TMS321x281x DSP System Control and Interrupts Reference Guide

Literature Number: SPRU078B
April 2002 − Revised October 2004

Texas Instruments

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

Preface

Read This First

About This Manual

This guide describes how various 281x digital signal processor (DSP) system controls and interrupts work with peripherals. It includes information on the:

- ❑ Flash and one-time programmable (OTP) memories
- ❑ Code security module (CSM), which is a security feature incorporated in 28x devices
- ❑ Clocking mechanisms including the oscillator, PLL, the watchdog function, and the low-power modes
- ❑ GPIO MUX registers used to select the operation of shared pins
- ❑ Peripheral frames and the device emulation registers
- ❑ Peripheral interrupt expansion (PIE) block that multiplexes numerous interrupt sources into a smaller set of interrupt inputs

Related Documentation From Texas Instruments

The following books describe the TMS320x281x and related support tools that are available on the TI website.

TMS320F2810, TMS320F2811, TMS320F2812, TMS320C2810, TMS320C2811, and TMS320C2812 Digital Signal Processors (literature number SPRS174) data sheet contains the electrical and timing specifications for these devices, as well as signal descriptions and pinouts for all of the available packages.

TMS320R2811 and TMS320R2812 Digital Signal Processors (literature number SPRS257) data sheet contains the electrical and timing specifications for these devices, as well as signal descriptions and pinouts for all of the available packages.

TMS320C28x DSP CPU and Instruction Set Reference Guide (literature number SPRU430) describes the central processing unit (CPU) and the assembly language instructions of the TMS320C28x™ fixed-point digital signal processors (DSPs). It also describes emulation features available on these DSPs.

TMS320x281x Analog-to-Digital Converter (ADC) Reference Guide (literature number SPRU060) describes the ADC module. The module is a 12−bit pipelined ADC. The analog circuits of this converter, referred to as the core in this document, include the front-end analog multiplexers (MUXs), sample−and−hold (S/H) circuits, the conversion core, voltage regulators, and other analog supporting circuits. Digital circuits, referred to as the wrapper in this document, include programmable conversion sequencer, result registers, interface to analog circuits, interface to device peripheral bus, and interface to other on-chip modules.

TMS320x281x Boot ROM Reference Guide (literature number SPRU095) describes the purpose and features of the bootloader (factory-programmed boot-loading software). It also describes other contents of the device on-chip boot ROM and identifies where all of the information is located within that memory.

TMS320x281x, 280x Enhanced Controller Area Network (eCAN) Reference Guide (literature number SPRU074) describes the eCAN that uses established protocol to communicate serially with other controllers in electrically noisy environments. With 32 fully configurable mailboxes and time-stamping feature, the eCAN module provides a versatile and robust serial communication interface. The eCAN module implemented in the C28x DSP is compatible with the CAN 2.0B standard (active).

TMS320x281x Event Manager (EV) Reference Guide (literature number SPRU065) describes the EV modules that provide a broad range of functions and features that are particularly useful in motion control and motor control applications. The EV modules include general-purpose (GP) timers, full-compare/PWM units, capture units, and quadrature-encoder pulse (QEP) circuits.

TMS320x281x External Interface (XINTF) Reference Guide (literature number SPRU067) describes the external interface (XINTF) of the 28x digital signal processors (DSPs).

TMS320x281x Multi-channel Buffered Serial Ports (McBSPs) Reference Guide (literature number SPRU061) describes the McBSP) available on the C28x devices. The McBSPs allow direct interface between a DSP and other devices in a system.

TMS320x281x, 280x Peripheral Reference Guide (literature number SPRU566) describes the peripheral reference guides of the 28x digital signal processors (DSPs).

TMS320x281x, 280x Serial Communication Interface (SCI) Reference Guide (literature number SPRU051) describes the SCI that is a two-wire

asynchronous serial port, commonly known as a UART. The SCI modules support digital communications between the CPU and other asynchronous peripherals that use the standard non-return-to-zero (NRZ) format.

TMS320x281x, 280x Serial Peripheral Interface (SPI) Reference Guide (literature number SPRU059) describes the SPI − 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 used for communications between the DSP controller and external peripherals or another controller.

TMS320x281x System Control and Interrupts Reference Guide (literature number SPRU078) describes the various interrupts and system control features of the 281x digital signal processors (DSPs).

The TMS320C28x Instruction Set Simulator Technical Overview (literature number SPRU608) describes the simulator, available within the Code Composer Studio for TMS320C2000 IDE, that simulates the instruction set of the C28x core.

TMS320C28x DSP/BIOS Application Programming Interface (API) Reference Guide (literature number SPRU625) describes development using DSP/BIOS.

3.3 V DSP for Digital Motor Control Application Report (literature number SPRA550). New generations of motor control digital signal processors (DSPs) lower their supply voltages from 5 V to 3.3 V to offer higher performance at lower cost. Replacing traditional 5-V digital control circuitry by 3.3-V designs introduce no additional system cost and no significant complication in interfacing with TTL and CMOS compatible components, as well as with mixed voltage ICs such as power transistor gate drivers. Just like 5-V based designs, good engineering practice should be exercised to minimize noise and EMI effects by proper component layout and PCB design when 3.3-V DSP, ADC, and digital circuitry are used in a mixed signal environment, with high and low voltage analog and switching signals, such as a motor control system. In addition, software techniques such as Random PWM method can be used by special features of the Texas Instruments (TI) TMS320x24xx DSP controllers to significantly reduce noise effects caused by EMI radiation.

This application report reviews designs of 3.3-V DSP versus 5-V DSP for low HP motor control applications.The application report first de-

scribes a scenario of a 3.3-V-only motor controller indicating that for most applications, no significant issue of interfacing between 3.3 V and 5 V exists. Cost-effective 3.3-V − 5-V interfacing techniques are then discussed for the situations where such interfacing is needed. On-chip 3.3-V ADC versus 5-V ADC is also discussed. Sensitivity and noise effects in 3.3-V and 5-V ADC conversions are addressed. Guidelines for component layout and printed circuit board (PCB) design that can reduce system's noise and EMI effects are summarized in the last section.

Thermo-Electric Cooler Control Using a TMS320F2812 DSP & DRV592 Power Amplifier Application Note (literature number SPRA873). This application report presents a thermoelectric cooler system consisting of a Texas Instruments TMS320F2812 digital signal processor (DSP) and DRV592 power amplifier. The DSP implements a digital proportional-integral-derivative feedback controller using an integrated 12-bit analog-to-digital converter to read the thermistor, and direct output of pulse-width-modulated waveforms to the H-bridge DRV592 power amplifier. The system presented provides up to 6.1 watts of heating or cooling to the laser mount, although the DRV592 amplifier is actually capable of delivering up to 15 watts when configured appropriately. The closed-loop TEC system is seen to achieve ±0.0006°C temperature accuracy, depending on the needed operating temperature range, with a step response settling time of 14 to 16 seconds. A complete description of the experimental system, along with software and software operating instructions, are provided.

Running an Application from Internal Flash Memory on the TMS320F281x DSP Application Report (literature number SPRA958). Several special requirements exist for running an application from on-chip flash memory on the TMS320F28x DSP. These requirements generally do not manifest themselves during development in RAM since the Code Composer Studio™ debugger can mask problems associated with initialized sections and how they are linked to memory. This application report covers the requirements needed to properly configure application software for execution from on-chip flash memory. Requirements for both DSP/BIOS™ and non-DSP/BIOS projects are presented. Some performance considerations and techniques are also discussed. Example code projects are included that run from on-chip flash on the eZdsp™ F2812 development board (or alternately any F2812, F2811, or F2810 DSP board). Code examples that run from internal RAM are also provided for completeness. These code examples provide a starting point for code development, if desired.

Trademarks

Code Composer Studio and C28x are trademarks of Texas Instruments.

This page intentionally left blank.

Contents

Figures

Tables

This page intentionally left blank.

Chapter 1

Memory

This chapter describes how Flash and one-time programmable (OTP) memories can be used with the 28x digital signal processor (DSP) device and peripherals. It also includes the registers associated with memory.

Topic **Page**

Flash and OTP Memory

1.1 Flash and OTP Memory

This section describes how to configure two kinds of memory − Flash and one-time programmable (OTP).

1.1.1 Flash Memory

The on-chip Flash in Flash devices is uniformly mapped in both program and data memory space. This Flash memory is always enabled on 28x devices and features:

- Multiple sectors
- Code security
- Low power modes
- Configurable waitstates that can be adjusted based on CPU frequency
- Flash pipeline mode for improved performance of linear code execution

1.1.2 OTP Memory

The 1K × 16 block of one-time programmable (OTP) memory can be used to program data or code. This block can be programmed one time and cannot be erased.

1.2 Flash and OTP Power Modes

The following operating states apply to the Flash and OTP memory:

- Reset or Sleep State: This is the state after a device reset. In this state, the bank and pump are in a sleep state (lowest power). A CPU read or fetch accesses to the Flash/OTP memory map area stalls the CPU. This access automatically initiates a change in power modes to the active or read state.
- Standby State: In this state, the bank and pump are in standby power mode state. A CPU read or fetch accesses to the Flash/OTP memory map area stalls the CPU. This access automatically initiates a change in power modes to the active state.
- Active or Read State: In this state, the bank and pump are in active power mode state (highest power). The CPU read or fetch access wait states to the Flash/OTP memory map area is controlled by the FBANKWAIT and FOTPWAIT registers. A prefetch mechanism called Flash pipeline can also be enabled for improving fetch performance for linear code execution.

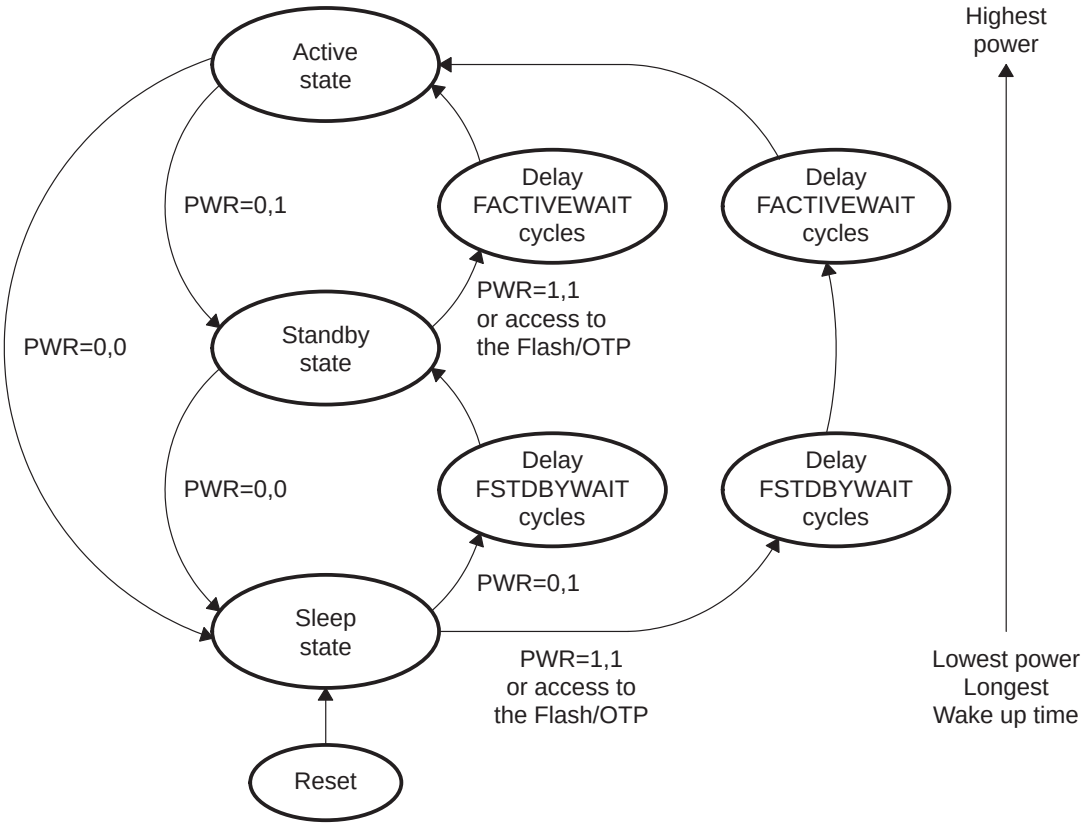
The Flash/OTP bank and pump are always in the same power mode during read or execution operations from the Flash/OTP. See Figure 1−1 for a graphic depiction of the states.

You can change the current Flash/OTP memory power state as follows:

- To move to a lower power state: Change the PWR mode bits from a higher power mode to a lower power mode. This change instantaneously moves the Flash/OTP bank to the lower power state. This register should be accessed only by code running outside the Flash/OTP memory.
- To move from a lower power state to a higher power state, there are two options:
  - Change the FPWR register from a lower state to a higher state. This access brings the Flash/OTP memory to the higher state.
  - Access the Flash or OTP memory by a read access or program fetch access. This access automatically brings the Flash/OTP memory to the active state.

There is a delay when moving from a lower power state to a higher one. See Figure 1−1. This delay is required to allow the Flash to stabilize at the higher power mode. If any access to the Flash/OTP memory occurs during this delay the CPU automatically stalls until the delay is complete.

Figure 1−1. Flash Power Mode State Diagram



The duration of the delay is determined by the FSTDBYWAIT and FACTIVEWAIT registers. Moving from the sleep state to a standby state is delayed by a count determined by the FSTDBYWAIT register. Moving from the standby state to the active state is delayed by a count determined by the FACTIVEWAIT register. Moving from the sleep mode (lowest power) to the active mode (highest power) is delayed by FSTDBYWAIT + FACTIVEWAIT.

1.2.1 Flash and OTP Performance

CPU read or fetch operations to the Flash/OTP can take one of the following forms:

- ❑ 32-bit instruction fetch
- ❑ 16 or 32-bit data space read
- ❑ 16-bit program space read

Once Flash is in the active power state, then a read or fetch access to the bank memory map area can be classified as three types:

- Flash Memory Random Access: The number of wait states, for a random access, is configured by the RANDWAIT bits in the FBANKWAIT register. This register defaults to a worst-case count and the user needs to program the appropriate number of wait states to improve performance based on the CPU clock rate and the access time of the Flash.
- Flash Memory Paged Access: The Flash array is organized into rows and columns. The rows contain 2048 bits of information. The first access to a row is considered a random access. Subsequent accesses within the same row can be made with a faster access time. This is termed a PAGE access.

  The Flash can take advantage of this by the user configuring a lower number of wait states in the PAGEWAIT bits in the FBANKWAIT register. This mode works for data space and program space reads as well as instruction fetches. See the device data sheet for more information on the access time of the Flash.
- OTP Access: Read or fetch accesses to the OTP are controlled by the OTPWAIT register bits in the FOTPWAIT register. Accesses to the OTP take longer than the Flash and there is no paged mode.

Notes:
1) Writes to the Flash/OTP memory map area are ignored. They complete in a single cycle.
2) When the Code Security Module (CSM) is secured, reads to the Flash/OTP memory map area from outside the secure zone take the same number of cycles as a normal access. However, the read operation returns a zero.
3) The Flash supports 0-wait accesses when the PAGEWAIT bits are set to zero. This assumes that the CPU speed is low enough to accommodate the access time.

1.2.2 28x Flash Pipeline Mode

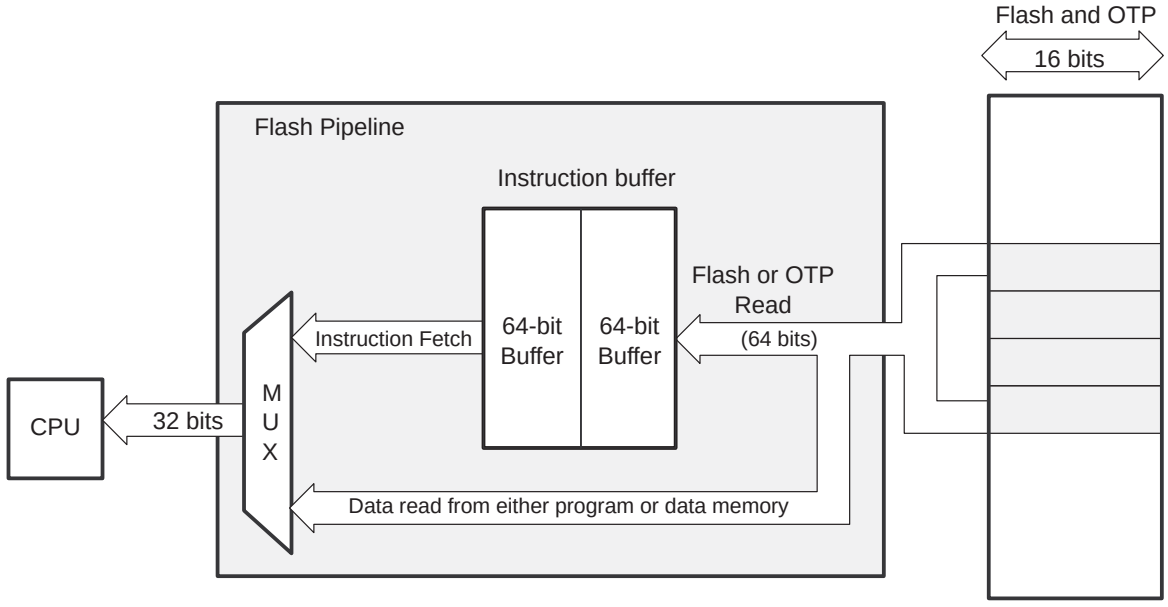
Flash memory is typically used to store application code. During code execution, instructions are fetched from sequential memory addresses, except when a discontinuity occurs. Usually the portion of the code that resides in sequential addresses makes up the majority of the application code and is referred to as linear code. To improve the performance of linear code execution, a Flash pipeline mode has been implemented. Setting the ENPIPE bit in the FOPT register enables this mode. The Flash pipeline mode is independent of the CPU pipeline. To allow you to maintain code timing compatibility between Flash and ROM devices, the Flash pipeline mode has also been implemented on ROM devices.

An instruction fetch from the Flash or OTP reads out 64 bits per access. The starting address of the access from Flash is automatically aligned to a 64-bit

boundary such that the instruction location is within the 64 bits to be fetched. With Flash pipeline mode enabled (see Figure 1−2), the 64 bits read from the instruction fetch are stored in a 64-bit wide by 2-level deep instruction pre-fetch buffer. The contents of this pre-fetch buffer are then sent to the CPU for processing as required.

Up to two 32-bit instructions or up to four 16-bit instructions can reside within a single 64-bit access. The majority of C28x instructions are 16 bits, so for every 64-bit instruction fetch from the Flash bank it is likely that there are up to four instructions in the pre-fetch buffer ready to process through the CPU. During the time it takes to process these instructions, the Flash pipeline automatically initiates another access to the Flash bank to pre-fetch the next 64 bits. In this manner, the Flash pipeline mode works in the background to keep the instruction pre-fetch buffers as full as possible. Using this technique, the overall efficiency of sequential code execution from Flash or OTP is improved significantly.

Figure 1−2. Flash Pipeline



The Flash pipeline pre-fetch is aborted only on a PC discontinuity caused by executing an instruction such as a branch, BANZ, call, or loop. When this occurs, the pre-fetch is aborted and the contents of the pre-fetch buffer are flushed. There are two possible scenario's when this occurs:

- If the destination address is within the Flash or OTP, the pre-fetch aborts and then resumes at the destination address.

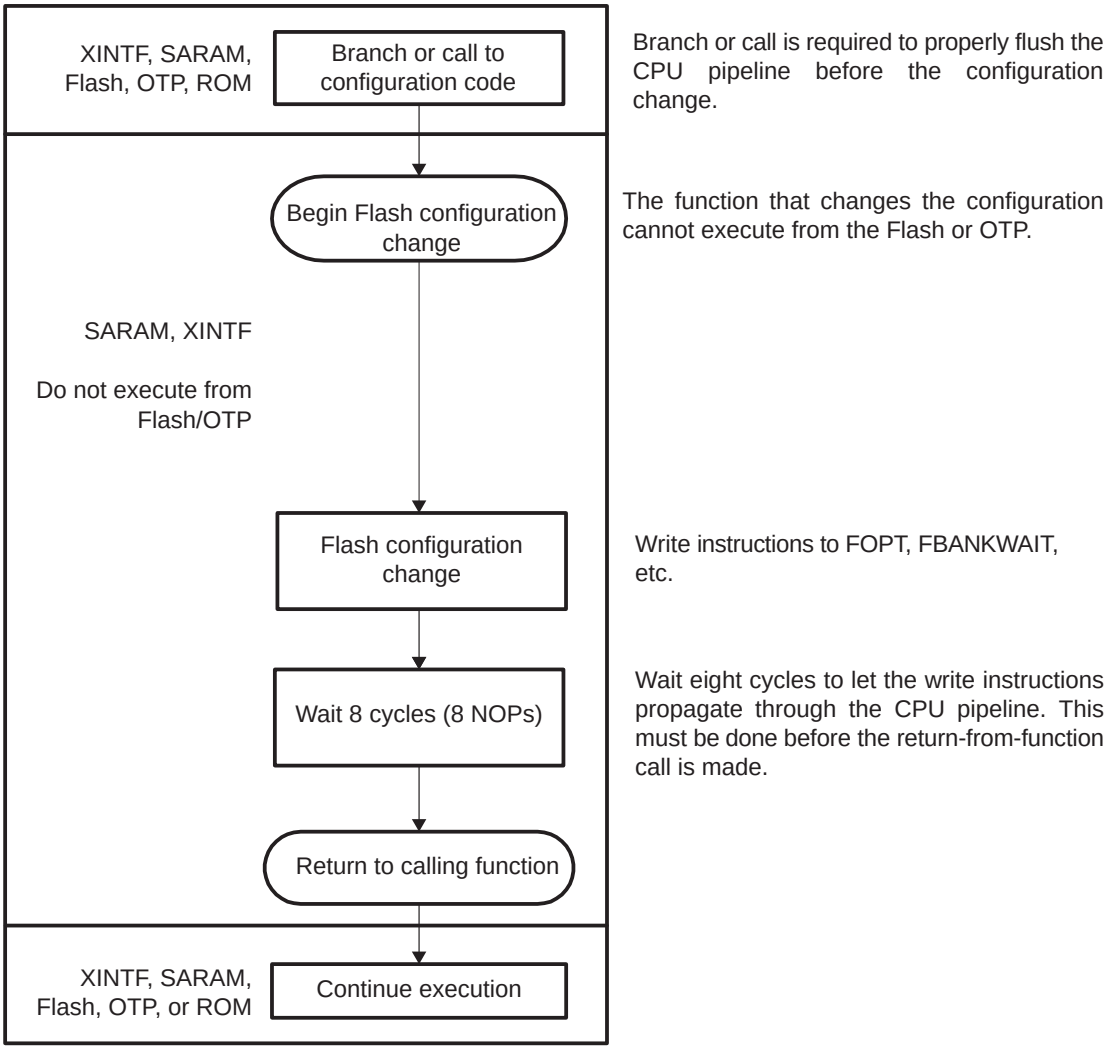
- If the destination address is outside of the Flash and OTP, the pre-fetch is aborted and begins again only when a branch is made back into the Flash or OTP. The Flash pipeline pre-fetch mechanism only applies to instruction fetches from program space. Data reads from data memory and from program memory do not utilize the pre-fetch buffer capability and thus bypass the pre-fetch buffer. For example, instructions such as MAC, DMAC, and PREAD read a data value from program memory. When this read happens, the pre-fetch buffer is bypassed but the buffer is not flushed. If an instruction pre-fetch is already in progress when a data read operation is initiated, then the data read will be stalled until the pre-fetch completes.

1.2.3 Procedure to Change the Flash Configuration Registers

During Flash configuration, no accesses to the Flash or OTP can be in progress. This includes instructions still in the CPU pipeline, data reads, and instruction pre-fetch operations. To be sure that no access takes place during the configuration change, you should follow the procedure shown in Figure 1−3 for any code that modifies the FOPT, FPWR, FBANKWAIT, or FOTPWAIT registers.

This procedure also applies to the ROM on devices where the Flash and OTP have been replaced with ROM.

Figure 1−3. Flash Configuration Access Flow Diagram



1.3 Flash and OTP Registers

The Flash and OTP memory can be configured by the registers shown in Table 1−1. The bit descriptions are in Figure 1−4 through Figure 1−6.

Table 1−1. Flash/OTP Configuration Registers(1)

Name	Address	Size (x16)	Description
		Configuration Registers	
FOPT	0x0000−0A80	1	Flash Option Register
Reserved	0x0000−0A81	1	Reserved
FPWR	0x0000−0A82	1	Flash Power Modes Register
FSTATUS	0x0000−0A83	1	Status Register
FSTDBYWAIT	0x0000−0A84	1	Flash Sleep To Standby Wait Register
FACTIVEWAIT	0x0000−0A85	1	Flash Standby To Active Wait Register
FBANKWAIT	0x0000−0A86	1	Flash Read Access Wait State Register
FOTPWAIT	0x0000−0A87	1	OTP Read Access Wait State Register

1) These registers are EALLOW protected.

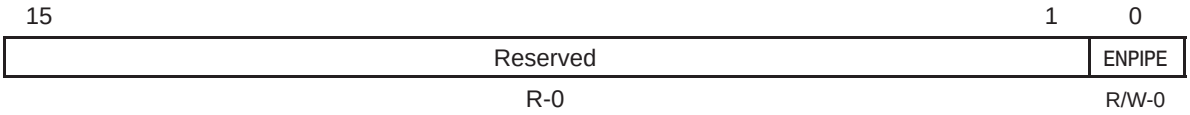
Note: Flash configuration registers should not be accessed while an access is in progress in Flash or OTP memory

The Flash configuration registers should not be accessed from code that is running from OTP or Flash memory or while an access may be in progress. All register accesses to the Flash registers should be made from code executing outside of Flash/OTP memory and an access should not be attempted until all activity on the Flash/OTP has completed. No hardware is included to protect against this.

You can read the Flash registers from code executing in Flash/OTP; however, do not write to the registers.

CPU write access to the registers can be enabled only by executing the EALLOW instruction. Write access is disabled when the EDIS instruction is executed. This protects the registers from spurious accesses. Read access is always available. The registers can be accessed through the JTAG port without the need to execute EALLOW. These registers support both 16-bit and 32-bit accesses.

Figure 1−4. Flash Options (FOPT) Register



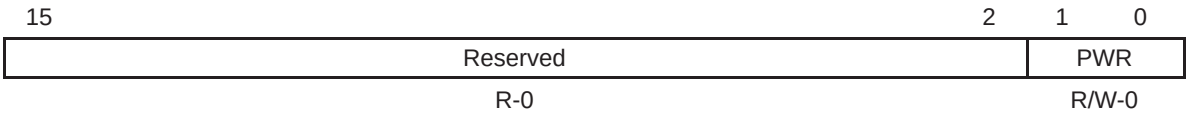
Legend: R = Read access, -0 = value after reset

Note: EALLOW-protected register

Table 1−2. Flash Options (FOPT) Register Field Descriptions

Bits	Field	Description
15−1	Reserved	
0	ENPIPE	Enable Flash Pipeline Mode Bit: Flash pipeline mode is active when this bit is set. The pipeline mode improves performance of instruction fetches by pre-fetching instructions. On Flash devices, ENPIPE affects fetches from Flash and OTP. On ROM devices, this bit affects fetches from the ROM blocks that replaced the Flash and OTP. When pipeline mode is enabled, the Flash waitstates (paged and random) must be greater than zero.

Figure 1−5. Flash Power Register (FPWR)



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

Note: EALLOW-protected register

Table 1−3. Flash Power Register (FPWR) Field Descriptions

Bits	Field	Description
15−2	Reserved	
1−0	PWR	Flash Power Mode Bits: Writing to these bits changes the current power mode of the Flash bank and pump. See section 1.2 for more information on changing the Flash bank power mode. On ROM devices, changing PWR has no effect on the power consumption of the ROM. Moving to standby or sleep mode causes the next access from the ROM to be delayed just as on Flash devices. 00 Pump and bank sleep (lowest power) 01 Pump and bank standby 10 Reserved (no effect) 11 Pump and bank active (highest power)

Figure 1−6. Flash Status Register (FSTATUS)

15	9	8
Reserved		3VSTAT
R-0		R/W1C-0

7	4	3	2	1	0
Reserved		ACTIVE-WAITS	STDBY-WAITS	PWRS	
R-0		R-0	R-0	R-0	

Legend: R = read access, -0 = value after reset, W1C = write 1 to clear

Note: EALLOW-protected register

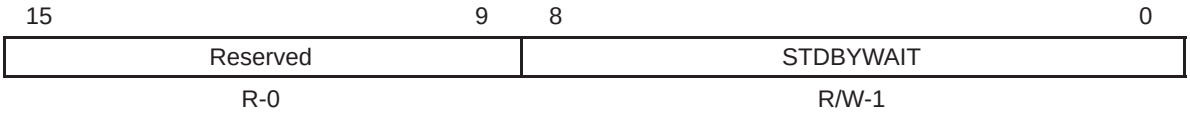
Table 1−4. Flash Status Register (FSTATUS) Field Descriptions

Bits	Field	Description
15−9	Reserved	
8	3VSTAT	V_{DD3V} Status Latch Bit: When set, this bit indicates that the 3 VSTAT signal from the pump module went to a high level. This signal indicates that the 3 V supply went out of allowable range. This bit is cleared by writing a 1, writes of 0 are ignored.
7−4	Reserved	
3	ACTIVEWAITS	Bank and Pump Standby To Active Wait Counter Status Bit: This bit indicates whether the respective wait counter is currently timing out an access. If the bit is set, then the counter is counting. If the bit is 0, then the counter is not counting.

Table 1−4. Flash Status Register (FSTATUS) Field Descriptions (Continued)

Bits	Field	Description
2	STDBYWAITS	Bank and Pump Sleep To Standby Wait Counter Status Bit: This bit indicates whether the respective wait counter is currently timing out an access. If the bit is set, then the counter is counting. If the bit is 0, then the counter is not counting.
1−0	PWRS	Power Modes Status Bits: These bits indicate the current power mode the Flash/OTP is in: 00 Pump and bank sleep (lowest power) 01 Pump and bank standby 10 Reserved 11 Pump and bank active (highest power) **Note:** The above bits only get set to the new power mode once the appropriate timing delays have expired.

Figure 1−7. Flash Standby Wait (FSTDBYWAIT) Register



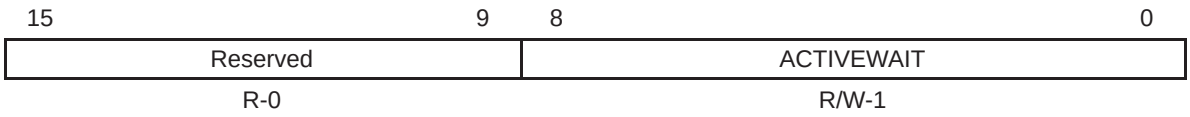
Legend: R = Read access, -0 = value after reset

Note: EALLOW-protected register

Table 1−5. Flash Standby Wait (FSTDBYWAIT) Register Field Descriptions

Bits	Field	Description
15−9	Reserved	
8−0	STDBYWAIT	Bank and Pump Sleep To Standby Wait Count. When the bank and pump modules are in sleep mode and a write is performed to the PWR bits in the FPWR register (to change to a higher default power mode) or a CPU read or fetch access is performed to the Flash bank or OTP, then a counter is initiated with the value specified in these register bits. The power mode for the bank and pump are set for standby mode. The counter then counts down to zero before the PWRS bits are set to standby mode. If a CPU read or fetch access to the Flash bank/OTP initiated the process, the CPU is stalled until the access completes (see ACTIVEWAIT bits). The STDBYWAIT bits specify the number of CPU clock cycles (0..511 SYSCLKOUT cycles) of delay. See the Flash and OTP Power Mode section for more details. This register should be left in its default state.

Figure 1−8. Flash Standby to Active Wait Counter (FACTIVEWAIT) Register



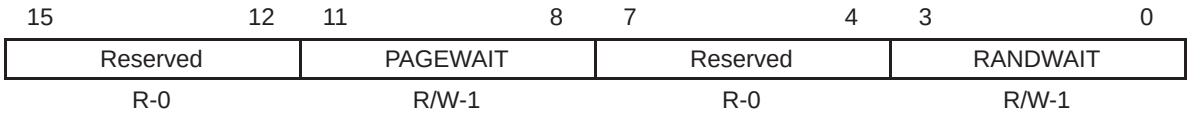
Legend: R = Read access, -0 = value after reset

Note: EALLOW-protected register

Table 1−6. Flash Standby to Active Wait Counter (FACTIVEWAIT) Register Field Descriptions

Bits	Field	Description
15−9	Reserved	
8−0	ACTIVEWAIT	Bank and Pump Standby To Active Wait Count: When the bank and pump modules are in standby mode and a write is performed to the PWR bits in the FPWR register (to change to a higher default power mode) or a CPU read or fetch access is performed to the Flash bank, then a counter is initiated with the value specified in these register bits. The power mode for the bank and pump are set for active mode. The counter then counts down to zero before allowing any CPU access to proceed. If a CPU read or fetch access to the Flash bank initiated the process, the CPU is stalled until the access completes (see PAGEWAIT and RANDWAIT bits). The ACTIVEWAIT bits specify the number of CPU clock cycles (0..511 SYSCLKOUT cycles) of delay. Refer to the Flash and OTP Power Modes section for more details. This register should be left in its default state.

Figure 1−9. Flash Waitstate (FBANKWAIT) Register



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

Note: EALLOW-protected register

Table 1−7. Flash Waitstate (FBANKWAIT) Register Field Descriptions

Bits	Field	Description
15−12	Reserved	
11−8	PAGEWAIT	Flash Paged Read Wait States: These register bits specify the number of wait states for a paged read operation in CPU clock cycles (0..15 SYSCLKOUT cycles) to the banks. See Notes 1 and 2. On ROM devices, these bits affect the wait states of the ROM block that replaced Flash.

Table 1−7. Flash Waitstate (FBANKWAIT) Register Field Descriptions (Continued)

Bits	Field	Description
7−4	Reserved	
3−0	RANDWAIT	Flash Random Read Wait States: These register bits specify the number of wait states for a random read operation in CPU clock cycles (0..15 SYSCLKOUT cycles) to the banks. See Notes 1 and 2.RANDWAIT must be set greater than 0. That is, at least 1 random waitstate must be used. See the device-specific data sheet for the minimum time required for Flash or ROM access. On ROM devices, these bits affect the wait states of the ROM block that replaced Flash. **Notes:** 1) You must set RANDWAIT to a value greater than or equal to the PAGEWAIT setting. No hardware is provided to detect a PAGEWAIT value that is greater then RANDWAIT. 2) When enabling Flash pipeline mode, you must set PAGEWAIT and RANDWAIT to a value greater than zero.

Figure 1−10. OTP Waitstate (FOTPWAIT) Register

15	5	4	0
Reserved		OTPWAIT	
R-0		R/W-1	

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

Note: EALLOW-protected register

Table 1−8. OTP Waitstate (FOTPWAIT) Register

Bits	Field	Description
15−5	Reserved	
4−0	OTPWAIT	OTP Read Wait States. These register bits specify the number of wait states for a read operation in CPU clock cycles (0..31 SYSCLKOUT cycles) to the OTP. See CPU Read Or Fetch Access From Flash/OTP section for more details. There is no PAGE mode in the OTP. OTPWAIT must be set greater than 0. That is, a minimum of 1 wait-state must be used. See the device-specific datasheet for the minimum time required for an OTP or ROM access. On ROM devices, these bits affect the wait states of the ROM block that replaced Flash.

Chapter 2

Code Security Module (CSM)

The code security module (CSM) is a security feature incorporated in 28x devices. It prevents access/visibility to on-chip memory to unauthorized persons—i.e., it prevents duplication/reverse engineering of proprietary code.

The word secure means access to on-chip memory is protected. The word unsecure means access to on-chip secure memory is not protected — i.e., the contents of the memory could be read by any means (through a debugging tool such as Code Composer Studio™, for example).

Topic **Page**

2.1 Functional Description

The security module restricts the CPU access to certain on-chip memory. This, in effect, blocks read and write access to various memories through the JTAG port or external peripherals. Security is defined with respect to the access of on-chip memory and prevents unauthorized copying of proprietary code or data.

The device is secure when CPU access to the on-chip secure memory locations is restricted. When secure, two levels of protection are possible, depending on where the program counter is currently pointing. If code is currently running from inside secure memory, only access through JTAG is blocked (i.e., the emulator). This allows secure code to access secure data. Conversely, if code is running from nonsecure memory, all accesses to secure memories are blocked. User code can dynamically jump in and out of secure memory, thereby allowing secure function calls from nonsecure memory. Similarly, interrupt service routines can be placed in secure memory, even if the main program loop is run from nonsecure memory.

Security is protected by a password of 128-bit data (eight 16-bit words) that is used to secure or unsecure the device.

The device is unsecured by executing the password match flow (PMF), described later in this chapter. Table 2−1 shows the levels of security.

Table 2−1. Security Levels

PMF Executed With Correct Password?	Operating Mode	Program Fetch Location	Security Description
No	Secure	Outside secure memory	Only fetches are allowed to secure memory
No	Secure	Inside secure memory	CPU has full access. JTAG port cannot read the secured memory contents.
Yes	Not Secure	Anywhere	Full access for CPU and JTAG port to secure memory

Passwords are stored in code security password locations (PWL) in flash/ROM memory (0x003F 7FF8−0x003F 7FFF). These locations store the password predetermined by the system designer.

In flash devices, the password can be changed anytime if the old password is known. In ROM devices, the password cannot be changed after the device is manufactured by Texas Instruments (TI).

If the PWL have all 128 bits as ones, the device is labeled unsecure. Since new flash devices have erased flash (all ones), only a read of the PWL is required to bring the device into unsecure mode. If the PWL have all 128 bits as zeros, the device is secure, regardless of the contents of the KEY registers. Do not use all zeros as a password or reset the device after performing a clear routine on the flash. If a device is reset when the PWL is all zeros, the device cannot be debugged or reprogrammed. To summarize, a device with an erased flash array is unsecure. A device with a cleared flash array is secure.

User accessible registers (eight 16-bit words) that are used to secure or unsecure the device are referred to as key registers. These registers are mapped in the memory space at addresses 0x0000 0AE0 − 0x0000 0AE7 and are EALLOW protected.

Note: Security of addresses between 0x3F7F80 and 0x3F7FF5

For code security operation, all addresses between 0x3F7F80 and 0x3F7FF5 cannot be used as program code or data, but must be programmed to 0x0000 when the Code Security Passwords are programmed. If security is not a concern, then these addresses can be used for code or data.

Code Security Module Disclaimer

The Code Security Module (“CSM”) included on this device was designed to password protect the data stored in the associated memory (either ROM or Flash) and is warranted by Texas Instruments (TI), in accordance with its standard terms and conditions, to conform to TI’s published specifications for the warranty period applicable for this device.

TI DOES NOT, HOWEVER, WARRANT OR REPRESENT THAT THE CSM CANNOT BE COMPROMISED OR BREACHED OR THAT THE DATA STORED IN THE ASSOCIATED MEMORY CANNOT BE ACCESSED THROUGH OTHER MEANS. MOREOVER, EXCEPT AS SET FORTH ABOVE, TI MAKES NO WARRANTIES OR REPRESENTATIONS CONCERNING THE CSM OR OPERATION OF THIS DEVICE, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL TI BE LIABLE FOR ANY CONSEQUENTIAL, SPECIAL, INDIRECT, INCIDENTAL, OR PUNITIVE DAMAGES, HOWEVER CAUSED, ARISING IN ANY WAY OUT OF YOUR USE OF THE CSM OR THIS DEVICE, WHETHER OR NOT TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. EXCLUDED DAMAGES INCLUDE, BUT ARE NOT LIMITED TO LOSS OF DATA, LOSS OF GOODWILL, LOSS OF USE OR INTERRUPTION OF BUSINESS OR OTHER ECONOMIC LOSS.

2.2 CSM Impact on Other On-Chip Resources

The CSM has no impact whatsoever on the following on-chip resources:

- Single-access RAM (SARAM) blocks not designated as secure − These memory blocks can be freely accessed and code run from them, whether the device is in secure or unsecure mode.
- Boot ROM contents − Visibility to the boot ROM contents is not impacted by the CSM.
- On-chip peripheral registers − The peripheral registers can be initialized by code running from on-chip or off-chip memory, whether the device is in secure or unsecure mode.
- PIE Vector Table − Vector tables can be read and written regardless of whether the device is in secure or unsecure mode. Table 2−2 and Table 2−3 show which on-chip resources are affected (or are not affected) by the CSM on the 281x devices. For other devices, see the device-specific data sheet.

Table 2−2. F281x/C281x Resources Affected by the CSM

Address	Block
0x0000 8000−0x0000 8FFF	L0 SARAM (4K X 16)(1)
0x0000 9000−0x0000 9FFF	L1 SARAM (4K X 16)(1)
0x003D 7800− 0x003D 7BFF	One-time programmable (OTP) or ROM (1K X 16)
0x003D 8000−0x003F 7FFF	Flash or ROM (128K X 16 or 64K X 16)

1) On R281x RAM devices, L0 and L1 SARAM are affected by the CSM. The CSM passwords, however, are set to all 1s so the CSM can easily be unlocked.

Table 2−3. 281x Resources Not Affected by the CSM

Address	Block
0x0000 0000−0x0000 03FF	M0 SARAM (1K X 16)
0x0000 0400−0x0000 07FF	M1 SARAM (1K X16)
0x0000 0800−0x0000 0CFF	Peripheral Frame 0 (2K X 16)
0x0000 0D00−0x0000 0FFF	PIE Vector RAM (256 X 16)
0x0000 2000−0x0000 3FFF	XINTF Zone 0
0x0000 4000−0x0000 5FFF	XINTF Zone 1
0x0000 6000−0x0000 6FFF	Peripheral Frame 1 (4K X 16)
0x0000 7000−0x0000 7FFF	Peripheral Frame 2 (4K X 16)
0x0008 0000−0x000F FFFF	XINTF Zone 2
0x0010 0000−0x0017 FFFF	XINTF Zone 6
0x003F 8000−0x003F 9FFF	H0 SARAM (8K X 16)
0x003F 0000−0x003F FFFF	XINTF Zone 7
0x003F F000−0x003F FFFF	Boot ROM (4K X 16)

To summarize, it is possible to load code onto the unprotected on-chip program RAM shown in Table 2−3 via the JTAG connector without any impact from the CSM. The code can be debugged and the peripheral registers initialized, independent of whether the device is in secure or unsecure mode.

2.3 Incorporating Code Security in User Applications

Code security is typically not required in the development phase of a project; however, security is needed once a robust code is developed. Before such a code is programmed in the flash memory (or committed to ROM), a password should be chosen to secure the device. Once a password is in place, the device is secured (i.e., programming a password at the appropriate locations and either performing a device reset or setting the FORCESEC bit (CSMSCR.15) is the action that secures the device). From that time on, access to debug the contents of secure memory by any means (via JTAG, code running off external/on-chip memory etc.) requires the supply of a valid password. A password is not needed to run the code out of secure memory (such as in a typical end-customer usage); however, access to secure memory contents for debug purpose requires a password.

Table 2−4. Code Security Module (CSM) Registers

Memory Address	Register Name	Reset Values	Register Description
KEY Registers − Accessible by the user			
0x0000−0AE0	KEY0†	0xFFFF	Low word of the 128-bit KEY register
0x0000−0AE1	KEY1†	0xFFFF	Second word of the 128-bit KEY register
0x0000−0AE2	KEY2†	0xFFFF	Third word of the 128-bit KEY register
0x0000−0AE3	KEY3†	0xFFFF	Fourth word of the 128-bit key
0x0000−0AE4	KEY4†	0xFFFF	Fifth word of the 128-bit key
0x0000−0AE5	KEY5†	0xFFFF	Sixth word of the 128-bit key
0x0000−0AE6	KEY6†	0xFFFF	Seventh word of the 128-bit key
0x0000−0AE7	KEY7†	0xFFFF	High word of the 128-bit KEY register
0x0000−0AEF	CSMSCR†		CSM status and control register
PWL in Memory − Reserved for passwords only (On R281x devices, all passwords are set to all Fs)			
0x003F−7FF8	PWL0	User defined	Low word of the 128-bit password
0x003F−7FF9	PWL1	User defined	Second word of the 128-bit password
0x003F−7FFA	PWL2	User defined	Third word of the 128-bit password
0x003F−7FFB	PWL3	User defined	Fourth word of the 128-bit password
0x003F−7FFC	PWL4	User defined	Fifth word of the 128-bit password
0x003F−7FFD	PWL5	User defined	Sixth word of the 128-bit password
0x003F−7FFE	PWL6	User defined	Seventh word of the 128-bit password
0x003F−7FFF	PWL7	User defined	High word of the 128-bit password

† EALLOW protected

Figure 2−1. CSM Status and Control (CSMSCR) Register

15	14 7	6 1	0
FORCE-SEC	Reserved	Reserved	SECURE
W-1	R-0	R-10111	R-1

Legend: R = Read access, W = write access, -0 = value after reset

Note: EALLOW-protected register

Table 2−5. CSM Status and Control (CSMSCR) Register Field Descriptions

Bits	Field	Description
15	FORCESEC	Writing a 1 clears the KEY registers and secures the device. A read always returns a zero.
0	SECURE	Read-only bit that reflects the security state of the device. 1 Device is secure (CSM locked) 0 Device is unsecure (CSM unlocked)

2.3.1 Environments That Require Security Unlocking

Following are the typical situations under which unsecuring can be required:

- Code development using debuggers (such as Code Composer Studio™)

  This is the most common environment during the design phase of a product.

- Flash programming using TI's flash utilities

  Flash programming is common during code development and testing. Once the user supplies the necessary password, the flash utilities disable the security logic before attempting to program the flash. The flash utilities can disable the code security logic in new devices without any authorization, since new devices come with an erased flash. However, reprogramming devices (that already contain custom passwords) require passwords to be supplied to the flash utilities in order to enable programming.

- Custom environment defined by the application

  In addition to the above, access to secure memory contents can be required in situations such as:

  - Using the on-chip bootloader to program the flash
  - Executing code from external memory or on-chip unsecure memory and requiring access to secure memory for lookup table. This is not a

suggested operating condition as supplying the password from external code could compromise code security.

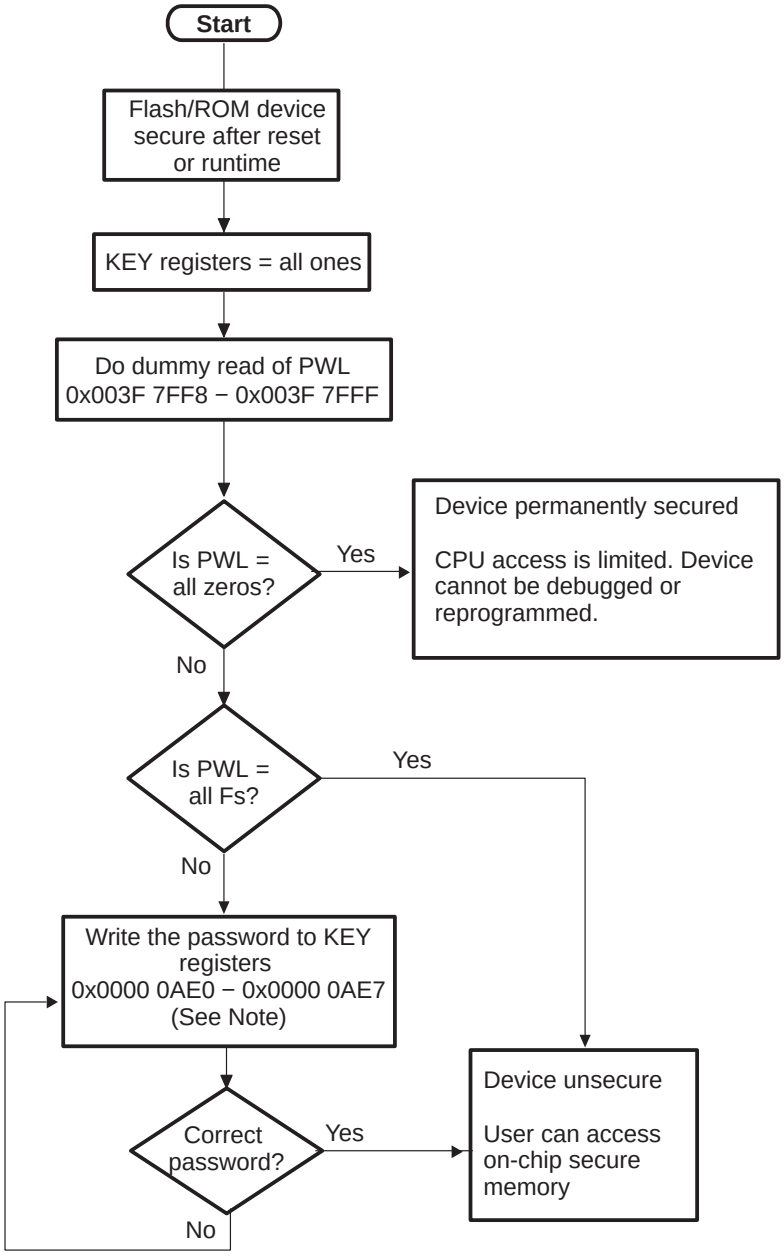
The unsecuring sequence is identical in all the above situations. This sequence is referred to as the *password match flow (PMF)* for simplicity. Figure 2−2 explains the sequence of operation that is required every time the user attempts to unsecure a device. A code example is listed for clarity.

2.3.2 Password Match Flow

Password match flow (PMF) is essentially a sequence of eight dummy reads from password locations (PWL) followed by eight writes to KEY registers.

Figure 2−2 shows how the PMF helps to initialize the security logic registers and disable security logic.

Figure 2−2. Password Match Flow (PMF)



Note: The KEY registers are EALLOW protected.

2.3.3 Unsecuring Considerations for Devices With/Without Code Security

Case 1 and Case 2 provide unsecuring considerations for devices with and without code security.

Case 1: Device With Code Security

A device with code security should have a predetermined password stored in the PWL (locations 0x003F 7FF8−0x003F 7FFF in memory). In addition, locations 0x3F7F80 − 0x3F7FF5 should be programmed with all 0x0000 and not used for program and/or data storage. The following are steps to unsecure this device:

1) Perform a dummy read of the PWL.

2) Write the password into the KEY registers (locations 0x0000 0AE0−0x0000 0AE7 in memory).

3) If the password is correct, the device becomes unsecure; otherwise, it stays secure.

Case 2: Device Without Code Security

A device without code security should have 0x FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF (128 bits of all ones) stored in the PWL. The following are steps to use this device:

4) Perform a dummy read of the PWL.

5) Secure memory is fully accessible immediately after this operation is completed.

Note: A dummy read operation must be performed prior to reading, writing, or programming secure memory, even though the device is not protected with a password.

2.3.4 C Code Example to Unsecure

```
volatile int *CSM = (volatile int *)0x000AE0;    //CSM register file
volatile int *PWL = (volatile int *)0x3F7FF8;    //Password location
volatile int tmp;
int i ;
// Read the 128-bits of the password location (PWL)
// in Flash/ROM at address 0x3F7FF8-0x3F7FFF.
// If the device is secure, then the values read will
// not actually be loaded into the temp variable, so
// this is called a dummy read.
for (i = 0; i<8; i++) tmp = *PWL++;

// If the password location (PWL) is all = ones (0xFFFF),
// then the device will now be unsecure.  If the password
// is not all ones (0xFFFF), then the code below is required
// to unsecure the CSM.
// Write the 128-bit password to the KEY registers
// If this password matches that stored in the
// PWL then the CSM will become unsecure.  If it does not
// match, then the device will remain secure.
// An example password of:
// 0x0123456789ABCDEFCDEF89AB45670123 is used.
asm("  EALLOW");    // Key registers are EALLOW protected
*CSM++ = 0x0123;    // Register KEY0 at 0xAE0
*CSM++ = 0x4567;    // Register KEY1 at 0xAE1
*CSM++ = 0x89AB;    // Register KEY2 at 0xAE2
*CSM++ = 0xCDEF;    // Register KEY3 at 0xAE3
*CSM++ = 0xCDEF;    // Register KEY4 at 0xAE4
*CSM++ = 0x89AB;    // Register KEY5 at 0xAE5
*CSM++ = 0x4567;    // Register KEY6 at 0xAE6
*CSM++ = 0x0123;    // Register KEY7 at 0xAE7
asm("  EDIS");
```

2.3.5 C Code Example to Resecure

```
volatile int *CSM = 0x000AE0;       //CSM register file
//Set FORCESEC bit
asm("  EALLOW");                     //CSMSCR register is EALLOW protected.
*CSM = 0x8000;
asm ("EDIS");
```

2.4 DOs and DON'Ts to Protect Security Logic

2.4.1 DOs

- To keep the debug and code development phase simple, use the device in the unsecure mode; i.e., use 128 bits of all ones as PWL words (or use a password that is easy to remember). Use passwords after the development phase when the code is frozen.
- Recheck the passwords in PWL before programming the COFF file using flash utilities.
- The flow of code execution can freely toggle back and forth between secure memory and unsecure memory without compromising security. To access data variables located in secure memory when the device is secured, code execution must currently be running from secure memory.
- Program locations 0x3F7F80 − 0x3F7FF5 with 0x0000 when using the CSM.

2.4.2 DON'Ts

- If code security is desired, do not embed the password in your application anywhere other than in the PWL or security can be compromised.
- Do not use 128 bits of all zeros as the password. This automatically secures the device, regardless of the contents of the KEY register. The device is not debuggable nor reprogrammable.
- Do not pull a reset after clearing the flash array but before erasing the array. This leaves zeros in the PWL that automatically secures the device, regardless of the contents of the KEY register. The device is not debuggable nor reprogrammable.
- Do not use locations 0x3F7F80 − 0x3F7FF5 to store program and/or data. These locations should be programmed to 0x0000 when using the CSM.

2.5 CSM Features − Summary

1) The flash is secured after a reset until the password match flow (PMF) is executed.

2) The standard way of running code out of the flash or ROM is to program the flash with the code (for ROM devices the program is hardcoded at device fabrication) and powering up the DSP in microcomputer mode. Since instruction fetches are always allowed from secure memory, regardless of the state of the CSM, the code functions correctly even without executing the PMF.

3) Secure memory cannot be modified while the device is secured.

4) Secure memory cannot be read from any code running from unsecure memory while the device is secured.

5) Secure memory cannot be read or written to by the debugger (i.e., Code Composer Studio™) at any time that the device is secured.

6) Complete access to secure memory from both the CPU code and the debugger is granted while the device is unsecured.

Chapter 3

Clocking

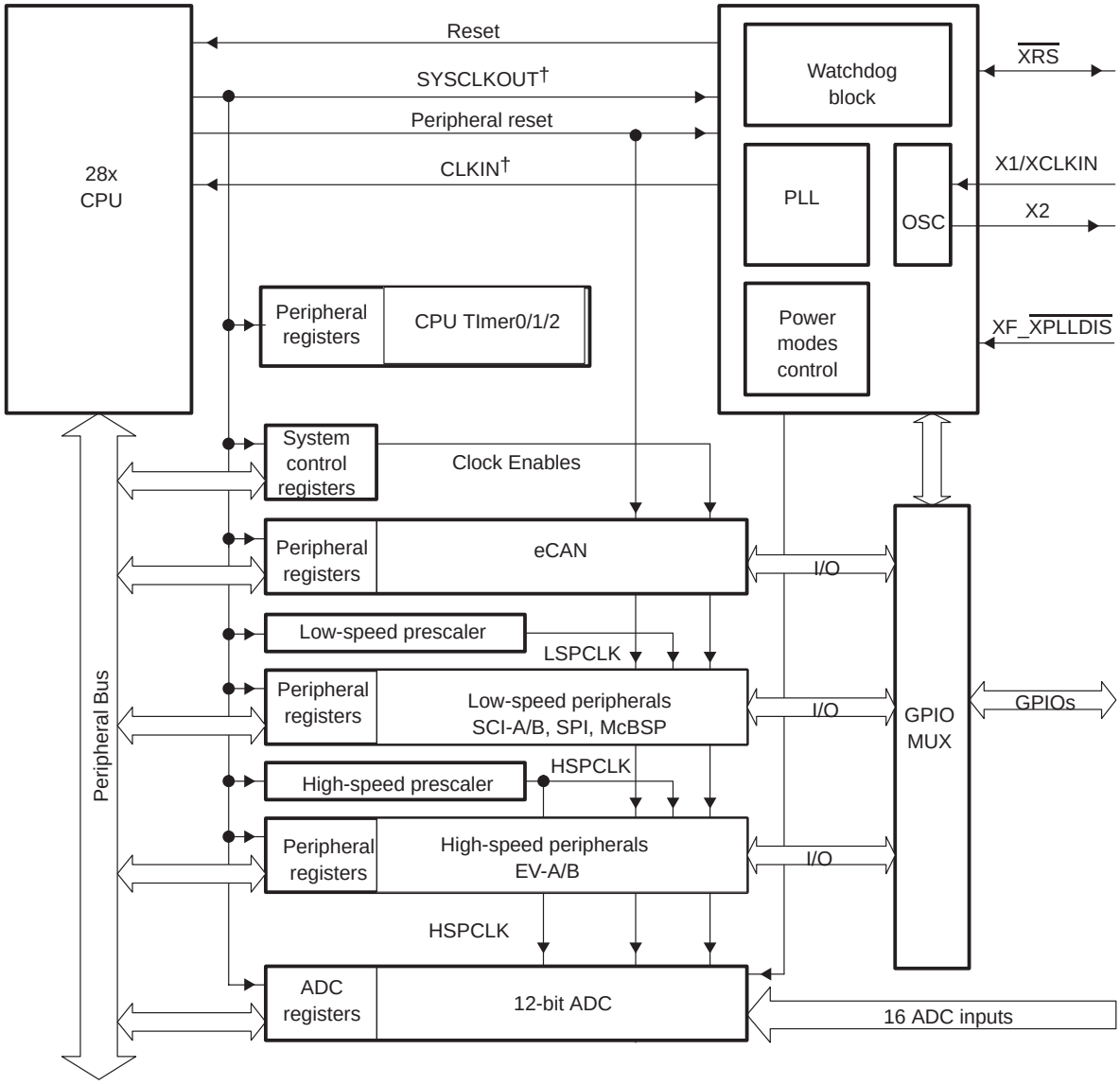
This section describes the 281x oscillator, PLL and clocking mechanisms, the watchdog function, and the low-power modes.

Topic **Page**

3.1 Clocking and System Control

Figure 3−1 shows the various clock and reset domains in the 281x devices.

Figure 3−1. Clock and Reset Domains



† CLKIN is the clock into the CPU. It is passed out of the CPU as SYSCLKOUT (that is, CLKIN is the same frequency as SYSCLKOUT).

The PLL, clocking, watchdog and low-power modes, are controlled by the registers listed in Table 3−1.

Table 3−1. PLL, Clocking, Watchdog, and Low-Power Mode Registers†

Name	Address	Size (x16)	Description
Reserved	0x0000 7010 0x0000 7019	10	
HISPCP	0x0000 701A	1	High-Speed Peripheral Clock Prescaler Register for HSPCLK Clock
LOSPCP	0x0000 701B	1	Low-Speed Peripheral Clock Prescaler Register for HSPCLK clock
PCLKCR	0x0000 701C	1	Peripheral Clock Control Register
Reserved	0x0000 701D	1	
LPMCR0	0x0000 701E	1	Low Power Mode Control Register 0
LPMCR1	0x0000 701F	1	Low Power Mode Control Register 1
Reserved	0x0000 7020	1	
PLLCR	0x0000 7021	1	PLL Control Register‡
SCSR	0x0000 7022	1	System Control & Status Register
WDCNTR	0x0000 7023	1	Watchdog Counter Register
Reserved	0x0000 7024	1	
WDKEY	0x0000 7025	1	Watchdog Reset Key Register
Reserved	0x0000 7026 0x0000 7028	3	
WDCR	0x0000 7029	1	Watchdog Control Register
Reserved	0x0000 702A 0x0000 702F	6	

† All of the registers in this table can be accessed only by executing the EALLOW instruction.
‡ The PLL control register (PLLCR) is reset to a known state by the $\overline{\text{XRS}}$ signal only.

The PCLKCR register enables/disables clocks to the various peripheral modules in the 281x devices. Figure 3−2 lists the bit descriptions of the PCLKCR register.

Figure 3−2. Peripheral Clock Control (PCLKCR) Register

15	14	13	12	11	10	9	8
Reserved	ECANENCLK	Reserved	MCBSPENCLK	SCIBENCLK	SCIAENCLK	Reserved	SPIENCLK
R-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R-0	R/W-0

7	4	3	2	1	0
Reserved		ADCENCLK	Reserved	EVBENCLK	EVAENCLK
R-0		R/W-0	R-0	R/W-0	R/W-0

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

Notes: 1) EALLOW-protected register

2) If a peripheral block is not used, then the clock to that peripheral can be turned off to minimize power consumption.

Table 3−2. Peripheral Clock Control (PCLKCR) Register Field Descriptions

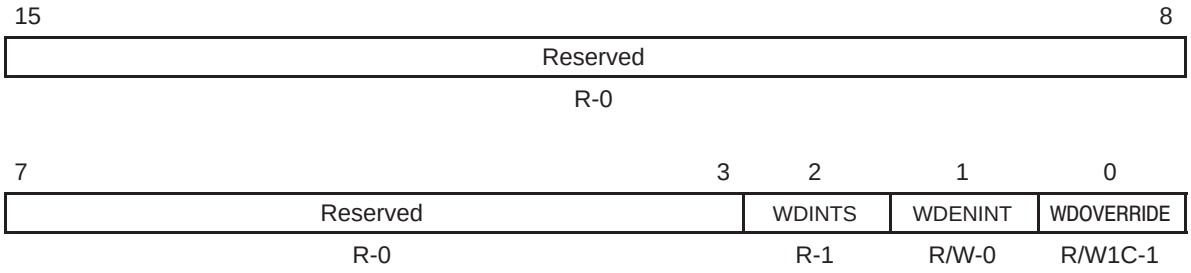
Bits	Field	Description
15	Reserved	Reserved
14	ECANENCLK	If ECANENCLK is set, it enables the system clock within the CAN peripheral. For low power operation, this bit is set to zero by the user or by reset.
13	Reserved	Reserved
12	MCBSPENCLK	If MCBSPENCLK is set, it enables the low-speed clock (LSPCLK) within the McBSP peripheral. For low power operation, this bit is set to zero by the user or by reset.
11	SCIBENCLK	If SCIBENCLK is set, it enables the low-speed clock (LSPCLK) within the SCI-B peripheral. For low power operation, this bit is set to zero by the user or by reset.
10	SCIAENCLK	If SCIAENCLK is set, it enables the low-speed clock (LSPCLK) within the SCI-A peripheral. For low power operation, this bit is set to zero by the user or by reset.
9	Reserved	Reserved
8	SPIAENCLK	If SPIAENCLK is set, it enables the low-speed clock (LSPCLK) within the SPI peripheral. For low power operation, this bit is set to zero by the user or by reset.
7−4	Reserved	
3	ADCENCLK	If ADCENCLK is set, it enables the high-speed clock (HSPCLK) within the ADC peripheral. For low power operation, this bit is set to zero by the user or by reset.
2	Reserved	Reserved

Table 3−2. Peripheral Clock Control (PCLKCR) Register Field Descriptions (Continued)

Bits	Field	Description
1	EVBENCLK	If EVBENCLK is set, it enables the high-speed clock (HSPCLK) within the EV-B peripheral. For low power operation, this bit is set to zero by the user or by reset.
0	EVAENCLK	If EVAENCLK is set, it enables the high-speed clock (HSPCLK) within the EV-A peripheral. For low power operation, this bit is set to zero by the user or by reset.

The system control and status register contains the watchdog override bit and the watchdog interrupt enable/disable bit. Figure 3−3 describes the bit functions of the SCSR register.

Figure 3−3. System Control and Status (SCSR) Register



Legend: R = Read access, -0 = value after reset, W1C = Write 1 to clear

Note: EALLOW-protected register

Table 3−3. System Control and Status (SCSR) Register Field Descriptions

Bits	Field	Description
15−3	Reserved	
2	WDINTS	Watchdog interrupt status bit. WDINTS reflects the current state of the $\overline{\text{WDINT}}$ signal from the watchdog block. If the watchdog interrupt is used to wake the device from IDLE or STANDBY, then you should check this bit to make sure the watchdog interrupt signal is no longer active (WDINTS = 1) before attempting to go back into IDLE or STANDBY mode.

Table 3−3. System Control and Status (SCSR) Register Field Descriptions (Continued)

Bits	Field	Description
1	WDENINT	Watchdog enable interrupt. 0 The $\overline{\text{WDRST}}$ output signal is enabled and the $\overline{\text{WDINT}}$ output signal is disabled. This is the default state on reset ($\overline{\text{XRS}}$). 1 The watchdog reset ($\overline{\text{WDRST}}$) output signal is disabled and the watchdog interrupt ($\overline{\text{WDINT}}$) output signal is enabled.
0	WDOVERRIDE	If WDOVERRIDE is set to 1, the user is allowed to change the state of the Watchdog disable (WDDIS) bit in the Watchdog Control (WDCR) register. If the WDOVERRIDE bit is cleared by writing a 1, the WDDIS bit cannot be modified by the user. This also enables the watchdog if it is currently disabled. Writing a 0 has no effect. If this bit is cleared, then it remains in this state until a reset occurs. The current state of this bit is readable by the user.

The HISPCP and LOSPCP registers are used to configure the high- and low-speed peripheral clocks, respectively. See Figure 3−4 for the HISPCP bit definitions and Figure 3−5 for the LOSPCP bit definitions.

Figure 3−4. High-Speed Peripheral Clock Prescaler (HISPCP) Register

15	3	2	0
Reserved		HSPCLK	
R-0		R/W-001	

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

Note: EALLOW-protected register

Table 3−4. High-Speed Peripheral Clock Prescaler (HISPCP) Register Field Descriptions

Bits	Field	Value	Description
15−3	Reserved		
2−0	HSPCLK		These bits configure the high-speed peripheral clock (HSPCLK) rate relative to SYSCLKOUT: If HISPCP ≠ 0, HSPCLK = SYSCLKOUT/(HISPCP† X 2) If HISPCP = 0, HSPCLK = SYSCLKOUT
		000	High speed clock = SYSCLKOUT/1
		001	High speed clock = SYSCLKOUT/2 (reset default)
		010	High speed clock = SYSCLKOUT/4
		011	High speed clock = SYSCLKOUT/6
		100	High speed clock = SYSCLKOUT/8
		101	High speed clock = SYSCLKOUT/10
		110	High speed clock = SYSCLKOUT/12
		111	High speed clock = SYSCLKOUT/14

† HISPCP in this equation denotes the value of bits 2:0 in the HISPCP register.

Figure 3−5. Low-Speed Peripheral Clock Prescaler (LOSPCP) Register

15	3	2	0
Reserved		LSPCLK	
R-0		R/W-010	

Legend: R = Read access, W = write access, -0 = value after reset

Note: EALLOW-protected register

Table 3−5. Low-Speed Peripheral Clock Prescaler (LOSPCP) Register Field Descriptions

Bits	Field	Description
15−3	Reserved	
2−0	LSPCLK	These bits configure the low-speed peripheral clock (LSPCLK) rate relative to SYSCLKOUT: If LOSPCP ≠ 0, LSPCLK = SYSCLKOUT/(LOSPCP† X 2) If LOSPCP = 0, LSPCLK = SYSCLKOUT
		000 Low speed clock = SYSCLKOUT/1
		001 Low speed clock= SYSCLKOUT/2
		010 Low speed clock= SYSCLKOUT/4 (reset default)

Table 3−5. Low-Speed Peripheral Clock Prescaler (LOSPCP) Register Field Descriptions (Continued)

Bits	Field	Description	
		011	Low speed clock= SYSCLKOUT/6
		100	Low speed clock= SYSCLKOUT/8
		101	Low speed clock= SYSCLKOUT/10
		110	Low speed clock= SYSCLKOUT/12
		111	Low speed clock= SYSCLKOUT/14

† LOSPCP in this equation denotes the value of bits 2:0 in the LOSPCP register.

3.2 OSC and PLL Block

The on-chip oscillator and phase-locked loop (PLL) block provides the clocking signals for the device, as well as control for low-power mode entry.

3.2.1 PLL-Based Clock Module

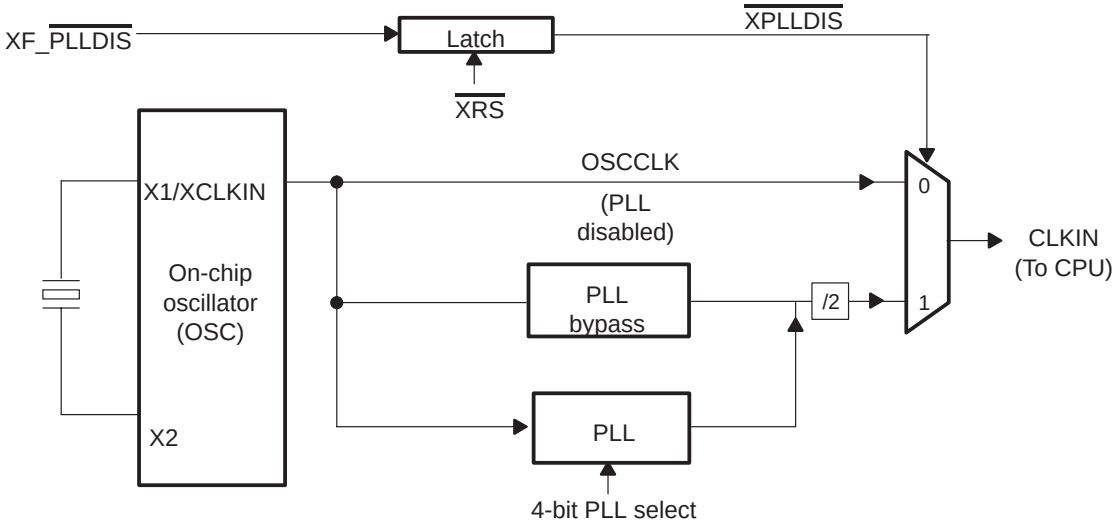
The F281x devices have an on-chip, PLL-based clock module. The PLL has a 4-bit ratio control to select different CPU clock rates.

The PLL-based clock module provides two modes of operation:

- Crystal-operation
  This mode allows the use of an external crystal/resonator to provide the time base to the device.
- External clock source operation
  This mode allows the internal oscillator to be bypassed. The device clocks are generated from an external clock source input on the XTAL1/CLKIN pin. In this case, an external oscillator clock is connected to the XTAL1/CLKIN pin.

Figure 3−6 shows the OSC and PLL block on the 281x.

Figure 3−6. OSC and PLL Block



The OSC circuit enables a crystal to be attached to the 281x devices using the X1/XCLKIN and X2 pins. If a crystal is not used, then an external oscillator can be directly connected to the X1/XCLKIN pin and the X2 pin is left unconnected.

See the *TMS320F2810, TMS320F2811, TMS320F2812, TMS320C2810, TMS320C2811, TMS320C2812 Digital Signal Processors Data Manual* (literature number SPRS174) and the *TMS320R2811 and TMS320R2812 Digital Signal Processors* (literature number SPRS257).

Table 3−6. Possible PLL Configuration Modes

PLL Mode	Remarks	SYSCLKOUT
PLL Disabled	Invoked by tying $\overline{\text{XPLLDIS}}$ pin low upon reset. PLL block is completely disabled. Clock input to the CPU (CLKIN) is directly derived from the clock signal present at the X1/XCLKIN pin.	XCLKIN
PLL Bypassed	Default PLL configuration upon power-up, if PLL is not disabled. The PLL itself is bypassed. However, the /2 module in the PLL block divides the clock input at the X1/XCLKIN pin by two before feeding it to the CPU.	XCLKIN/2
PLL Enabled	Achieved by writing a non-zero value “n” into PLLCR register. The /2 module in the PLL block now divides the output of the PLL by two before feeding it to the CPU.	(XCLKIN * n) / 2

The PLLCR register DIV field (bits 3−0) is used to change the PLL multiplier of the device. When the CPU writes to the DIV bits, the PLL logic switches the CPU clock (CLKIN) to OSCCLK/2. Once the PLL is stable and has locked at the new specified frequency, the PLL switches CLKIN to the new frequency as shown in Figure 3−7. The time it takes for the PLL to switch from OSCCLK/2 to the new frequency is 131072 OSCCLK cycles. For time-critical software, you should insert a software delay of the required locking period after writing to the PLLCR register in order for the PLL to complete the locking sequence.

Figure 3−7. PLLCR Register

15	4	3	0
Reserved		DIV	
R-0		R/W-0	

Legend: R = Read access, W = write access, -0 = value after reset

Note: EALLOW-protected register

Table 3−7. PLLCR Register Field Descriptions

Bits	Name	Description
15−4	Reserved	
3−0	DIV	The DIV field controls whether the PLL is bypassed or not, and sets the PLL clocking ratio when not bypassed. 0000 CLKIN = OSCCLK/2 (PLL bypass) 0001 CLKIN = (OSCCLK * 1.0)/2 0010 CLKIN = (OSCCLK * 2.0)/2

Table 3−7. PLLCR Register Field Descriptions (Continued)

Bits	Name	Description	
		0011	CLKIN = (OSCCLK * 3.0)/2
		0100	CLKIN = (OSCCLK * 4.0)/2
		0101	CLKIN = (OSCCLK * 5.0)/2
		0110	CLKIN = (OSCCLK * 6.0)/2
		0111	CLKIN = (OSCCLK * 7.0)/2
		1000	CLKIN = (OSCCLK * 8.0)/2
		1001	CLKIN = (OSCCLK * 9.0)/2
		1010	CLKIN = (OSCCLK * 10.0)/2
		1011−1111	Reserved

† The PLLCR register is reset to a known state by the $\overline{\text{XRS}}$ reset line. If a reset is issued by the debugger, the PLL clocking ratio is not changed.

3.2.2 External Reference Oscillator Clock Option

TI recommends that customers have the resonator/crystal vendor characterize the operation of their device with the DSP chip. The resonator/crystal vendor has the equipment and expertise to tune the tank circuit. The vendor can also advise the customer regarding the proper tank component values that ensures start-up and stability over the entire operating range.

3.3 Low-Power Modes Block

The low-power modes on the 281x devices are similar to the 240x devices. Table 3−8 summarizes the various modes.

Table 3−8. 281x Low-Power Modes

Mode	LPMCR0[1:0]	OSCCLK	CLKIN	SYSCLKOUT	Exit†
IDLE	00	On	On	On‡	$\overline{\text{XRS}}$, WAKEINT, Any Enabled Interrupt, XNMI_XINT13
STANDBY	01	On (watchdog still running)	Off	Off	$\overline{\text{XRS}}$, WAKEINT, XINT1, XNMI_XINT13, T1/2/3/4CTRIP, C1/2/3/4/5/6TRIP, SCIRXDA, SCIRXDB, CANRX, Debugger§
HALT	1X	Off (oscillator and PLL turned off, watchdog not functional)	Off	Off	$\overline{\text{XRS}}$, XNMI_XINT13, Debugger§

† The Exit column lists which signals or under what conditions the low power mode is exited. This signal must be kept low long enough for an interrupt to be recognized by the device. Otherwise the IDLE mode is not exited and the device goes back into the indicated low power mode.
‡ The IDLE mode on the 28x behaves differently than on the 24x/240x. On the 28x, the clock output from the CPU (SYSCLKOUT) is still functional while on the 24x/240x the clock is turned off.
§ On the 28x, the JTAG port can still function even if the clock to the CPU (CLKIN) is turned off.

The various low-power modes operate as follows:

IDLE Mode: This mode is exited by any enabled interrupt or an NMI that is recognized by the processor. The LPM block performs no tasks during this mode as long as the LPMCR0[1:0] bits are set to 00.

HALT Mode: Only the $\overline{\text{XRS}}$ and XNMI_XINT13 external signals can wake the device from HALT mode. The XNMI input to the CPU has an enable/disable bit in the XMNICR register.

STANDBY Mode: All other signals (including XNMI) wake the device from STANDBY mode if selected by the LPMCR1 register. You must select which signal(s) wake the device. The selected signal(s) are also qualified by the OSCCLK before waking the device. The number of OSCCLK cycles is specified in the LPMCR0 register.

The low-power modes are controlled by the LPMCR0 register (see Figure 3−8) and the LPMCR1 register (see Figure 3−9).

Figure 3−8. Low Power Mode Control 0 (LPMCR0) Register

15	8	7	2	1	0
Reserved		QUALSTDBY		LPM	
R-0		R/W-1		R/W-0	

Legend: R = Read access, W = write access, -0 = value after reset

Note: EALLOW-protected register

Table 3−9. Low Power Mode Control 0 (LPMCR0) Register Field Descriptions

Bits	Field	Description†
15−8	Reserved	Reserved
7−2	QUALSTDBY	Select number of OSCCLK clock cycles to qualify the selected inputs when waking the device from STANDBY mode: 000000 = 2 OSCCLKs 000001 = 3 OSCCLKs . 111111 = 65 OSCCLKs
1−0	LPM‡	These bits set the low power mode for the device. 00 Set the low power mode to IDLE 01 Set the low power mode to STANDBY 1x Set the low power mode to HALT

† These bits are cleared by a reset ($\overline{\text{XRS}}$).

‡ The low power mode bits (LPM) are only valid when the IDLE instruction is executed. Therefore, the user must set the LPM bits to the appropriate mode before executing the IDLE instruction.

Figure 3−9. Low Power Mode Control 1 (LPMCR1) Register

15	14	13	12	11	10	9	8
CANRX	SCIRXB	SCIRXA	C6TRIP	C5TRIP	C4TRIP	C3TRIP	C2TRIP
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
C1TRIP	T4CTRIP	T3CTRIP	T2CTRIP	T1CTRIP	WDINT	XNMI	XINT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Legend: R = Read access, W = write access, -0 = value after reset

Note: EALLOW-protected register

Table 3−10. Low Power Mode Control 1 (LPMCR1) Register Field Descriptions

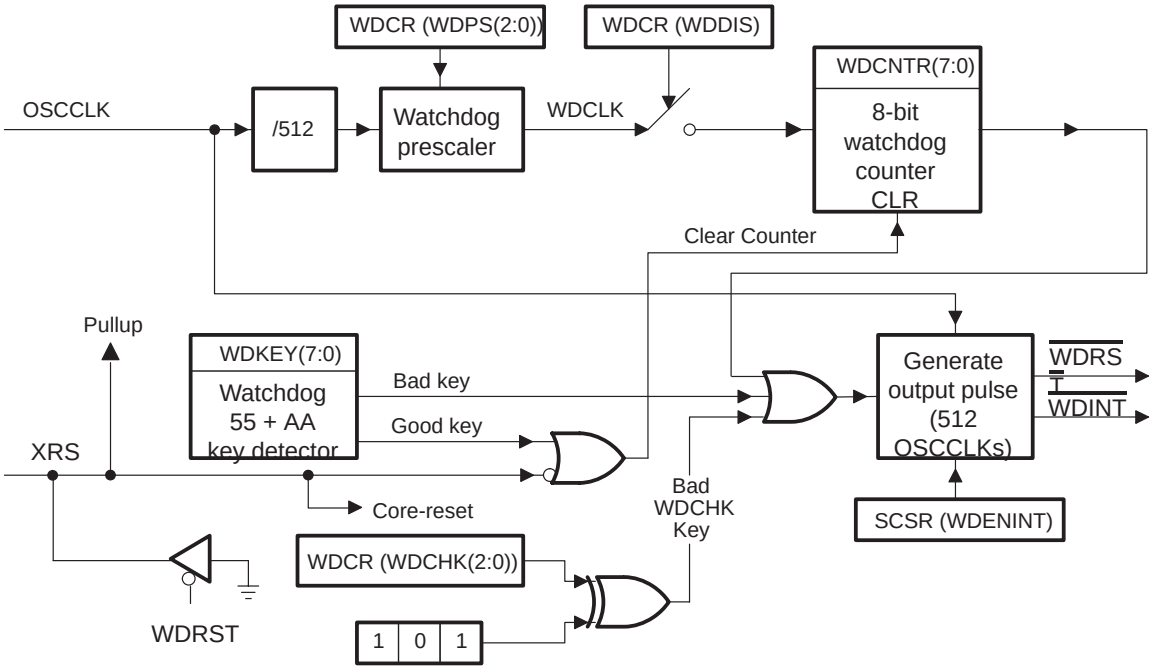
Bits	Field	Description
15	CANRX	If the respective bit is set to 1, it enables the selected signal to wake the device from STANDBY mode. If the bit is cleared, the signal has no effect.
14	SCIRXB	
13	SCIRXA	
12	C6TRIP	
11	C5TRIP	
10	C4TRIP	
9	C3TRIP	
8	C2TRIP	
7	C1TRIP	
6	T4CTRIP	
5	T3CTRIP	
4	T2CTRIP	
3	T1CTRIP	
2	$\overline{\text{WDINT}}$	
1	XNMI_XINT13	
0	XINT1	

† These bits are cleared by a reset ($\overline{\text{XRS}}$).

3.4 Watchdog Block

The watchdog block on the 281x is similar to the one used on the 240x devices. The watchdog module generates an output pulse, 512 oscillator clocks wide (OSCCLK), whenever the 8-bit watchdog up counter has reached its maximum value. To prevent this, the user disables the counter or the software must periodically write a 0x55 + 0xAA sequence into the watchdog key register which resets the watchdog counter. Figure 3−10 shows the various functional blocks within the watchdog module.

Figure 3−10. Watchdog Module



NOTE A: The $\overline{\text{WDRST}}$ signal is driven low for 512 OSCCLK cycles.

The watchdog interrupt ($\overline{\text{WDINT}}$) signal enables the watchdog to be used as a wakeup from IDLE/STANDBY mode timer. If the watchdog interrupt is used to wake up from an IDLE or STANDBY low power mode condition, then you should make sure that the $\overline{\text{WDINT}}$ signal goes back high again before attempting to go back into the IDLE or STANDBY mode. You can determine the state of this signal by reading the WDENINT bit in the SCSR register.

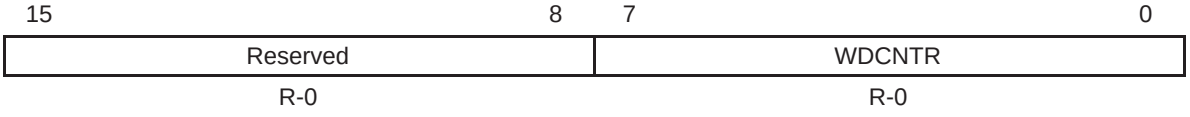
In STANDBY mode, all peripherals are turned off on the device. The only peripheral that remains functional is the watchdog. The WATCHDOG module runs off the PLL clock or the oscillator clock. The $\overline{\text{WDINT}}$ signal is fed to the

LPM block so that it can wake the device from STANDBY (if enabled). See Low-Power Modes Block section of this data sheet for more details.

In IDLE mode, the $\overline{\text{WDINT}}$ signal can generate an interrupt to the CPU, (the WAKEINT interrupt in the PIE), to take the CPU out of IDLE mode.

In HALT mode, this feature cannot be used because the oscillator (and PLL) are turned off and, therefore, so is the watchdog.

Figure 3−11. Watchdog Counter (WDCNTR) Register



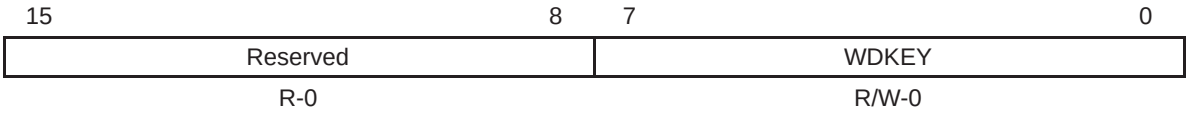
Legend: R = Read access, W = write access, -0 = value after reset

Note: EALLOW-protected register

Table 3−11. Watchdog Counter (WDCNTR) Register Field Descriptions

Bits	Field	Description
15−8	Reserved	
7−0	WDCNTR	These bits contain the current value of the WD counter. The 8-bit counter continually increments at the watchdog clock (WDCLK), rate. If the counter overflows, then the watchdog initiates a reset. If the WDKEY register is written with a valid combination, then the counter is reset to zero. The watchdog clock rate is configured in the WDCR register.

Figure 3−12. Watchdog Reset Key (WDKEY) Register



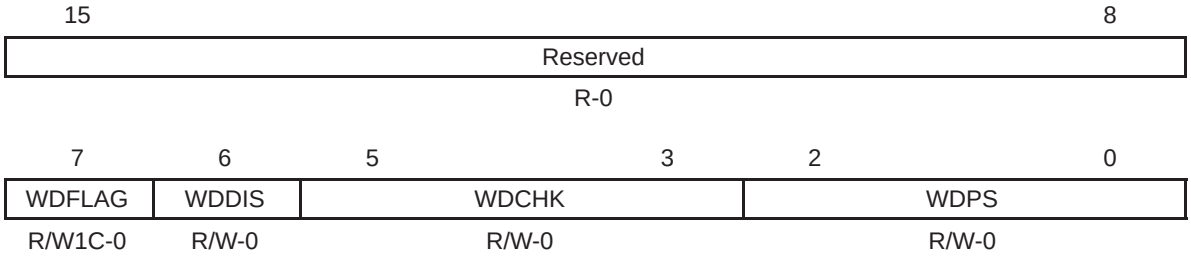
Legend: R = Read access, W = write access, -0 = value after reset

Note: EALLOW-protected register

Table 3−12. Watchdog Reset Key (WDKEY) Register Field Descriptions

Bits	Field	Description
15−8	Reserved	Reserved
7−0	WDKEY	Writing 0x55 followed by 0xAA causes the WDCNTR bits to be cleared. Writing any other value causes an immediate watchdog reset to be generated. Reads return the value of the WDCR register.

Figure 3−13. Watchdog Control (WDCR) Register



Legend: R = Read access, W = write access, W1C = write 1 to clear, -0 = value after reset

Note: EALLOW-protected register

Table 3−13. Watchdog Control (WDCR) Register Field Descriptions

Bits	Field	Description
15−8	Reserved	
7	WDFLAG	Watchdog reset status flag bit. WDFLAG, if set, indicates a watchdog reset ($\overline{\text{WDRST}}$) generated the reset condition. If 0, then it was an external device or power-up reset condition. This bit remains latched until the user writes a 1 to clear the condition. Writes of 0 are ignored.
6	WDDIS	Watchdog disable. Writing a 1 to WDDIS disables the watchdog module. Writing a 0 enables the module. WDDISt can only be modified if the WDOVERRIDE bit in the SCSR2 register is set to 1. On reset, the watchdog module is enabled.
5−3	WDCHK(2−0)	Watchdog check. You must ALWAYS write 1,0,1 to these bits whenever a write to this register is performed. Writing any other value causes an immediate reset to the core (if WD enabled).
2−0	WDPS(2−0)	These bits configure the watchdog counter clock (WDCLK) rate relative to OSCCLK/512: 000 WDCLK = OSCCLK/512/1 001 WDCLK = OSCCLK/512/1 010 WDCLK = OSCCLK/512/2 011 WDCLK = OSCCLK/512/4 100 WDCLK = OSCCLK/512/8 101 WDCLK = OSCCLK/512/16 110 WDCLK = OSCCLK/512/32 111 WDCLK = OSCCLK/512/64

When the $\overline{\text{XRS}}$ line is low, the WDFLAG bit is forced low. The WDFLAG bit is only set if a rising edge on $\overline{\text{WDRST}}$ signal is detected (after synch and a 4 cycle

delay) and the $\overline{\text{XRS}}$ signal is high. If the $\overline{\text{XRS}}$ signal is low when $\overline{\text{WDRST}}$ goes high, then the WDFLAG bit remains at 0. In a typical application, the $\overline{\text{WDRST}}$ signal connects to the $\overline{\text{XRS}}$ input. Hence to distinguish between a watchdog reset and an external device reset, an external reset must be longer in duration then the watchdog pulse.

3.4.1 Emulation Considerations

The watchdog module behaves as follows under various debug conditions:

CPU Suspended: When the CPU is suspended, the watchdog clock (WDCLK) is suspended.

Run-Free Mode: When the CPU is placed in run-free mode, then the watchdog module resumes operation as normal.

Real-Time Single-Step Mode: When the CPU is in real-time single-step mode, the watchdog clock (WDCLK) is suspended. The watchdog remains suspended even within real-time interrupts.

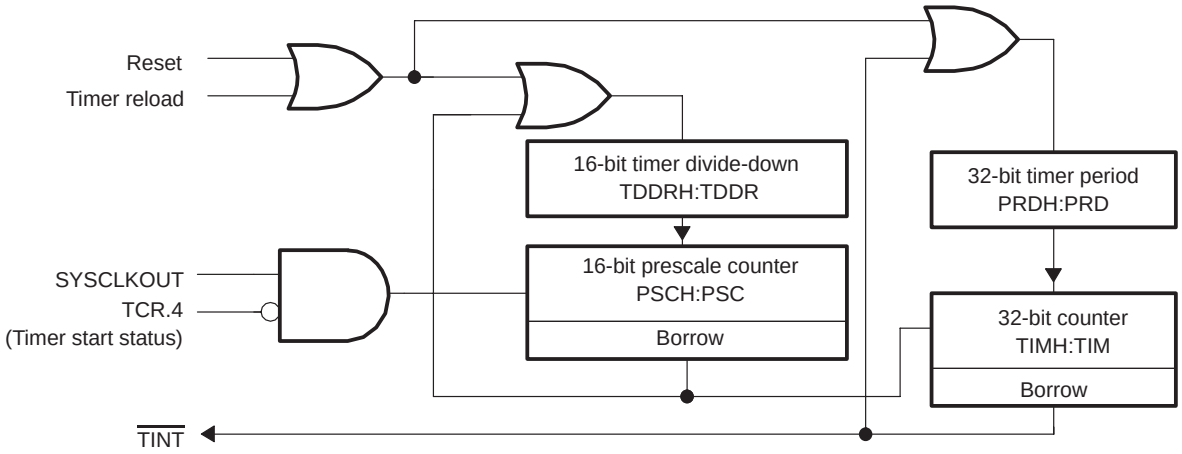
Real-Time Run-Free Mode: When the CPU is in real-time run-free mode, the watchdog operates as normal.

3.5 32-Bit CPU Timers 0/1/2

This section describes the three 32-bit CPU-timers on the 281x devices (TIMER0/1/2).

CPU-Timers 1 and 2 are reserved for the real-time OS (such as DSP-BIOS).† CPU-Timer 0 can be used in user applications.

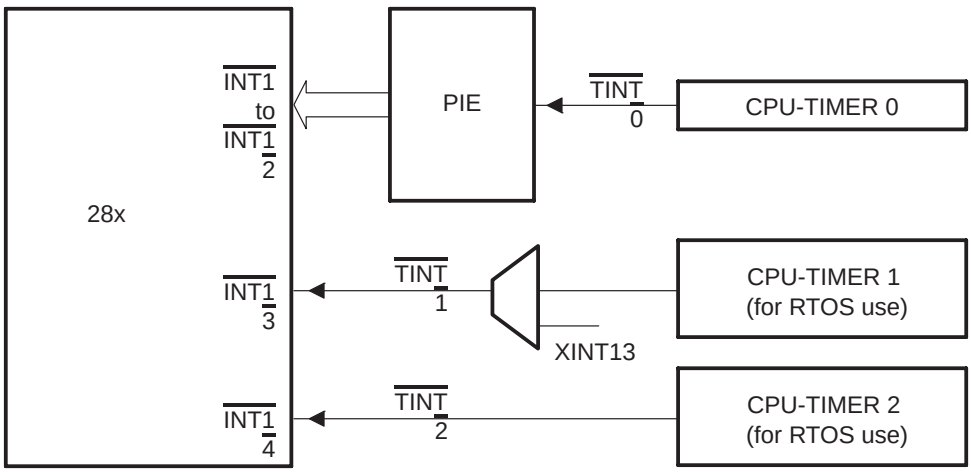
Figure 3−14. CPU-Timers



NOTE A: The CPU-Timers are different from the general-purpose (GP) timers that are present in the event manager modules (EVA, EVB).

In the 281x devices, the CPU-timer interrupt signals ($\overline{\text{TINT0}}$, $\overline{\text{TINT1}}$, $\overline{\text{TINT2}}$) are connected as shown in Figure 3−15.

Figure 3−15. CPU-Timer Interrupts Signals and Output Signal



NOTES: A. The timer registers are connected to the Memory Bus of the 28x processor.
B. The timing of the timers is synchronized to SYSCLKOUT of the processor clock.

The general operation of the CPU-timer is as follows: The 32-bit counter register TIMH:TIM is loaded with the value in the period register PRDH:PRD. The counter register decrements at the SYSCLKOUT rate of the 28x. When the counter reaches 0, a timer interrupt output signal generates an interrupt pulse. The registers listed in Table 3−14 are used to configure the timers.

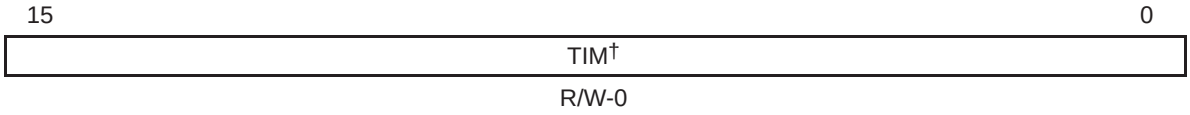
Table 3−14. CPU-Timers 0, 1, 2 Configuration and Control Registers

Name	Address	Size (x16)	Description
TIMER0TIM	0x0000 0C00	1	CPU-Timer 0, Counter Register
TIMER0TIMH	0x0000 0C01	1	CPU-Timer 0, Counter Register High
TIMER0PRD	0x0000 0C02	1	CPU-Timer 0, Period Register
TIMER0PRDH	0x0000 0C03	1	CPU-Timer 0, Period Register High
TIMER0TCR	0x0000 0C04	1	CPU-Timer 0, Control Register
Reserved	0x0000 0C05	1	
TIMER0TPR	0x0000 0C06	1	CPU-Timer 0, Prescale Register
TIMER0TPRH	0x0000 0C07	1	CPU-Timer 0, Prescale Register High
TIMER1TIM	0x0000 0C08	1	CPU-Timer 1, Counter Register
TIMER1TIMH	0x0000 0C09	1	CPU-Timer 1, Counter Register High
TIMER1PRD	0x0000 0C0A	1	CPU-Timer 1, Period Register
TIMER1PRDH	0x0000 0C0B	1	CPU-Timer 1, Period Register High
TIMER1TCR	0x0000 0C0C	1	CPU-Timer 1, Control Register
Reserved	0x0000 0C0D	1	
TIMER1TPR	0x0000 0C0E	1	CPU-Timer 1, Prescale Register
TIMER1TPRH	0x0000 0C0F	1	CPU-Timer 1, Prescale Register High
TIMER2TIM	0x0000 0C10	1	CPU-Timer 2, Counter Register
TIMER2TIMH	0x0000 0C11	1	CPU-Timer 2, Counter Register High
TIMER2PRD	0x0000 0C12	1	CPU-Timer 2, Period Register
TIMER2PRDH	0x0000 0C13	1	CPU-Timer 2, Period Register High
TIMER2TCR	0x0000 0C14	1	CPU-Timer 2, Control Register
Reserved	0x0000 0C15	1	

Table 3−14. CPU-Timers 0, 1, 2 Configuration and Control Registers (Continued)

Name	Address	Size (x16)	Description
TIMER2TPR	0x0000 0C16	1	CPU-Timer 2, Prescale Register
TIMER2TPRH	0x0000 0C17	1	CPU-Timer 2, Prescale Register High
Reserved	0x0000 0C18 0x0000 0C3F	40	

Figure 3−16. TIMERxTIM Register



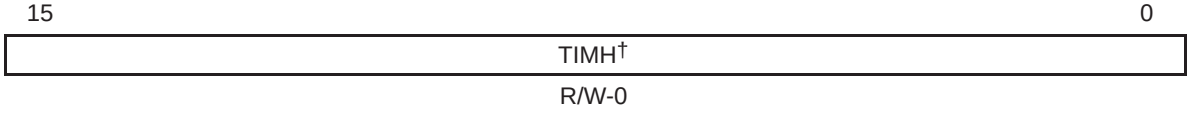
Legend: R = Read access, W = write access, -0 = value after reset

Table 3−15. TIMERxTIM Register Field Descriptions

Bits	Field	Description
15−0	TIM	CPU-Timer Counter Registers (TIMH:TIM): The TIM register holds the low 16 bits of the current 32-bit count of the timer. The TIMH register holds the high 16 bits of the current 32-bit count of the timer. The TIMH:TIM decrements by one every (TDDRH:TDDR+1) clock cycles, where TDDRH:TDDR is the timer prescale divide-down value. When the TIMH:TIM decrements to zero, the TIMH:TIM register is reloaded with the period value contained in the PRDH:PRD registers. The timer interrupt (TINT) signal is generated.

† x = 0, 1, or 2

Figure 3−17. TIMERxTIMH Register



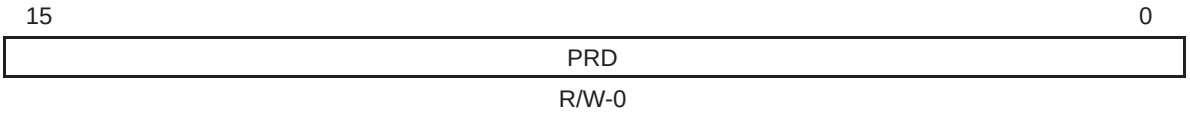
Legend: R = Read access, W = write access, -0 = value after reset

Table 3−16. TIMERxTIMH Register Field Descriptions

Bits	Field	Description
15−0	TIMH	See description for TIMERxTIM.

† x = 0, 1, or 2

Figure 3−18. TIMERxPRD Register†



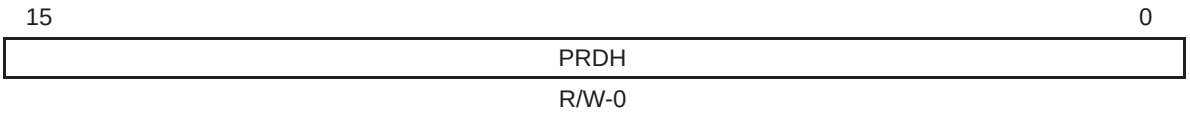
Legend: R = Read access, W = write access, -0 = value after reset

Table 3−17. TIMERxPRD Register Field Descriptions

Bits	Field	Description
15−0	PRD	CPU-Timer Period Registers (PRDH:PRD): The PRD register holds the low 16 bits of the 32-bit period. The PRDH register holds the high 16 bits of the 32-bit period. When the TIMH:TIM decrements to zero, the TIMH:TIM register is reloaded with the period value contained in the PRDH:PRD registers, at the start of the next timer input clock cycle (the output of the prescaler). The PRDH:PRD contents are also loaded into the TIMH:TIM when you set the timer reload bit (TRB) in the Timer Control Register (TCR).

† x = 0, 1, or 2

Figure 3−19. TIMERxPRDH Register †



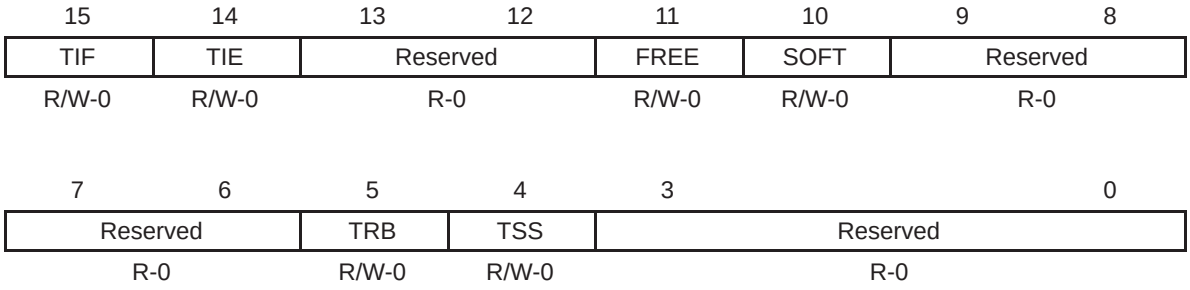
Legend: R = Read access, W = write access, -0 = value after reset

Table 3−18. TIMERxPRDH Register Field Description

Bits	Field	Description
15−0	PRDH	See description for TIMERxPRD

† x = 0, 1, or 2

Figure 3−20. TIMERxTCR Register†



Legend: R = Read access, W = write access, -0 = value after reset

Table 3−19. TIMERxTCR Register Field Descriptions

Bits	Field	Description
15	TIF	CPU-Timer Interrupt Flag. This flag gets set when the timer decrements to zero. TIF can be cleared by software writing a 1, but it can only be set by the timer reaching zero. 0 Writing a zero has no effect. 1 Writing a 1 to this bit clears it
14	TIE	CPU-Timer Interrupt Enable. If the timer decrements to zero, and TIE is set, the timer asserts its interrupt request.
13−12	Reserved	Reserved
11 10	FREE SOFT	CPU-Timer Emulation Modes: These bits are special emulation bits that determine the state of the timer when a breakpoint is encountered in the high-level language debugger. If the FREE bit is set to 1, then, upon a software breakpoint, the timer continues to run (that is, free runs). In this case, SOFT is a *don't care*. But if FREE is 0, then SOFT takes effect. In this case, if SOFT = 0, the timer halts the next time the TIMH:TIM decrements. If the SOFT bit is 1, then the timer halts when the TIMH:TIM has decremented to zero. **FREE / SOFT / CPU-Timer Emulation Mode** 0 0 Stop after the next decrement of the TIMH:TIM (hard stop) 0 1 Stop after the TIMH:TIM decrements to 0 (soft stop) 1 0 Free run 1 1 Free run Note: That in the SOFT STOP mode, the timer generates an interrupt before shutting down (since reaching 0 is the interrupt causing condition).
9−6	Reserved	Reserved

Table 3−19. TIMERxTCR Register Field Descriptions (Continued)

Bits	Field	Description
5	TRB	CPU-Timer Reload bit. When you write a 1 to TRB, the TIMH:TIM is loaded with the value in the PRDH:PRD, and the prescaler counter (PSCH:PSC) is loaded with the value in the timer divide-down register (TDDRH:TDDR). The TRB bit is always read as zero.
4	TSS	CPU-Timer stop status bit. TSS is a 1-bit flag that stops or starts the timer. 0 To start or restart the timer, set TSS to 0. At reset, TSS is cleared to 0 and the timer immediately starts. 1 To stop the timer, set TSS to 1.
3−0	Reserved	Reserved

† x = 0, 1, or 2

Figure 3−21. TIMERxTPR Register†

15	8	7	0
PSC		TDDR	
R-0		R/W-0	

Legend: R = Read access, W = write access, -0 = value after reset

Table 3−20. TIMERxTPR Register Field Descriptions

Bits	Field	Description
15−8	PSC	CPU-Timer Prescale Counter. These bits hold the current prescale count for the timer. For every timer clock source cycle that the PSCH:PSC value is greater than 0, the PSCH:PSC decrements by one. One timer clock (output of the timer prescaler) cycle after the PSCH:PSC reaches 0, the PSCH:PSC is loaded with the contents of the TDDRH:TDDR, and the timer counter register (TIMH:TIM) decrements by one. The PSCH:PSC is also reloaded whenever the timer reload bit (TRB) is set by software. The PSCH:PSC can be checked by reading the register, but it cannot be set directly. It must get its value from the timer divide-down register (TDDRH:TDDR). At reset, the PSCH:PSC is set to 0.
7−0	TDDR	CPU-Timer Divide-Down. Every (TDDRH:TDDR + 1) timer clock source cycles, the timer counter register (TIMH:TIM) decrements by one. At reset, the TDDRH:TDDR bits are cleared to 0. To increase the overall timer count by an integer factor, write this factor minus one to the TDDRH:TDDR bits. When the prescaler counter (PSCH:PSC) value is 0, one timer clock source cycle later, the contents of the TDDRH:TDDR reload the PSCH:PSC, and the TIMH:TIM decrements by one. TDDRH:TDDR also reloads the PSCH:PSC whenever the timer reload bit (TRB) is set by software.

† x = 0, 1, or 2

Table 3−21. TIMERxTPRH Register†

15	8	7	0
PSCH		TDDRH	
R-0		R/W-0	

Legend: R = Read access, W = write access, -0 = value after reset

Table 3−22. TIMERxTPRH Register Field Descriptions

Bits	Field	Description
15−8	PSCH	See description of TIMERxTPR.
7−0	TDDRH	See description of TIMERxTPR.

† x = 0, 1, or 2

Chapter 4

General-Purpose Input/Output (GPIO)

The GPIO MUX registers are used to select the operation of shared pins on the 281x devices. The pins can be individually selected to operate as Digital I/O or connected to Peripheral I/O signals (via the GPxMUX registers). If selected for Digital I/O mode, registers are provided to configure the pin direction (via the GPxDIR registers) and to qualify the input signal to remove unwanted noise (via the GPxQUAL) registers).

Topic **Page**

4.1 GPIO MUX

Table 4−1 lists the GPIO MUX Registers.

Table 4−1. GPIO MUX Registers†‡

Name	Address	Size (x16)	Register Description
GPAMUX	0x0000 70C0	1	GPIO A MUX Control Register
GPADIR	0x0000 70C1	1	GPIO A Direction Control Register
GPAQUAL	0x0000 70C2	1	GPIO A Input Qualification Control Register
Reserved	0x0000 70C3	1	
GPBMUX	0x0000 70C4	1	GPIO B MUX Control Register
GPBDIR	0x0000 70C5	1	GPIO B Direction Control Register
GPBQUAL	0x0000 70C6	1	GPIO B Input Qualification Control Register
Reserved	0x0000 70C7 − 0x0000 70CB	5	
GPDMUX	0x0000 70CC	1	GPIO D MUX Control Register
GPDDIR	0x0000 70CD	1	GPIO D Direction Control Register
GPDQUAL	0x0000 70CE	1	GPIO D Input Qualification Control Register
Reserved	0x0000 70CF	1	
GPEMUX	0x0000 70D0	1	GPIO E MUX Control Register
GPEDIR	0x0000 70D1	1	GPIO E Direction Control Register
GPEQUAL	0x0000 70D2	1	GPIO E Input Qualification Control Register
Reserved	0x0000 70D3	1	
GPFMUX	0x0000 70D4	1	GPIO F MUX Control Register
GPFDIR	0x0000 70D5	1	GPIO F Direction Control Register
Reserved	0x0000 70D6 − 0x0000 70D7	2	
GPGMUX	0x0000 70D8	1	GPIO G MUX Control Register

† For reserved locations, read values are undefined and writes have no effect.
‡ These registers are EALLOW protected. This prevents spurious writes from overwriting the contents and corrupting the system.

Table 4−1. GPIO MUX Registers†‡ (Continued)

Name	Address	Size (x16)	Register Description
GPGDIR	0x0000 70D9	1	GPIO G Direction Control Register
Reserved	0x0000 70DA − 0x0000 70DF	6	

† For reserved locations, read values are undefined and writes have no effect.
‡ These registers are EALLOW protected. This prevents spurious writes from overwriting the contents and corrupting the system.

If configured for Digital I/O mode, additional registers are provided for setting individual I/O signals (via the GPxSET registers), for clearing individual I/O signals (via the GPxCLEAR registers), for toggling individual I/O signals (via the GPxTOGGLE registers), or for reading/writing to the individual I/O signals (via the GPxDAT registers). Table 4−2 lists the GPIO Data Registers.

Table 4−2. GPIO Data Registers†‡

Name	Address	Size (x16)	Register Description
GPADAT	0x0000 70E0	1	GPIO A Data Register
GPASET	0x0000 70E1	1	GPIO A Set Register
GPACLEAR	0x0000 70E2	1	GPIO A Clear Register
GPATOGGLE	0x0000 70E3	1	GPIO A Toggle Register
GPBDAT	0x0000 70E4	1	GPIO B Data Register
GPBSET	0x0000 70E5	1	GPIO B Set Register
GPBCLEAR	0x0000 70E6	1	GPIO B Clear Register
GPBTOGGLE	0x0000 70E7	1	GPIO B Toggle Register
Reserved	0x0000 70E8 − 0x0000 70EB	4	
GPDDAT	0x0000 70EC	1	GPIO D Data Register
GPDSET	0x0000 70ED	1	GPIO D Set Register
GPDCLEAR	0x0000 70EE	1	GPIO D Clear Register
GPDTOGGLE	0x0000 70EF	1	GPIO D Toggle Register

† For reserved locations, read values are undefined and writes have no effect.
‡ These registers are NOT EALLOW protected and is typically accessed regularly by the user.

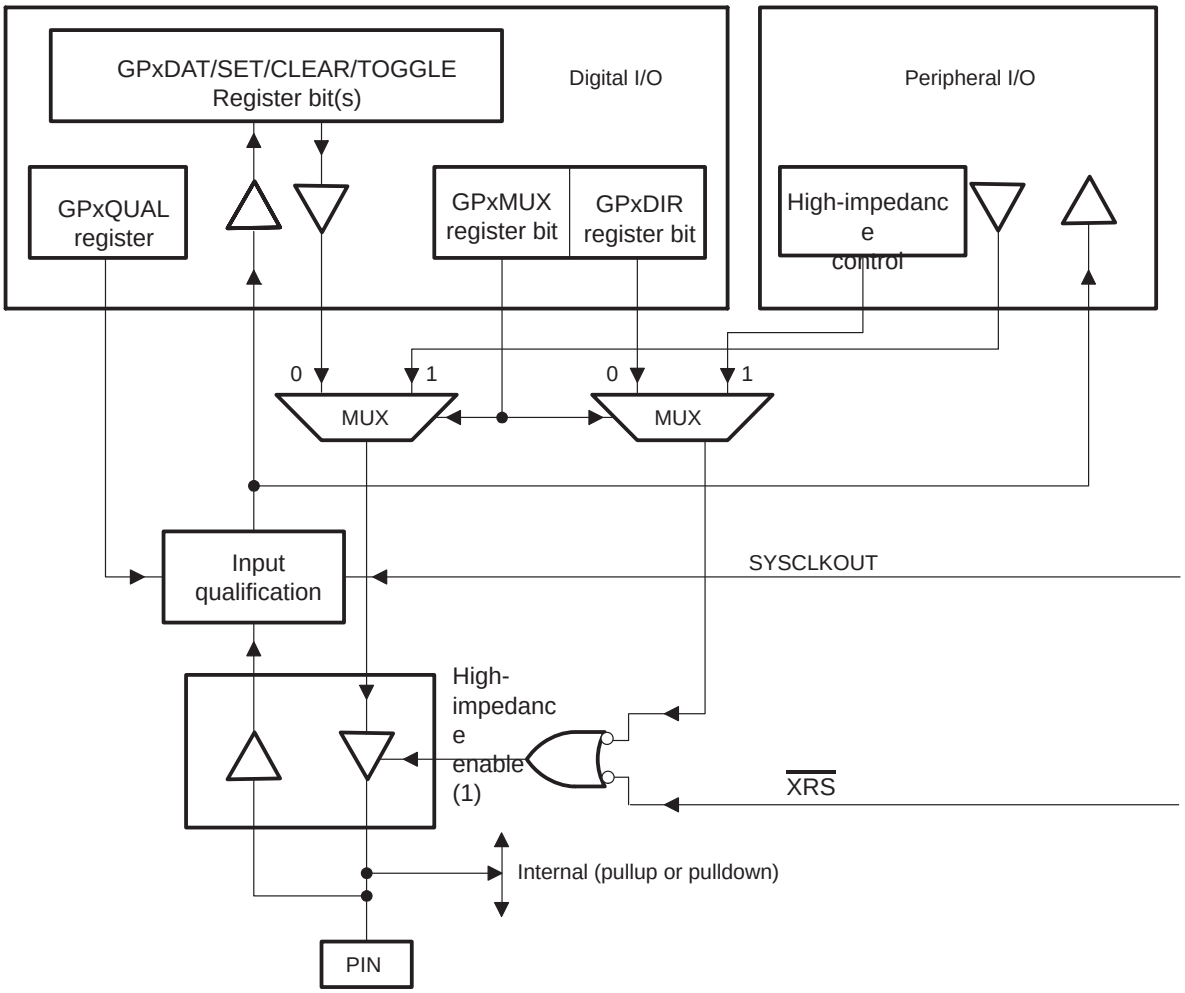
Table 4−2. GPIO Data Registers†‡ (Continued)

Name	Address	Size (x16)	Register Description
GPEDAT	0x0000 70F0	1	GPIO E Data Register
GPESET	0x0000 70F1	1	GPIO E Set Register
GPECLEAR	0x0000 70F2	1	GPIO E Clear Register
GPETOGGLE	0x0000 70F3	1	GPIO E Toggle Register
GPFDAT	0x0000 70F4	1	GPIO F Data Register
GPFSET	0x0000 70F5	1	GPIO F Set Register
GPFCLEAR	0x0000 70F6	1	GPIO F Clear Register
GPFTOGGLE	0x0000 70F7	1	GPIO F Toggle Register
GPGDAT	0x0000 70F8	1	GPIO G Data Register
GPGSET	0x0000 70F9	1	GPIO G Set Register
GPGCLEAR	0x0000 70FA	1	GPIO G Clear Register
GPGTOGGLE	0x0000 70FB	1	GPIO G Toggle Register
Reserved	0x0000 70FC − 0x0000 70FF	4	

† For reserved locations, read values are undefined and writes have no effect.
‡ These registers are NOT EALLOW protected and is typically accessed regularly by the user.

Figure 4−1 shows how the various register bits select the various modes of operation.

Figure 4−1. GPIO/Peripheral Pin MUXing



Notes: 1) Via the GPxDAT register, the state of any PIN can be read, regardless of the operating mode.

2) The GPxQUAL register specifies the qualification sampling period. The sampling window is six samples wide and the output is only changed when all samples are the same (all 0s or all 1s) as shown in Figure 4−3. This feature removes unwanted spikes from the input signal.

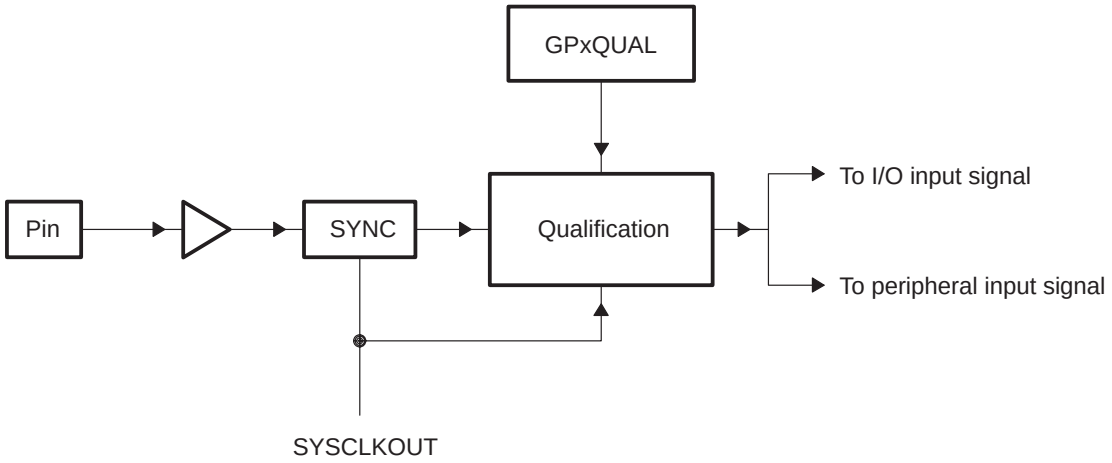
The input function of the GPIO pin and the input path to the peripheral are always enabled. It is the output function of the GPIO pin that is multiplexed with the output path of the primary (peripheral) function. Since the output buffer of a pin connects back to the input buffer, any GPIO signal present at the pin is propagated to the peripheral module as well. Therefore, when a pin is configured for GPIO operation, the corresponding peripheral functionality (and interrupt−generating capability) must be disabled. Otherwise, interrupts may be inadvertently triggered.

4.2 Input Qualification

Two types of input qualification are performed on inputs.

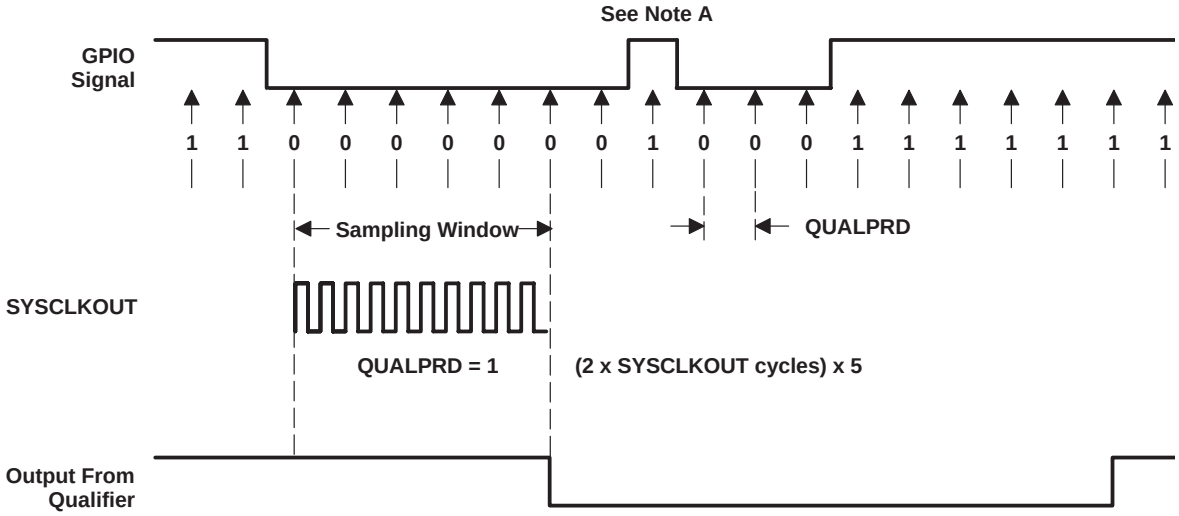
In type 1 qualification, the input signal is first synchronized to SYSCLKOUT and the signal qualified by the specified number of signals before the input is allowed to change. Figure 4−2 shows how type 1 input qualification is performed to eliminate unwanted noise.

Figure 4−2. Type 1 Input Qualification



To qualify the signal, the input is synchronized to SYSCLKOUT and then sampled at a regular period. The sampling period is specified by the GPxQUAL register. The sampling window is six samples wide and the input is changed only when all six samples are the same as shown in Figure 4−3. Because the incoming signal is asynchronous, there can be up to one SYSCLKOUT delay for synchronization before the qualification sampling window begins.

Figure 4−3. Input Qualifier Clock Cycles

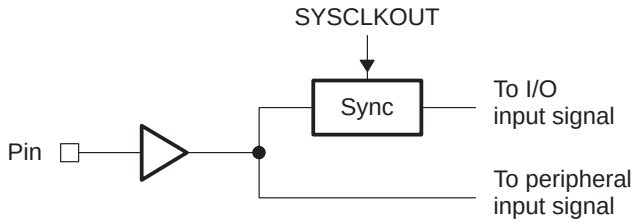


NOTES: A. This glitch will be ignored by the input qualifier. The QUALPRD bit field specifies the qualification sampling perio from 00 to 0xFF. Input qualification is not applicable when QUALPRD = 00. For any other value “n”, the qualifica period in 2n SYSCLKOUT cycles (i.e., at every 2n SYSCLKOUT cycle, the GPIO pin will be sampled). Six consec must be of the same value for a given input to be recognized.

B. For the qualifier to detect the change, the input should be stable for 10 SYSCLKOUT cycles or greater. In other wo should be stable for (5 x QUALPRD x 2) SYSCLKOUT cycles. This would ensure six sampling windows for dete Since external signals are driven asynchronously, an 11-SYSCLKOUT-wide pulse ensures reliable recognition.

Figure 4−4 shows type 2 qualification, where only the digital I/O signal is synchronized to the core clock (SYSCLKOUT). The peripheral signal connects directly to the pin without qualification or synchronization because some peripherals perform their own synchronization.

Figure 4−4. Type 2 Input Qualification



The type of input qualification used for each signal is indicated in the GPxMUX Register definitions in Section 4.3.

4.3 Register Functional Overview

Each GP I/O port is controlled by the MUX, direction, data, set, clear, and toggle registers. In the following section, the bit mapping for the GPxMUX and GPxDIR registers is shown. The corresponding data, set, clear, and toggle registers have identical bit to I/O mapping.

GPxMUX Registers

Each I/O port has one MUX register. The MUX registers are used to select between the corresponding peripheral operation and the I/O operation of each of the I/O pins. At reset all GP I/O pins are configured as I/O pins.

If GPxMUX.bit = 0, then the pin is configured as an I/O

If GPxMUX.bit = 1, then the pin is configured for the peripheral functionality

As shown in Figure 4−1, the input function of the I/O and the input path to the peripheral are always enabled. It is the output path of the peripheral that is MUXed with the output of the I/O. Therefore, when a pin is configured as an I/O pin, the corresponding peripheral function must be disabled. Otherwise, interrupts may be inadvertently triggered.

GPxDIR Registers

Each I/O port has one direction control register. The direction register controls whether the corresponding I/O pin is configured as an input or an output. At reset all GP I/O pins are configured as inputs.

If GPxDIR.bit = 0, then the pin is configured as an input

If GPxDIR.bit = 1, then the pin is configured as an output

On reset, the default value for all GPxMUX and GPxDIR register bits is 0. That is, at reset all I/O ports are configured as input pins. Before changing the direction of the port from input to output using the GPxDIR register bit, the current level of the pin is reflected in the GPxDAT register. The GPxDAT register is described later in this section. When the direction of the port is changed from input to output, the value already in the GPxDAT register is used to determine the state of the pin.

For example, if a pin has an internal pullup, then after reset the pullup would force the GPxDAT Register bit to a1 to reflect the current state of the pin. When the direction of the port changes from an input to an output the 1 already in the GPxDAT Register forces the pin to the same level. In this manner, the pin can be switched from input to output without a glitch.

GPxDAT Registers

Each I/O port has one data register. The data register is a R/W register that reflects the current state of the input I/O signal after qualification. Writing to the register sets the corresponding state of any I/O signal that is configured as an output.

If GPxDAT.bit = 0, AND the pin is an output, then pull the pin low

If GPxDAT.bit = 1, AND the pin is an output, then pull the pin high

When using the GPxDAT register to change the level of an output pin, you should be cautious not to accidentally change the level of another pin. For example, if you mean to change the level of GPIOA0 by writing to the GPADAT register bit 0, using a read-modify-write instruction. The problem can occur if another I/O port A signal changes the level between the read and the write stage of the instruction. If this happens, the signal level that changed is overwritten by the original value read during the read stage of the instruction. You can avoid this scenario by using the GPxSET, GPxCLEAR, and GPxTOGGLE registers instead.

GPxSET Registers

Each I/O port has one set register. The set registers are write-only registers that read back 0. If the corresponding pin is configured as an output, then writing a 1 to that bit in the set register will pull the corresponding pin high. Writing a 0 to a bit has no effect.

If GPxSET.bit = 0, ignored

If GPxSET.bit = 1, AND the pin is an output, then pull the pin high

GPxCLEAR Registers

Each I/O port has one clear register. The clear registers are write-only registers that read back 0. If the corresponding pin is configured as an output, then writing a 1 to that bit in the clear register will pull the corresponding pin low. Writing a 0 to a bit has no effect.

If GPxCLEAR.bit = 0, ignored

If GPxCLEAR.bit = 1, AND the pin is an output, then pull the pin low

GPxTOGGLE Registers

Each I/O port has one toggle register. The toggle registers are write-only registers that read back 0. If the corresponding pin is configured as an output,

then writing a 1 to that bit in the toggle register will pull the corresponding pin in the opposite direction. That is, if the output pin is low, writing a 1 to the corresponding bit in the toggle register will pull the pin high. Likewise, if the output pin is high, writing a 1 to the corresponding bit in the toggle register will pull the pin low. Writing a 0 to a bit has no effect.

If GPxTOGGLE.bit = 0, ignored

If GPxTOGGLE.bit = 1, AND the pin is an output, then pull the pin in the opposite direction

4.4 Register Bit to I/O Mapping

The following tables describe the register bit to I/O mapping on the 281x devices. For each port, the bit mapping is the same for each of the controlling registers: MUX, direction, set, clear, and toggle. The functionality of these registers is described in more detail in Section 4.3.

Some of the input signals are qualified using either type 1 or type 2 qualification. The type of qualification for each signal is indicated in the Input Qual column of the register bit to I/O mapping tables. More detailed information for the input qualification types is in Section 4.2.

Table 4−3 shows the MUX, direction, set, clear, and toggle register bit to I/O mapping for GPIO port A. Unused I/Os are reserved for future use.

Table 4−3. GPIO A Register Bit to I/O Mapping

Register Bit	Peripheral Name GPAMUX Bit = 1	GPIO Name GPAMUX Bit = 0	GPAMUX/DIR Type	Input Qual
EV-A Peripheral				
0	PWM1 (O)	GPIOA0	R/W-0	Type 1
1	PWM2 (O)	GPIOA1	R/W-0	Type 1
2	PWM3 (O)	GPIOA2	R/W-0	Type 1
3	PWM4 (O)	GPIOA3	R/W-0	Type 1
4	PWM5 (O)	GPIOA4	R/W-0	Type 1
5	PWM6 (O)	GPIOA5	R/W-0	Type 1
6	T1PWM_T1CMP (0)	GPIOA6	R/W-0	Type 1
7	T2PWM_T2CMP (0)	GPIOA7	R/W-0	Type 1
8	CAP1_QEP1 (I)	GPIOA8	R/W-0	Type 1
9	CAP2_QEP2 (I)	GPIOA9	R/W-0	Type 1
10	CAP3_QEPI1 (I)	GPIOA10	R/W-0	Type 1
11	TDIRA (I)	GPIOA11	R/W-0	Type 1
12	TCLKINA (I)	GPIOA12	R/W-0	Type 1
13	$\overline{\text{C1TRIP}}$ (I)	GPIOA13	R/W-0	Type 1

Note: GPAMUX and GPADIR are EALLOW−protected registers.

Table 4−3. GPIO A Register Bit to I/O Mapping (Continued)

Register Bit	Peripheral Name GPAMUX Bit = 1	GPIO Name GPAMUX Bit = 0	GPAMUX/DIR Type	Input Qual
14	$\overline{\text{C2TRIP}}$ (I)	GPIOA14	R/W-0	Type 1
15	$\overline{\text{C3TRIP}}$ (I)	GPIOA15	R/W-0	Type 1

Note: GPAMUX and GPADIR are EALLOW−protected registers.

Figure 4−5. GPIO A Input Qualification Control (GPAQUAL) Register

15	8	7	0
Reserved		QUALPRD	
R-0		R/W-0	

Legend: R = Read access, W = write access, -0 = value after reset

Note: EALLOW-protected register

Table 4−4. GPIO A Input Qualification Control (GPAQUAL) Register Field Descriptions

Bits	Field	Description
15−8	Reserved	
7−0	QUALPRD	Specifies the qualification sampling period: 0x00 No qualification (just SYNC to SYSCLKOUT) 0x01 QUALPRD = 2 SYSCLKOUT cycles 0x02 QUALPRD = 4 SYSCLKOUT cycles … 0xFF QUALPRD = 510 SYSCLKOUT cycles

Table 4−5 shows the MUX, direction, set, clear, and toggle register bit to I/O mapping for GPIO port B. Unused I/Os are reserved for future use.

Table 4−5. GPIO B Register Bit to I/O Mapping

Register Bit	Peripheral Name GPBMUX Bit = 1	GPIO Name GPBMUX Bit = 0	GPBMUX/DIR Type	Input Qual
EV-B Peripheral				
0	PWM7 (O)	GPIOB0	R/W−0	Type 1
1	PWM8 (O)	GPIOB1	R/W-0	Type 1
2	PWM9 (O)	GPIOB2	R/W-0	Type 1
3	PWM10 (O)	GPIOB3	R/W-0	Type 1

Note: GPBMUX and GPBDIR are EALLOW−protected registers.

Table 4−5. GPIO B Register Bit to I/O Mapping(Continued)

Register Bit	Peripheral Name GPBMUX Bit = 1	GPIO Name GPBMUX Bit = 0	Type	Input Qual
4	PWM11 (O)	GPIOB4	R/W-0	Type 1
5	PWM12 (O)	GPIOB5	R/W-0	Type 1
6	T3PWM_T3CMP (0)	GPIOB6	R/W-0	Type 1
7	T4PWM_T4CMP (0)	GPIOB7	R/W-0	Type 1
8	CAP4_QEP3 (I)	GPIOB8	R/W-0	Type 1
9	CAP5_QEP4 (I)	GPIOB9	R/W-0	Type 1
10	CAP6_QEPI2 (I)	GPIOB10	R/W-0	Type 1
11	TDIRB (I)	GPIOB11	R/W-0	Type 1
12	TCLKINB (I)	GPIOB12	R/W-0	Type 1
13	$\overline{\text{C4TRIP}}$ (I)	GPIOB13	R/W-0	Type 1
14	$\overline{\text{C5TRIP}}$ (I)	GPIOB14	R/W-0	Type 1
15	$\overline{\text{C6TRIP}}$ (I)	GPIOB15	R/W-0	Type 1

Note: GPBMUX and GPBDIR are EALLOW−protected registers.

Figure 4−6. GPIO B Input Qualification Control (GPBQUAL) Register

15	8	7	0
Reserved		QUALPRD	
R-0		R/W-0	

Legend: R = Read access, W = write access, -0 = value after reset

Note: EALLOW-protected register

Table 4−6. GPIO B Input Qualification Control (GPBQUAL) Register Field Descriptions

Bits	Field	Description
15−8	Reserved	
7−0	QUALPRD	Specifies the qualification sampling period: 0x00 No qualification (just SYNC to SYSCLKOUT) 0x01 QUALPRD = 2 SYSCLKOUT cycles 0x02 QUALPRD = 4 SYSCLKOUT cycles … 0xFF QUALPRD = 510 SYSCLKOUT cycles

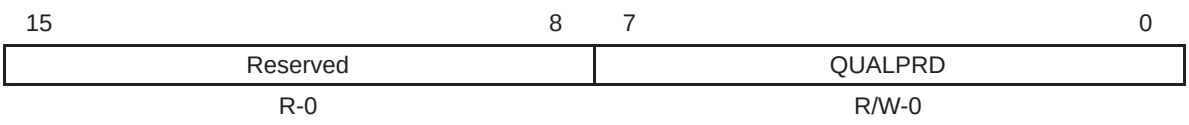
Table 4−7 shows the MUX, direction, set, clear, and toggle register bit to I/O mapping for GPIO port D. Unused I/Os are reserved for future use.

Table 4−7. GPIO D Register Bit to I/O Mapping

Register Bit	Peripheral Name GPDMUX Bit = 1	GPIO Name GPDMUX Bit = 0	GPDMUX/DIR Type	Input Qual
EV-A Peripheral				
0	$\overline{\text{T1CTRIP\_PDPINTA}}$ (I)	GPIOD0	R/W-0	Type 1
1	$\overline{\text{T2CTRIP}}$ (I)	GPIOD1	R/W-0	Type 1
2	Reserved	Reserved	R-0	−
3	Reserved	Reserved	R-0	−
4	Reserved	Reserved	R-0	−
EV-B Peripheral				
5	$\overline{\text{T3CTRIP\_PDPINTB}}$ (I)	GPIOD5	R/W-0	Type 1
6	$\overline{\text{T4CTRIP}}$ (I)	GPIOD6	R/W-0	Type 1
7	Reserved	Reserved	R-0	−
8	Reserved	Reserved	R-0	−
9	Reserved	Reserved	R-0	−
10	Reserved	Reserved	R-0	−
11	Reserved	Reserved	R-0	−
12	Reserved	Reserved	R-0	−
13	Reserved	Reserved	R-0	−
14	Reserved	Reserved	R-0	−
15	Reserved	Reserved	R-0	−

Note: GPDMUX and GPDDIR are EALLOW−protected registers.

Figure 4−7. GPIO D Input Qualification Control (GPDQUAL) Register



Legend: R = Read access, W = write access, -0 = value after reset

Note: EALLOW-protected register

Table 4−8. GPIO D Input Qualification Control (GPDQUAL) Register Field Descriptions

Bits	Field	Description
15−8	Reserved	
7−0	QUALPRD	Specifies the qualification sampling period: 0x00 No qualification (just SYNC to SYSCLKOUT) 0x01 QUALPRD = 2 SYSCLKOUT cycles 0x02 QUALPRD = 4 SYSCLKOUT cycles . 0xFF QUALPRD = 510 SYSCLKOUT cycles

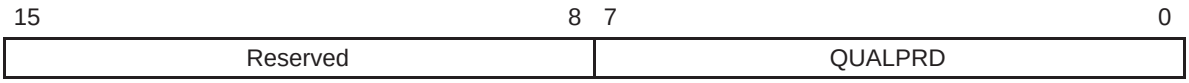
Table 4−9 shows the MUX, direction, set, clear, and toggle register bit to I/O mapping for GPIO port E. Unused I/Os are reserved for future use.

Table 4−9. GPIO E MUX Register Bit to I/O Mapping

Register Bit	Peripheral Name GPEMUX Bit = 0	GPIO Name GPEMUX Bit = 0	GPEMUXDIR Type	Input Qual
Interrupts				
0	XINT1_$\overline{\text{XBIO}}$ (I)	GPIOE0	R/W-0	Type 1
1	XINT2_ADCSOC (I)	GPIOE1	R/W-0	Type 1
2	XNMI_XINT13 (I)	GPIOE2	R/W-0	Type 1
3	Reserved	Reserved	R-0	−
4	Reserved	Reserved	R-0	−
5	Reserved	Reserved	R-0	−
6	Reserved	Reserved	R-0	−
7	Reserved	Reserved	R-0	−
8	Reserved	Reserved	R-0	−
9	Reserved	Reserved	R-0	−
10	Reserved	Reserved	R-0	−
11	Reserved	Reserved	R-0	−
12	Reserved	Reserved	R-0	−
13	Reserved	Reserved	R-0	−
14	Reserved	Reserved	R-0	−
15	Reserved	Reserved	R-0	−

Note: GPEMUX and GPEDIR are EALLOW-protected registers.

Figure 4−8. GPIO E Input Qualification Control (GPEQUAL) Register



Legend: R = Read access, W = write access, -0 = value after reset

Note: EALLOW-protected register

Table 4−10. GPIO E Input Qualification Control (GPEQUAL) Register Field Descriptions

Bits	Field	Description
15−8	Reserved	
7−0	QUALPRD	Specifies the qualification sampling period: 0x00 No qualification (just SYNC to SYSCLKOUT) 0x01 QUALPRD = SYSCLKOUT/2 0x02 QUALPRD = SYSCLKOUT/4 . 0xFF QUALPRD = SYSCLKOUT/510

Table 4−11 shows the MUX, direction, set, clear, and toggle register bit to I/O mapping for GPIO port E. Unused I/Os are reserved for future use.

Table 4−11. GPIO F Register to Bit I/O Mapping

Register Bit	Peripheral Name GPFMUX Bit = 1	GPIO Name GPFMUX Bit = 0	GPFMUX/DIR Type	Input Qual
SPI Peripheral				
0	SPISIMO (O)	GPIOF0	R/W-0	Type 2
1	SPISOMI (I)	GPIOF1	R/W-0	Type 2
2	SPICLK (I/O)	GPIOF2	R/W-0	Type 2
3	SPISTE (I/O)	GPIOF3	R/W-0	Type 2
SCIA Peripheral				
4	SCITXDA (O)	GPIOF4	R/W-0	Type 2
5	SCIRXDA (I)	GPIOF5	R/W-0	Type 2
CAN Peripheral				
6	CANTX (O)	GPIOF6	R/W-0	Type 2
7	CANRX (I)	GPIOF7	R/W-0	Type 2

Note: GPFMUX and GPFDIR are EALLOW-protected registers.

Table 4−11. GPIO F Register to Bit I/O Mapping (Continued)(Continued)

Register Bit	GPIO Name GPFMUX Bit = 1	GPFMUX Bit = 0	Type	Input Qual
McBSP Peripheral				
8	MCLKX (I/O)	GPIOF8	R/W-0	Type 2
9	MCLKR (I/O)	GPIOF9	R/W-0	Type 2
10	MFSX (I/O)	GPIOF10	R/W-0	Type 2
11	MFSR (I/O)	GPIOF11	R/W-0	Type 2
12	MDX (O)	GPIOF12	R/W-0	Type 2
13	MDR (I)	GPIOF13	R/W-0	Type 2
XF CPU Output Signal				
14	XF (0)	GPIOF14	R/W-0	Type 2
15	Reserved	Reserved	R-0	−

Note: GPFMUX and GPFDIR are EALLOW-protected registers.

Table 4−12 shows the MUX, direction, set, clear, and toggle register bit to I/O mapping for GPIO port G. Unused I/Os are reserved for future use.

Table 4−12. GPIO G Register to Bit I/O Mapping

Register Bit	Peripheral Name GPGMUX Bit = 1	GPIO Name GPGMUX Bit = 0	GPGMUX/DIR Type	Input Qual
0	Reserved	Reserved	R-0	−
1	Reserved	Reserved	R-0	−
2	Reserved	Reserved	R-0	−
3	Reserved	Reserved	R-0	−
SCI-B Peripheral				
4	SCITXDB (O)	GPIOG4	R/W-0	Type 2
5	SCIRXDB (I)	GPIOG5	R/W-0	Type 2
6	Reserved	Reserved	R-0	−
7	Reserved	Reserved	R-0	−
8	Reserved	Reserved	R-0	−

Note: GPGMUX and GPGDIR are EALLOW-protected registers.

Table 4−12. GPIO G Register to Bit I/O Mapping (Continued)

Register Bit	Peripheral Name GPGMUX Bit = 1	GPIO Name GPGMUX Bit = 0	GPGMUX/DIR Type	Input Qual
9	Reserved	Reserved	R-0	−
10	Reserved	Reserved	R-0	−
11	Reserved	Reserved	R-0	−
12	Reserved	Reserved	R-0	−
13	Reserved	Reserved	R-0	−
14	Reserved	Reserved	R-0	−
15	Reserved	Reserved	R-0	−

Note: GPGMUX and GPGDIR are EALLOW-protected registers.

Chapter 5

Peripheral Frames

This chapter describes the peripheral frames. It also describes the device emulation registers.

Topic **Page**

5.1 Peripheral Frame Registers

The 281x devices contain three peripheral register spaces. The spaces are categorized as follows:

- Peripheral Frame 0: These are peripherals that are mapped directly to the CPU memory bus. See Table 5−1.
- Peripheral Frame 1: These are peripherals that are mapped to the 32-bit peripheral bus. See Table 5−2.
- Peripheral Frame 2: These are peripherals that are mapped to the 16-bit peripheral bus. See Table 5−3.

Table 5−1. Peripheral Frame 0 Registers

Name	Address Range	Size (x16)	Access Type†
Device Emulation Registers	0x0000 0880 0x0000 09FF	384	EALLOW protected
Reserved	0x0000 0A00 0x0000 0B00	128	
FLASH Registers‡	0x0000 0A80 0x0000 0ADF	96	EALLOW protected CSM Protected
Code Security Module Registers	0x0000 0AE0 0x0000 0AEF	16	EALLOW protected
Reserved	0x0000 0AF0 0x0000 0B1F	48	
XINTF Registers	0x0000 0B20 0x0000 0B3F	32	Not EALLOW protected
Reserved	0x0000 0B40 0x0000 0BFF	192	
CPU-TIMER0/1/2 Registers	0x0000 0C00 0x0000 0C3F	64	Not EALLOW protected
Reserved	0x0000 0C40 0x0000 0CDF	160	
PIE Registers	0x0000 0CE0 0x0000 0CFF	32	Not EALLOW protected

† If registers are EALLOW protected, you cannot perform writes until you execute the EALLOW instruction. The EDIS instruction disables writes to prevent stray code or pointers from corrupting register contents.
‡ The flash registers are also protected by the Code Security Module (CSM).

Table 5−1. Peripheral Frame 0 Registers (Continued)

Name	Address Range	Size (x16)	Access Type†
PIE Vector Table	0x0000 0D00 0x0000 0DFF	256	EALLOW protected
Reserved	0x0000 0E00 0x0000 0FFF	512	

† If registers are EALLOW protected, you cannot perform writes until you execute the EALLOW instruction. The EDIS instruction disables writes to prevent stray code or pointers from corrupting register contents.
‡ The flash registers are also protected by the Code Security Module (CSM).

Table 5−2. Peripheral Frame 1 Registers

Name†	Address Range	Size (x16)	Access Type
eCAN Registers	0x00 6000 0x00 60FF	256 (128 x 32)	Some eCAN control registers (and selected bits in other eCAN control registers) are EALLOW-protected. See Section 5.2 on page 5-5.
eCAN Mailbox RAM	0x00 6100 0x00 61FF	256 (128 x 32)	Not EALLOW-protected
Reserved	0x00 6200 0x00 6FFF	3584	

† Peripheral Frame 1 allows 16-bit and 32-bit accesses. All 32-bit accesses are aligned to even address boundaries.

Table 5−3. Peripheral Frame 2 Registers†

Name	Address Range	Size (x16)	Access Type
Reserved	0x0000 7000 0x0000 700F	16	
System Control Registers	0x0000 7010 0x0000 702F	32	EALLOW Protected
Reserved	0x0000 7030 0x0000 703F	16	
SPI Registers	0x0000 7040 0x0000 704F	16	Not EALLOW Protected
SCI-A Registers	0x0000 7050 0x0000 705F	16	Not EALLOW Protected
Reserved	0x0000 7060 0x0000 706F	16	

† Peripheral Frame 2 only allows 16-bit accesses. All 32-bit accesses are ignored (invalid data can be returned or written).

Table 5−3. Peripheral Frame 2 Registers† (Continued)

Name	Address Range	Size (x16)	Access Type
External Interrupt Registers	0x0000 7070 0x0000 707F	16	Not EALLOW Protected
Reserved	0x0000 7080 0x0000 70BF	64	
GPIO MUX Registers	0x0000 70C0 0x0000 70DF	32	EALLOW Protected
GPIO Data Registers	0x0000 70E0 0x0000 70FF	32	Not EALLOW Protected
ADC Registers	0x0000 7100 0x0000 711F	32	Not EALLOW Protected
Reserved	0x0000 7120 0x0000 73FF	736	
EV-A Registers	0x0000 7400 0x0000 743F	64	Not EALLOW Protected
Reserved	0x0000 7440 0x0000 74FF	192	
EV-B Registers	0x0000 7500 0x0000 753F	64	Not EALLOW Protected
Reserved	0x0000 7540 0x0000 774F	528	
SCI-B Registers	0x0000 7750 0x0000 775F	16	Not EALLOW Protected
Reserved	0x0000 7760 0x0000 77FF	160	
McBSP Registers	0x0000 7800 0x0000 783F	64	Not EALLOW Protected
Reserved	0x0000 7840 0x0000 7FFF	1984	

† Peripheral Frame 2 only allows 16-bit accesses. All 32-bit accesses are ignored (invalid data can be returned or written).

5.2 EALLOW Protected Registers

Several control registers on the 281x devices are protected from spurious CPU writes by the EALLOW protection mechanism. The EALLOW bit in status register 1 (ST1) indicates if the state of protection as shown in Table 5−4.

Table 5−4. Access to EALLOW Protected Registers

EALLOW Bit	CPU Writes	CPU Reads	JTAG Writes	JTAG Reads
0	Ignored	Allowed	Allowed†	Allowed
1	Allowed	Allowed	Allowed	Allowed

† The EALLOW bit is overridden via the JTAG port, allowing full access of protected registers during debug from the Code Composer Studio interface.

At reset the EALLOW bit is cleared enabling EALLOW protection. While protected, all writes to protected registers by the CPU are ignored and only CPU reads, JTAG reads, and JTAG writes are allowed. If this bit is set, by executing the EALLOW instruction, then the CPU is allowed to write freely to protected registers. After modifying registers, they can once again be protected by executing the EDI instruction to clear the EALLOW bit.

The following registers are EALLOW protected:

- Device Emulation Registers
- Flash Registers
- CSM Registers
- PIE Vector Table
- System Control Registers
- GPIO MUX Registers
- Certain eCAN Registers

Table 5−5. EALLOW Protected Device Emulation Registers

Name	Address Range	Size (x16)	Description
DEVICECNF	0x0000 0880 0x0000 0881	2	Device Configuration Register
PROTSTART	0x0000 0884	1	Block Protection Start Address Register
PROTRANGE	0x0000 0885	1	Block Protection Range Address Register

Table 5−6. EALLOW Protected Flash/OTP Configuration Registers

Name	Address	Size (x16)	Description
		Configuration Registers	
FOPT	0x0000−0A80	1	Flash Option Register
FPWR	0x0000−0A82	1	Flash Power Modes Register
FSTATUS	0x0000−0A83	1	Status Register
FSTDBYWAIT	0x0000−0A84	1	Flash Sleep To Standby Wait State Register
FACTIVEWAIT	0x0000−0A85	1	Flash Standby To Active Wait State Register
FBANKWAIT	0x0000−0A86	1	Flash Read Access Wait State Register
FOTPWAIT	0x0000−0A87	1	OTP Read Access Wait State Register

Table 5−7. EALLOW Protected Code Security Module (CSM) Registers

Register Name	Memory Address	Reset Values	Register Description
KEY Registers − Accessible by the User			
KEY0	0x0000 − 0AE0	0xFFFF	Low word of the 128-bit KEY register
KEY1	0x0000 − 0AE1	0xFFFF	Second word of the 128-bit KEY register
KEY2	0x0000 − 0AE2	0xFFFF	Third word of the 128-bit KEY register
KEY3	0x0000 − 0AE3	0xFFFF	Fourth word of the 128-bit KEY register
KEY4	0x0000 − 0AE4	0xFFFF	Fifth word of the 128-bit KEY register
KEY5	0x0000 − 0AE5	0xFFFF	Sixth word of the 128-bit KEY register
KEY6	0x0000 − 0AE6	0xFFFF	Seventh word of the 128-bit KEY register
KEY7	0x0000 − 0AE7	0xFFFF	High word of the 128-bit KEY register
CSMSCR	0x0000 − 0AEF		CSM status and control register

Table 5−8. EALLOW Protected PIE Vector Table

Name	Address	Size (x16)	Description	CPU Priority	PIE Group Priority
Not used	0x0000 0D00	2	Reserved	−	−
	0x0000 0D02				
	0x0000 0D04				
	0x0000 0D06				
	0x0000 0D08				
	0x0000 0D0A				
	0x0000 0D0C				
	0x0000 0D0E				
	0x0000 0D10				
	0x0000 0D12				
	0x0000 0D14				
	0x0000 0D16				
	0x0000 0D18				
INT13	0x0000 0D1A	2	External Interrupt 13 (XINT13) or CPU-Timer 1 (for RTOS use)	17	−
INT14	0x0000 0D1C	2	CPU-Timer 2 (for RTOS use)	18	−
DATALOG	0x0000 0D1E	2	CPU Data Logging Interrupt	19 (lowest)	−
RTOSINT	0x0000 0D20	2	CPU Real-Time OS Interrupt	4	−
EMUINT	0x0000 0D22	2	CPU Emulation Interrupt	2	−
NMI	0x0000 0D24	2	External Non-Maskable Interrupt	3	−
ILLEGAL	0x0000 0D26	2	Illegal Operation	−	−
USER0	0x0000 0D28	2	User-Defined Trap	−	−
.	.	.	.	.	.
USER11	0x0000 0D3E	2	User-Defined Trap	−	−
INT1.1	0x0000 0D40	2	Group 1 Interrupt Vectors	5	1 (highest)
.	.	.			.
INT1.8	0x0000 0D4E	2			8 (lowest)
.	.	.	Group 2 Interrupt Vectors	6	
.	.	.	to Group 11 Interrupt Vectors	to	
.	.	.		15	
INT12.1	0x0000 0DF0	2	Group 12 Interrupt Vectors	16	1 (highest)
.	.	.			.
INT12.8	0x0000 0DFE	2			8 (lowest)

Table 5−9. EALLOW Protected PLL, Clocking, Watchdog, and Low-Power Mode Registers

Name	Address	Size (x16)	Description
HISPCP	0x0000 701A	1	High-Speed Peripheral Clock Prescaler Register for HSPCLK Clock
LOSPCP	0x0000 701B	1	Low-Speed Peripheral Clock Prescaler Register for HSPCLK Clock
PCLKCR	0x0000 701C	1	Peripheral Clock Control Register
LPMCR0	0x0000 701E	1	Low Power Mode Control Register 0
LPMCR1	0x0000 701F	1	Low Power Mode Control Register 1
PLLCR	0x0000 7021	1	PLL Control Register‡
SCSR	0x0000 7022	1	System Control and Status Register
WDCNTR	0x0000 7023	1	Watchdog Counter Register
WDKEY	0x0000 7025	1	Watchdog Reset Key Register
WDCR	0x0000 7029	1	Watchdog Control Register

Table 5−10. EALLOW Protected GPIO MUX Registers

Name	Address	Size (x16)	Register Description
GPAMUX	0x0000 70C0	1	GPIO A MUX Control Register
GPADIR	0x0000 70C1	1	GPIO A Direction Control Register
GPAQUAL	0x0000 70C2	1	GPIO A Input Qualification Control Register
GPBMUX	0x0000 70C4	1	GPIO B MUX Control Register
GPBDIR	0x0000 70C5	1	GPIO B Direction Control Register
GPBQUAL	0x0000 70C6	1	GPIO B Input Qualification Control Register
GPDMUX	0x0000 70CC	1	GPIO D MUX Control Register
GPDDIR	0x0000 70CD	1	GPIO D Direction Control Register
GPDQUAL	0x0000 70CE	1	GPIO D Input Qualification Control Register
GPEMUX	0x0000 70D0	1	GPIO E MUX Control Register
GPEDIR	0x0000 70D1	1	GPIO E Direction Control Register
GPEQUAL	0x0000 70D2	1	GPIO E Input Qualification Control Register

Table 5−10. EALLOW Protected GPIO MUX Registers(Continued)

Name	Address	Size (x16)	Register Description
GPFMUX	0x0000 70D4	1	GPIO F MUX Control Register
GPFDIR	0x0000 70D5	1	GPIO F Direction Control Register
GPGMUX	0x0000 70D8	1	GPIO G MUX Control Register
GPGDIR	0x0000 70D9	1	GPIO G Direction Control Register

Table 5−11. EALLOW Protected eCAN Registers

Name	Address	Size (x16)	Description
CANMC	0x0000−6014	2	Master Control Register†
CANBTC	0x0000−6016	2	Bit Timing Configuration Register‡
CANGIM	0x0000−6020	2	Global Interrupt Mask Register§
CANMIM	0x0000−6024	2	Mailbox Interrupt Mask Register
CANTSC	0x0000−602E	2	Time Stamp Counter
CANTIOC	0x0000−602A	1	I/O Control Register for CANTXA Pin¶
CANRIOC	0x0000−602C	1	I/O Control Register for CANRXA Pin#

† Only bits CANMC[15:9] and [7:6] are protected
‡ Only bits BCR[23:16] and [10:0] are protected
§ Only bits CANGIM[17:16] , [14:8], and [2:0] are protected
¶ Only IOCONT1[3] is protected
# Only IOCONT2[3] is protected

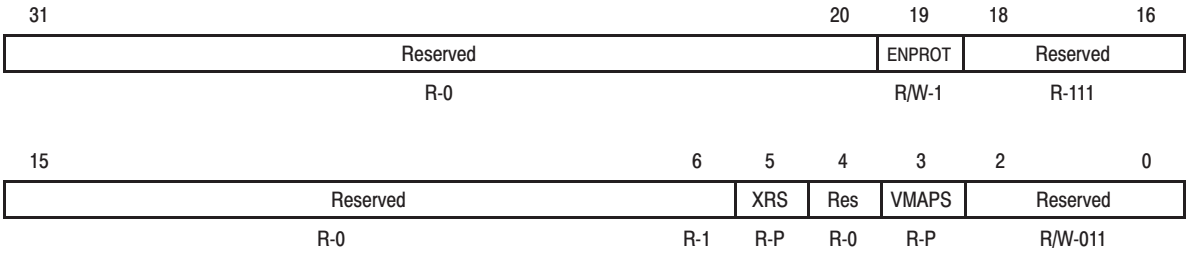
5.3 Device Emulation Registers

These registers are used to control the protection mode of the C28x CPU and to monitor some critical device signals. The registers are defined in Table 5−12.

Table 5−12. Device Emulation Registers

Name	Address Range	Size (x16)	Description
DEVICECNF	0x0000 0880 0x0000 0881	2	Device Configuration Register
Reserved	0x0000 0882	1	
DEVICEID	0x0000 0883	2	Device ID Register
PROTSTART	0x0000 0884	1	Block Protection Start Address Register
PROTRANGE	0x0000 0885	1	Block Protection Range Address Register
Reserved	0x0000 0886 0x0000 09FF	378	

Figure 5−1. Device Configuration (DEVICECNF) Register



Legend: R = Read, W = Write, P = pin value after reset, -n = reset value

Note: EALLOW-protected register

Table 5−13. Device Configuration (DEVICECNF) Register Field Descriptions

Bits	Field	Description
31−20	Reserved	
19	ENPROT	Enable Write-Read Protection Mode Bit
		0 Disables write-read protection mode
		1 Enables write-read protection as specified by the PROTSTART and PROTRANGE registers

Table 5−13. Device Configuration (DEVICECNF) Register Field Descriptions (Continued)

Bits	Field	Description
18−6	Reserved	
5	XRS	Reset Input Signal Status. This is connected directly to the $\overline{\text{XRS}}$ input pin.
4	Reserved	
3	VMAPS	VMAP Configure Status. This indicates the status of VMAP.
2−0	Reserved	

Figure 5−2. DEVICEID Register

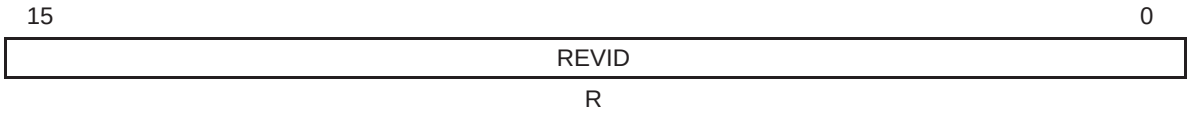


Table 5−14. DEVICEID Register Field Descriptions

Bits	Field	Description
15−0	REVID	These 16 bits specify the silicon revision number for the particular part. This number always starts with 0x0000 on the first revision of the silicon and is incremented on any subsequent revisions. 0x0000 Revision 0 (for first silicon) 0x0001 Revision A 0x0002 Revision B and so forth

5.4 Write-Followed-by-Read Protection

The PROTSTART and PROTRANGE registers set the memory address range for which CPU “write” followed by “read” operations are protected (operations occur in sequence rather then in their natural pipeline order). This is necessary protection for certain peripheral operations.

Example: The following lines of code perform a write to register 1 (REG1) location and then the next instruction performs a read from Register 2 (REG2) location. On the processor memory bus, with block protection disabled, the read operation is issued before the write as shown:

```
MOV    @REG1,AL      -----  +
TBIT   @REG2,#BIT_X  -----  |----> Read
                     +----> Write
```

If block protection is enabled, then the read is stalled until the write occurs as shown:

```
MOV    @REG1,AL      -----  +
TBIT   @REG2,#BIT_X  ---    +      |
                     |      +----> Write
                     +------> Read
```

NOTE: The C28x CPU automatically protects writes followed by reads to the same memory address. The protection mechanism described above is for cases where the address is not the same, but within a given region in memory (as defined by the PROTSTART and PROTRANGE registers).

Table 5−15. PROTSTART and PROTRANGE Registers

Name	Address	Size	Type	Reset	Description
PROTSTART	0x0000 0884	16	R/W	0x0100†	The PROTSTART register sets the starting address relative to the 16 most significant bits of the processors lower 22-bit address reach. Hence, the smallest resolution is 64 words.
PROTRANGE	0x0000 0885	16	R/W	0x00FF†	The PROTRANGE register sets the block size (from the starting address), starting with 64 words and incrementing by binary multiples (64, 128, 256, 512, 1K, 2K, 4K, 8K, 16K, ..., 2M).

† The default values of these registers on reset are selected to cover the Peripheral Frame 1, Peripheral Frame 2, and XINTF Zone 1 areas of the memory map (address range 0x0000 4000 to 0x0000 8000).

Table 5−16. PROTSTART Valid Values†

Start Address	Register Value	Register Bits															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0000 0000	0x0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0000 0040	0x0001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0x0000 0080	0x0002	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0x0000 00C0	0x0003	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
0x003F FF00	0xFFFC	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0x003F FF40	0xFFFD	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
0x003F FF80	0xFFFE	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0x003F FFC0	0xFFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

† The quickest way to calculate register value is to divide the desired block starting address by 64.

Table 5−17. PROTRANGE Valid Values‡

Block Size	Register Value	Register Bits															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
64	0x0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
128	0x0001	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
256	0x0003	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
512	0x0007	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
1K	0x000F	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
2K	0x001F	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
4K	0x003F	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
8K	0x007F	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
16K	0x00FF	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
32K	0x01FF	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
64K	0x03FF	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
128K	0x07FF	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
256K	0x0FFF	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
512K	0x1FFF	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1M	0x3FFF	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2M	0x7FFF	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4M	0xFFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

‡ Not all register values are valid. The PROTSTART address value must be a multiple of the range value. For example: if the block size is set to 4K, then the start address can only be at any 4K boundary.

Chapter 6

Peripheral Interrupt Expansion (PIE)

The peripheral interrupt expansion (PIE) block multiplexes numerous interrupt sources into a smaller set of interrupt inputs. The PIE block can support 96 individual interrupts that are grouped into blocks of eight. Each group is fed into one of 12 core interrupt lines ($\overline{\text{INT1}}$ to $\overline{\text{INT12}}$). Each of the 96 interrupts is supported by its own vector stored in a dedicated RAM block that you can modify. The CPU, upon servicing the interrupt, automatically fetches the appropriate interrupt vector. It takes nine CPU clock cycles to fetch the vector and save critical CPU registers. Therefore, the CPU can respond quickly to interrupt events. Prioritization of interrupts is controlled in hardware and software. Each individual interrupt can be enabled/disabled within the PIE block.

Topic **Page**

6.1 Overview of the PIE Controller

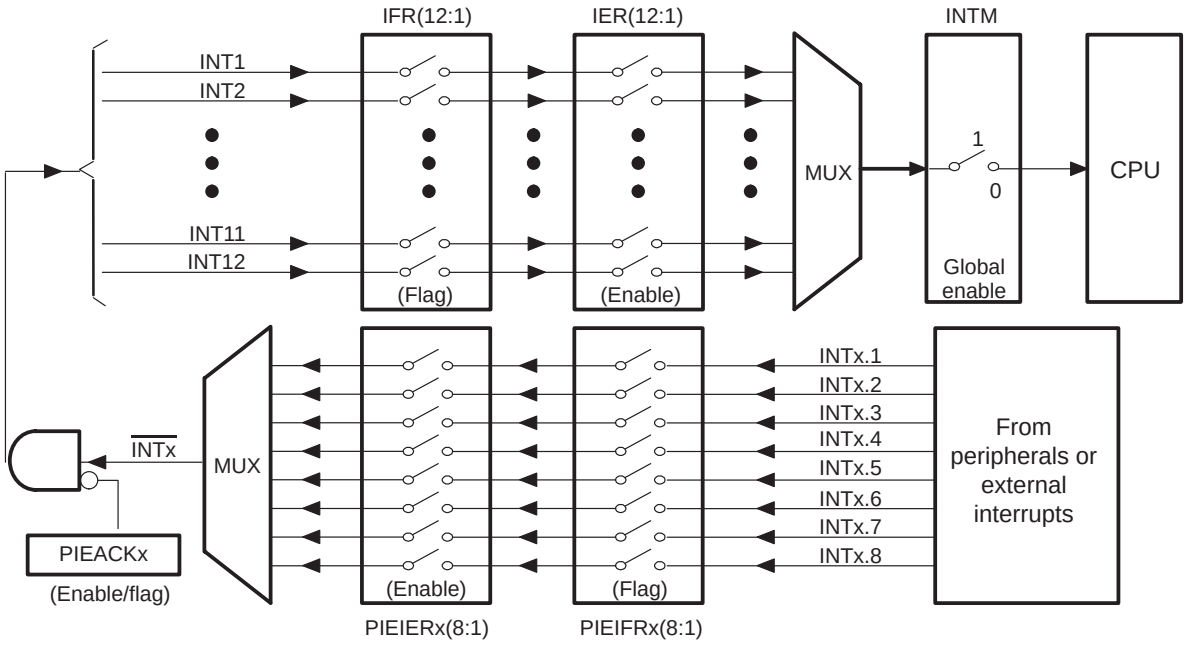
The 28x CPU supports one nonmaskable interrupt (NMI) and 16 maskable prioritized interrupt requests (INT1−INT14, RTOSINT, and DLOGINT) at the CPU level. The 28x devices have many peripherals and each peripheral is capable of generating one or more interrupts in response to many events at the peripheral level. Because the CPU does not have sufficient capacity to handle all peripheral interrupt requests at the CPU level, a centralized peripheral interrupt expansion (PIE) controller is required to arbitrate the interrupt requests from various sources such as peripherals and other external pins.

The PIE vector table is used to store the address (vector) of each interrupt service routine (ISR) within the system. There is one vector per interrupt source including all MUXed and nonMUXed interrupts. You populate the vector table during device initialization and you can update it during operation.

Interrupt Operation Sequence

Figure 6−1 shows an overview of the interrupt operation sequence for all multiplexed PIE interrupts. Interrupt sources that are not multiplexed are fed directly to the CPU.

Figure 6−1. Overview: Multiplexing of Interrupts Using the PIE Block



❑ **Peripheral Level**

An interrupt-generating event occurs in a peripheral. The interrupt flag (IF) bit corresponding to that event is set in a register for that particular peripheral.

If the corresponding interrupt enable (IE) bit is set, the peripheral generates an interrupt request to the PIE controller. If the interrupt is not enabled at the peripheral level, then 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 interrupt request is asserted to the PIE.

Interrupt flags within the peripheral registers must be manually cleared. See the peripheral reference guide for a specific peripheral for more information.

❑ **PIE Level**

The PIE block multiplexes eight peripheral and external pin interrupts into one CPU interrupt. These interrupts are divided into 12 groups: PIE group 1 − PIE group 12. The interrupts within a group are multiplexed into one CPU interrupt. For example, PIE group 1 is multiplexed into CPU interrupt 1 (INT1) while PIE group 12 is multiplexed into CPU interrupt 12 (INT12). Interrupt sources connected to the remaining CPU interrupts are not multiplexed. For the nonmultiplexed interrupts, the PIE passes the request directly to the CPU.

For multiplexed interrupt sources, each interrupt group in the PIE block has an associated flag bit (PIEIFRx.y) and enable bit (PIEIERx.y). In addition, there is one acknowledge bit (PIEACK) for every PIE interrupt group (INT1 to INT12) referred to as PIEACKx. Figure 6−2 illustrates the behavior of the PIE hardware under various PIEIFR and PIEIER register conditions.

Once the request is made to the PIE controller, the corresponding PIE interrupt flag (PIEIFRx.y) bit is set. If the PIE interrupt enable (PIEIERx.y) bit is also set for the given interrupt then the PIE checks the corresponding PIEACKx bit to determine if the CPU is ready for an interrupt from that group. If the PIEACKx bit is clear for that group, then the PIE sends the interrupt request to the CPU. If PIEACKx is set, then the PIE waits until it is cleared to send the request for INTx. See Section 6.3 for details.

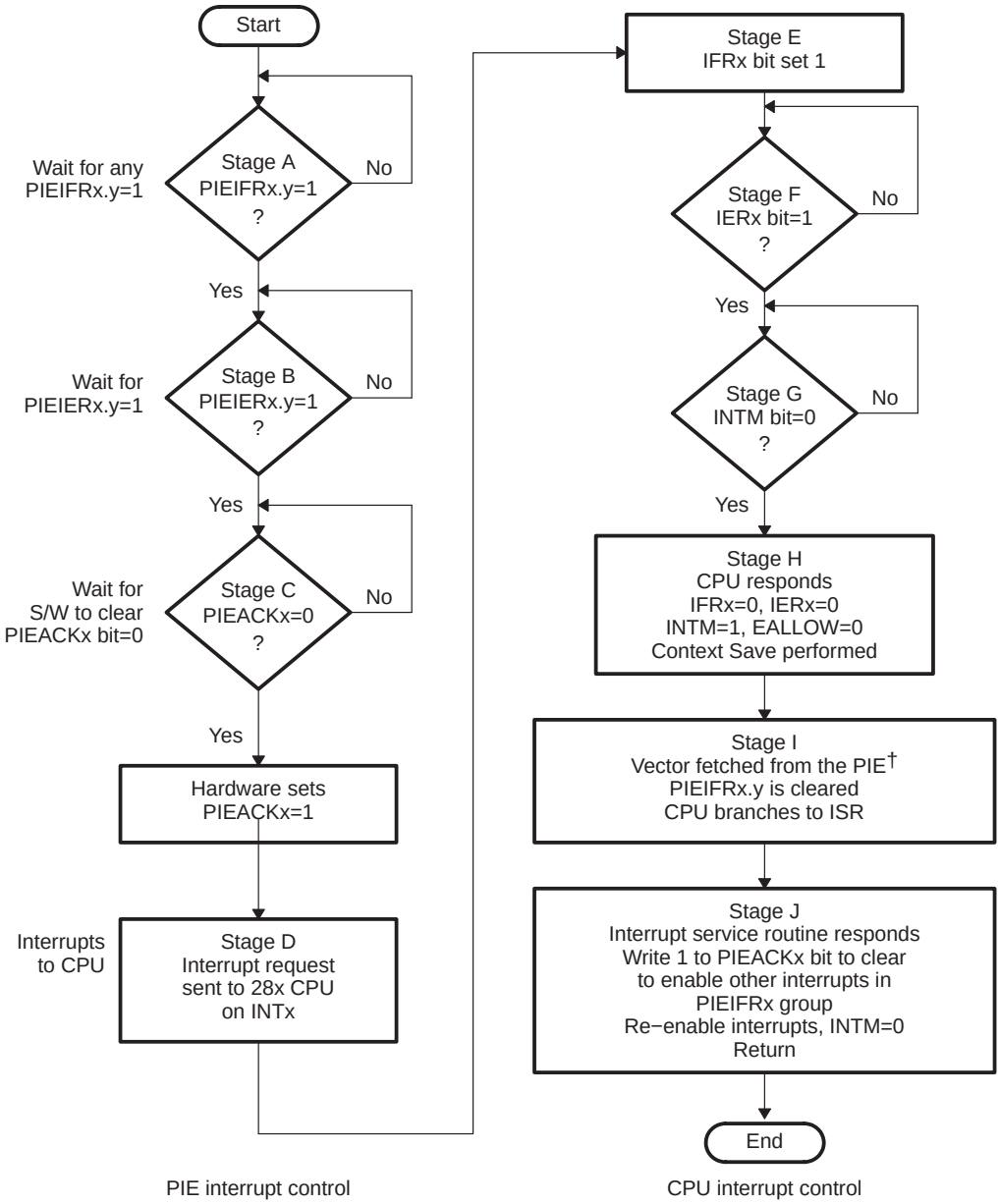
❑ **CPU Level**

Once the request is sent to the CPU, the CPU level interrupt flag (IFR) bit corresponding to INTx is set. After a flag has been latched in the IFR, the corresponding interrupt is not serviced until it is appropriately enabled in the CPU interrupt enable (IER) register or the debug interrupt enable register (DBGIER) and the global interrupt mask (INTM) bit.

As shown in Table 6−1, the requirements for enabling the maskable interrupt at the CPU level depends on the interrupt handling process being used. In the

standard process, which happens most of the time, the DBGIER register is not used. When the 28x is in real-time emulation mode and the CPU is halted, a different process is used. In this special case, the DBGIER is used and the INTM bit is ignored. If the DSP is in real-time mode and the CPU is running, the standard interrupt-handling process applies.

Figure 6−2. Typical PIE/CPU Interrupt Response − INTx.y



† For multiplexed interrupts, the PIE will respond with the highest priority interrupt that is both flagged and enabled. If there is no interrupt that is both flagged and enabled. If there is no interrupt both flagged and enabled, then the highest priority interrupt within the group (INTx.1) is used. See Section 6.3.3 for more details.

Table 6−1. Enabling Interrupt

Interrupt Handling Process	Interrupt Enabled If…
Standard	INTM = 0 and bit in IER is 1
DSP in real-time mode and halted	Bit in IER is 1 and DBGIER is 1

The CPU then prepares to service the interrupt. This preparation process is described in detail in *TMS320C28x DSP CPU and Instruction Set Reference Guide* (literature number SPRU430). In preparation, the corresponding CPU IFR and IER bits are cleared, EALLOW and LOOP are cleared, INTM and DBGM are set, the pipeline is flushed and the return address is stored, and the automatic context save is performed. The vector of the ISR is then fetched from the PIE module. If the interrupt request comes from a multiplexed interrupt, the PIE module uses the group PIEIERx and PIEIFRx registers to decode which interrupt needs to be serviced. This decode process is described in detail in Section 6.3.3.

The address for the interrupt service routine that is executed is fetched directly from the PIE interrupt vector table. There is one 32-bit vector for each of the possible 96 interrupts within the PIE. Interrupt flags within the PIE module (PIEIFRx.y) are automatically cleared when the interrupt vector is fetched. The PIE acknowledge bit for a given interrupt group, however, must be cleared manually when ready to receive more interrupts from the PIE group.

6.2 Vector Table Mapping

On 28xx devices, the interrupt vector table can be mapped to five distinct locations in memory. In practice only the PIE vector table mapping is used for F28xx devices.

This vector mapping is controlled by the following mode bits/signals:

VMAP: VMAP is found in Status Register 1 ST1 (bit 3). A device reset sets this bit to 1. The state of this bit can be modified by writing to ST1 or by SETC/CLRC VMAP instructions. For normal F2810/12 operation leave this bit set.

M0M1MAP: M0M1MAP is found in Status Register 1 ST1 (bit 11). A device reset sets this bit to 1. The state of this bit can be modified by writing to ST1 or by SETC/CLRC M0M1MAP instructions. For normal 28xx device operation, this bit should remain set. M0M1MAP = 0 is reserved for TI testing only.

MP/MC: This bit is found in XINTCNF2 Register (bit 8). On the devices with an external interface (XINTF) the default value of this bit, on reset, is set by the XMP/MC input device signal. On devices without an XINTF, the XMP/MC is tied low internally. The state of this bit can be modified after reset by writing to the XINTCNF2 register (address 0x0000 0B34).

ENPIE: ENPIE is found in PIECTRL Register (bit 0). The default value of this bit, on reset, is set to 0 (PIE disabled). The state of this bit can be modified after reset by writing to the PIECTRL register (address 0x0000 0CE0).

Using these bits and signals the possible vector table mappings are shown in Table 6−2.

Table 6−2. Interrupt Vector Table Mapping†

Vector MAPS	Vectors Fetched From	Address Range	VMAP	M0M1MAP	MP/$\overline{MC}$	ENPIE
M1 Vector‡	M1 SARAM Block	0x000000−0x00003F	0	0	X	X
M0 Vector‡	M0 SARAM Block	0x000000−0x00003F	0	1	X	X
BROM Vector	ROM Block	0x3FFFC0−0x3FFFFF	1	X	0	0
XINTF Vector§	XINTF Zone 7 Block	0x3FFFC0−0x3FFFFF	1	X	1	0
PIE Vector	PIE Block	0x000D00−0x000DFF	1	X	X	1

† On the 281x devices, the VMAP and M0M1MAP modes are set to 1 on reset. The ENPIE mode is forced to 0 on reset.
‡ Vector map MO and M1 Vector is a reserved mode only. On the28x devices these are used as RAM.
§ Valid on F2812, C2812, and R2812 devices only

The M1 and M0 vector table mapping is reserved for TI testing only. When using other vector mappings, the M0 and M1 memory blocks are treated as RAM blocks and can be used freely without any restrictions.

After a device reset operation, the vector table is mapped as shown in Table 6−3.

Table 6−3. Vector Table Mapping After Reset Operation†

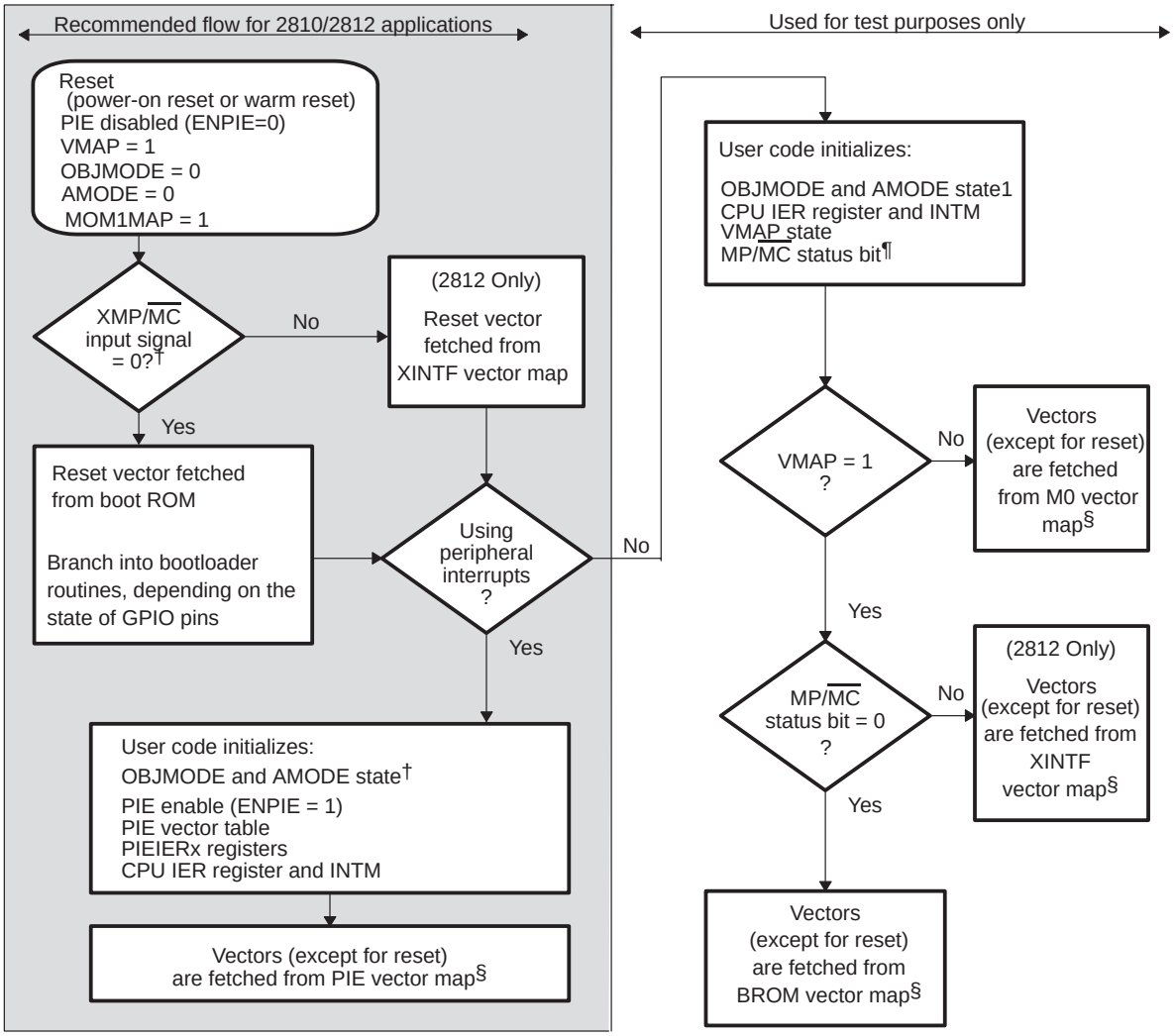
VECTOR MAPS	RESET FETCHED FROM	ADDRESS RANGE	VMAP	M0M1MAP	MP/$\overline{MC}$	ENPIE
BROM Vector	ROM Block	0x3FFFC0−0x3FFFFF	1	1	0	0
XINTF Vector§	XINTF Zone 7 Block	0x3FFFC0−0x3FFFFF	1	1	1	0

† On the 28x devices, the VMAP and M0M1MAP modes are set to 1 on reset. The ENPIE mode is forced to 0 on reset.
§ Valid on F2812, C2812, and R2812 devices only

After the reset and boot is complete, the PIE vector table should be initialized by the user's code. Then the application enables the PIE vector table. From that point on the interrupt vectors are fetched from the PIE vector table. Note: when a reset occurs, the reset vector is always fetched from the vector table as shown in Table 6−3. After a reset the PIE vector table is always disabled.

Figure 6−3 illustrates the process by which the vector table mapping is selected.

Figure 6−3. Reset Flow Diagram



† The XMP/$\overline{MC}$ input signal is tied low internally on the F2810.
‡ The compatibility operating mode of the F2810 and F2812 is determined by a combination of the OBJMODE and AMODE bits in Status Register 1 (ST1):

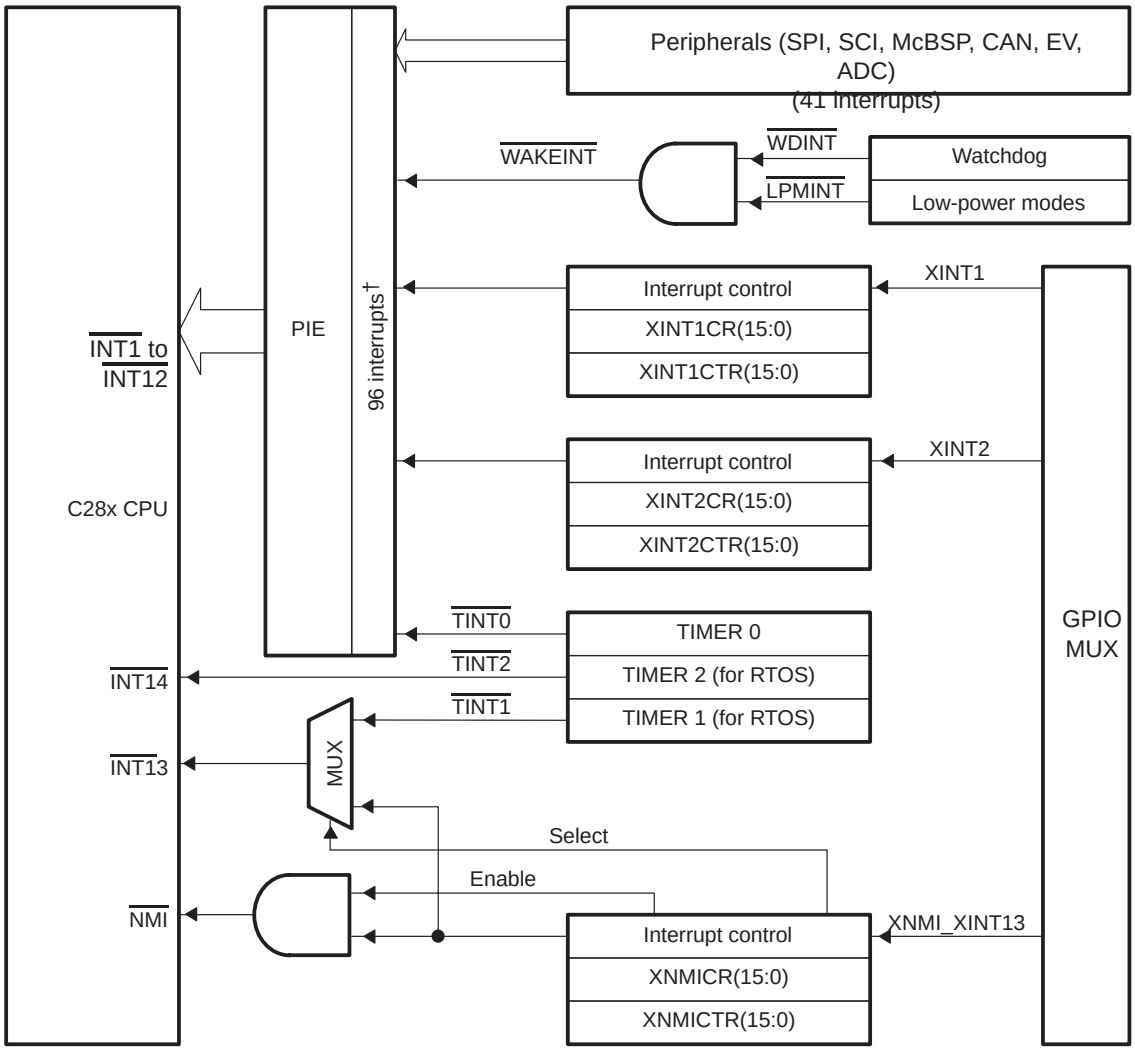
Operating Mode	OBJMODE	AMODE	
C28x Mode	1	0	
C2xLP Source-Compatible	1	1	
C27x Object-Compatible	0	0	(Default at reset)

§ The reset vector is always fetched from either the BROM or XINTF vector map depending on the XMP/$\overline{MC}$ input signal.
¶ The state of the XMP/$\overline{MC}$ signal is latched into the MP/$\overline{MC}$ bit at reset, it can then be modified by software.

6.3 Interrupt Sources

Figure 6−4 shows how the various interrupt sources are multiplexed within the F2810 and F2812 devices. This MUXing scheme may not be exactly the same on all F28xx devices. See the data sheet of your particular device for details.

Figure 6−4. Interrupt Sources



† Out of a possible 96 interrupts, 45 are currently used by peripherals.

Notes: 1) In the GPIO MUX, the XINT1, XINT2 and XNMI signals are synchronized and optionally qualified by a user programmable number of clock cycles. This filters out glitches from the input source. See the GPIO MUX section for more details.

2) The WAKEINT input must be synchronized (using SYSCLKOUT of the processor) before being fed to the PIE block.

6.3.1 Procedure for Handling Multiplexed Interrupts

The PIE module multiplexes eight peripheral and external pin interrupts into one CPU interrupt. These interrupts are divided into 12 groups: PIE group 1 − PIE group 12. Each group has an associated enable PIEIER and flag PIEIFR register. These registers are used to control the flow of interrupts to the CPU. The PIE module also uses the PIEIER and PIEIFR registers to decode to which interrupt service routine the CPU should branch.

There are three main rules that should be followed when clearing bits within the PIEIFR and the PIEIER registers:

1) **Never clear a PIEIFR bit**. An incoming interrupt may be lost while the read−modify−write operation takes place. To clear a PIEIFR bit, the pending interrupt must be serviced. If you want to clear the PIEIFR bit without executing the normal service routine, then use the following procedure:

Step 1: Set the EALLOW bit to allow modification to the PIE vector table.

Step 2: Modify the PIE vector table so that the vector for the peripheral's service routine points to a temporary ISR. This temporary ISR will only perform a return from interrupt (IRET) operation.

Step 3: Enable the interrupt so that the interrupt will be serviced by the temporary ISR.

Step 4: After the temporary interrupt routine is serviced, the PIEIFR bit will be clear

Step 5: Modify the PIE vector table to re−map the peripheral's service routine to the proper service routine.

Step 6: Clear the EALLOW bit.

The CPU IFR register is integrated within the CPU. Because of this clearing a bit within the CPU IFR register can be done without concern for losing an incoming interrupt.

2) **Software-prioritizing interrupts.** Use the method found in *C28x Peripheral Examples in C* (literature number SPRC097).

   Use the CPU IER register as a global priority and the individual PIEIER registers for group priorities. In this case the PIEIER register is only modified within an interrupt. In addition, only the PIEIER for the same group as the interrupt being serviced is modified. This modification is done while the PIEACK bit holds additional interrupts back from the CPU.

Never disable a PIEIER bit for a group when servicing an interrupt from an unrelated group.

3) **Disabling interrupts using PIEIER**. If the PIEIER registers are used to enable and then later disable an interrupt then the procedure described in Section 6.3.2 must be followed.

6.3.2 Procedures for Enabling And Disabling Multiplexed Peripheral Interrupts

The proper procedure for enabling or disabling an interrupt is by using the peripheral interrupt enable/disable flags. The primary purpose of the PIEIER and CPU IER registers is for software prioritization of interrupts within the same interrupt group. The software *package C28x Peripheral Examples in C* (literature number SPRC097) includes an example showing this method of software prioritizing interrupts. Should bits within the PIEIER registers need to be cleared outside of this context, one of the following two procedures should be followed:

1) **Use the PIEIERx register to disable the interrupt and preserve the associated PIEIFRx flags.**

To clear bits within a PIEIERx register while preserving the associated flags in the PIEIFRx register the following procedure should be followed:

Step 1: Disable global interrupts (INTM = 1)

Step 2: Clear the PIEIERx.y bit to disable the interrupt for a given peripheral. This can be done for one or more peripherals within the same group.

Step 3: Wait 5 cycles. This delay is required to insure that any interrupt that was incoming to the CPU has been flagged within the CPU IFR register.

Step 4: Clear the CPU IFRx bit for the peripheral group. This is a safe operation on the CPU IFR register.

Step 5: Clear the PIEACKx bit for the peripheral group

Step 6: Enable global interrupts (INTM = 0)

2) **Use the PIEIERx register to disable the interrupt and clear the associated PIEIFRx flags.**

   To perform a software reset of a peripheral interrupt and clear the associated flag in the PIEIFRx register and CPU IFR register, then the following procedure should be followed:

Step 1: Disable global interrupts (INTM = 1)

Step 2: Set the EALLOW bit.

Step 3: Modify the PIE vector table to temporarily map the vector of the specific peripheral interrupt to a empty interrupt service routine (ISR). This empty ISR will only perform a return from interrupt (IRET) instruction. This is the safe way to clear a single PIEIFRx.y bit without losing any interrupts from other peripherals within the group.

Step 4: Disable the peripheral interrupt at the peripheral register.

Step 5: Enable global interrupts (INTM = 0).

Step 6: Wait for any pending interrupt from the peripheral to be serviced by the empty ISR routine.

Step 7: Disable global interrupts (INTM = 1).

Step 8: Modify the PIE vector table to map the peripheral vector back to its original ISR.

Step 9: Clear the EALLOW bit.

Step 10: Disable the PIEIER bit for given peripheral.

Step 11: Clear the IFR bit for given peripheral group (this is safe operation on CPU IFR register).

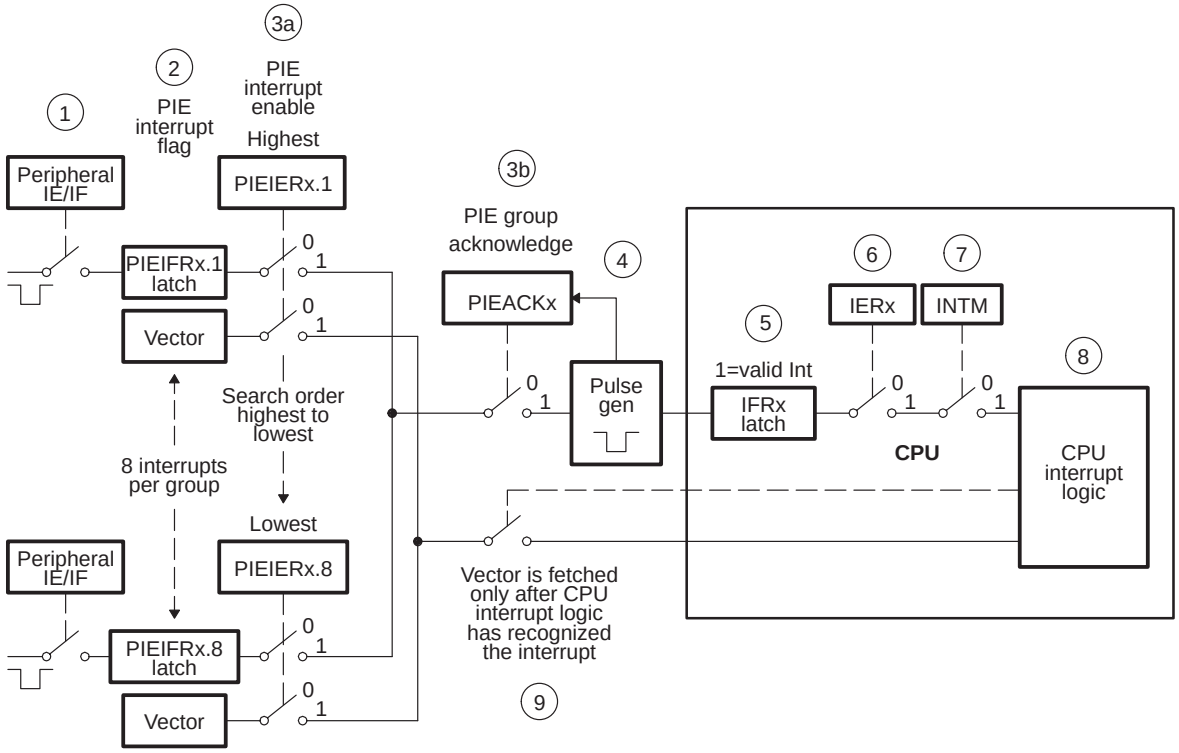
Step 12: Clear the PIEACK bit for the PIE group.

Step 13: Enable global interrupts.

6.3.3 Flow of a Multiplexed Interrupt Request From a Peripheral to the CPU

Figure 6−5 shows the flow with the steps shown in circled numbers. Following the diagram, the steps are described.

Figure 6−5. Multiplexed Interrupt Request Flow Diagram



Step 1: Any peripheral or external interrupt within the PIE group generates an interrupt. If interrupts are enabled within the peripheral module then the interrupt request is sent to the PIE module.

Step 2: The PIE module recognizes that interrupt y within PIE group x (INTx.y) has asserted an interrupt and the appropriate PIE interrupt flag bit is latched: PIEIFRx.y = 1.

Step 3: For the interrupt request to be sent from the PIE to the CPU, both of the following conditions must be true:

1) The proper enable bit must be set (PIEIERx.y = 1) and
2) The PIEACKx bit for the group must be clear.

Step 4: If both conditions in Step 3 are true, then an interrupt request is sent to the CPU and the acknowledge bit is again set (PIEACKx = 1). The

PIEACKx bit will remain set until you clear it to indicate that additional interrupts from the group can be sent from the PIE to the CPU.

Step 5: The CPU interrupt flag bit is set (CPU IFRx = 1) to indicate a pending interrupt x at the CPU level.

Step 6: If the CPU interrupt is enabled (CPU IER bit x = 1, or DBGIER bit x = 1) AND (7) the global interrupt mask is clear (INTM = 0) then the CPU will service the INTx.

Step 7: The CPU recognizes the interrupt and performs the automatic context save, clears the IER bit, sets INTM, and clears EALLOW. All of the steps that the CPU takes in order to prepare to service the interrupt are documented in the *TMS320C28x DSP CPU and Instruction Set Reference Guide* (literature number SPRU430).

Step 8: The CPU will then request the appropriate vector from the PIE.

Step 9: For multiplexed interrupts, the PIE module uses the current value in the PIEIERx and PIEIFRx registers to decode which vector address should be used.

There are two possible cases:

- The vector for the highest priority interrupt within the group that is both a) enabled in the PIEIERx register, and b) flagged as pending in the PIEIFRx is fetched and used as the branch address. In this manner if an even higher priority enabled interrupt was flagged after Step 4**,** it will be serviced first.
- If no flagged interrupts within the group are enabled, then the PIE will respond with the vector for the highest priority interrupt within that group. That is the branch address used for INTx.1. This behavior corresponds to the 28x TRAP or INT instructions.

Note:

Because the PIEIERx register is used to determine which vector will be used for the branch, you must take care when clearing bits within the PIEIERx register. The proper procedure for clearing bits within a PIEIERx register is described in Section 6.3.2. Failure to follow these steps can result in changes occurring to the PIEIERx register after an interrupt has been passed to the CPU at point (5) in Figure 6−5. In this case the PIE will respond as if a TRAP or INT instruction was executed unless there are other interrupts both pending and enabled.

At this point, the PIEIFRx.y bit is cleared and the CPU branches to the vector of the interrupt fetched from the PIE.

6.3.4 The PIE Vector Table

The PIE vector table (see Table 6−4) consists of a 256 x 16 SARAM block that can also be used as RAM (in data space only) if the PIE block is not in use. The PIE vector table contents are undefined on reset. The CPU fixes interrupt priority for INT1 to INT12. The PIE controls priority for each group of eight interrupts. For example, if INT1.1 should occur simultaneously with INT8.1, both interrupts are presented to the CPU simultaneously by the PIE block, and the CPU services INT1.1 first. If INT1.1 should occur simultaneously with INT1.8, then INT1.1 is sent to the CPU first and then INT1.8 follows. Interrupt prioritization is performed during the vector fetch portion of the interrupt processing.

A TRAP 1 to TRAP 12 instruction or an INTR INT1 to INTR INT12 instruction fetches the vector from the first location of each group (INTR1.1 to INT12.1). Similarly an OR IFR,#16-bit operation causes the vector to be fetched from INTR1.1 to INTR12.1 locations if the respective interrupt flag is set. All other TRAP, INTR, OR IFR,#16-bit operations fetch the vector from the respective table location. You should avoid using such operations for INTR1 to INTR12. The TRAP #0 operation returns a vector value of 0x000000 The vector table is EALLOW protected.

Table 6−4. 281x PIE Vector Table

Name	Vector ID	Address	Size (x16)	Description	CPU Priority	Pie Group Priority
Reset	0	0x0000 0D00	2	Reset is always fetched from location 0x003F FFC0 in Boot ROM or XINTF Zone 7	1 (highest)	−
INT1	1	0x0000 0D02	2	Not used. See PIE Group 1	5	−
INT2	2	0x0000 0D04	2	Not used. See PIE Group 2	6	−
INT3	3	0x0000 0D06	2	Not used. See PIE Group 3	7	−
INT4	4	0x0000 0D08	2	Not used. See PIE Group 4	8	−
INT5	5	0x0000 0D0A	2	Not used. See PIE Group 5	9	−
INT6	6	0x0000 0D0C	2	Not used. See PIE Group 6	10	−
INT7	7	0x0000 0D0E	2	Not used. See PIE Group 7	11	−
INT8	8	0x0000 0D10	2	Not used. See PIE Group 8	12	−

Notes: 1) All the locations within the PIE vector table are EALLOW protected.
2) The VECTOR ID is used by DSP/BIOS.
3) Reset is always fetched from location 0x003F FFC0 in Boot ROM or XINTF Zone 7.

Table 6−4. 281x PIE Vector Table (Continued)

Name	Vector ID	Address	Size (x16)	Description	CPU Priority	Pie Group Priority
INT9	9	0x0000 0D12	2	Not used. See PIE Group 9	13	−
INT10	10	0x0000 0D14	2	Not used. See PIE Group 10	14	−
INT11	11	0x0000 0D16	2	Not used. See PIE Group 11	15	−
INT12	12	0x0000 0D18	2	Not used. See PIE Group 12	16	−
INT13	13	0x0000 0D1A	2	External Interrupt 13 (XINT13) or CPU−Timer1 (for TI/RTOS use)	17	−
INT14	14	0x0000 0D1C	2	CPU−Timer2 (for TI/RTOS use)	18	−
DATALOG	15	0x0000 0D1E	2	CPU Data Logging Interrupt	19 (lowest)	−
RTOSINT	16	0x0000 0D20	2	CPU Real−Time OS Interrupt	4	−
EMUINT	17	0x0000 0D22	2	CPU Emulation Interrupt	2	−
NMI	18	0x0000 0D24	2	External Nonmaskable Interrupt	3	−
ILLEGAL	19	0x0000 0D26	2	Illegal Operation	−	−
USER1	20	0x0000 0D28	2	User−Defined Trap	−	−
USER2	21	0x0000 0D2A	2	User Defined Trap	−	−
USER3	22	0x0000 0D2C	2	User Defined Trap	−	−
USER4	23	0x0000 0D2E	2	User Defined Trap	−	−
USER5	24	0x0000 0D30	2	User Defined Trap	−	−
USER6	25	0x0000 0D32	2	User Defined Trap	−	−
USER7	26	0x0000 0D34	2	User Defined Trap	−	−
USER8	27	0x0000 0D36	2	User Defined Trap	−	−
USER9	28	0x0000 0D38	2	User Defined Trap	−	−
USER10	29	0x0000 0D3A	2	User Defined Trap	−	−

Notes: 1) All the locations within the PIE vector table are EALLOW protected.
2) The VECTOR ID is used by DSP/BIOS.
3) Reset is always fetched from location 0x003F FFC0 in Boot ROM or XINTF Zone 7.

Table 6−4. 281x PIE Vector Table (Continued)

Name	Vector ID	Address	Size (x16)	Description		CPU Priority	Pie Group Priority
USER11	30	0x0000 0D3C	2	User Defined Trap		−	−
USER12	31	0x0000 0D3E	2	User Defined Trap		−	−
PIE Group 1 Vectors - MUXed into CPU INT1							
INT1.1	32	0x0000 0D40	2	PDPINTA	(EV−A)	5	1 (highest)
INT1.2	33	0x0000 0D42	2	PDPINTB	(EV−B)	5	2
INT1.3	34	0x0000 0D44	2	Reserved		5	3
INT1.4	35	0x0000 0D46	2	XINT1		5	4
INT1.5	36	0x0000 0D48	2	XINT2		5	5
INT1.6	37	0x0000 0D4A	2	ADCINT	(ADC)	5	6
INT1.7	38	0x0000 0D4C	2	TINT0	(CPU−Timer0)	5	7
INT1.8	39	0x0000 0D4E	2	WAKEINT	(LPM/WD)	5	8 (lowest)
PIE Group 2 Vectors − MUXed into CPU INT2							
INT2.1	40	0x0000 0D50	2	CMP1INT	(EV−A)	6	1 (highest)
INT2.2	41	0x0000 0D52	2	CMP2INT	(EV−A)	6	2
INT2.3	42	0x0000 0D54	2	CMP3INT	(EV−A)	6	3
INT2.4	43	0x0000 0D56	2	T1PINT	(EV−A)	6	4
INT2.5	44	0x0000 0D58	2	T1CINT	(EV−A)	6	5
INT2.6	45	0x0000 0D5A	2	T1UFINT	(EV−A)	6	6
INT2.7	46	0x0000 0D5C	2	T1OFINT	(EV−A)	6	7
INT2.8	47	0x0000 0D5E	2	Reserved		6	8 (lowest)
PIE Group 3 Vectors − MUXed into CPU INT3							
INT3.1	48	0x0000 0D60	2	T2PINT	(EV−A)	7	1 (highest)
INT3.2	49	0x0000 0D62	2	T2CINT	(EV−A)	7	2

Notes: 1) All the locations within the PIE vector table are EALLOW protected.
2) The VECTOR ID is used by DSP/BIOS.
3) Reset is always fetched from location 0x003F FFC0 in Boot ROM or XINTF Zone 7.

Table 6−4. 281x PIE Vector Table (Continued)

Name	Vector ID	Address	Size (x16)	Description		CPU Priority	Pie Group Priority
INT3.3	50	0x0000 0D64	2	T2UFINT	(EV−A)	7	3
INT3.4	51	0x0000 0D66	2	T2OFINT	(EV−A)	7	4
INT3.5	52	0x0000 0D68	2	CAPINT1	(EV−A)	7	5
INT3.6	53	0x0000 0D6A	2	CAPINT2	(EV−A)	7	6
INT3.7	54	0x0000 0D6C	2	CAPINT3	(EV−A)	7	7
INT3.8	55	0x0000 0D6E	2	Reserved		7	8 (lowest)
PIE Group 4 Vectors − MUXed into CPU INT4							
INT4.1	56	0x0000 0D70	2	CMP4INT	(EV−B)	8	1 (highest)
INT4.2	57	0x0000 0D72	2	CMP5INT	(EV−B)	8	2
INT4.3	58	0x0000 0D74	2	CMP6INT	(EV−B)	8	3
INT4.4	59	0x0000 0D76	2	T3PINT	(EV−B)	8	4
INT4.5	60	0x0000 0D78	2	T3CINT	(EV−B)	8	5
INT4.6	61	0x0000 0D7A	2	T3UFINT	(EV−B)	8	6
INT4.7	62	0x0000 0D7C	2	T3OFINT	(EV−B)	8	7
INT4.8	63	0x0000 0D7E	2	Reserved		8	8 (lowest)
PIE Group 5 Vectors − MUXed into CPU INT5							
INT5.1	64	0x0000 0D80	2	T4PINT	(EV−B)	9	1 (highest)
INT5.2	65	0x0000 0D82	2	T4CINT	(EV−B)	9	2
INT5.3	66	0x0000 0D84	2	T4UFINT	(EV−B)	9	3
INT5.4	67	0x0000 0D86	2	T4OFINT	(EV−B)	9	4
INT5.5	68	0x0000 0D88	2	CAPINT4	(EV−B)	9	5
INT5.6	69	0x0000 0D8A	2	CAPINT5	(EV−B)	9	6
INT5.7	70	0x0000 0D8C	2	CAPINT6	(EV−B)	9	7

Notes: 1) All the locations within the PIE vector table are EALLOW protected.

2) The VECTOR ID is used by DSP/BIOS.

3) Reset is always fetched from location 0x003F FFC0 in Boot ROM or XINTF Zone 7.

Table 6−4. 281x PIE Vector Table (Continued)

Name	Vector ID	Address	Size (x16)	Description		CPU Priority	Pie Group Priority
INT5.8	71	0x0000 0D8E	2	Reserved		9	8 (lowest)
PIE Group 6 Vectors − MUXed into CPU INT6							
INT6.1	72	0x0000 0D90	2	SPIRXINTA	(SPI)	10	1 (highest)
INT6.2	73	0x0000 0D92	2	SPITXINTA	(SPI)	10	2
INT6.3	74	0x0000 0D94	2	Reserved		10	3
INT6.4	75	0x0000 0D96	2	Reserved		10	4
INT6.5	76	0x0000 0D98	2	MRINT	(McBSP)	10	5
INT6.6	77	0x0000 0D9A	2	MXINT	(McBSP)	10	6
INT6.7	78	0x0000 0D9C	2	Reserved		10	7
INT6.8	79	0x0000 0D9E	2	Reserved		10	8 (lowest)
PIE Group 7 Vectors − MUXed into CPU INT7							
INT7.1	80	0x0000 0DA0	2	Reserved		11	1 (highest)
INT7.2	81	0x0000 0DA2	2	Reserved		11	2
INT7.3	82	0x0000 0DA4	2	Reserved		11	3
INT7.4	83	0x0000 0DA6	2	Reserved		11	4
INT7.5	84	0x0000 0DA8	2	Reserved		11	5
INT7.6	85	0x0000 0DAA	2	Reserved		11	6
INT7.7	86	0x0000 0DAC	2	Reserved		11	7
INT7.8	87	0x0000 0DAE	2	Reserved		11	8 (lowest)
PIE Group 8 Vectors − MUXed into CPU INT8							
INT8.1	88	0x0000 0DB0	2	Reserved		12	1 (highest)
INT8.2	89	0x0000 0DB2	2	Reserved		12	2
INT8.3	90	0x0000 0DB4	2	Reserved		12	3

Notes: 1) All the locations within the PIE vector table are EALLOW protected.
2) The VECTOR ID is used by DSP/BIOS.
3) Reset is always fetched from location 0x003F FFC0 in Boot ROM or XINTF Zone 7.

Table 6−4. 281x PIE Vector Table (Continued)

Name	Vector ID	Address	Size (x16)	Description		CPU Priority	Pie Group Priority
INT8.4	91	0x0000 0DB6	2	Reserved		12	4
INT8.5	92	0x0000 0DB8	2	Reserved		12	5
INT8.6	93	0x0000 0DBA	2	Reserved		12	6
INT8.7	94	0x0000 0DBC	2	Reserved		12	7
INT8.8	95	0x0000 0DBE	2	Reserved		12	8 (lowest)
PIE Group 9 Vectors − MUXed into CPU INT9							
INT9.1	96	0x0000 0DC0	2	SCIRXINTA	(SCI−A)	13	1 (highest)
INT9.2	97	0x0000 0DC2	2	SCITXINTA	(SCI−A)	13	2
INT9.3	98	0x0000 0DC4	2	SCIRXINTB	(SCI−B)	13	3
INT9.4	99	0x0000 0DC6	2	SCITXINTB	(SCI−B)	13	4
INT9.5	100	0x0000 0DC8	2	ECAN0INT	(ECAN)	13	5
INT9.6	101	0x0000 0DCA	2	ECAN1INT	(ECAN)	13	6
INT9.7	102	0x0000 0DCC	2	Reserved		13	7
INT9.8	103	0x0000 0DCE	2	Reserved		13	8 (lowest)
PIE Group 10 Vectors − MUXed into CPU INT10							
INT10.1	104	0x0000 0DD0	2	Reserved		14	1 (highest)
INT10.2	105	0x0000 0DD2	2	Reserved		14	2
INT10.3	106	0x0000 0DD4	2	Reserved		14	3
INT10.4	107	0x0000 0DD6	2	Reserved		14	4
INT10.5	108	0x0000 0DD8	2	Reserved		14	5
INT10.6	109	0x0000 0DDA	2	Reserved		14	6
INT10.7	110	0x0000 0DDC	2	Reserved		14	7
INT10.8	111	0x0000 0DDE	2	Reserved		14	8 (lowest)

Notes: 1) All the locations within the PIE vector table are EALLOW protected.
2) The VECTOR ID is used by DSP/BIOS.
3) Reset is always fetched from location 0x003F FFC0 in Boot ROM or XINTF Zone 7.

Table 6−4. 281x PIE Vector Table (Continued)

Name	Vector ID	Address	Size (x16)	Description	CPU Priority	Pie Group Priority
PIE Group 11 Vectors − MUXed into CPU INT11						
INT11.1	112	0x0000 0DE0	2	Reserved	15	1 (highest)
INT11.2	113	0x0000 0DE2	2	Reserved	15	2
INT11.3	114	0x0000 0DE4	2	Reserved	15	3
INT11.4	115	0x0000 0DE6	2	Reserved	15	4
INT11.5	116	0x0000 0DE8	2	Reserved	15	5
INT11.6	117	0x0000 0DEA	2	Reserved	15	6
INT11.7	118	0x0000 0DEC	2	Reserved	15	7
INT11.8	119	0x0000 0DEE	2	Reserved	15	8 (lowest)
PIE Group 12 Vectors − MUXed into CPU INT12						
INT12.1	120	0x0000 0DF0	2	Reserved	16	1 (highest)
INT12.2	121	0x0000 0DF2	2	Reserved	16	2
INT12.3	122	0x0000 0DF4	2	Reserved	16	3
INT12.4	123	0x0000 0DF6	2	Reserved	16	4
INT12.5	124	0x0000 0DF8	2	Reserved	16	5
INT12.6	125	0x0000 0DFA	2	Reserved	16	6
INT12.7	126	0x0000 0DFC	2	Reserved	16	7
INT12.8	127	0x0000 0DFE	2	Reserved	16	8 (lowest)

Notes: 1) All the locations within the PIE vector table are EALLOW protected.
2) The VECTOR ID is used by DSP/BIOS.
3) Reset is always fetched from location 0x003F FFC0 in Boot ROM or XINTF Zone 7.

The interrupt grouping for peripherals and external interrupts connected to the PIE module is shown in Table 6−5. Each row in the table shows the 8 interrupts multiplexed into a particular CPU interrupt.

Table 6−5. 281x PIE Peripheral Interrupts†

CPU Interrupts	PIE Interrupts							
	INTx.8	**INTx.7**	**INTx.6**	**INTx.5**	**INTx.4**	**INTx.3**	**INTx.2**	**INTx.1**
INT1.y	$\overline{\text{WAKEINT}}$ (LPM/WD)	TINT0 (TIMER 0)	ADCINT (ADC)	XINT2	XINT1	Reserved	PDPINTB (EV-B)	PDPINTA (EV-A)
INT2.y	Reserved	T1OFINT (EV-A)	T1UFINT (EV-A)	T1CINT (EV-A)	T1PINT (EV-A)	CMP3INT (EV-A)	CMP2INT (EV-A)	CMP1INT (EV-A)
INT3.y	Reserved	CAPINT3 (EV-A)	CAPINT2 (EV-A)	CAPINT1 (EV-A)	T2OFINT (EV-A)	T2UFINT (EV-A)	T2CINT (EV-A)	T2PINT (EV-A)
INT4.y	Reserved	T3OFINT (EV-B)	T3UFINT (EV-B)	T3CINT (EV-B)	T3PINT (EV-B)	CMP6INT (EV-B)	CMP5INT (EV-B)	CMP4INT (EV-B)
INT5.y	Reserved	CAPINT6 (EV-B)	CAPINT5 (EV-B)	CAPINT4 (EV-B)	T4OFINT (EV-B)	T4UFINT (EV-B)	T4CINT (EV-B)	T4PINT (EV-B)
INT6.y	Reserved	Reserved	MXINT (McBSP)	MRINT (McBSP)	Reserved	Reserved	SPITXINTA (SPI)	SPIRXINTA (SPI)
INT7.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
INT8.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
INT9.y	Reserved	Reserved	ECAN1INT (CAN)	ECAN0INT (CAN)	SCITXINTB (SCI-B)	SCIRXINTB (SCI-B)	SCITXINTA (SCI-A)	SCIRXINTA (SCI-A)
INT10.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
INT11.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
INT12.y	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

† Out of the 96 possible interrupts, 45 interrupts are currently used. the remaining interrupts are reserved for future devices. However, these interrupts can be used as software interrupts if they are enabled at the PIEIFRx level.

6.4 PIE Configuration Registers

The registers controlling the functionality of the PIE block are shown in Table 6−6.

Table 6−6. PIE Configuration and Control Registers

Name	Address	Size (x16)	Description
PIECTRL	0x0000−0CE0	1	PIE, Control Register
PIEACK	0x0000−0CE1	1	PIE, Acknowledge Register
PIEIER1	0x0000−0CE2	1	PIE, INT1 Group Enable Register
PIEIFR1	0x0000−0CE3	1	PIE, INT1 Group Flag Register
PIEIER2	0x0000−0CE4	1	PIE, INT2 Group Enable Register
PIEIFR2	0x0000−0CE5	1	PIE, INT2 Group Flag Register
PIEIER3	0x0000−0CE6	1	PIE, INT3 Group Enable Register
PIEIFR3	0x0000−0CE7	1	PIE, INT3 Group Flag Register
PIEIER4	0x0000−0CE8	1	PIE, INT4 Group Enable Register
PIEIFR4	0x0000−0CE9	1	PIE, INT4 Group Flag Register
PIEIER5	0x0000−0CEA	1	PIE, INT5 Group Enable Register
PIEIFR5	0x0000−0CEB	1	PIE, INT5 Group Flag Register
PIEIER6	0x0000−0CEC	1	PIE, INT6 Group Enable Register
PIEIFR6	0x0000−0CED	1	PIE, INT6 Group Flag Register
PIEIER7	0x0000−0CEE	1	PIE, INT7 Group Enable Register
PIEIFR7	0x0000−0CEF	1	PIE, INT7 Group Flag Register
PIEIER8	0x0000−0CF0	1	PIE, INT8 Group Enable Register
PIEIFR8	0x0000−0CF1	1	PIE, INT8 Group Flag Register
PIEIER9	0x0000−0CF2	1	PIE, INT9 Group Enable Register
PIEIFR9	0x0000−0CF3	1	PIE, INT9 Group Flag Register
PIEIER10	0x0000−0CF4	1	PIE, INT10 Group Enable Register
PIEIFR10	0x0000−0CF5	1	PIE, INT10 Group Flag Register
PIEIER11	0x0000−0CF6	1	PIE, INT11 Group Enable Register

Note: The PIE configuration and control registers are not protected by EALLOW mode. The PIE vector table is protected.

Table 6−6. PIE Configuration and Control Registers (Continued)

Name	Address	Size (x16)	Description
PIEIFR11	0x0000−0CF7	1	PIE, INT11 Group Flag Register
PIEIER12	0x0000−0CF8	1	PIE, INT12 Group Enable Register
PIEIFR12	0x0000−0CF9	1	PIE, INT12 Group Flag Register
Reserved	0x0000−0CFA 0x0000−0CFF	6	Reserved

Note: The PIE configuration and control registers are not protected by EALLOW mode. The PIE vector table is protected.

6.5 PIE Interrupt Registers

Table 6−7. PIECTRL Register-Address CE0

15	1	0
PIEVECT		ENPIE
R-0		R/W-0

Legend: R = Read access, W = write access, -0 = value after reset

Table 6−8. PIECTRL Register-Field Descriptions

Bits	Field	Description
15−1	PIEVECT	These bits indicate the address within the PIE vector table from which the vector was fetched. The least significant bit of the address is ignored and only bits 1 to 15 of the address is shown. You can read the vector value to determine which interrupt generated the vector fetch. **Example** `If PIECTRL = 0x0d27` then the vector from address 0x0D26 (illegal operation) was fetched.
0	ENPIE	Enable vector fetching from PIE block. When ENPIE is set to 1, all vectors are fetched from the PIE vector table. If this bit is set to 0, the PIE block is disabled and vectors are fetched from the CPU vector table in boot ROM or external interface Zone 7. All PIE block registers (PIEACK, PIEIFR, PIEIER) can be accessed even when the PIE block is disabled. **Note:** The reset vector is never fetched from the PIE, even when it is enabled. This vector is always fetched from boot ROM or XINTF Zone 7 depending on the state of the XMPNMC input signal.

Figure 6−6. PIE Interrupt Acknowledge Register (PIEACK) Register-Address CE1

15	12	11	0
Reserved		PIEACKx	
R-0		R/W1C-0	

Legend: R = Read access, W1C = write1 to clear, -0 = value after reset

Table 6−9. PIE Interrupt Acknowledge Register (PIEACK) Register Field Descriptions

Bits	Field	Description
15−12	Reserved	
11−0	PIEACKx	Writing a 1 to the respective interrupt bit clears the bit and enables the PIE block to drive a pulse into the core interrupt input, if an interrupt is pending on any of the group interrupts. Reading this register indicates if an interrupt is pending in the respective group. Bit 0 refers to INT1 up to Bit 11, which refers to INT12. Note: Writes of 0 are ignored.

6.5.1 PIE Interrupt Flag Registers

There are twelve PIEIFR registers, one for each CPU interrupt used by the PIE module (INT1−INT12).

Figure 6−7. PIEIFRx Register (x = 1 to 12)

15							8
Reserved							
R-0							

7	6	5	4	3	2	1	0
INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Table 6−10. PIEIFRx Register (x = 1 to 12) Field Descriptions

Bits	Field	Description
15−8	Reserved	
7	INTx.8	These register bits indicate if an interrupt is currently active. They behave very much like the CPU interrupt flag register. When an interrupt is active, the respective register bit is set. The bit is cleared when the interrupt is serviced or by writing a 0 to the register bit. This register can also be read to determine which interrupts are active or pending. x = 1 to 12. INTx means CPU INT1 to INT12
6	INTx.7	
5	INTx.6	
4	INTx.5	
3	INTx.4	
2	INTx.3	
1	INTx.2	
0	INTx.1	

Notes: 1) All of the above registers reset values are set by reset.
2) Hardware has priority over CPU accesses to the PIEIFR registers.
3) The PIEIFR register bit is cleared during the interrupt vector fetch portion of the interrupt processing.

Note:

Never clear a PIEIFR bit. An interrupt may be lost during the read-modify-write operation. See Section 6.3.1 for a method to clear flagged interrupts.

6.5.2 PIE Interrupt Enable Registers

There are twelve PIEIER registers, one for each CPU interrupt used by the PIE module (INT1−INT12).

Figure 6−8. PIEIERx Register (x = 1 to 12)

15							8
Reserved							
R-0							

7	6	5	4	3	2	1	0
INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

Table 6−11. PIEIERx Register (x = 1 to 12) Field Descriptions

Bits	Field	Description
15−8	Reserved	
7	INTx.8	These register bits individually enable an interrupt within a group and behave very much like the core interrupt enable register. Setting a bit to 1 enables the servicing of the respective interrupt. Setting a bit to 0 disables the servicing of the interrupt.
6	INTx.7	
5	INTx.6	x = 1 to 12. INTx means CPU INT1 to INT12
4	INTx.5	
3	INTx.4	
2	INTx.3	
1	INTx.2	
0	INTx.1	

Note: All of the above registers reset values are set by reset.

Note:

Care must be taken when clearing PIEIER bits during normal operation. See Section 6.3.2 for the proper procedure for handling these bits.

6.5.3 CPU Interrupt Flag Register (IFR)

The CPU interrupt flag register (IFR), is a 16-bit, CPU register and is used to identify and clear pending interrupts. The IFR contains flag bits for all the

maskable interrupts at the CPU level (INT1−INT14, DLOGINT and RTOSINT). When the PIE is enabled, the PIE module multiplexes interrupt sources for INT1−INT12.

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.

To identify pending interrupts, use the PUSH IFR instruction and then test the value on the stack. Use the OR IFR instruction to set IFR bits and use the AND IFR instruction to manually clear pending interrupts. All pending interrupts are cleared with the AND IFR #0 instruction or by a hardware reset.

The following events also clear an IFR flag:

- ❑ The CPU acknowledges the interrupt.
- ❑ The 28x device is reset.

Notes:

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

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 6−9. Interrupt Flag Register (IFR) — CPU Register

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

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

Table 6−12. Interrupt Flag Register (IFR) Field Descriptions

Bits	Field	Description
15	RTOSINT	Real-time operating system flag. RTOSINT is the flag for RTOS interrupts. 0 No RTOS interrupt is pending 1 At least one RTOS interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
14	DLOGINT	Data logging interrupt fag. DLOGINT is the flag for data logging interrupts. 0 No DLOGINT is pending 1 At least one DLOGINT interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
13	INT14	Interrupt 14 flag. INT14 is the flag for interrupts connected to CPU interrupt level INT14. 0 No INT14 interrupt is pending 1 At least one INT14 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
12	INT13	Interrupt 13 flag. INT13 is the flag for interrupts connected to CPU interrupt level INT13. 0 No INT13 interrupt is pending 1 At least one INT13 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
11	INT12	Interrupt 12 flag. INT12 is the flag for interrupts connected to CPU interrupt level INT12. 0 No INT12 interrupt is pending 1 At least one INT12 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request

Table 6−12. Interrupt Flag Register (IFR) Field Descriptions (Continued)

Bits	Field	Description
10	INT11	Interrupt 11 flag. INT11 is the flag for interrupts connected to CPU interrupt level INT11. 0 No INT11 interrupt is pending 1 At least one INT11 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
9	INT10	Interrupt 10 flag. INT10 is the flag for interrupts connected to CPU interrupt level INT10. 0 No INT10 interrupt is pending 1 At least one INT6 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
8	INT9	Interrupt 9 flag. INT9 is the flag for interrupts connected to CPU interrupt level INT6. 0 No INT9 interrupt is pending 1 At least one INT9 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
7	INT8	Interrupt 8 flag. INT8 is the flag for interrupts connected to CPU interrupt level INT6. 0 No INT8 interrupt is pending 1 At least one INT8 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
6	INT7	Interrupt 7 flag. INT7 is the flag for interrupts connected to CPU interrupt level INT7. 0 No INT7 interrupt is pending 1 At least one INT7 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
5	INT6	Interrupt 6 flag. INT6 is the flag for interrupts connected to CPU interrupt level INT6. 0 No INT6 interrupt is pending 1 At least one INT6 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
4	INT5	Interrupt 5 flag. INT5 is the flag for interrupts connected to CPU interrupt level INT5. 0 No INT5 interrupt is pending 1 At least one INT5 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request

Table 6−12. Interrupt Flag Register (IFR) Field Descriptions (Continued)

Bits	Field	Description
3	INT4	Interrupt 4 flag. INT4 is the flag for interrupts connected to CPU interrupt level INT4. 0 No INT4 interrupt is pending 1 At least one INT4 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
2	INT3	Interrupt 3 flag. INT3 is the flag for interrupts connected to CPU interrupt level INT3. 0 No INT3 interrupt is pending 1 At least one INT3 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
1	INT2	Interrupt 2 flag. INT2 is the flag for interrupts connected to CPU interrupt level INT2. 0 No INT2 interrupt is pending 1 At least one INT2 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request
0	INT1	Interrupt 1 flag. INT1 is the flag for interrupts connected to CPU interrupt level INT1. 0 No INT1 interrupt is pending 1 At least one INT1 interrupt is pending. Write a 0 to this bit to clear it to 0 and clear the interrupt request

6.5.4 Interrupt Enable Register (IER) and Debug Interrupt Enable Register (DBGIER)

The IER is a 16-bit CPU register. The IER contains enable bits for all the maskable CPU interrupt levels (INT1−INT14, RTOSINT and DLOGINT). Neither NMI nor XRS is included in the IER; thus, IER has no effect on these interrupts.

You can read the IER to identify enabled or disabled interrupt levels, and you can write to the IER to enable or disable interrupt levels. To enable an interrupt level, set its corresponding IER bit to one using the OR IER instruction. To disable an interrupt level, set its corresponding IER bit to zero using the AND IER instruction. When an interrupt is disabled, it is not acknowledged, regardless of the value of the INTM bit. When an interrupt is enabled, it is acknowledged if the corresponding IFR bit is one and the INTM bit is zero.

When using the OR IER and AND IER instructions to modify IER bits make sure they do not modify the state of bit 15 (RTOSINT) unless a real-time operating system is present.

When a hardware interrupt is serviced or an INTR instruction is executed, the corresponding IER bit is cleared automatically. When an interrupt is requested by the TRAP instruction the IER bit is not cleared automatically. In the case of the TRAP instruction if the bit needs to be cleared it must be done by the interrupt service routine.

At reset, all the IER bits are cleared to 0, disabling all maskable CPU level interrupts.

The IER register is shown in Figure 6−10, and descriptions of the bits follow the figure.

Figure 6−10. Interrupt Enable Register (IER) — CPU Register

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
R/W-0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0

7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0	R/W–0

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

Table 6−13. Interrupt Enable Register (IER) Field Descriptions

Bits	Field	Description
15	RTOSINT	Real-time operating system interrupt enable. RTOSINT enables or disables the CPU RTOS interrupt. 0 Level INT6 is disabled 1 Level INT6 is enabled
14	DLOGINT	Data logging interrupt enable. DLOGINT enables or disables the CPU data logging interrupt. 0 Level INT6 is disabled 1 Level INT6 is enabled
13	INT14	Interrupt 14 enable. INT14 enables or disables CPU interrupt level INT14. 0 Level INT14 is disabled 1 Level INT14 is enabled
12	INT13	Interrupt 13 enable. INT13 enables or disables CPU interrupt level INT13. 0 Level INT13 is disabled

Table 6−13. Interrupt Enable Register (IER) Field Descriptions (Continued)

Bits	Field	Description
		1 Level INT13 is enabled
11	INT12	Interrupt 12 enable. INT12 enables or disables CPU interrupt level INT12.
		0 Level INT12 is disabled
		1 Level INT12 is enabled
10	INT11	Interrupt 11 enable. INT11 enables or disables CPU interrupt level INT11.
		0 Level INT11 is disabled
		1 Level INT11 is enabled
9	INT10	Interrupt 10 enable. INT10 enables or disables CPU interrupt level INT10.
		0 Level INT10 is disabled
		1 Level INT10 is enabled
8	INT9	Interrupt 9 enable. INT9 enables or disables CPU interrupt level INT9.
		0 Level INT9 is disabled
		1 Level INT9 is enabled
7	INT8	Interrupt 8 enable. INT8 enables or disables CPU interrupt level INT8.
		0 Level INT8 is disabled
		1 Level INT8 is enabled
6	INT7	Interrupt 7 enable. INT7 enables or disables CPU interrupt level INT7.
		0 Level INT7 is disabled
		1 Level INT7 is enabled
5	INT6	Interrupt 6 enable. INT6 enables or disables CPU interrupt level INT6.
		0 Level INT6 is disabled
		1 Level INT6 is enabled
4	INT5	Interrupt 5 enable. INT5 enables or disables CPU interrupt level INT5.
		0 Level INT5 is disabled
		1 Level INT5 is enabled
3	INT4	Interrupt 4 enable. INT4 enables or disables CPU interrupt level INT4.

Table 6−13. Interrupt Enable Register (IER) Field Descriptions (Continued)

Bits	Field	Description
		0 Level INT4 is disabled
		1 Level INT4 is enabled
2	INT3	Interrupt 3 enable. INT3 enables or disables CPU interrupt level INT3.
		0 Level INT3 is disabled
		1 Level INT3 is enabled
1	INT2	Interrupt 2 enable. INT2 enables or disables CPU interrupt level INT2.
		0 Level INT2 is disabled
		1 Level INT2 is enabled
0	INT1	Interrupt 1 enable. INT1 enables or disables CPU interrupt level INT1.
		0 Level INT1 is disabled
		1 Level INT1 is enabled

The Debug Interrupt Enable Register (DBGIER) is used only when the CPU is halted in real-time emulation mode. An interrupt enabled in the DBGIER is defined as a time-critical interrupt. When the CPU is halted in real-time mode, the only interrupts that are serviced are time-critical interrupts that are also enabled in the IER. If the CPU is running in real-time emulation mode, the standard interrupt-handling process is used and the DEBIER is ignored.

As with the IER, you can read the DBGIER to identify enabled or disabled interrupts and write to the DBGIER to enable or disable interrupts. To enable an interrupt, set its corresponding bit to 1. To disable an interrupt, set its corresponding bit to 0. Use the PUSH DBGIER instruction to read from the DBGIER and POP DBGIER to write to the DEBIER register. At reset, all the DBGIER bits are set to 0.

Figure 6−11. Debug Interrupt Enable Register (DBGIER) — CPU Register

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
R/W-0							

7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0							

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

Table 6−14. Debug Interrupt Enable Register (DBGIER) Field Descriptions

Bits	Field	Description
15	RTOSINT	Real-time operating system interrupt enable. RTOSINT enables or disables the CPU RTOS interrupt. 0 Level INT6 is disabled 1 Level INT6 is enabled
14	DLOGINT	Data logging interrupt enable. DLOGINT enables or disables the CPU data logging interrupt. 0 Level INT6 is disabled 1 Level INT6 is enabled
13	INT14	Interrupt 14 enable. INT14 enables or disables CPU interrupt level INT14. 0 Level INT14 is disabled 1 Level INT14 is enabled
12	INT13	Interrupt 13 enable. INT13 enables or disables CPU interrupt level INT13. 0 Level INT13 is disabled 1 Level INT13 is enabled
11	INT12	Interrupt 12 enable. INT12 enables or disables CPU interrupt level INT12. 0 Level INT12 is disabled 1 Level INT12 is enabled
10	INT11	Interrupt 11 enable. INT11 enables or disables CPU interrupt level INT11. 0 Level INT11 is disabled 1 Level INT11 is enabled

Table 6−14. Debug Interrupt Enable Register (DBGIER) Field Descriptions (Continued)

Bits	Field	Description
9	INT10	Interrupt 10 enable. INT10 enables or disables CPU interrupt level INT10. 0 Level INT10 is disabled 1 Level INT10 is enabled
8	INT9	Interrupt 9 enable. INT9 enables or disables CPU interrupt level INT9. 0 Level INT9 is disabled 1 Level INT9 is enabled
7	INT8	Interrupt 8 enable. INT8 enables or disables CPU interrupt level INT8. 0 Level INT8 is disabled 1 Level INT8 is enabled
6	INT7	Interrupt 7 enable. INT7 enables or disables CPU interrupt level INT77. 0 Level INT7 is disabled 1 Level INT7 is enabled
5	INT6	Interrupt 6 enable. INT6 enables or disables CPU interrupt level INT6. 0 Level INT6 is disabled 1 Level INT6 is enabled
4	INT5	Interrupt 5 enable. INT5 enables or disables CPU interrupt level INT5. 0 Level INT5 is disabled 1 Level INT5 is enabled
3	INT4	Interrupt 4 enable. INT4 enables or disables CPU interrupt level INT4. 0 Level INT4 is disabled 1 Level INT4 is enabled
2	INT3	Interrupt 3 enable. INT3 enables or disables CPU interrupt level INT3. 0 Level INT3 is disabled 1 Level INT3 is enabled

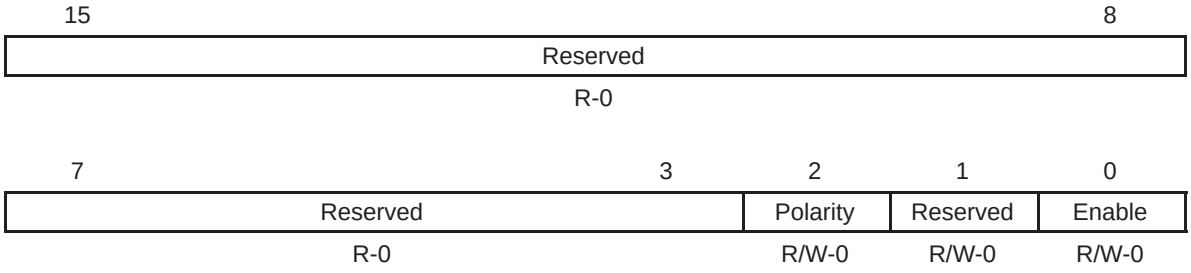
Table 6−14. Debug Interrupt Enable Register (DBGIER) Field Descriptions (Continued)

Bits	Field	Description
1	INT2	Interrupt 2 enable. INT2 enables or disables CPU interrupt level INT2. 0 Level INT2 is disabled 1 Level INT2 is enabled
0	INT1	Interrupt 1 enable. INT1 enables or disables CPU interrupt level INT1. 0 Level INT1 is disabled 1 Level INT1 is enabled

6.6 External Interrupt Control Registers

Some devices support three masked external interrupts XINT1, XINT2, XINT13. XINT13 is multiplexed with one non-maskable interrupt XNMI. Each of these external interrupts can be selected for negative or positive edge triggered and can also be enabled or disabled (including XNMI). The masked interrupts also contain a 16-bit free running up counter that is reset to zero when a valid interrupt edge is detected. This counter can be used to accurately time stamp the interrupt.

Figure 6−12. External Interrupt 1 Control Register (XINT1CR) — Address 7070h



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

Table 6−15. External Interrupt 1 Control Register (XINT1CR) Field Descriptions

Bits	Field	Description
15−3	Reserved	Reads return zero; writes have no effect.
2	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)
1	Reserved	Reads return zero; writes have no effect
0	Enable	This read/write bit enables or disables external interrupt XINT1. 0 Disable interrupt 1 Enable interrupt

Figure 6−13. External Interrupt 2 Control Register (XINT2CR) — Address 7071h

15	8
Reserved	
R-0	

7	3	2	1	0
Reserved		Polarity	Reserved	Enable
R-0		R/W-0	R/W-0	R/W-0

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

Table 6−16. External Interrupt 2 Control Register (XINT2CR) Field Descriptions

Bits	Field	Description
15−3	Reserved	Reads return zero; writes have no effect.
2	Polarity	This read/write bit determines whether interrupts are generated on the rising edge or the falling edge of a signal on the pin. Interrupt generated on a falling edge (high-to-low transition) Interrupt generated on a rising edge low-to-high transition)
1	Reserved	Reads return zero; writes have no effect
0	Enable	This read/write bit enables or disables external interrupt XINT2. Disable interrupt Enable interrupt

Figure 6−14. External NMI Interrupt Control Register (XNMICR) — Address 7077h

15	8
Reserved	
R-0	

7	3	2	1	0
Reserved		Polarity	Select	Enable
R-0		R/W-0	R/W-0	R/W-0

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

Table 6−17. External NMI Interrupt Control Register (XNMICR) Field Descriptions

Bits	Field	Description
15−3	Reserved	Reads return zero; writes have no effect.
2	Polarity	This read/write bit determines whether interrupts are generated on the rising edge or the falling edge of the 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)
1	Select	Select the source for INT13 0 Timer 1 connected To INT13 1 XNMI_XINT13 connected To INT13
0	Enable	This read/write bit enables or disables external interrupt NMI 0 Disable XNMI interrupt 1 Enable XNMI interrupt

The XNMI Control Register (XNMICR) can be used to enable or disable the NMI interrupt to the CPU. In additions, you can select the source for the INT13 CPU interrupt. As shown in Figure 6−4, the source of the INT13 interrupt can either the internal CPU Timer1 or the external XNMI_XINT13 signal.

On the F281x devices, CPU Timer1 is reserved for use by TI software. The INT13 interrupt can, however, still be connected to XNMI_XINT13 for customer use.

Table 6−18 shows the relationship between the XNMICR Register settings and the interrupt sources to the 28x CPU.

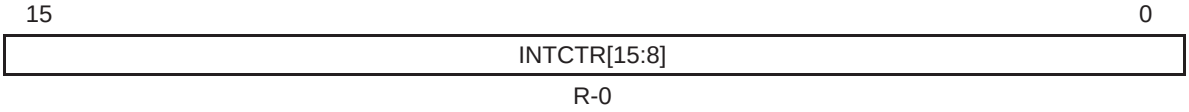
Table 6−18. XNMICR Register Settings and Interrupt Sources

XNMICR Register Bits		28x CPU Interrupt		Timestamp
ENABLE	SELECT	NMI	INT13	(XNMICTR)
0	0	Disabled	CPU Timer 1	None
0	1	Disabled	XNMI_XINT13	None
1	0	XNMI_XINT13	CPU Timer 1	XNMI_XINT13
1	1	XNMI_XINT13	XNMI_XINT13	XNMI_XINT13

When ENABLE = 1 and SELECT = 1, both the NMI and INT13 CPU interrupts will respond to the XNMI_XINT13 signal.

For each external interrupt, there is also a 16-bit counter that is reset to 0x000 whenever an interrupt edge is detected. These counters can be used to accurately time stamp an occurrence of the interrupt.

Figure 6−15. External Interrupt 1 Counter (XINT1CTR) — Address 7078h

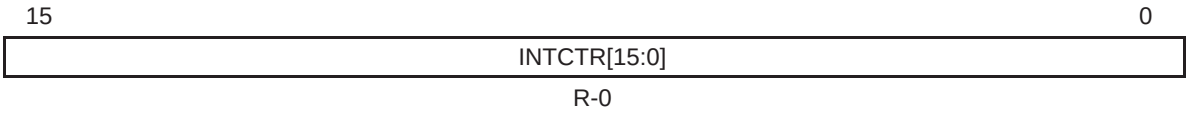


Note: R = Read access, –0 = value after reset

Table 6−19. External Interrupt 1 Counter (XINT1CTR) Field Descriptions

Bits	Field	Description
15−0	INTCTR	This is a free running 16-bit up-counter that is clocked at the SYSCLKOUT rate. The counter value is reset to 0x0000 when a valid interrupt edge is detected and then continues counting until the next valid interrupt edge is detected. When the interrupt is disabled, the counter stops. The counter is a free-running counter and wraps around to zero when the max value is reached. The counter is a read only register and can only be reset to zero by a valid interrupt edge or by reset.

Figure 6−16. External Interrupt 2 Counter (XINT2CTR) — Address 7079h

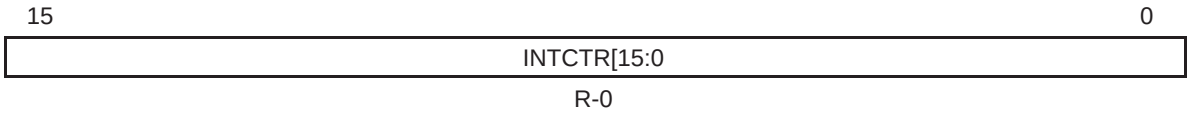


Note: R = Read access, –0 = value after reset

Table 6−20. External Interrupt 2 Counter (XINT2CTR) Field Descriptions

Bits	Field	Description
15−0	INTCTR	This is a free running 16-bit up-counter that is clocked at the SYSCLKOUT rate. The counter value is reset to 0x0000 when a valid interrupt edge is detected and then continues counting until the next valid interrupt edge is detected. When the interrupt is disabled, the counter stops. The counter is a free-running counter and wraps around to zero when the max value is reached. The counter is a read only register and can only be reset to zero by a valid interrupt edge or by reset.

Figure 6−17. External NMI Interrupt Counter (XNMICTR) — Address 707Fh



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

Table 6−21. External NMI Interrupt Counter (XNMICTR) Field Descriptions

Bits	Field	Description
15−0	INTCTR	This is a free running 16-bit up-counter that is clocked at the SYSCLKOUT rate. The counter value is reset to 0x0000 when a valid interrupt edge is detected and then continues counting until the next valid interrupt edge is detected. When the interrupt is disabled, the counter stops. The counter is a free-running counter and wraps around to zero when the max value is reached. The counter is a read only register and can only be reset to zero by a valid interrupt edge or by reset.

This page intentionally left blank.

Appendix A

Revision History

This document was revised to SPRU078B from SPRU078A, which was released in July 2003. The scope of the revisions was limited to technical changes as described in A.1. This appendix lists only revisions made in the most recent version.

A.1 Changes Made in This Revision

Global change: Title of this reference guide was changed to 281x DSP System Control and Interrupts Reference Guide to reflect the devices added since the last version. In various sections of the guide, F2810 and F2812 were also changed to 281x.

The following technical changes were made in this revision:

Page	Additions/Modifications/Deletions
iii	Updated the Preface
1-4	Changed title of Figure 1−1
1-5	Rewrote Section 1.2.2, Flash Pipeline Mode
1-7	Added Section 1.2.3, Procedure to change the Flash Configuration Registers
1-8	Added Figure 1−3, Access Flow Diagram
1-8	Modified Figure 1−3
1-10	Added to description of ENPIPE bit in the Flash Options Register (Figure 1−4)
1-10	Added to the description of the PWR bits in the Flash Power Register (Figure 1−5)
1-12	Added to the description of the STDBYWAIT bits in the Flash Standby Wait Register (Figure 1−7)
1-13	Added to the description of the ACTIVEWAIT bits in the Flash Standby to Active Wait Counter Register (Figure 1−8)
1-13	Added to the PAGEWAIT and RANDWAIT bit descriptions in the Flash Waitstate Register (Figure 1−9)
1-14	Added to the OTPWAIT bit description in the OTP Waitstate Register (Figure 1−10)
2-4	Added to the block information in Table 2−2, F281x/C281x Resources Affected by the CSM
2-6	Added a note to Table 2−4 and corrected typo 0AEB to 0AE3 under memory addresses
2-11	Corrected code in Section 2.3.4, C Code Example to Unsecure
3-2	Modified Figure 3−1, Clock and Reset Domains
3-5	Modified description of WDINTS (bit 2 of the SCSR Register)
3-10	Added a paragraph preceding Figure 3−7
3-12	Modified Table 3−8, 281x Low Power Modes
3-13	Added to descriptions of bits in the Low Power Mode Control 0 Register (Figure 3−8)
3-15	Modified first sentence of section 3.4

Page	Additions/Modifications/Deletions
3-15	Modified the first paragraph following Figure 3−10
3-16	Changed reset value for WDCNTR bits in the Watchdog Counter Register (Figure 3−11)
3-17	Modified reset value of WDDIS (bit 6 of the WDCR register) in Figure 3−13
4-6	Added Section 4.2, Input Qualification
4-5	Changed the title of Figure 4−1 from “Modes of Operating” to “GPIO/Peripheral Pin MUXing” and modified the figure
4-7	Modified Figure 4−3, Input Qualifier Clock Cycles
4-7	Added Figure 4−4, Type 2 Input Qualification
4-8	Added information to GPxDIR Registers in Section 4.3.
4-8	Added information to GPxDAT Registers in Section 4.3.
4-11	Added Section 4.4, Register Bit to I/O Mapping
4-11	Modified Table 4−3
4-12	Modified Table 4−5
4-14	Modified Table 4−7
4-15	Modified Table 4−9
4-16	Modified Table 4−11
4-17	Modified Table 4−12
6-29	Modified Note 1
6-40	Modified Figure 6−14
6-41	Added Table 6−18

This page intentionally left blank.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

O

P

R

S

T

U

V

W

X