2-3 for bit descriptions of the SCSR1 register.

Table 13–9. Peripheral Clock Enable Bits

Peripheral	SCSR1 Bits	Description
EVA	SCSR1.2	Event Manager A
EVB	SCSR1.3	Event Manager B
CAN	SCSR1.4	Controller Area Network
SPI	SCSR1.5	Serial Peripheral Interface
SCI	SCSR1.6	Serial Communications Interface
ADC	SCSR1.7	Analog-to-Digital Converter

13.5.3 System Control Registers

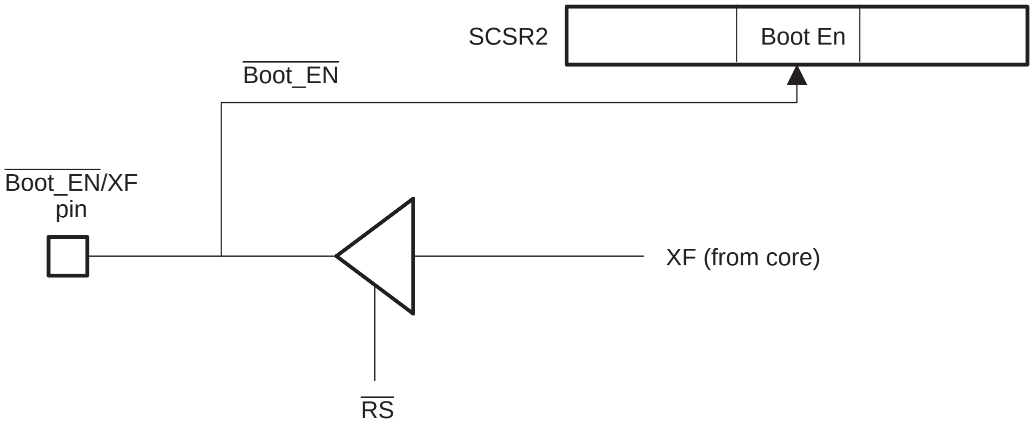
’240x devices have two system control and status registers: SCSR1 and SCSR2 (see section 2.2.1 on page 2-3). These registers have control and status bits for several on-chip modules. These register bits should be initialized after reset to enable/disable on-chip functionality for the selected application. ’24x has only one SCSR register and all its on-chip peripherals are powered up after reset. The SCSR2 register is unique to ’240x and its peripherals are disabled after reset.

13.5.3.1 Boot_EN/XF Pin Operation

During the Reset phase (i.e., RS low), this pin functions as a Boot_EN input pin, and its logic level is latched into bit 3 of the SCSR2 register. If the bit is set to 1, Boot ROM is active.

At the completion of the Reset phase (rising edge of RS), this pin will be XF output (“external flag”) function, and the Boot_EN function is no longer available through this pin. However, the Boot_EN bit in SCSR2 can be used to control the visibility of the Boot ROM or the Flash array.

Figure 13–8. Functional Block Diagram for Boot_EN/XF Feature

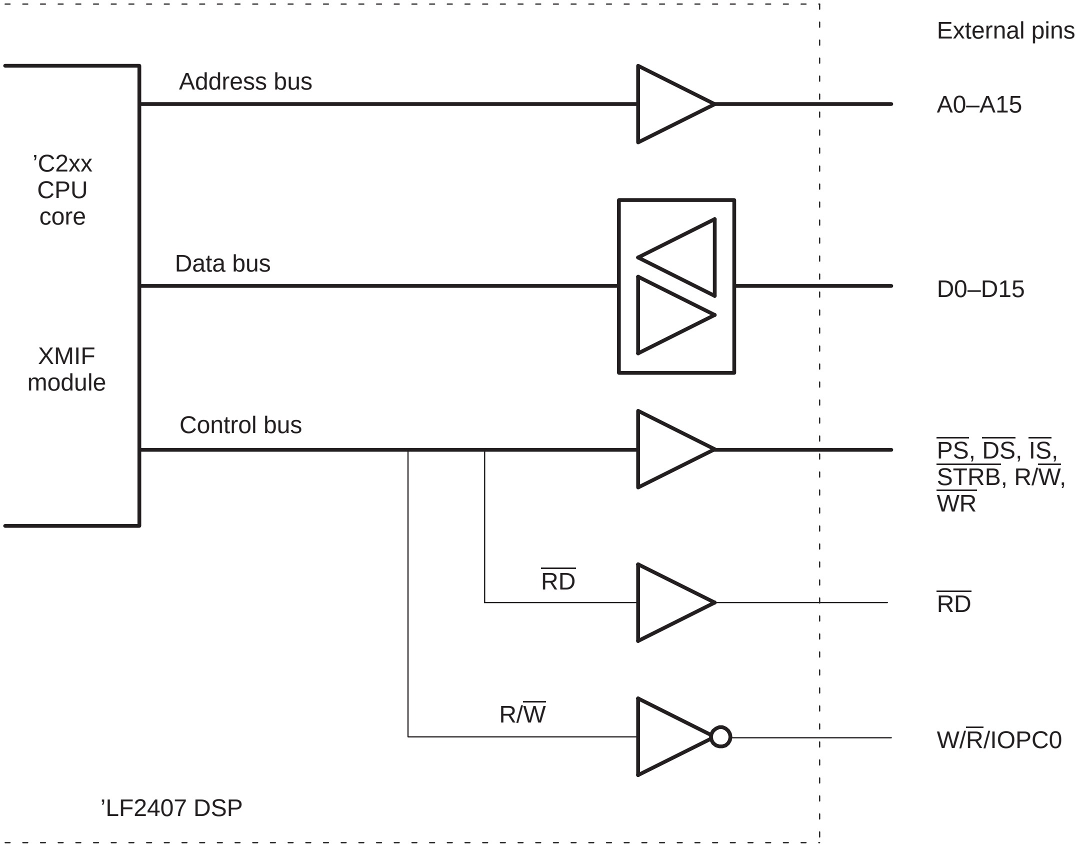


Operating Mode	Program Memory Active for the CPU	Boot_EN Pin / SCSR2 Bit 3	Comment
Functional	Boot ROM: 0x0000–0x00FF	0	Can be changed later by software bit in SCSR2
Functional	Flash Array: 0x0000–0x7FFF	1	Can be changed later by software bit in SCSR2

13.5.3.2 Fast $\overline{\text{RD}}$ Strobe Operation

’LF2407 is the only device that supports external memory interface (XMIF) to expand the internal memory space with the addition of external memory devices. The interface offers decode signals for Program, Data, and I/O space. ’LF2407 external memory interface signals have critical timings while interfacing zero-wait-state memory at higher CPU clock speeds. CPU memory reads are single-cycle and read-enable ($\overline{\text{RD}}$) timing is critical to meet the output-enable timing for memories that can be interfaced to this device. To alleviate the memory read interface timing, an additional signal W/$\overline{\text{R}}$ is provided to be used as output-enable signal instead of $\overline{\text{RD}}$. W/$\overline{\text{R}}$ is essentially an inverted R/$\overline{\text{W}}$ signal from the core. The W/$\overline{\text{R}}$ signal will remain all the time from reset and will go high during external write cycles. Refer to the memory interface timings in the TMS320LF2407, TMS320LF2406, TMS320LF2402, TMS320LC2406, TMS320LC2404, TMS320LC2402 DSP Controllers Data Sheet (literature number SPRS094) for additional timing details.

Figure 13–9. Functional Block Diagram of XMIF Signals on ’LF2407



13.6 Digital I/O (GPIO Pins)

Some members of the ’240x family have more GPIO pins than the ’24x devices. This necessitates additional registers. The bit definitions for some multiplexed pins such as XF, CLKOUT, etc. are different from those of the ’24x. See Chapter 5 for more details.

Note that when multiplexed I/O pins are in input mode, the pin is connected to both the I/O data register *and* the shared peripheral. The I/O Mux control register (**MCRx**) in ’240x devices is synonymous to the **OCRx** in ’24x devices. Both MCRx and OCRx have the same function in ’240x and ’24x devices, respectively.

13.6.1 Digital I/O and Shared Pin Functions for the ’240x

’LF2407 device has a total of 41 pins shared between Primary functions and I/Os.

Table 13–10 lists all the pins that are shared between the Primary functions and the dedicated I/O Ports A, B, C, D, E, F.

Table 13–10. ’LF2407 Shared Pin Configuration

Shared Pin Functions		Mux Control Register	Mux Control Bit #	MCRx.n Value at Reset	IOP Data and Direction Register	IOP Data Bit #	IOP Direction Bit #
Primary Function (1)	I/O (0)						
SCITXD	IOPA0	MCRA	0	0	PADATDIR	0	8
SCIRXD	IOPA1	MCRA	1	0	PADATDIR	1	9
XINT1	IOPA2	MCRA	2	0	PADATDIR	2	10
CAP1/QEP1	IOPA3	MCRA	3	0	PADATDIR	3	11
CAP2/QEP2	IOPA4	MCRA	4	0	PADATDIR	4	12
CAP3	IOPA5	MCRA	5	0	PADATDIR	5	13
PWM1	IOPA6	MCRA	6	0	PADATDIR	6	14
PWM2	IOPA7	MCRA	7	0	PADATDIR	7	15
PWM3	IOPB0	MCRA	8	0	PBDATDIR	0	8
PWM4	IOPB1	MCRA	9	0	PBDATDIR	1	9
PWM5	IOPB2	MCRA	10	0	PBDATDIR	2	10
PWM6	IOPB3	MCRA	11	0	PBDATDIR	3	11
T1PWM/CMP	IOPB4	MCRA	12	0	PBDATDIR	4	12
T2PWM/CMP	IOPB5	MCRA	13	0	PBDATDIR	5	13
TDIRA	IOPB6	MCRA	14	0	PBDATDIR	6	14
TCLKINA	IOPB7	MCRA	15	0	PBDATDIR	7	15
W/$\overline{\text{R}}$	IOPC0	MCRB	0	1	PCDATDIR	0	8
$\overline{\text{BIO}}$	IOPC1	MCRB	1	1	PCDATDIR	1	9
SPISIMO	IOPC2	MCRB	2	0	PCDATDIR	2	10
SPISOMI	IOPC3	MCRB	3	0	PCDATDIR	3	11
SPICLK	IOPC4	MCRB	4	0	PCDATDIR	4	12
SPISTE	IOPC5	MCRB	5	0	PCDATDIR	5	13
CANTX	IOPC6	MCRB	6	0	PCDATDIR	6	14
CANRX	IOPC7	MCRB	7	0	PCDATDIR	7	15

Table 13–10. ’LF2407 Shared Pin Configuration (Continued)

Shared Pin Functions		Mux Control Register	Mux Control Bit #	MCRx.n Value at Reset	IOP Data and Direction Register	IOP Data Bit #	IOP Direction Bit #
Primary Function (1)	**I/O (0)**						
XINT2/ADCSOC	IOPD0	MCRB	8	0	PDDATDIR	0	8
EMU0	IOPD1	Reserved	9	1	PDDATDIR	–	–
EMU1	IOPD2	Reserved	10	1	PDDATDIR	–	–
TCK	IOPD3	Reserved	11	1	PDDATDIR	–	–
TDI	IOPD4	Reserved	12	1	PDDATDIR	–	–
TDO	IOPD5	Reserved	13	1	PDDATDIR	–	–
TMS	IOPD6	Reserved	14	1	PDDATDIR	–	–
TMS2	IOPD7	Reserved	15	1	PDDATDIR	–	–
CLKOUT	IOPE0	MCRC	0	1	PEDATDIR	0	8
PWM7	IOPE1	MCRC	1	0	PEDATDIR	1	9
PWM8	IOPE2	MCRC	2	0	PEDATDIR	2	10
PWM9	IOPE3	MCRC	3	0	PEDATDIR	3	11
PWM10	IOPE4	MCRC	4	0	PEDATDIR	4	12
PWM11	IOPE5	MCRC	5	0	PEDATDIR	5	13
PWM12	IOPE6	MCRC	6	0	PEDATDIR	6	14
CAP4/QEP3	IOPE7	MCRC	7	0	PEDATDIR	7	15
CAP5/QEP4	IOPF0	MCRC	8	0	PFDATDIR	0	8
CAP6	IOPF1	MCRC	9	0	PFDATDIR	1	9
T3PWM/CMP	IOPF2	MCRC	10	0	PFDATDIR	2	10
T4PWM/CMP	IOPF3	MCRC	11	0	PFDATDIR	3	11
TDIRB	IOPF4	MCRC	12	0	PFDATDIR	4	12
TCLKINB	IOPF5	MCRC	13	0	PFDATDIR	5	13
IOPF6	IOPF6	X	X	X	PFDATDIR	6	14

13.7 Event Manager Module (EVB)

The event manager module available on ’240x devices is identical to the event manager in the ’24x family. EVA and EVB are exactly identical modules, except that their registers start at 7400h and 7500h, respectively, in the peripheral space. The functional description of the event manager, available in the *TMS320F243/F241/C242 DSP Controllers Reference Guide* (literature number SPRU276C), is applicable to EVB as well. EVA and EVB modules and signals are uniquely identified and in Table 13–11 for comparison.

Table 13–11. Event Manager Module and Signal Names for EVA and EVB

EV Modules	EVA Modules	EVA Pins	EVB Modules	EVB Pins
GP Timers	Timer 1	T1PWM/T1CMP	Timer 3	T3PWM/T3CMP
	Timer 2	T2PWM/T2CMP	Timer 4	T4PWM/T4CMP
Compare Units	Compare 1	PWM1/2	Compare 4	PWM7/8
	Compare 2	PWM3/4	Compare 5	PWM9/10
	Compare 3	PWM5/6	Compare 6	PWM11/12
Capture Units	Capture 1	CAP1	Capture 4	CAP4
	Capture 2	CAP2	Capture 5	CAP5
	Capture 3	CAP3	Capture 6	CAP6
QEP	QEP 1	QEP1	QEP 3	QEP3
	QEP 2	QEP2	QEP 4	QEP4
External Inputs	Direction	TDIRA	Direction	TDIRB
	External Clock	TCLKINA	External Clock	TCLKINB

Appendix A

Programmable Register Address Summary

Table A–1 shows the peripheral register map for ’240x devices. The shaded table entries represent the registers that are in addition to the ’24x registers. These details are also explained in the TMS320LF2407, TMS320LF2406, TMS320LF2402, TMS320LC2406, TMS320LC2404, TMS320LC2402 DSP Controllers Data Sheet (literature number SPRS094). Note that:

❑ The ADC registers are completely different from those on the ’24x

❑ The ADC register map has been moved from 7030h to 70A0h

❑ The second event manager (EVB) has been placed at 7500h

Table A–1. Summary of Programmable Registers on the ’240x

Data Memory Address	Register Mnemonic	Register Name	Data Page	Page
		Interrupt and System		
7010h	PIRQR0	Peripheral Interrupt Request Register 0	E0h (224)	2-28
7011h	PIRQR1	Peripheral Interrupt Request Register 1	E0h (224)	2-29
7012h	PIRQR2	Peripheral Interrupt Request Register 2	E0h (224)	2-30
7014h	PIACKR0	Peripheral Interrupt Acknowledge Register 0	E0h (224)	2-31
7015h	PIACKR1	Peripheral Interrupt Acknowledge Register 1	E0h (224)	2-32
7016h	PIACKR2	Peripheral Interrupt Acknowledge Register 2	E0h (224)	2-33
7018h	SCSR1	System Control and Status Register 1	E0h (224)	2-3
7019h	SCSR2	System Control and Status Register 2	E0h (224)	2-5
701Ch	DINR	Device Identification Number Register	E0h (224)	2-7
701Eh	PIVR	Peripheral Interrupt Vector Register	E0h (224)	2-27
		Watchdog		
7023h	WDCNTR	Watchdog Counter Register	E0h (224)	11-8
7025h	WDKEY	Watchdog Reset Key Register	E0h (224)	11-9
7029h	WDCR	Watchdog Timer Control Register	E0h (224)	11-9
		Serial Peripheral Interface (SPI)		
7040h	SPICCR	SPI Configuration Control Register	E0h (224)	9-18
7041h	SPICTL	SPI Operation Control Register	E0h (224)	9-20
7042h	SPISTS	SPI Status Register	E0h (224)	9-21
7044h	SPIBRR	SPI Baud Rate Register	E0h (224)	9-23
7046h	SPIRXEMU	SPI Emulation Buffer Register	E0h (224)	9-24
7047h	SPIRXBUF	SPI Serial Receive Buffer Register	E0h (224)	9-25
7048h	SPITXBUF	SPI Serial Transmit Buffer Register	E0h (224)	9-26
7049h	SPIDAT	SPI Serial Data Register	E0h (224)	9-27
704Fh	SPIPRI	SPI Priority Control Register	E0h (224)	9-28
		Serial Communications Interface (SCI)		
7050h	SCICCR	SCI Communication Control Register	E0h (224)	8-20
7051h	SCICTL1	SCI Control Register 1	E0h (224)	8-22
7052h	SCIHBAUD	SCI Baud-Select Register, high bits	E0h (224)	8-25
7053h	SCILBAUD	SCI Baud-Select Register, low bits	E0h (224)	8-25

Table A–1. Summary of Programmable Registers on the ’240x (Continued)

Data Memory Address	Register Mnemonic	Register Name	Data Page	Page
7054h	SCICTL2	SCI Control Register 2	E0h (224)	8-26
7055h	SCIRXST	SCI Receiver Status Register	E0h (224)	8-27
7056h	SCIRXEMU	SCI Emulation Data Buffer Register	E0h (224)	8-30
7057h	SCIRXBUF	SCI Receiver Data Buffer Register	E0h (224)	8-30
7059h	SCITXBUF	SCI Transmit Data Buffer Register	E0h (224)	8-30
705Fh	SCIPRI	SCI Priority Control Register	E0h (224)	8-31
		External Interrupt		
7070h	XINT1CR	External Interrupt 1 Control Register	E0h (224)	2-36
7071h	XINT2CR	External Interrupt 2 Control Register	E0h (224)	2-37
		Digital I/O		
7090h	MCRA	I/O Mux Control Register A	E1h (225)	5-5
7092h	MCRB	I/O Mux Control Register B	E1h (225)	5-6
7094h	MCRC	I/O Mux Control Register C	E1h (225)	5-8
7098h	PADATDIR	Port A Data and Direction Control Register	E1h (225)	5-9
709Ah	PBDATDIR	Port B Data and Direction Control Register	E1h (225)	5-11
709Ch	PCDATDIR	Port C Data and Direction Control Register	E1h (225)	5-12
709Eh	PDDATDIR	Port D Data and Direction Control Register	E1h (225)	5-13
7095h	PEDATDIR	Port E Data and Direction Control Register	E1h (225)	5-14
7096h	PFDATDIR	Port F Data and Direction Control Register	E1h (225)	5-15
		Analog-to-Digital Converter (ADC) (10-Bit)		
70A0h	ADCTRL1	ADC Control Register 1	E1h (225)	7-20
70A1h	ADCTRL2	ADC Control Register 2	E1h (225)	7-23
70A2h	MAX_CONV	Maximum Conversion Channels Register	E1h (225)	7-27
70A3h	CHSELSEQ1	Channel Select Sequencing Control Register 1	E1h (225)	7-31
70A4h	CHSELSEQ2	Channel Select Sequencing Control Register 2	E1h (225)	7-31
70A5h	CHSELSEQ3	Channel Select Sequencing Control Register 3	E1h (225)	7-31

Table A–1. Summary of Programmable Registers on the ’240x (Continued)

Data Memory Address	Register Mnemonic	Register Name	Data Page	Page
70A6h	CHSELSEQ4	Channel Select Sequencing Control Register 4	E1h (225)	7-31
70A7h	AUTO_SEQ_SR	Autosequence Status Register	E1h (225)	7-29
70A8h	RESULT0	Conversion Result Buffer Register 0	E1h (225)	–
70A9h	RESULT1	Conversion Result Buffer Register 1	E1h (225)	–
70AAh	RESULT2	Conversion Result Buffer Register 2	E1h (225)	–
70ABh	RESULT3	Conversion Result Buffer Register 3	E1h (225)	–
70ACh	RESULT4	Conversion Result Buffer Register 4	E1h (225)	–
70ADh	RESULT5	Conversion Result Buffer Register 5	E1h (225)	–
70AEh	RESULT6	Conversion Result Buffer Register 6	E1h (225)	–
70AFh	RESULT7	Conversion Result Buffer Register 7	E1h (225)	–
70B0h	RESULT8	Conversion Result Buffer Register 8	E1h (225)	–
70B1h	RESULT9	Conversion Result Buffer Register 9	E1h (225)	–
70B2h	RESULT10	Conversion Result Buffer Register 10	E1h (225)	–
70B3h	RESULT11	Conversion Result Buffer Register 11	E1h (225)	–
70B4h	RESULT12	Conversion Result Buffer Register 12	E1h (225)	–
70B5h	RESULT13	Conversion Result Buffer Register 13	E1h (225)	–
70B6h	RESULT14	Conversion Result Buffer Register 14	E1h (225)	–
70B7h	RESULT15	Conversion Result Buffer Register 15	E1h (225)	–
70B8h	CALIBRATION	Calibration result which is used to correct subsequent conversions	E1h (225)	–
		Controller Area Network (CAN)		
7100h	MDER	Mailbox Direction/Enable Register	E2h (226)	10-18
7101h	TCR	Transmission Control Register	E2h (226)	10-19
7102h	RCR	Receive Control Register	E2h (226)	10-21
7103h	MCR	Master Control Register	E2h (226)	10-22
7104h	BCR2	Bit Configuration Register 2	E2h (226)	10-25
7105h	BCR1	Bit Configuration Register 1	E2h (226)	10-25
7106h	ESR	Error Status Register	E2h (226)	10-29
7107h	GSR	Global Status Register	E2h (226)	10-28
7108h	CEC	Transmit and Receive Error Counters	E2h (226)	10-31
7109h	CAN_IFR	Interrupt Flag Register	E2h (226)	10-33
710Ah	CAN_IMR	Interrupt Mask Register	E2h (226)	10-35

Table A–1. Summary of Programmable Registers on the ’240x (Continued)

Data Memory Address	Register Mnemonic	Register Name	Data Page	Page
710Bh	LAM0_H	Local Acceptance Mask (MBOX 0 and MBOX 1)	E2h (226)	10-17
710Ch	LAM0_L	Local Acceptance Mask (MBOX 0 and MBOX 1)	E2h (226)	10-17
710Dh	LAM1_H	Local Acceptance Mask (MBOX 2 and MBOX 3)	E2h (226)	10-17
710Eh	LAM1_L	Local Acceptance Mask (MBOX 2 and MBOX 3)	E2h (226)	10-17
7200h	MSGID0L	CAN Message ID for Mailbox 0 (lower 16 bits)	E4h (228)	10-11
7201h	MSGID0H	CAN Message ID for Mailbox 0 (upper 16 bits)	E4h (228)	10-10
7202h	MSGCTRL0	MBOX 0 RTR and DLC	E4h (228)	10-11
7204h	MBOX0A	CAN 2 of 8 bytes of Mailbox 0	E4h (228)	–
7205h	MBOX0B	CAN 2 of 8 bytes of Mailbox 0	E4h (228)	–
7206h	MBOX0C	CAN 2 of 8 bytes of Mailbox 0	E4h (228)	–
7207h	MBOX0D	CAN 2 of 8 bytes of Mailbox 0	E4h (228)	–
7208h	MSGID1L	CAN Message ID for mailbox 1 (lower 16 bits)	E4h (228)	10-11
7209h	MSGID1H	CAN Message ID for mailbox 1 (upper 16 bits)	E4h (228)	10-10
720Ah	MSGCTRL1	MBOX 1 RTR and DLC	E4h (228)	10-11
720Ch	MBOX1A	CAN 2 of 8 bytes of Mailbox 1	E4h (228)	–
720Dh	MBOX1B	CAN 2 of 8 bytes of Mailbox 1	E4h (228)	–
720Eh	MBOX1C	CAN 2 of 8 bytes of Mailbox 1	E4h (228)	–
720Fh	MBOX1D	CAN 2 of 8 bytes of Mailbox 1	E4h (228)	–
7210h	MSGID2L	CAN Message ID for mailbox 2 (lower 16 bits)	E4h (228)	10-11
7211h	MSGID2H	CAN Message ID for mailbox 2 (upper 16 bits)	E4h (228)	10-10
7212h	MSGCTRL2	MBOX 2 RTR and DLC	E4h (228)	10-11
7214h	MBOX2A	CAN 2 of 8 bytes of Mailbox 2	E4h (228)	–
7215h	MBOX2B	CAN 2 of 8 bytes of Mailbox 2	E4h (228)	–
7216h	MBOX2C	CAN 2 of 8 bytes of Mailbox 2	E4h (228)	–
7217h	MBOX2D	CAN 2 of 8 bytes of Mailbox 2	E4h (228)	–

Table A–1. Summary of Programmable Registers on the ’240x (Continued)

Data Memory Address	Register Mnemonic	Register Name	Data Page	Page
7218h	MSGID3L	CAN Message ID for Mailbox 3 (lower 16 bits)	E4h (228)	10-11
7219h	MSGID3H	CAN Message ID for Mailbox 3 (upper 16 bits)	E4h (228)	10-10
721Ah	MSGCTRL3	MBOX 3 RTR and DLC	E4h (228)	10-11
721Ch	MBOX3A	CAN 2 of 8 bytes of Mailbox 3	E4h (228)	–
721Dh	MBOX3B	CAN 2 of 8 bytes of Mailbox 3	E4h (228)	–
721Eh	MBOX3C	CAN 2 of 8 bytes of Mailbox 3	E4h (228)	–
721Fh	MBOX3D	CAN 2 of 8 bytes of Mailbox 3	E4h (228)	–
7220h	MSGID4L	CAN Message ID for Mailbox 4 (lower 16 bits)	E4h (228)	10-11
7221h	MSGID4H	CAN Message ID for Mailbox 4 (upper 16 bits)	E4h (228)	10-10
7222h	MSGCTRL4	MBOX 4 RTR and DLC	E4h (228)	10-11
7224h	MBOX4A	CAN 2 of 8 bytes of Mailbox 4	E4h (228)	–
7225h	MBOX4B	CAN 2 of 8 bytes of Mailbox 4	E4h (228)	–
7226h	MBOX4C	CAN 2 of 8 bytes of Mailbox 4	E4h (228)	–
7227h	MBOX4D	CAN 2 of 8 bytes of Mailbox 4	E4h (228)	–
7228h	MSGID5L	CAN Message ID for Mailbox 5 (lower 16 bits)	E4h (228)	10-11
7229h	MSGID5H	CAN Message ID for Mailbox 5 (upper 16 bits)	E4h (228)	10-10
722Ah	MSGCTRL5	MBOX 5 RTR and DLC	E4h (228)	10-11
722Ch	MBOX5A	CAN 2 of 8 bytes of Mailbox 5	E4h (228)	–
722Dh	MBOX5B	CAN 2 of 8 bytes of Mailbox 5	E4h (228)	–
722Eh	MBOX5C	CAN 2 of 8 bytes of Mailbox 5	E4h (228)	–
722Fh	MBOX5D	CAN 2 of 8 bytes of Mailbox 5	E4h (228)	–
		Event Manager A (EVA)		
7400h	GPTCONA	GP Timer Control Register A	E8h (232)	6-33
7401h	T1CNT	Timer 1 Counter Register	E8h (232)	–
7402h	T1CMPR	Timer 1 Compare Register	E8h (232)	–
7403h	T1PR	Timer 1 Period Register	E8h (232)	–
7404h	T1CON	Timer 1 Control Register	E8h (232)	6-31
7405h	T2CNT	Timer 2 Counter Register	E8h (232)	–

Table A–1. Summary of Programmable Registers on the ’240x (Continued)

Data Memory Address	Register Mnemonic	Register Name	Data Page	Page
7406h	T2CMPR	Timer 2 Compare Register	E8h (232)	–
7407h	T2PR	Timer 2 Period Register	E8h (232)	–
7408h	T2CON	Timer 2 Control Register	E8h (232)	6-31
7411h	COMCONA	Compare Control Register A	E8h (232)	6-39
7413h	ACTRA	Compare Action Control Register A	E8h (232)	6-42
7415h	DBTCONA	Dead-Band Timer Control Register A	E8h (232)	6-48
7417h	CMPR1	Compare Register 1	E8h (232)	–
7418h	CMPR2	Compare Register 2	E8h (232)	–
7419h	CMPR3	Compare Register 3	E8h (232)	–
7420h	CAPCONA	Capture Control Register A	E8h (232)	6-70
7422h	CAPFIFOA	Capture FIFO Status Register A	E8h (232)	6-74
7423h	CAP1FIFO	Two-Level-Deep Capture FIFO stack 1	E8h (232)	–
7424h	CAP2FIFO	Two-Level-Deep Capture FIFO stack 2	E8h (232)	–
7425h	CAP3FIFO	Two-Level-Deep Capture FIFO stack 3	E8h (232)	–
7427h	CAP1FBOT	Bottom Register of Capture FIFO stack 1	E8h (232)	–
7428h	CAP2FBOT	Bottom Register of Capture FIFO stack 2	E8h (232)	–
7429h	CAP3FBOT	Bottom Register of Capture FIFO stack 3	E8h (232)	–
742Ch	EVAIMRA	EVA Interrupt Mask Register A	E8h (232)	6-89
742Dh	EVAIMRB	EVA Interrupt Mask Register B	E8h (232)	6-90
742Eh	EVAIMRC	EVA Interrupt Mask Register C	E8h (232)	6-90
742Fh	EVAIFRA	EVA Interrupt Flag Register A	E8h (232)	6-85
7430h	EVAIFRB	EVA Interrupt Flag Register B	E8h (232)	6-87
7431h	EVAIFRC	EVA Interrupt Flag Register C	E8h (232)	6-88
		Event Manager B (EVB)		
7500h	GPTCONB	GP Timer Control Register B	EAh (234)	6-34
7501h	T3CNT	Timer 3 Counter Register	EAh (234)	–
7502h	T3CMPR	Timer 3 Compare Register	EAh (234)	–
7503h	T3PR	Timer 3 Period Register	EAh (234)	–
7504h	T3CON	Timer 3 Control Register	EAh (234)	6-31
7505h	T4CNT	Timer 4 Counter Register	EAh (234)	–

Table A–1. Summary of Programmable Registers on the ’240x (Continued)

Data Memory Address	Register Mnemonic	Register Name	Data Page	Page
7506h	T4CMPR	Timer 4 Compare Register	EAh (234)	–
7507h	T4PR	Timer 4 Period Register	EAh (234)	–
7508h	T4CON	Timer 4 Control Register	EAh (234)	6-31
7511h	COMCONB	Compare Control Register B	EAh (234)	6-41
7513h	ACTRB	Compare Action Control Register B	EAh (234)	6-44
7515h	DBTCONB	Dead-Band Timer Control Register B	EAh (234)	6-49
7517h	CMPR4	Compare Register 4	EAh (234)	–
7518h	CMPR5	Compare Register 5	EAh (234)	–
7519h	CMPR6	Compare Register 6	EAh (234)	–
7520h	CAPCONB	Capture Control Register B	EAh (234)	6-72
7522h	CAPFIFOB	Capture FIFO Status Register B	EAh (234)	6-75
7523h	CAP4FIFO	Two-Level-Deep Capture FIFO stack 4	EAh (234)	–
7524h	CAP5FIFO	Two-Level-Deep Capture FIFO stack 5	EAh (234)	–
7525h	CAP6FIFO	Two-Level-Deep Capture FIFO stack 6	EAh (234)	–
7527h	CAP4FBOT	Bottom Register of Capture FIFO stack 4	EAh (234)	–
7528h	CAP5FBOT	Bottom Register of Capture FIFO stack 5	EAh (234)	–
7529h	CAP6FBOT	Bottom Register of Capture FIFO stack 6	EAh (234)	–
752Ch	EVBIMRA	EVB Interrupt Mask Register A	EAh (234)	6-95
752Dh	EVBIMRB	EVB Interrupt Mask Register B	EAh (234)	6-96
752Eh	EVBIMRC	EVB Interrupt Mask Register C	EAh (234)	6-96
752Fh	EVBIFRA	EVB Interrupt Flag Register A	EAh (234)	6-91
7530h	EVBIFRB	EVB Interrupt Flag Register B	EAh (234)	6-93
7531h	EVBIFRC	EVB Interrupt Flag Register C	EAh (234)	6-94
IO-FF0Fh	FCMR	Flash Control Mode Register	EAh (234)	–
IO-FFFFh	WSGR	Wait State Generator Register	EAh (234)	3-22

Appendix B

Program Examples

This appendix provides:

- A brief introduction to the tools used for generating executable COFF files that run on the ’240x devices.

- Sample programs to test some of the peripherals available in the ’240x devices.

This appendix is not intended to teach you how to use the software development tools. The following documents cover these tools in detail:

TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User’s Guide
(literature number SPRU018)

TMS320C2x/C2xx/C5x Optimizing C Compiler User’s Guide
(literature number SPRU024)

TMS320C2xx C Source Debugger User’s Guide
(literature number SPRU151)

For further information about ordering these documents, see *Related Documentation From Texas Instruments* on page v of the Preface.

Topic **Page**

B.1 About These Program Examples

Figure B–1 illustrates the basic process for generating executable COFF files:

1) Use any ASCII editor to create:
   - ❑ An assembly language program (*test.asm* in the figure)
   - ❑ A linker command file (*240x.cmd* in the figure) that defines address ranges according to the architecture of the particular device and where the various sections of the user code should be located
2) Assemble the program. The command shown under Step 2 in the figure generates an object file (.obj) and list file (.lst) containing a listing of assembler messages.
3) Use the linker to bring together the information in the object file and the command file and create an executable file (*test.out* in the figure). The command shown also generates a map file, which explains how the linker assigned the individual sections in the memory.

Note:

The procedure here applies to the PC™ development environment and is given only as an example.

Figure B–1. Procedure for Generating Executable Files

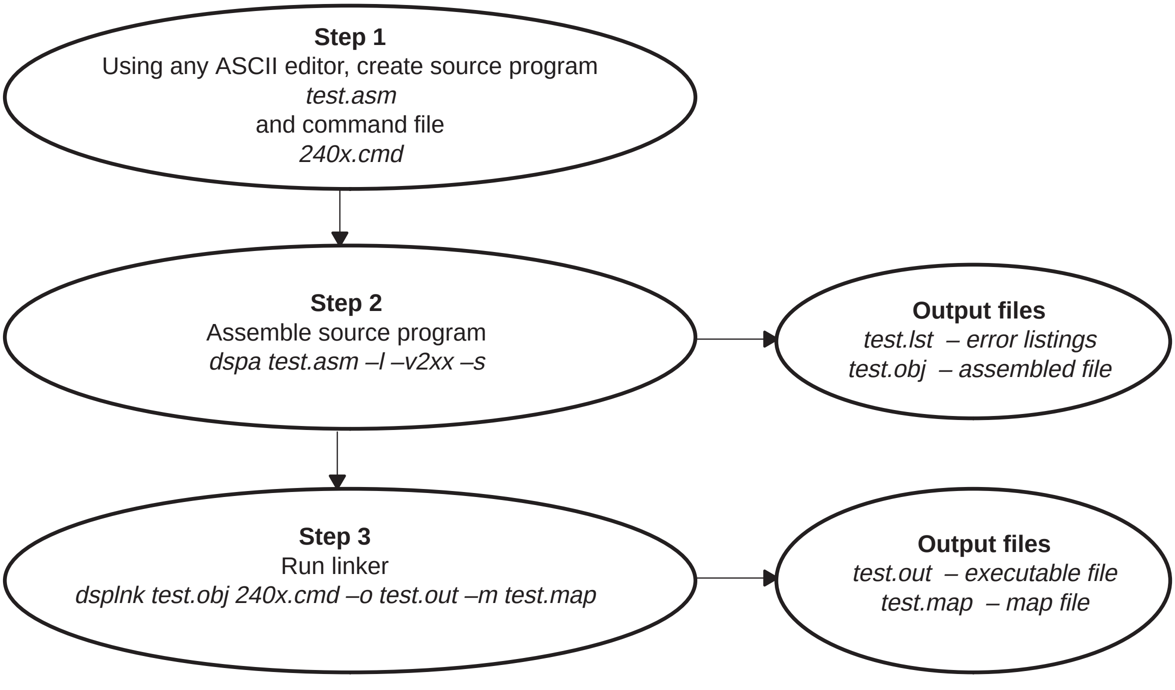

Step 1
Using any ASCII editor, create source program
test.asm
and command file
240x.cmd
Step 2
Assemble source program
dspa test.asm –l –v2xx –s
Output files
test.lst – error listings
test.obj – assembled file
Step 3
Run linker
dsplnk test.obj 240x.cmd –o test.out –m test.map
Output files
test.out – executable file
test.map – map file

Table B–1. Common Files For All Example Programs

Program	Functional Description
240x_PM.cmd	Linker command file that defines the program, data, and I/O memory maps of the target hardware. It also locates the various sections in the user code into predetermined segments of memory. This .cmd file locates user code (vectors and .text sections) in program memory beginning at 0000h.
240x.h	Header file that designates labels for the addresses of the various registers.
vector.h	File that contains the vectors for various interrupts.

Table B–2. Program Examples

Program	Functional Description
SPI.asm	Program to output serial data through the SPI port
SCI.asm	Program to check the SCI module in ’240x
ADC.asm	Program to check ADC of ’240x
GPIO_OUT.asm	Program that checks GPIO pins of ’240x as outputs
GPIO_IN.asm	Program that checks GPIO pins of ’240x as inputs
REM_ANS.asm REM_REQ.asm	Programs that perform RTR (Remote Transmission Request) operations in the CAN module
EV_T1INT.asm	Program to check the operation of timer 1 in EVA
CAP.asm	Program to check the operation of capture units in the EV modules

B.2 Program Examples

```
/******************************************************************************/
/* File Name:  240x_PM.cmd                                                     */
/* Description:   Linker command file to place user code (vectors & .text)    */
/* sections beginning at 0000h of program memory. .text is loaded at 40h.     */
/* This file should be modified if it is desired to load code in B0 memory or */
/* if on–chip SARAM is to be used.                                            */
/******************************************************************************/

MEMORY
{

PAGE 0:                                /* PROGRAM MEMORY                      */

EX1_PM  :ORIGIN=0h    ,  LENGTH=0FDFFh /* 63.5K external RAM                  */
B0_PM   :ORIGIN=0FE00h,  LENGTH=0200h  /* On–chip DARAM if CNF=1, else        */
                                       /* external                            */
                                       /* B0 = FE00 to FEFF or FF00 to FFFF   */

PAGE 1:                                /* DATA MEMORY                         */

REGS    :ORIGIN=0h    ,  LENGTH=60h    /* Memory mapped regs & reservd address*/
BLK_B2  :ORIGIN=60h   ,  LENGTH=20h    /* Block B2                            */
BLK_B0  :ORIGIN=200h  ,  LENGTH=200h   /* Block B0, On–chip DARAM if CNF=0    */
BLK_B1  :ORIGIN=300h  ,  LENGTH=200h   /* Block B1                            */
EX1_DM  :ORIGIN=0800h ,  LENGTH=7800h  /* External data RAM  1                */
EX2_DM  :ORIGIN=8000h ,  LENGTH=8000h  /* External data RAM  2                */

PAGE 2:                                /* I/O MEMORY                          */

IO_IN   :ORIGIN=0FFF0h, LENGTH=0Fh     /* On–chip I/O mapped peripherals      */
IO_EX   :ORIGIN=0000h , LENGTH=0FFF0h  /* External I/O mapped peripherals     */


}

SECTIONS

{

        vectors :{}  > EX1_PM   PAGE 0
       .text    :{}  > EX1_PM   PAGE 0
       .bss     :{}  > BLK_B2   PAGE 1
       .data    :{}  > BLK_B1   PAGE 1
}
```

```
;***************************************************************************
; File name:  240x.h
;
; Description:  240x register definitions, Bit codes for BIT instruction
;***************************************************************************

; 240x CPU core registers

IMR                 .set 0004h     ; Interrupt Mask Register
IFR                 .set 0006h     ; Interrupt Flag Register

; System configuration and interrupt registers

SCSR1               .set 7018h     ; System Control &  Status register. 1
SCSR2               .set 7019h     ; System Control &  Status register. 2
DINR                .set 701Ch     ; Device Identification Number register.
PIVR                .set 701Eh     ; Peripheral Interrupt Vector register.
PIRQR0              .set 7010h     ; Peripheral Interrupt Request register 0
PIRQR1              .set 7011h     ; Peripheral Interrupt Request register 1
PIRQR2              .set 7012h     ; Peripheral Interrupt Request register 2
PIACKR0             .set 7014h     ; Peripheral Interrupt Acknowledge register 0
PIACKR1             .set 7015h     ; Peripheral Interrupt Acknowledge register 1
PIACKR2             .set 7016h     ; Peripheral Interrupt Acknowledge register 2

; External interrupt configuration registers

XINT1CR             .set 7070h     ; External interrupt 1 control register
XINT2CR             .set 7071h     ; External interrupt 2 control register

; Digital I/O registers

MCRA                .set 7090h     ; I/O Mux Control Register A
MCRB                .set 7092h     ; I/O Mux Control Register B
MCRC                .set 7094h     ; I/O Mux Control Register C
PADATDIR            .set 7098h     ; I/O port A Data & Direction register
PBDATDIR            .set 709Ah     ; I/O port B Data & Direction register
PCDATDIR            .set 709Ch     ; I/O port C Data & Direction register
PDDATDIR            .set 709Eh     ; I/O port D Data & Direction register
PEDATDIR            .set 7095h     ; I/O port E Data & Direction register
PFDATDIR            .set 7096h     ; I/O port F Data & Direction register

; Watchdog (WD) registers

WDCNTR              .set 7023h     ; WD Counter register
WDKEY               .set 7025h     ; WD Key register
WDCR                .set 7029h     ; WD Control register

; ADC registers

ADCTRL1             .set 70A0h     ; ADC Control register 1
ADCTRL2             .set 70A1h     ; ADC Control register 2
MAXCONV             .set 70A2h     ; Maximum conversion channels register
CHSELSEQ1           .set 70A3h     ; Channel select Sequencing control register 1
CHSELSEQ2           .set 70A4h     ; Channel select Sequencing control register 2
CHSELSEQ3           .set 70A5h     ; Channel select Sequencing control register 3
```

```
CHSELSEQ4          .set 70A6h     ; Channel select Sequencing control register 4
AUTO_SEQ_SR        .set 70A7h     ; Auto–sequence status register
RESULT0            .set 70A8h     ; Conversion result buffer register 0
RESULT1            .set 70A9h     ; Conversion result buffer register 1
RESULT2            .set 70Aah     ; Conversion result buffer register 2
RESULT3            .set 70Abh     ; Conversion result buffer register 3
RESULT4            .set 70Ach     ; Conversion result buffer register 4
RESULT5            .set 70Adh     ; Conversion result buffer register 5
RESULT6            .set 70Aeh     ; Conversion result buffer register 6
RESULT7            .set 70Afh     ; Conversion result buffer register 7
RESULT8            .set 70B0h     ; Conversion result buffer register 8
RESULT9            .set 70B1h     ; Conversion result buffer register 9
RESULT10           .set 70B2h     ; Conversion result buffer register 10
RESULT11           .set 70B3h     ; Conversion result buffer register 11
RESULT12           .set 70B4h     ; Conversion result buffer register 12
RESULT13           .set 70B5h     ; Conversion result buffer register 13
RESULT14           .set 70B6h     ; Conversion result buffer register 14
RESULT15           .set 70B7h     ; Conversion result buffer register 15
CALIBRATION        .set 70B8h     ; Calib result, used to correct
                                  ; subsequent conversions

; SPI registers

SPICCR             .set 7040h     ; SPI Config Control register
SPICTL             .set 7041h     ; SPI Operation Control register
SPISTS             .set 7042h     ; SPI Status register
SPIBRR             .set 7044h     ; SPI Baud rate control register
SPIRXEMU           .set 7046h     ; SPI Emulation buffer register
SPIRXBUF           .set 7047h     ; SPI Serial receive buffer register
SPITXBUF           .set 7048h     ; SPI Serial transmit buffer register
SPIDAT             .set 7049h     ; SPI Serial data register
SPIPRI             .set 704Fh     ; SPI Priority control register

; SCI registers

SCICCR             .set 7050h     ; SCI Communication control register
SCICTL1            .set 7051h     ; SCI Control register 1
SCIHBAUD           .set 7052h     ; SCI Baud Rate MS byte register
SCILBAUD           .set 7053h     ; SCI Baud Rate LS byte register
SCICTL2            .set 7054h     ; SCI Control register 2
SCIRXST            .set 7055h     ; SCI Receiver Status register
SCIRXEMU           .set 7056h     ; SCI Emulation Data Buffer register
SCIRXBUF           .set 7057h     ; SCI Receiver Data buffer register
SCITXBUF           .set 7059h     ; SCI Transmit Data buffer register
SCIPRI             .set 705Fh     ; SCI Priority control register

; Event Manager A  (EVA) registers

GPTCONA            .set 7400h     ; GP Timer control register A
T1CNT              .set 7401h     ; GP Timer 1 counter register
T1CMPR             .set 7402h     ; GP Timer 1 compare register
T1PR               .set 7403h     ; GP Timer 1 period register
T1CON              .set 7404h     ; GP Timer 1 control register
T2CNT              .set 7405h     ; GP Timer 2 counter register
T2CMPR             .set 7406h     ; GP Timer 2 compare register
```

```
T2PR                 .set 7407h     ; GP Timer 2 period register
T2CON                .set 7408h     ; GP Timer 2 control register

COMCONA              .set 7411h     ; Compare control register A
ACTRA                .set 7413h     ; Full compare Action control register A
DBTCONA              .set 7415h     ; Dead–band timer control register A

CMPR1                .set 7417h     ; Full compare unit compare register1
CMPR2                .set 7418h     ; Full compare unit compare register2
CMPR3                .set 7419h     ; Full compare unit compare register3

CAPCONA              .set 7420h     ; Capture control register A
CAPFIFOA             .set 7422h     ; Capture FIFO status register A

CAP1FIFO             .set 7423h     ; Capture Channel 1 FIFO Top
CAP2FIFO             .set 7424h     ; Capture Channel 2 FIFO Top
CAP3FIFO             .set 7425h     ; Capture Channel 3 FIFO Top

CAP1FBOT             .set 7427h     ; Bottom reg. of capture FIFO stack 1
CAP2FBOT             .set 7428h     ; Bottom reg. of capture FIFO stack 2
CAP3FBOT             .set 7429h     ; Bottom reg. of capture FIFO stack 3

EVAIMRA              .set 742Ch     ; Group A Interrupt Mask Register
EVAIMRB              .set 742Dh     ; Group B Interrupt Mask Register
EVAIMRC              .set 742Eh     ; Group C Interrupt Mask Register

EVAIFRA              .set 742Fh     ; Group A Interrupt Flag Register
EVAIFRB              .set 7430h     ; Group B Interrupt Flag Register
EVAIFRC              .set 7431h     ; Group C Interrupt Flag Register

; Event Manager B  (EVB) registers

GPTCONB              .set 7500h     ; GP Timer control register B
T3CNT                .set 7501h     ; GP Timer 3 counter register
T3CMPR               .set 7502h     ; GP Timer 3 compare register
T3PR                 .set 7503h     ; GP Timer 3 period register
T3CON                .set 7504h     ; GP Timer 3 control register
T4CNT                .set 7505h     ; GP Timer 4 counter register
T4CMPR               .set 7506h     ; GP Timer 4 compare register
T4PR                 .set 7507h     ; GP Timer 4 period register
T4CON                .set 7508h     ; GP Timer 4 control register

COMCONB              .set 7511h     ; Compare control register B
ACTRB                .set 7513h     ; Full compare Action control register B
DBTCONB              .set 7515h     ; Dead–band timer control register B

CMPR4                .set 7517h     ; Full compare unit compare register4
CMPR5                .set 7518h     ; Full compare unit compare register5
CMPR6                .set 7519h     ; Full compare unit compare register6

CAPCONB              .set 7520h     ; Capture control register B
CAPFIFOB             .set 7522h     ; Capture FIFO status register B

CAP4FIFO             .set 7523h     ; Capture Channel 4 FIFO Top
CAP5FIFO             .set 7524h     ; Capture Channel 5 FIFO Top
```

```
CAP6FIFO           .set 7525h     ; Capture Channel 6 FIFO Top

CAP4FBOT           .set 7527h     ; Bottom reg. of capture FIFO stack 4
CAP5FBOT           .set 7527h     ; Bottom reg. of capture FIFO stack 5
CAP6FBOT           .set 7527h     ; Bottom reg. of capture FIFO stack 6

EVBIMRA            .set 752Ch     ; Group A Interrupt Mask Register
EVBIMRB            .set 752Dh     ; Group B Interrupt Mask Register
EVBIMRC            .set 752Eh     ; Group C Interrupt Mask Register

EVBIFRA            .set 752Fh     ; Group A Interrupt Flag Register
EVBIFRB            .set 7530h     ; Group B Interrupt Flag Register
EVBIFRC            .set 7531h     ; Group C Interrupt Flag Register

; CAN registers

CANMDER            .set 7100h     ; CAN Mailbox Direction/Enable register
CANTCR             .set 7101h     ; CAN Transmission Control register
CANRCR             .set 7102h     ; CAN Recieve Control register
CANMCR             .set 7103h     ; CAN Master Control register
CANBCR2            .set 7104h     ; CAN Bit Config register 2
CANBCR1            .set 7105h     ; CAN Bit Config register 1
CANESR             .set 7106h     ; CAN Error Status register
CANGSR             .set 7107h     ; CAN Global Status register
CANCEC             .set 7108h     ; CAN Trans and Rcv Err counters
CANIFR             .set 7109h     ; CAN Interrupt Flag Register
CANIMR             .set 710ah     ; CAN Interrupt Mask Register
CANLAM0H           .set 710bh     ; CAN Local Acceptance Mask MBX0/1
CANLAM0L           .set 710ch     ; CAN Local Acceptance Mask MBX0/1
CANLAM1H           .set 710dh     ; CAN Local Acceptance Mask MBX2/3
CANLAM1L           .set 710eh     ; CAN Local Acceptance Mask MBX2/3

CANMSGID0L         .set 7200h     ; CAN Message ID for mailbox 0 (lower 16 bits)
CANMSGID0H         .set 7201h     ; CAN Message ID for mailbox 0 (upper 16 bits)
CANMSGCTRL0        .set 7202h     ; CAN RTR and DLC
CANMBX0A           .set 7204h     ; CAN 2 of 8 bytes of Mailbox 0
CANMBX0B           .set 7205h     ; CAN 2 of 8 bytes of Mailbox 0
CANMBX0C           .set 7206h     ; CAN 2 of 8 bytes of Mailbox 0
CANMBX0D           .set 7207h     ; CAN 2 of 8 bytes of Mailbox 0

CANMSGID1L         .set 7208h     ; CAN Message ID for mailbox 1 (lower 16 bits)
CANMSGID1H         .set 7209h     ; CAN Message ID for mailbox 1 (upper 16 bits)
CANMSGCTRL1        .set 720Ah     ; CAN RTR and DLC
CANMBX1A           .set 720Ch     ; CAN 2 of 8 bytes of Mailbox 1
CANMBX1B           .set 720Dh     ; CAN 2 of 8 bytes of Mailbox 1
CANMBX1C           .set 720Eh     ; CAN 2 of 8 bytes of Mailbox 1
CANMBX1D           .set 720Fh     ; CAN 2 of 8 bytes of Mailbox 1

CANMSGID2L         .set 7210h     ; CAN Message ID for mailbox 2 (lower 16 bits)
CANMSGID2H         .set 7211h     ; CAN Message ID for mailbox 2 (upper 16 bits)
CANMSGCTRL2        .set 7212h     ; CAN RTR and DLC
CANMBX2A           .set 7214h     ; CAN 2 of 8 bytes of Mailbox 2
CANMBX2B           .set 7215h     ; CAN 2 of 8 bytes of Mailbox 2
CANMBX2C           .set 7216h     ; CAN 2 of 8 bytes of Mailbox 2
CANMBX2D           .set 7217h     ; CAN 2 of 8 bytes of Mailbox 2
```

```
CANMSGID3L         .set 7218h     ; CAN Message ID for mailbox 3 (lower 16 bits)
CANMSGID3H         .set 7219h     ; CAN Message ID for mailbox 3 (upper 16 bits)
CANMSGCTRL3        .set 721Ah     ; CAN RTR and DLC
CANMBX3A           .set 721Ch     ; CAN 2 of 8 bytes of Mailbox 3
CANMBX3B           .set 721Dh     ; CAN 2 of 8 bytes of Mailbox 3
CANMBX3C           .set 721Eh     ; CAN 2 of 8 bytes of Mailbox 3
CANMBX3D           .set 721Fh     ; CAN 2 of 8 bytes of Mailbox 3

CANMSGID4L         .set 7220h     ; CAN Message ID for mailbox 4 (lower 16 bits)
CANMSGID4H         .set 7221h     ; CAN Message ID for mailbox 4 (upper 16 bits)
CANMSGCTRL4        .set 7222h     ; CAN RTR and DLC
CANMBX4A           .set 7224h     ; CAN 2 of 8 bytes of Mailbox 4
CANMBX4B           .set 7225h     ; CAN 2 of 8 bytes of Mailbox 4
CANMBX4C           .set 7226h     ; CAN 2 of 8 bytes of Mailbox 4
CANMBX4D           .set 7227h     ; CAN 2 of 8 bytes of Mailbox 4

CANMSGID5L         .set 7228h     ; CAN Message ID for mailbox 5 (lower 16 bits)
CANMSGID5H         .set 7229h     ; CAN Message ID for mailbox 5 (upper 16 bits)
CANMSGCTRL5        .set 722Ah     ; CAN RTR and DLC
CANMBX5A           .set 722Ch     ; CAN 2 of 8 bytes of Mailbox 5
CANMBX5B           .set 722Dh     ; CAN 2 of 8 bytes of Mailbox 5
CANMBX5C           .set 722Eh     ; CAN 2 of 8 bytes of Mailbox 5
CANMBX5D           .set 722Fh     ; CAN 2 of 8 bytes of Mailbox 5

;-------------------------------------------------------------
; I/O space mapped registers
;-------------------------------------------------------------
WSGR               .set 0FFFFh    ; Wait–State Generator Control register
FCMR               .set 0FF0Fh    ; Flash control mode register

;--------------------------------------------------------------
; Bit codes for Test bit instruction (BIT) (15 Loads bit 0 into TC)
;-------------------------------------------------------------
BIT15              .set 0000h     ; Bit Code for 15
BIT14              .set 0001h     ; Bit Code for 14
BIT13              .set 0002h     ; Bit Code for 13
BIT12              .set 0003h     ; Bit Code for 12
BIT11              .set 0004h     ; Bit Code for 11
BIT10              .set 0005h     ; Bit Code for 10
BIT9               .set 0006h     ; Bit Code for 9
BIT8               .set 0007h     ; Bit Code for 8
BIT7               .set 0008h     ; Bit Code for 7
BIT6               .set 0009h     ; Bit Code for 6
BIT5               .set 000Ah     ; Bit Code for 5
BIT4               .set 000Bh     ; Bit Code for 4
BIT3               .set 000Ch     ; Bit Code for 3
BIT2               .set 000Dh     ; Bit Code for 2
BIT1               .set 000Eh     ; Bit Code for 1
BIT0               .set 000Fh     ; Bit Code for 0
```

```
;************************************************************
; File name:          vector.h
; Interrupt Vector declarations
; This section contains the vectors for various interrupts in
; the ’240x. Unused interrupts are shown to branch
; to a ”phantom” interrupt service routine which disables the
; watchdog and loops on itself. Users should replace the label
; PHANTOM with the label of their interrupt subroutines in case
; these interrupts are used.
;************************************************************

            .sect  ”vectors”

RSVECT      B    START            ; Reset Vector
INT1        B    GISR1            ; Interrupt Level 1
INT2        B    GISR2            ; Interrupt Level 2
INT3        B    GISR3            ; Interrupt Level 3
INT4        B    GISR4            ; Interrupt Level 4
INT5        B    GISR5            ; Interrupt Level 5
INT6        B    GISR6            ; Interrupt Level 6
RESERVED    B    PHANTOM          ; Reserved
SW_INT8     B    PHANTOM          ; Software Interrupt
SW_INT9     B    PHANTOM          ; Software Interrupt
SW_INT10    B    PHANTOM          ; Software Interrupt
SW_INT11    B    PHANTOM          ; Software Interrupt
SW_INT12    B    PHANTOM          ; Software Interrupt
SW_INT13    B    PHANTOM          ; Software Interrupt
SW_INT14    B    PHANTOM          ; Software Interrupt
SW_INT15    B    PHANTOM          ; Software Interrupt
SW_INT16    B    PHANTOM          ; Software Interrupt
TRAP        B    PHANTOM          ; Trap vector
NMI         B    NMI              ; Non–maskable Interrupt
EMU_TRAP    B    PHANTOM          ; Emulator Trap
SW_INT20    B    PHANTOM          ; Software Interrupt
SW_INT21    B    PHANTOM          ; Software Interrupt
SW_INT22    B    PHANTOM          ; Software Interrupt
SW_INT23    B    PHANTOM          ; Software Interrupt
SW_INT24    B    PHANTOM          ; Software Interrupt
SW_INT25    B    PHANTOM          ; Software Interrupt
SW_INT26    B    PHANTOM          ; Software Interrupt
SW_INT27    B    PHANTOM          ; Software Interrupt
SW_INT28    B    PHANTOM          ; Software Interrupt
SW_INT29    B    PHANTOM          ; Software Interrupt
SW_INT30    B    PHANTOM          ; Software Interrupt
SW_INT31    B    PHANTOM          ; Software Interrupt
```

```
;=================================================================================
* File Name:  SPI.asm
* Description: PROGRAM TO OUTPUT SERIAL DATA THROUGH THE SPI PORT
* This program outputs a set of incrementing words (that roll over) through
* the SPI. If a Digital–to–analog (DAC) converter is connected to the SPI,
* the DAC outputs a sawtooth waveform. The program sends data to the serial DAC
* by means of the SPI. For this example, the TLC5618 serial DAC from TI was used.
;=================================================================================

       .include  240x.h
       .include  vector.h
;-----------------------------------------------------------------------------
; Variable Declarations for on chip RAM Blocks
;-----------------------------------------------------------------------------
       .bss      GPR0,1    ;General purpose registers.
       .bss      GPR3,1

;-----------------------------------------------------------------------------
; M A C R O – Definitions
;-----------------------------------------------------------------------------

KICK_DOG  .macro                    ;Watchdog reset macro
          LDP     #00E0h
          SPLK    #05555h, WDKEY
          SPLK    #0AAAAh, WDKEY
          LDP     #0h
          .endm

;===========================================================================
; M A I N   C O D E  – starts here
;===========================================================================
          .text
START:    LDP     #0
          SETC    INTM              ;Disable interrupts during initialization.
          SPLK    #0h,GPR3
          OUT     GPR3,WSGR         ;Set XMIF to run with no wait states.

          CLRC    SXM               ;Clear Sign Extension Mode
          CLRC    OVM               ;Reset Overflow Mode
          CLRC    CNF               ;Config Block B0 to Data mem.

          LDP     #WDCR>>7
          SPLK    #006Fh,WDCR       ;Disable WD

          KICK_DOG

          LDP     #SCSR1>>7         ;Set PLL for x4 mode
          SPLK    #0020h,SCSR1      ;Enable clock to SPI module
```

```
;===========================================================================
; SPI Initialization
;===========================================================================

SPI_INIT:    LDP     #SPICCR>>7
             SPLK    #000Fh, SPICCR       ;16 char bits,
             SPLK    #0006h, SPICTL       ;Enable master mode, normal clock
                                          ;and enable talk.
             SPLK    #0002h, SPIBRR       ;Set up the SPI to max speed.

             LDP     #MCRB>>7             ;Set up the GPIO pins to function
             SPLK    #003CH, MCRB         ;as SPI pins

             LDP     #SPICCR>>7
             SPLK    #008Fh, SPICCR       ;Relinquish SPI from Reset.

;===========================================================================
; This section generates the sawtooth by ramping a counter down to zero
; reloading it every time it under–flows.
;===========================================================================;

LP:          LAR     AR0,#07FEh     ;Load AR0 with a count

XMIT_VALUE:  LDP     #0
             SAR     AR0,GPR0
             LACC    GPR0
             ADD     #8000H         ;MSB should be one (DAC requirement)
             XOR     #07FFH         ;To change the direction of counting to
                                    ;upward

             LDP     #SPITXBUF>>7
             SACL    SPITXBUF       ;Write xmit value to SPI Trasmit Buffer.

             LDP     #SPISTS>>7
XMIT_RDY:    BIT     SPISTS,BIT6    ;Test SPI_INT bit
             BCND    XMIT_RDY, NTC  ;If SPI_INT=0,then repeat loop
                                    ;i.e. wait for the completion of
                                    ;transmission.
             LDP     #SPIRXBUF>>7   ;else read SPIRXBUF
             LACC    SPIRXBUF       ;dummy read to clear SPI_INT flag.

             KICK_DOG

             MAR     *,AR0
             BANZ    XMIT_VALUE     ;xmit next value, if counter is non zero.
             B       LP             ;if counter reaches zero repeat loop
                                    ;re–loading the counter.

PHANTOM      RET
GISR1        RET
GISR2        RET
GISR3        RET
GISR4        RET
GISR5        RET
GISR6        RET
```

```
;=============================================================================
* File Name: SCI.asm
* Description: PROGRAM TO PERFORM A LOOPBACK IN THE SCI MODULE IN ’240x
* An 8 bit value is transmitted through the SCITXD pin at a baud rate of
* 9600 bits/sec. SCITXD–SCIRXD pins are connected together, if external
* loopback is desired i.e. if it is desired to echo the bit–stream back. The SCI
* receives the bit–stream and stores the received data in memory for verification.
* This program is capable of doing internal loopback AND external loopback,
* depending on the value written in SCICCR.
;=============================================================================
          .include        240x.h

KICK_DOG  .macro                 ;Watchdog reset macro
          LDP      #00E0h
          SPLK     #05555h, WDKEY
          SPLK     #0AAAAh, WDKEY
          LDP      #0h
          .endm
;=============================================================================
; M A I N   C O D E  – starts here
;=============================================================================
          .text
START:
          LDP     #0
          SETC    INTM           ;Disable interrupts
          CLRC    SXM            ;Clear Sign Extension Mode
          CLRC    OVM            ;Reset Overflow Mode
          SETC    CNF            ;Config Block B0 to Data mem.

          LDP     #00E0h
          SPLK    #006Fh,WDCR    ;Disable WD

          KICK_DOG
          SPLK    #0h,GPR0       ;Set wait state generator for:
          OUT     GPR0,WSGR      ;Program Space, 0–7 wait states
          LDP     #00E0h
          SPLK    #0040h,SCSR1   ;Enable clock for SCI module
```

```
;===========================================================================
;SCI TRANSMISSION TEST – starts here
;===========================================================================

SCI:          LDP     #0E1h
              SPLK    #0FFFFh,MCRA

              LAR     AR0, #SCITXBUF         ;Load AR0 with SCI_TX_BUF address
              LAR     AR1, #SCIRXBUF         ;Load AR1 with SCI_RX_BUF address
              LAR     AR2, #20h              ;AR2 is the counter
              LAR     AR3, #60h              ;AR3 is the pointer
              LDP     #SCICCR>>7
              SPLK    #17h, SCICCR           ;17 for internal
                                             ;loopback 07–External
                                             ;1 stop bit,odd parity,8 char bits,
                                             ;async mode, idle–line protocol
              SPLK    #0003h, SCICTL1        ;Enable TX, RX, internal SCICLK,
                                             ;Disable RX ERR, SLEEP, TXWAKE
              SPLK    #0000h, SCICTL2        ;Disable RX & TX INTs

              SPLK    #0000h, SCIHBAUD
              SPLK    #0186h, SCILBAUD       ;Baud Rate=9600 b/s (30 MHz SYSCLK)

              LDP     #SCICCR>>7
              SPLK    #0023h, SCICTL1        ;Relinquish SCI from Reset.

XMIT_CHAR:    LACL    #55h                   ;Load ACC with xmit character
              MAR *,AR0
              SACL    *,AR1                  ;Write xmit char to TX buffer

              LDP     #SCICTL2>>7
XMIT_RDY:     BIT     SCICTL2,BIT7           ;Test TXRDY bit
              BCND    XMIT_RDY,NTC           ;If TXRDY=0,then repeat loop

RCV_RDY:      BIT     SCIRXST,BIT6           ;Test TXRDY bit
              BCND    RCV_RDY,NTC            ;If TXRDY=0,then repeat loop

READ_CHR:     LACL    *,AR3                  ;The received (echoed) character is
                                             ;stored in 60h
              SACL    *+,AR2                 ;This loop is executed 20h times
              BANZ    XMIT_CHAR              ;Repeat the loop again

LOOP          B   LOOP                       ;Program idles here after executing
                                             ;transmit loops
```

```
;============================================================================*
* File name :   ADC.asm                                                     *
* Description : PROGRAM TO INITIALIZE THE ADC MODULE OF 240x                *
* This program initializes the ADC module of the ’240x and does a conversion *
* of all the analog input channels. The results of the conversion are       *
* available in the RESULTSn register, which can be accessed by the user     *
* application. The ADC operates as one 16–state sequencer & the conversions *
* are stopped once the sequencer reaches EOS (End of sequence)              *
;============================================================================*

                .title   ”ADC”

                .bss     GPR0,1             ; General purpose register
                .include 240x.h
                .copy    ”vector.h”

;-------------------------------------------------------------
; M A C R O – Definitions
;-------------------------------------------------------------
KICK_DOG        .macro                      ; Watchdog reset macro
                LDP      #00E0h             ; DP––>7000h–707Fh
                SPLK     #05555h, WDKEY
                SPLK     #0AAAAh, WDKEY
                LDP      #0h                ; DP––>0000h–007Fh
                .endm

                .text
START:          LDP      #0h                ; Set DP=0
                SETC     INTM               ; Disable interrupts
                CLRC     SXM
                SPLK     #0000h,IMR         ; Mask all core interrupts
                LACC     IFR                ; Read Interrupt flags
                SACL     IFR                ; Clear all interrupt flags
                LDP      #00E0h             ; (E0=224)(E0*80=7000)
                SPLK     #006Fh, WDCR       ; Disable WD
                SPLK     #0080h,SCSR1       ; Enable clock to ADC module
                KICK_DOG
                SPLK     #0h,GPR0           ; Set wait state generator for:
                OUT      GPR0,WSGR          ; Program Space, 0–7 wait states

* Initialize ADC registers

                LDP      #0E1h
                SPLK     #0100000000000000b,ADCTRL1 ; Reset ADC module
                SPLK     #0011000000010000b,ADCTRL1 ; Take ADC out of reset
                        ; ||||||||||||||||
                        ; 5432109876543210
                        ; 15 – RSVD | 14 – Reset(1) | 13,12 – Soft & Free
                        ; 11,10,9,8 – Acq.prescalers | 7 – Clock prescaler
                        ; 6 – Cont.run (1) | 5 – Int.priority (Hi.0)
                        ; 4 – Seq.casc (0–dual)
```

```
* Setup a maximum of 16 conversions

                SPLK     #15, MAXCONV       ; Setup for 16 conversions

* Program the conversion sequence. This is the sequence of channels that will
* be used for the 16 conversions.

                SPLK     #03210h, CHSELSEQ1 ; Convert Channels 0,1,2,3
                SPLK     #07654h, CHSELSEQ2 ; Convert Channels 4,5,6,7
                SPLK     #0BA98h, CHSELSEQ3 ; Convert Channels 8,9,10,11
                SPLK     #0FEDCh, CHSELSEQ4 ; Convert Channels 12,13,14,15

                SPLK     #0010000000000000b,ADCTRL2 ; Start the conversions
                        ; ||||||||||||||||
                        ; 5432109876543210

CHK_EOS1:       BIT      ADCTRL2, BIT12     ; Wait for SEQ1 Busy bit to
                                            ; clear
                BCND     CHK_EOS1, TC       ; If TC=1, keep looping.

                RPT      #8
                NOP

LOOP:           B        LOOP               ; The conversion results are now
                                            ; available in the RESULTSn regs.
GISR1:          RET
GISR2:          RET
GISR3:          RET
GISR4:          RET
GISR5:          RET
GISR6:          RET
PHANTOM:        RET
                .end
```

```
;============================================================================
* File name :   GPIO_OUT.asm
* Description : PROGRAM TO CHECK THE GPIO PINS OF 240x as outputs        *
* This program writes a running pattern of 0's  to the GPIO pins of 240x *
* It ouputs a total of 8 bit patterns to the five GPIO ports (A,B,C,E,F) *
* Each bit pattern forces a particular bit low and forces the other 7    *
* bits high. This goes on in an endless loop.                            *
;============================================================================


                .title  " 240x GPIO"

                .data                   ; Loaded @ 300h in data memory
b0              .word   0FFFEh          ; Turn-on GPIO0
b1              .word   0FFFDh          ; Turn-on GPIO1
b2              .word   0FFFBh          ; Turn-on GPIO2
b3              .word   0FFF7h          ; Turn-on GPIO3
b4              .word   0FFEFh          ; Turn-on GPIO4
b5              .word   0FFDFh          ; Turn-on GPIO5
b6              .word   0FFBFh          ; Turn-on GPIO6
b7              .word   0FF7Fh          ; Turn-on GPIO7
GPR0            .word   0               ; Gen purp reg

                .include 240x.h

;-------------------------------------------------------------
; M A C R O - Definitions
;-------------------------------------------------------------
KICK_DOG        .macro                  ;Watchdog reset macro
                LDP     #00E0h          ;DP-->7000h-707Fh
                SPLK    #05555h, WDKEY
                SPLK    #0AAAAh, WDKEY
                LDP     #0h             ;DP-->0000h-007Fh
                .endm

                .text
START:          LDP     #0h             ; Set DP=0
                SETC    INTM            ; Disable interrupts
                SETC    CNF
                SPLK    #0000h,IMR      ; Mask all core interrupts
                LACC    IFR             ; Read Interrupt flags
                SACL    IFR             ; Clear all interrupt flags
                LDP     #00E0h          ; (E0=224)(E0*80=7000)
                SPLK    #0000h,SCSR1
                SPLK    #006Fh, WDCR    ; Disable WD
                KICK_DOG
                SPLK    #0h,GPR0        ; Set wait state generator for
                OUT     GPR0,WSGR       ; external address space

                LDP     #00E1h
                SPLK    #00000h,MCRA    ; Select IOPAn & IOPBn as GPIO pins
                SPLK    #0FF00h,MCRB    ; Select IOPCn as GPIO pins
                SPLK    #00000h,MCRC    ; Select IOPEn & IOPFn as GPIO pins

                SPLK    #0FFFFh, PADATDIR  ; All pins are o/p's
                SPLK    #0FFFFh, PBDATDIR  ; and forced high
                SPLK    #0FFFFh, PCDATDIR  ;
                SPLK    #0FFFFh, PEDATDIR  ;
                SPLK    #0FFFFh, PFDATDIR  ;


MAIN            LDP     #0
                LAR     AR0,#300h          ; AR0 points to bit pattern in
                                           ; data memory
                LAR     AR1,#7             ; AR1 is the counter

LOOP            MAR     *,AR0
                LACC    *+,AR2             ; Load bit pattern in accumulator
                LDP     #00E1h
                SACL    PADATDIR           ; Output the same bit pattern
                SACL    PBDATDIR           ; to all the GPIO ports
                SACL    PCDATDIR
                SACL    PEDATDIR
                SACL    PFDATDIR

                CALL    DELAY              ; Delay provided in between
                                           ; each pattern
                MAR     *,AR1              ; Check if all 8 patterns have
                BANZ    LOOP               ; been output. If not, continue.
                B       MAIN

DELAY           LAR     AR2,#0FFFFh
D_LOOP          RPT     #0FFh
                NOP
                BANZ    D_LOOP
                RET

PHANTOM         KICK_DOG                   ;Resets WD counter
                B       PHANTOM
```

```
;=========================================================================
* File name :   GPIO_IN.asm                                              *
* Description : PROGRAM TO CHECK GPIO PINS OF 240x as inputs             *
* All GPIO bits are programmed as inputs and the values read from the    *
* GPIO pins are written in 60h,61h,62h,63h,64h of Data memory            *
;=========================================================================

                .title   ” 240x GPIO”

                .bss     GPR0,1          ; Gen purp reg

                .include 240x.h

;–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
; M A C R O – Definitions
;–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
KICK_DOG        .macro                   ; Watchdog reset macro
                LDP      #00E0h          ; DP––>7000h–707Fh
                SPLK     #05555h, WDKEY
                SPLK     #0AAAAh, WDKEY
                LDP      #0h             ; DP––>0000h–007Fh
                .endm

                .text
START:          LDP      #0h             ; Set DP=0
                SETC     INTM            ; Disable interrupts
                SETC     CNF
                SPLK     #0000h,IMR      ; Mask all core interrupts
                LACC     IFR             ; Read Interrupt flags
                SACL     IFR             ; Clear all interrupt flags
                LDP      #00E0h          ; (E0=224)(E0*80=7000)
                SPLK     #006Fh, WDCR    ; Disable WD
                SPLK     #0,SCSR1        ; Put PLL in x4 mode.

                KICK_DOG
                SPLK     #0h,GPR0        ; Set wait state generator for
                OUT      GPR0,WSGR       ; external address space.
                LDP      #00E1h
                SPLK     #00000h,MCRA    ; Select IOPAn & IOPBn as GPIO pins
                SPLK     #0FF00h,MCRB    ; Select IOPCn as GPIO pins
                SPLK     #00000h,MCRC    ; Select IOPEn & IOPFn as GPIO pins

                SPLK     #0h, PADATDIR   ; All GPIO pins are programmed
                SPLK     #0h, PBDATDIR   ; as inputs
                SPLK     #0h, PCDATDIR
                SPLK     #0h, PEDATDIR
                SPLK     #0h, PFDATDIR

MAIN            LDP      #0              ; This loop reads the level on
                LAR      AR0,#60h        ; the GPIO pins. The bit patterns
                MAR      *,AR0           ; read from the 5 GPIO ports
                LDP      #00E1h          ; is copied in the data memory

                LACL     PADATDIR
                SACL     *+
                LACL     PBDATDIR
                SACL     *+
                LACL     PCDATDIR
                SACL     *+
                LACL     PEDATDIR
                SACL     *+
                LACL     PFDATDIR
                SACL     *+
                B        MAIN

PHANTOM         KICK_DOG                ;Resets WD counter
                B        PHANTOM
```

```
;================================================================================
* File name :   REM_ANS.asm                                                      *
* Description : PROGRAM TO INITIATE AUTO–ANSWER TO A REMOTE FRAME                *
*               REQUEST IN CAN                                                   *
* The two CAN modules must be connected to each other with appropriate           *
* termination resistors. Reception and transmission by MBX2. Low priority        *
* interrupt used. Transmit acknowledge for MBX2 is set after running this        *
* program and the message is transmitted.To be used along with REM_REQ.asm       *
* PERIPHERAL CODE : A, TEST CODE : 0 After successful completion of              *
* this program, the value A000 must be present in 3A0h (DM)                      *
* Error code:A001 –– Remote request not received from the remote node            *
;================================================================================

                .title   ”REM_ANS”        ; Title
                .include 240x.h           ; Variable and register declaration
                .include vector.h         ; Vector label declaration

;--------------------------------------------------------------------------------
; Constant definitions
;--------------------------------------------------------------------------------

DP_PF1          .set     0E0h             ; Page 1 of peripheral file (7000h/80h
DP_CAN          .set     0E2h             ; CAN Register (7100h)
DP_CAN2         .set     0E4h             ; CAN RAM (7200h)
;--------------------------------------------------------------------------------
; M A C R O – Definitions
;--------------------------------------------------------------------------------

KICK_DOG        .macro                    ; Watchdog reset macro
                LDP      #00E0h
                SPLK     #05555h, WDKEY
                SPLK     #0AAAAh, WDKEY
                LDP      #0h
                .endm

;================================================================================
; M A I N   C O D E  – starts here
;================================================================================
                .text

START:          SETC     INTM             ; Disable interrupts
                LDP      #DP_PF1
                SPLK     #0010h,SCSR1     ; Enable clock to CAN module
                SPLK     06Fh,WDCR        ; Disable Watchdog
                KICK_DOG

                LDP      #7h              ; Write error code to start with
                SPLK     #0A001h,020h     ; at 3A0h in B1 memory
                LDP      #225
                SPLK     #00C0H,MCRB      ; Configure CAN pins

                LAR      AR0,#300h        ; AR0 => Copy CAN RAM (B0)
                LAR      AR1,#3h          ; AR1 => counter
                LAR      AR2,#7214h       ; AR2 => MBX2
```

```
;**************************************************************************
; Enable 1 core interrupt
;**************************************************************************

                LDPK     #0
                SPLK     #0000000000010000b, IMR ; core interrupt mask register
;                         ||||||||||||||||
;                         FEDCBA9876543210

                SPLK     #000ffh,IFR     ; Clear all core interrupt flags
                CLRC     INTM            ; enable interrupt

;**************************************************************************
;******************** CAN Initialization************************************
;**************************************************************************

                LDP      #DP_CAN

                SPLK     #1001111111111110b,CANLAM1H   ; Set LAM
                SPLK     #1111111111111111b,CANLAM1L   ; 1:don’t care

                SPLK     #1011111111111111b,CANIMR     ; Enable all interrupts

; bit 0                  Warning level
; bit 1                  Error passive
; bit 2                  Bus off
; bit 3                  Wake up
; bit 4                  Write denied
; bit 5                  Abort acknowledge
; bit 6                  Receive message lost interrupt
; bit 7                  Error interrupt priority level  1=low
; bit 8–D                Mailbox interrupt mask
; bit E                  Reserved
; bit F                  Mailbox interrupt priority level. 1=low

;**************************************************************************
;***********           Configure CAN before writing              **********
;**************************************************************************

                LDP      #DP_CAN

                SPLK     #0000000000000000b,CANMDER
;                         ||||||||||||||||
;                         FEDCBA9876543210
; bit 0–5                disable each mailbox


                SPLK     #0000000100000000b,CANMCR
;                         ||||||||||||||||
;                         FEDCBA9876543210

; bit 8                  CDR: Change data field request

;**************************************************************************
;***********                Write CAN Mailboxes                  **********
```

```
;***************************************************************************

                LDP     #DP_CAN2

                SPLK    #1111111111111111b,CANMSGID2H
;                        ||||||||||||||||
;                        FEDCBA9876543210

; bit 0–12              upper 13 bits of extended identifier
; bit 13                Auto answer mode bit
; bit 14                Acceptance mask enable bit
; bit 15                Identifier extension bit

                SPLK    #1111111111111111b,CANMSGID2L
;                        ||||||||||||||||
;                        FEDCBA9876543210

; bit 0–15              lower part of extended identifier

                SPLK    #0000000000001000b,CANMSGCTRL2
;                        ||||||||||||||||
;                        FEDCBA9876543210

; bit 0–3               Data length code: 1000 = 8 bytes
; bit 4                 0: data frame

                SPLK    #0BEBEh,CANMBX2A   ; Message to be transmitted
                SPLK    #0BABAh,CANMBX2B   ; to the remote node
                SPLK    #0DEDEh,CANMBX2C
                SPLK    #0DADAh,CANMBX2D

;***************************************************************************
;***********    Set parameters after writing                     **********
;***************************************************************************

                LDP     #DP_CAN
                SPLK    #0000000000000000b,CANMCR
;                        ||||||||||||||||
;                        FEDCBA9876543210
; bit 8                 CDR: Change data field request

                SPLK    #0000000000000100b,CANMDER
;                        ||||||||||||||||
;                        FEDCBA9876543210

; bit 0–5               Enable  MBX2
; bit 6                 MBX2 configured as Transmit MBX

;***************************************************************************
;***********         CAN Registers configuration    **********************
;***************************************************************************
                SPLK    #0001000000000000b,CANMCR
;                        ||||||||||||||||
;                        FEDCBA9876543210
; bit 12                  Change configuration request

W_CCE           BIT       CANGSR,#0Bh     ; Wait for Change config Enable
                BCND      W_CCE,NTC

                SPLK      #0000000000000000b,CANBCR2
;                          ||||||||||||||||
;                          FEDCBA9876543210

; bit 0–7                 Baud rate prescaler
; bit 8–15                Reserved

                SPLK      #0000010101010111b,CANBCR1
;                          ||||||||||||||||
;                          FEDCBA9876543210

; bit 0–2                 TSEG1
; bit 3–6                 TSEG2
; bit 7                   Sample point setting (1: 3 times, 0: once)
; bit 8–A                 Synchronization jump width
; bit B                   Synchronization on falling edge
; bit C–F                 Reserved

                SPLK      #0000000000000000b,CANMCR
;                          ||||||||||||||||
;                          FEDCBA9876543210

; bit 12                  Change configuration request
;

W_NCCE          BIT       CANGSR,#0Bh     ; Wait for Change config disable
                BCND      W_NCCE,TC

W_ERROR         LACL      CANESR          ; Check errors
                BCND      W_ERROR,NEQ

LOOP            B         LOOP            ; Wait for Receive Interrupt

;===================================================================
; ISR used to copy MBX2 RAM when an interrupt is received
;===================================================================

GISR5:

LOOP_READ       MAR       *,AR2
                LACL      *+,AR0          ; Copy MBX2 contents in Accumulator
                SACL      *+,AR1          ; Copy MBX2 contents in B0
                BANZ      LOOP_READ       ; Copy all 4 words

                LDP       #7h             ; Write A000 at 3A0h in B1 memory
                SPLK      #0A000h,020h    ; if this ISR is executed once.

                CLRC      INTM
                RET
```

```
GISR1:                   RET
GISR2:                   RET
GISR3:                   RET
GISR4:                   RET
GISR6:                   RET

PHANTOM         RET
                .end

; When data in MBX2 is transmitted in response to a ”Remote frame request,”
; the MBX2 data is copied from 300h onwards in DM. Note that TRS bit is not
; set for MBX2. The transmission of MBX2 data is automatic ,in response to
; a ”Remote frame request.”
```

```
;==============================================================================
* File name :   REM_REQ.asm                                                   *
* Description : PROGRAM TO TRANSMIT A REMOTE FRAME REQUEST IN THE CAN OF 240x *
* The two CAN modules must be connected to each other with appropriate        *
* termination resistors. Transmission of a remote frame by MBX3 and reception *
* of the data frame in MBX0. To be used along with REM_ANS.asm                *
* PERIPHERAL CODE : A, TEST CODE : 0 After successful completion of           *
* this program, the value A000 must be present in 3A0h (DM)                   *
* Error code:A001 –– Error in initialization/ communication                   *
;==============================================================================

                .title   ”REM_REQ”       ; Title
                .include 240x.h          ; Variable and register declaration
                .include vector.h        ; Vector label declaration

;-----------------------------------------------------------------------------
; Other constant definitions
;-----------------------------------------------------------------------------

DP_PF1          .set     0E0h            ; Page 1 of peripheral file (7000h/80h
DP_CAN          .set     0E2h            ; Can Registers (7100h)
DP_CAN2         .set     0E4h            ; Can RAM (7200h)
;-----------------------------------------------------------------------------
; M A C R O – Definitions
;-----------------------------------------------------------------------------

KICK_DOG        .macro                   ; Watchdog reset macro
                LDP      #00E0h
                SPLK     #05555h, WDKEY
                SPLK     #0AAAAh, WDKEY
                LDP      #0h
                .endm

;==============================================================================
; M A I N   C O D E  – starts here
;==============================================================================
                .text

START:          SETC     INTM            ; Disable interrupts
                LDP      #DP_PF1
                SPLK     #0010h,SCSR1    ; Enable clock to CAN module
                SPLK     06Fh,WDCR       ; Disable Watchdog
                KICK_DOG

                LDP      #7h             ; Write error code to start with
                SPLK     #0A001h,020h    ; at 3A0h in B1 memory
                LDP      #225
                SPLK     #00C0H,MCRB     ; Configure CAN pins

                LAR      AR0,#7204h      ; AR0 => MBX0
                LAR      AR1,#300h       ; AR1 => B0 RAM
                LAR      AR2,#3h         ; AR2 => Counter
                LAR      AR3,#721ch      ; AR3 => MBX3
```

```
;***************************************************************************
;******************** CAN Initialization***********************************
;***************************************************************************

                LDP      #DP_CAN
                SPLK     #1001111111111111b,CANLAM0H  ; Set LAM0
                SPLK     #1111111111111111b,CANLAM0L  ; 1:don’t care

                SPLK     #1011111111111111b,CANIMR    ; Enable all interrupts

;***************************************************************************
;***********           Configure CAN before writing           **********
;***************************************************************************

                LDP      #DP_CAN

                SPLK     #0000000000000000b,CANMDER
;                         ||||||||||||||||
;                         FEDCBA9876543210

; bit 0–5                disable each mailbox

                SPLK     #0000000100000000b,CANMCR
;                         ||||||||||||||||
;                         FEDCBA9876543210

; bit 8                  CDR: Change data field request

;***************************************************************************
;***********                Write CAN Mailboxes                **********
;***************************************************************************

                LDP      #DP_CAN2

                SPLK     #1111111111111111b,CANMSGID3H
;                         ||||||||||||||||
;                         FEDCBA9876543210

; bit 0–12               upper 13 bits of extended identifier
; bit 13                 Auto answer mode bit
; bit 14                 Acceptance mask enable bit
; bit 15                 Identifier extension bit

                SPLK     #1111111111111111b,CANMSGID3L
;                         ||||||||||||||||
;                         FEDCBA9876543210

; bit 0–15               lower part of extended identifier

                SPLK     #0000000000011000b,CANMSGCTRL3
;                         ||||||||||||||||
;                         FEDCBA9876543210

; bit 0–3                Data length code. 1000 = 8 bytes
; bit 4                  1: Remote frame

                SPLK     #1111111111111111b,CANMSGID0H
;                         ||||||||||||||||
;                         FEDCBA9876543210

; bit 0–12               upper 13 bits of extended identifier
; bit 13                 Auto answer mode bit
; bit 14                 Acceptance mask enable bit
; bit 15                 Identifier extension bit

                SPLK     #1111111111111110b,CANMSGID0L
;                         ||||||||||||||||
;                         FEDCBA9876543210

; bit 0–15               lower part of extended identifier

                SPLK     #0000000000001000b,CANMSGCTRL0
;                         ||||||||||||||||
;                         FEDCBA9876543210

;***************************************************************************
;***********    Set parameters after writing                   **********
;***************************************************************************

                LDP      #DP_CAN
                SPLK     #0000000000000000b,CANMCR
;                         ||||||||||||||||
;                         FEDCBA9876543210
; bit 8                  CDR: Change data field request

                SPLK     #0000000001001001b,CANMDER
;                         ||||||||||||||||
;                         FEDCBA9876543210

; bit 0–5                enable mailbox 3 and mailbox 0
; bit 7                  0: mailbox 3 = transmit

;***************************************************************************
;***********         CAN Registers configuration    ***********************
;***************************************************************************
                SPLK     #0001000000000000b,CANMCR
;                         ||||||||||||||||
;                         FEDCBA9876543210

; bit 12                 Change conf register

W_CCE           BIT       CANGSR,#0Bh    ; Wait for Change config Enable
                BCND      W_CCE,NTC

                SPLK      #0000000000000000b,CANBCR2
;                          ||||||||||||||||
;                          FEDCBA9876543210

; bit 0–7                Baud rate prescaler
; bit 8–15               Reserved

                SPLK     #0000010101010111b,CANBCR1
;                         ||||||||||||||||
;                         FEDCBA9876543210

; bit 0–2                TSEG1
; bit 3–6                TSEG2
; bit 7                  Sample point setting (1: 3 times, 0: once)
; bit 8–A                Synchronization jump width
; bit B                  Synchronization on falling edge
; bit C–F                Reserved


                SPLK     #0000000000000000b,CANMCR
;                         ||||||||||||||||
;                         FEDCBA9876543210

; bit 12                 Change conf register

W_NCCE          BIT      CANGSR,#0Bh     ; Wait for Change config disable
                BCND     W_NCCE,TC

;***************************************************************************
;***********                       TRANSMIT                      **********
;***************************************************************************
                SPLK     #0020h,CANTCR   ; Transmit request for MBX3

W_TA            BIT      CANTCR,2        ; Wait for transmission acknowledge
                BCND     W_TA,NTC        ; for MBX3
                SPLK     #2000h,CANTCR   ; reset TA

RX_LOOP:
W_RA            BIT      CANRCR,BIT4     ; Wait for data from remote node
                BCND     W_RA,NTC        ; to be written into MBX0

LOOP_READ2      MAR      *,AR0           ; Copy MBX0 contents in Accumulator
                LACL     *+,AR1          ; Copy MBX0 contents in B0
                SACL     *+,AR2          ; Copy all 4 words
                BANZ     LOOP_READ2

                LAR      AR1,#300h       ; AR1 => B0 RAM
                MAR      *,AR1

CHECK           LACL     *+              ; Check the received data
                XOR      #0BEBEh         ; The remote node transmits
                BCND     LOOP,NEQ        ; BEBEh, BABAh, DEDEh & DADAh
                LACL     *+              ; The correct reception of those
                XOR      #0BABAh         ; 4 words are checked in this loop.
                BCND     LOOP,NEQ
                LACL     *+
                XOR      #0DEDEh
                BCND     LOOP,NEQ
                LACL     *+
                XOR      #0DADAh
                BCND     LOOP,NEQ

PASS            LDP      #7h              ; Received data is correct
                SPLK     #0A000h,020h     ; Write A000 in 3A0

LOOP            B        LOOP

GISR1:                    RET
GISR2:                    RET
GISR3:                    RET
GISR4:                    RET
GISR5:                    RET
GISR6:                    RET

PHANTOM         RET
                .end
```

```
;=========================================================================
* File name :   EV_T1INT.asm                                            *
* Description : PROGRAM TO CHECK THE OPERATION OF TIMER1 IN EVA          *
* Mode:         Continous Up/Down counting, x/128                        *
* Output:       OF,UF,CMPR & PERIOD  interrupts that toggles IOPB0,1,2,3 *
;=========================================================================

                .title ” EV test routine” ; Title
                .include ”240x.h”         ; Variable and register declaration
                .include vector.h         ; Vector label declaration

;-------------------------------------------------------------
; M A C R O – Definitions
;-------------------------------------------------------------
KICK_DOG        .macro                    ;Watchdog reset macro
                LDP      #00E0h           ;DP––>7000h–707Fh
                SPLK     #05555h, WDKEY
                SPLK     #0AAAAh, WDKEY
                LDP      #0h              ;DP––>0000h–007Fh
                .endm

                .text

START:          LDP      #0h              ; set DP=0
                SETC     INTM             ; Disable interrupts
                SPLK     #0000h,IMR       ; Mask all core interrupts
                LACC     IFR              ; Read Interrupt flags
                SACL     IFR              ; Clear all interrupt flags
                LDP      #WDKEY >> 7h     ; Peripheral page
                SPLK     #0004h,SCSR1     ; EVA module clock enable
                SPLK     #006Fh, WDCR     ; Disable WD
                KICK_DOG
                MAR      *,AR0
                LDP      #0E1h            ; Peripheral page
                SPLK     #1111111100000000b,PBDATDIR
                                          ; set IOPBn as outputs,0

* Load TIMER 1 registers
                LDP      #GPTCONA >> 7h   ; Peripheral page
                SPLK     #0000000000000000b,GPTCONA
                SPLK     #0000000000000000b,T1CNT ; zero timer 1 count
                SPLK     #0000111101000010b,T1CON
                                          ;000 01 Cont, Up/Down
                                          ;111 x/128
                                          ;01 Tenable reserved for T1
                                          ;00 Internal clk
                                          ;00 LD CMPR whencntr =0
                                          ;1 enable compare
                                          ;0 use own period register

                SPLK     #1111111111111111b,T1PR
                SPLK     #0000000011111111b,T1CMPR
                SPLK     #0000011110000000b,EVAIMRA
                                          ; Enable OV,U,C,P interrupt bits
                SPLK     #0000011110000000b,EVAIFRA
                                           ; clear interrupts

                LDP      #0
                SPLK     #0000000000000010b,IMR   ; Enable INT2
                CLRC     INTM

wait:           NOP                        ; main loop
                NOP
                B        wait

GISR2:          NOP                        ; Int2 GISR
                LDP      #PIVR >> 7h       ; Peripheral page
                LACL     PIVR              ; PIVR value
                XOR      #002ah            ; T1 overflow
                BCND     SISR2a,eq
                LACL     PIVR
                XOR      #0029h            ; T1 underflow
                BCND     SISR29,eq
                LACL     PIVR
                XOR      #0028h            ; T1 Compare
                BCND     SISR28,eq
                LACL     PIVR
                XOR      #0027h            ; T1 Period
                BCND     SISR27,eq
                RET

SISR2a:
                LDP      #0E1h             ; Peripheral page
                SPLK     #0FF01h,PBDATDIR ; Set IOPB0
                CALL     DELAY
                LDP      #GPTCONA >> 7h    ; Peripheral page
                LACC     #0400h            ; clear overflow int. flag
                SACL     EVAIFRA           ; in EVAIFRA
                CLRC     INTM              ; Enable all interrupts
                RET

SISR29:
                LDP      #0E1h             ; Peripheral page
                SPLK     #0FF02h,PBDATDIR ; Set IOPB1
                CALL     DELAY
                LDP      #GPTCONA >> 7h    ; Peripheral page
                LACC     #0200h            ; clear underflow int. flag
                SACL     EVAIFRA           ; in EVAIFRA
                CLRC     INTM              ; Enable all interrupts
                RET

SISR28:
                LDP      #0E1h             ; Peripheral page
                SPLK     #0FF04h,PBDATDIR ; Set IOPB2
                CALL     DELAY
                LDP      #GPTCONA >> 7h    ; Peripheral page
                LACC     #0100h            ; clear compare int. flag
                SACL     EVAIFRA           ; in EVAIFRA
                CLRC     INTM              ; Enable all interrupts
                RET


SISR27:
                LDP       #0E1h             ; Peripheral page
                SPLK      #0FF08h,PBDATDIR ; Set IOPB3
                CALL      DELAY
                LDP       #GPTCONA >> 7h    ; Peripheral page
                LACC      #0080h            ; clear period int. flag
                SACL      EVAIFRA           ; in EVAIFRA
                CLRC      INTM              ; Enable all interrupts
                RET

DELAY           LAR       AR0,#01h          ; Gen. purpose delay
D_LOOP          RPT       #01h              ; Delay parameters may need to be
                NOP                         ; modified for easy observation
                BANZ      D_LOOP
                RET

GISR1:          RET
GISR3:          RET
GISR4:          RET
GISR5:          RET
GISR6:          RET

PHANTOM         RET

                .end
```

```
;===========================================================================
* File name :   CAP.asm                                                    *
* Description : PROGRAM TO CHECK THE CAPTURE UNITS OF ’240X                *
* This program checks the Capture units of EVA & EVB. On each EV module,   *
* the capture units are setup to detect different transitions. On EVA,     *
* CAP1 detects a rising edge, CAP2 detects a falling edge and CAP3 detects *
* both edges. All capture interrupts are enabled. Timers 1 & 2 provide     *
* input signals (through external connections) and also serve as a         *
* time–base for these capture units. Upon detection, the capture interrupt *
* reads the proper CAPFIFO value to ensure that the capture units detected *
* the correct transition. The same scheme is implemented on EVB, to check  *
* CAP4,5 and 6 using Timers 3 and 4.                                       *
;===========================================================================

* Both Timers count in CONTINUOUS–UP mode.
*     CAP1 is rising edge detect   (T1 CMP Active low)           *
*     CAP2 is falling edge detect  (T1 CMP Active low)           *
*     CAP3 on both edges           (T2 CMP Active high)          *

* This program tests the following in EVA :–                     *
*     CAP1 & CAP2  using Timer 1                                 *
*     CAP3         using Timer 2                                 *

* This program tests the following in EVB :–                     *
*     CAP4 & CAP5  using Timer 3                                 *
*     CAP6         using Timer 4                                 *

* COMMENTS: Connect T1CMP to CAP1,2; T2CMP to CAP3 inputs        *
*           Connect T3CMP to CAP4,5; T4CMP to CAP6 inputs        *

* PERIPHERAL CODE : 5 (EVA) and 6 (EVB)                          *
* TEST CODE :       CAP 1,2,3,4,5,6 – 1,2,3,1,2,3 respectively   *

                .title   ” EV capture test” ; Title
                .include ”240x.h”           ; Variable and register declaration
                .include ”vector.h”         ; Vector label declaration

del             .set     0fffh              ; define delay

                .text
START:          LDP      #0h                ; set DP=0
                SETC     INTM               ; Disable interrupts
                SPLK     #0000h,IMR         ; Mask all core interrupts
                LACC     IFR                ; Read Interrupt flags
                SACL     IFR                ; Clear all interrupt flags
                LDP      #WDKEY >> 7h       ; Peripheral page
                SPLK     #006Fh, WDCR       ; Disable WD if VCCP=5V

                LDP      #SCSR1>>7
                SPLK     #000Ch,SCSR1       ; EVA & EVB modules clock enable

                LDP      #EVAIMRA>>7        ; Peripheral page
                SPLK     #0FFFFh,EVAIFRA    ; clear all EVA interrupt flags
                SPLK     #0FFFFh,EVAIFRB
                SPLK     #0FFFFh,EVAIFRC
                LDP      #EVBIMRA>>7         ; Peripheral page
                SPLK     #0FFFFh,EVBIFRA     ; clear all EVB interrupt flags
                SPLK     #0FFFFh,EVBIFRB
                SPLK     #0FFFFh,EVBIFRC

                LAR      AR7,#del            ; Load AR7 with delay value
                MAR      *,AR7               ; Set ARP to ar7
                LDP      #0E1h               ; Peripheral page
                SPLK     #1111111111111111b,MCRA   ; enable all EV signals
                SPLK     #1111111111111111b,MCRC   ; enable all EV signals

*=====================================================================
* EVA Capture test
* This portion of the code tests the EVA Capture unit. It is assumed
* that the test is failed, unless an interrupt is called (error code 4)
* GISR4 verifies the values in CAPFIFO and reports the results.

* PERIPHERAL CODE : 5, TEST CODE : 1,2,3 After successful completion
* of this test case, the value 5100,5200,5300 must be present in 351h,
* 352, 353 (DM) respectively
*
* Error code: 5101 –– CAP1 value is incorrect
*             5102 –– CAP2 value is incorrect
*             5103 –– CAP3 value is incorrect
*=====================================================================
* Load EVA TIMERS registers
                LDP      #GPTCONA >> 7h      ; Peripheral page
                SPLK     #0000000001001001b,GPTCONA
                                             ; 0000 0000 0
                                             ; 1 – Enable Compare o/ps
                                             ; 00 reserved
                                             ; 10 – T2 CMP active hi
                                             ; 01 – T1 CMP active lo
                SPLK     #0000000000000000b,T1CNT  ; zero timer 1 count
                SPLK     #0000000000000000b,T2CNT  ; zero timer 2 count
                SPLK     #0001011101000010b,T1CON
                                             ; 000 10 Cont, Up
                                             ; 111        x/128,
                                             ; 0 reserved for T1,Tenable select
                                             ; 1 Tenable for Timer 1
                                             ; 00 Internal clk
                                             ; 00 cntr =0
                                             ; 1 enable compare
                                             ; 0 use own period register
                SPLK     #0001011111000011b,T2CON
                                             ; TSWT1=1: Use Timer 1 tenable bit
                                             ; SELT1PR=1: Use Timer 1 period
                                             ; register

                SPLK     #1111111111111111b,T1PR
                SPLK     #0011111100000000b,T1CMPR
                SPLK     #0011111100000000b,T2CMPR
                SPLK     #0000000000000000b,EVAIMRA
                SPLK     #0000000000000000b,EVAIMRB
                                             ; disable group A,B interrupts
```

```
* Load Capture registers

                SPLK     #0011001001101100b,CAPCONA
                                            ; 0 clear capture registers
                                            ; 01–enable Capture 1,2 disable QEP
                                            ; 1 –enable Capture 3
                                            ; 0 –reserved
                                            ; 0 –Use GPTimer 2 for CAP3
                                            ; 1 –Use GPTimer 1  for CAP1,2
                                            ; 0 –No ADC start on CAP3 interrupt
                                            ; 01 –CAP1 is rising edge detect
                                            ; 10 –CAP2 is falling edge detect
                                            ; 11 –CAP3 on both edges
                                            ; 00 –reserved
                SPLK     #0000000000000111b,EVAIMRC
                                            ; 0000 0000 0000 0
                                            ; 111, enable CAP3,CAP2,CAP1
                                            ; interrupts

                LDP      #6h                ; Write the failure code to begin
                                            ; with.
                SPLK     #5101h,51h         ; This will be overwritten if the
                                            ; test passes
                SPLK     #5201h,52h
                SPLK     #5301h,53h

                LDP      #0
                SPLK     #0000000000001000b,IMR    ; Enable INT4
                CLRC     INTM               ; Enable interrupts globally

                CALL     CAPDLY

*======================================================================
* EVB Capture test
* This portion of the code tests the EVB Capture unit. It is assumed
* that the test is failed, unless an interrupt is called (error code 4)
* GISR4 verifies the values in CAPFIFO and reports the results.

* PERIPHERAL CODE : 6, TEST CODE : 1 After successful completion
* of this test case, the value 6100,6200,6300 must be present in 361h,
* 362, 363 (DM) respectively.
*
* Error code: 6101 –– CAP1 value is incorrect
*             6201 –– CAP2 value is incorrect
*             6301 –– CAP3 value is incorrect
*======================================================================

* Load EVB TIMERS registers
                SETC     INTM
                LDP      #GPTCONB >> 7h     ; Peripheral page
                SPLK     #0000000001001001b,GPTCONB
                                            ; 0000 0000 0
                                            ; 1 – Enable Compare o/ps
                                            ; 00 reserved
                                            ; 10 – T2 CMP active hi
                                            ; 01 – T1 CMP active lo
                SPLK     #0000000000000000b,T3CNT  ; zero timer 3 count
                SPLK     #0000000000000000b,T4CNT  ; zero timer 4 count
                SPLK     #0001011101000010b,T3CON
                                            ; 000 10 Cont, Up
                                            ; 111        x/128,
                                            ; 0 reserved for T3,Tenable select
                                            ; 1 Tenable for Timer 3
                                            ; 00 Internal clk
                                            ; 00 cntr =0
                                            ; 1 enable compare
                                            ; 0 use own period register
                SPLK     #0001011111000011b,T4CON
                                            ; TSWT3=1: Use Timer 3 tenable bit
                                            ; SELT3PR=1: Use Timer 3 period
                                            ; register

                SPLK     #1111111111111111b,T3PR
                SPLK     #0011111100000000b,T3CMPR
                SPLK     #0011111100000000b,T4CMPR
                SPLK     #0000000000000000b,EVBIMRA
                SPLK     #0000000000000000b,EVBIMRB
                                            ; disable group A,B interrupts

* Load Capture registers

                SPLK     #0011001001101100b,CAPCONB
                                            ; 0 clear capture registers
                                            ; 01–enable Capture 4,5 disable QEP
                                            ; 1 –enable Capture 6
                                            ; 0 –reserved
                                            ; 0 –Use GPTimer 4 for CAP6
                                            ; 1 –Use GPTimer 3  for CAP4,5
                                            ; 0 –No ADC start on CAP6 interrupt
                                            ; 01 –CAP4 is rising edge detect
                                            ; 10 –CAP5 is falling edge detect
                                            ; 11 –CAP6 on both edges
                                            ; 00 –reserved
                SPLK     #0000000000000111b,EVBIMRC
                                            ; 0000 0000 0000 0
                                            ; 111, enable CAP6,CAP5,CAP4
                                            ; interrupts

                LDP      #6h                ; Write the failure code to begin
                                            ; with.
                SPLK     #6101h,61h         ; This will be overwritten if the
                                            ; test passes
                SPLK     #6201h,62h
                SPLK     #6301h,63h
                CLRC     INTM               ; Enable interrupts globally

                CALL     CAPDLY
```

```
;=====================================================================
; Exit routine
;=====================================================================
                LDP      #0h
                SPLK     #0h,IMR            ; Mask all interrupts
                LACC     IFR                ; Read Interrupt flags
                SACL     IFR                ; Clear all interrupt flags
                SETC     INTM
                LDP      #EVAIMRA>>7        ; Peripheral page
                SPLK     #0h,EVAIMRA        ; Mask all EVA interrupts
                SPLK     #0h,EVAIMRB
                SPLK     #0h,EVAIMRC
                SPLK     #0FFFFh,EVAIFRA    ; clear all EVA interrupt flags
                SPLK     #0FFFFh,EVAIFRB
                SPLK     #0FFFFh,EVAIFRC
                LDP      #EVBIMRA>>7        ; Peripheral page
                SPLK     #0h,EVBIMRA        ; Mask all EVB interrupts
                SPLK     #0h,EVBIMRB
                SPLK     #0h,EVBIMRC
                SPLK     #0FFFFh,EVBIFRA    ; clear all EVB interrupt flags
                SPLK     #0FFFFh,EVBIFRB
                SPLK     #0FFFFh,EVBIFRC
                LDP      #SCSR1>>7
                SPLK     #0000h,SCSR1       ; disable EVA & EVB clocks
                LDP      #GPTCONA>>7
                SPLK     #0000000000000000b,T1CON
                SPLK     #0000000000000000b,T2CON
                LDP      #GPTCONB>>7
                SPLK     #0000000000000000b,T3CON
                SPLK     #0000000000000000b,T4CON

DONE            B        DONE               ; End of test module

;==================================================================
; ISR
;==================================================================
GISR4:                                      ; Int4 GISR
                NOP
                LDP      #PIVR >> 7h        ; Peripheral page
                LACL     PIVR               ; PIVR value
                XOR      #0033h             ; CAP1 interrupt
                BCND     SISR33,eq
                LACL     PIVR               ; PIVR value
                XOR      #0034h             ; CAP2 interrupt
                BCND     SISR34,eq
                LACL     PIVR               ; PIVR value
                XOR      #0035h             ; CAP3 interrupt
                BCND     SISR35,eq
                LACL     PIVR               ; PIVR value
                XOR      #0036h             ; CAP4 interrupt
                BCND     SISR36,eq
                LACL     PIVR               ; PIVR value
                XOR      #0037h             ; CAP5 interrupt
                BCND     SISR37,eq
                LACL     PIVR               ; PIVR value
```

```
               XOR      #0038h              ; CAP6 interrupt
               BCND     SISR38,eq
               RET

SISR33:                                     ; CAP1 SISR
               LDP      #GPTCONA >> 7h      ; Peripheral page
               SPLK     #0001h,EVAIFRC      ; clear Capture flag
               LDP      #0h
               BLDD     #CAP1FIFO,70h
               BLDD     #CAP1FIFO,71h
               LACL     70h                 ; Check FIFO values
               XOR      #0h
               BCND     CAP1FAIL,NEQ
               LACL     71h
               XOR      #0h
               BCND     CAP1PASS,EQ
CAP1FAIL                                    ; Report CAP1 error
               LDP      #6h
               SPLK     #5101h,51h
               B        END_INT
CAP1PASS
               LDP      #6h
               SPLK     #5100h,51h
END_INT        CLRC     INTM
               RET


SISR34:                                     ; CAP2 SISR
               LDP      #GPTCONA >> 7h      ; Peripheral page
               SPLK     #0002h,EVAIFRC      ; clear Capture flag
               LDP      #0h
               BLDD     #CAP2FIFO,72h
               BLDD     #CAP2FIFO,73h
               LACL     72h                 ; Check FIFO values
               XOR      #3F00h
               BCND     CAP2FAIL,NEQ
               LACL     73h
               XOR      #3F00h
               BCND     CAP2PASS,EQ
CAP2FAIL                                    ; Report CAP2 error
               LDP      #6h
               SPLK     #5201h,52h
               B        END_INT
CAP2PASS
               LDP      #6h
               SPLK     #5200h,52h
               CLRC     INTM
               RET

SISR35:                                     ; CAP3 SISR
               LDP      #GPTCONA >> 7h      ; Peripheral page
               SPLK     #0004h,EVAIFRC      ; clear Capture flag
               LDP      #0h
               BLDD     #CAP3FIFO,74h
               BLDD     #CAP3FIFO,75h
```

```
                LACL     74h                 ; Check FIFO values
                XOR      #0h
                BCND     CAP3FAIL,NEQ
                LACL     75h
                XOR      #3F00h
                BCND     CAP3PASS,EQ
CAP3FAIL                                     ; Report CAP3 error
                LDP      #6h
                SPLK     #5301h,53h
                B        END_INT
CAP3PASS
                LDP      #6h
                SPLK     #5300h,53h
                CLRC     INTM
                RET

SISR36:                                      ; CAP4 SISR
                LDP      #GPTCONB >> 7h      ; Peripheral page
                SPLK     #0001h,EVBIFRC      ; clear Capture flag
                LDP      #0h
                BLDD     #CAP4FIFO,76h
                BLDD     #CAP4FIFO,77h
                LACL     76h                 ; Check FIFO values
                XOR      #0h
                BCND     CAP4FAIL,NEQ
                LACL     77h
                XOR      #0h
                BCND     CAP4PASS,EQ
CAP4FAIL                                     ; Report CAP4 error
                LDP      #6h
                SPLK     #6101h,61h
                B        END_INT
CAP4PASS
                LDP      #6h
                SPLK     #6100h,61h
                CLRC     INTM
                RET

SISR37:                                      ; CAP5 SISR
                LDP      #GPTCONB >> 7h      ; Peripheral page
                SPLK     #0002h,EVBIFRC      ; clear Capture flag
                LDP      #0h
                BLDD     #CAP5FIFO,78h
                BLDD     #CAP5FIFO,79h
                LACL     78h                 ; Check FIFO values
                XOR      #3F00h
                BCND     CAP5FAIL,NEQ
                LACL     79h
                XOR      #3F00h
                BCND     CAP5PASS,EQ
CAP5FAIL                                     ; Report CAP5 error
                LDP      #6h
                SPLK     #6201h,62h
                B        END_INT
```

```
CAP5PASS
                LDP      #6h
                SPLK     #6200h,62h
                CLRC     INTM
                RET

SISR38:                                   ; CAP6 SISR
                LDP      #GPTCONB >> 7h   ; Peripheral page
                SPLK     #0004h,EVBIFRC   ; clear Capture flag
                LDP      #0h
                BLDD     #CAP6FIFO,7Ah
                BLDD     #CAP6FIFO,7Bh
                LACL     7Ah              ; Check FIFO values
                XOR      #0h
                BCND     CAP6FAIL,NEQ
                LACL     7Bh
                XOR      #3F00h
                BCND     CAP6PASS,EQ
CAP6FAIL                                  ; Report CAP6 error
                LDP      #6h
                SPLK     #6301h,63h
                B        END_INT
CAP6PASS
                LDP      #6h
                SPLK     #6300h,63h
                CLRC     INTM
                RET

;========================================================================
; Delay routine
;========================================================================

CAPDLY          MAR      *,AR0            ; Routine to generate delay
                                          ; between modes
                LAR      AR0,#0FFFFh
CAPDLP2         RPT      #0FFh
                NOP
                BANZ     CAPDLP2
                RET

PHANTOM         RET

GISR1           RET
GISR2           RET
GISR3           RET
GISR5           RET
GISR6           RET
                .end
```

Appendix C

TMS320F240x Boot ROM Loader: Protocols and Interfacing

Topic Page

C.1 Introduction

The ’240x Digital Signal Processors (DSPs) contain an on-chip read-only memory (ROM) containing bootloader code. This code loads code from an external serial boot device at reset, and transfers control to the code loaded from the external device. This chapter describes working with this feature of the device.

The ’240x device Boot ROM offers the user two options—it can load code through either asynchronous or synchronous serial transfer.

The synchronous transfer is done through the Serial Peripheral Interface (SPI), and the asynchronous transfer is done through the Serial Communications Interface (SCI). The code is loaded to a user-specified location which is completely flexible—it can be anywhere in program memory where RAM is available. The serial transfer packet must contain the address as specified in the applicable Serial Transfer Format, which is described in section C.2.

C.1.1 Boot-Load Sequence

There are a few things that must be set up correctly for the control to transfer to the Boot ROM and a valid boot load to occur:

1) **Micocontroller mode.** The ’LF240x device must be placed in microcontroller mode, by pulling the MP/$\overline{\text{MC}}$ pin LOW.

2) **Boot ROM loader invocation.** The bootloader is invoked by pulling the $\overline{\text{BOOT\_EN}}$/XF pin low through a resistor, prior to device reset. This transfers control to the boot-load program located in the on-chip ROM. At reset time, internal logic takes a “snapshot” of this pin and if this pin is a low level, then the Boot ROM appears in the memory map as shown in Figure C–2. Otherwise, the on-chip flash memory is enabled and the program counter begins execution at 0000h. This pin can be driven high/low from a control source such as a host processor or through a jumper via a resistor, allowing control of the boot sequence of the DSP. **The resistor must be present, since the XF pin is an output at all other times.**

3) **SCI or SPI selection.** The bootloader code selects the source of the incoming code, depending on the state of the IOPC2 pin on the device. The code takes a snapshot of this code after being invoked, and determines which loader (SPI or SCI) to invoke based on the status of this pin.

   - If IOPC2 is pulled low, an SCI transfer is commenced
   - If IOPC2 is pulled high, an SPI transfer is commenced
   - **Note that the SPI selection is invalid on devices without the SPI**

It is suggested that this pin should be driven via a resistor as well, since if the SPI is used at any time during the operation of the system, SPISIMO will be an output.

4) **Destination check.** The incoming destination is now compared to FE00h to FFFFh. If the destination matches this range, the CNF bit (Bit 12) in the status register ST1 is set, configuring the DARAM memory block B0 to Program Memory Space. Any other checks are not preformed and it is entirely up to the host/external boot device to supply a valid combination of memory destination address and length for the incoming code.This means that the target code must fit into the internal memory or external memory must be available.

5) **Data transfer.** Once the incoming destination and length are fetched (this protocol is defined in separate sections for the SCI and SPI), the actual data transfer commences. The fetch is basically destination, length, and data with no error-checking. On the SCI, the incoming data is echoed back, allowing the host to implement error-checking, if desired.

6) **Execution of incoming code.** Once the Boot ROM loader completes transfer of the packet, a branch is made into the incoming code.

7) **Watchdog.** The watchdog timer on the device is active during the entire sequence and is being reset at key points in the code. When a branch is made into the user code, it is the responsibility of the user code to handle watchdog overflow as appropriate.

8) **Restrictions on the incoming code.**

   - The combination of the destination address and transfer length *must* point to valid locations. There is absolutely no error-handling whatsoever.
   - The combination also must point to a memory block that is contiguous.
   - The address check is performed only on the first location of the incoming destination. It is expected that this allows for enough space for the rest of the incoming words. This means that if you have external program memory at range FDFEh to FDFFh (two words) and you attempt to load code into the range FDFEh to FEFFh (attempting to use the internal memory FE00h to FEFFh), this combination is invalid since the destination check will *not* switch B0 into the program space upon encountering the destination FDFEh. Lastly, the incoming address and length are expected to be 16 bits, as defined in the SPI and SCI transfer protocols.

Figure C–1. Example Hardware Configuration for ’LF240x Boot ROM Operation

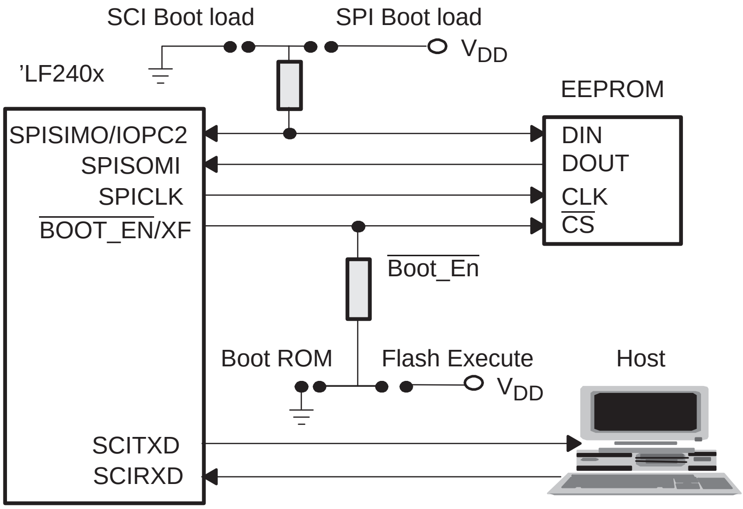
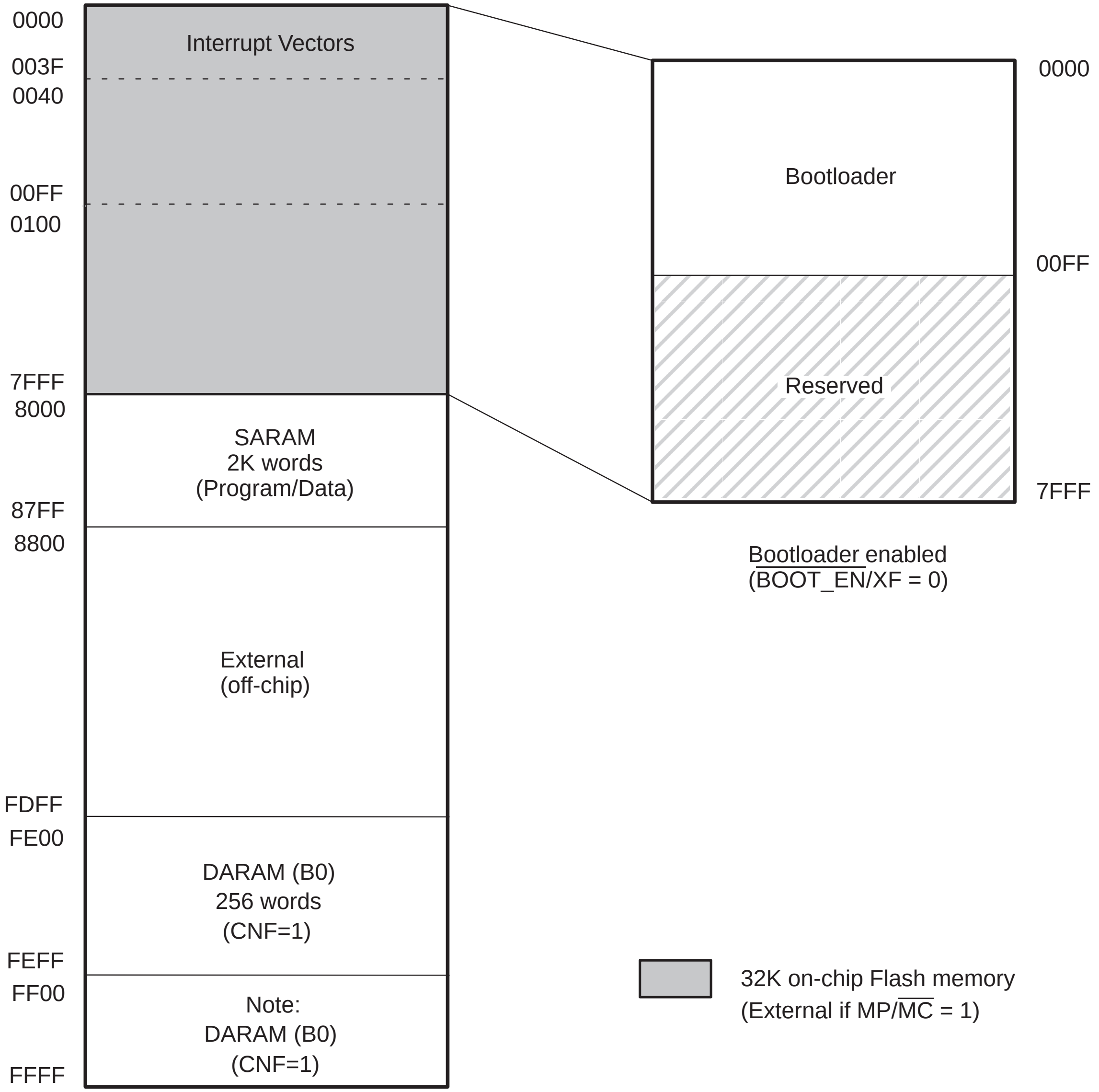


Figure C–2. Memory Maps for the 'LF240x Devices in Microcontroller Mode



Note: When boot ROM is enabled, on-chip locations 0000–00FFh in program memory is mapped to the bootloader. Boot ROM and Flash Memory share the same starting address, and hence, are not visible (active) at the same time. If the BOOT_EN/XF pin = 0 during reset, the BOOT_EN bit in SCSR2 register (bit 3) will be set and enable the Boot ROM at 0000 in program space. While Boot ROM is enabled, the entire Flash memory will be disabled. The SCSR2.3 bit should be disabled (0) to have Flash array enabled instead of Boot ROM. See Appendix C for bootloader details.

C.2 Protocol Definitions

The transfer of data is done according to a defined protocol for the SPI and SCI. The protocol for the synchronous transfer over the SPI is discussed in section C.2.1, and the protocol for the SCI transfer is discussed in section C.2.2.

C.2.1 SPI Synchronous Transfer Protocol and Data Formats

The ROM loader expects an 8-bit-wide SPI-compatible EEPROM device to be present on the SPI pins as indicated in Figure C–1. If the download is to be performed from a SPI port on another device, then that device must be set up to operate in the slave mode and mimic a serial EEPROM. Immediately after entering the SPI loader, the pin functions for the SPI pins are set to primary, and the SPI is initialized. The initialization is done at the slowest speed possible. The data transfer is done in “burst” mode for the EEPROM. The transfer is carried out entirely in byte mode (SPI at 8 bits/character). A step-by-step description of the sequence follows:

1) The SPI is initialized.

2) The XF pin is now used as a chip-select for the EEPROM.

3) The SPI outputs a read command for the EEPROM (03h).

4) The SPI sends the EEPROM an address 0000h; i.e., the host requires that the EEPROM must have the downloadable packet beginning at address 0000h in the EEPROM.

5) From this point onward, the next two bytes fetched constitute the destination address.

   The most significant byte of this word is the byte read first, and the least significant byte is the next byte fetched. This is true of all word transfers on the SPI.

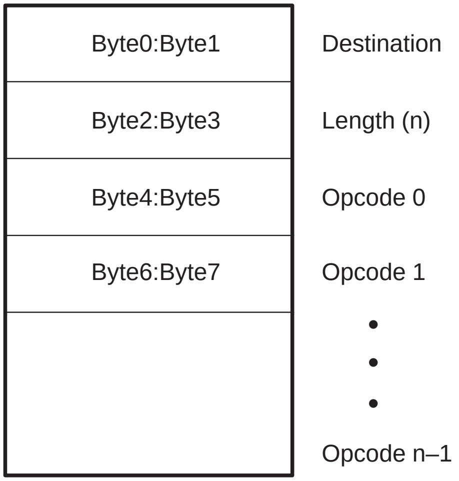
6) The next word (two bytes) fetched is the length N.

7) The destination is checked to see if it is in the range FE00h to FFFFh. If necessary, the DARAM block B0 is configured in program memory space.

8) From now on, N words are fetched and stored in program memory at the address pointed to by destination. The EEPROM is read off in one continuous burst.

9) Finally, once the last word is stored, a simple branch is made into the code at the destination address; therefore, the entry point for the boot-loaded code must be at the destination address.

Figure C–3. SPI Data Packet Definition



C.2.2 SCI Asynchronous Transfer Protocol and Data Formats

The SCI-based loader operation is more involved than the SPI-based loader operation. The SCI-based loader incorporates a mechanism for baud-rate matching. Once the baud rate from the host is matched, the SCI loader commences the transfer. Section C.2.2.1 describes the baud rate protocol.

C.2.2.1 Baud Rate Protocol

The baud rate over the communication link is always 38400 bps. The baud rate protocol is necessary because the ’LF240x device may be operated at different speeds. The underlying assumption for the baud rate matching is that the device is clocked at a clock frequency from a given, predetermined set. The host is required to send “probe” characters, with the hexadecimal value 0Dh (same as the carriage return character). The target listens in on the serial port, at the set speeds, in succession. Every time a character is detected, it is compared to 0Dh. If more than three characters do not match, the target tries a new baud rate. If the baud rate is correct and the character matches 0Dh, then the target expects to receive nine correct characters back-to-back. If any other character is received, the baud match fails. Once the nine characters are received correctly, the target sends an acknowledge character. Once the acknowledge character is sent, each and every character hereafter is bounced back to the host to ensure data transfer integrity. All the communications are with 8 bits/character, 1 stop bit, and no parity. Baud rate locks are possible at clock speeds/CLKIN combinations listed in Table C–1. A flowchart of the baud rate match protocol is shown in Figure C–4.

Table C–1. Clock Speeds at Which Baud Rate Locks

CLKOUT (MHz)	CLKIN (MHz)
30	7.5
28	7.0
24	6.0
20	5.0
16	4.0

C.2.2.2 Data Transfer

Once the communications are synchronized, the actual data transfer is commenced. The first two bytes fetched are interpreted as the destination. The next two bytes fetched are the length. Once the destination is known, a check is performed to see if the destination lies within B0. If it does, then the bootloader will switch the block B0 into program memory and transfer code into B0. After this, the user code is transferred to the destination and then a branch is executed to the first address of the code. So, as is the case with the SPI, the entry point of the code must be at the first location for the SCI.

C.2.2.3 SCI Data Transfer Completion

A noteworthy point is the completion of transmission of the SCI data echo. There can be a character still in transmission when the control is transferred to the user code. So, if the user code does anything which disturbs the transmission of the SCI, then that last character may be lost. An example is changing the bit definition of the SCI transmit pin. If this is allowed, the host must take this fact into account. A second possible option is to incorporate a small delay in the user code, or perform a SCI check to confirm that the character transmission is complete.

Figure C–4. Flowchart for the Serial Loader Baud Rate Match Algorithm

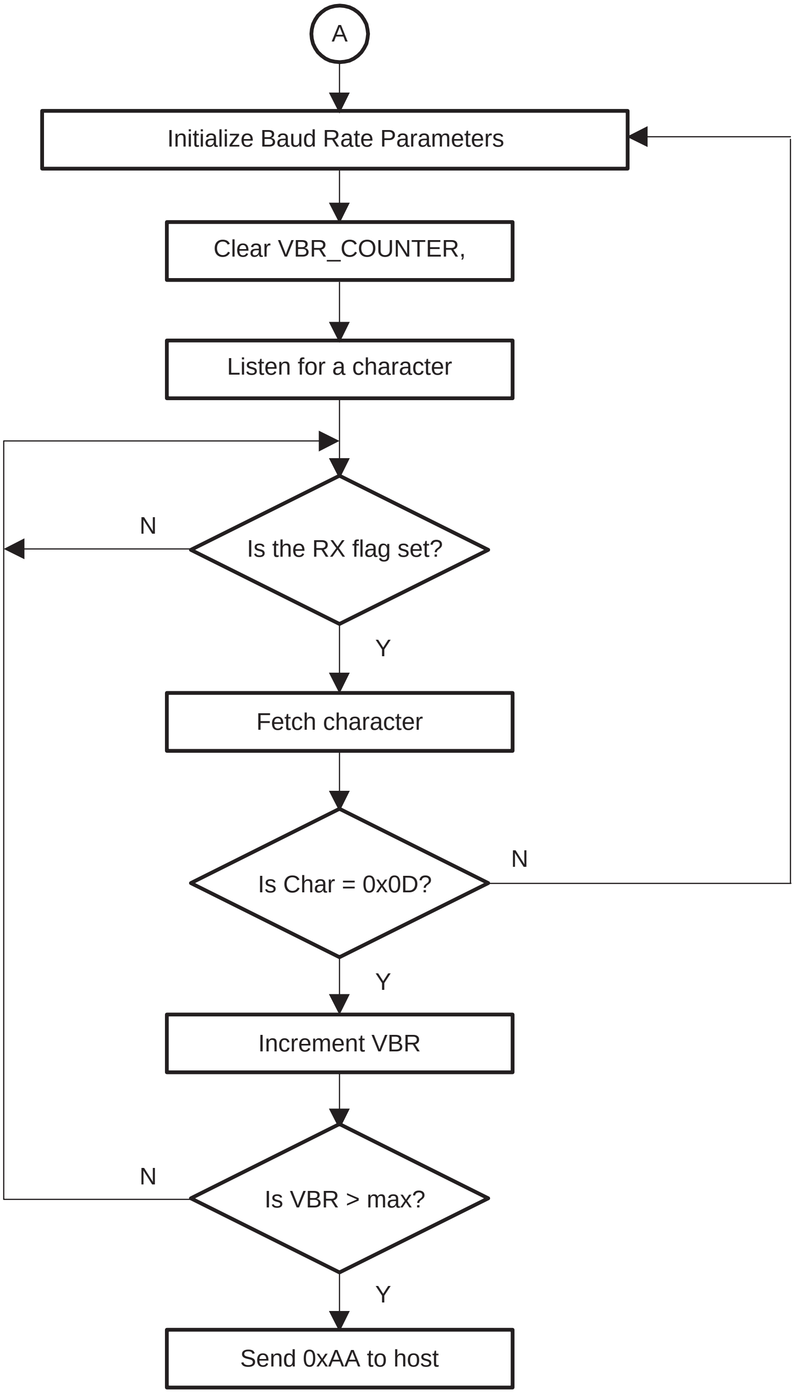


Figure C–5. Flowcharts for (a) Serial Asynchronous Loader and the Fetch Header Routine

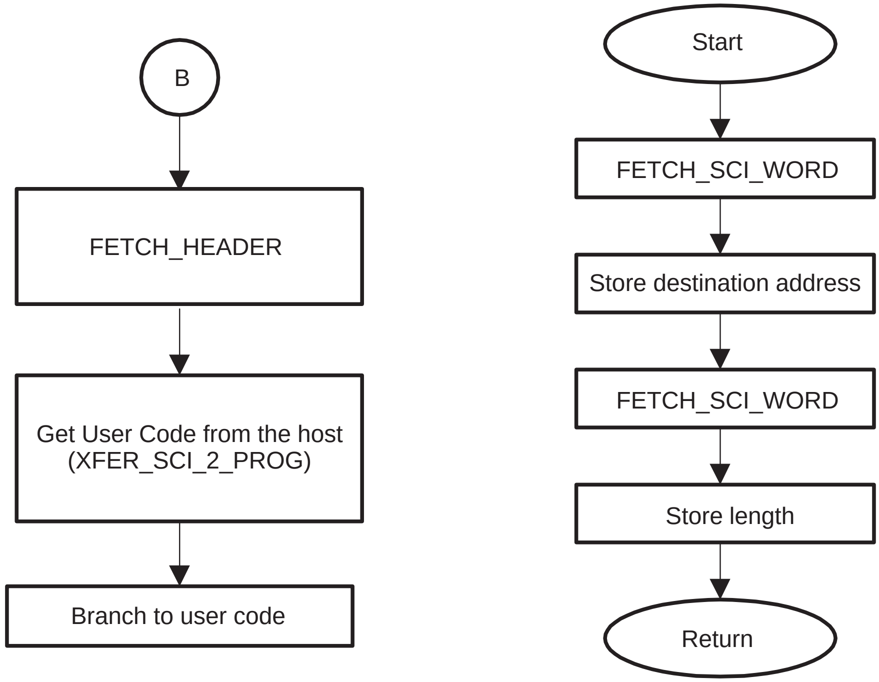
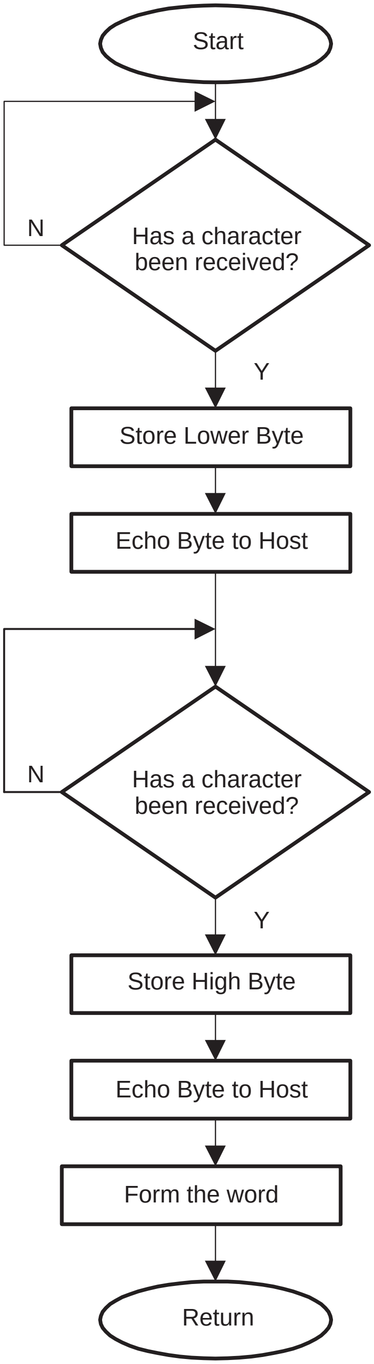


Figure C–6. Flowchart for FETCH_SCI_WORD



```
;******************************************************************************
; File Name:    BOOT.asm
; Originator:   Digital Control Systems Group,
;               Texas Instruments
;******************************************************************************
; Constant definitions
;******************************************************************************
READ_COMMAND    .set    0300H           ;Serial EEPROM Read Command in HByte
VBR_MAX         .set    09h             ;# times valid char needs to be received
CRC_MAX         .set    03h             ;# retries at each speed.

;******************************************************************************
; Debug directives
;------------------------------------------------------------------------------
                .def    GPR0            ;General purpose registers.
                .def    GPR1
                .def    GPR2
                .def    GPR3
                .def    DEST
                .def    LENGTH
                .def    data_buf
                .def    VBR_CNTR
                .def    DELAY
                .def    CHAR_RETRY_CNTR
                .def    BAUD_TBL_PTR
;******************************************************************************
;Include header file for peripheral address references.
;
;******************************************************************************
                .include        24x.h
;******************************************************************************
; Variable Declarations for on chip RAM Blocks
;------------------------------------------------------------------------------
                .bss    GPR0,1          ;General purpose registers.
                .bss    GPR1,1
                .bss    GPR2,1
                .bss    GPR3,1
                .bss    DEST,1
                .bss    LENGTH,1
                .bss    stk0,1
                .bss    stk1,1
                .bss    data_buf,1
                .bss    VBR_CNTR,1
                .bss    DELAY,1
                .bss    CHAR_RETRY_CNTR,1
                .bss    BAUD_TBL_PTR,1

;******************************************************************************
; M A C R O – Definitions
;******************************************************************************
POINT_B1        .macro
                LDP     #06h
                .endm

POINT_PF1       .macro
```

```
                LDP     #0E0h
                .endm


;******************************************************************************
; M A I N   C O D E  – starts here
;******************************************************************************
                .text
START:          LDP    #WDCR>>7
                SPLK   #0060h,SCSR      ;CLKOUT=CPUCLK

SELECT_LOADER:
                LDP    #PCDATDIR>>7
                SPLK   #0000H,PCDATDIR  ;Config all as i/ps.
                                        ;We DEPEND on SPI Pins to be I/Os by
                                        ;default at reset(controlled by OCRB).
                LACC   PCDATDIR
                AND    #0004H           ;Mask for check on IOPC2.

                BCND   SCI_LOADER,EQ    ;if IOPC2 is low, branch to SCI load
                                        ;else SPI loader and set SPISTE high.

                SETC   XF               ;Drive (!CS=XF) High.

;******************************************************************************
; SPI Initialization
;******************************************************************************
SPI_INIT:       LACC   OCRB             ;Set up the SPI pins to primary
                                        ;functions.
                OR     #001CH
                SACL   OCRB
                LDP    #SPICCR>>7
                SPLK   #0007h, SPICCR   ;8 char bits,
                SPLK   #000Eh, SPICTL   ;Enable master mode and enable talk.
                SPLK   #007fh, SPIBRR   ;SPI Speed =ASAP = as slow as possible
                SPLK   #0087h,SPICCR    ;Relinquish SPI from Reset.
;******************************************************************************
;Select the Serial EEPROM
;by driving the select line low.
;******************************************************************************
CS_ACTIVE       CLRC   XF               ;Drive (!CS=XF) Low.

;******************************************************************************
;Next send the Serial EEPROM a ’Read Command’ – it is then
;read out in burst mode, two bytes at a time by using GET_WORD
; Note that CS stays low all the time.
;******************************************************************************
                LACC   #READ_COMMAND    ;Load Read Command for EEPROM
                CALL   XMIT_VALUE       ;Transmit Read Command.

;******************************************************************************
;Now send a word(16 bits) to the EEPROM as address.
;Hard coded zero bytes are sent by the GET_WORD, but this is fine
; since we define the EEPROM to contain boot code at origin.
;
```

```
;*****************************************************************************
                CALL    GET_WORD         ;Get word Sends two zero chars
                                         ;i.e. Top address in EEPROM
;*****************************************************************************
;Ok, now do two word transfers and get the two words for
;(DEST)ination and (LENGTH) of code to be boot–loaded.
;*****************************************************************************
                CALL    GET_WORD
                SACL    DEST
                SACL    GPR1             ;GPR1 used as dest ptr in TBLW
                CALL    CHECK_DEST       ;Decide is B0 is to be switched to
                                         ;program space
                CALL    GET_WORD
                SACL    LENGTH
;*****************************************************************************
;This segment does all the work to tranfer the code to program memory.
;
;*****************************************************************************
                MAR     *,AR0
                LAR     AR0,LENGTH       ;Load AR0 and set ARP=AR0
                SBRK    #1               ;AR=length–1, since the loop is
                                         ;executed N times for AR=(N–1).

XFER_TO_PROG:   LDP     #00E0h           ;DP now points to WDOG/SPI/SCI
                CALL    KICK_DOG         ;Watchdog reset routine
                                         ;KICK_DOG needs the DP to be 0xE0.

                CALL    GET_WORD         ;get WORD doesn’t depend on DP.

                SACL    GPR0             ;Store word in GPR0 (temp storage)
                LACC    GPR1             ;Get current dest ptr in ACC.
                TBLW    GPR0             ;GPR0 is transferred to PGM_MEM
                                         ;pointed to by acc

                ADD     #1               ;Acc now points to the next location.
                SACL    GPR1             ;Store incremented pointer.
                BANZ    XFER_TO_PROG     ;Repeat length–1 times

;*****************************************************************************
;OK. Finally the program is loaded in the memory and lets branch to
;it and get there !
;As a last thing The Chip Select is de–activated.
;*****************************************************************************



CS_NOT_ACTIVE:
                SETC    XF               ;Drive (!CS=XF) High.
                POINT_B1
                LACC    DEST             ;Branch to Boot Loader Code.
                BACC                     ;
```

```
;******************************************************************************
;
;G E T _ W O R D
;
;This routine gets a word from the EEPROM and packs it.
;It’s returned in the accumulator
;
;
;
;Exit Conditions:
;                    1. DP is set to B1 on Exit.
;                    2. ACC,GPR0 are destroyed.
;                    3. Result returned in ACC
;                    4. Doesnt care about DP on enter.
;
;******************************************************************************

GET_WORD:       LACC   #0000H             ;Zero Character
                CALL   XMIT_VALUE         ;Transmit char & get response
                POINT_B1
                AND    #0FFH              ;MSByte of  start address is in the
                                          ;acc. Get rid of any higher bits
                SACL   GPR0               ;
                LACC   #0000H             ;Zero Character
                CALL   XMIT_VALUE         ;Transmit char & get response
                POINT_B1
                                          ;LSByte of start address is in ACC
                AND    #00FFH             ;Mask any upper byte
                ADD    GPR0,8             ;Bring in the MS Byte.
                RET

;******************************************************************************
; Transmit a char on the SPI Bus and return received data in accumulator
;
;
;Exit Conditions:
;                    1. DP is set to B1 on Exit
;                    2. ACC is destroyed.
;                    3. Doesnt care about DP on enter
;
;******************************************************************************
XMIT_VALUE:     LDP    #SPITXBUF>>7
                SACL   SPITXBUF           ;Write xmit value to SPI TX Buffer.
XMIT_NCOMPL:    BIT    SPISTS,BIT6        ;Test SPI_INT bit
                BCND   XMIT_NCOMPL,NTC    ;If (bit=TC=0) ,then wait for TX Compl
                                          ;i.e. wait for the completion of TX
                LACC   SPIRXBUF           ;Read,also clears SPI_INT flag.
                RET

;******************************************************************************
;    The rest of the implementation is the Asynchronous serial port loader.
;******************************************************************************
```

```
;Initialisation
;*****************************************************************************
SCI_LOADER:
UART_INIT:

;*****************************************************************************
;SCI Initialization
;*****************************************************************************
SCI_INIT:       LDP     #OCRA>>7
                LACC    OCRA            ; Set up pins as SCI pins.
                OR      #0003H
                SACL    OCRA
                LDP     #SCICCR>>7      ;1 stop bit,no parity,8bits/ch
                SPLK    #0007h, SCICCR  ;async mode,idle–line protocol
                LACK    #0

                SACL    SCICTL2         ;Disable RX Int, TX Int.
                SACL    SCIHBAUD
                SACL    SCIPRI

;*****************************************************************************
;The SCI module is held in ’reset’ untill we load the parameter
;for the Baud Rate register from the Baud Rate table in SCILBAUD
;So the next line stays commented out!
;*****************************************************************************
;               SPLK    #0023h, SCICTL1      ;Relinquish SCI from Reset.
;*****************************************************************************
;Baudrate lock protocol with Host
;*****************************************************************************
CLR_VBR_CNTR:   POINT_B1
                SACL    CHAR_RETRY_CNTR ;Clear retry counter
                SACL    VBR_CNTR        ;Clear valid baud rate counter
                SACL    BAUD_TBL_PTR    ;BAUD_TBL_PTR is really only
                                        ;the offset from BAUD_TBL.
UI00
SET_BAUD:       LACC    BAUD_TBL_PTR
                ADD     #BAUD_TBL
                POINT_PF1

                SPLK    #0013h, SCICTL1 ;Enable TX,RX,internal SCICLK,
                TBLR    SCILBAUD
                SPLK    #0023h, SCICTL1 ;Relinquish SCI from Reset.

UI01            CALL    KICK_DOG
                BIT     SCIRXST,BIT6    ;Test RXRDY bit
                BCND    UI01, NTC       ;If RXRDY=0,then repeat loop
                LACC    SCIRXBUF        ;First byte is Lo byte

       ;Check if Char is as expected
CHECK_CHAR      AND     #0FFh           ;Clear upper byte
                SUB     #00Dh           ;Compare with ”CR”
                BCND    BAUD_RETRY, NEQ
```

```
INC_VBRC        POINT_B1
                LACC    VBR_CNTR        ;Inc VBR counter
                ADD     #1h
                SACL    VBR_CNTR
                SUB     #VBR_MAX        ;Is VBR counter > max value ?
                POINT_PF1
                BCND    UI01, NEQ       ;No! fetch another char

SND_ECHO        LACC    #0Aah           ;Yes!
                SACL    SCITXBUF        ;Indicate Host Baudrate lock
                B       BAUD_DETECTED

BAUD_RETRY      POINT_B1
                SPLK    #0h, VBR_CNTR
                LACC    CHAR_RETRY_CNTR ;Inc CRC counter
                ADD     #1h
                SACL    CHAR_RETRY_CNTR
                SUB     #CRC_MAX        ;Is CRC > max value ?
                BCND    INC_TBL_PTR,GEQ ;Yes! try next baudrate
                POINT_PF1
                B       UI01            ;No! fetch another char

INC_TBL_PTR     LACC    BAUD_TBL_PTR    ;Inc CRC counter
                ADD     #1h
                AND     #0007H          ; BAUD_TBL_PTR is MOD(8)
                SACL    BAUD_TBL_PTR
                SPLK    #0h, CHAR_RETRY_CNTR
                B       UI00
BAUD_DETECTED:
;******************************************************************************
;M A I N   P R O G R A M
;******************************************************************************
MAIN:
                ;Load & Execute incoming algorithm.

M00             CALL    FETCH_HEADER
                CALL    CHECK_DEST

M01:            CALL    XFER_SCI_2_PROG
                LACC    DEST
                BACC                    ; Branch to the address we
                                        ; loaded code to.


;******************************************************************************
; Routine Name: F E T C H _ H E A D E R       Routine Type: SR
;******************************************************************************
FETCH_HEADER:   CALL    FETCH_SCI_WORD
                LACC    data_buf
                SACL    DEST
                CALL    FETCH_SCI_WORD
                LACC    data_buf
                SACL    LENGTH
                RET
```

```
;*****************************************************************************
; Routine Name: X F E R _ S C I _ 2 _ P R O G       Routine Type: SR
;*****************************************************************************
XFER_SCI_2_PROG:
                MAR     *, AR0
                LAR     AR0, LENGTH
                LACC    DEST            ;ACC=dest address

XSP0            CALL    FETCH_SCI_WORD
                TBLW    data_buf        ;data_buff-->[*ACC]
                ADD     #01h            ;ACC++
                BANZ    XSP0            ;loop ”length” times
                RET

;*****************************************************************************
; Routine Name: F E T C H _ S C I _ W O R D       Routine Type: SR
;
; Description: Version which expects Lo byte / Hi byte sequence from Host &
;              also echoes byte
;*****************************************************************************
FETCH_SCI_WORD:
                POINT_B1
                SACL    stk0
                LDP     #SCIRXST>>7
FSW0            CALL    KICK_DOG
                BIT     SCIRXST,BIT6    ;Test RXRDY bit
                BCND    FSW0, NTC       ;If RXRDY=0,then repeat loop
                LACC    SCIRXBUF        ;First byte is Lo byte
                SACL    SCITXBUF        ;Echo byte back
                AND     #0FFh           ;Clear upper byte

FSW1            CALL    KICK_DOG
                BIT     SCIRXST,BIT6    ;Test RXRDY bit
                BCND    FSW1, NTC       ;If RXRDY=0,then repeat loop
                NOP
                ADD     SCIRXBUF,8      ;Concatenate Hi byte to Lo
                SFL                     ;used ’cause 7 is max in SACH
                SACH    SCITXBUF,7      ;Echo byte back (after SFL 8)

                POINT_B1
                SFR                     ;restore ACC as before
                SACL    data_buf        ;Save received word

                LACC    stk0
                RET

;*****************************************************************************
; Check the destination address, and switch B0 into program space if needed.
;*****************************************************************************

CHECK_DEST:     LACC    DEST
                AND     #0FE00H         ;Anywhere in B0 means flip B0
                SUB     #0FE00h         ;use both pri & sec B0 ranges
                AND     #0FFFFH         ;mask bits in Acc High.
                BCND    CHK_DST_EXIT,NEQ
                SETC    CNF
CHK_DST_EXIT:   RET

;******************************************************************************

KICK_DOG:       SPLK    #05555h, WDKEY
                SPLK    #0AAAAh, WDKEY
                RET
;******************************************************************************
; Table of SCI_LBAUD Contents.
;******************************************************************************
;            SCI_LBAUD @38.4 Kbps ;
;------------------------------------------------------------------------------
BAUD_TBL:              .word 130  ;
                       .word 117  ;
                       .word 104  ;
                       .word 97   ;
                       .word 91   ;
                       .word 78   ;
                       .word 65   ;
                       .word 52   ;
                       .end
```

Appendix D

Glossary

A

A0–A15: Collectively, the external address bus; the 16 pins are used in parallel to address external data memory, program memory, or I/O space.

ACC: See *accumulator.*

ACCH: *Accumulator high word.* The upper 16 bits of the accumulator. See also *accumulator*.

ACCL: *Accumulator low word.* The lower 16 bits of the accumulator. See also *accumulator*.

accumulator: A 32-bit register that stores the results of operations in the central arithmetic logic unit (CALU) and provides an input for subsequent CALU operations. The accumulator also performs shift and rotate operations.

address: The location of program code or data stored in memory.

addressing mode: A method by which an instruction interprets its operands to acquire the data it needs. See also *direct addressing*; *immediate addressing*; *indirect addressing*.

analog-to-digital (A/D) converter: A circuit that translates an analog signal to a digital signal.

AR: See *auxiliary register*.

AR0–AR7: *Auxiliary registers 0 through 7.* See *auxiliary register*.

ARAU: See *auxiliary register arithmetic unit (ARAU).*

ARB: See *auxiliary register pointer buffer (ARB)*.

ARP: See *auxiliary register pointer (ARP).*

auxiliary register: One of eight 16-bit registers (AR7–AR0) used as pointers to addresses in data space. The registers are operated on by the auxiliary register arithmetic unit (ARAU) and are selected by the auxiliary register pointer (ARP).

auxiliary register arithmetic unit (ARAU): A 16-bit arithmetic unit used to increment, decrement, or compare the contents of the auxiliary registers. Its primary function is manipulating auxiliary register values for indirect addressing.

auxiliary register pointer (ARP): A 3-bit field in status register ST0 that points to the current auxiliary register.

auxiliary register pointer buffer (ARB): A 3-bit field in status register ST1 that holds the previous value of the auxiliary register pointer (ARP).

B

B0: An on-chip block of dual-access RAM that can be configured as either data memory or program memory, depending on the value of the CNF bit in status register ST1.

B1: An on-chip block of dual-access RAM available for data memory.

B2: An on-chip block of dual-access RAM available for data memory.

$\overline{\text{BIO}}$ pin: A general-purpose input pin that can be tested by the conditional branch instruction (BCND) that causes a branch when $\overline{\text{BIO}}$ is driven low externally.

bit-reversed indexed addressing: A method of indirect addressing that allows efficient I/O operations by resequencing the data points in a radix-2 fast Fourier transform (FFT) program. The direction of carry propagation in the ARAU is reversed.

bootloader: A built-in segment of code that transfers code from an external source to a 16-bit external program destination at reset.

branch: A switching of program control to a nonsequential program-memory address.

BRR: The value in the baud select registers (SPI).

C

C bit: See *carry bit.*

CALU: See *central arithmetic logic unit (CALU).*

carry bit: Bit 9 of status register ST1; used by the CALU for extended arithmetic operations and accumulator shifts and rotates. The carry bit can be tested by conditional instructions.

central arithmetic logic unit (CALU): The 32-bit wide main arithmetic logic unit for the ’24x CPU that performs arithmetic and logic operations. It accepts 32-bit values for operations, and its 32-bit output is held in the accumulator.

CLKIN: *Input clock signal*. A clock source signal supplied to the on-chip clock generator at the CLKIN/X2 pin or generated internally by the on-chip oscillator. The clock generator divides or multiplies CLKIN to produce the CPU clock signal, CLKOUT.

CLKOUT: *Master clock output signal*. The output signal of the on-chip clock generator. The CLKOUT high pulse signifies the CPU’s logic phase (when internal values are changed), and the CLKOUT low pulse signifies the CPU’s latch phase (when the values are held constant).

CNF bit: *DARAM configuration bit*. Bit 12 in status register ST1. CNF is used to determine whether the on-chip RAM block B0 is mapped to program space or data space.

codec: A device that codes in one direction of transmission and decodes in another direction of transmission.

COFF: *Common object file format.* A system of files configured according to a standard developed by AT&T. These files are relocatable in memory space.

context saving/restoring: Saving the system status when the device enters a subroutine (such as an interrupt service routine) and restoring the system status when exiting the subroutine. On the ’24x, only the program counter value is saved and restored automatically; other context saving and restoring must be performed by the subroutine.

CPU: *Central processing unit*. The ’24x CPU is the portion of the processor involved in arithmetic, shifting, and Boolean logic operations, as well as the generation of data- and program-memory addresses. The CPU includes the central arithmetic logic unit (CALU), the multiplier, and the auxiliary register arithmetic unit (ARAU).

CPU cycle: The time required for the CPU to go through one logic phase (during which internal values are changed) and one latch phase (during which the values are held constant).

current AR: See *current auxiliary register*.

current auxiliary register: The auxiliary register pointed to by the auxiliary register pointer (ARP). The auxiliary registers are AR0 (ARP = 0) through AR7 (ARP = 7). See also *auxiliary register*; *next auxiliary register*.

current data page: The data page indicated by the content of the data page pointer (DP). See also *data page*; *DP*.

D

D0–D15: Collectively, the external data bus; the 16 pins are used in parallel to transfer data between the ’24x and external data memory, program memory, or I/O space.

DARAM: *Dual-access RAM*. RAM that can be accessed twice in a single CPU clock cycle. For example, your code can read from and write to DARAM in the same clock cycle.

DARAM configuration bit (CNF): See *CNF bit*.

data-address generation logic: Logic circuitry that generates the addresses for data memory reads and writes. This circuitry, which includes the auxiliary registers and the ARAU, can generate one address per machine cycle. See also *program-address generation logic.*

data page: One block of 128 words in data memory. Data memory contains 512 data pages. Data page 0 is the first page of data memory (addresses 0000h–007Fh); data page 511 is the last page (addresses FF80h–FFFFh). See also *data page pointer (DP)*; *direct addressing*.

data page 0: Addresses 0000h–007Fh in data memory; contains the memory-mapped registers, a reserved test/emulation area for special information transfers, and the scratch-pad RAM block (B2).

data page pointer (DP): A 9-bit field in status register ST0 that specifies which of the 512 data pages is currently selected for direct address generation. When an instruction uses direct addressing to access a data-memory value, the DP provides the nine MSBs of the data-memory address, and the instruction provides the seven LSBs.

data-read address bus (DRAB): A 16-bit internal bus that carries the address for each read from data memory.

data read bus (DRDB): A 16-bit internal bus that carries data from data memory to the CALU and the ARAU.

data-write address bus (DWAB): A 16-bit internal bus that carries the address for each write to data memory.

data write bus (DWEB): A 16-bit internal bus that carries data to both program memory and data memory.

decode phase: The phase of the pipeline in which the instruction is decoded. See also *pipeline*; *instruction-fetch phase*; *operand-fetch phase; instruction-execute phase*.

direct addressing: One of the methods used by an instruction to address data-memory. In direct addressing, the data-page pointer (DP) holds the nine MSBs of the address (the current data page), and the instruction word provides the seven LSBs of the address (the offset). See also *indirect addressing*.

DP: See *data page pointer (DP).*

DRAB: See *data-read address bus (DRAB).*

DRDB: See *data read bus (DRDB).*

$\overline{\text{DS}}$: *Data memory select pin*. The ’24x asserts $\overline{\text{DS}}$ to indicate an access to external data memory (local or global).

DSWS: *Data-space wait-state bit(s).* A value in the wait-state generator control register (WSGR) that determines the number of wait states applied to reads from and writes to off-chip data space.

dual-access RAM: See *DARAM*.

dummy cycle: A CPU cycle in which the CPU intentionally reloads the program counter with the same address.

DWAB: See *data-write address bus (DWAB).*

DWEB: See *data write bus (DWEB).*

E

execute phase: The fourth phase of the pipeline; the phase in which the instruction is executed. See also *pipeline*; *instruction-fetch phase*; *instruction-decode phase*; *operand-fetch phase*.

external interrupt: A hardware interrupt triggered by an external event sending an input through an interrupt pin.

F

FIFO buffer: *First-in, first-out buffer.* A portion of memory in which data is stored and then retrieved in the same order in which it was stored. The synchronous serial port has two four-word-deep FIFO buffers: one for its transmit operation and one for its receive operation.

flash memory: Electrically erasable and programmable, nonvolatile (read-only) memory.

G

general-purpose input/output pins: Pins that can be used to accept input signals or send output signals. These pins are the input pin $\overline{\text{BIO}}$, the output pin XF, and the GPIO pins.

H

hardware interrupt: An interrupt triggered through physical connections with on-chip peripherals or external devices.

I

IFR: See *interrupt flag register (IFR)*.

immediate addressing: One of the methods for obtaining data values used by an instruction; the data value is a constant embedded directly into the instruction word; data memory is not accessed.

immediate operand/immediate value: A constant given as an operand in an instruction that is using immediate addressing.

IMR: See *interrupt mask register (IMR)*.

indirect addressing: One of the methods for obtaining data values used by an instruction. When an instruction uses indirect addressing, data memory is addressed by the current auxiliary register. See also *direct addressing*.

input clock signal: See *CLKIN*.

input shifter: A 16- to 32-bit left barrel shifter that shifts incoming 16-bit data from 0 to 16 positions left relative to the 32-bit output.

instruction-decode phase: The second phase of the pipeline; the phase in which the instruction is decoded. See also *pipeline*; *instruction-fetch phase*; *operand-fetch phase; instruction-execute phase*.

instruction-execute phase: The fourth phase of the pipeline; the phase in which the instruction is executed. See also *pipeline*; *instruction-fetch phase*; *instruction-decode phase*; *operand-fetch phase*.

instruction-fetch phase: The first phase of the pipeline; the phase in which the instruction is fetched from program-memory. See also *pipeline*; *instruction-decode phase*; *operand-fetch phase; instruction-execute phase*.

instruction register (IR): A 16-bit register that contains the instruction being executed.

instruction word: A 16-bit value representing all or half of an instruction. An instruction that is fully represented by 16 bits uses one instruction word. An instruction that must be represented by 32 bits uses two instruction words (the second word is a constant).

internal interrupt: A hardware interrupt caused by an on-chip peripheral.

interrupt: A signal sent to the CPU that (when not masked or disabled) forces the CPU into a subroutine called an interrupt service routine (ISR). This signal can be triggered by an external device, an on-chip peripheral, or an instruction (INTR, NMI, or TRAP).

interrupt flag register (IFR): A 16-bit memory-mapped register that indicates pending interrupts. Read the IFR to identify pending interrupts and write to the IFR to clear selected interrupts. Writing a 1 to any IFR flag bit clears that bit to 0.

interrupt latency: The delay between the time an interrupt request is made and the time it is serviced.

interrupt mask register (IMR): A 16-bit memory-mapped register used to mask external and internal interrupts. Writing a 1 to any IMR bit position enables the corresponding interrupt (when INTM = 0).

interrupt mode bit (INTM): Bit 9 in status register ST0; either enables all maskable interrupts that are not masked by the IMR or disables all maskable interrupts.

interrupt service routine (ISR): A module of code that is executed in response to a hardware or software interrupt.

interrupt trap: See *interrupt service routine (ISR)*.

interrupt vector: A branch instruction that leads the CPU to an interrupt service routine (ISR).

interrupt vector location: An address in program memory where an interrupt vector resides. When an interrupt is acknowledged, the CPU branches to the interrupt vector location and fetches the interrupt vector.

INTM bit: See *interrupt mode bit (INTM)*.

I/O-mapped register: One of the on-chip registers mapped to addresses in I/O (input/output) space. These registers, which include the registers for the on-chip peripherals, must be accessed with the IN and OUT instructions. See also *memory-mapped register*.

IR: See *instruction register (IR).*

$\overline{\textbf{IS}}$: *I/O space select pin*. The ’24x asserts $\overline{\text{IS}}$ to indicate an access to external I/O space.

ISR: See *interrupt service routine (ISR)*.

ISWS: *I/O-space wait-state bit(s).* A value in the wait-state generator control register (WSGR) that determines the number of wait states applied to reads from and writes to off-chip I/O space.

L

latch phase: The phase of a CPU cycle during which internal values are held constant. See also *logic phase*; *CLKOUT1*.

logic phase: The phase of a CPU cycle during which internal values are changed. See also *latch phase*; *CLKOUT1*.

long-immediate value: A 16-bit constant given as an operand of an instruction that is using immediate addressing.

LSB: *Least significant bit.* The lowest order bit in a word. When used in plural form (LSBs), refers to a specified number of low-order bits, beginning with the lowest order bit and counting to the left. For example, the four LSBs of a 16-bit value are bits 0 through 3. See also *MSB*.

M

machine cycle: See *CPU cycle*.

maskable interrupt: A hardware interrupt that can be enabled or disabled through software. See also *nonmaskable interrupt*.

master clock output signal: See *CLKOUT1*.

master phase: See *logic phase*.

memory-mapped register: One of the on-chip registers mapped to addresses in data memory. See also *I/O-mapped register*.

microcontroller mode: A mode in which the on-chip ROM or flash memory in program memory space is enabled. This mode is selected with the MP/$\overline{\text{MC}}$ pin.

microprocessor mode: A mode in which the on-chip ROM or flash memory is disabled and external program memory is enabled. This mode is selected with the MP/$\overline{\text{MC}}$ pin.

microstack (MSTACK): A register used for temporary storage of the program counter (PC) value when an instruction needs to use the PC to address a second operand.

MIPS: Million instructions per second.

MP/$\overline{\text{MC}}$ pin: A pin that indicates whether the processor is operating in microprocessor mode or microcontroller mode. MP/$\overline{\text{MC}}$ high selects microprocessor mode; MP/$\overline{\text{MC}}$ low selects microcontroller mode. This pin is used to execute the on-chip bootloader/user code at reset. When MP/$\overline{\text{MC}}$ is held low during reset, program control transfers to on-chip non-volatile memory at location 0000h. When MP/$\overline{\text{MC}}$ is held high, control transfers to 0000h in external program memory.

MSB: *Most significant bit.* The highest order bit in a word. When used in plural form (MSBs), refers to a specified number of high-order bits, beginning with the highest order bit and counting to the right. For example, the eight MSBs of a 16-bit value are bits 15 through 8. See also *LSB*.

MSTACK: See *microstack*.

multiplier: A part of the CPU that performs 16-bit × 16-bit multiplication and generates a 32-bit product. The multiplier operates using either signed or unsigned 2s-complement arithmetic.

N

next AR: See *next auxiliary register*.

next auxiliary register: The register that is pointed to by the auxiliary register pointer (ARP) when an instruction that modifies ARP is finished executing. See also *auxiliary register*; *current auxiliary register*.

nonmaskable interrupt: An interrupt that can be neither masked by the interrupt mask register (IMR) nor disabled by the INTM bit of status register ST0.

NPAR: *Next program address register.* Part of the program-address generation logic. This register provides the address of the next instruction to the program counter (PC), the program address register (PAR), the micro stack (MSTACK), or the stack.

O

operand: A value to be used or manipulated by an instruction; specified in the instruction.

operand-fetch phase: The third phase of the pipeline; the phase in which an operand or operands are fetched from memory. See also *pipeline*; *instruction-fetch phase*; *instruction-decode phase; instruction-execute phase*.

output shifter: 32- to 16-bit barrel left shifter. Shifts the 32-bit accumulator output from 0 to 7 bits left for quantization management, and outputs either the 16-bit high or low half of the shifted 32-bit data to the data write bus (DWEB).

OV bit: *Overflow flag bit.* Bit 12 of status register ST0; indicates whether the result of an arithmetic operation has exceeded the capacity of the accumulator.

overflow (in a register): A condition in which the result of an arithmetic operation exceeds the capacity of the register used to hold that result.

overflow mode: The mode in which an overflow in the accumulator causes the accumulator to be loaded with a preset value. If the overflow is in the positive direction, the accumulator is loaded with its most positive number. If the overflow is in the negative direction, the accumulator is filled with its most negative number.

OVM bit: *Overflow mode bit.* Bit 11 of status register ST0; enables or disables overflow mode. See also *overflow mode*.

P

PAB: See *program address bus (PAB).*

PAR: *Program address register.* A register that holds the address currently being driven on the program address bus for as many cycles as it takes to complete all memory operations scheduled for the current machine cycle.

PC: See *program counter (PC).*

PCB: *Printed circuit board*.

pending interrupt: A maskable interrupt that has been successfully requested but is awaiting acknowledgement by the CPU.

pipeline: A method of executing instructions in an assembly line fashion. The ’24x pipeline has four independent phases. During a given CPU cycle, four different instructions can be active, each at a different stage of completion. See also *instruction-fetch phase*; *instruction-decode phase*; *operand-fetch phase; instruction-execute phase*.

PLL: Phase lock loop circuit.

PM bits: See *product shift mode bits (PM).*

power-down mode: The mode in which the processor enters a dormant state and dissipates considerably less power than during normal operation. This mode is initiated by the execution of an IDLE instruction. During a power-down mode, all internal contents are maintained so that operation continues unaltered when the power-down mode is terminated. The contents of all on-chip RAM also remains unchanged.

PRDB: See *program read bus (PRDB).*

PREG: See *product register (PREG).*

product register (PREG): A 32-bit register that holds the results of a multiply operation.

product shifter: A 32-bit shifter that performs a 0-, 1-, or 4-bit left shift, or a 6-bit right shift of the multiplier product based on the value of the product shift mode bits (PM).

product shift mode: One of four modes (no-shift, shift-left-by-one, shift-left-by-four, or shift-right-by-six) used by the product shifter.

product shift mode bits (PM): Bits 0 and 1 of status register ST1; they identify which of four shift modes (no-shift, left-shift-by-one, left-shift-by-four, or right-shift-by-six) will be used by the product shifter.

program address bus (PAB): A 16-bit internal bus that provides the addresses for program-memory reads and writes.

program-address generation logic: Logic circuitry that generates the addresses for program memory reads and writes, and an operand address in instructions that require two registers to address operands. This circuitry can generate one address per machine cycle. See also *data-address generation logic.*

program control logic: Logic circuitry that decodes instructions, manages the pipeline, stores status of operations, and decodes conditional operations.

program counter (PC): A register that indicates the location of the next instruction to be executed.

program read bus (PRDB): A 16-bit internal bus that carries instruction code and immediate operands, as well as table information, from program memory to the CPU.

$\overline{\text{PS}}$: *Program select pin*. The ’24x asserts $\overline{\text{PS}}$ to indicate an access to external program memory.

PSLWS: *Lower program-space wait-state bits.* A value in the wait-state generator control register (WSGR) that determines the number of wait states applied to reads from and writes to off-chip lower program space (addresses 0000h–7FFFh). See also *PSUWS*.

PSUWS: *Upper program-space wait-state bits.* A value in the wait-state generator control register (WSGR) that determines the number of wait states applied to reads from and writes to off-chip upper program space (addresses 8000h–FFFFh). See also *PSLWS*.

R

$\overline{\text{RD}}$: *Read select pin*. The ’24x asserts $\overline{\text{RD}}$ to request a read from external program, data, or I/O space. $\overline{\text{RD}}$ can be connected directly to the output enable pin of an external device.

READY: *External device ready pin*. Used to create wait states externally. When this pin is driven low, the ’24x waits one CPU cycle and then tests READY again. After READY is driven low, the ’24x does not continue processing until READY is driven high.

repeat counter (RPTC): A 16-bit register that counts the number of times a single instruction is repeated. RPTC is loaded by an RPT instruction.

reset: A way to bring the processor to a known state by setting the registers and control bits to predetermined values and signaling execution to start at address 0000h.

reset pin ($\overline{\text{RS}}$): A pin that causes a reset.

reset vector: The interrupt vector for reset.

return address: The address of the instruction to be executed when the CPU returns from a subroutine or interrupt service routine.

RPTC: See *repeat counter (RPTC).*

$\overline{\text{RS}}$: *Reset pin*. When driven low, causes a reset on any ’24x device.

R/$\overline{\text{W}}$: *Read/write pin.* Indicates the direction of transfer between the ’24x and external program, data, or I/O space.

S

scratch-pad RAM: Another name for DARAM block B2 in data space (32 words).

short-immediate value: An 8-, 9-, or 13-bit constant given as an operand of an instruction that is using immediate addressing.

sign bit: The MSB of a value when it is seen by the CPU to indicate the sign (negative or positive) of the value.

sign extend: Fill the unused high order bits of a register with copies of the sign bit in that register.

sign-extension mode (SXM) bit: Bit 10 of status register ST1; enables or disables sign extension in the input shifter. It also differentiates between logic and arithmetic shifts of the accumulator.

slave phase: See *latch phase*.

software interrupt: An interrupt caused by the execution of an INTR, NMI, or TRAP instruction.

software stack: A program control feature that allows you to extend the hardware stack into data memory with the PSHD and POPD instructions. The stack can be directly stored and recovered from data memory, one word at time. This feature is useful for deep subroutine nesting or protection against stack overflow.

ST0 and ST1: See *status registers ST0 and ST1*.

stack: A block of memory reserved for storing return addresses for subroutines and interrupt service routines. The ’24x stack is 16 bits wide and eight levels deep.

status registers ST0 and ST1: Two 16-bit registers that contain bits for determining processor modes, addressing pointer values, and indicating various processor conditions and arithmetic logic results. These registers can be stored into and loaded from data memory, allowing the status of the machine to be saved and restored for subroutines.

$\overline{\textbf{STRB}}$: *External access active strobe*. The ’24x asserts $\overline{\text{STRB}}$ during accesses to external program, data, or I/O space.

SXM bit: See *sign-extension mode bit (SXM).*

T

TC bit: *Test/control flag bit*. Bit 11 of status register ST1; stores the results of test operations done in the central arithmetic logic unit (CALU) or the auxiliary register arithmetic unit (ARAU). The TC bit can be tested by conditional instructions.

temporary register (TREG): A 16-bit register that holds one of the operands for a multiply operation; the dynamic shift count for the LACT, ADDT, and SUBT instructions; or the dynamic bit position for the BITT instruction.

TOS: *Top of stack.* Top level of the 8-level last-in, first-out hardware stack.

TREG: See *temporary register (TREG).*

TTL: *Transistor-to-transistor logic*.

V

vector: See *interrupt vector*.

vector location: See *interrupt vector location*.

W

wait state: A CLKOUT cycle during which the CPU waits when reading from or writing to slower external memory.

wait-state generator: An on-chip peripheral that generates a limited number of wait states for a given off-chip memory space (program, data, or I/O). Wait states are set in the wait-state generator control register (WSGR).

$\overline{\textbf{WE}}$: *Write enable pin*. The ’24x asserts $\overline{\text{WE}}$ to request a write to external program, data, or I/O space.

WSGR: *Wait-state generator control register.* This register, which is mapped to I/O memory, controls the wait-state generator.

X

XF bit: *XF-pin status bit.* Bit 4 of status register ST1 that is used to read or change the logic level on the XF pin.

XF pin: *External flag pin*. A general-purpose output pin whose status can be read or changed by way of the XF bit in status register ST1.

XINT1–XINT2: External pins used to generate general-purpose hardware interrupts.

Z

zero fill: A way to fill the unused low or high order bits in a register by inserting 0s.

Index