TMS320F243/F241/C242 DSP Controllers Reference Guide

System and Peripherals

Literature Number: SPRU276C
January 2000

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 TMS320F243/F241/C242 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 is intended to 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.

This is an example of a caution statement.

A caution statement describes a situation that could potentially damage your software or equipment.

CAUTION

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.

TMS320C242 DSP Controller (literature number SPRS063) data sheet contains the electrical and timing specifications for this device, as well as signal descriptions and pinouts for all of the available packages.

TMS320F243/F241 DSP Controllers (literature number SPRS064) 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

Contents

Figures

Tables

Chapter 1

Introduction

The TMS320F243/F241/C242 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 three ’24x devices: ’F243, ’F241, and ’C242. 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 ’24x DSP controller products, and introduces the ’F243, ’F241, and ’C242 devices.

Topic **Page**

1.1 TMS320 Family Overview

The TMS320 family consists of fixed- and 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 ’24x 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 ’F243/F241/C242 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 TMS320C24x 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 ’24x 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 ’24x series yields a device that is an affordable alternative to traditional microcontroller units (MCUs) and expensive multichip designs. At 20 million instructions per second (MIPS), the ’24x DSP controllers offer significant performance over traditional 16-bit microcontrollers and microprocessors.

The 16-bit fixed-point DSP core of the ’24x 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 ad-

vanced control algorithms for techniques such as adaptive control, Kalman filtering, and state control. The ’24x DSP controllers offer reliability and programmability. In contrast, analog control systems 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 ’24x 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 ’24x 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 it 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 ’24x 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 ’F243/F241/C242 devices includes:

- Event Manager: Timers and PWM generators for digital motor control
- CAN Interface: Controller Area Network (CAN) 2.0b compatible, with 6 mailboxes (not available in ’C242)
- A/D: 10-bit ±1, 1-µs conversion, 8 channel, analog-to-digital converter
- SPI: Serial Peripheral Interface – synchronous serial port (not available in ’C242)
- SCI: Serial Communications Interface – asynchronous serial port (universal asynchronous receiver and transmitter – UART)
- Watchdog timer, without real timer interrupt (RTI) capability
- General purpose bi-directional digital I/O (GPIO) pins

Note:

For device pinouts, electrical characteristics, and timing specifications of ’F243/F241/C242 devices, refer to the following datasheets:

- *TMS320F243, TMS320F241 DSP Controllers* (SPRS064)
- *TMS320C242 DSP Controllers* (SPRS063)

1.4 'F243/F241/C242 Highlights

- The 'F243 has an external memory interface and is intended primarily for emulation tools.
- The 'F241 is similar to the 'F243 but lacks external memory interface and has reduced GPIO pins.
- The 'C242 is a minimum-cost motor control device.

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

Table 1–1. 'F243/F241/C242 Device Configurations

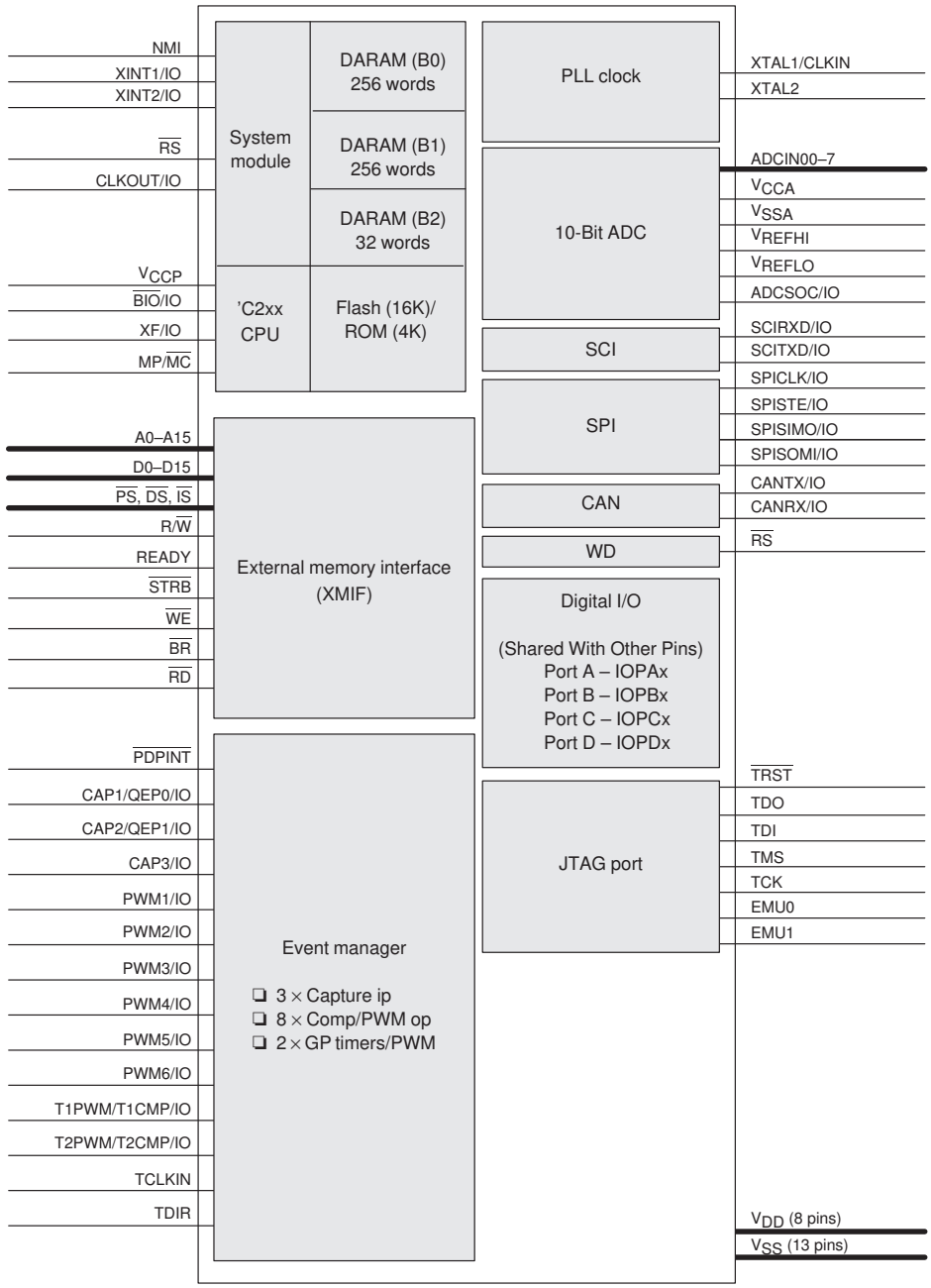
Device Feature	'F243	'F241	'C242
'C2xLP CPU with 544 words DARAM + JTAG	Yes	Yes	Yes
5V Flash	8k x 16	8k x 16	–
ROM	–	–	4k x 16
Event Manager (EV2)	Yes	Yes	Yes
CAN	Yes	Yes	No
SPI	Yes	Yes	No
SCI	Yes	Yes	Yes
10 bit, 8 channel A/D with dual conversion	Yes	Yes	Yes
WD (no RTI)	Yes	Yes	Yes
General Purpose Digital I/O	26 – Shared with other functions 6 dedicated to I/O†	26 – Shared with other functions	20 – Shared with other functions 6 dedicated to I/O†
External Interrupts	$\overline{\text{PDPINT}}$, $\overline{\text{NMI}}$, XINT1, XINT2	$\overline{\text{PDPINT}}$, $\overline{\text{NMI}}$, XINT1, XINT2	$\overline{\text{PDPINT}}$, $\overline{\text{NMI}}$, XINT1, XINT2
External memory interface	Yes	No	No
Package	144 TQFP	64 pin QFP or 68 pin PLCC	64 pin QFP or 68 pin PLCC 'F241 compatible

†These six dedicated I/O pins are not the same pins for 'F243 and 'C242.

Note: The PMT pin should be connected to ground for proper operation.

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

Figure 1–1. 'F243/F241/C242 Device Overview



Notes: 1) CAN and SPI are not available in 'C242.
2) XMIF is not available in 'F241 and 'C242.

1.5 'F243/F241/C242 Features

Some additional features of the 'X243/241/242 devices include:

- All peripheral registers have at least a subset of the functionality of the 'C240 peripherals and are address and bit-position compatible with the TMS320C240.
- Many peripherals have reduced functionality relative to the 'F/C240.
- SPI functionality has been increased to allow up to 16-bit characters with double buffering on transmit.
- All peripherals are accessed via the peripheral bus.
- All peripherals are clocked at the same rate as the CPU.
- The analog-to-digital converter has a 10-bit ±1 LSB accuracy, 1μs conversion time.
- 64-pin QFP or 68-pin PLCC package (for 'F241/C242).
- 144-pin QFP package (for 'F243).
- Compared to the 'C240, the following functions are absent:
  - Software Programmable PLL
  - Power-On Reset ($\overline{\text{PORESET}}$)

Chapter 2

Peripheral Interrupt Expansion (PIE)

This chapter describes how the peripheral interrupt expansion (PIE) is used to increase interrupt request capacity.

Topic **Page**

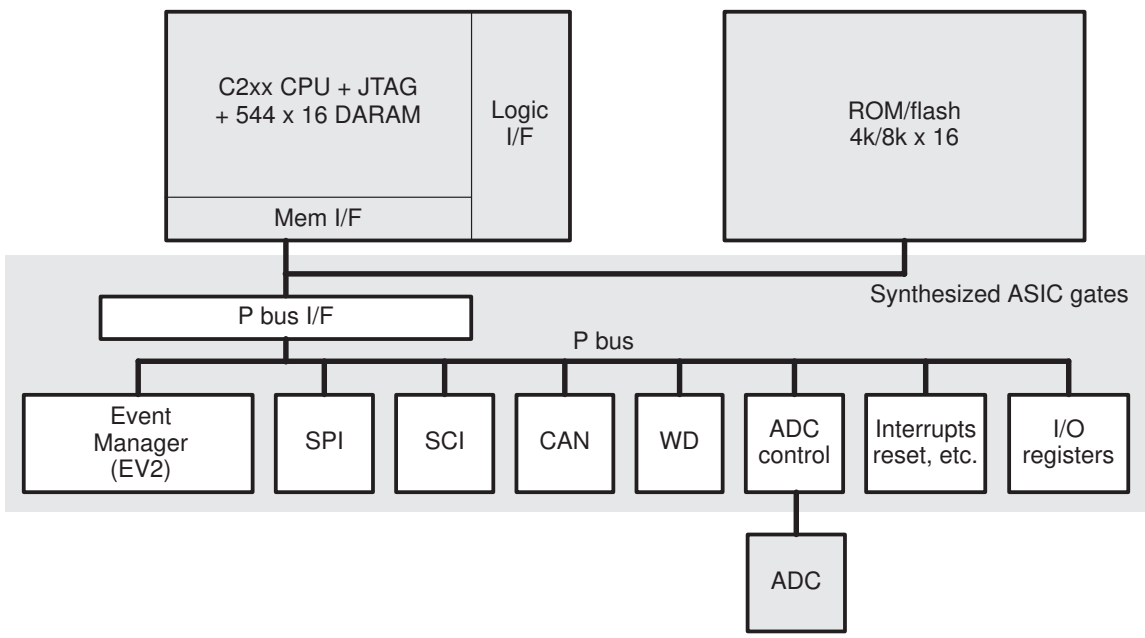
2.1 Architecture Summary

The ’F243/F241/C242 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 ’F243/F241/C242.

Peripherals interface to the internal memory of the CPU through the peripheral bus (PBUS) interface. All on-chip peripherals are accessed through 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 850ns A/D converter.

These devices have 26 (32 for ’F243) 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, and 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. ’F243/F241/C242 Device Architecture



2.2 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 2–1 provides the interrupt source priority and vectors for the ’F243/F241/C242 devices.

Table 2–1. ’F243/F241/C242 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	Non-maskable interrupt	Non-maskable interrupt
(a) **INT1** *(level 1)*						
Overall Priority	**Interrupt Name**	**CPU Interrupt Vector**	**Peripheral Interrupt Vector**	**Maskable?**	**Source Peripheral**	**Description**
4	$\overline{\text{PDPINT}}$	INT1 0002h	0020h	Y	EV	Power Drive Protection Interrupt Pin
5	ADCINT		0004h	Y	ADC	ADC interrupt in high priority mode
6	XINT1		0001h	Y	External interrupt logic	External interrupt pins in high priority mode
7	XINT2		0011h	Y	External interrupt logic	External interrupt pins in high priority mode
8	SPIINT		0005h	Y	SPI	SPI interrupt in high priority mode
9	RXINT		0006h	Y	SCI	SCI receiver interrupt in high priority mode
10	TXINT		0007h	Y	SCI	SCI transmitter interrupt in high priority mode
11	CANMBINT		0040h	Y	CAN	CAN mailbox interrupt (high priority mode)
12	CANERINT		0041h	Y	CAN	CAN error interrupt (high priority mode)

Table 2–1. 'F243/F241/C242 Interrupt Source Priority and Vectors (Continued)

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

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
13	CMP1INT	INT2 0004h	0021h	Y	EV	Compare 1 interrupt
14	CMP2INT		0022h	Y	EV	Compare 2 interrupt
15	CMP3INT		0023h	Y	EV	Compare 3 interrupt
16	TPINT1		0027h	Y	EV	Timer 1 period interrupt
17	TCINT1		0028h	Y	EV	Timer 1 compare interrupt
18	TUFINT1		0029h	Y	EV	Timer 1 underflow interrupt
19	TOFINT1		002Ah	Y	EV	Timer 1 overflow interrupt

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

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
20	TPINT2	INT3 0006h	002Bh	Y	EV	Timer 2 period interrupt
21	TCINT2		002Ch	Y	EV	Timer 2 compare interrupt
22	TUFINT2		002Dh	Y	EV	Timer 2 underflow interrupt
23	TOFINT2		002Eh	Y	EV	Timer 2 overflow interrupt

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

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
24	CAPINT1	INT4	0033h	Y	EV	Capture 1 interrupt
25	CAPINT2	0008h	0034h	Y	EV	Capture 2 interrupt
26	CAPINT3		0035h	Y	EV	Capture 3 interrupt

Table 2–1. 'F243/F241/C242 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		0006h	Y	SCI	SCI receiver interrupt (low priority mode)
29	TXINT		0007h	Y	SCI	SCI transmitter interrupt (low priority mode)
30	CANMBINT		0040h	Y	CAN	CAN mailbox interrupt (low priority mode)
31	CANERINT		0041h	Y	CAN	CAN error interrupt (low priority mode)

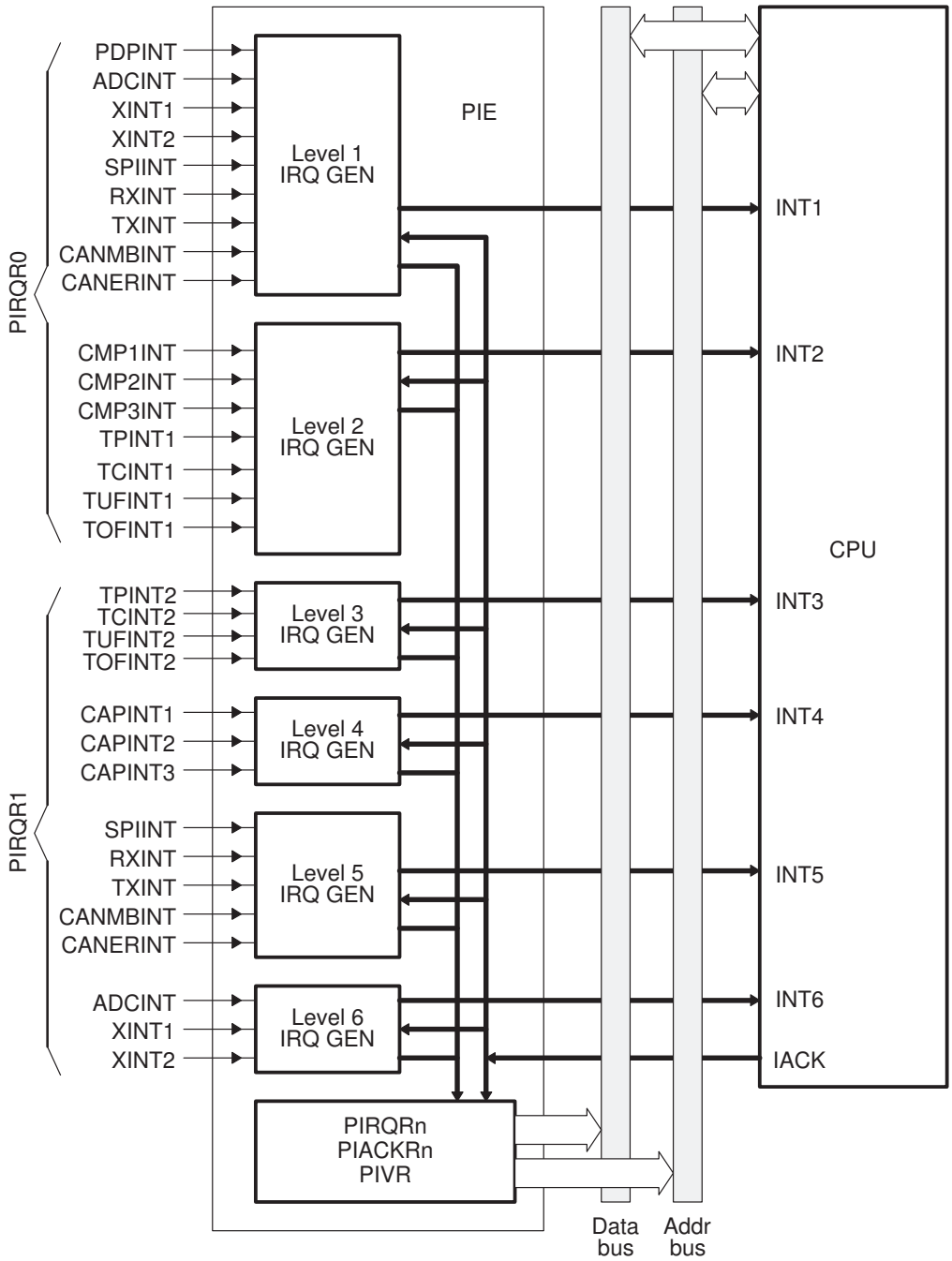
(f) **INT6** *(level 6)*

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		0001h	Y	External interrupt logic	External interrupt pins (low priority mode)
34	XINT2		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

2.3 Peripheral Interrupt Expansion (PIE)

The ’24x CPU supports one nonmaskable interrupt (NMI) and six maskable prioritized interrupt requests. The ’24x devices have many peripherals, and each peripheral is capable of generating one or more interrupts in response to many events. Because the ’C24x CPU does not have sufficient capacity to handle all peripheral interrupt requests, a centralized interrupt controller (PIE) is required to arbitrate the interrupt requests from all the different sources (see Figure 2–2).

Figure 2–2. Peripheral Interrupt Expansion Block Diagram



2.3.1 Interrupt Hierarchy

The number of interrupt requests 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.

2.3.2 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 a request to the CPU. There is an interrupt flag bit and an interrupt enable bit located in the peripheral 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 requests (INTn) to the CPU. The request to the 'C24x 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 2–1, *'X243/241/242 Interrupt Source Priority and Vectors*, on page 2-3.

2.3.3 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 2–1, *'X243/241/242 Interrupt Source Priority and Vectors*, on page 2-3. 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.

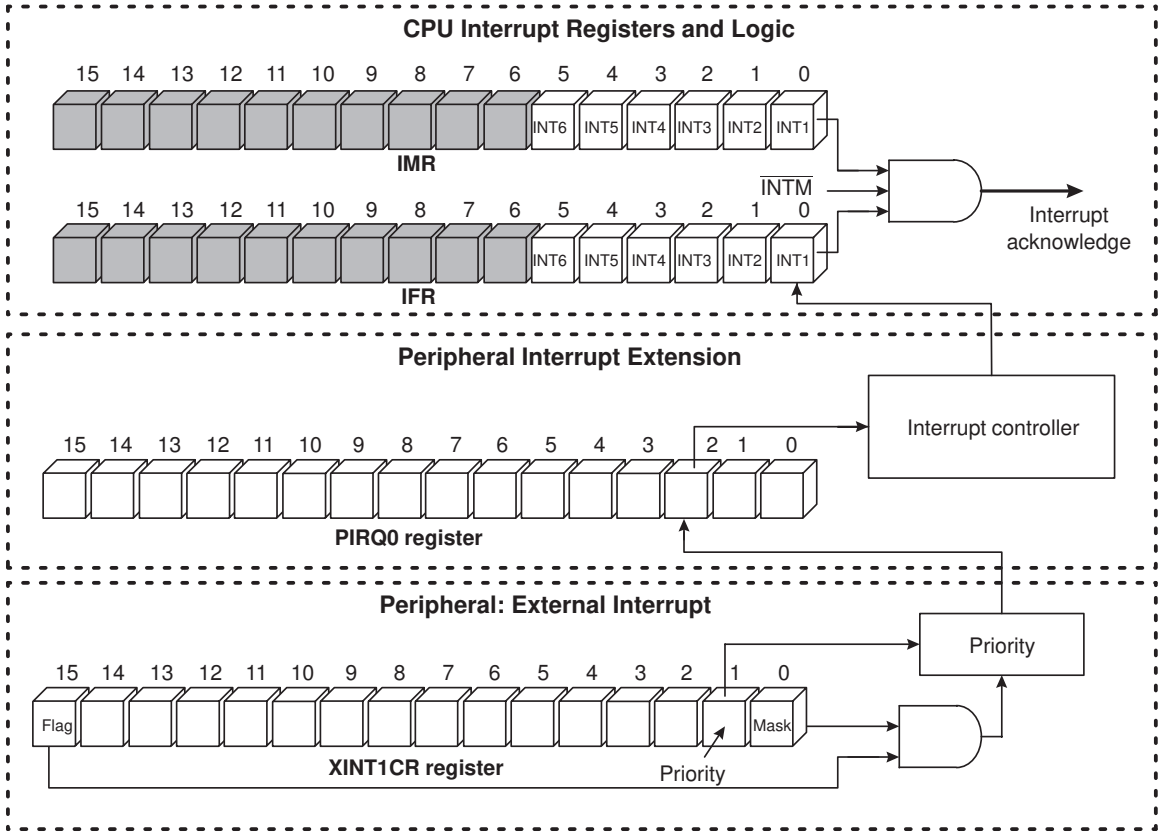
2.4 Interrupt Vectors

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 2–3 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, 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 2–3. Interrupt Requests



2.4.1 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.

2.4.2 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.

2.4.3 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 re-enabled. (All interrupts are automatically disabled when an interrupt is taken.) If the PIVR is not read before interrupts are re-enabled, another interrupt is acknowledged and a new peripheral interrupt vector is loaded into the PIVR, causing permanent loss of the original peripheral interrupt vector.

2.5 Interrupt Operation Sequence

An interrupt generating event occurs in a peripheral. Refer to Figure 2–4 for ’24x interrupt response and flow in each module of the ’24x.The interrupt flag bit (IF) corresponding to that event is set in a register in the peripheral. If the corresponding interrupt enable bit (IE) 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 2–1 *’X243/241/242 Interrupt Source Priority and Vectors,* on page 2-3).

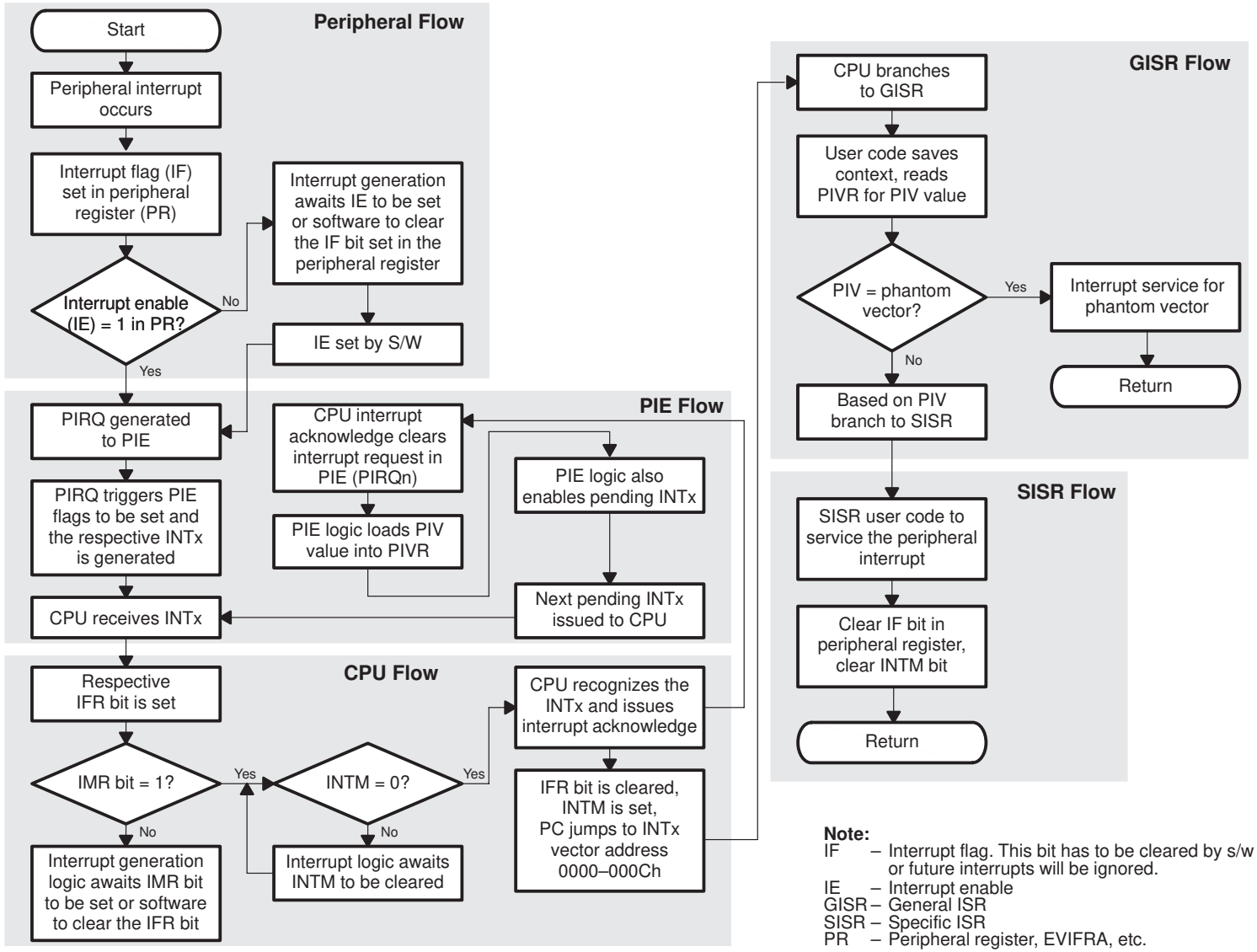
The PIE controller decodes the PAB value and generates a peripheral interrupt acknowledge to clear the PIRQ bit associated with the CPU interrupt being acknowledged. The PIE controller then loads the peripheral interrupt vector register (PIVR) with the appropriate peripheral interrupt vector (or the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

Re-enabling interrupts

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

Figure 2–4. ’24x Interrupt Response and Flow



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.

2.6 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 re-enables interrupt. The latency can be longer than the minimum if the interrupt request occurs during an uninterruptible operation, for example, 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.

2.7 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
```

2.8 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)

2.8.1 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 ’24x 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 2–5. 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 zero, R = Read access, W1C = Write 1 to this bit to clear it, value following dash (–) = value after reset

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

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

2.8.2 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 2–6, and descriptions of the bits follow the figure.

Figure 2–6. 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 zero, 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 zeros.

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.

2.9 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 acknowledge register 0 (PIACKR0)
- ❑ The peripheral interrupt acknowledge register 1 (PIACKR1)

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

2.9.1 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 2–7.

Figure 2–7. 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; value following dash (–) = value after reset

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

2.9.2 Peripheral Interrupt Request Registers (PIRQR0 and PIRQR1)

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 2–8 and PIRQR1 is shown in Figure 2–9.

Figure 2–8. 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, value following dash (–) = value after reset

Bits 15–0 **IRQ0.15–IRQ0.0**. Peripheral interrupt request bits. See Table 2–1, *'F243/F241/C242 Interrupt Source Priority and Vectors*, on page 2-3 to see which peripheral interrupt corresponds to each register bit.

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 2–2. Peripheral Interrupt Request Descriptions (PIRQR0)

Bit position	Interrupt	Interrupt Description	Interrupt Level
IRQ 0.0	PDPINT	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

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

Bit position	Interrupt	Interrupt Description	Interrupt Level
IRQ 0.8	CANERINT	CAN error interrupt. High priority	INT1
IRQ 0.9	CMP1INT	Compare 1 interrupt	INT2
IRQ 0.10	CMP2INT	Compare 2 interrupt	INT2
IRQ 0.11	CMP3INT	Compare 3 interrupt	INT2
IRQ 0.12	TPINT1	Timer 1 period interrupt	INT2
IRQ 0.13	TCINT1	Timer 1 compare interrupt	INT2
IRQ 0.14	TUFINT1	Timer 1 underflow interrupt	INT2
IRQ 0.15	TOFINT1	Timer 1 overflow interrupt	INT2

Figure 2–9. 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, value following dash (–) = value after reset

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

Bits 14–0 **IRQ1.14–IRQ1.0**. Peripheral interrupt request bits,. See Table 2–1, *'F243/F241/C242 Interrupt Source Priority and Vectors*, on page 2-3, to see which peripheral interrupt corresponds to each register bit.

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 (PIRQR1)

Bit position	Interrupt	Interrupt Description	Interrupt Level
IRQ 1.0	TPINT2	Timer 2 period interrupt	INT3
IRQ 1.1	TCINT2	Timer 2 compare interrupt	INT3
IRQ 1.2	TUFINT2	Timer 2 underflow interrupt	INT3
IRQ 1.3	TOFINT2	Timer 2 overflow interrupt	INT3
IRQ 1.4	CAPINT1	Capture 1 interrupt	INT4
IRQ 1.5	CAPINT2	Capture 2 interrupt	INT4
IRQ 1.6	CAPINT3	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

2.9.3 Peripheral Interrupt Acknowledge Registers (PIACKR0 and PIACKR1)

The peripheral interrupt acknowledge registers (PIACKRx) are memory mapped to enable an easy test of the peripheral interrupt acknowledges. There are two of these 16-bit registers; and therefore, the PIE controller can support up to 32 peripheral interrupts. These registers are generally used for test purposes only and are not for user applications. PIACKR0 is shown in Figure 2–10 and PIACKR1 is shown in Figure 2–11.

Figure 2–10. 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, value following dash (–) = value after reset

Bits 15–0 **IACK0.15–IACK0.0**. Peripheral interrupt acknowledge bits. See Table 2–1 to see which peripheral interrupt corresponds to each register bit. 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 2–4. 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	TPINT1	Timer 1 period interrupt	INT2
IAK 0.13	TCINT1	Timer 1 compare interrupt	INT2
IAK 0.14	TUFINT1	Timer 1 underflow interrupt	INT2
IAK 0.15	TOFINT1	Timer 1 overflow interrupt	INT2

Figure 2–11. 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, value following dash (–) = 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 for PIACKR0.

Table 2–5. Peripheral Interrupt Acknowledge Descriptions (PIACKR1)

Bit position	Interrupt	Interrupt Description	Interrupt Level
IAK 1.0	TPINT2	Timer 2 period interrupt	INT3
IAK 1.1	TCINT2	Timer 2 compare interrupt	INT3
IAK 1.2	TUFINT2	Timer 2 underflow interrupt	INT3
IAK 1.3	TOFINT2	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

Chapter 3

Memory

This chapter describes the RAM, ROM, and Flash availability in the ’F243/F241/C242 devices.

In addition to dual-access RAM (DARAM – B0, B1, B2), which is part of the CPU core, the ’24x devices include flash EPROM or ROM for additional on-chip program memory. The ’C242 is a ROM-only device, while the ’F243/241 are flash-only devices.

The ’F243 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 (32K-word local data space plus a 32K-word local/global data space)
- A 64K-word I/O space

This chapter shows memory maps for program, data, and I/O spaces. It also describes available ’F243 memory configuration options.

Topic **Page**

3.1 Factory Masked On-Chip ROM ('C242 only)

The 'C242 has a maskable 4K × 16 ROM module in program memory space (000h–FFFh). This ROM is always enabled since the 'C242 lacks an external memory interface. This ROM is programmed with customer-specific code.

3.2 Flash

The 'F243 and 'F241 devices have an 8K × 16 on-chip Flash EEPROM memory module. This memory is mapped in the program space (0000h–1FFFh).

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 'F243/'F241, refer to the *TMS320F20x/F24x DSP Embedded Flash Memory Technical Reference* (SPRU282).

3.3 Overview of Memory and I/O Spaces

The ’24x 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 ’F243 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**
  - **Local data memory** (64K words) holds data used by the instructions.
  - **Global data memory** (32K words) shares data with other processors or serves as additional data space.
- **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 ’F243/F241/C242 devices include on-chip memory to aid in system performance and integration.

The advantages of operating from on-chip memory are:

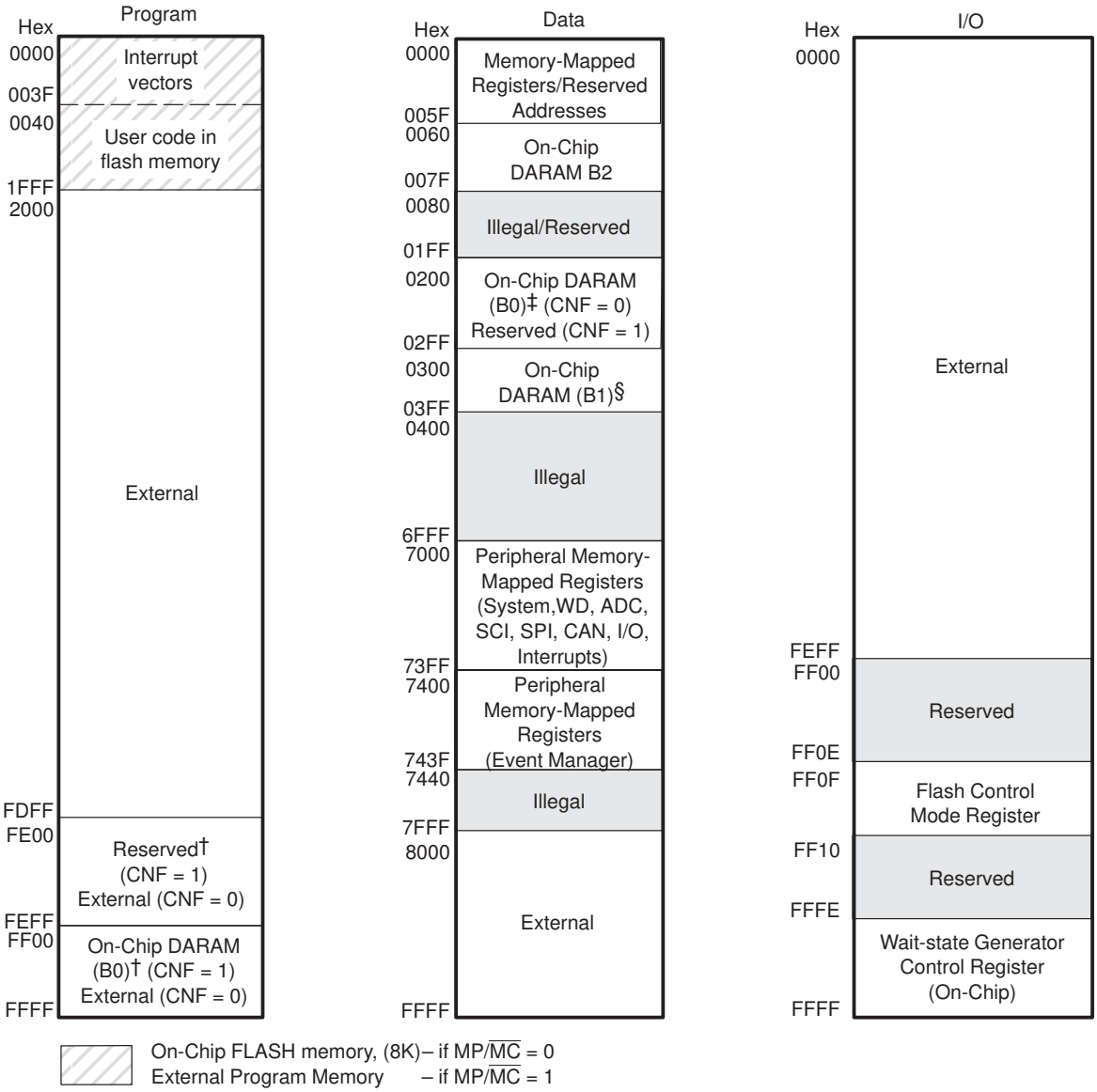
- Higher performance than external memory (because the wait states required for slower external memories are avoided)
- Lower cost than external memory
- Lower power consumption than external memory

The advantage of operating from external memory is the ability to access a larger address space. The ’F241 and ’C242 do not have external memory interfaces and have only on-chip memory. Figure 3–1, Figure 3–2, and Figure 3–3 depict the memory map of ’24x devices.

CAUTION

Access to an illegal address will generate an NMI.

Figure 3–1. Memory Map for ’F243

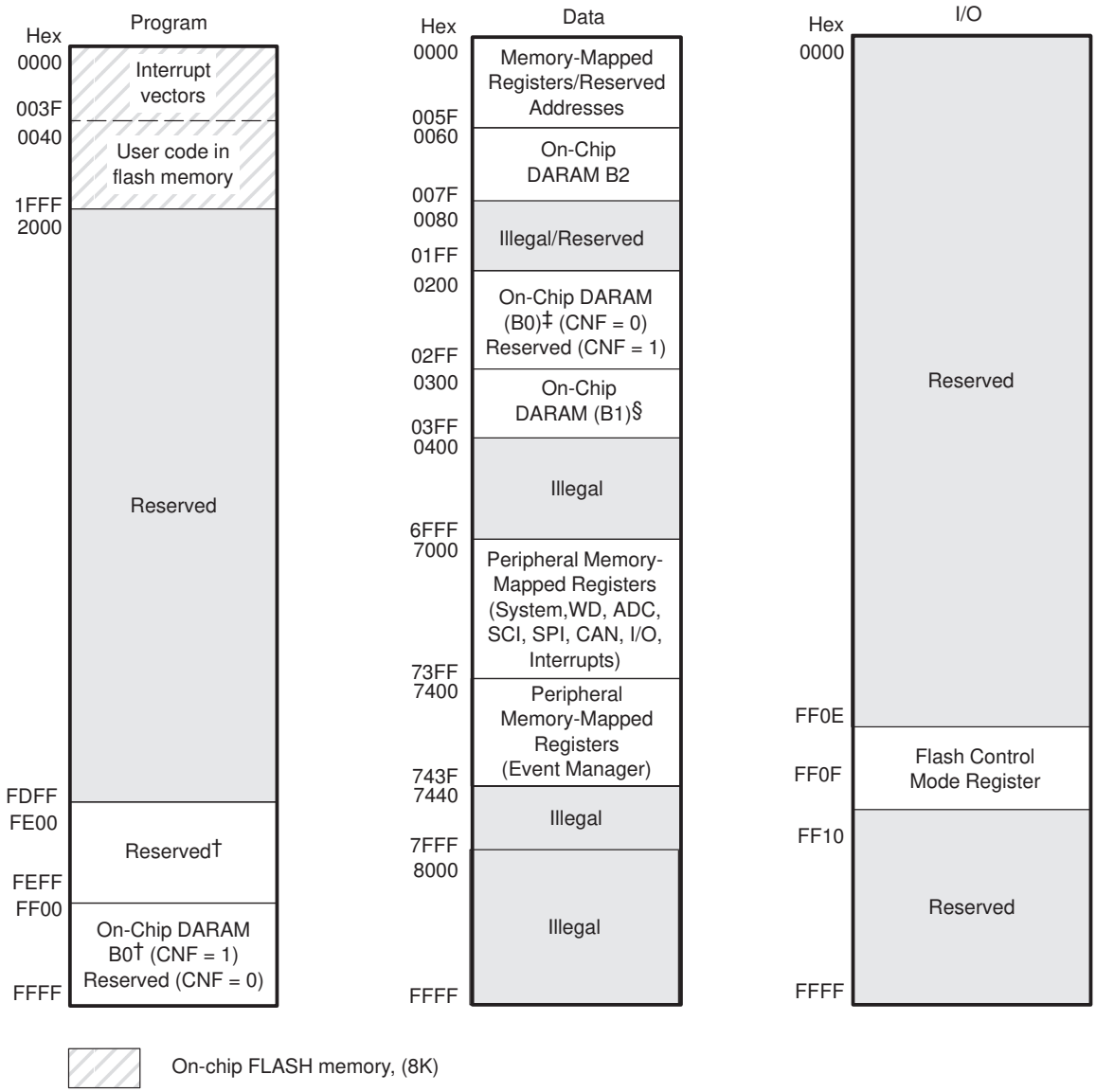


† When CNF = 1, addresses FE00h–FEFFh and FF00h–FFFFh are mapped to the same physical block (B0) in program-memory space. For example, a write to FE00h will have the same effect as a write to FF00h. For simplicity, addresses FE00h–FEFFh are referred to here as reserved when CNF = 1.

‡ 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 will have the same effect as a write to 0200h. For simplicity, addresses 0100h–01FFh are referred to here as reserved. Note that addresses 0080h – 00FFh are illegal.

§ 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 here as illegal.

Figure 3–2. Memory Map for ’F241

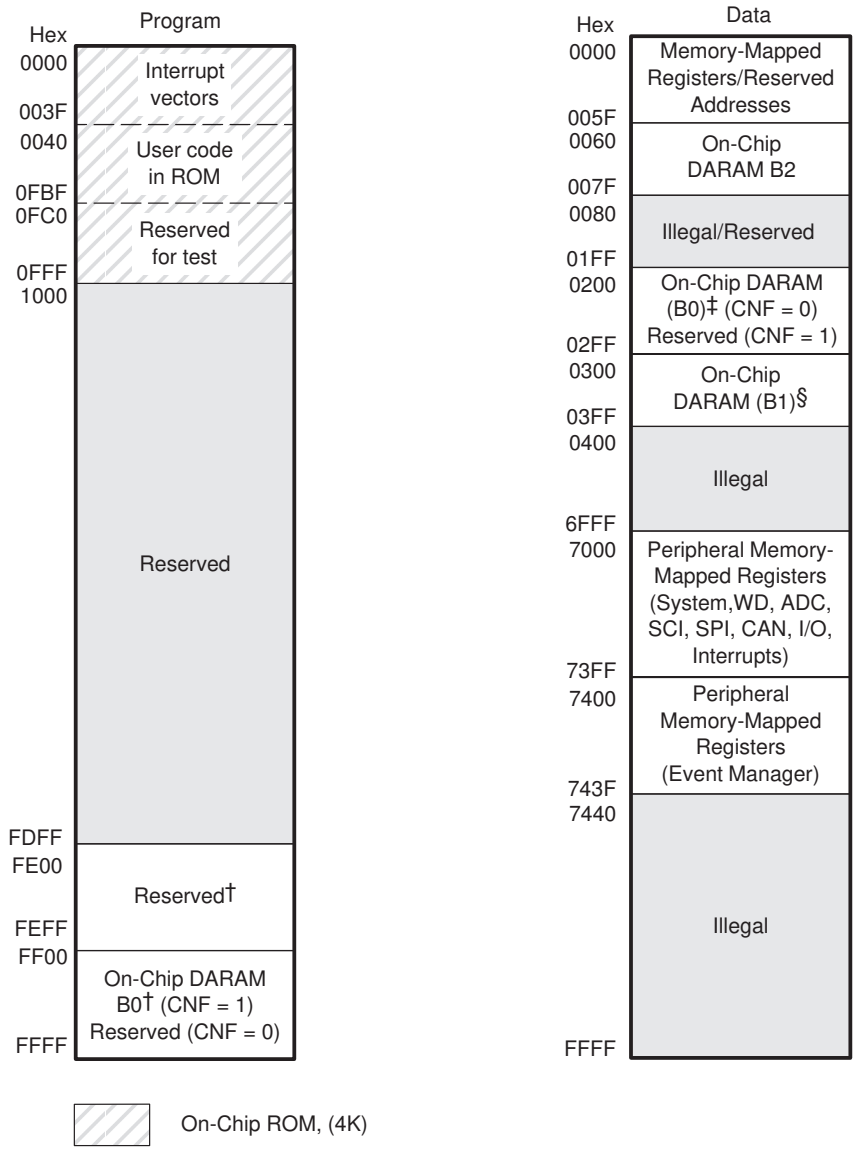


† When CNF = 1, addresses FE00h–FEFFh and FF00h–FFFFh are mapped to the same physical block (B0) in program-memory space. For example, a write to FE00h will have the same effect as a write to FF00h. For simplicity, addresses FE00h–FEFFh are referred to here as reserved when CNF = 1.

‡ 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 will have the same effect as a write to 0200h. For simplicity, addresses 0100h–01FFh are referred to here as reserved. Note that addresses 0080h – through 00FFh are illegal.

§ 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 here as illegal.

Figure 3–3. Memory Map for ’C242



† 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 will have 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 will have the same effect as a write to 0200h. For simplicity, addresses 0100h–01FFh are referred to as reserved. Note that addresses 0080h – through 00FFh are illegal.

§ 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 illegal.

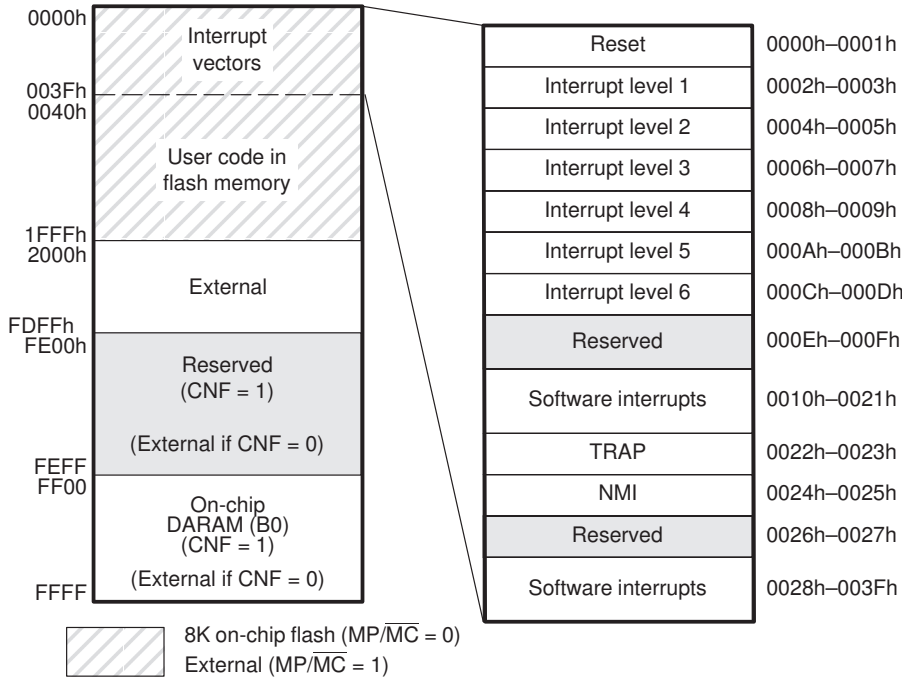
NOTE : There is no external memory space for program, data, or I/O in the ’C242. I/O space is reserved in ’C242. Hence, IN and OUT instructions are not useful in ’C242.

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 ’F243. This number includes on-chip DARAM and flash EEPROM. Whenever an off-chip memory location needs to be accessed, the appropriate control signals for external access ($\overline{PS}$, $\overline{DS}$, $\overline{STRB}$, etc.) are automatically generated.

Figure 3–4 shows the ’F243 program memory map.

Figure 3–4. Program Memory Map for ’F243



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/$\overline{\text{MC}}$ 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/$\overline{\text{MC}}$ = 0.** The device is configured in microcomputer 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–1FFFh will be made to on-chip memory.
  - **MP/$\overline{\text{MC}}$ = 1.** The device is configured in microprocessor mode. The device fetches the reset vector from external memory. Accesses to program memory addresses 0000h–1FFFh will be made to off-chip memory.

  Regardless of the value of MP/$\overline{\text{MC}}$, the ’24x fetches its reset vector at location 0000h in program memory. Note that there is no MP/$\overline{\text{MC}}$ pin available on the ’F241 and ’C242.

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) can be configured as part of either local data memory or global data memory, depending on the value loaded in the global memory allocation register (GREG). Note that addresses 8000h–FFFFh are not accessible in ’F241 and ’C242.

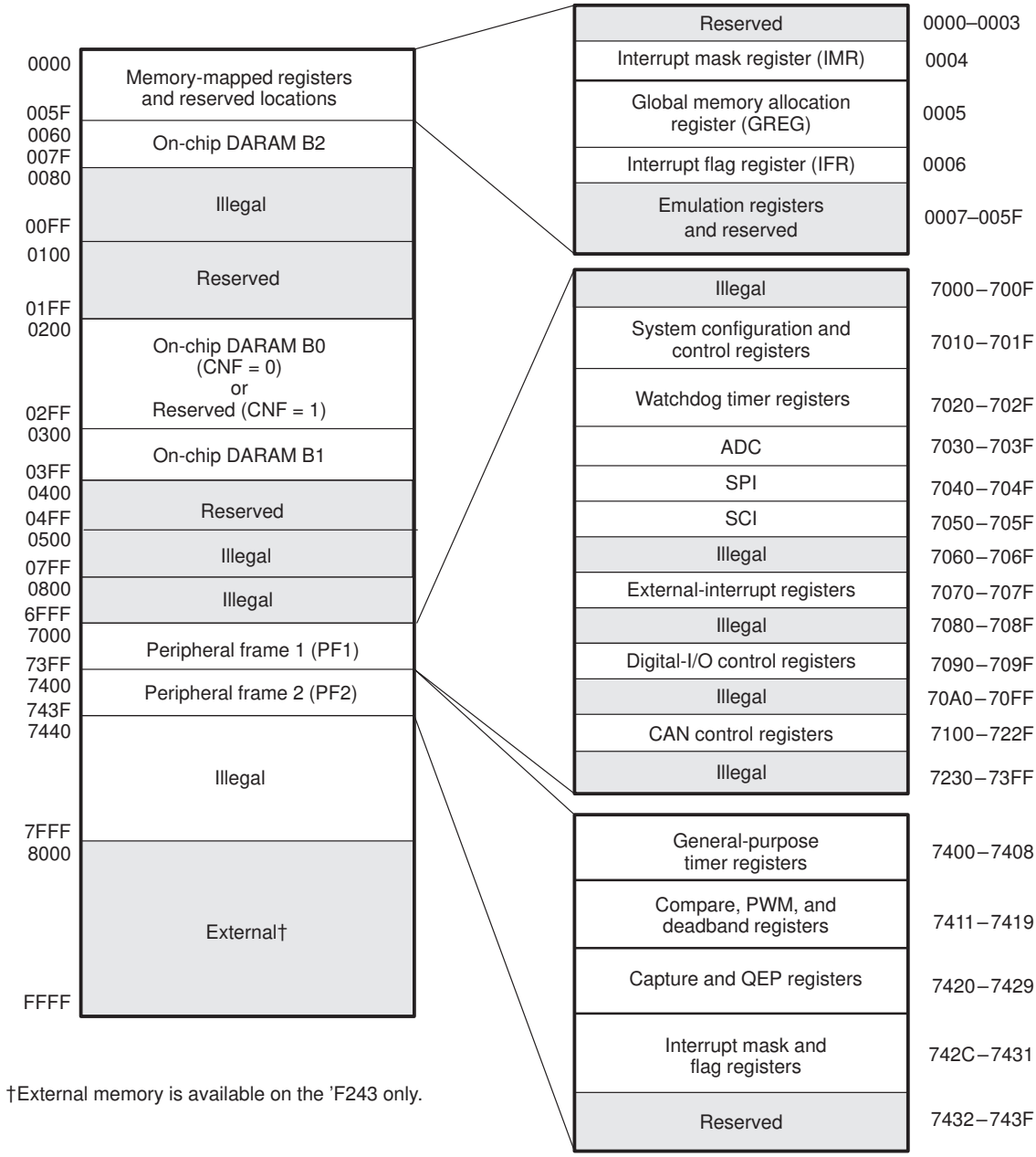
3.5.1 Local Data Memory

Figure 3–5 shows the data memory map for the ’F243/F241/C242. 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 ’F243.

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

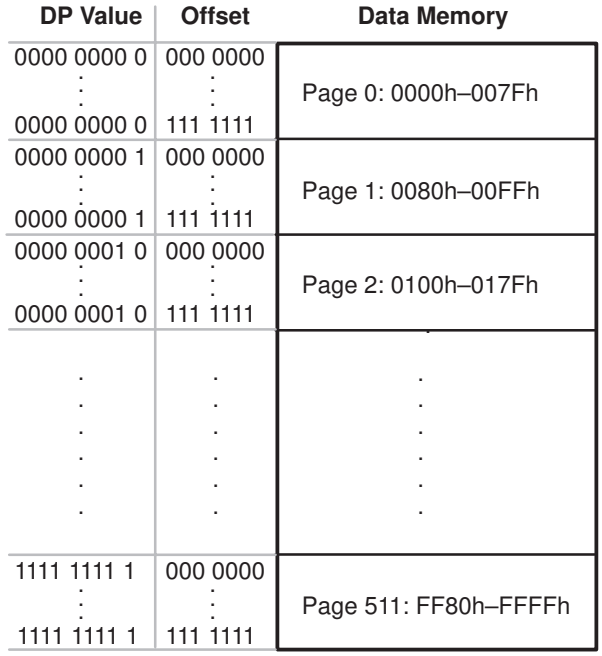
When direct addressing is used, data memory is addressed in blocks of 128 words called data pages. Figure 3–6 shows how these blocks are addressed. The entire 64K of data memory consists of 512 data pages labeled 0 through 511. The current data page is determined by the value in the 9-bit data page pointer (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).

Figure 3–5. ’X24x Peripheral Memory Map



†External memory is available on the ’F243 only.

Figure 3–6. Local Data Memory Pages



Data Page 0 Address Map

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

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	GREG	Global memory allocation register
0006h	IFR	Interrupt flag register
0023h–0027h	–	Reserved
002Bh–002Fh	–	Reserved for test/emulation
0060h–007Fh	B2	Scratch-pad RAM (DARAM B2)

Local Data Memory Configuration

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

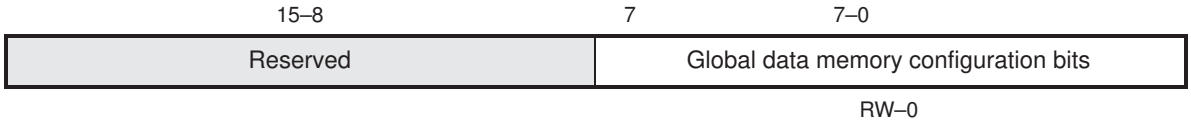
- **CNF bit.** The CNF bit (bit 12) of status register ST1 determines whether the on-chip DARAM B0 is mapped into local data space or program space.
  - **CNF = 1.** B0 is used for program space.
  - **CNF = 0.** B0 is used for data space.

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

3.5.2 Global Data Memory

Addresses in the upper 32K words (8000h–FFFFh) of local data memory can be used for global data memory. The global memory allocation register (GREG) determines the size of the global data-memory space, which is between 256 and 32K words. The GREG is connected to the eight LSBs of the internal data bus and is memory-mapped to data-memory location 0005h.

Figure 3–7. Global Data Memory Configuration Register – Address 0005h



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

Table 3–2 shows the allowable GREG values and shows the corresponding address range set aside for global data memory. Any remaining addresses within 8000h–FFFFh are available for local data memory.

Note:

Choose only the GREG values listed in Table 3–2. Other values lead to fragmented memory maps.

Table 3–2. Global Data Memory Configurations

GREG Value		Local Memory		Global Memory	
High Byte	**Low Byte**	**Range**	**Words**	**Range**	**Words**
XXXX XXXX	0000 0000	0000h–FFFFh	65 536	–	0
XXXX XXXX	1000 0000	0000h–7FFFh	32 768	8000h–FFFFh	32 768
XXXX XXXX	1100 0000	0000h–BFFFh	49 152	C000h–FFFFh	16 384
XXXX XXXX	1110 0000	0000h–DFFFh	57 344	E000h–FFFFh	8 192
XXXX XXXX	1111 0000	0000h–EFFFh	61 440	F000h–FFFFh	4 096
XXXX XXXX	1111 1000	0000h–F7FFh	63 488	F800h–FFFFh	2 048
XXXX XXXX	1111 1100	0000h–FBFFh	64 512	FC00h–FFFFh	1 024
XXXX XXXX	1111 1110	0000h–FDFFh	65 024	FE00h–FFFFh	512
XXXX XXXX	1111 1111	0000h–FEFFh	65 280	FF00h–FFFFh	256

Note: X = Don't care

When a program accesses any external data address, the 'F243 drives the $\overline{DS}$ signal low. If that address is within the range defined by the GREG as a global address, $\overline{BR}$ signal is also asserted. Because $\overline{BR}$ differentiates local and global accesses, the addresses configured by the GREG value are an additional data space. The external data-address range is extended by the selected amount of global space (up to 32K words).

As an example of configuring global memory, suppose you want to designate 8K data-memory addresses as global addresses. To do this, you write the 8-bit value 11100000 to the GREG (see Figure 3–8). This designates addresses E000h–FFFFh in data memory as global data addresses (see Figure 3–9).

Figure 3–8. GREG Register Set to Configure 8K for Global Data Memory

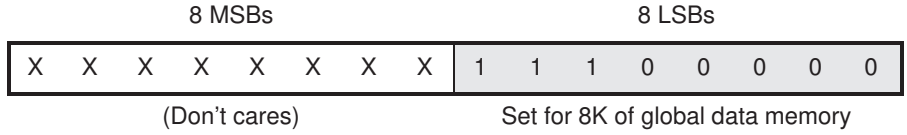
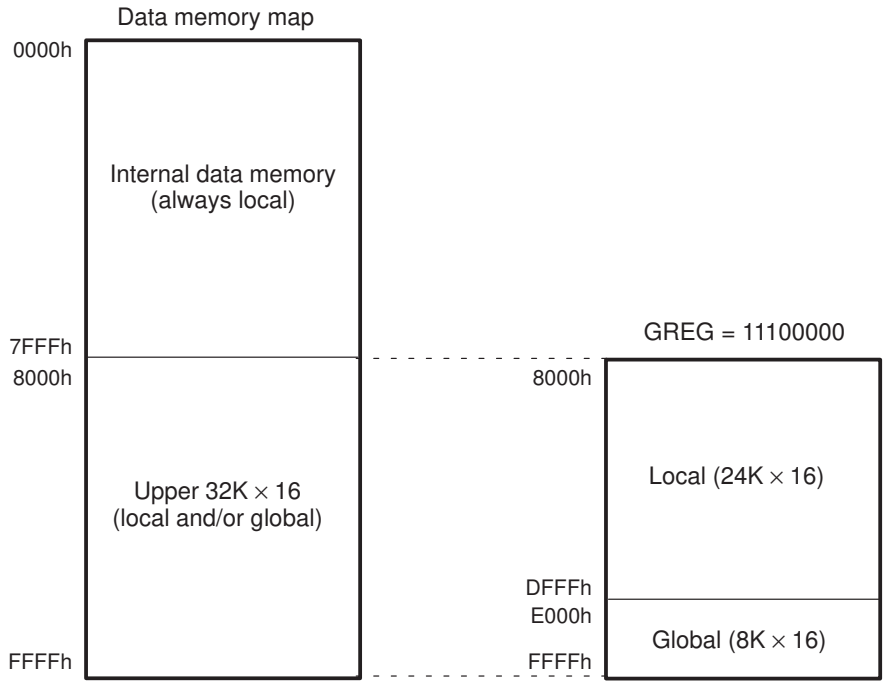


Figure 3–9. Global and Local Data Memory for GREG = 11100000

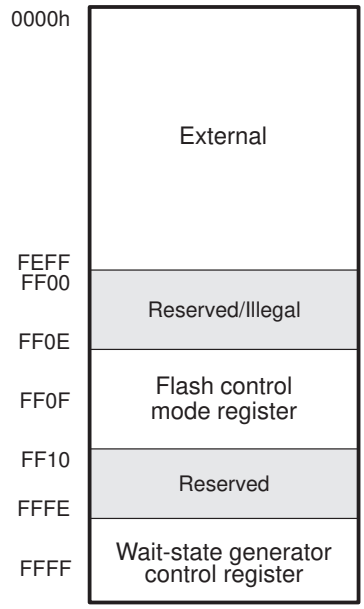


Note: Global memory space is applicable only for ’F243. ’F241 and ’C242 do not have global memory.

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 ’F243.

Figure 3–10. I/O Space Address Map for ’F243



3.7 XMIF Qualifier Signal Description

The ’F243 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	$\overline{PS}$
Data space	32K	$\overline{DS}$
Global data space	32K	$\overline{BR}$
I/O space	64K	$\overline{IS}$

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

Table 3–3. 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.
$\overline{PS}$	Program space strobe
$\overline{DS}$	Data space strobe
$\overline{IS}$	I/O space strobe
$\overline{STRB}$	External memory access strobe
$\overline{WE}$	Write strobe
$\overline{RD}$	Read strobe
R/$\overline{W}$	Read / Write qualifier
$\overline{BR}$	Bus request – used to access global memory.
MP/$\overline{MC}$	MicroProcessor / MicroComputer selection pin
VIS_CLK	Visibility clock – Same as CLKOUT but timing is aligned for external buses in visibility mode.
$\overline{VIS\_OE}$	Is active low whenever the external data bus is driving as an o/p 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 ’F243 device behaves like an ’F241/C242; that is, it 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 ’F243 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

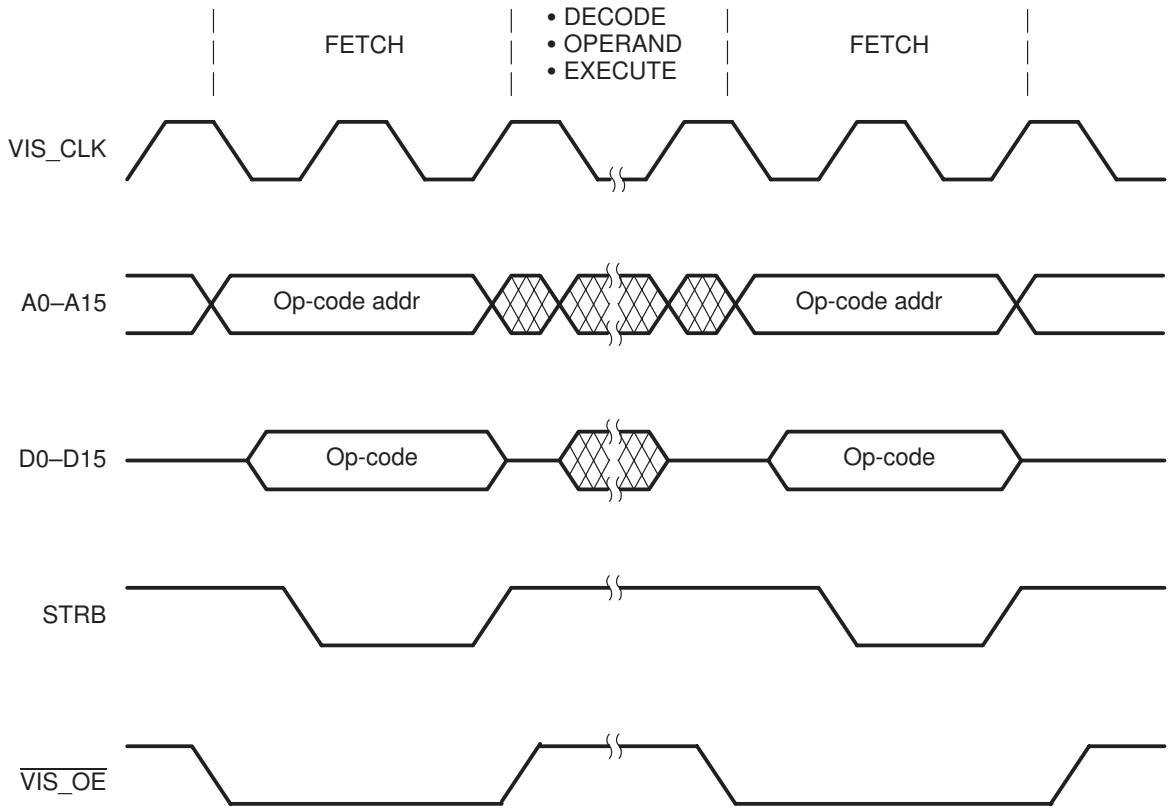
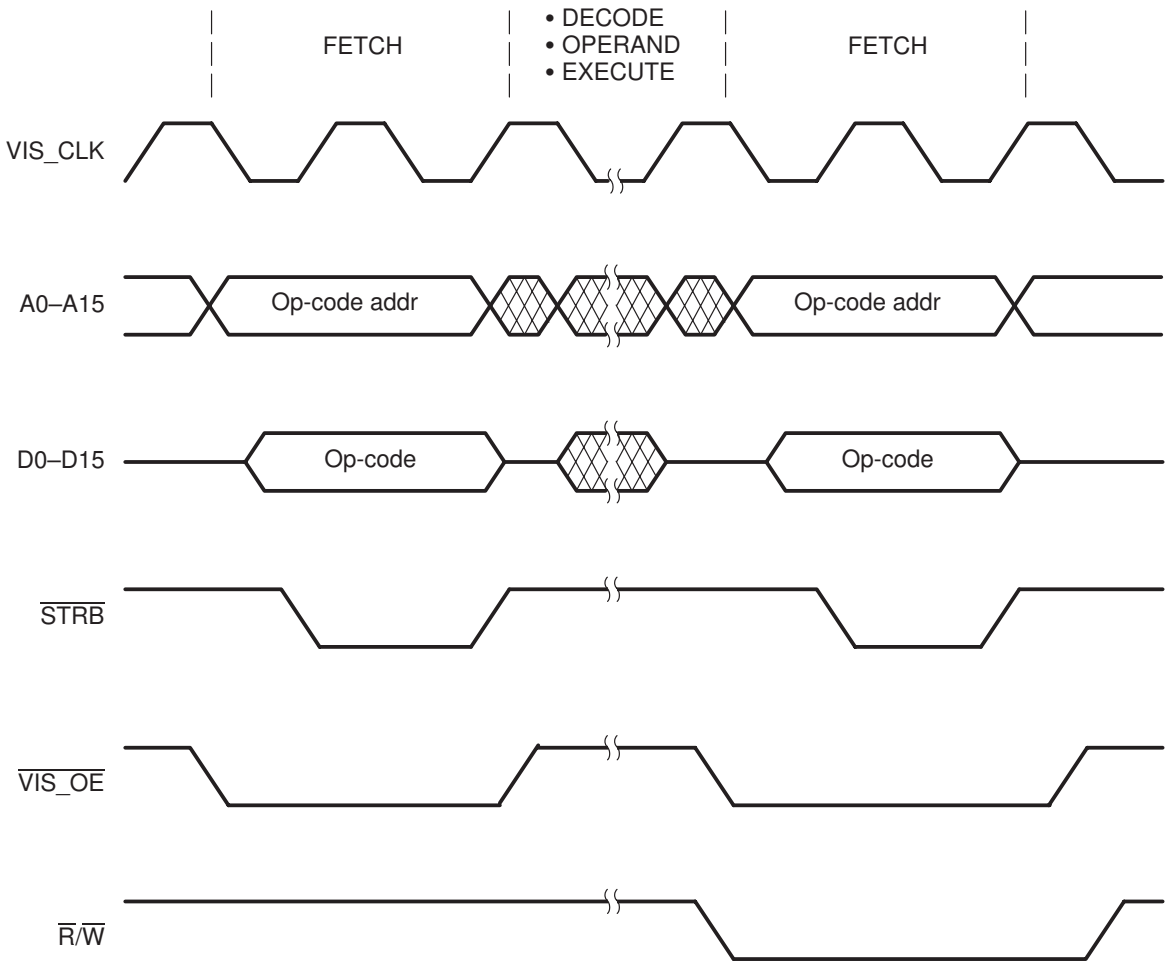


Figure 3–12. Data Address/ Data — Visibility Functional Timing



3.8 Program and Data Spaces

$\overline{PS}$ and $\overline{STRB}$ 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*, on page 3-20).

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

3.9 I/O Space

I/O space accesses are distinguished from program and data memory accesses by $\overline{IS}$ 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 $\overline{IS}$ and $\overline{STRB}$ 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 $\overline{WE}$ goes low and a half-cycle after $\overline{WE}$ 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 ’F243 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 ’F243 always take at least two CLKOUT cycles. The ’F243 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 ’F243 waits one CLKOUT cycle and checks READY again. The ’F243 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 ’F243 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 ’F243 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. ’F243 Wait-State Generator Control Register (WSGR) — I/O-Space Address FFFFh (’F243)

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; value following dash (-) = 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 output 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 3–4 shows how to set the number of wait states you want for each type of off-chip memory.

Table 3–4. Setting the Number of Wait States With the ’F243 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 ’F243/F241/C242 devices use the phase locked loop (PLL) circuit embedded in the ’24x 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/IOPD0 – This is the clock output pin. It is multiplexed with GPIO pin IOPD0 and 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 (SCSR), described in Section 5.3 on page 5-3. This pin is configured to output CLKOUT from the CPU following a device reset.

4.2 Phase Locked Loop

The PLL embedded in the '24x CPU is used in its multiply-by-four ($\times 4$) mode. The PLL is hardwired to this multiplication factor which cannot be changed. For 20-MHz device operation, a 5-MHz input clock must be supplied.

4.3 Watchdog Timer Clock

A low frequency clock, WDCLK, is used to clock the watchdog timer. WDCLK has a nominal frequency of 39062.5 Hz when CPUCLK = 20 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

All ’24x devices have 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 while the peripherals (clocked with CLKOUT) continue to run. The CPU exits the IDLE state if reset, or if it receives an interrupt request.

4.4.1 Clock Domains

All ’24x 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 ’24 xCPU also contains support for a second IDLE mode, IDLE2, implemented in external logic. By asserting the IDLE2 input to the ’24x 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 (SCSR) described in section 5.3 on page 5-3.

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 interrupt, external interrupt, reset
IDLE2 – (LPM1)	01	Off	Off	On	On	On	Wakeup interrupts, external interrupt, reset
HALT – (LPM2) {PLL/OSC power down}	1X	Off	Off	Off	Off	Off	Reset only

4.4.2 Wake Up from Low Power Modes

Wake up from a low-power mode is caused by one of the four events described below.

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.

Note: In Table 4–1, external interrupts include $\overline{\text{PDPINT}}$.

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.

Chapter 5

Reset and External Interrupts

This chapter describes the two reset sources in the ’F243/F241/C242 devices, and shows the configuration and external interrupt control registers.

Topic **Page**

5.1 Reset

The ’F243/F241/C242 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 ’F243/F241/C242 device is resetting itself.

The external reset pin and watchdog timer reset are ORed together to drive the reset input to the CPU. There is a watchdog timer reset flag in the watchdog timer control register that can be used to determine if the WD was the cause of a reset.

5.2 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 (SCSR). See Figure 5–1, *System Control and Status Register*, on page 5-3. 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.

5.3 Configuration Registers

Figure 5–1. System Control and Status Register (SCSR) — Address 7018h

Note: R = Read access, W = Write access, C = Clear only; value following dash (-) = value after reset

Bit 15 **Reserved**. Reads return zero; writes are undefined.

Bit 14 **CLKSRC**. Clock out pin source select.

0 CPU Clock out (CLKOUT), 20 MHz on a 20 MHz device.

1 Watchdog Clock (WDCLK), 39062.5 Hz on a 20 MHz device

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 5–1, *Low-Power Modes Summary,* for a description of the low power modes.

Table 5–1. Low-Power Modes Summary

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

Bit 11–1 **Reserved.** Reads return zero; writes are undefined.

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

When an illegal address occurs, this bit is set. This bit is ***not*** cleared by reset; it is up to software to clear this bit following an illegal address detect. This bit is cleared by writing a 1 to it.

Figure 5–2. 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 (DIN).

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

Device	Rev #	DIN #
’F241	1	0031h
’F241	2	0032h
’F241	3	0033h
’C242	1	0051h
etc.		

5.4 External Interrupt Control Registers

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

5.4.1 External Interrupt 1 Control Register (XINT1CR)

Figure 5–3. 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; value following dash (-) = 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 2–1, *'F243/F241/C242 Interrupt Source Priority and Vectors,* in Chapter Two on page 2-3.

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

5.4.2 External Interrupt 2 Control Register (XINT2CR)

Figure 5–4. 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; value following dash (-) = 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 2–1, *'F243/F241/C242 Interrupt Source Priority and Vectors,* in Chapter 2 on page 2-3.

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 6

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 six 16-bit registers. These registers are divided into two types:

- Output Control registers (OCRx) – 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 bi-directional I/O pins.

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

Topic **Page**

6.1 Digital I/O Ports Register Implementation on ’F243/F241/C242 Devices

Table 6–1 lists the registers available to the digital I/O module as implemented on the ’F243/F241/C242 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 6–1. Shared Pin Configuration

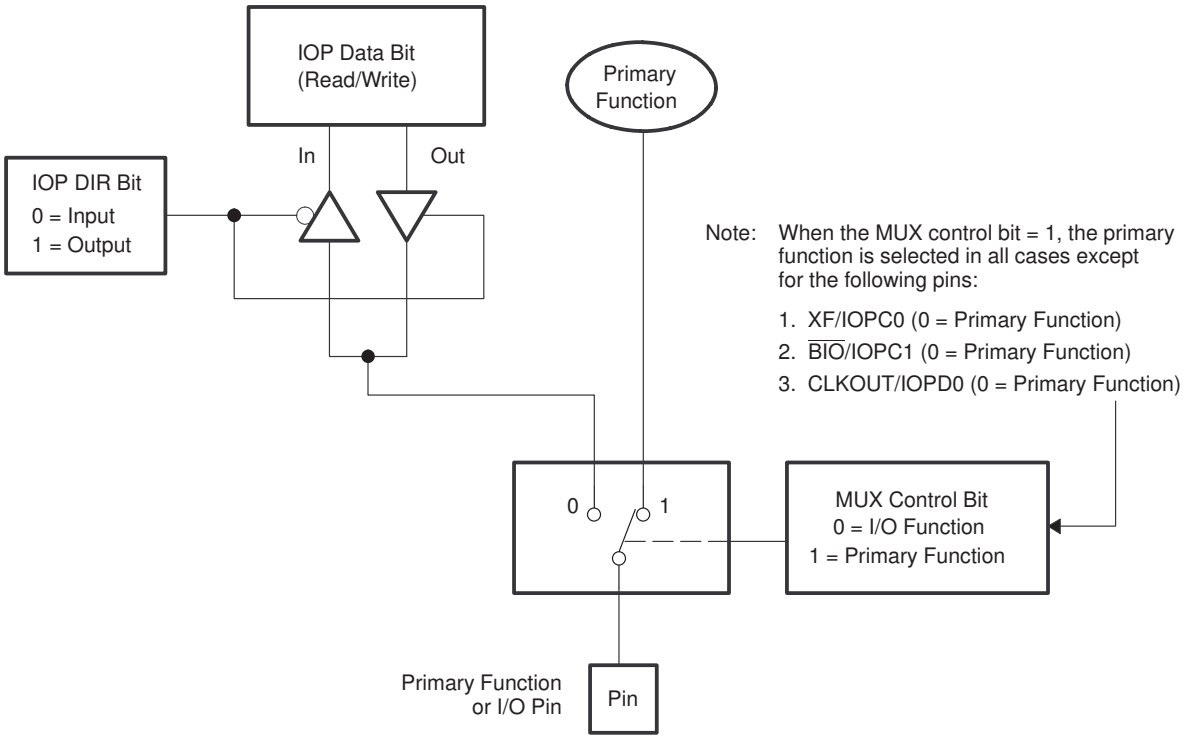


Table 6–1. ’F243/F241/C242 Digital I/O Port Control Register Implementation

Address Offset	Register Mnemonic	Description
7090h	OCRA	Output Control register A
7092h	OCRB	Output Control register B
7094h	Reserved	
7096h	Reserved	
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

6.2 I/O MUX Control Registers

There are two I/O mux control registers: I/O mux control register A (OCRA) and I/O mux control register B (OCRB).

6.2.1 I/O Mux Control Register A

Figure 6–2. I/O Mux Control Register A (OCRA) — Address 7090h

15	14	13	12	11	10	9	8
CRA.15	CRA.14	CRA.13	CRA.12	CRA.11	CRA.10	CRA.9	CRA.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
CRA.7	CRA.6	CRA.5	CRA.4	CRA.3	CRA.2	CRA.1	CRA.0
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

Table 6–2. I/O Mux Control Register A (OCRA) Configuration

		Pin Function Selected	
Bit #	**Name.bit #**	**(CRA.n = 1)**	**(CRA.n = 0)**
0	CRA.0	SCITXD	IOPA0
1	CRA.1	SCIRXD	IOPA1
2	CRA.2	XINT1	IOPA2
3	CRA.3	CAP1/QEP0	IOPA3
4	CRA.4	CAP2/QEP1	IOPA4
5	CRA.5	CAP3	IOPA5
6	CRA.6	CMP1	IOPA6
7	CRA.7	CMP2	IOPA7
8	CRA.8	CMP3	IOPB0
9	CRA.9	CMP4	IOPB1
10	CRA.10	CMP5	IOPB2
11	CRA.11	CMP6	IOPB3
12	CRA.12	T1CMP	IOPB4
13	CRA.13	T2CMP	IOPB5
14	CRA.14	TDIR	IOPB6
15	CRA.15	TCLKIN	IOPB7

6.2.2 I/O Mux Output Control Register B

Output control register B (OCRB) has the following two possible configurations depending on the target device:

1) For ’F241/243, OCRB is shown in Figure 6–3 and Table 6–3.

2) For ’C242, OCRB is shown in Figure 6–4 and Table 6–4.

   Note: The ’C242 does not have the SPI or CAN peripheral modules.

Figure 6–3. Output Control Register B (OCRB) for ’F243/241 only— Address 7092h

15–10	9	8
Reserved	CRB.9	CRB.8
R-0	RW-0	RW-0

7	6	5	4	3	2	1	0
CRB.7†	CRB.6†	CRB.5†	CRB.4†	CRB.3†	CRB.2†	CRB.1	CRB.0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

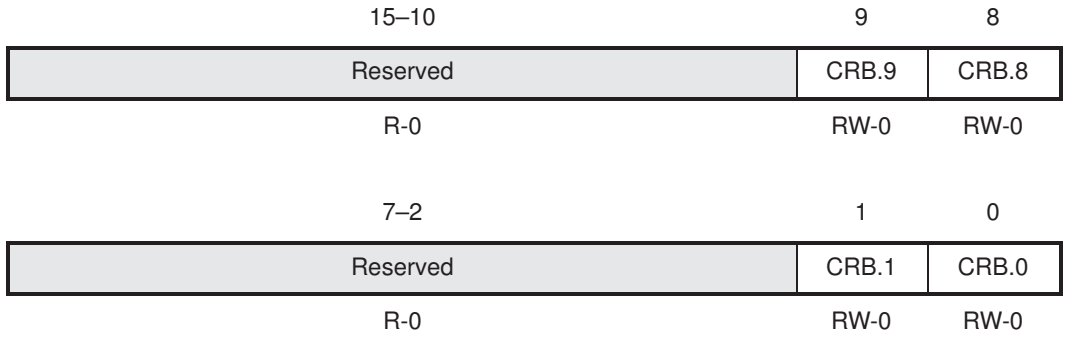
Note: R = Read access; W = Write access; value following dash (-) = value after reset

Note: †These bits are reserved on the ’C242.

Table 6–3. I/O Mux Control Register B (OCRB) Configuration Table for ’F243/F241 Only

		Pin Function Selected	
Bit #	**Name.bit #**	**(CRB.n = 1)**	**(CRB.n = 0)**
0	CRB.0	IOPC0	XF
1	CRB.1	IOPC1	$\overline{\text{BIO}}$
2	CRB.2	SPISIMO	IOPC2
3	CRB.3	SPISOMI	IOPC3
4	CRB.4	SPICLK	IOPC4
5	CRB.5	SPISTE	IOPC5
6	CRB.6	CANTX	IOPC6
7	CRB.7	CANRX	IOPC7
8	CRB.8	IOPD0	CLKOUT
9	CRB.9	XINT2/ADCSOC	IOPD1
10	Reserved		
11	Reserved		
12	Reserved		
13	Reserved		
14	Reserved		
15	Reserved		

Figure 6–4. Output Control Register B (OCRB) for ’C242 only— Address 7092h



Note: R = Read access; W = Write access; value following dash (-) = value after reset

Table 6–4. I/O Mux Control Register B (OCRB) Configuration Table for ’C242 Only

		Pin Function Selected	
Bit #	**Name.bit #**	**(CRB.n = 1)**	**(CRB.n = 0)**
0	CRB.0	IOPC0	XF
1	CRB.1	IOPC1	$\overline{\text{BIO}}$
2	Reserved		
3	Reserved		
4	Reserved		
5	Reserved		
6	Reserved		
7	Reserved		
8	CRB.8	IOPD0	CLKOUT
9	CRB.9	XINT2/ADCSOC	IOPD1
10	Reserved		
11	Reserved		
12	Reserved		
13	Reserved		
14	Reserved		
15	Reserved		

6.3 Data and Direction Control Registers

There are four data and direction control registers. Refer to Table 6–1, *'F243/F241/C242 Digital I/O Port Control Registers Implementation*, on page 6-3 for the address locations of each register.

Figure 6–5. Port A Data and Direction Control Register (PADATDIR) — Address 7098h

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; value following dash (-) = 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*

Figure 6–6. Port B Data and Direction Control Register (PBDATDIR) — Address 709Ah

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; value following dash (-) = 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*.

Figure 6–7. Port C Data and Direction Control Register (PCDATDIR) —Address 709Ch

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; value following dash (-) = 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:

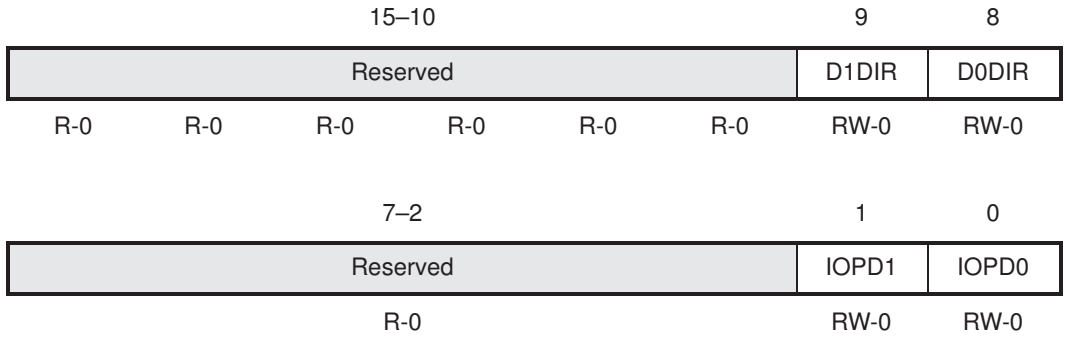
0 Set corresponding I/O pin *low*
1 Set corresponding I/O pin *high*

The port D data and direction control register has the following two possible configurations depending on the target device:

1) For ’C242, PDDATDIR is shown in Figure 6–8.

2) For ’F243 and ’F241, PDDATDIR is shown in Figure 6–9.

Figure 6–8. Port D Data and Direction Control Register (PDDATDIR) for ’C242 only — Address 709Eh



Note: R = Read access; W = Write access; value following dash (-) = value after reset

Bits 9–8 DnDIR

0 Configure corresponding pin as an input
1 Configure corresponding pin as an output

Bits 1–0 IOPDn

If DnDIR = 0, then:

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

If DnDIR = 1, then:

0 Set corresponding I/O pin *low*
1 Set corresponding I/O pin *high*

Bits 15-10, 7-2 Reserved

Reads return zero; writes have no effect.

Figure 6–9. Port D Data and Direction Control Register (PDDATDIR) for ’F243 and ’F241 — Address 709Eh

15	14	13	12	11	10	9	8
D7DIR†	D6DIR†	D5DIR†	D4DIR†	D3DIR†	D2DIR†	D1DIR	D0DIR
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

7	6	5	4	3	2	1	0
IOPD7†	IOPD6†	IOPD5†	IOPD4†	IOPD3†	IOPD2†	IOPD1	IOPD0
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

† The I/O pins associated with these bits are only bonded out on the ’F243. Therefore, these bits have no effect on the ’F241, even though they can be written to and read from.

Bits 15–8 DnDIR

0 Configure corresponding pin as an input
1 Configure corresponding pin as an output

Bits 7–0 IOPDn

If DnDIR = 0, then:

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

If DnDIR = 1, then:

0 Set corresponding I/O pin *low*
1 Set corresponding I/O pin *high*

Chapter 7

Event Manager (EV2)

This chapter describes the ’F243/F241/C242 Event Manager (EV2) module. Most of the EV2 device pins are shared with general purpose digital I/O signals. This pin sharing and how it is controlled is described in Chapter 6, *Digital Input/Output (I/O)*.

The EV2 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 EV2 module of ’F243/F241/C242 devices and the EV module of ’240 devices.

Topic **Page**

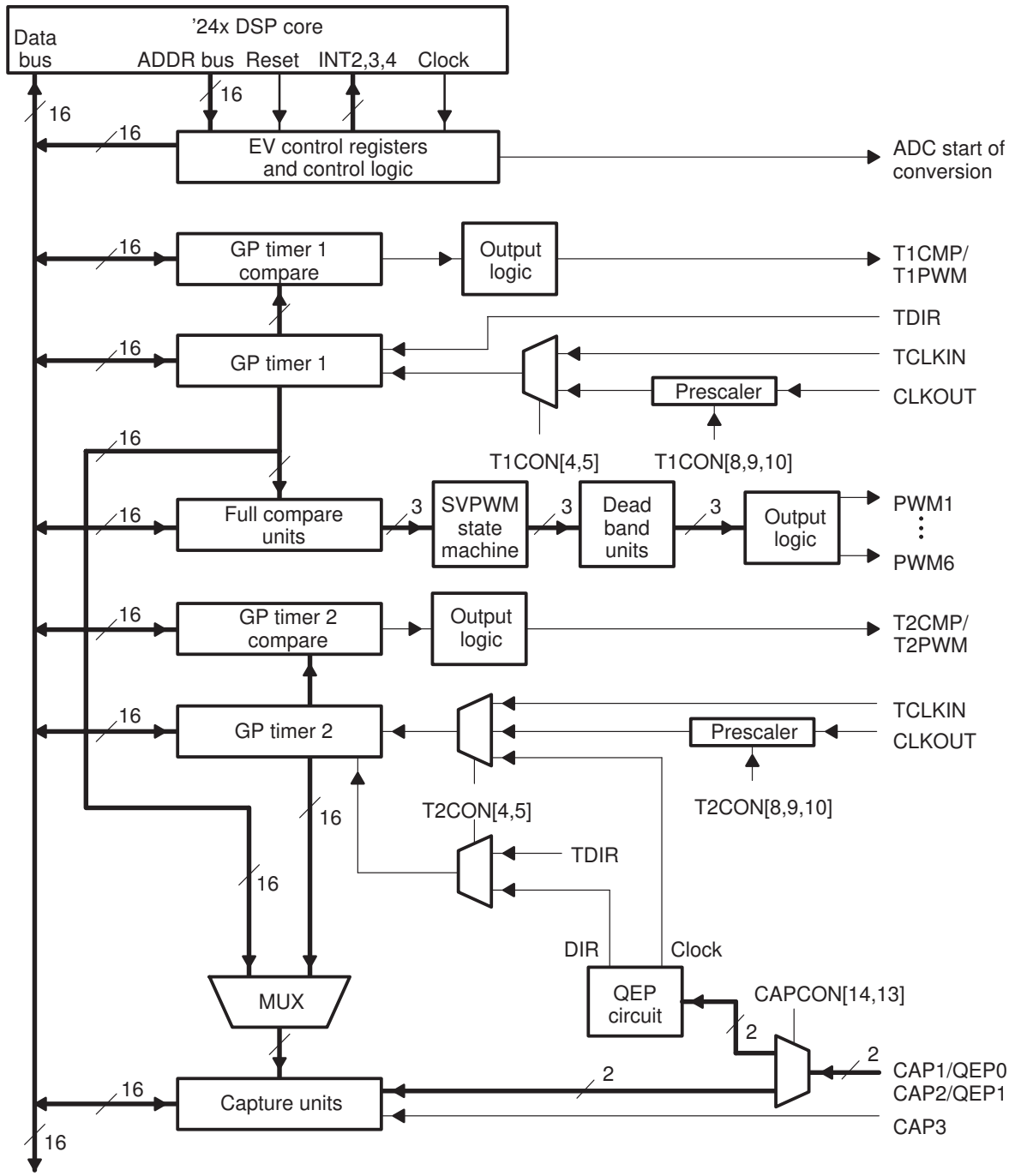
7.1 Event Manager (EV2) Functional Blocks

The EV2 module in ’24x devices contain the following functional blocks:

- ❑ Two general purpose (GP) timers (described in section 7.3 on page 7-11).
- ❑ Three compare units (described in section 7.4 on page 7-34).
- ❑ Pulse-width modulation (PWM) circuits that include space vector PWM circuits, dead-band generation units, and output logic (described in section 7.5 on page 7-40, section 7.6 on page 7-48, and section 7.7 on page 7-52).
- ❑ Three capture units (described in section 7.8 on page 7-58).
- ❑ Quadrature encoder pulse (QEP) circuit (described in section 7.9, page 7-58).
- ❑ Interrupt logic (described in section 7.10 on page 7-68).

Figure 7–1 shows a block diagram of the EV2 module.

Figure 7–1. Event Manager (EV2) Block Diagram



7.1.1 Differences between ’C240 EV and ’24x EV2

- GP timer 3 has been removed.
- 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 since GP timer 3 is not present.
- 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.
- Capture 4 has been removed.
- A capture 3 event is now able to start the ADC.
- The capture units can now use GP timer 2 or GP timer 1 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.
- 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 en-

able 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.
- $\overline{\text{PDPINT}}$ is now enabled following reset.
- Only one write is required to initialize COMCON, not two as on the ’C240.

7.1.2 EV2 Pins

The EV2 module has eight device pins available for compare/PWM outputs:

- Two GP timer compare/PWM output pins:
  - T1CMP/T1PWM
  - T2CMP/T2PWM
- Six (full) compare/PWM output pins:
  - PWM1
  - PWM2
  - PWM3
  - PWM4
  - PWM5
  - PWM6

The EV2 module uses three device pins, CAP1/QEP0, CAP2/QEP1, and CAP3, as capture or quadrature encoder pulse inputs.

The timers in the EV2 module can be programmed to operate based on an external clock or the internal device clock. The device pin TCLKIN supplies the external clock input.

The device pin TDIR 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 7–1.

Table 7–1. Event Manager Pins

Pin Name	Description
CAP1/QEP0	Capture Unit 1 input, QEP circuit input 0
CAP2/QEP1	Capture Unit 2 input, QEP circuit input 1
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
TCLKIN	External clock input for Timers
TDIR	External timer direction input

7.1.3 Power Drive Protection

An interrupt is generated when the device pin power drive protection interrupt ($\overline{\text{PDPINT}}$) is pulled low. This interrupt is provided for the safe operation of systems such as power converters and motor drives. If $\overline{\text{PDPINT}}$ is unmasked, all EV2 output pins will be put in the high-impedance state by hardware immediately after the $\overline{\text{PDPINT}}$ pin is pulled low. The interrupt flag associated with $\overline{\text{PDPINT}}$ is also set when such an event occurs; however, it must wait until the transition on $\overline{\text{PDPINT}}$ has been qualified and synchronized with the internal clock. The qualification and synchronization causes a delay of *2* clock cycles. If $\overline{\text{PDPINT}}$ is unmasked, the flag keeps the EV2 outputs in the high-impedance state and generates a peripheral interrupt request. The setting of the flag does not depend on whether $\overline{\text{PDPINT}}$ is masked: it happens when a qualified transition occurs on the $\overline{\text{PDPINT}}$ pin. $\overline{\text{PDPINT}}$ 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.

7.1.4 EV2 Registers

The Event Manager registers occupy a 64 word (16-bit) frame 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 EV2 on that device.

On the ’F243/F241/C242 devices (as with the ’C240 device) the Event Manager registers are located in the range 7400h to 743Fh.

The undefined registers and undefined bits of the EV2 registers all return zero when read by user software. Writes have no effect. See Section 7.2, on page 7-9.

7.1.5 EV2 Interrupts

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 ’24x interrupt requests have the following stages of response:

- *Interrupt source*. If peripheral interrupt conditions occur, the respective flag bits in registers EVIFRA, EVIFRB or EVIFRC 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 enabled or disabled by interrupt mask registers EVIMRA, EVIMRB, and EVIMRC. 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 (INT2, 3, or 4) based on the preassigned priority of the received interrupts. See Table 2–1, *’F243/F241/C242 Interrupt Source Priority and Vectors*, on page 2-3 for priority and vector values.
- *CPU response*. On receipt of INT2, 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 INT2, 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 a CPU 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 EVIFRA, EVIFRB, or EVIFRC that caused the serviced interrupt. Code will return after enabling the CPU's global interrupt bit INTM (clear INTM bit).

7.2 Event Manager (EV2) Register Addresses

Table 7–2 through Table 7–5 display the addresses of the Event Manager registers.

Table 7–2. Addresses of Timer Registers

Address	Register	Name	
7400h	GPTCON	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 7–3. Addresses of Compare Control Registers

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

Table 7–4. Addresses of Capture Registers

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

Table 7–5. Addresses of EV2 Interrupt Registers

Address	Register	Name
742Ch	EVIMRA	Interrupt mask register A
742Dh	EVIMRB	Interrupt mask register B
742Eh	EVIMRC	Interrupt mask register C
742Fh	EVIFRA	Interrupt flag register A
7430h	EVIFRB	Interrupt flag register B
7431h	EVIFRC	Interrupt flag register C

7.3 General Purpose (GP) Timers

There are two general purpose (GP) timers in the EV2 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 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 7–2 shows a block diagram of a GP timer. Each GP timer includes:

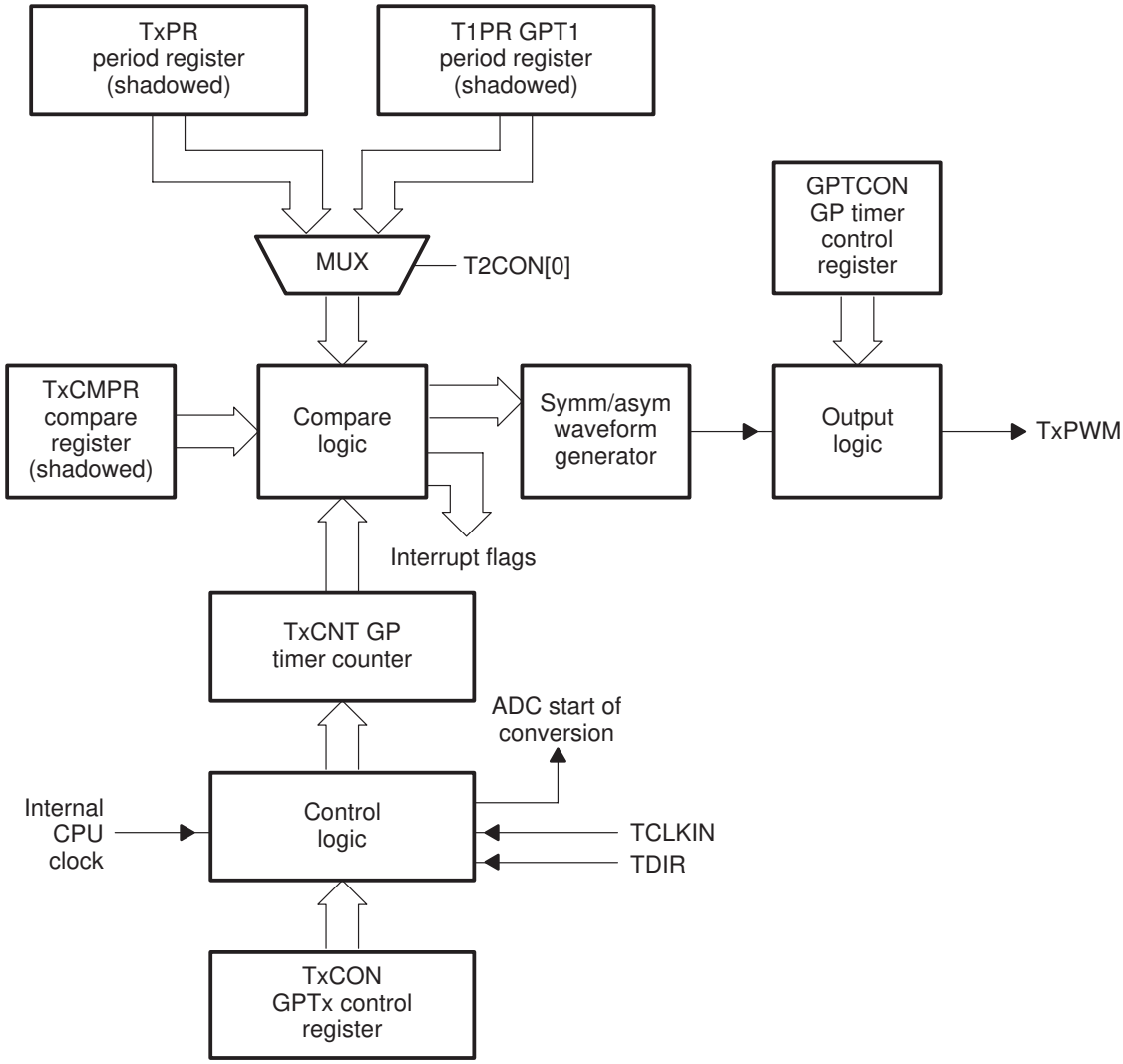
- One readable and writeable (RW) 16-bit up- and up/down-counter register TxCNT (x = 1, 2). This register stores the current value of the counter and keeps incrementing or decrementing depending on the direction of counting.
- One RW 16-bit timer compare register (shadowed), TxCMPR (x = 1, 2)
- One RW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

Another overall control register, GPTCON, specifies the action to be taken by the timers on different timer events, and indicates the counting directions of the GP timers. GPTCON 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 7–2, the mux is applicable only when the figure represents timer 2.

Figure 7–2. General Purpose Timer Block Diagram (x = 1 or 2)



GP Timer Inputs

The inputs to the GP timers are:

- The internal device (CPU) clock.
- An external clock, TCLKIN, that has a maximum frequency of one-fourth that of the device clock.
- Direction input, TDIR, 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
- 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)

T1CON and T2CON are the two individual timer control registers (in contrast to the overall GP timer control register, GPTCON). These registers control the operational mode of the timer. See Figure 7–11 on page 7-29.

Overall GP Timer Control Register (GPTCON)

The control register GPTCON specifies the action to be taken by the timers on different timer events and indicates their counting directions. See Figure 7–12 on page 7-31.

GP Timer Compare Registers (TxCMPR)

T1CMPR is the associated compare register for Timer 1 and T2CMPR is the associated compare register for Timer 2. Both have an associated PWM output pin, T1PWM and T2PWM.

Figure 7–3. GP Timer x Compare Register (TxCMPR; x = 1 or 2) — Addresses 7402h and 7406h

15	14	13	12	11	10	9	8
D15	D14	D13	D12	D11	D10	D9	D8
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0
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; value following dash (–) is value after reset

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 bit 1 in T1CON 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 GPTCON, one device clock cycle after the match.
- If the compare interrupt flag has been selected by the appropriate GPTCON 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.

GP Timer Period Register (TxPR)

T1PR is the 16-bit period register for Timer 1 and T2PR is the 16-bit period register for Timer 2. The value in the period register of a GP Timer determines the period of the timer.

Figure 7–4. GP Timer x Period Register (TxPR; x = 1 or 2) — Addresses 7403h and 7407h

15	14	13	12	11	10	9	8
D15	D14	D13	D12	D11	D10	D9	D8
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0
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; value following dash (–) is value after reset

The operation of a GP timer stops and holds at its current value, resets to 0, or starts counting downward when a match occurs between the period register and the timer counter, depending on the counting mode of the timer.

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 GPTCON 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 GPTCON during all timer operations as follows:

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

The input pin TDIR determines the direction of counting when a GP timer is in directional up/down-counting mode. When TDIR is high, upward counting is specified; when TDIR 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, TCLKIN. The frequency of the external clock must be less than or equal to one-fourth of that of the device clock. GP timer 2 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 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 by proper configuration of T2CON in the following ways:

- Set the TSWT1 bit in T2CON to start GP timer 2 counting with the *tenable* bit in T1CON (thus both timer counters start simultaneously).
- Initialize the timer counters in GP timers 1 and 2 with different values before starting synchronized operation.
- 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.

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 GPTCON 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 eight interrupt flags in EVIFRA and EVIFRB for the GP timers. Each GP timer can generate four interrupts upon the following events:

- ❏ Overflow: TxOFINT (x = 1 or 2)
- ❏ Underflow: TxUFINT (x = 1 or 2)
- ❏ Compare match: TxCINT (x = 1 or 2)
- ❏ Period match: TxPINT (x = 1 or 2)

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.

7.3.1 GP Timer Counting Operation

GP Timer x Counter Register (TxCNT)

T1CNT and T2CNT are 16-bit readable and writeable up and up/down counter registers for Timers 1 and 2. These registers store the current value of the counter and keep incrementing or decrementing depending on the direction of counting.

Figure 7–5. GP Timer x Counter Register (TxCNT; x = 1 or 2) — Addresses 7401h and 7405h

15	14	13	12	11	10	9	8
D15	D14	D13	D12	D11	D10	D9	D8
RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0	RW–0
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; value following dash (–) is value after reset

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 in 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 GPTCON 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 GPTCON 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.

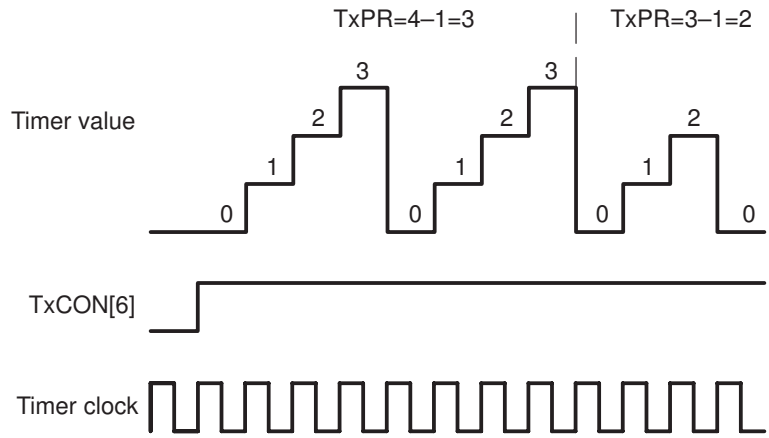
The initial value of the GP timer can be any value between 0h and FFFFh inclusive. When the initial value is greater than the value in the period register, the timer counts up to FFFFh, resets to 0, and continues the operation as if the initial value was 0. When the initial value in the timer counter 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 GPTCON 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. TDIR 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 7–6 shows the continuous up-counting mode of the GP timer.

Figure 7–6. GP Timer Continuous Up-Counting Mode (TxPR = 3 or 2)



As shown in Figure 7–6, *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 TDIR 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 TDIR 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 TDIR 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 TDIR 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.

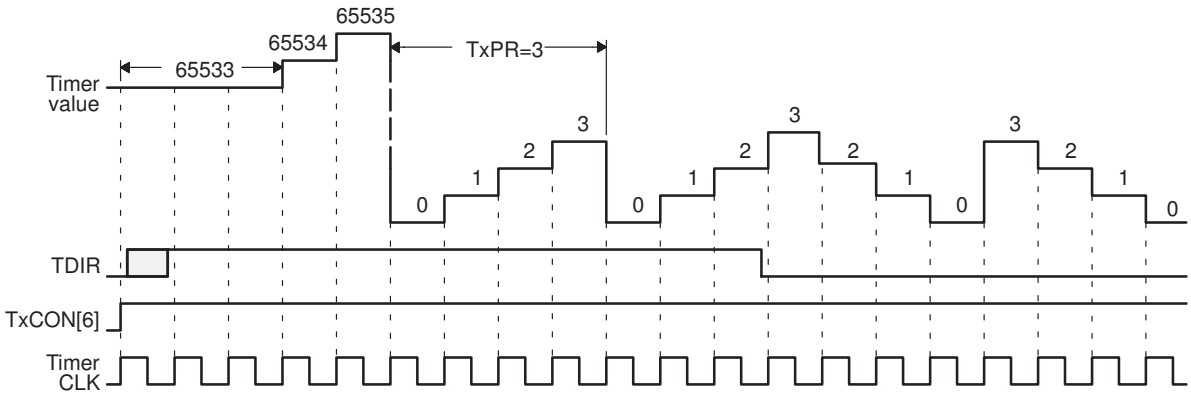
The period, underflow, and overflow interrupt flags, interrupts, and associated actions are generated on respective events in the same manner as they are generated in the continuous up-counting mode.

The latency from a change of TDIR 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 GPTCON: 1 means counting up; 0 means counting down. Either the external clock from the TCLKIN pin or the internal device clock can be used as the input clock for the timer in this mode.

Figure 7–7 shows the directional up/down-counting mode of the GP timers.

Figure 7–7. GP Timer Directional Up/Down-Counting Mode: Prescale Factor 1 and TxPR = 3



The directional up/down-counting mode of GP timer 2 can be used with the quadrature encoder pulse (QEP) circuits in the EV2 module. In this case, the QEP circuits provide both the counting clock and direction for GP timer 2. 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 TDIR 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

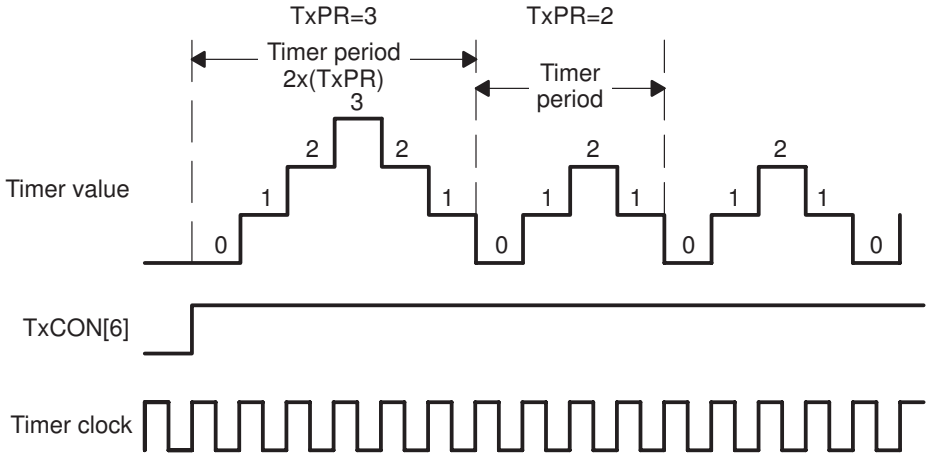
as that of the period register, the timer counts down to 0 and continues again as if the initial value was 0. If the initial value of the timer is between 0 and the contents of the period register, the timer counts up to the period value and continues to finish the period as if the initial counter value 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 GPTCON is 1 when the timer counts upward and 0 when the timer counts downward. Either the external clock from the TCLKIN pin or the internal device clock can be selected as the input clock. TDIR input is ignored by the timer in this mode.

Figure 7–8 shows the continuous up/down-counting mode of the GP timer.

Figure 7–8. 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.

7.3.2 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 GPTCON, one device clock cycle after the match.
- ❏ If the compare interrupt flag has been selected by the appropriate GPTCON 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 GPTCON
- ❏ 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

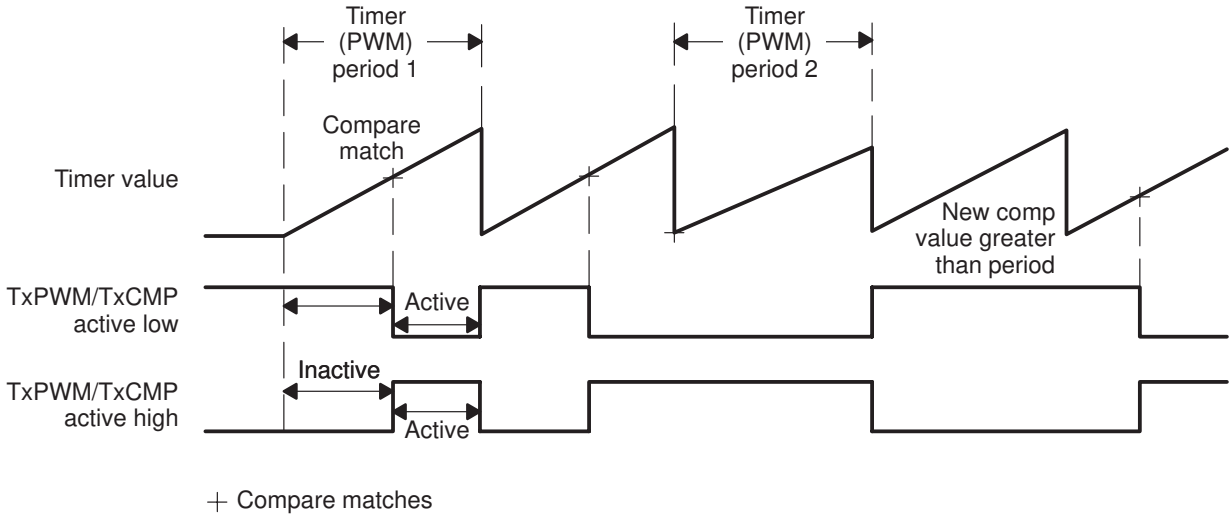
An asymmetric waveform (Figure 7–9) is generated when the GP timer is in continuous up-counting mode. When the GP timer is in this mode, the output of the waveform generator changes according to the following sequence:

- ❏ 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 7–9. GP Timer Compare/PWM Output in Up-Counting Mode



Symmetric Waveform Generation

A symmetric waveform (Figure 7–10) 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

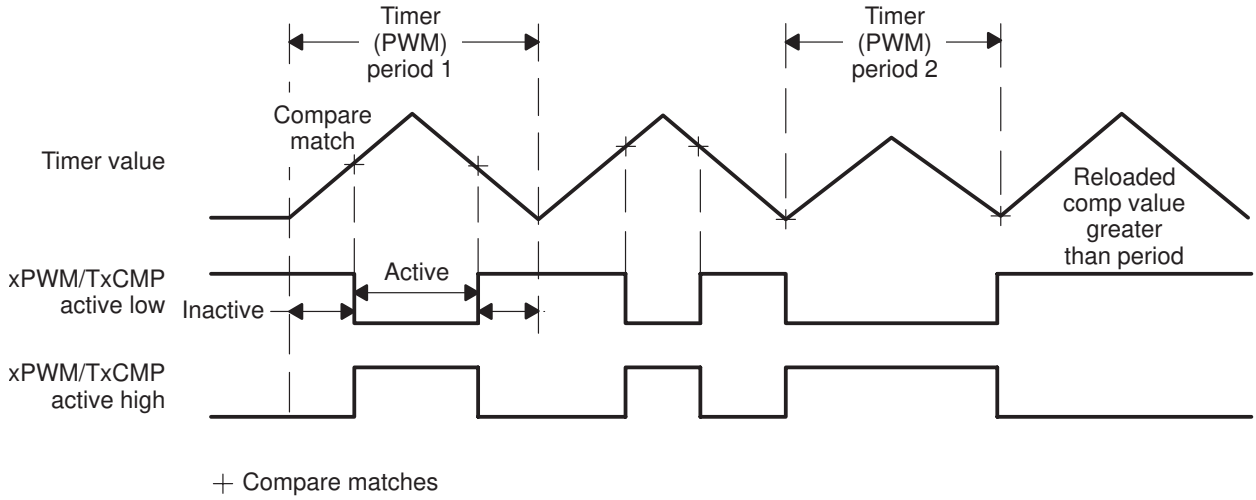
- ❏ Resets to 0 at the end of the period if there is no second compare match, and the new compare value for the following period is not 0

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 7–10. 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 GPTCON 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 GPTCON 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 7–6 for the continuous up-counting mode and according to Table 7–7 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 7–6. 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 7–7. 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:

- ❏ GPTCON[6] is set to 0 by software
- ❏ PDPINT 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) – $(TxCMPR)_{up}$ + (TxPR) – $(TxCMPR)_{dn}$ cycles of the scaled input clock, where $(TxCMPR)_{up}$ is the compare value on the way up and $(TxCMPR)_{dn}$ 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 $(TxCMPR)_{up}$ is 0. The output remains active until the end of the period if $(TxCMPR)_{dn}$ 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 $(TxCMPR)_{up}$ is greater than or equal to (TxPR). Similarly, the second transition is lost when $(TxCMPR)_{dn}$ is greater than or equal to (TxPR). The GP timer compare output is inactive for the entire period if both $(TxCMPR)_{up}$ and $TxCMPR)_{dn}$ are greater than or equal to (TxPR) for the up/down-counting mode.

Figure 7–9, *GP Timer Compare/PWM Output in Up-Counting Mode* (page 7-25) shows the compare operation of a GP timer in the up-counting mode. Figure 7–10, *GP Timer Compare/PWM Output in Up/Down-Counting Modes* (page 7-26) shows the compare operation of a GP timer in the up/down-counting mode.

7.3.3 Timer Control Registers (TxCON and GPTCON)

The addresses of the GP timer registers are given in Table 7–2 on page 7-9. The bit definition of the individual GP timer control registers, TxCON, is shown in Figure 7–11. The bit definition of the overall GP timer control register, GPTCON, is shown in Figure 7–12 on page 7-31.

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

T1CON and T2CON are the two individual timer control registers (in contrast to the overall GP timer control register, GPTCON). These registers control the operational mode of the timer.

Figure 7–11. GP Timer x Control Register (TxCON; x = 1 or 2) — Addresses 7404h and 7408h

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	TENABLE	TCLKS1	TCLKS0	TCLD1	TCLD0	TECMPR	SELT1PR
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 15–14 **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.** (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.

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**. Period register select. This bit is a reserved bit in T1CON.

0 Use own period register
1 Use T1PR as period register ignoring own period register

Overall GP Timer Control Register (GPTCON)

GPTCON provides status of GP Timers 1 and 2, and controls ADC start, compare outputs, and their polarities.

Figure 7–12. GP Timer Control Register (GPTCON) — Address 7400h

Note: R = Read access; W = Write access; value following dash (-) = 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

Bit 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 PDPINT 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

Bit 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

7.3.4 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.

7.3.5 GP Timer Reset

When any RESET event occurs, the following happens:

- ❏ All GP timer register bits, except for the counting direction indication bits in GPTCON, 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 PDPINT; thus, all GP timer interrupts are masked, except for PDPINT.
- ❏ All GP timer compare outputs are put in the high-impedance state.

7.4 Compare Units

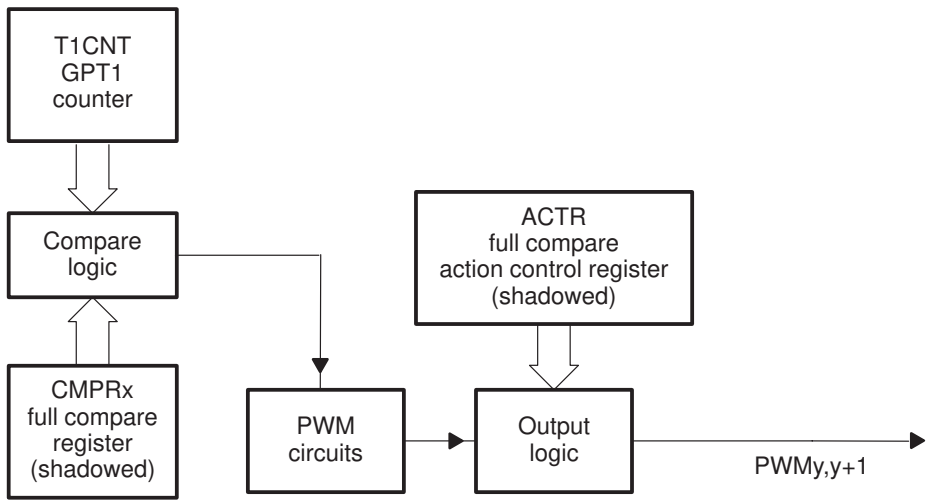
There are three (full) compare units (compare units 1, 2, and 3) in the EV2 module. Each compare unit has two associated PWM outputs. The time base for the compare units is provided by GP timer 1.

The three compare units include:

- ❑ Three 16-bit compare registers (CMPRx, x = 1, 2, 3), all with an associated shadow register, (RW)
- ❑ One 16-bit compare control register (COMCON), (RW)
- ❑ One 16-bit action control register (ACTR), with associated shadow register, (RW)
- ❑ Six PWM (3-state) output pins, PWMy, y = 1, 2, 3, 4, 5, 6
- ❑ Control and interrupt logic

The functional block diagram of a compare unit is shown in Figure 7–13.

Figure 7–13. Compare Unit Block Diagram (x = 1, 2, 3; y = 1, 3, 5)



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

Compare Inputs/Outputs

The inputs to a compare unit include:

- Control signals from control registers
- GP timer 1 (T1CNT) 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 COMCON. 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 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 (ACTR). The bits in ACTR 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:

- ❑ Setting up T1PR
- ❑ Setting up ACTR
- ❑ Initializing CMPRx
- ❑ Setting up COMCON
- ❑ Setting up T1CON

7.4.1 Compare Units Registers

The addresses of registers associated with compare units and associated PWM circuits are shown in Table 7–3, *Addresses of Compare Control Registers* (page 7-9), and further discussed in the following subsections.

Compare Control Register (COMCON)

The operation of the compare units is controlled by the compare control register (COMCON). The bit definition of COMCON is summarized in Figure 7–14, COMCON is readable and writeable.

Figure 7–14. Compare Control Register (COMCON) — 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; value following dash (-) = value after reset

Bit 15 **CENABLE**. Compare enable.

- 0 Disable compare operation. All shadowed registers (CMPRx, ACTR) 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

Bit 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 PDPINT 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

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

Compare Action Control Register (ACTR)

The compare action control register (ACTR) controls the action that takes place on each of the six compare output pins (PWMx, x = 1–6) on a compare event, if the compare operation is enabled by COMCON[15]. ACTR is double buffered. The condition on which the ACTR is reloaded is defined by bits in COMCON. ACTR also contains the SVRDIR, D2, D1, and D0 bits needed for space vector PWM operation. The bit configuration of ACTR is described in Figure 7–15.

Figure 7–15. Compare Action Control Register (ACTR) — 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; value following dash (-) = 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

7.4.2 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.

7.4.3 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.

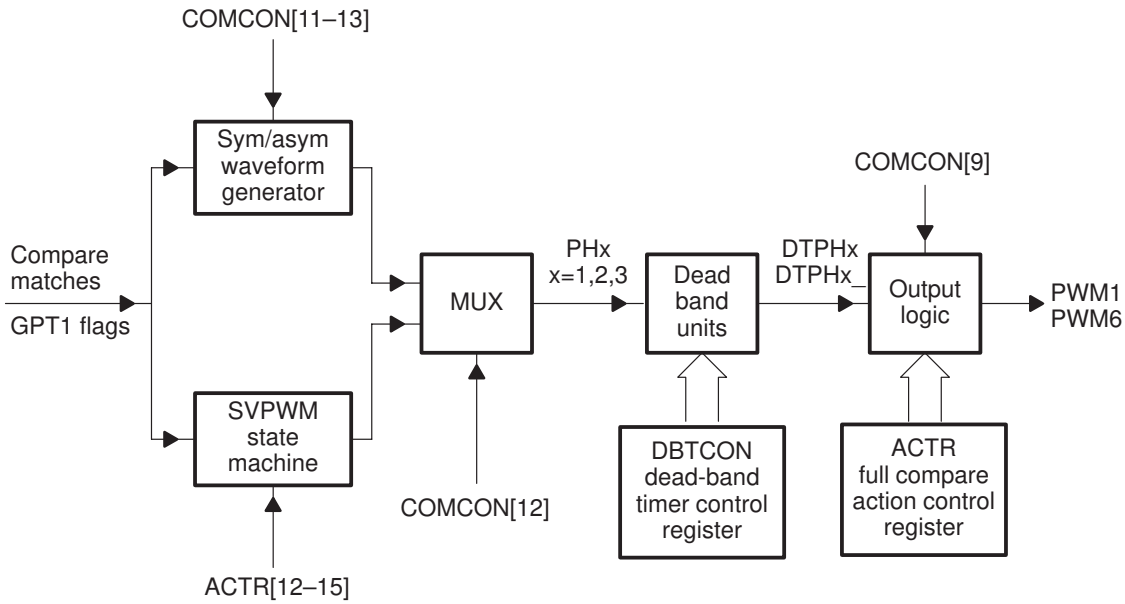
7.5 PWM Circuits Associated with Compare Units

The PWM circuits associated with compare units make it possible to generate six PWM output channels with programmable dead-band and output polarity. The PWM circuits functional block diagram is shown in Figure 7–16. It includes the following functional units:

- ❏ Asymmetric/Symmetric Waveform Generators
- ❏ Programmable Dead-Band Unit (DBU)
- ❏ Output Logic
- ❏ Space Vector (SV) PWM State Machine

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 7.5.2 and 7.5.3, respectively. The space vector PWM state machine and the space vector PWM technique are described later in this chapter.

Figure 7–16. 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, COMCON, ACTR, and DBTCON.

7.5.1 PWM Generation Capability of Event Manager

The PWM waveform generation capability of the event manager 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

7.5.2 Programmable Dead-Band Unit

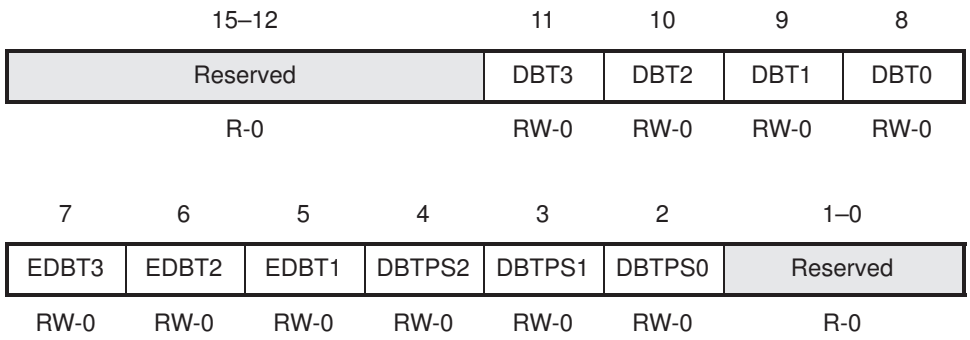
The programmable dead-band unit features:

- ❑ One 16-bit dead-band control register, DBTCON (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 Register (DBTCON)

The operation of the dead-band unit is controlled by the dead-band timer control register (DBTCON). The bit description of DBTCON is given in Figure 7–17.

Figure 7–17. Dead-Band Timer Control Register (DBTCON) — Address xx15h



Note: R = Read access; W = Write access; value following dash (-) = 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.

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 DBTCON bits. If you assume the value in DBTCON[11–8] is *m,* and the value in DBTCON[4–2] corresponds to prescaler *x/p,* then the dead-band value is ($p \times m$) device clock cycles.

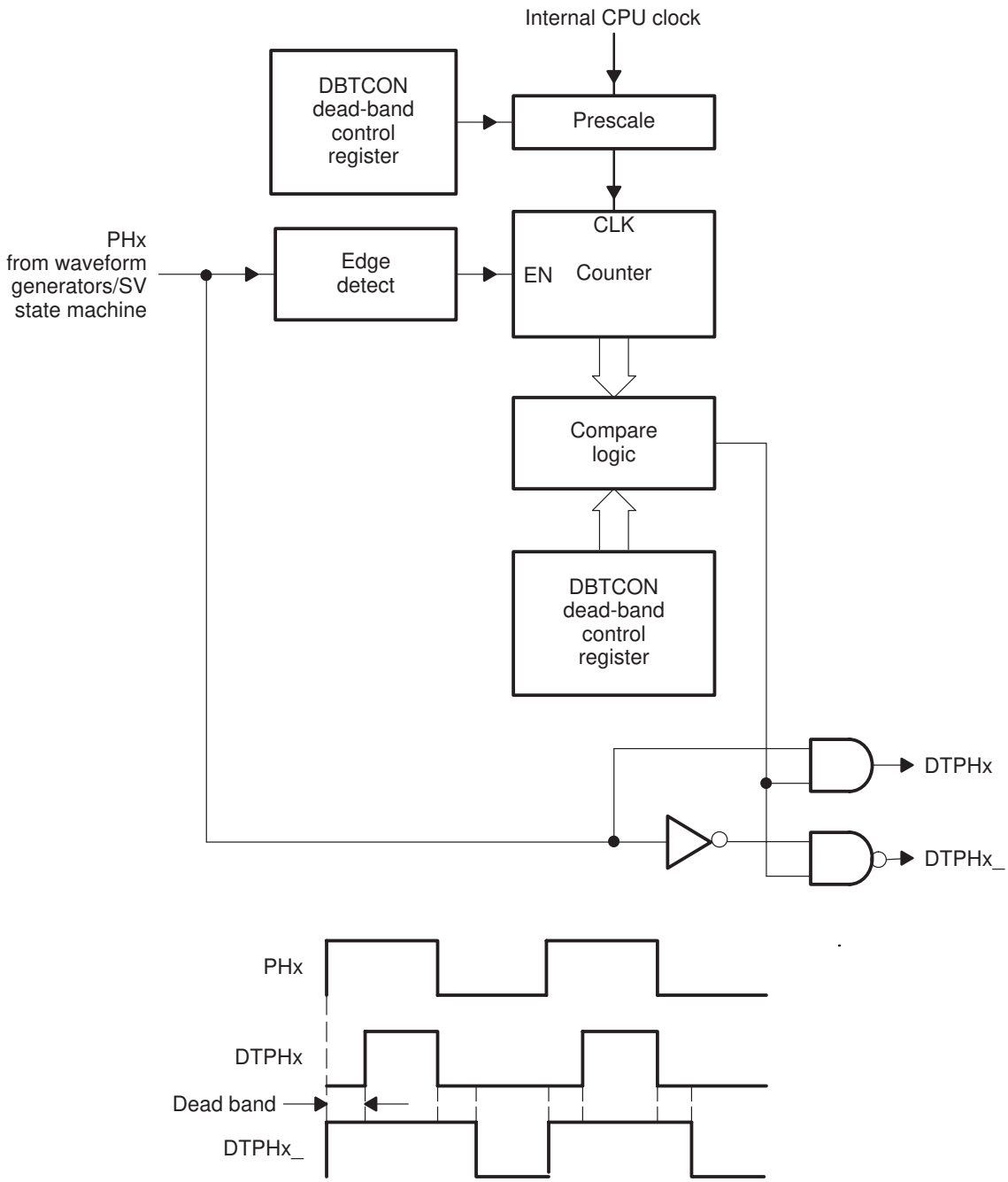
Table 7–8 on page 7-44 shows the dead-band generated by typical bit combinations in DBTCON. The values are based on a 50 ns device clock. Figure 7–18 on page 7-45 shows the block diagram of the dead-band logic for one compare unit.

Table 7–8. Dead-Band Generation Examples

DBT3–DBT0 (*m*) (DBTCON[11–8])	DBTPS2–DBTPS0 (*p*) (DBTCON[4–2])					
	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 7–18. Dead-Band Unit Block Diagram (x = 1, 2, or 3)



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.

7.5.3 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 = 1–6. 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 six PWM output pins can all be put in the high-impedance state by any of the following:

- ❏ Software clearing the COMCON[9] and COMCON[8] bits, respectively
- ❏ Hardware pulling PDPINT low when PDPINT is unmasked
- ❏ The occurrence of any reset event

Active PDPINT (when enabled) and system reset override the bits in COMCON and ACTR

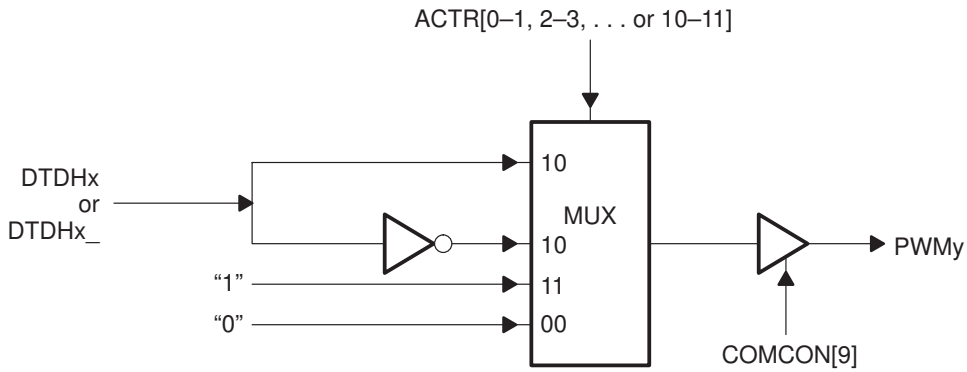
Figure 7–19, on page 7-47, shows a block diagram of the output logic circuit (OLC). The inputs of Output Logic for the compare units are:

- ❏ DTPH1, DTPH1_, DTPH2, DTPH2_, DTPH3, and DTPH3_ from the dead-band unit and compare match signals
- ❏ The control bits of ACTR
- ❏ PDPINT and RESET

The outputs of the output logic for the compare units are:

- ❏ PWMx, x = 1–6

Figure 7–19. Output Logic Block Diagram (x = 1, 2, or 3; y = 1, 2, 3, 4, 5, or 6)



Output logic for PWM mode

7.6 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 7–22 on page 7-52). 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.

7.6.1 Generation of PWM Outputs with Event Manager

Each of the three compare units, together with GP timer 1, 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 the EV2 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 (ACTR) 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 EV2 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.

7.6.2 Register Setup for PWM Generation

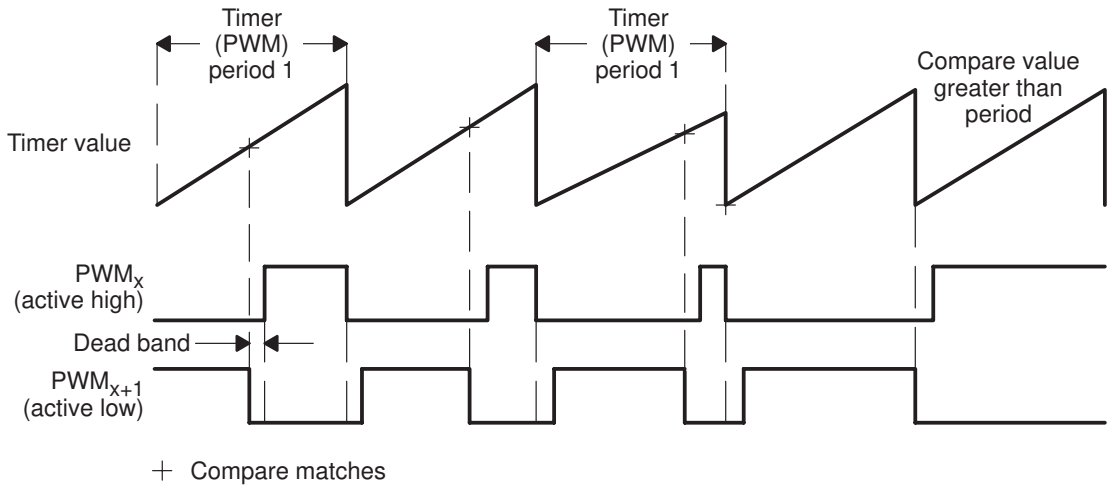
All three kinds of PWM waveform generations with compare units and associated circuits require configuration of the same Event Manager registers. The setup process for PWM generation includes the following steps:

- ❏ Setup and load ACTR.
- ❏ Setup and load DBTCON, if dead-band is to be used.
- ❏ Initialize CMPRx.
- ❏ Setup and load COMCON.
- ❏ Setup and load T1CON to start the operation.
- ❏ Rewrite CMPRx with newly determined values.

7.6.3 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 7–20. The width of each pulse can only be changed from one side of the pulse.

Figure 7–20. Asymmetric PWM waveform generation with compare unit and PWM Circuits (x = 1, 3, or 5)



+ Compare matches

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 COMCON 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 DBTCON(11:8). This is the period for the 4-bit dead-band timers. One dead-band value is used for all PWM output channels.

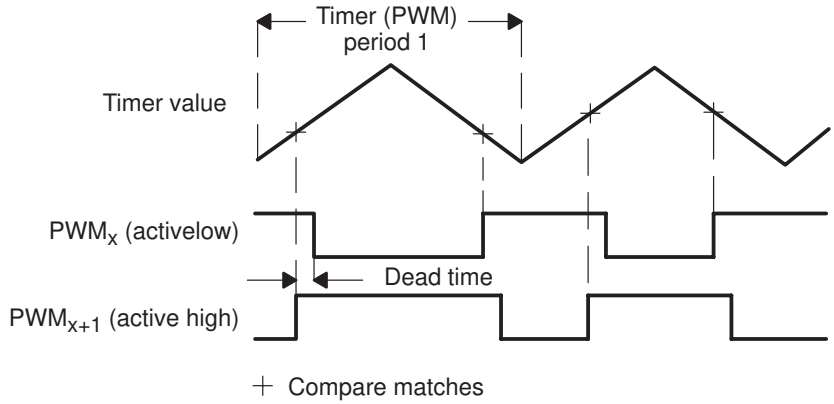
By proper configuration of ACTR with software, a normal PWM signal can be generated on one output associated with a compare unit while the other is held low (off) or high (on), at the beginning, middle, or end of a PWM period. Such software controlled flexibility of PWM outputs is particularly useful in switched reluctance motor control applications.

After GP timer 1 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.

7.6.4 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 7–21 shows two examples of symmetric PWM waveforms.

Figure 7–21. 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 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 itmakes 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.

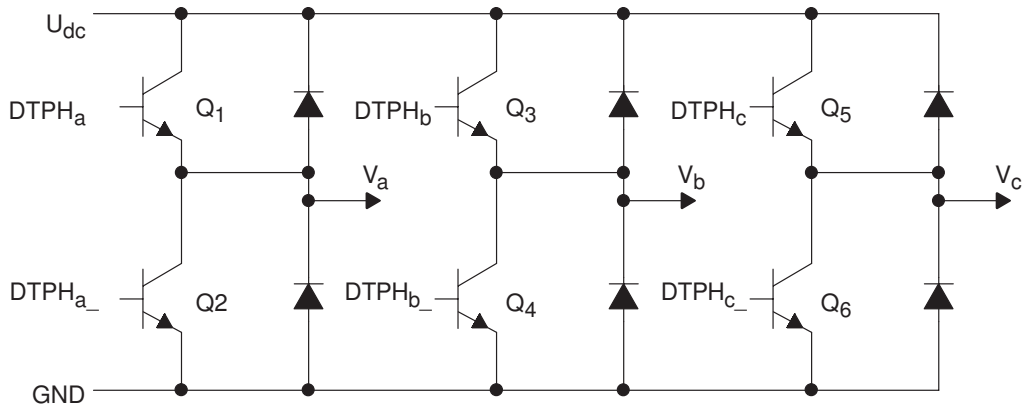
7.7 Space Vector PWM

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.

7.7.1 3-Phase Power Inverter

The structure of a typical 3-phase power inverter is shown in Figure 7–22, 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 7–22. 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 7–9, on page 7-53, where a, b, and c represent the values of $DTPH_a$, $DTPH_b$, and $DTPH_c$, respectively.

Table 7–9. 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

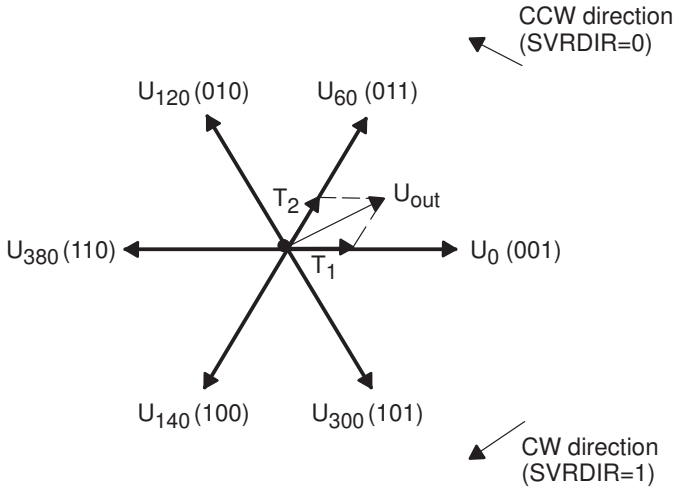
Note: 0 = off, and 1 = on

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 non-zero vectors and two zero vectors. The non-zero 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 7–23 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 7–23. 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.

7.7.2 Space Vector PWM Waveform Generation with Event Manager

The EV2 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 ACTR to define the polarity of the compare output pins
- ❑ Configure COMCON to enable compare operation and space vector PWM mode, and set the reload condition for CMPRx to be underflow
- ❑ Put GP timer 1 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 ACTR[14–12] and 1 in ACTR[15], or the switching pattern of U_{x+60} in ACTR[14–12] and 0 in ACTR[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 EV2 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 ACTR[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 ACTR[15] is 1, or to the pattern of U_y if ACTR[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 7–24 shows examples of the symmetric space vector PWM waveforms.

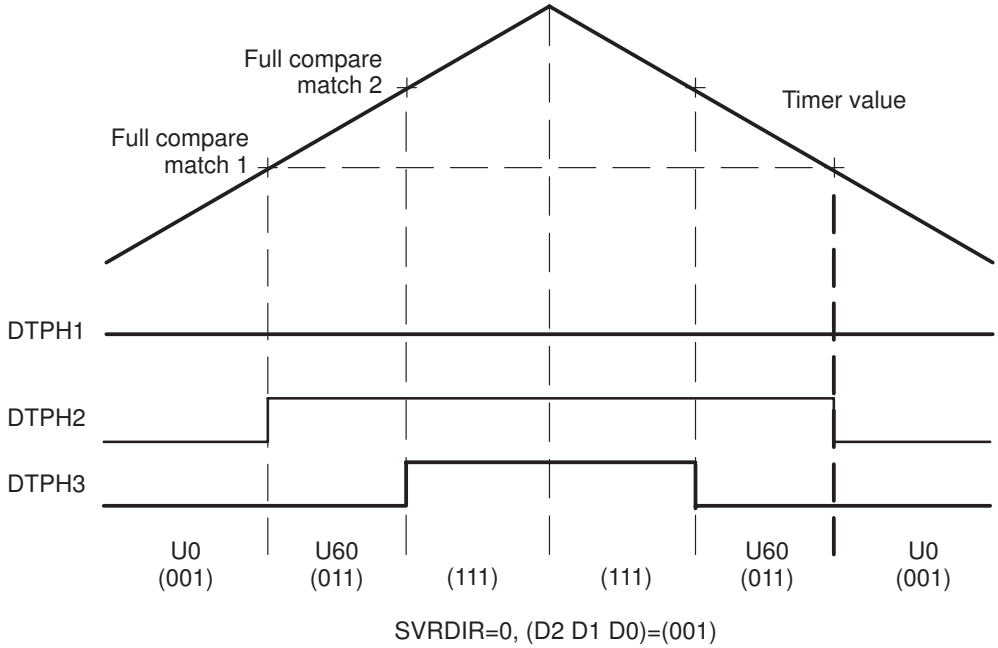
The Unused Compare Register

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.

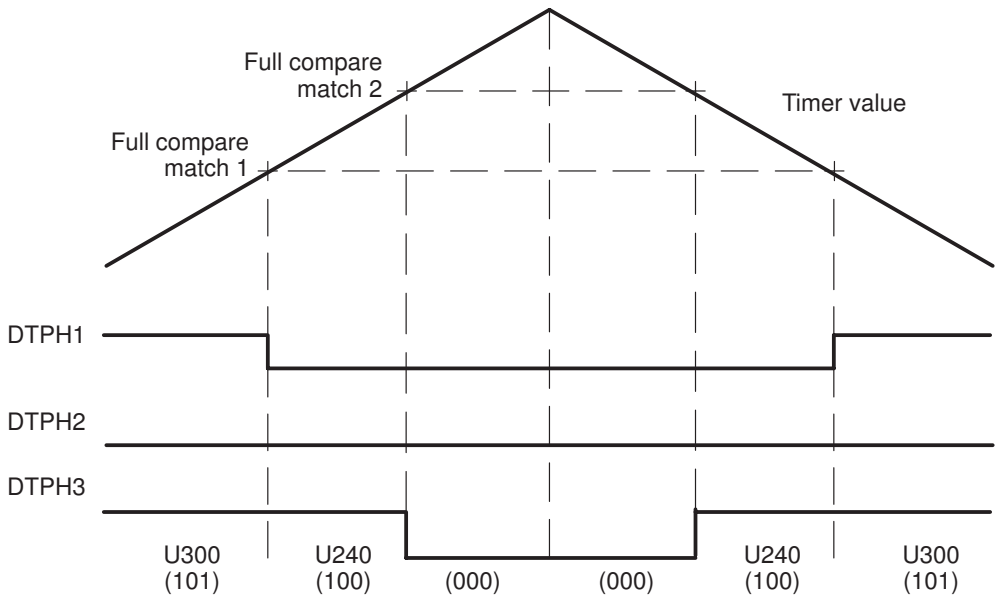
7.7.3 Space Vector PWM Boundary Conditions

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 7–24. Symmetric Space Vector PWM Waveforms



SVRDIR=0, (D2 D1 D0)=(001)

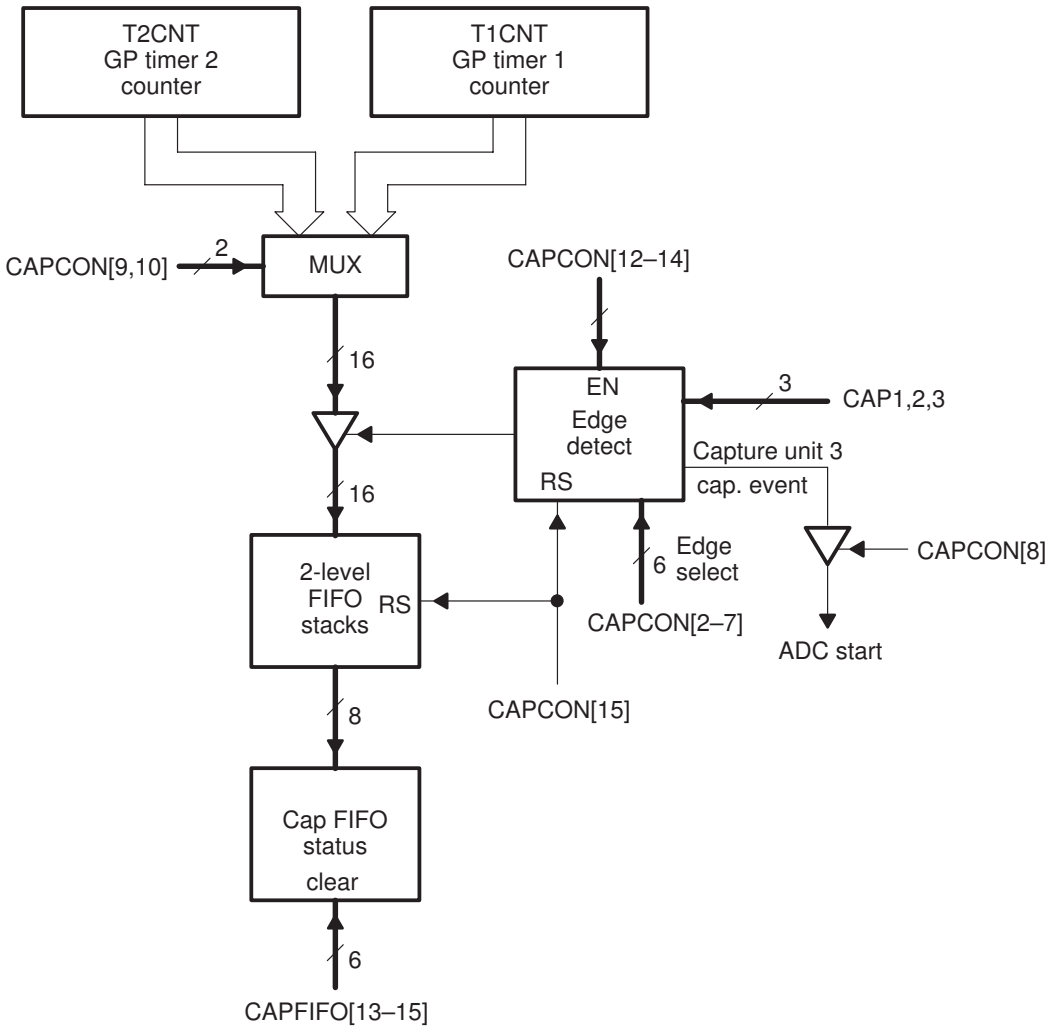


SVRDIR=1, (D2 D1 D0)=(101)

7.8 Capture Units

Capture units enable logging of transitions on capture input pins. There are three capture units: Capture Units 1, 2, and 3 and each is associated with a capture input pin. Each capture unit can choose GP timer 2 or 1 as its time base. The value of GP timer 2 or 1 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 7–25 shows a block diagram of a capture unit.

Figure 7–25. Capture Units Block Diagram



7.8.1 Capture Unit Features

Capture units have the following features:

- ❏ One 16-bit capture control register, CAPCON (RW)
- ❏ One 16-bit capture FIFO status register, CAPFIFO
- ❏ Selection of GP timer 1 or 2 as the time base
- ❏ Three 16-bit 2-level-deep FIFO stacks, one for each capture unit.
- ❏ Three Schmitt-triggered capture input pins, CAP1, CAP2, and CAP3, 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 can also be used as QEP inputs to QEP circuit)
- ❏ User-specified transition (rising edge, falling edge, or both edges) detection
- ❏ Three maskable interrupt flags, one for each capture unit

7.8.2 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 CAPFIFO 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

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.

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 CAPFIFO 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 CAPCON.

7.8.3 Capture Unit Registers

The operation of the capture units is controlled by two 16-bit control registers, CAPCON and CAPFIFO. T1CON and T2CON registers are also used to control the operation of the capture units since the time base for capture circuits is provided by either GP timer 1 or 2. Additionally, CAPCON is also used to control the operation of the QEP circuit. Table 7–4 on page 7-9 shows the addresses of these registers.

Capture Control Register (CAPCON)

Figure 7–26. Capture Control Register (CAPCON) — Address 7420h

15	14–13	12	11	10	9	8
CAPRES	CAPQEPN	CAP3EN	Reserved	CAP3TSEL	CAP12TSEL	CAP3TOADC
RW-0	RW-0	RW-0	R-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; value following dash (-) = 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 FIFO Status Register (CAPFIFO)

CAPFIFO contains the status bits for each of the three FIFO stacks of the capture units. The bit description of CAPFIFO is given in Figure 7–27. If a write occurs to the CAPnFIFO status bits at the same time as they are being updated (because of a capture event), the write data takes precedence.

Figure 7–27. Capture FIFO Status Register (CAPFIFO) — 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; value following dash (-) = 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.

7.8.4 Capture Unit FIFO Stacks

Each capture unit has a dedicated 2-level-deep FIFO stack. The top stack consistes of CAP1FIFO, CAP2FIFO, and CAP3FIFO. The bottom stack consists of CAP1FBOT, CAP2FBOT, and CAP3FBOT. 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.

7.8.5 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.

7.9 Quadrature Encoder Pulse (QEP) Circuit

The 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/QEP0 and CAP2/QEP1. 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 pins is disabled.

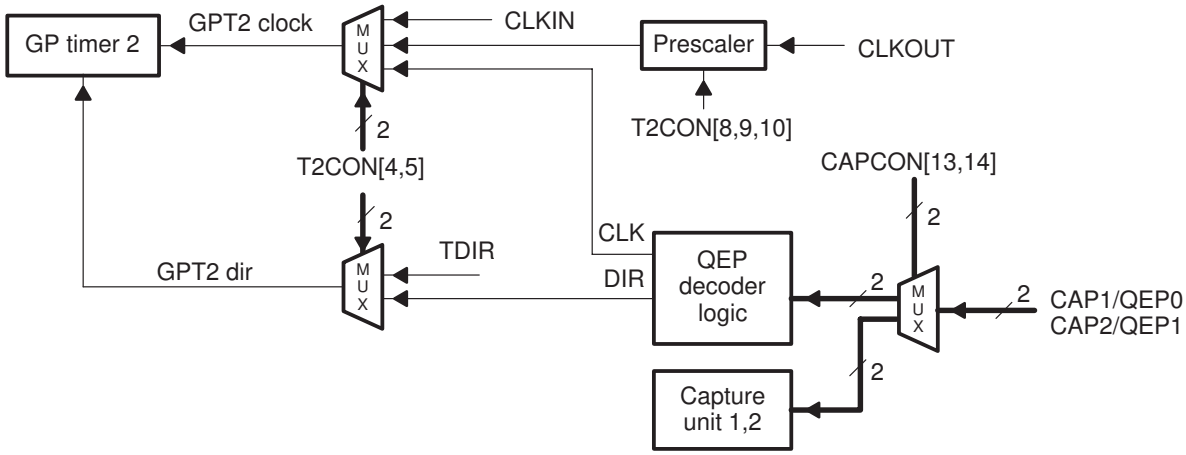
7.9.1 QEP Pins

The two QEP input pins are shared between capture units 1 and 2, and the QEP circuit. Proper configuration of CAPCON bits is required to enable the QEP circuit and disable capture units 1 and 2, thus assigning the two associated input pins for use by the QEP circuit.

7.9.2 QEP Circuit Time Base

The time base for the QEP circuit is provided by GP timer 2. The GP timer must be put in directional up/down-count mode with the QEP circuit as the clock source. Figure 7–28 shows the block diagram of the QEP circuit.

Figure 7–28. Quadrature Encoder Pulse (QEP) Circuit Block Diagram



7.9.3 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

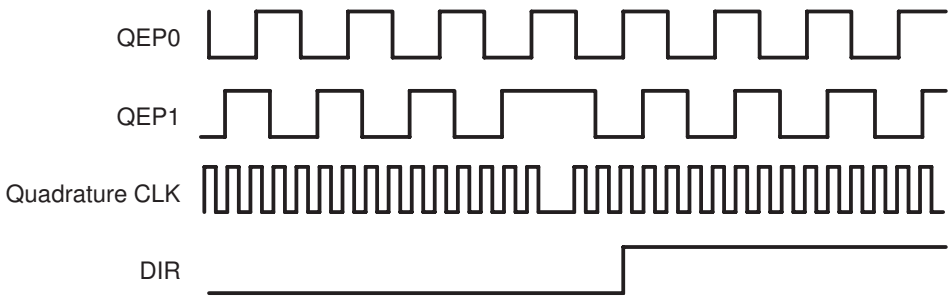
The direction detection logic of the QEP circuit in the EV2 module determines which one of the sequences is the leading sequence. It then generates a direction signal as the direction input to GP timer 2. The timer counts up if CAP1/QEP0 input is the leading sequence, and counts down if CAP2/QEP1 is 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 is four times that of each input sequence. This quadrature-clock is connected to the clock input of GP timer 2.

Quadrature Encoded Pulse Decoding Example

Figure 7–29 shows an example of quadrature encoded pulses and the derived clock and counting direction.

Figure 7–29. Quadrature Encoded Pulses and Decoded Timer Clock and Direction



7.9.4 QEP Counting

GP timer 2 always starts counting from its current value. A desired value can be loaded to GP timer 2's counter prior to enabling the QEP mode. When the QEP circuit is selected as the clock source, the timer ignores the TDIR and TCLKIN 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.

7.9.5 Register Setup for the QEP Circuit

To start the operation of the QEP circuit:

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 CAPCON to enable the QEP circuit.

7.10 Event Manager (EV2) Interrupts

EV2 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 EV2 interrupt group. Table 7–10 shows all EV2 interrupts and their priority and grouping. There is an interrupt flag register and an interrupt mask register for each EV2 interrupt group: EVIFRA, EVIFRB, and EVIFRC, and EVIMRA, EVIMRB, and EVIMRC. A flag in EVIFRx (x = A, B, or C) is masked (will not generate a peripheral interrupt request) if the corresponding bit in EVIMRx is 0.

7.10.1 EV2 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 7–10. Event Manager (EV2) Interrupts

Group	Interrupt	Priority within group	Vector (ID)	Description/Source	INT
	PDPINT	1 (highest)	0020h	Power Drive Protection Interrupt	1
A	CMP1INT	2	0021h	Compare Unit 1 compare interrupt	
	CMP2INT	3	0022h	Compare Unit 2 compare interrupt	
	CMP3INT	4	0023h	Compare Unit 3 compare interrupt	
	T1PINT	5	0027h	GP timer 1 period interrupt	2
	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	
	T2CINT	2	002Ch	GP timer 2 compare interrupt	3
	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	
	CAP2INT	2	0034h	Capture Unit 2 interrupt	4
	CAP3INT	3	0035h	Capture Unit 3 interrupt	

Table 7–11. 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 EV2 module, the corresponding interrupt flag in one of the EV2 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 EVIMRx 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).

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.

7.10.2 EV2 Interrupt Flag Registers

Addresses of EV2 interrupt registers are shown in Table 7–5 on page 7-10. 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 EVIFRx are readable registers, occurrence of an interrupt event can be monitored by software polling the appropriate bit in EVIFRx when the interrupt is masked.

EV2 Interrupt Flag Register A (EVIFRA)

Figure 7–30. EV2 Interrupt Flag Register A (EVIFRA) — Address 742Fh

15–11	10	9	8
Reserved	T1OFINT	T1UFINT	T1CINT
R-0	RW-0	RW-0	RW-0

7	6–4	3	2	1	0
T1PINT	Reserved	CMP3INT	CMP2INT	CMP1INT	PDPINT
RW-0	R-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access; W = Write access; value following dash (-) = value after reset

Bits 15–11 **Reserved**. Reads return zero; writes have no effect.

Bit 10 **T1OFINT**. GP timer 1 overflow interrupt.

Read: 0 Flag is reset
1 Flag is set
Write: 0 No effect
1 Reset flag

Bit 9 **T1UFINT**. GP timer 1 underflow interrupt.

Read: 0 Flag is reset
1 Flag is set
Write: 0 No effect
1 Reset flag

Bit 8 **T1CINT**. GP timer 1 compare interrupt.

Read: 0 Flag is reset
1 Flag is set
Write: 0 No effect
1 Reset flag

Bit 7 **T1PINT.** GP timer 1 period interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 6–4 **Reserved**. Reads return zero; writes have no effect.

Bit 3 **CMP3INT**. Compare 3 interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 2 **CMP2INT**. Compare 2 interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 1 **CMP1INT**. Compare 1 interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

Bit 0 **PDPINT**. Power drive protection interrupt.

Read:	0	Flag is reset
	1	Flag is set
Write:	0	No effect
	1	Reset flag

EV2 Interrupt Flag Register B (EVIFRB)

Figure 7–31. EV2 Interrupt Flag Register B (EVIFRB) — Address 7430h

15–4	3	2	1	0
Reserved	T2OFINT	T2UFINT	T2CINT	T2PINT
R-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access; W = Write access; value following dash (-) = value after reset

Bits 15–4 **Reserved**. Reads return zero; writes have no effect.

Bit 3 **T2OFINT**. GP timer 2 overflow interrupt.

Read: 0 Flag is reset
1 Flag is set
Write: 0 No effect
1 Reset flag

Bit 2 **T2UFINT**. GP timer 2 underflow interrupt.

Read: 0 Flag is reset
1 Flag is set
Write: 0 No effect
1 Reset flag

Bit 1 **T2CINT**. GP timer 2 compare interrupt.

Read: 0 Flag is reset
1 Flag is set
Write: 0 No effect
1 Reset flag

Bit 0 **T2PINT**. GP timer 2 period interrupt.

Read: 0 Flag is reset
1 Flag is set
Write: 0 No effect
1 Reset flag

EV2 Interrupt Flag Register C (EVIFRC)

Figure 7–32. EV2 Interrupt Flag Register C (EVIFRC) — Address 7431h

15–3	2	1	0
Reserved	CAP3INT	CAP2INT	CAP1INT
R-0	RW-0	RW-0	RW-0

Note: R = Read access; W = Write access; value following dash (-) = value after reset

Bits 15–3 **Reserved**. Reads return zero; writes have no effect.

Bit 2 **CAP3INT**. Capture 3 interrupt.

Read: 0 Flag is reset
1 Flag is set
Write: 0 No effect
1 Reset flag

Bit 1 **CAP2INT**. Capture 2 interrupt.

Read: 0 Flag is reset
1 Flag is set
Write: 0 No effect
1 Reset flag

Bit 0 **CAP1INT**. Capture 1 interrupt.

Read: 0 Flag is reset
1 Flag is set
Write: 0 No effect
1 Reset flag

EV2 Interrupt Mask Register A (EVIMRA)

Figure 7–33. EV2 Interrupt Mask Register A (EVIMRA) — 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	PDPINT enable
RW-0	R-0	RW-0	RW-0	RW-0	RW-1

Note: R = Read access; W = Write access; value following dash (-) = 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

Bit 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 **PDPINT ENABLE.** This is enabled (set to 1) following reset.

0 Disable
1 Enable

EV2 Interrupt Mask Register B (EVIMRB)

Figure 7–34. EV2 Interrupt Mask Register B — 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; value following dash (-) = 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

EV2 Interrupt Mask Register C (EVIMRC)

Figure 7–35. EV2 Interrupt Mask Register C — 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; value following dash (-) = 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

Chapter 8

Analog-to-Digital Converter (ADC)

This chapter contains a general description of the pseudo-dual 10-bit analog-to-digital converter (ADC) module. This peripheral has identical functionality to the dual ADCs on the ’C240 device; however, instead of having two ADC converters with a 6 µs conversion time, it has one 600 ns converter (12 × 50 ns period clocks) with control logic acting as if there are two converters: pseudo-ADC#1 and pseudo-ADC#2. However, because of various synchronization delays, a single conversion takes about 1 µs with a 20-MHz clock and a prescale of 1.

When simultaneous conversion is requested, two consecutive conversions are performed 850 ns apart for a total simultaneous dual-conversion time of 1700 ns. The result for pseudo-ADC#1 is converted first. With prescale factors greater than one, the convert-time formula becomes more complex. For more information, see section 8.2.3, *Analog Signal Sampling/Conversion,* on page 8-6.

The ADC has eight analog inputs.

Apart from the fact that this ADC peripheral uses one converter to simulate the operation of two converters, the only other difference between this module and that on the ’C240 is the pre-scale divider values for the ADC clock.

Topic **Page**

8.1 ADC Overview

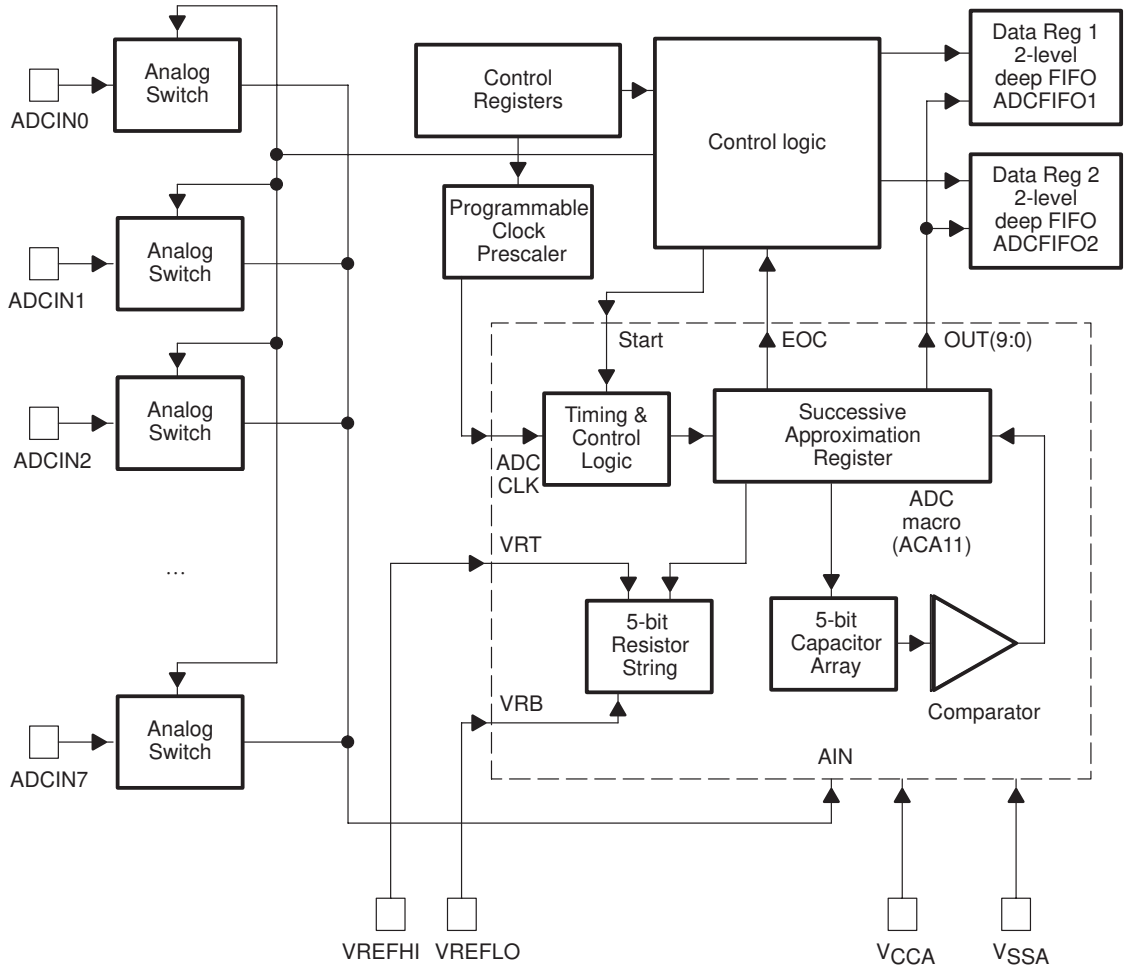
This pseudo-dual ADC is based around a 10-bit string/capacitor converter with the switched capacitor string providing an inherent sample-and-hold function. (Note: There is only one converter with only one inherent sample and hold circuit). This peripheral behaves as though there are two analog converters, ADC#1 and ADC#2, but it uses only one converter. This feature makes the ADC software compatible with the 'C240's ADC, and also allows two values (ex. voltage and current) to be converted almost simultaneously with one conversion request.)

Minimum conversion time for the ADC unit is 850 ns when the converter is clocked at 20 MHz. External high and low reference voltage must be supplied. The upper and lower references can be set to any voltage less than or equal to 5V by connecting VREFHI and VREFLO to external reference voltages.

The ADC module, shown in Figure 8–1, has the following features:

- 8 analog inputs, ADCIN0–ADCIN7
- Almost simultaneous measurement of two analog inputs, 800ns apart
- Single conversion and continuous conversion modes
- Conversion can be started by software, an internal event, and/or an external event
- VREFHI and VREFLO (high- and low-voltage) reference inputs
- Two-level-deep digital result registers that contain the digital values of completed conversions
- Two programmable ADC module control registers
- Programmable clock prescaler
- Interrupt or polled operation
- Programmable priority interrupt requests: either a high or a low priority interrupt request can be generated

Figure 8–1. TMS320C24x Pseudo-Dual ADC Module



8.2 ADC Operation

The digital result of the conversion process for the 10-bit ADC is approximated by the following equation:

$$Digital\ Result = 1023 \times \frac{Input\ Voltage - V_{REFLO}}{V_{REFHI} - V_{REFLO}}$$

8.2.1 ADC Module Pin Descriptions

Pin	Description
V_{CCA}	Analog 5V supply
V_{SSA}	Analog ground
VREFHI	High analog reference voltage
VREFLO	Low analog reference voltage
ADCIN0–ADCIN7	Analog Inputs for conversion
ADCSOC/XINT2/IOPD1	External start-of-conversion pin

- The analog supply pins, V_{CCA} and V_{SSA}, are separate from any digital voltage supply pins. Standard isolation techniques must be used to isolate the digital and analog supply pins. Analog power lines connected to V_{CCA} and V_{SSA} should be as short as possible with the two lines properly decoupled. All other standard noise reduction techniques should be used to ensure accurate conversion.

- The reference voltages for the ADC module must satisfy the following condition:

$$0 \leq V_{REFLO} < V_{REFHI} \leq 5V$$

- The analog input voltage for conversion should be bounded by the values V_{REFHI} and V_{REFLO}:

$$V_{REFLO} \leq V_{ADCIN} \leq V_{REFHI}$$

- When the ADCSOC/XINT2/IOPD1 pin is not used as a GPIO (that is, as IOPD1), it can be used as XINT2 and/or ADCSOC depending on what is enabled. (XINT2 is enabled by writing a 1 to bit 0 of XINT2CR; ADCSOC is enabled by writing a 1 to bit 9 of ADCTRL2.) Note that while the polarity of XINT2 is programmable, ADCSOC is recognized for a rising edge only.

8.2.2 ADC Module Operational Modes

Functions of the ADC module include:

- Two input channels (one for each pseudo ADC unit) can be sampled and converted almost simultaneously (850 ns apart with 20-MHz clock).
- Each pseudo-ADC unit can perform single or continuous S/H and conversion operations. When in continuous dual-conversion mode, each pseudo-ADC generates a result every 1700ns (with a 20-MHz clock and a prescale factor of 1). When in continuous single-conversion mode, the selected pseudo-ADC generates a result every 1 μs (under the same clock conditions).
- ADC#1 and ADC#2 have two 2-Level-deep FIFO result registers.
- Conversion can be started by software, an external signal transition on a device pin (ADCSOC), or by certain event manager events.
- The ADC control register is double buffered (with a shadow register) and can be written to at any time. A new conversion can start either immediately or when the previous conversion process is completed.
- In single-conversion mode, an interrupt flag is set and the peripheral interrupt request (PIRQ) is generated at the end of each conversion, if it is unmasked/enabled.
- In dual-conversion mode, an interrupt flag is set and the peripheral interrupt request (PIRQ) is generated at the end of each pair of conversions, if it is unmasked/enabled.
- The result of previous conversions stored in data register 1 (for ADC#1) and in data register 2 for (ADC#2) is lost when a third result is stored in the 2-word-deep data FIFO.

8.2.3 Analog Signal Sampling/Conversion

The analog-to-digital conversion circuit requires a clock (ADCCLK) with a frequency of 20 MHz or less. The analog part of the conversion itself takes about 12 ADCCLK cycles, but additional clock cycles are required for synchronization between the analog converter and the digital control logic.

To allow this peripheral to be used in devices with clock rates other than 20MHz, there is a system clock prescaler to divide the system clock down to an acceptable rate below 20 MHz. The prescaler allows the ADC to be used in continuous-conversion mode at lower sample rates.

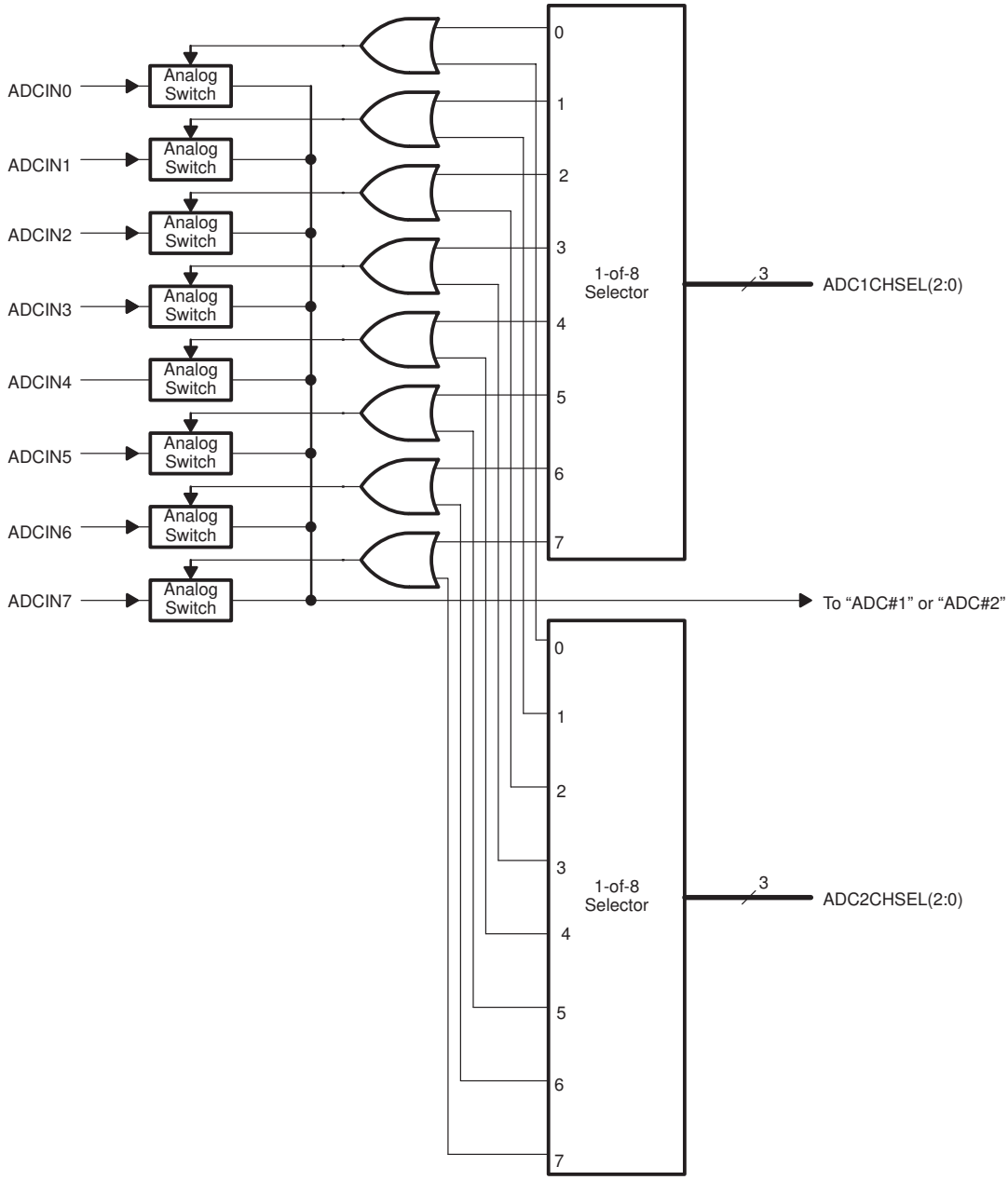
Table 8–1. Prescaler Values

Clock Prescale Bits			Prescale Value	ADC Clock
Bit 2	Bit 1	Bit 0		
0	0	0	1	CLKOUT/1
0	0	1	2	CLKOUT/2
0	1	0	4	CLKOUT/4
0	1	1	8	CLKOUT/8
1	0	0	12	CLKOUT/12
1	0	1	16	CLKOUT/16
1	1	0	24	CLKOUT/24
1	1	1	32	CLKOUT/32

8.2.4 Analog Input Selection

Input pins ADCIN0 through ADCIN7 are associated with both pseudo-ADC modules. In this implementation, there are eight input select signals leaving the ADC module to control up to eight analog switch inputs. The select signal for ADC#1 input 0 is ORed with the select signal for ADC#2 input 0, etc. (See Figure 8–2, *Input Selection With Eight or Less Analog Inputs*.) The digital result of any ADCINn channel selected by ADC2CHSEL bits (bits 6–4 of the ADCTRL1 register) is stored in the ADCFIFO2 register. Similarly, the digital result of any ADCINn channel selected by ADC1CHSEL bits (bits 3–1 of the ADCTRL1 register) is stored in the ADCFIFO1 register.

Figure 8–2. Input Selection With Eight or Less Analog Inputs



8.2.5 Interrupts

In single-conversion mode, the interrupt flag is set at the end of every conversion. In dual-conversion mode, the interrupt flag is set at the end of every pair of conversions. Channel 1 is converted first, and then channel 2. Hence, the interrupt flag is set when the conversion of channel 2 is complete.

The ADC can assert one peripheral interrupt request (PIRQ) to the peripheral interrupt expansion (PIE) controller. This can be a high priority request or a low priority request. The priority is determined by the INTPRI bit, which is bit 11 of the ADCTRL2 register. A 0 in INTPRI makes the interrupt request a high priority, and a 1 makes the request a low priority. The same interrupt vector is used with either priority. The interrupt vector is discussed in Chapter 3.

There is an interrupt flag bit in ADC Control Register 1 (ADCTRL1) that gets set at the end of an A/D conversion. If the interrupt enable bit in ADCTRL1 is set, the peripheral asserts it's interrupt request (PIRQ) to the PIE controller. If the interrupt enable is not set when the conversion completes (but is set at a later time), and if the interrupt flag has not been cleared, then PIRQ is asserted. The peripheral interrupt request and the interrupt flag inside the PIE that generate INTx interrupts are cleared in response to a peripheral interrupt acknowledge.

8.3 ADC Registers

Table 8–2 lists the addresses of the ADC registers.

Table 8–2. Addresses of ADC Registers

Address	Name	Description
7032h	ADCTRL1	ADC Control Register 1
7034h	ADCTRL2	ADC Control Register 2
7036h	ADCFIFO1	2-Level-Deep Data Register FIFO for pseudo-ADC#1
7038h	ADCFIFO2	2-Level-Deep Data Register FIFO for pseudo-ADC#2

8.3.1 Shadowed Bits

Many of the control register bits are described as shadowed. This means that changing the value of one of these bits does not take effect until the current conversion is complete.

8.4 ADC Control Registers

There are two ADC control registers: ADC control register 1 and ADC control register 2.

8.4.1 ADC Control Register 1

ADC control register 1 controls the following:

- ❑ Start of conversion
- ❑ Channel selection
- ❑ ADC module enable/disable function
- ❑ Interrupt enable
- ❑ Emulation operation and end of conversion

Figure 8–3. ADC Control Register 1 (ADCTRL1) — Address 7032h

15	14	13	12	11	10	9	8
Suspend Soft	Suspend Free	ADCIMSTART	ADC2EN	ADC1EN	ADCCONRUN	ADCINTEN	ADCINTFLAG
RW-0	RW-0	RW-0	SRW-0	SRW-0	SRW-0	SRW-0	RC-0

7	6–4	3–1	0
ADCEOC	ADC2CHSEL	ADC1CHSEL	ADCSOC
R-0	SRW-0	SRW-0	SRW-0

Note: R = Read access; W = Write access; C = Clear by writing a 1; S = Shadowed; value following dash (-) = value after reset

Bits 15,14 Soft & Free bits

These bits determine what happens 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 (in this case, the current conversion) is complete.

Bit 15 Soft	Bit 14 Free	
0	0	Stop conversion immediately
1	0	Complete current conversion before stopping
X	1	Free run, continue operation regardless of suspend

These bits are not shadowed.

Bit 13 **ADCIMSTART**. Start ADC conversion immediately.

Writing a 1 to this bit starts a conversion immediately (the currently active conversion aborts). This bit is not shadowed.

0 No action

1 Immediately start conversion

Bit 12 **ADC2EN**. Enable/Disable bit for pseudo ADC#2

0 ADC#2 disabled (No sample/hold/conversion can take place; data register ADCFIFO2 will not change.)

1 ADC#2 is enabled

This bit is shadowed. This bit can be written while a previous conversion is still going on; however, changing this bit will not take effect until after the current conversion is completed.

Bit 11 **ADC1EN**. Enable/Disable bit for pseudo-ADC#1.

0 ADC#1 disabled (No sample/hold/conversion can take place; data register ADCFIFO1 will not change.)

1 ADC#1 is enabled

This bit is shadowed. This bit can be written while a previous conversion is in progress; however, changing this bit does not take effect until after the current conversion is complete.

Bit 10 **ADCCONRUN**.

This bit puts the ADC unit into continuous-conversion mode. This bit can be written while a previous conversion is in progress; however, changing this bit does not take effect until after the current conversion is complete.

0 No continuous conversion

1 Continuous conversion enabled

Bit 9 **ADCINTEN**. Enable interrupts.

If the ADCINTEN bit is set, an interrupt is requested when the ADCINTFLAG is set. This bit is cleared on reset and is shadowed. Interrupt flag ADCINTFLAG gets set at the end of a conversion to allow polling, regardless of the value of ADCINTEN.

0 Interrupt disabled

1 Interrupt enabled

Bit 8 **ADCINTFLAG**. ADC interrupt flag bit.

This bit indicates if an interrupt event has occurred. Writing a 1 clears this bit.

0	No interrupt event
1	An interrupt event has occurred

Bit 7 **ADCEOC**.

This bit is set to a one while the ADC conversion is in progress. It is cleared when the conversion is complete; that is, at the same time the ADCINTFLAG is set.

0	End of conversion
1	Conversion is in progress

Bits 6–4 **ADC2CHSEL**. Selects channels for ADC2

000	Channel 0 (ADCIN0)
001	Channel 1 (ADCIN1)
010	Channel 2 (ADCIN2)
011	Channel 3 (ADCIN3)
100	Channel 4 (ADCIN4)
101	Channel 5 (ADCIN5)
110	Channel 6 (ADCIN6)
111	Channel 7 (ADCIN7)

Bits 3–1 **ADC1CHSEL**. Selects channels for ADC1

000	Channel 0 (ADCIN0)
001	Channel 1 (ADCIN1)
010	Channel 2 (ADCIN2)
011	Channel 3 (ADCIN3)
100	Channel 4 (ADCIN4)
101	Channel 5 (ADCIN5)
110	Channel 6 (ADCIN6)
111	Channel 7 (ADCIN7)

Note:

When both ADC#1 and ADC#2 are enabled (dual conversion), each start-of-conversion operation has two results (one for ADC#1 and one for ADC#2). One result is written in ADCFIFO1 and the other in ADCFIFO2.

Bit 0 **ADCSOC**. ADC start of conversion bit.

0 No action

1 Start conversion

This bit is shadowed. Writing a 1 to this bit starts a conversion as soon as the current conversion (if one is currently active) is complete.

8.4.2 ADC Control Register 2

ADC control register 2 selects the ADC input clock prescaler and conversion mode, and shows the ADC FIFO status.

Figure 8–4. ADC Control Register 2 (ADCTRL2) — Address 7034h

15	14	13	12	11	10	9	8
Reserved	IM	EVSOCP	EXTSOCP	INTPRI	ADCEVSOC	ADCEXTSOC	Reserved
R-0	W-0	SRC-0	SRC-0	RW-0	SRW-0	SRW-0	R-0

7–6	5	4–3	2–0
ADCFIFO2	Reserved	ADCFIFO1	ADCPSCALE
R-0	R-0	R-0	SRW-0

Note: R = Read access; W = Write access; C = Clear by writing a 1; S = Shadowed; value following dash (-) = value after reset

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

Bit 14 **IM**. Interrupt Mode

0 Interrupt flag set when ADCFIFOn has at least one word

1 ADC logic sets the interrupt flag only when ADCFIFOn has two words (results) in it

Bit 13 **EVSOCP**. Event Manager Start Of Conversion Pending.

0 No Conversion Pending

1 Conversion Pending

This bit is shadowed. Write 1 to clear. Writing 0 has no effect.

Bit 12 **EXTSOCP**. External Start Of Conversion Pending.

0 No Conversion Pending

1 Conversion Pending

This bit is shadowed. Write 1 to clear; writing a 0 has no effect.

Bit 11 **INTPRI**. ADC interrupt request priority.

0 High priority

1 Low Priority

This bit is not shadowed.

Bit 10 **ADCEVSOC**. Event Manager SOC mask bit.

0 No action

1 Causes ADC conversion to be started by an Event Manager signal. The Event Manager can start a conversion on a compare register match, a period register match, or an underflow. This bit is shadowed.

Bit 9 **ADCEXTSOC**. External signal; start of conversion bit.

0 No action

1 Causes ADC conversion to be started by a signal from the ADCSOC device pin. This bit is shadowed

Bit 8 **Reserved**. Reads return zero; writes have no effect.

Bits 7–6 **ADCFIFO2**. FIFO2 status.

These two bits indicate ADC#2 data register FIFO status. Two conversion results can be stored before performing any READ operations. However, after two conversions, if the third conversion is made, the oldest result is lost. These bits are NOT shadowed.

00 FIFO2 is empty

01 FIFO2 has one result

10 FIFO2 has two results

11 FIFO2 had two results and another result was received; first result has been lost

Bit 5 **Reserved**. Reads return zero; writes have no effect.

Bits 4–3 **ADCFIFO1**. FIFO1 status.

These two bits indicate ADC#1 data register FIFO status. Two conversion results can be stored before performing any READ operations. However, after two conversions, if the third conversion is made, the oldest result is lost. These bits are NOT shadowed.

00 FIFO1 is empty

01 FIFO1 has one result

10 FIFO1 has two results

11 FIFO1 had two results and another result was received; first result has been lost

Bits 2–0 **ADCPSCALE**. ADC input clock prescaler.

These bits define the ADC clock prescale factor. The prescale values are defined in Table 8–1, *Prescaler Values,* on page 8-6.

8.4.3 ADC Digital Result Registers

The digital result registers contain a 10-bit digital result following conversion of the analog input. These are read only registers that are cleared on reset. The results are stored in a two-level FIFO. This provides the flexibility of converting two variables before reading them from the data registers. However, if a third conversion is made when there are two unread values in the FIFO, the first converted value is lost.

Figure 8–5. ADC Data Registers 1 and 2 (ADCFIFO1, 2) — Addresses 7036h and 7038h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0	0	0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Note: R = Read access; value following dash (-) = value after reset

Bits 15–6 **D9–D0**. 10 bits of analog to digital converted data.

Bits 5–0 **Reserved**. Always read as 0.

Chapter 9

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**

9.1 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.

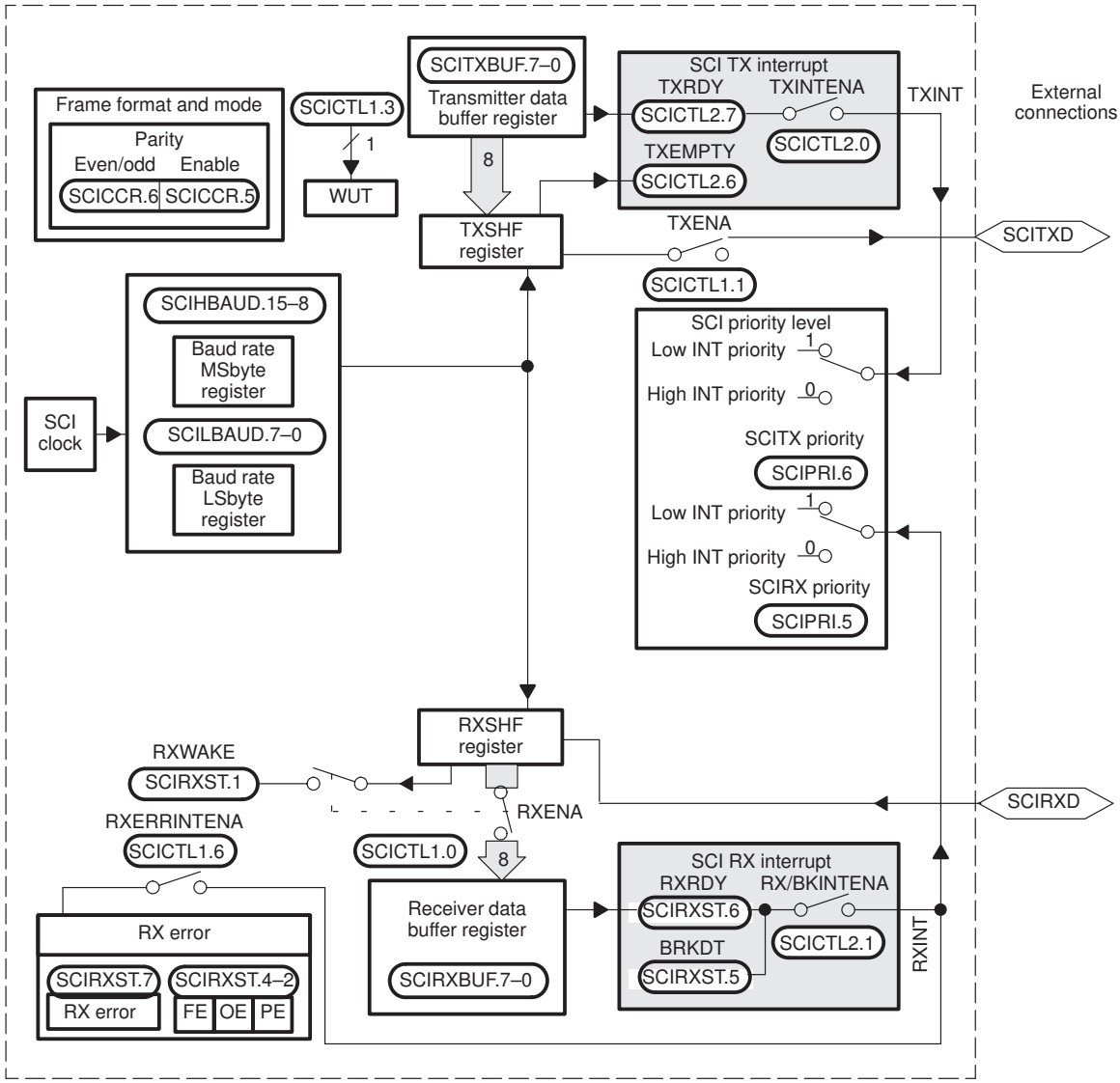
9.1.1 SCI Physical Description

The SCI module, shown in Figure 9–1, 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 20-MHz CLKOUT: 38.14 bps to 1250.0 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 9–1. SCI Block Diagram



9.1.2 Architecture

The major elements used in full duplex are shown in Figure 9–1, *SCI Block Diagram* and include:

- A transmitter (TX) and its major registers (upper half of Figure 9–1)
  - 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 9–1)
  - 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.

9.1.3 SCI Register Addresses

Table 9–1. 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.	9.6.1	9-21
7051h	SCICTL1	SCI control register 1	Controls the RX/TX and receiver error interrupt enable, TXWAKE and SLEEP functions, and the SCI software reset.	9.6.2	9-23
7052h	SCIHBAUD	SCI baud register, high bits	Stores the data (MSbyte) required to generate the bit rate.	9.6.3	9-26
7053h	SCILBAUD	SCI baud register, low bits	Stores the data (LSbyte) required to generate the bit rate.	9.6.3	9-26
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.	9.6.4	9-27
7055h	SCIRXST	SCI receiver status register	Contains seven receiver status flags.	9.6.5	9-28
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)	9.6.6.1	9-31
7057h	SCIRXBUF	SCI receiver data buffer register	Contains the current data from the receiver shift register.	9.6.6.2	9-31
7058h	—	Reserved	Reserved		
7059h	SCITXBUF	SCI transmit data buffer register	Stores data bits to be transmitted by the SCITX.	9.6.7	9-32
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.	9.6.8	9-32

9.1.4 Multiprocessor and Asynchronous Communication Modes

The SCI has two multiprocessor protocols, the *idle-line* multiprocessor mode (see section 9.3.1 on page 9-10) and the *address-bit* multiprocessor mode (see section 9.3.2 on page 9-12). 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 9.4, on page 9-14) 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

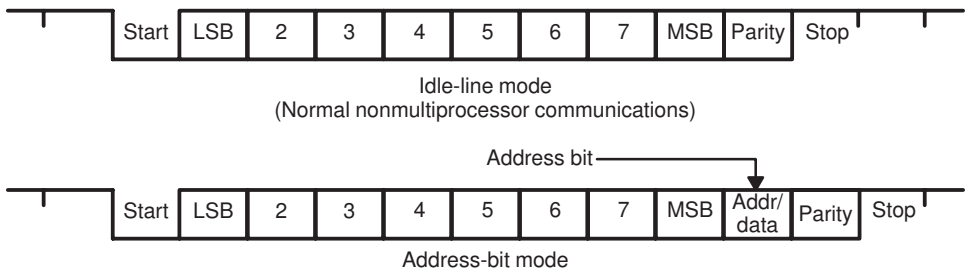
9.2 SCI Programmable Data Format

SCI data, both receive and transmit, is in NRZ (nonreturn-to-zero) format. The NRZ data format, shown in Figure 9–2, 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 9–2.

Figure 9–2. 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 9–2.

Table 9–2. 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 9–4 (page 9-22).
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.

9.3 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 address-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 9.3.1 on page 9-10) 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 9.3.2 on page 9-12) adds an extra bit (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.

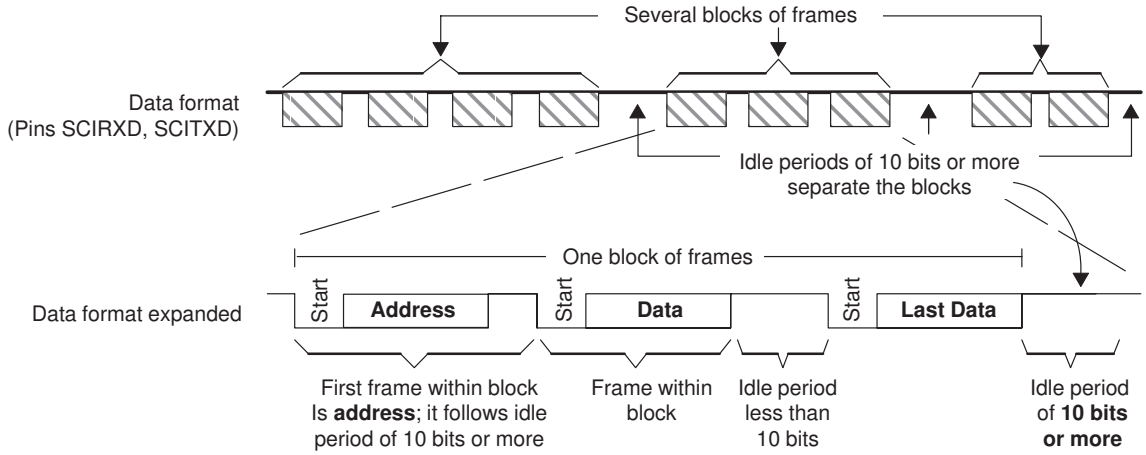
In both multiprocessor modes, the receipt sequence is:

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.

9.3.1 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 9–3 (ADDR/IDLE MODE bit is SCICCR.3).

Figure 9–3. 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.

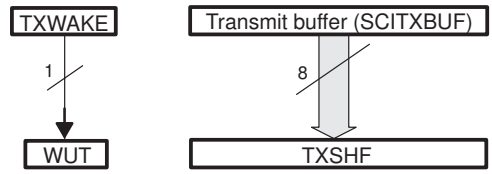
5) If the CPU *is not being addressed*, the SLEEP bit remains set. This lets the CPU continue to execute its main program without being interrupted by the SCI port until the next detection of a block start.

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. (To transmit the idle time, a “don’t care” byte has to be written to SCITXBUF after setting TXWAKE and before sending the address.)

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 9–4. (Figure 9–1, *SCI Block Diagram* on page 9-4 shows this in additional detail.)

Figure 9–4. 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 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*.

9.3.2 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 9–5, ADDR/IDLE MODE bit is 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.

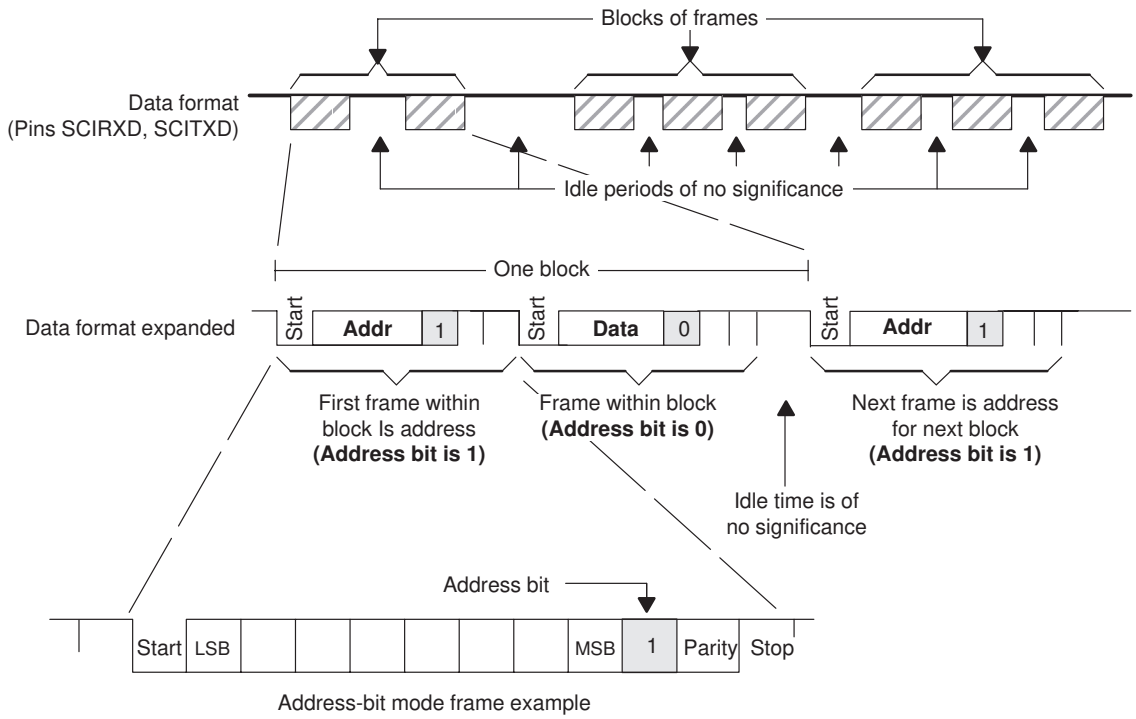
3) Since TXSHF and WUT are both double-buffered, SCITXBUF and TXWAKE can be written to immediately after TXSHF and WUT are loaded.

4) To transmit non-address frames in the block, you should 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 9–5. Address-Bit Multiprocessor Communication Format



9.4 SCI Communication Format

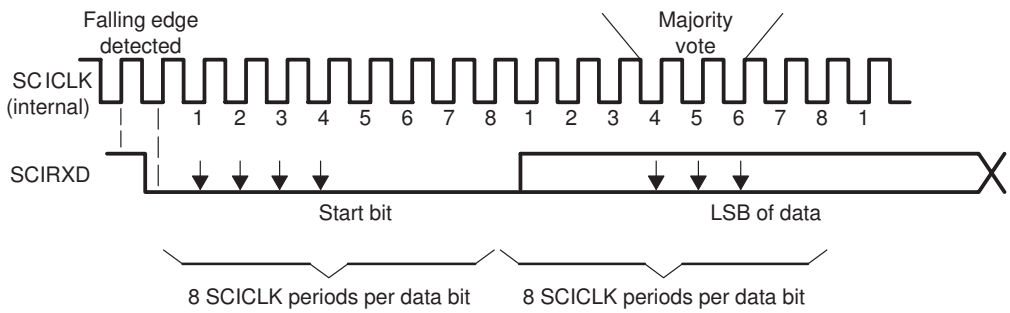
The SCI asynchronous communication format uses either single line (one-way) or two line (two-way) communications. In this mode, the frame consists of a start bit, one to eight data bits, an optional even/odd parity bit, and one or two stop bits (shown in Figure 9–6). There are *eight SCICLK periods* 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 9–6. 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 9–6 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 9–6. SCI Asynchronous Communications Format

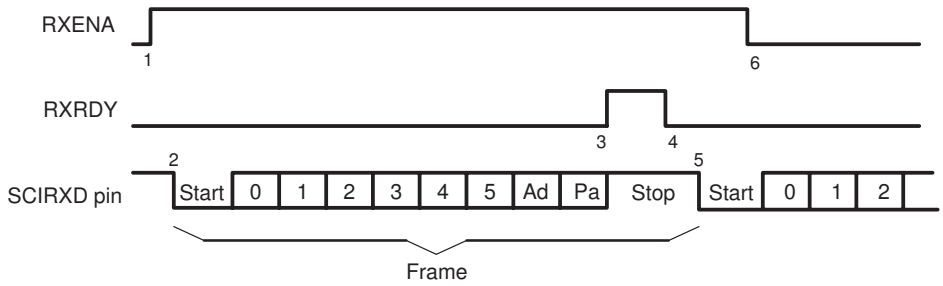


9.4.1 Receiver Signals in Communication Modes

Figure 9–7 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 9–7. 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.

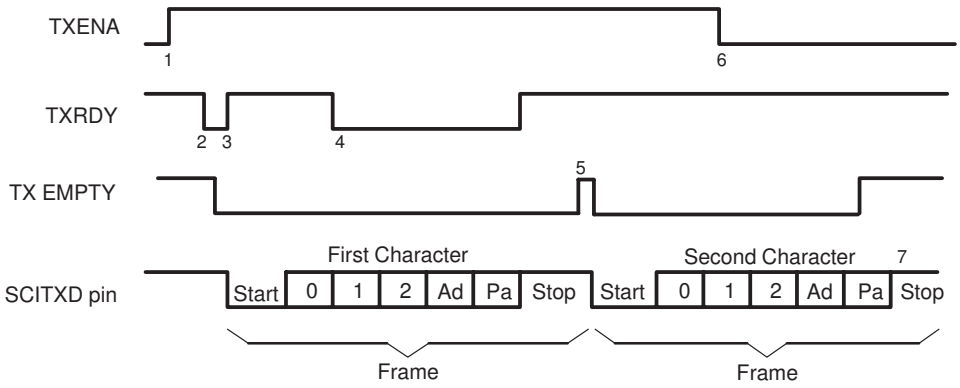
6) Bit RXENA is brought low to disable the receiver. Data continues to be assembled in RXSHF but is not transferred to the receive buffer register.

9.4.2 Transmitter Signals in Communication Modes

Figure 9–8 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 9–8. 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.

9.5 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 that 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

Interrupt generation due to the RXRDY and BRKDT bits are controlled by RX/BK_INT_ENA bit (SCICTL2.1).

Interrupt generation due to the RX_ERROR bit is controlled by RX_ERR_INT_ENA bit (SCICTL1.6).

9.5.1 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 9.6.3, *Baud-Select Registers*, for the formula to use to calculate the SCI asynchronous baud.

Table 9–3. Asynchronous Baud Register Values for Common SCI Bit Rates

	Device Clock Frequency, 20 MHz		
Ideal Baud	**BRR**	**Actual Baud**	**% Error**
2400	1041	2399	–0.04
4800	520	4798	–0.04
8192	304	8197	0.06
9600	259	9615	0.16
19200	129	19231	0.16
38400	64	38461	0.16

Note: The maximum CLKOUT frequency for ’24x devices is 20MHz.

9.6 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 9–9, and described in the sections that follow.

Figure 9–9. SCI Control Registers

Address Offset	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

9.6.1 SCI Communication Control Register

The SCI communication control (SCICCR) register defines the character format, protocol, and communications mode used by the SCI.

Figure 9–10. 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; value following dash (–) = 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 9–4 lists the bit values and character lengths for SCI CHAR2-0 bits.

Table 9–4. 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

9.6.2 SCI Control Register 1

The SCI control register 1 controls the receiver/transmitter enable, TXWAKE and SLEEP functions, and the SCI software reset.

Figure 9–11. 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, value following dash (–) = 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 9–5 lists the affected flags.

Table 9–5. 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.

9.6.3 Baud-Select Registers

The values in the baud-select registers (SCIHBAUD and SCILBAUD) specify the baud rate for the SCI.

Figure 9–12. SCI 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, value following dash (–) = value after reset

Figure 9–13. SCI 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, value following dash (–) = 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 $1 \leq \text{BRR} \leq 65535$. 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.

9.6.4 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 9–14. 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; value following dash (–) = 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

9.6.5 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 9–16 on page 9-30 shows the relationships between several of the register's bits.

Figure 9–15. 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; value following dash (–) = 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

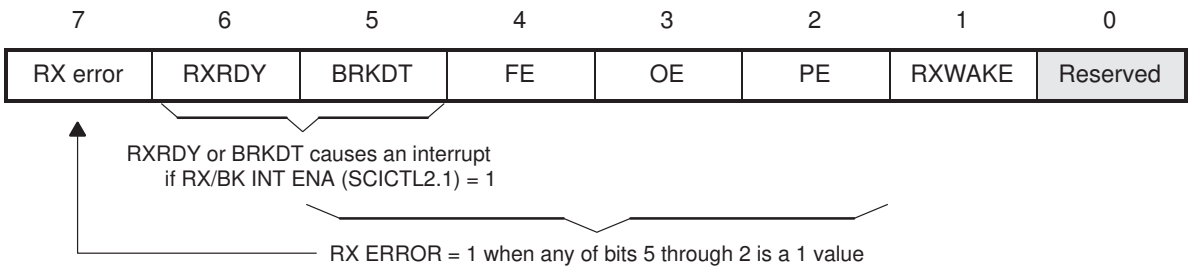
Bit 1 **RXWAKE**. Receiver wakeup-detect flag.

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 9–16. Register SCIRXST Bit Associations — Address 7055h



9.6.6 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.

9.6.6.1 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 9–17. SCI 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; value following dash (–) = value after reset (x = indeterminate)

9.6.6.2 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 9–18. SCI 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; value following dash (–) is value after reset (x = indeterminate)

9.6.7 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 9–19. 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; value following dash (–) = value after reset

9.6.8 Priority Control Register

The priority control register (SCIPRI) 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 9–20. 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; value following dash (–) = 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 10

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 zeros when read.

Topic **Page**

10.1 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.

10.1.1 SPI Physical Description

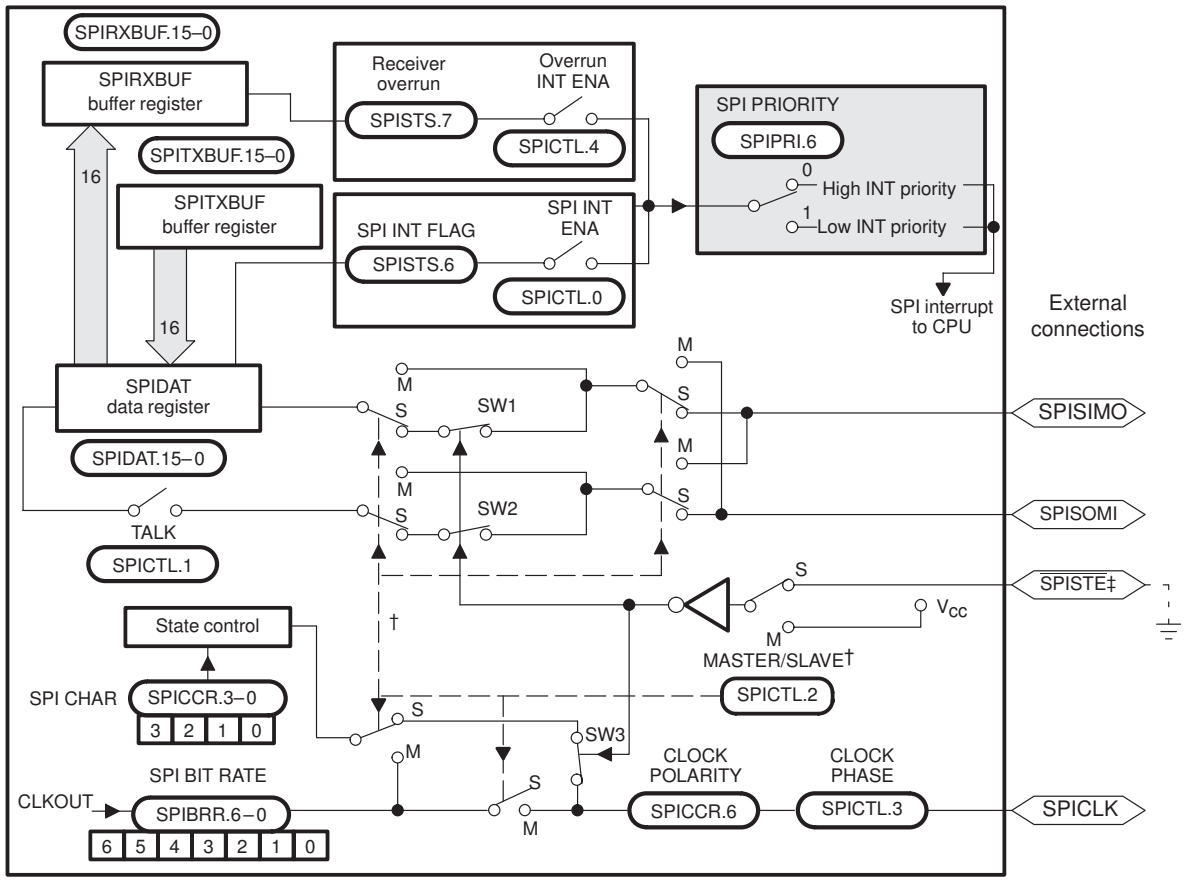
The SPI module, as shown in Figure 10–1, consists of:

- Four I/O pins:
  - SPISIMO (SPI slave in, master out)
  - SPISOMI (SPI slave out, master in)
  - SPICLK (SPI clock)
  - 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 10–1. SPI Module Block Diagram



† The diagram is shown in slave mode.

‡ The SPISTE pin is shown 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.

10.2 SPI Control Registers

Nine registers inside the SPI module (listed in Table 10–1) 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 10–1. 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

10.3 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.

10.3.1 Introduction to Operation

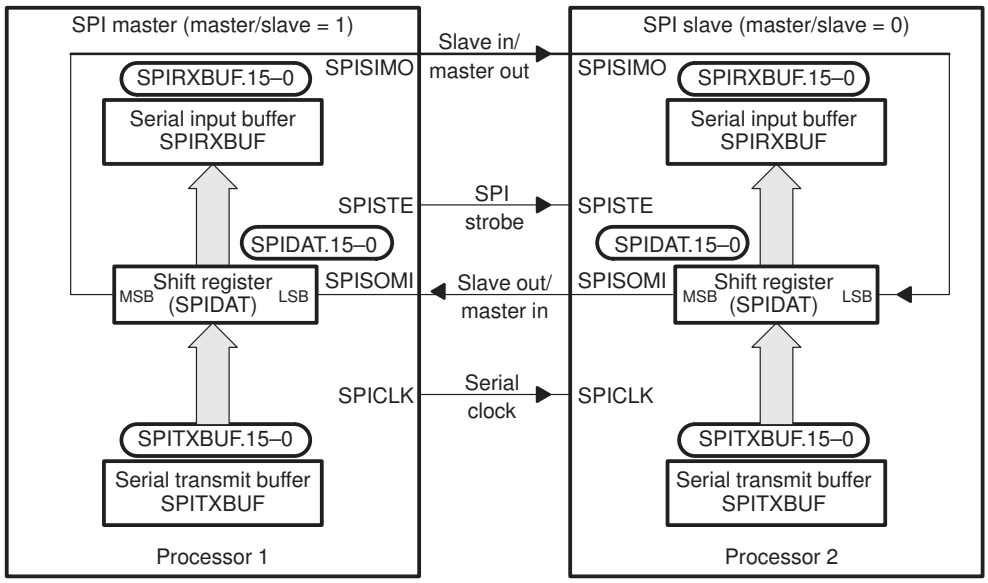
Figure 10–2 shows typical connections of the SPI for communications between two controllers: a master and a slave.

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 10.3.2, *SPI Module Slave and Master Operation Modes*, on page 10-7). 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 10–2. SPI Master/Slave Connection



10.3.2 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 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 SPISTE pin operates as the slave select pin. An active-low signal on the 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.

10.4 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)

10.4.1 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

10.4.2 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.

10.4.3 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

10.4.4 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.

10.4.5 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

10.4.6 Data Format

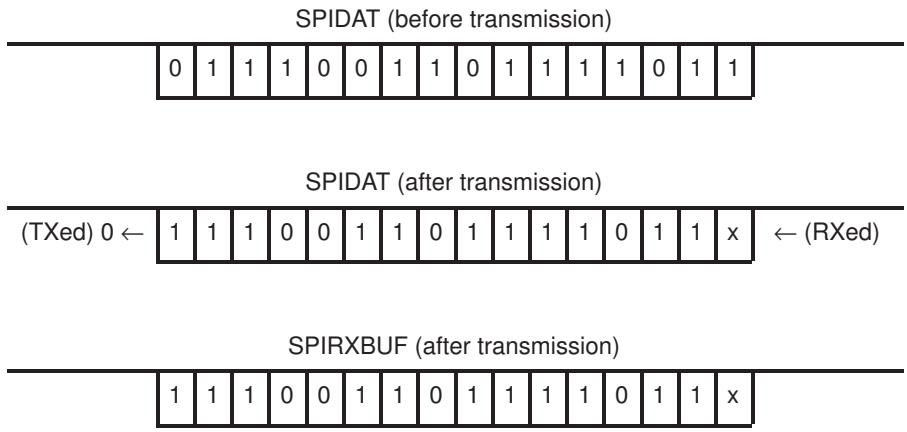
Four bits (SPICCR.3–0) specify the number of bits (1 to 16) in the data character. This information directs the state control logic to count the number of bits received or transmitted to determine when a complete character has been processed. The following statements apply to characters with fewer than 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 10–1).

Example 10–1. 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



Note: x = 1 if SPISOMI data is high; x = 0 if SPISOMI data is low; master mode is assumed.

10.4.7 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 10–1 shows how to determine the SPI baud rates.

Equation 10–1. 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 10–2 shows how to determine the maximum baud rate at which a ’C24x can communicate. Assume that CLKOUT = 20 MHz.

Example 10–2. Maximum Baud-Rate Calculation

$$\text{SPI Baud Rate} = \frac{\text{CLKOUT}}{(\text{SPIBRR} + 1)}$$

$$= \frac{20 \times 10^6}{(3 + 1)}$$

$$= 5 \times 10^6 \text{ bps}$$

The maximum master baud rate would be 5.0 Mbps.

10.4.8 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.

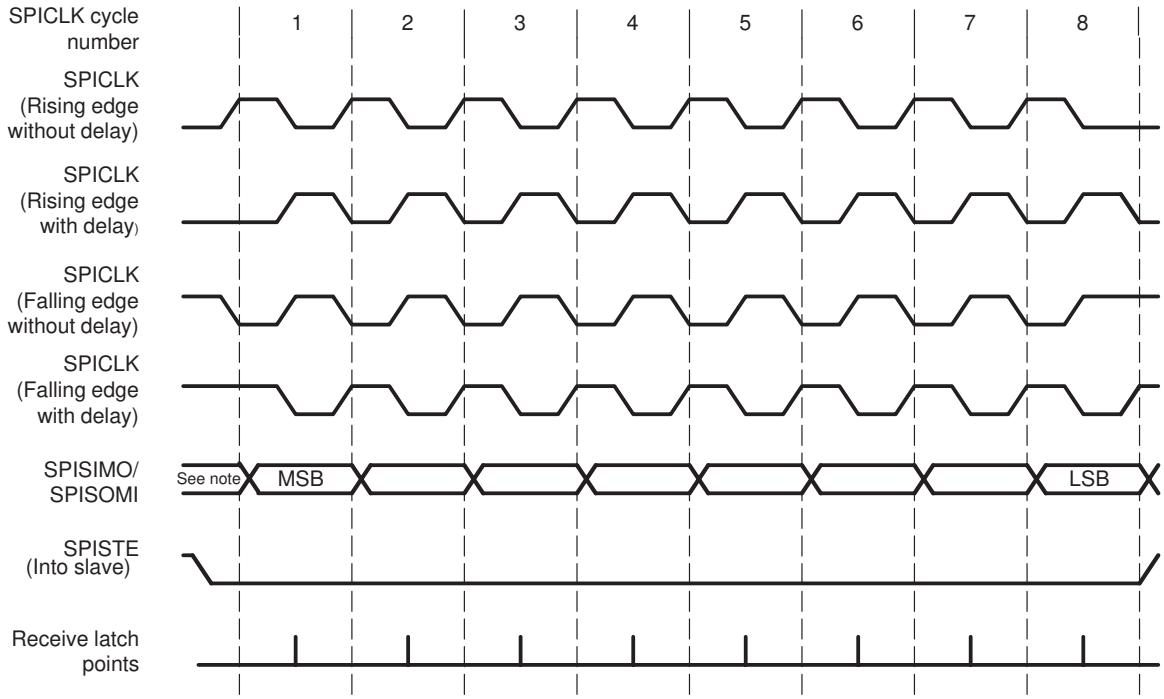
- ❑ Rising Edge With Delay. The SPI transmits data one half-cycle ahead of the rising edge of the SPICLK signal and receives data on the rising edge of the SPICLK signal.

The selection procedure for the SPI clocking scheme is shown in Table 10–2. Examples of these four clocking schemes relative to transmitted and received data are shown in Figure 10–3.

Table 10–2. 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

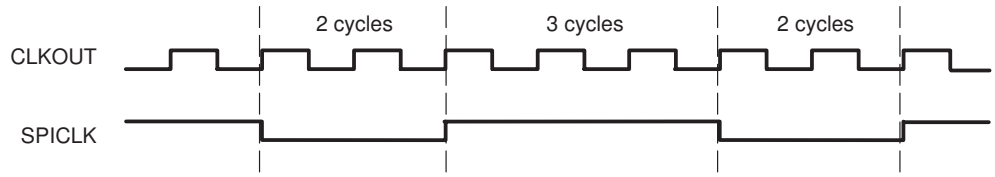
Figure 10–3. SPICLK Signal Options



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 10–4.

Figure 10–4. SPI: SPICLK-CLKOUT Characteristic when (BRR + 1) is Odd, BRR > 3, and CLOCK POLARITY = 1



10.4.9 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 output control register B [OCRB]).

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.

10.4.10 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.

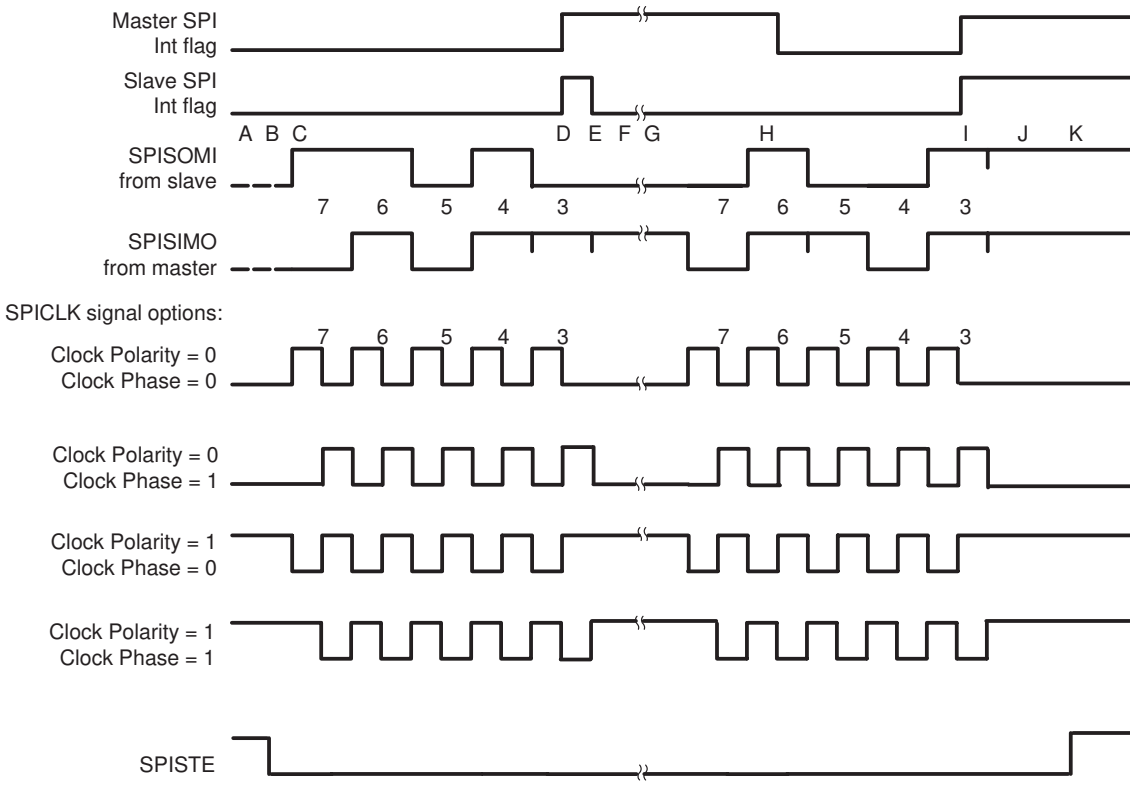
10.4.11 Data Transfer Example

The timing diagram, shown in Figure 10–5, 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 asymmetrical (Figure 10–4, *SPI: SPICLK-CLKOUT Characteristic when (BRR + 1) is Odd, BRR > 3, and CLOCK POLARITY = 1*) shares similar characterizations with Figure 10–5 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 10–5, *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 10–5. 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).

10.5 SPI Control Registers

The SPI is controlled and accessed through registers in the control register file. Figure 10–6 lists the SPI control registers and bit numbers.

Figure 10–6. SPI Control Registers

Addr. Offset	Register Name	Bit number								
		15–8	7	6	5	4	3	2	1	0
7040h	SPICCR	Reserved	SPI SW Reset	Clock Polarity	Reserved		SPI CHAR3	SPI CHAR2	SPI CHAR1	SPI CHAR0
7041h	SPICTL	Reserved	Reserved			Overrun INT ENA	Clock Phase	Master/ Slave	TALK	SPI INT ENA
7042h	SPISTS	Reserved	Receiver Overrun	SPI INT Flag	TX BUF Full	Reserved				
7043h	—	Reserved	Reserved							
7044h	SPIBRR	Reserved	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
7045h	—	Reserved	Reserved							
7046h	SPIRXEMU	ERXB15–8	ERXB7	ERXB6	ERXB5	ERXB4	ERXB3	ERXB2	ERXB1	ERXB0
7047h	SPIRXBUF	RXB15–8	RXB7	RXB6	RXB5	RXB4	RXB3	RXB2	RXB1	RXB0
7048h	SPITXBUF	TXB15–8	TXB7	TXB6	TXB5	TXB4	TXB3	TXB2	TXB1	TXB0
7049h	SPIDAT	SDAT15–8	SDAT7	SDAT6	SDAT5	SDAT4	SDAT3	SDAT2	SDAT1	SDAT0
704Ah	—	Reserved	Reserved							
704Bh	—	Reserved	Reserved							
704Ch	—	Reserved	Reserved							
704Dh	—	Reserved	Reserved							
704Eh	—	Reserved	Reserved							
704Fh	SPIPRI	Reserved	Reserved	SPI Priority	SPI SUSP Soft	SPI SUSP Free	Reserved			

10.5.1 SPI Configuration Control Register (SPICCR)

The SPI Configuration Control Register (SPICCR) controls the setup of the SPI for operation.

Figure 10–7. 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; value following dash (–) = 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 10.4.10 on page 10-15.)

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 10.4.8, *SPI Clocking Schemes*, on page 10-12.

0 Data is output on the rising edge and input on the 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 the falling edge and input on the rising edge. When no SPI data is sent, 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 10–3 lists the character length selected by the bit values.

Table 10–3. 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

10.5.2 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 10–8. 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; value following dash (–) = 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 10–3 on page 10-13). 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

10.5.3 SPI Status Register (SPISTS)

The SPISTS register contains the receive buffer status bits.

Figure 10–9. 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, value following dash (–) = value after reset

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.

† The RECEIVER_OVERRUN_FLAG bit and SPI_INT_FLAG bit share the same interrupt vector.
‡ Writing a 0 to bits 5, 6, and 7 has no effect.

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.

10.5.4 SPI Baud Rate Register (SPIBRR)

The SPIBRR contains the bits used for baud-rate selection.

Figure 10–10. 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; value following dash (–) = 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 10–2.

Equation 10–2. 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

10.5.5 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 10–11. 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; value following dash (–) = value after reset (x = indeterminate)

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.

10.5.6 SPI Serial Receive Buffer Register (SPIRXBUF)

The SPIRXBUF contains the received data. Reading the SPIRXBUF clears the SPI INT FLAG bit (SPISTS.6).

Figure 10–12. 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; value following dash (–) = value after reset (x = indeterminate)

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.

10.5.7 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 to 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 10–13. 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

Notes: 1) R = Read access; value following dash (–) = value after reset (x = indeterminate)
2) Writes to SPITXBUF must be left-justified

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.

10.5.8 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 10–14. 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; value following dash (–) = value after reset (x = indeterminate)

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.

10.5.9 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 10–15. 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; value following dash (–) = 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 Soft	Bit 4 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.

10.6 SPI Example Waveforms

Figure 10–16. CLOCK_POLARITY = 0, CLOCK_PHASE = 0 (All data transitions are during the rising edge. Inactive level is low.)

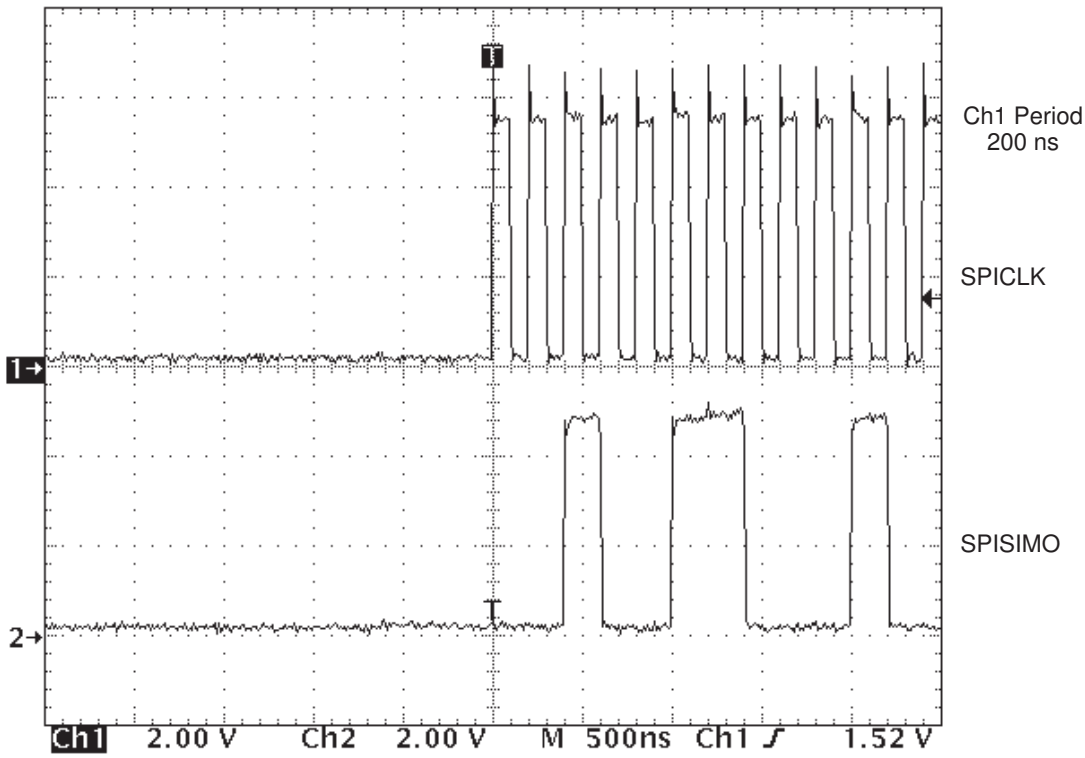


Figure 10–17. CLOCK_POLARITY = 0, CLOCK_PHASE = 1 (Add data transitions are during the rising edge, but delayed by half clock cycle. Inactive level is low.)

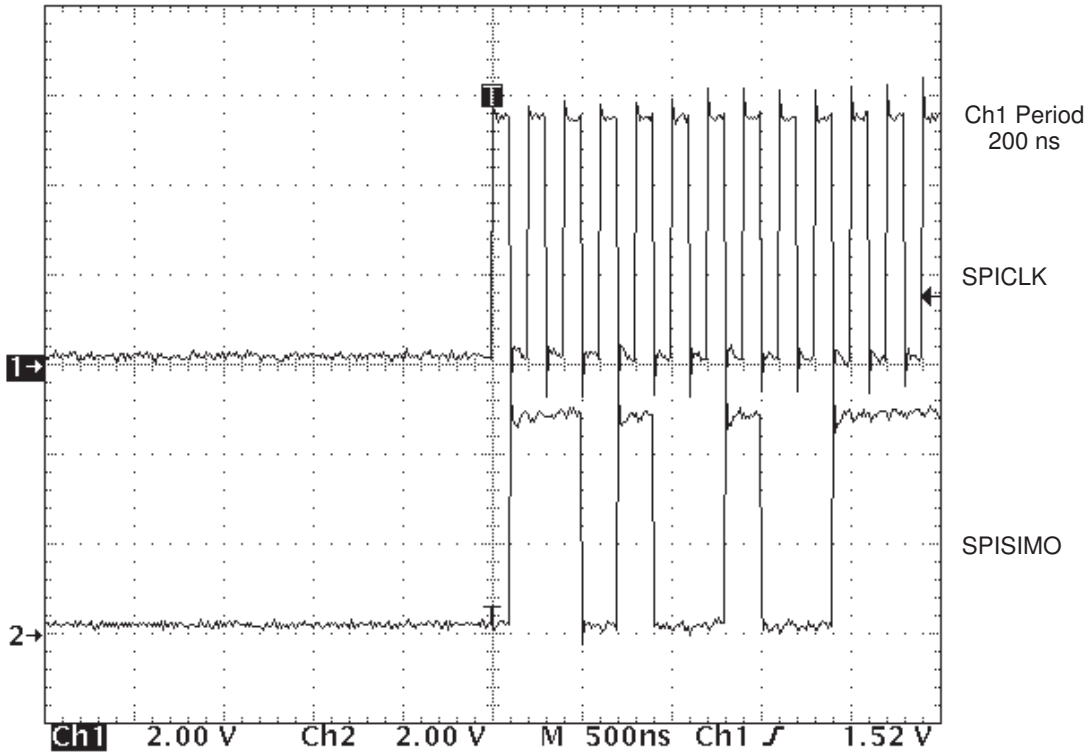


Figure 10–18. CLOCK_POLARITY = 1, CLOCK_PHASE = 0 (All data transitions are during the falling edge. Inactive level is high.)

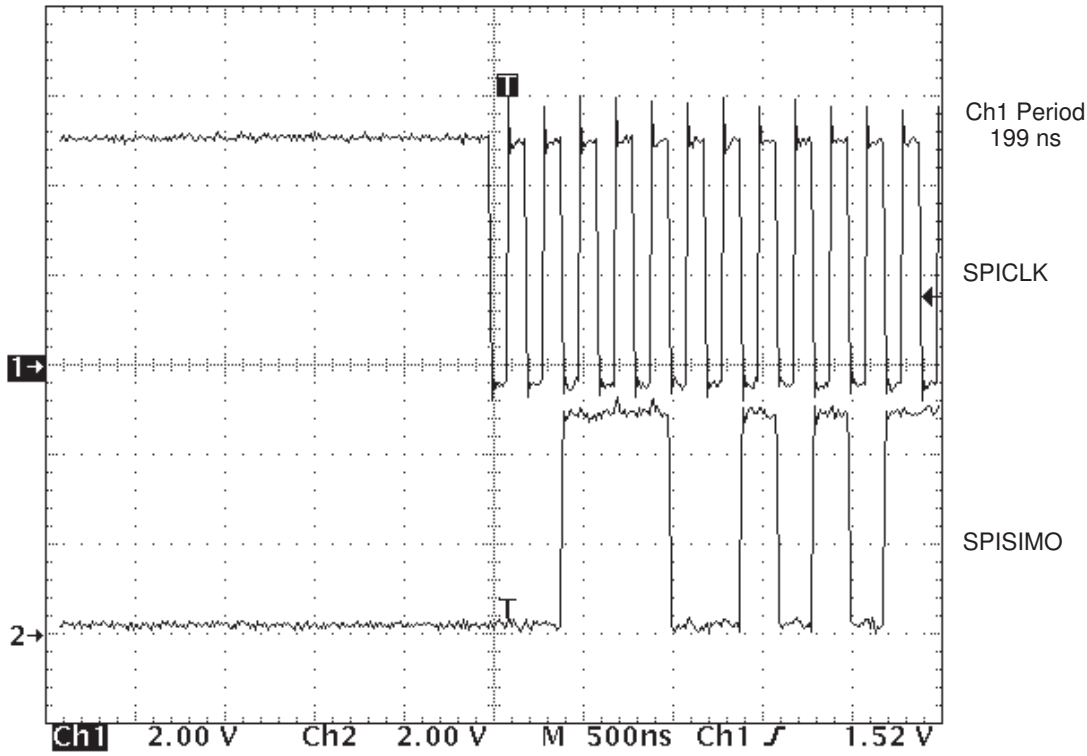


Figure 10–19. CLOCK_POLARITY = 1, CLOCK_PHASE = 1 (Add data transitions are during the falling edge, but delayed by half clock cycle. Inactive level is high.)

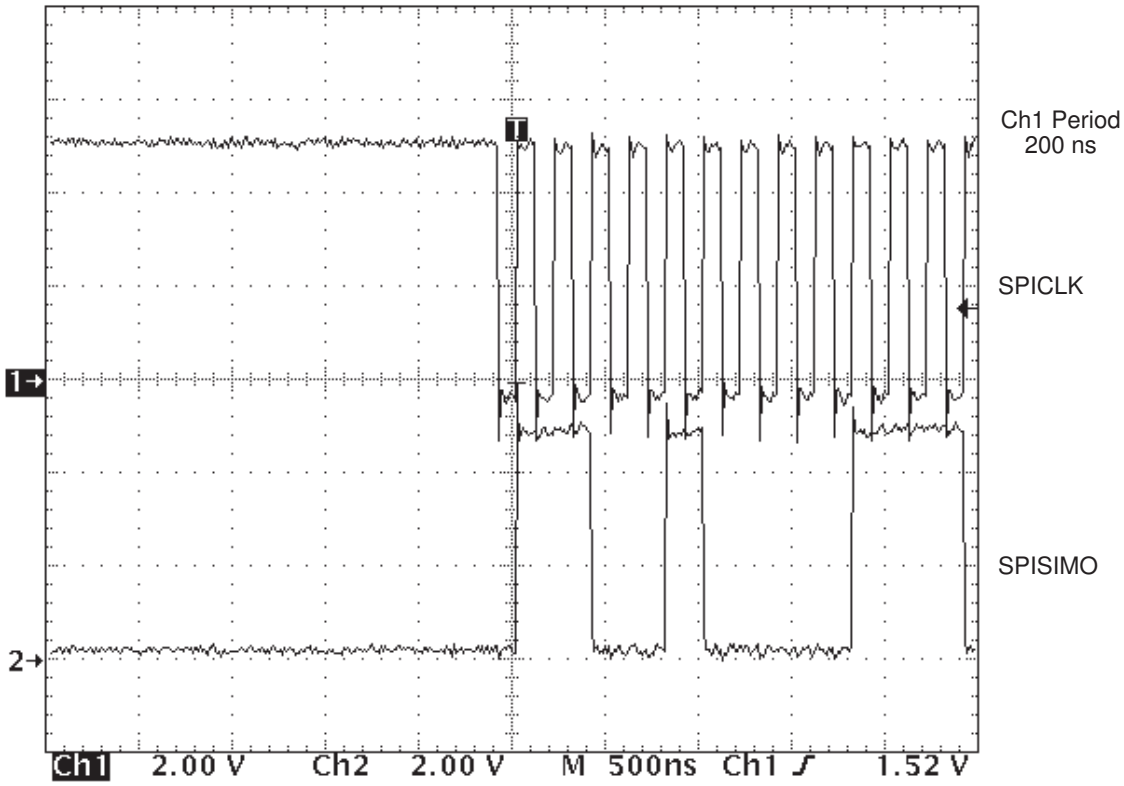


Figure 10–20. SPISTE Behavior in Master Mode (Master lowers SPISTE during the entire 16 bits of transmission.)

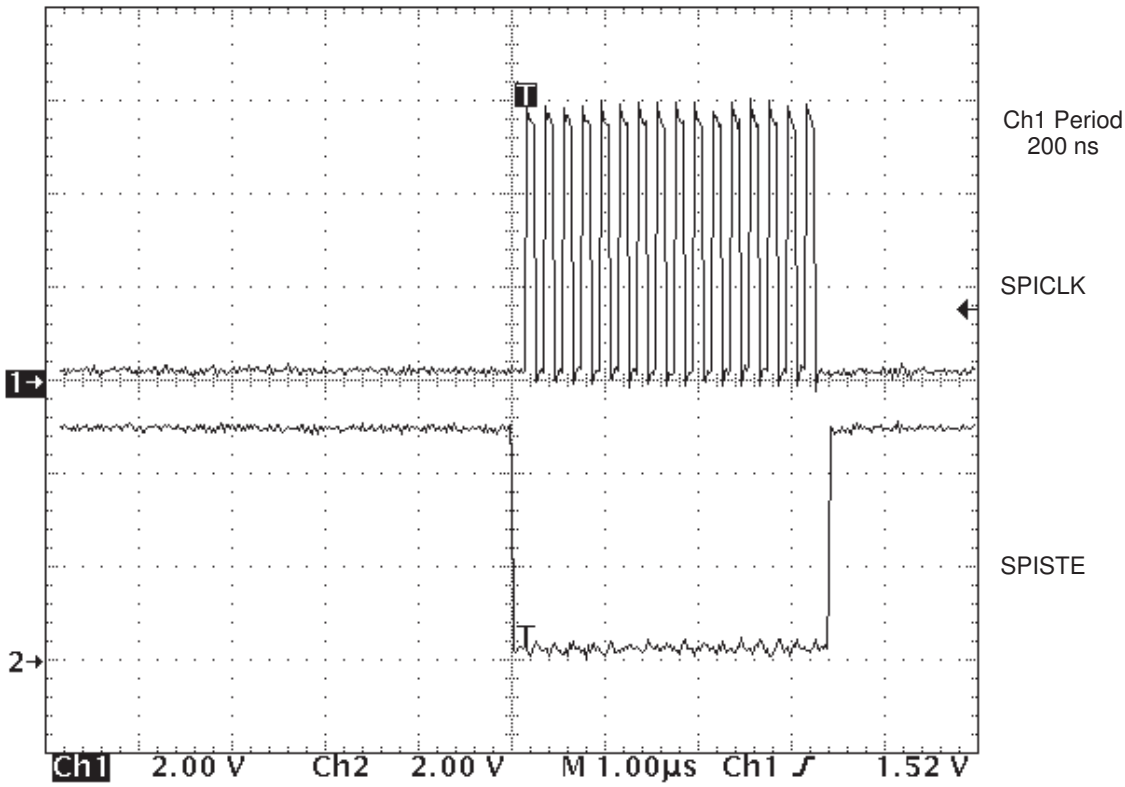
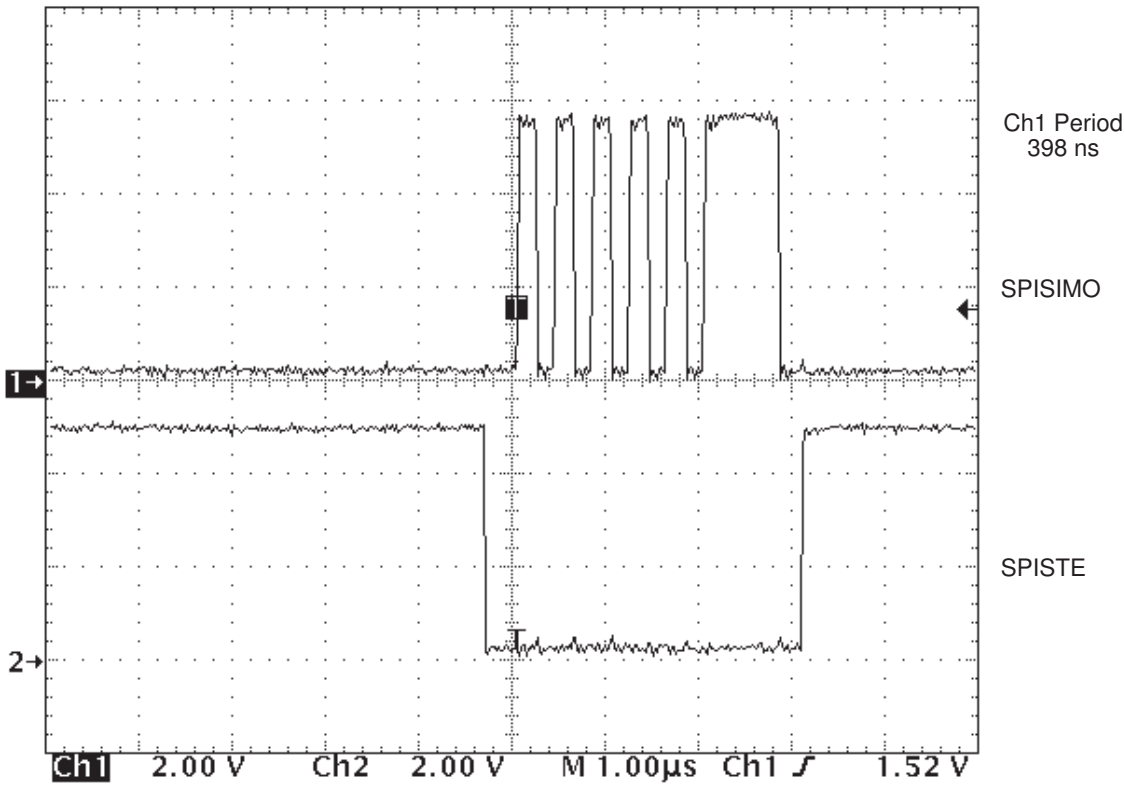


Figure 10–21. SPISTE Behavior in Slave Mode (Slave's SPISTE is lowered during the entire 16 bits of transmission.)



Chapter 11

CAN Controller Module

This chapter describes the controller area network (CAN) module available on the ’F241 and ’F243 devices. 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**

11.1 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

11.2 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.

11.2.1 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

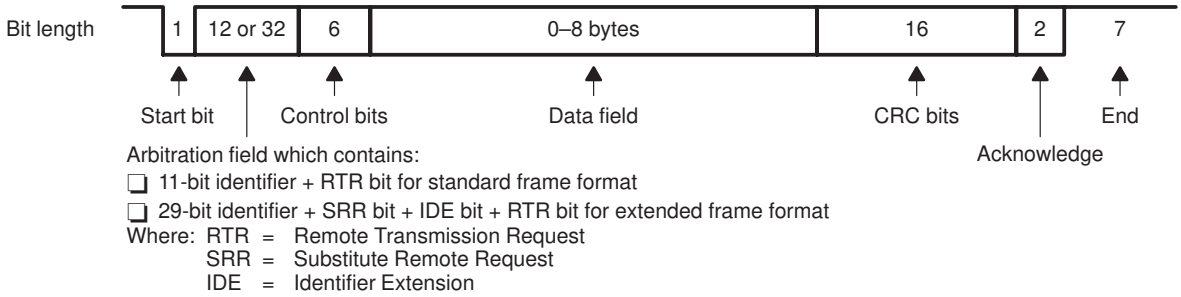
In addition, CAN specification version 2.0B defines two different formats that differ in the length of the identifier field: standard frames with an 11-bit identifier and extended frames with a 29-bit identifier.

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 11–1, 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 11–1. CAN Data Frame

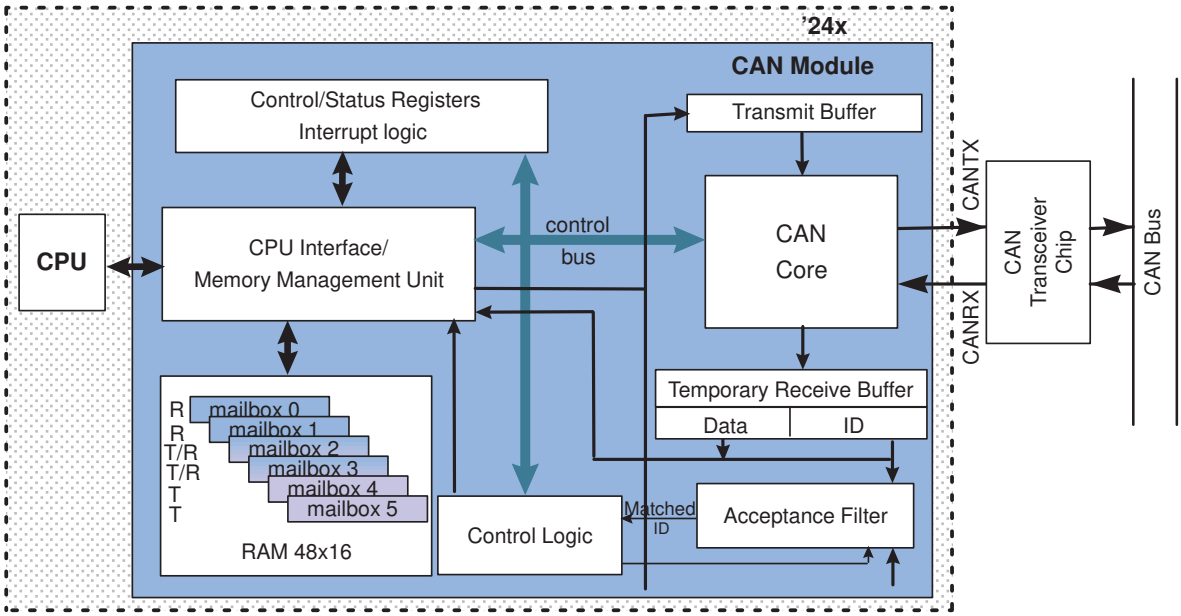


Note: Unless otherwise noted, numbers are amount of bits in field.

11.2.2 CAN Controller Architecture

Figure 11–2 shows the basic architecture of the CAN controller.

Figure 11–2. TMS320F243/F241 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 11–1 shows the configuration details of the mailboxes.

Table 11–1. 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	—

11.2.3 Memory Map

Figure 11–3 shows memory space, and Table 11–2 and Table 11–3 give the register and mailbox locations in the CAN module.

Figure 11–3. TMS320F243/F241 CAN Module Memory Space

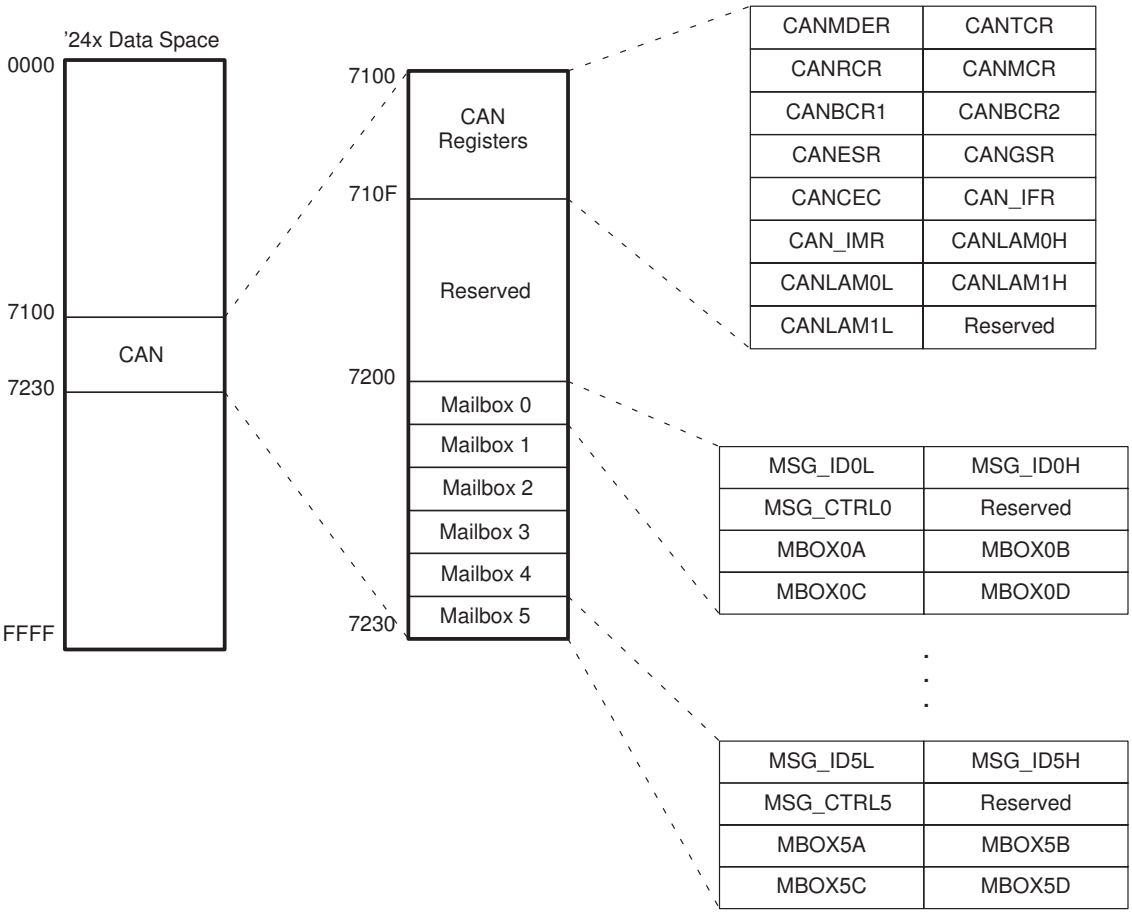


Table 11–2. Register Addresses

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 11–3 shows the mailbox locations in the RAM.

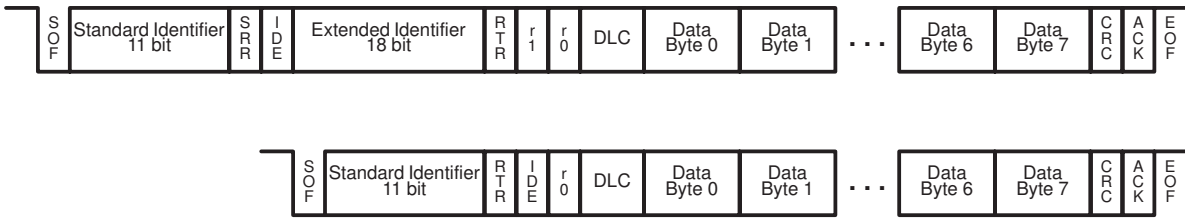
Table 11–3. Mailbox Addresses

	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

11.3 Message Objects

CAN allows messages to be sent, received, and stored by using data frames. Figure 11–4 illustrates the structure of the data frames with extended and standard identifiers.

Figure 11–4. 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

11.3.1 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.

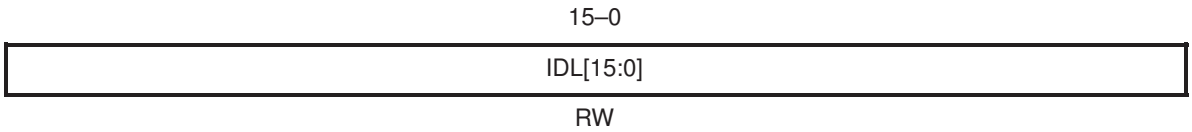
2) Message Identifiers:

Each one of the six mailboxes has its own message identifier stored in two 16-bit registers. Figure 11–5 shows the message identifier high word and Figure 11–6 shows the message identifier low word.

Figure 11–5. Message Identifier for High Word Mailboxes 0–5 (MSGIDnH)

Note: R = Read access; W = Write access

Figure 11–6. Message Identifier for Low Word Mailboxes 0–5 (MSGIDnL)



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.

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 11–7 illustrates the layout and default mode of the message control field.

Figure 11–7. Message Control Field 0–5 (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

11.3.2 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.

11.3.3 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.

11.3.4 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.

11.3.5 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

11.3.6 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 11–8 (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 11–8 (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 11–8 (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 11–8 (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.

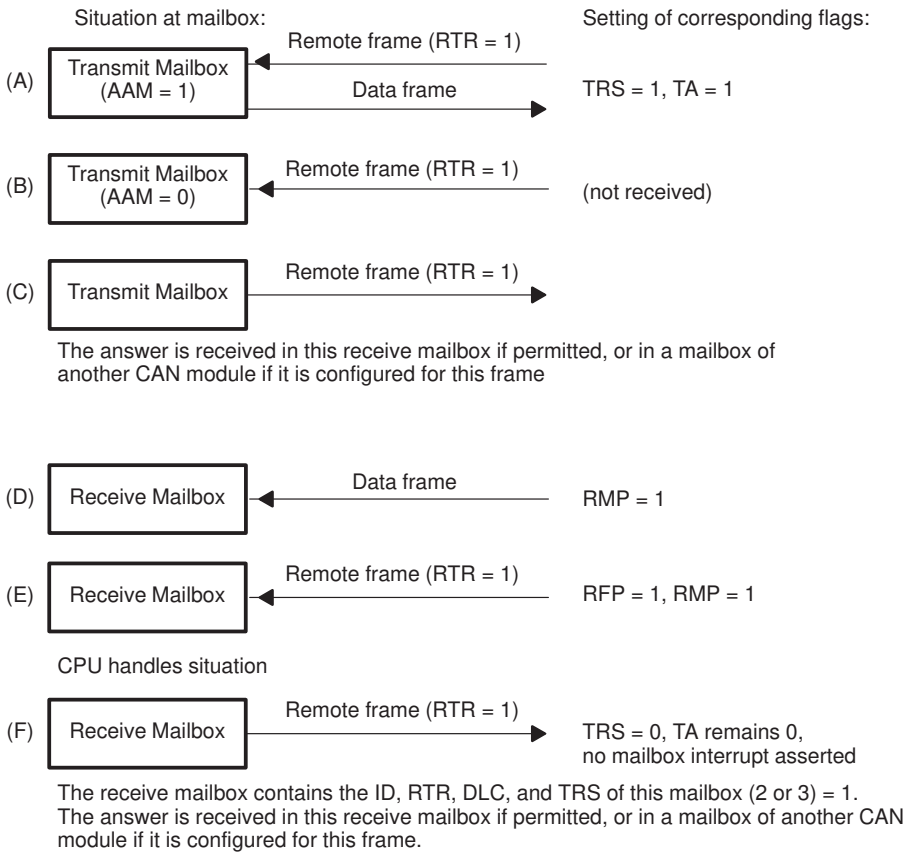
To summarize: A mailbox can be configured in four different ways:

- ❑ 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 11–8. Remote Frame Requests



11.3.7 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 11–9 illustrates the LAMn_H high word and Figure 11–10 illustrates the LAM_H low word.

Figure 11–9. Local Acceptance Mask Register n (0, 1) High Word (LAMn_H)

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.

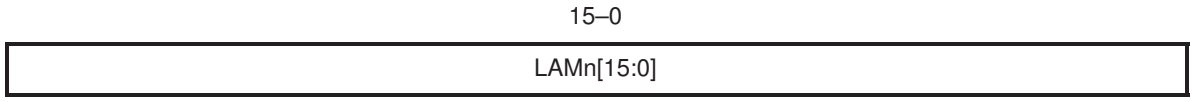
Bits 14–13 **Reserved.**

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 11–10. Local Acceptance Mask Register n (0, 1) Low Word (LAMn_L)



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.

11.4 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.

11.4.1 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 11–11 illustrates this register.

Figure 11–11.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

11.4.2 Transmit Control Register (TCR)

The transmit control register (TCR) contains bits that control the transmission of messages (see Figure 11–12).

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 11–12. 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.

11.4.3 Receive Control Register (RCR)

The receive control register (RCR) contains the bits which control the reception of messages and remote frame handling.

Figure 11–13. 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.

11.4.4 Master Control Register (MCR)

The Master Control Register is used to control the behavior of the CAN core module.

Figure 11–14. 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 11–4).

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.

Bit 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.

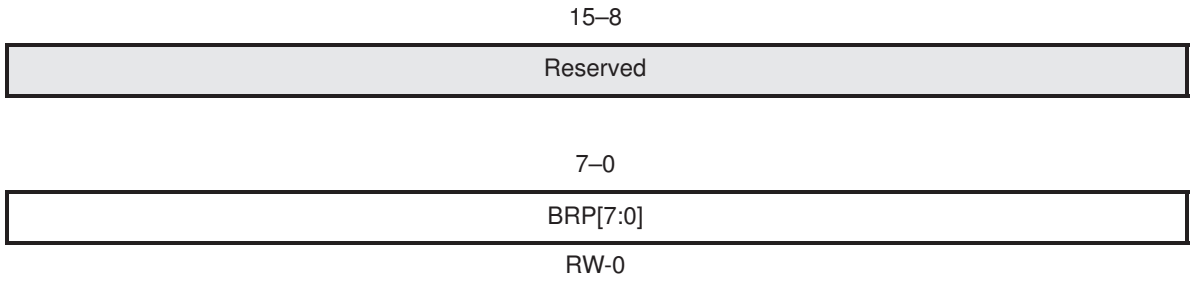
11.4.5 Bit Configuration Registers (BCR*n*)

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 11–15. Bit Configuration Register 2 (BCR2) — Address 7104h



Note: R = Read access; W = Write access; value following dash (–) = value after reset

Bits 15–8 Reserved.

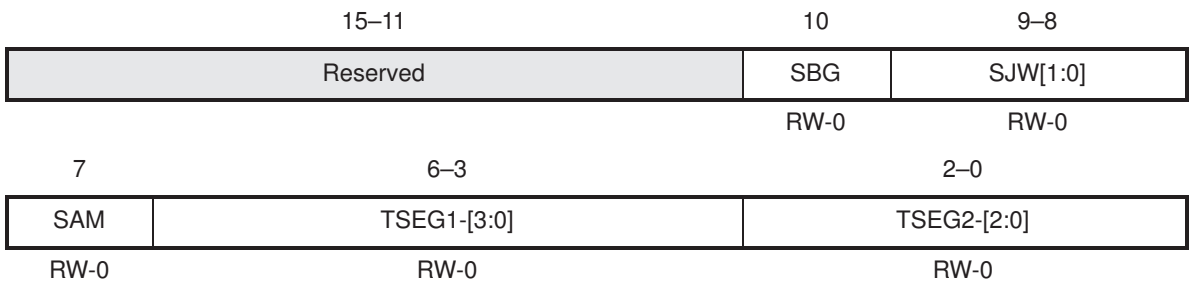
Bits 7–0 BRP. Baud Rate Prescaler

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 11–16. Bit Configuration Register 1 (BCR1) — Address 7105h



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).

$$TSEG = PROP\_SEG + PHASE\_SEG1.$$

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:

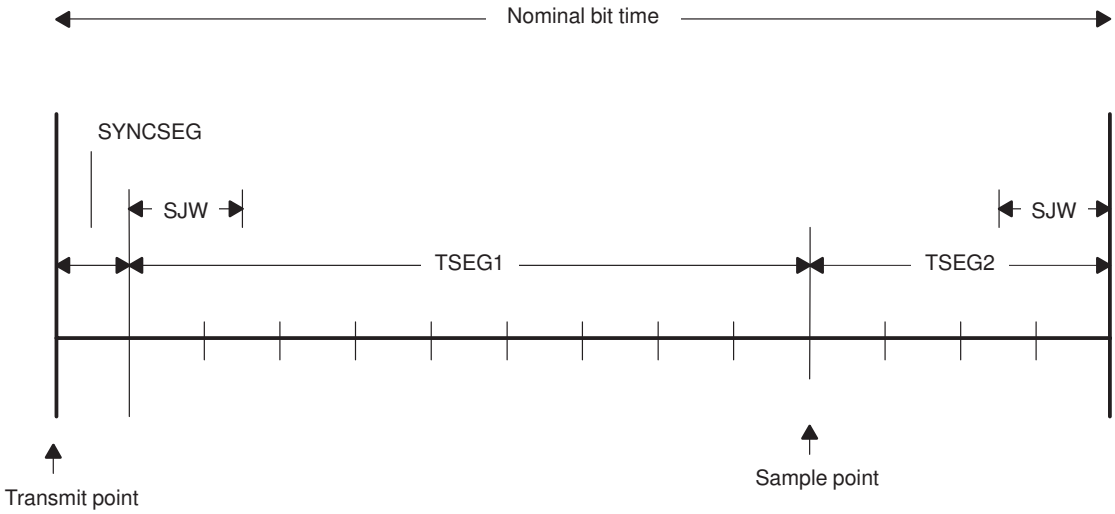
$$TSEG2_{min} = 1 + SJW.$$

The value of TSEG2 is programmable from 2 to 8 TQ in compliance with the formula:

$$(SJW + SBG + 1) \leq TSEG2 \leq 8.$$

CAN Bit Timing

Figure 11–17. 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 11–4. 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

11.5 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.

11.5.1 Global Status Register (GSR)

Figure 11–18. Global Status Register (GSR) — Address 7107h

15–8
Reserved

7–6	5	4	3	2	1	0
Reserved	SMA	CCE	PDA	Reserved	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.

11.5.2 Error Status Register (ESR)

The error status register (see Figure 11–19) 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 11–19. 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.

11.5.3 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 11–20. 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 $\times$ 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.

11.6 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.

11.6.1 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 11–21. 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
Reserved	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 6 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.

11.6.2 CAN Interrupt Mask Register (CAN_IMR)

The setup for the interrupt mask register (see Figure 11–22) 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 11–22. 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.

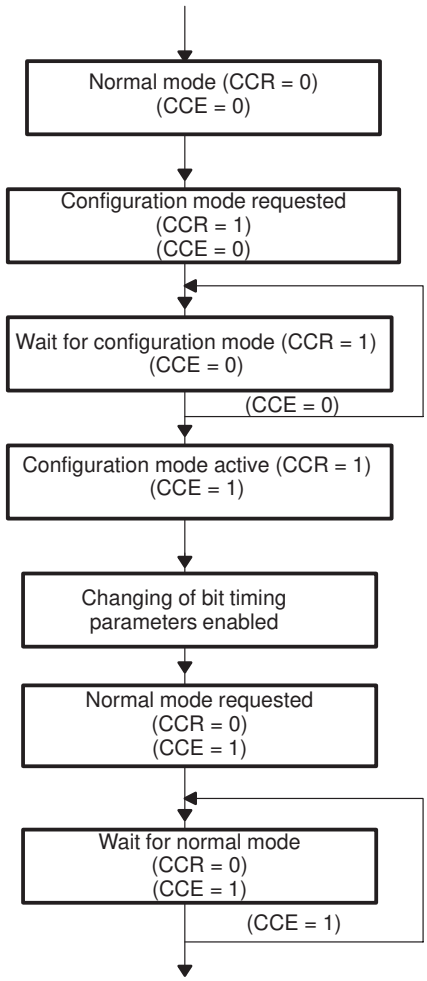
0 The named interrupts generate high priority requests; that is, on line CAERRIRQn with CAERRPRI set to 1.

1 The named interrupts generate low priority requests; that is, on line CAERRIRQn with CAERRPRI set to 0.

11.7 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 11–23. CAN Initialization



11.8 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.

11.9 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 11–5 provides a listing of the notation, definition, and register and bit number.

Table 11–5. 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 11–5. 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 12

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 ’X243/241/242 WD timer and that on the ’X240 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**

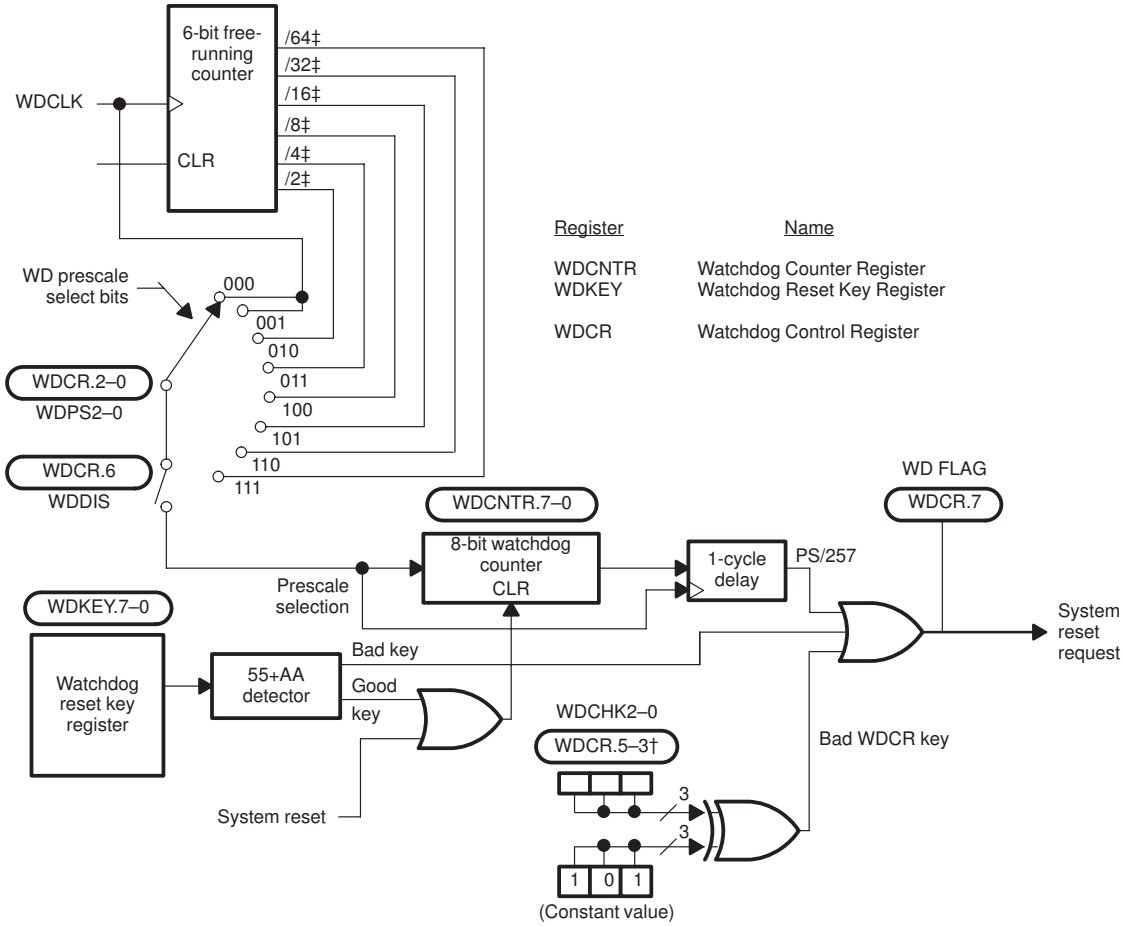
12.1 Watchdog Timer Features

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
- A WD flag (WD FLAG) bit that indicates if the WD timer initiated a system reset
- 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 12–1 shows a block diagram of the WD module

Figure 12–1. Block Diagram of the WD Module



† 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.

12.2 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)

12.3 Watchdog Timer Clock

A low-frequency clock (WDCLK) is used to clock the watchdog timer. WDCLK has a nominal frequency of 39062.5 Hz when CPUCLK = 20 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 section 4.4, *Low Power Modes*, on page 4-3). 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 WDCNTR should be reset before it overflows to prevent a device reset.

12.3.1 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 CLKOUT.

Note that the watchdog timer clock does not run when the real-time monitor is running. This is different from the 'F/C240.

12.3.2 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.

12.3.3 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 6.55 ms to 419.43 ms for a WDCLK rate of 39062.5 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.

12.3.4 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.

The following shows a typical sequence written to WDKEY after power-up (Table 12–1):

Table 12–1. 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 re-enables 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.

12.3.4.1 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 power-down mode when WDCLK is not active. For software development or flash programming purposes, the WD timer can be disabled by applying 5V to the V_{CCP} pin (on Flash devices) or the WDDIS pin (on ROM devices) and setting the WDDIS bit in the WD control register (WDCR.6). However, if the hardware and software conditions are not met, the WD timer will not be disabled.

12.3.4.2 WD Check Bit Logic

The WD check bits (WDCR.5–3, described in detail in section 12.4.3 on page 12-10) 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.

12.3.4.3 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 (6.55 ms for a 39062.5 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.

12.4 Watchdog Control Registers

The WD module control registers are shown in Table 12–2 and discussed in detail in the subsections that follow the table.

Table 12–2. 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

12.4.1 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 12–2. 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; value following dash (–) = 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.

12.4.2 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 12–3. 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; value following dash (–) = 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.

12.4.3 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 12–4. 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 only; W = Write access; Wc = Write access conditional on VCCP or WDDIS Pins being high; value following dash (–) = value after reset (-x = depends on cause of reset)

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 12–3 show the overflow times for each prescaler setting when the WDCLK is running at 39062.5 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 12–3 because of the added uncertainty resulting from not clearing the prescaler.

Table 12–3. WD Overflow (Timeout) Selections

WD Prescale Select Bits				39.0625 kHz WDCLK†	
WDPS2	WDPS1	WDPS0	WDCLK Divider	Overflow Frequency (Hz)	Minimum Overflow (ms)
0	0	X	1	152.59	6.55
0	1	0	2	76.29	13.11
0	1	1	4	38.15	26.21
1	0	0	8	19.07	52.43
1	0	1	16	9.54	104.86
1	1	0	32	4.77	209.72
1	1	1	64	2.38	419.43

X = Don't care
† Generated by a 5-MHz input clock

Note: **WDDIS pin**

0	WD always active
1 (5V)	Provides software disable feature using WDDIS bit in WDCR

Chapter 13

’240–’24x Compatibility

This section describes how to use an ’F/C240 device to develop ’F/C24x software.

The software changes required between ’X240 code and ’C24x code have been kept to an absolute minimum. All register addresses, and almost all bit positions and functions, are identical between the ’X240 and ’C24x devices.

Topic **Page**

13.1 General

If porting code from an ’X240 to an ’F243/F241 or ’C242 device, the total code size must be less than 8k or 4k words respectively.

Low power mode 2 (Halt) is the lowest power mode on the ’C24x. It is similar to the LPM3 (oscillator power down) on the ’X240. There is no equivalent to LPM2 (PLL power down) on the ’X240. The low power mode bits are in a different register (SCSR) and in different bit positions on the ’C24x.

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 ’X240; however, it does on ’C24x devices. Furthermore, an illegal address generates a reset on the ’X240, and an NMI on the ’C24x. The NMI service routine must poll the ILLADDR bit in SCSR to determine whether the NMI was caused by an illegal address or the NMI pin.

External interrupts XINT2 and XINT3 on the ’X240 are similar to external interrupts XINT1 and XINT2 on the ’C24x. 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 clock out control bits are in a different register (SCSR) and bit position.

13.2 Event Manager

In order to port code from ’X240 to ’C24x:

- 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 ’X240 to the ’C24x.
- The 32-bit timer mode cannot be used.
- Capture 3 on the ’X240 cannot be used, when porting code from the ’X240 to the ’C24x. Capture 4 should be changed into capture 3.
- The capture units can use either GP Timer 1 or 2 as a time base.
- 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 ’X240.
- 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 ’X240 to ’C24x 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.

13.3 Analog-to-Digital Converter

The ADC clock prescale values will have to be changed when porting code from a ’X240 to a ’C24x device. Only ADC inputs 0–3 and 8–11 can be used on the ’X240 (these are equivalent to inputs 0–7 on ’C24x devices). Otherwise, further software changes will be required.

13.4 Serial Communication 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 ’X240 to a ’C24x 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.

13.5 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 ’X240 to ’C24x 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 an ’X240 to a ’C24x 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.

13.6 Watchdog Timer

Because ’C24x devices do not have an RTI, all code that uses the RTI peripheral (if any) must be removed when porting code from ’X240 to ’C24x devices.

Appendix A

Summary of Programmable Registers on the TMS320F243/F241/C242

Table A–1. Programmable Registers on the TMS320F243/F241/C242

			Shown in	
Address	**Register**	**Name**	**Figure**	**Page**
Internal	ST0	CPU Status Register 0	SPRU160	Chapter 4
Internal	ST1	CPU Status Register 1	SPRU160	Chapter 4
0004h	IMR	CPU Interrupt Mask Register	Figure 2–6	2-19
0005h	GREG	Global Data Memory Configuration Register	Figure 3–7	3-12
0006h	IFR	CPU Interrupt Flag Register	Figure 2–5	2-18
7010h	PIRQR0	Peripheral Interrupt Request Register 0	Figure 2–8	2-22
7011h	PIRQR1	Peripheral Interrupt Request Register 1	Figure 2–9	2-23
7014h	PIACKR0	Peripheral Interrupt Acknowledge Register 0	Figure 2–10	2-24
7015h	PIACKR1	Peripheral Interrupt Acknowledge Register 1	Figure 2–11	2-25
7018h	SCSR	System Control and Status Register	Figure 5–1	5-3
701Ch	DINR	Device Identification Name Register	Figure 5–2	5-4
701Eh	PIVR	Peripheral Interrupt Vector Register	Figure 2–7	2-21
7023h	WDCNTR	Watchdog Counter Register	Figure 12–2	12-9
7025h	WDKEY	Watchdog Reset Key Register	Figure 12–3	12-9
7029h	WDCR	Watchdog Timer Control Register	Figure 12–4	12-10
7032h	ADCTRL1	ADC Control Register 1	Figure 8–3	8-10
7034h	ADCTRL2	ADC Control Register 2	Figure 8–4	8-13
7036h	ADCFIFO1	ADC Data Register FIFO 1	Figure 8–5	8-15
7038h	ADCFIFO2	ADC Data Register FIFO 2	Figure 8–5	8-15

Table A–1. Programmable Registers on the TMS320F243/F241/C242 (Continued)

			Shown in	
Address	**Register**	**Name**	**Figure**	**Page**
7040h	SPICCR	SPI Configuration Control Register	Figure 10–7	10-18
7041h	SPICTL	SPI Operation Control Register	Figure 10–8	10-20
7042h	SPISTS	SPI Status Register	Figure 10–9	10-22
7044h	SPIBRR	SPI Baud Rate Control Register	Figure 10–10	10-23
7046h	SPIRXEMU	SPI Emulation Buffer Register	Figure 10–11	10-24
7047h	SPIRXBUF	SPI Serial Receive Buffer Register	Figure 10–12	10-25
7048h	SPITXBUF	SPI Serial Transmit Buffer Register	Figure 10–13	10-26
7049h	SPIDAT	SPI Serial Data Register	Figure 10–14	10-27
704Fh	SPIPRI	SPI Priority Control Register	Figure 10–15	10-28
7050h	SCICCR	SCI Communication Control Register	Figure 9–10	9-21
7051h	SCICTL1	SCI Control Register 1	Figure 9–11	9-23
7052h	SCIHBAUD	SCI Baud-Select Register, High Bits	Figure 9–12	9-26
7053h	SCILBAUD	SCI Baud-Select Register, Low Bits	Figure 9–13	9-26
7054h	SCICTL2	SCI Control Register 2	Figure 9–14	9-27
7055h	SCIRXST	SCI Receiver Status Register	Figure 9–15	9-28
7056h	SCIRXEMU	SCI Emulation Data Buffer Register	Figure 9–17	9-31
7057h	SCIRXBUF	SCI Receiver Data Buffer Register	Figure 9–18	9-31
7059h	SCITXBUF	SCI Transmit Data Buffer Register	Figure 9–19	9-32
705Fh	SCIPRI	SCI Priority Control Register	Figure 9–20	9-32
7070h	XINT1CR	External Interrupt 1 Control Register	Figure 5–3	5-5
7071h	XINT2CR	External Interrupt 2 Control Register	Figure 5–4	5-6
7090h	OCRA	I/O Mux Control Register A	Figure 6–2	6-4
7092h	OCRB	I/O Mux Control Register B	Figure 6–3	6-6
7098h	PADATDIR	I/O Port A Data and Direction Register	Figure 6–5	6-8
709Ah	PBDATDIR	I/O Port B Data and Direction Register	Figure 6–6	6-9

Table A–1. Programmable Registers on the TMS320F243/F241/C242 (Continued)

			Shown in	
Address	**Register**	**Name**	**Figure**	**Page**
709Ch	PCDATDIR	I/O Port C Data and Direction Register	Figure 6–7	6-10
709Eh	PDDATDIR	I/O Port D Data and Direction Register	Figure 6–8	6-11
7100h	MDER	Mailbox Direction/Enable Register	Figure 11–11	11-18
7101h	TCR	Transmission Control Register	Figure 11–12	11-19
7102h	RCR	Receive Control Register	Figure 11–13	11-21
7103h	MCR	Master Control Register	Figure 11–14	11-22
7104h	BCR2	Bit Configuration Register 2	Figure 11–15	11-25
7105h	BCR1	Bit Configuration Register 1	Figure 11–16	11-25
7106h	ESR	Error Status Register	Figure 11–19	11-29
7107h	GSR	Global Status Register	Figure 11–18	11-28
7108h	CEC	CAN Error Counter Registers	Figure 11–20	11-31
7109h	CAN_IFR	Interrupt Flag Register	Figure 11–21	11-33
710Ah	CAN_IMR	Global Interrupt Mask Register	Figure 11–22	11-35
710Bh	LAM0_H	Local Acceptance Mask Mailbox 0 and 1	Figure 11–9	11-17
710Ch	LAM0_L	Local Acceptance Mask Mailbox 0 and 1	Figure 11–10	11-17
710Dh	LAM1_H	Local Acceptance Mask Mailbox 2 and 3	Figure 11–9	11-17
710Eh	LAM1_L	Local Acceptance Mask Mailbox 2 and 3	Figure 11–10	11-17
7200h	MSGID0L	CAN Message ID for Mailbox 0 (lower 16 bits)	Figure 11–6	11-10
7201h	MSGID0H	CAN Message ID for Mailbox 0 (upper 16 bits)	Figure 11–5	11-10
7202h	MSGCTRL0	CAN Message Control Field 0	Figure 11–7	11-11
7204h	MBX0A	CAN 2 of 8 Bytes of Mailbox 0	Not shown	Not shown
7205h	MBX0B	CAN 2 of 8 Bytes of Mailbox 0	Not shown	Not shown
7206h	MBX0C	CAN 2 of 8 Bytes of Mailbox 0	Not shown	Not shown
7207h	MBX0D	CAN 2 of 8 Bytes of Mailbox 0	Not shown	Not shown

Table A–1. Programmable Registers on the TMS320F243/F241/C242 (Continued)

			Shown in	
Address	**Register**	**Name**	**Figure**	**Page**
7208h	MSGID1L	CAN Message ID for Mailbox 1 (lower 16 bits)	Figure 11–6	11-10
7209h	MSGID1H	CAN Message ID for Mailbox 1 (upper 16 bits)	Figure 11–5	11-10
720Ah	MSGCTRL1	CAN Message Control Field 1	Figure 11–7	11-11
720Ch	MBX1A	CAN 2 of 8 Bytes of Mailbox 1	Not shown	Not shown
720Dh	MBX1B	CAN 2 of 8 Bytes of Mailbox 1	Not shown	Not shown
720Eh	MBX1C	CAN 2 of 8 Bytes of Mailbox 1	Not shown	Not shown
720Fh	MBX1D	CAN 2 of 8 Bytes of Mailbox 1	Not shown	Not shown
7210h	MSGID2L	CAN Message ID for Mailbox 2 (lower 16 bits)	Figure 11–6	11-10
7211h	MSGID2H	CAN Message ID for Mailbox 2 (upper 16 bits)	Figure 11–5	11-10
7212h	MSGCTRL2	CAN Message Control Field 2	Figure 11–7	11-11
7214h	MBX2A	CAN 2 of 8 Bytes of Mailbox 2	Not shown	Not shown
7215h	MBX2B	CAN 2 of 8 Bytes of Mailbox 2	Not shown	Not shown
7216h	MBX2C	CAN 2 of 8 Bytes of Mailbox 2	Not shown	Not shown
7217h	MBX2D	CAN 2 of 8 Bytes of Mailbox 2	Not shown	Not shown
7218h	MSGID3L	CAN Message ID for Mailbox 3 (lower 16 bits)	Figure 11–6	11-10
7219h	MSGID3H	CAN Message ID for Mailbox 3 (upper 16 bits)	Figure 11–5	11-10
721Ah	MSGCTRL3	CAN Message Control Field 3	Figure 11–7	11-11
721Ch	MBX3A	CAN 2 of 8 Bytes of Mailbox 3	Not shown	Not shown
721Dh	MBX3B	CAN 2 of 8 Bytes of Mailbox 3	Not shown	Not shown
721Eh	MBX3C	CAN 2 of 8 Bytes of Mailbox 3	Not shown	Not shown
721Fh	MBX3D	CAN 2 of 8 Bytes of Mailbox 3	Not shown	Not shown
7220h	MSGID4L	CAN Message ID for Mailbox 4 (lower 16 bits)	Figure 11–6	11-10

Table A–1. Programmable Registers on the TMS320F243/F241/C242 (Continued)

			Shown in	
Address	**Register**	**Name**	**Figure**	**Page**
7221h	MSGID4H	CAN Message ID for Mailbox 4 (upper 16 bits)	Figure 11–5	11-10
7222h	MSGCTRL4	CAN Message Control Field 4	Figure 11–7	11-11
7224h	MBX4A	CAN 2 of 8 Bytes of Mailbox 4	Not shown	Not shown
7225h	MBX4B	CAN 2 of 8 Bytes of Mailbox 4	Not shown	Not shown
7226h	MBX4C	CAN 2 of 8 Bytes of Mailbox 4	Not shown	Not shown
7227h	MBX4D	CAN 2 of 8 Bytes of Mailbox 4	Not shown	Not shown
7228h	MSGID5L	CAN Message ID for Mailbox 5 (lower 16 bits)	Figure 11–6	11-10
7229h	MSGID5H	CAN Message ID for Mailbox 5 (upper 16 bits)	Figure 11–5	11-10
722Ah	MSGCTRL5	CAN Message Control Field 5	Figure 11–7	11-11
722Ch	MBX5A	CAN 2 of 8 Bytes of Mailbox 5	Not shown	Not shown
722Dh	MBX5B	CAN 2 of 8 Bytes of Mailbox 5	Not shown	Not shown
722Eh	MBX5C	CAN 2 of 8 Bytes of Mailbox 5	Not shown	Not shown
722Fh	MBX5D	CAN 2 of 8 Bytes of Mailbox 5	Not shown	Not shown
7400h	GPTCON	GP Timer Control Register	Figure 7–12	7-31
7401h	T1CNT	GP Timer 1 Counter Register	Figure 7–5	7-19
7402h	T1CMPR	GP Timer 1 Compare Register	Figure 7–3	7-14
7403h	T1PR	GP Timer 1 Period Register	Figure 7–4	7-15
7404h	T1CON	GP Timer 1 Control Register	Figure 7–11	7-29
7405h	T2CNT	GP Timer 2 Counter Register	Figure 7–5	7-19
7406h	T2CMPR	GP Timer 2 Compare Register	Figure 7–3	7-14
7407h	T2PR	GP Timer 2 Period Register	Figure 7–4	7-15
7408h	T2CON	GP Timer 2 Control Register	Figure 7–11	7-29
7411h	COMCON	Compare Control Register	Figure 7–14	7-37
7413h	ACTR	Full-Compare Action Control Register	Figure 7–15	7-38

Table A–1. Programmable Registers on the TMS320F243/F241/C242 (Continued)

			Shown in	
Address	**Register**	**Name**	**Figure**	**Page**
7415h	DBTCON	Dead-Band Timer Control Register	Figure 7–17	7-42
7417h	CMPR1	Full-Compare Unit Compare Register 1	Not shown	Not shown
7418h	CMPR2	Full-Compare Unit Compare Register 2	Not shown	Not shown
7419h	CMPR3	Full-Compare Unit Compare Register 3	Not shown	Not shown
7420h	CAPCON	Capture Control Register	Figure 7–26	7-60
7422h	CAPFIFO	Capture FIFO Status Register	Figure 7–27	7-62
7423h	CAP1FIFO	Two-Level-Deep Capture FIFO Stack 1	Not shown	Not shown
7424h	CAP2FIFO	Two-Level-Deep Capture FIFO Stack 2	Not shown	Not shown
7425h	CAP3FIFO	Two-Level-Deep Capture FIFO Stack 3	Not shown	Not shown
7427h	CAP1FBOT	Capture 1 FIFO Bottom Stack Register	Not shown	Not shown
7428h	CAP2FBOT	Capture 2 FIFO Bottom Stack Register	Not shown	Not shown
7429h	CAP3FBOT	Capture 3 FIFO Bottom Stack Register	Not shown	Not shown
742Ch	EVIMRA	EV Interrupt Mask Register A	Figure 7–33	7-74
742Dh	EVIMRB	EV Interrupt Mask Register B	Figure 7–34	7-75
742Eh	EVIMRC	EV Interrupt Mask Register C	Figure 7–35	7-76
742Fh	EVIFRA	Interrupt Flag Register A	Figure 7–30	7-70
7430h	EVIFRB	Interrupt Flag Register B	Figure 7–31	7-72
7431h	EVIFRC	Interrupt Flag Register C	Figure 7–32	7-73
FF0Fh (I/O space)	FCMR	Flash Control Mode Register	Not shown	Not shown
FFFFh	WSGR	Wait State Generator Register	Figure 3–13	3-21

† Address in I/O space
‡ Address in program memory space

Notes: 1) CAN and SPI are not available in ’C242. Registers pertaining to these peripherals should not be accessed in ’C242.
2) WSGR is absent in ’F241 and ’C242.
3) FCMR is absent in ’C242.

Appendix B

Program Examples

This appendix provides:

- A brief introduction to the tools used for generating executable COFF files that run on the ’24x devices.
- Sample programs to test some of the peripherals available in the ’24x 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 (*24x.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

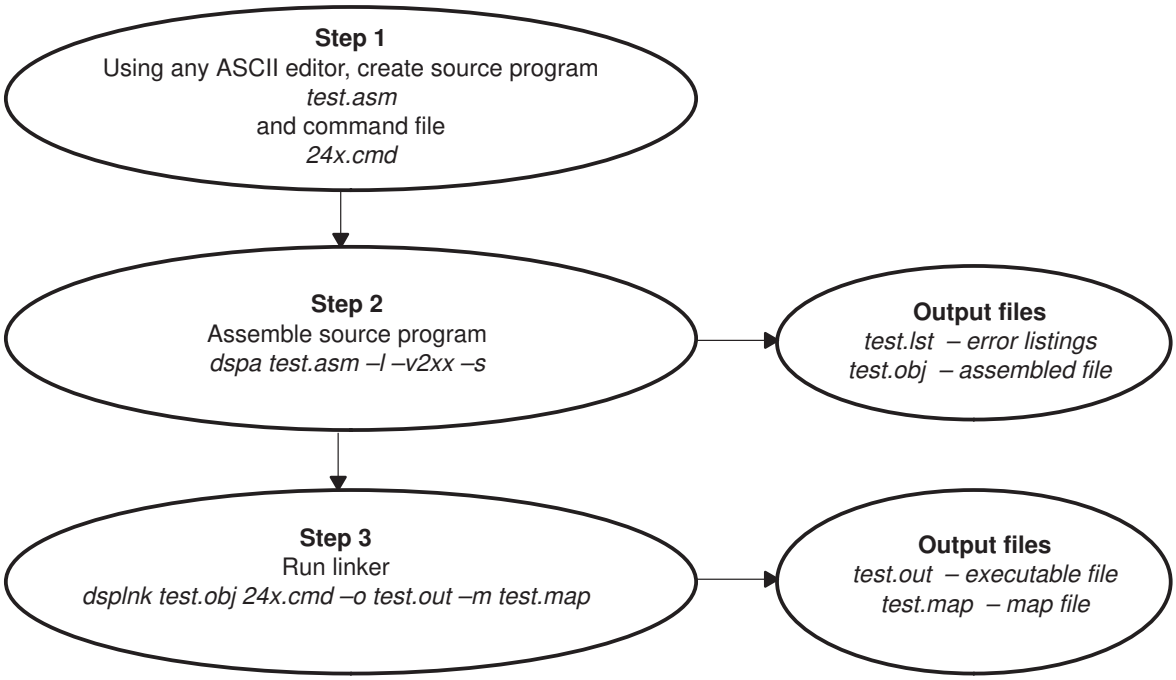


Table B–1. Common Files For All Example Programs

Program	Functional Description
24x_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.
24x.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 ’24x
ADC.asm	Program to check ADC of ’24x
GPIO_OUT.asm	Program that checks GPIO pins of ’24x as outputs
GPIO_IN.asm	Program that checks GPIO pins of ’24x 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 EV2
EV.CAPT.asm	Program to check the operation of capture units in EV2

B.2 Program Examples

```
/*************************************************************************/
/* File Name:  24x_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 .bss and .data sections need to be located elsewhere in data memory.

/*************************************************************************/

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:    24x.h
;
; Description:  241,242,243 register definitions.
;***************************************************************************

; 24x core registers
IMR             .set 0004h      ; Interrupt Mask Register
GREG            .set 0005h      ; Global memory allocation Register
IFR             .set 0006h      ; Interrupt Flag Register

; System configuration and interrupt registers

SCSR            .set 7018h      ; System Control &  Status Reg. X241/2/3 only.
DIN             .set 701Ch      ; Device Identification Register.
PIVR            .set 701Eh      ; Peripheral Interrupt Vector Reg. X241/2/3 only.
PIRQR0          .set 7010h      ; Periph Interrupt Request Reg 0. X241/2/3 only.
PIRQR1          .set 7011h      ; Periph Interrupt Request Reg 1. X241/2/3 only.

; External interrupt configuration registers
XINT1CR         .set 7070h      ; Ext. interrupt 1 config reg for X241/2/3 only.
XINT2CR         .set 7071h      ; External interrupt 2 config. X241/2/3 only.

; Digital I/O registers
OCRA            .set 7090h      ; Output Control Reg A
OCRB            .set 7092h      ; Output Control Reg B
PADATDIR        .set 7098h      ; I/O port A Data & Direction reg.
PBDATDIR        .set 709Ah      ; I/O port B Data & Direction reg.
PCDATDIR        .set 709Ch      ; I/O port C Data & Direction reg.
PDDATDIR        .set 709Eh      ; I/O port D Data & Direction reg.

; Watchdog (WD) registers
WDCNTR          .set 7023h      ; WD Counter reg
WDKEY           .set 7025h      ; WD Key reg
WDCR            .set 7029h      ; WD Control reg

; ADC registers
ADCTRL1         .set 7032h      ; ADC Control Reg1
ADCTRL2         .set 7034h      ; ADC Control Reg2
ADCFIFO1        .set 7036h      ; ADC DATA REG FIFO for ADC1
ADCFIFO2        .set 7038h      ; ADC DATA REG FIFO for ADC2

; SPI registers
SPICCR          .set 7040h      ; SPI Config Control Reg
SPICTL          .set 7041h      ; SPI Operation Control Reg
SPISTS          .set 7042h      ; SPI Status Reg
SPIBRR          .set 7044h      ; SPI Baud rate control reg
SPIRXEMU        .set 7046h      ; SPI Emulation buffer reg
SPIRXBUF        .set 7047h      ; SPI Serial receive buffer reg
SPITXBUF        .set 7048h      ; SPI Serial transmit buffer reg
SPIDAT          .set 7049h      ; SPI Serial data reg
SPIPRI          .set 704Fh      ; SPI Priority control reg
```

```
; SCI registers
SCICCR          .set 7050h      ; SCI Communication control reg
SCICTL1         .set 7051h      ; SCI Control reg1
SCIHBAUD        .set 7052h      ; SCI Baud Rate MSbyte reg
SCILBAUD        .set 7053h      ; SCI Baud Rate LSbyte reg
SCICTL2         .set 7054h      ; SCI Control reg2
SCIRXST         .set 7055h      ; SCI Receiver Status reg
SCIRXEMU        .set 7056h      ; SCI Emulation Data Buffer reg
SCIRXBUF        .set 7057h      ; SCI Receiver Data buffer reg
SCITXBUF        .set 7059h      ; SCI Transmit Data buffer reg
SCIPRI          .set 705Fh      ; SCI Priority control reg

; Event Manager (EV) registers
GPTCON          .set 7400h      ; GP Timer control register.
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.

COMCON          .set 7411h      ; Compare control register.
ACTR            .set 7413h      ; Full compare action control register.
DBTCON          .set 7415h      ; Dead-band timer control register.

CMPR1           .set 7417h      ; Full compare unit compare register1.
CMPR2           .set 7418h      ; Full compare unit compare register2.
CMPR3           .set 7419h      ; Full compare unit compare register3.

CAPCON          .set 7420h      ; Capture control register.
CAPFIFO         .set 7422h      ; Capture FIFO status register.

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 Register of Capture FIFO Stack 1
CAP2FBOT         .set 7428h      ; Bottom Register of Capture FIFO Stack 2
CAP3FBOT         .set 7429h      ; Bottom Register of Capture FIFO Stack 3

EVIMRA          .set 742Ch      ; Group A Interrupt Mask Register
EVIMRB          .set 742Dh      ; Group B Interrupt Mask Register
EVIMRC          .set 742Eh      ; Group C Interrupt Mask Register

EVIFRA          .set 742Fh      ; Group A Interrupt Flag Register
EVIFRB          .set 7430h      ; Group B Interrupt Flag Register
EVIFRC          .set 7431h      ; Group C Interrupt Flag Register

; CAN registers. F241/3 only.
CANMDER         .set 7100h      ; CAN Mailbox Direction/Enable reg
CANTCR          .set 7101h      ; CAN Transmission Control Reg
CANRCR          .set 7102h      ; CAN Recieve COntrol Reg
CANMCR          .set 7103h      ; CAN Master Control Reg
```

```
CANBCR2         .set 7104h      ; CAN Bit COnfig Reg 2
CANBCR1         .set 7105h      ; CAN Bit Config Reg 1
CANESR          .set 7106h      ; CAN Error Status Reg
CANGSR          .set 7107h      ; CAN Global Status Reg
CANCEC          .set 7108h      ; CAN Trans and Rcv Err counters
CANIFR          .set 7109h      ; CAN Interrupt Flag Registers
CANIMR          .set 710ah      ; CAN Interrupt Mask Registers
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 Reg
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 ’24x. 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  24x.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 if Vccp = 5V

          KICK_DOG
```

```
;=========================================================================
; 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     #OCRB>>7              ;Set up the GPIO pins to function
              SPLK    #003CH, OCRB          ;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 '24x
* 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        24x.h
          .bss     GPR0,1
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 if Vccp = 5V

          KICK_DOG
          SPLK     #0h,GPR0        ;Set wait state generator for:
          OUT      GPR0,WSGR       ;Program Space, 0-7 wait states
          LDP      #00E0h
          SPLK     #0000h,SCSR     ;CLKOUT = CPUCLK
```

```
;=========================================================================
;SCI TRANSMISSION TEST - starts here
;=========================================================================

SCI:          LDP     #0E1h
              SPLK    #0FFFFh,OCRA

              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    #0103h, SCILBAUD        ;Baud Rate=9600 b/s

              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 CHECK ADC OF 24x
* This program checks the conversion ability of selected ADC channels
* Simple program without using any interrupts, performs simulataneous
* conversion of any two channels. The results can be viewed in the
* ADC FIFO's.
;=============================================================================
   .title " 24x ADC1"

       .bss   GPR0,1
       .include 24x.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 if VCCP=5V (706F)
          KICK_DOG
          SPLK     #0h,GPR0         ;Set wait state generator for:
          OUT      GPR0,WSGR        ;Program Space, 0-7 wait states
                                    ;(E1=225)(E0*80=7080)
          LDP      #0E0h
LOOP      LACL     ADCFIFO1         ; Clear ADC FIFOs
          LACL     ADCFIFO1
          LACL     ADCFIFO2
          LACL     ADCFIFO2
```

```
;*************************************************************************
;***********          Configure all ADC registers                ***********
;*************************************************************************

    SPLK   #0000000000000000b, ADCTRL2
;           ||||||||||||||||
;           FEDCBA9876543210

; bit 0-2 000 Prescaler value
; bit 3-4     FIFO2 status
; bit 5       Reserved
; bit 6-7     FIFO1 status
; bit 8       Reserved
; bit 9   0   Mask external SOC input
; bit A   0   Mask EV SOC input
; bit B-F     Reserved

LOOP:  SPLK   #1101100000000001b, ADCTRL1
;              ||||||||||||||||
;              FEDCBA9876543210

; bit 0   1   Start of conversion
; bit 1-3 000 Channel 0 address
; bit 4-6 001 Channel 1 address
; bit 7   0   End of convert
; bit 8   1   Interrupt flag - write 1 to clear
; bit 9   1   Interrupt mask - enable with 1, mask 0
; bit A   0   Continuous run mode disabled
; bit B   1   Enable ADC2
; bit C   1   Enable ADC1
; bit D   1   Immediate start - 0 = no action
; bit E   1   Free run - ignore suspend
; bit F   1   Soft - Not applicable with bit E = 1

           B   LOOP
PHANTOM    KICK_DOG            ;Resets WD counter
           B   PHANTOM
```

```
;===============================================================================
* File Name: GPIO_OUT
* Description:  This program writes a running pattern of zeros to the GPIO pins of
* 24x. It ouputs a total of 8 bit patterns to all the four GPIO ports.
* Each bit pattern forces a particular bit low and forces the other 7
* bits high. This goes on in an endless loop.
* .data section must be loaded at 60h.
;===============================================================================
        .title " 24x GPIO"

        .data                   ; Loaded @ 60h 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 24x.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 if VCCP=5V (706F)
          KICK_DOG
          SPLK    #0h,GPR0      ;Set wait state generator for:
          OUT     GPR0,WSGR     ;Program Space, 0-7 wait states
                                ;(E1=225)(E0*80=7080)
          LDP     #00E1h        ;
          SPLK    #0103h,OCRB   ;Select IOPC0,1 & IOPD0,1

          SPLK    #0FFFFh, PADATDIR  ; All pins are o/p's
          SPLK    #0FFFFh, PBDATDIR  ; and forced high
          SPLK    #0FFFFh, PCDATDIR  ;
          SPLK    #0FFFFh, PDDATDIR  ;
```

```
MAIN    LDP     #0
        LAR     AR0,#60h        ; AR0 points to bit pattern
        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 4 GPIO ports
        SACL    PCDATDIR
        SACL    PDDATDIR
        CALL    DELAY
        MAR     *,AR1           ; Check if all 8 patterns have
        BANZ    LOOP            ; been output. If not, continue.
        B       MAIN

DELAY   LAR     AR2,#0FFFFh
D_LOOP  RPT     #015h
        NOP
        BANZ    D_LOOP
        RET

PHANTOM KICK_DOG                ;Resets WD counter
        B       PHANTOM
```

```
;===============================================================================
* File Name: GPI0_IN
* Description: PROGRAM TO CHECK GPIO BITS OF 24x as inputs
* All GPIO bits are programmed as inputs and the values read from the
* GPIO pins are written in 60h,61h,62h,63h of DM
================================================================================
          .title " 24x GPIO"


          .include 24x.h
          .bss      GPR0,1
;---------------------------------------------------------------
; 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 if VCCP=5V (706F)
          KICK_DOG
          SPLK    #0h,GPR0          ;Set wait state generator for:
          OUT     GPR0,WSGR         ;Program Space, 0-7 wait states
                                    ;(E1=225)(E0*80=7080)
          LDP     #00E1h
          SPLK    #0103h,OCRB       ;Select IOPC0,1 & IOPD0,1

          SPLK    #0h, PADATDIR     ;All GPIO pins are programmed
          SPLK    #0h, PBDATDIR     ;as inputs
          SPLK    #0h, PCDATDIR     ;
          SPLK    #0h, PDDATDIR     ;
```

```
MAIN      LDP     #0                ;This loop reads the level on
          LAR     AR0,#60h          ;the GPIO pins. The bit patterns
          MAR     *,AR0             ;read from the 4 GPIO ports
          LDP     #00E1h            ;is copied in the data memory
          LACL    PADATDIR
          SACL    *+
          LACL    PBDATDIR
          SACL    *+
          LACL    PCDATDIR
          SACL    *+
          LACL    PDDATDIR
          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 24x.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     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,OCRB     ; 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 24x *
* 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 24x.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     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 EV
* 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 "24x.h"             ;Variable and register declaration

;-------------------------------------------------------------
;Vector address declarations
;-------------------------------------------------------------
                .sect   "vectors"

RSVECT          B    START          ;Reset Vector
INT1            B    PHANTOM        ;Interrupt Level 1
INT2            B    GISR2          ;Interrupt Level 2
INT3            B    PHANTOM        ;Interrupt Level 3
INT4            B    PHANTOM        ;Interrupt Level 4
INT5            B    PHANTOM        ;Interrupt Level 5
INT6            B    PHANTOM        ;Interrupt Level 6
RESERVED        B    PHANTOM        ;Reserved
SW_INT8         B    PHANTOM        ;User S/W Interrupt
SW_INT9         B    PHANTOM        ;User S/W Interrupt
SW_INT10        B    PHANTOM        ;User S/W Interrupt
SW_INT11        B    PHANTOM        ;User S/W Interrupt
SW_INT12        B    PHANTOM        ;User S/W Interrupt
SW_INT13        B    PHANTOM        ;User S/W Interrupt
SW_INT14        B    PHANTOM        ;User S/W Interrupt
SW_INT15        B    PHANTOM        ;User S/W Interrupt
SW_INT16        B    PHANTOM        ;User S/W Interrupt
TRAP            B    PHANTOM        ;Trap vector
NMI             B    PHANTOM        ;Non-maskable Interrupt
EMU_TRAP        B    PHANTOM        ;Emulator Trap
SW_INT20        B    PHANTOM        ;User S/W Interrupt
SW_INT21        B    PHANTOM        ;User S/W Interrupt
SW_INT22        B    PHANTOM        ;User S/W Interrupt
SW_INT23        B    PHANTOM        ;User S/W Interrupt

;-------------------------------------------------------------------
;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    #006Fh, WDCR                 ;Disable WD if VCCP=5V
          KICK_DOG
          MAR     *,AR0
          LDP     #0E1h                        ;Peripheral page
          SPLK    #1111111100000000b,PBDATDIR
                                               ;set IOPBn as outputs,0

* Load TIMER 1 registers
          LDP     #GPTCON >> 7h                ;Peripheral page
          SPLK    #0000000000000000b,GPTCON
          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,EVIMRA
                                               ;Enable OV,U,C,P interrupt bits
          SPLK    #0000011110000000b,EVIFRA
                                               ;clear interrupts
          LDP     #0
          SPLK    #0000000000000010b,IMR
          CLRC    INTM                         ;Enable INT2

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     #GPTCON >> 7h        ;Peripheral page
          LACC    #0400h               ;clear overflow int. flag
          SACL    EVIFRA               ;in EVIFRA
          CLRC    INTM                 ;Enable all interrupts
          RET

SISR29:
          LDP     #0E1h                ;Peripheral page
          SPLK    #0FF02h,PBDATDIR     ;Set IOPB1
          CALL    DELAY
          LDP     #GPTCON >> 7h        ;Peripheral page
          LACC    #0200h               ;clear underflow int. flag
          SACL    EVIFRA               ;in EVIFRA
          CLRC    INTM                 ;Enable all interrupts
          RET

SISR28:
          LDP     #0E1h                ;Peripheral page
          SPLK    #0FF04h,PBDATDIR     ;Set IOPB2
          CALL    DELAY
          LDP     #GPTCON >> 7h        ;Peripheral page
          LACC    #0100h               ;clear compare int. flag
          SACL    EVIFRA               ;in EVIFRA
          CLRC    INTM                 ;Enable all interrupts
          RET

SISR27:
          LDP     #0E1h                ;Peripheral page
          SPLK    #0FF08h,PBDATDI      ;Set IOPB3
          CALL    DELAY
          LDP     #GPTCON >> 7h        ;Peripheral page
          LACC    #0080h               ;clear period int. flag
          SACL    EVIFRA               ;in EVIFRA
          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

PHANTOM:  RET
          .end                         ;Assembler module end directive –optional
```

```
;=============================================================================
* File Name: EV_Capt.asm
* Description: PROGRAM TO CHECK THE OPERATION OF CAPTURE UNITS IN EV
* Function: To test CAP1, CAP2 & CAP3 using Timer 1
* Mode: Timer 1 at x/128
* Output: Port C bits 2,3,4 are toggled by capture interrupt
* Set up: Connect T1CMP to CAP1,2,3 inputs
* ISR: CAP1,2,3 interrupts read, CAPFIFO
*      CAP1 on rising edge,  FIFO value at 70h,71h, toggle OPC2
*      CAP2 on falling edge, FIFO value at 72h,73h, toggle OPC3
*      CAP3 on both edges,   FIFO value at 74h,75h, toggle OPC4
;=============================================================================

          .title " EV capture test"     ;Title
          .include "24x.h"              ;Variable and register declaration

;-------------------------------------------------------------
;Vector address declarations
;-------------------------------------------------------------
                .sect   "vectors"

RSVECT          B    START          ;Reset Vector
INT1            B    PHANTOM        ;Interrupt Level 1
INT2            B    PHANTOM        ;Interrupt Level 2
INT3            B    PHANTOM        ;Interrupt Level 3
INT4            B    GISR4          ;Interrupt Level 4
INT5            B    PHANTOM        ;Interrupt Level 5
INT6            B    PHANTOM        ;Interrupt Level 6
RESERVED        B    PHANTOM        ;Reserved
SW_INT8         B    PHANTOM        ;User S/W Interrupt
SW_INT9         B    PHANTOM        ;User S/W Interrupt
SW_INT10        B    PHANTOM        ;User S/W Interrupt
SW_INT11        B    PHANTOM        ;User S/W Interrupt
SW_INT12        B    PHANTOM        ;User S/W Interrupt
SW_INT13        B    PHANTOM        ;User S/W Interrupt
SW_INT14        B    PHANTOM        ;User S/W Interrupt
SW_INT15        B    PHANTOM        ;User S/W Interrupt
SW_INT16        B    PHANTOM        ;User S/W Interrupt
TRAP            B    PHANTOM        ;Trap vector
NMI             B    PHANTOM        ;Non-maskable Interrupt
EMU_TRAP        B    PHANTOM        ;Emulator Trap
SW_INT20        B    PHANTOM        ;User S/W Interrupt
SW_INT21        B    PHANTOM        ;User S/W Interrupt
SW_INT22        B    PHANTOM        ;User S/W Interrupt
SW_INT23        B    PHANTOM        ;User S/W Interrupt

del             .set 0fffh          ;define delay

;-------------------------------------------------------------
;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    #006Fh, WDCR              ;Disable WD if VCCP=5V
          KICK_DOG
          LAR     ar7,#del                  ;Load AR7 with delay value
          MAR     *,ar7                     ;Set ARP to ar7
          LDP     #PCDATDIR >> 7h           ;Peripheral page
          SPLK    #0001110000000000b,PCDATDIR
                                            ;set IOPC 2,3,4 as outputs
          SPLK    #1111111111111111b,OCRA
                                            ;enable all EV signals

* Load TIMER 1 registers
          LDP     #GPTCON >> 7h             ;Peripheral page
          SPLK    #0000000001001001b,GPTCON
                                            ;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    #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    #1111111111111111b,T1PR
          SPLK    #0011111100000000b,T1CMPR
          SPLK    #0000000000000000b,EVIMRA
          SPLK    #0000000000000000b,EVIMRB
                                            ;disable group A,B interrupts
```

```
* Load Capture registers
          SPLK    #0011000001101100b,CAPCON
                                            ;0 clear capture registers
                                            ;01-enable Capture 1,2 disable QEP
                                            ;1 -enable Capture 3
                                            ;0 -reserved
                                            ;0 -Use GPTimer 2 for CAP3
                                            ;0 -Use GPTimer 2 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,EVIMRC
                                            ;0000 0000 0000 0
                                            ;111, enable CAP3,CAP2,CAP1 interrupts

          LDP     #0
          SPLK    #0000000000001000b,IMR    ;Enable INT4
          CLRC    INTM                      ;Enable interrupts globally
          LDP     #GPTCON >> 7h             ;Peripheral page

wait:     NOP                               ;main loop
          B       wait

GISR1:    ret                               ;Int1 GISR
GISR2:    ret                               ;Int2 GISR
GISR3:    ret                               ;Int3 GISR

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
          RET

SISR33:                                     ;CAP1 SISR
          LDP     #PCDATDIR >> 7h           ;Peripheral page
          SPLK    #3c38h,PCDATDIR           ;clear OPC2
          CALL    DELAY
          SPLK    #3c3ch,PCDATDIR           ;set OPC2
          LDP     #GPTCON >> 7h             ;Peripheral page
          SPLK    #0001h,EVIFRC             ;clear Capture flag
          LDP     #0h
          BLDD    #CAP1FIFO,70h
          BLDD    #CAP1FIFO,71h
          CLRC    INTM
          RET
```

```
SISR34:                                     ;CAP2 SISR
          LDP     #PCDATDIR >> 7h           ;Peripheral page
          SPLK    #3c34h,PCDATDIR           ;clear OPC3
          CALL    DELAY
          SPLK    #3c3ch,PCDATDIR           ;set OPC3
          LDP     #GPTCON >> 7h             ;Peripheral page
          SPLK    #0002h,EVIFRC             ;clear Capture flag
          LDP     #0h
          BLDD    #CAP2FIFO,72h
          BLDD    #CAP2FIFO,73h
          CLRC    INTM
          RET

SISR35:                                     ;CAP3 SISR
          LDP     #PCDATDIR >> 7h           ;Peripheral page
          SPLK    #3c2ch,PCDATDIR           ;clear OPC4
          CALL    DELAY
          SPLK    #3c3ch,PCDATDIR           ;set OPC4
          LDP     #GPTCON >> 7h             ;Peripheral page
          SPLK    #0004h,EVIFRC             ;clear Capture flag
          LDP     #0h
          BLDD    #CAP3FIFO,74h
          BLDD    #CAP3FIFO,75h
          CLRC    INTM
          RET

DELAY:    RPT     #0ffh                     ;sub-routine for delay.
          NOP
          BANZ    DELAY,ar7
          LAR     ar7,#del
          RET

PHANTOM: ret
          .end
```

Appendix C

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.

boot loader: A built-in segment of code that transfers code from an external source to a 16-bit external program destination at reset.

$\overline{\text{BR}}$: *Bus request pin*. This pin is tied to the $\overline{\text{BR}}$ signal, which is asserted when a global data memory access is initiated.

branch: A switching of program control to a nonsequential program-memory address.

BRR: The value in the baud select registers.

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, CLKOUT1.

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 CLKOUT1 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.

global data space: One of the four ’24x address spaces. The global data space can be used to share data with other processors within a system and can serve as additional data space. See also *local data space*.

GREG: *Global memory allocation register*. A memory-mapped register used for specifying the size of the global data memory. Addresses not allocated by the GREG for global data memory are available for local data memory.

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)*.

$\overline{\text{IS}}$: *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*.

local data space: The portion of data-memory addresses that are not allocated as global by the global memory allocation register (GREG). If none of the data-memory addresses are allocated for global use, all of data space is local. See also *global data space*.

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*.

microcomputer 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 microcomputer mode. MP/$\overline{\text{MC}}$ high selects microprocessor mode; MP/$\overline{\text{MC}}$ low selects microcomputer 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 $\times$ 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*.

$\overline{\text{NMI}}$: A hardware interrupt that uses the same logic as the maskable interrupts but cannot be masked. It is often used as a soft reset. See also *maskable interrupt*; *nonmaskable interrupt*.

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

RAMEN: *RAM enable pin*. This pin enables or disables on-chip single-access RAM.

$\overline{\text{RD}}$: *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.

$\overline{\text{STRB}}$: *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).

$\overline{\text{WE}}$: *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.

Appendix D

Updates To This Document

This appendix provides a summary of the updates in this version of the document. Updates within paragraphs appear in a **bold typeface**.

In general, all technical changes noted in SPRZ151A, the Manual Update Sheet for the B revision of this user's guide, have been incorporated in this version of the document.

Page:	Changed or Added:
1-6	Added the following note below Table 1–1, *'F243/F241/C242 Device Configurations*: Note: The PMT pin should be connected to ground for proper operation.
3–21	Removed the "Rs" indicating read access from below the register in Figure 3–13, *'F243 Wait-State Generator Control Register (WSGR).*
7–29	Changed the bit 7 description in Figure 7–11, *GP Timer x Control Register (TxCON; x = 1 or 2), to T2SWT1.*
Chp. 10	Added example waveform figures to the end of the chapter.
Chp. 11	In general, Chapter 11, *CAN Controller Module*, has been revised to make it more readable. New CAN example programs have been provided in Appendix B.
11–32	Added the following note after the bulleted list in Section 11.6, *Interrupt Logic*: 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.
12–8	Changed the bit 7 description for WDCR in Table 12–2, *WD Module Control Registers*, to Reserved.
12–10	Changed the bit 7 description in Figure 12–4, *WD Timer Control Register (WDCR),* to Reserved.

Index

A

B

C

D

E

F

G

I

L

M

N

O

P

Q

R

S

T

W

X