TEXAS INSTRUMENTS

MSP430x1xx Family

User's Guide

2000

Mixed Signal Products

SLAU049

IMPORTANT NOTICE

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

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

Customers are responsible for their applications using TI components.

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

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

Preface

Read This First

About This Manual

The MSP430x1xx User's Guide is intended to assist the development of MSP430x1xx family products by assembling together and presenting hardware and software information in a manner that is easy for engineers and programmers to use.

This manual discusses modules and peripherals of the MSP430x1xx family of devices. Each discussion presents the module or peripheral in a general sense. Not all features and functions of all modules or peripherals are present on all devices. In addition, modules or peripherals may differ in their exact implementation between device families, or may not be fully implemented on an individual device or device family. Therefore, a user must always consult the data sheet of any device of interest to determine what peripherals and modules are implemented, and exactly how they are implemented on that particular device.

How to Use This Manual

This document contains the following chapters:

Chapter 14 – Comparator_A
Chapter 15 – ADC12
Appendix A – Peripheral File Map
Appendix B – Instruction Set Description
Appendix C – Flash Memory

Notational Conventions

This document uses the following conventions.

- Program listings, program examples, and interactive displays are shown in a `special typeface` similar to a typewriter's.

  Here is a sample program listing:

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

Related Documentation From Texas Instruments

For related documentation see the web site http://www.ti.com/sc/msp430.

FCC Warning

This equipment is intended for use in a laboratory test environment only. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to subpart J of part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in other environments may cause interference with radio communications, in which case the user at his own expense will be required to take whatever measures may be required to correct this interference.

Contents

Figures

Tables

Examples

Notes, Cautions, and Warnings

Chapter 1

Introduction

This chapter outlines the features and capabilities of the Texas Instruments (TI™) MSP430x1xx family of microcontrollers.

The MSP430 employs a von-Neumann architecture, therefore, all memory and peripherals are in one address space.

The MSP430 devices constitute a family of ultralow-power, 16-bit RISC microcontrollers with an advanced architecture and rich peripheral set. The architecture uses advanced timing and design features, as well as a highly orthogonal structure to deliver a processor that is both powerful and flexible. The MSP430 consumes less than 400 μA in active mode operating at 1 MHz in a typical 3-V system and can wake up from a <2-μA standby mode to fully synchronized operation in less than 6 μs. These exceptionally low current requirements, combined with the fast wake-up time, enable a user to build a system with minimum current consumption and maximum battery life.

Additionally, the MSP430x1xx family has an abundant mix of peripherals and memory sizes enabling true system-on-a-chip designs. The peripherals include a 12-bit A/D, slope A/D, multiple timers (some with capture/compare registers and PWM output capability), on-chip clock generation, H/W multiplier, USART(s), Watchdog Timer, GPIO, and others.

See http://www.ti.com for the latest device information and literature for the MSP430 family.

Topic **Page**

1.1 Features and Capabilities

The TI MSP430x1xx family of controllers has the following features and capabilities:

- Ultralow-power architecture:
  - 0.1– 400 μA nominal operating current @1 MHz
  - 1.8 – 3.6 V operation (2.5–5.5 V for C11x, P11x, and E11x devices)
  - 6 μs wake-up from standby mode
  - Extensive interrupt capability relieves need for polling
- Flexible and powerful processing capabilities:
  - Seven source-address modes
  - Four destination-address modes
  - Only 27 core instructions
  - Prioritized, nested interrupts
  - No interrupt or subroutine level limits
  - Large register file
  - Ram execution capability
  - Efficient table processing
  - Fast hex-to-decimal conversion
- Extensive, memory-mapped peripheral set including:
  - Integrated 12-bit A/D converter
  - Integrated precision comparator
  - Multiple timers and PWM capability
  - Slope A/D conversion (all devices)
  - Integrated USART(s)
  - Watchdog Timer
  - Multiple I/O with extensive interrupt capability
  - Integrated programmable oscillator
  - 32-kHz crystal oscillator (all devices)
  - 450-kHz – 8-MHz crystal oscillator (selected devices)
- Powerful, easy-to-use development tools including:
  - Simulator (including peripheral and interrupt simulation)
  - C compiler
  - Assembler
  - Linker
  - Emulators
  - Flash emulator kit
  - Evaluation kits
  - Device programmer
  - Application notes
  - Example code

- Versatile ultralow-power device options including:
  - Masked ROM
  - OTP (in-system programmable)
  - Flash (in-system programmable)
  - EPROM (UV-erasable, in-system programmable)
  - –40°C to +85°C operating temperature range
  - Up to 64K addressing space
  - Memory mixes to support all types of applications

1.2 11x Devices

The 11x devices contain the following peripherals:

- Basic Clock System (on-chip DCO + one or two crystal oscillators)
- Watchdog Timer/General Purpose Timer
- Timer_A3 (16-bit timer with 3 capture/compare registers and PWM output)
- I/O Port1, 2 (8 I/O's each, all with interrupt)

Available 11x devices are:

MSP430C111	2KB ROM, 128B RAM
MSP430C112	4KB ROM, 256B RAM
MSP430P112	4KB OTP, 256B RAM
PMS430E112	4KB EPROM, 256B RAM
MSP430F110	1KB +128B Flash, 128B RAM
MSP430F112	4KB +256B Flash, 256B RAM

1.3 11x1 Devices

The 11x1 devices contain the following peripherals:

- Basic Clock System (on-chip DCO + one or two crystal oscillators)
- Watchdog Timer/General Purpose Timer
- Timer_A3 (16-bit timer with 3 capture/compare registers and PWM output)
- I/O Port1, 2 (8 I/O's each, all with interrupt)
- Comparator_A (precision analog comparator, ideal for slope A/D conversion)

Available 11x1 devices are:

MSP430C1111	2KB ROM, 128B RAM
MSP430C1121	4KB ROM, 256B RAM
MSP430F1101	1KB +128B Flash, 128B RAM
MSP430F1121	4KB +256B Flash, 256B RAM

1.4 13x Devices

The 13x devices contain the following peripherals:

- Basic Clock System (on-chip DCO + one or two crystal oscillators)
- Watchdog Timer/General Purpose Timer
- Timer_A3 (16-bit timer with 3 capture/compare registers and PWM output)
- Timer_B3 (16-bit timer with 3 capture/compare registers and PWM output)
- I/O Port1, 2 (8 I/O's each, all with interrupt)
- I/O Port3, 4, 5, 6 (8 I/O's each)
- Comparator_A (precision analog comparator, ideal for slope A/D conversion)
- ADC12 (12-bit A/D)
- USART0

13x devices include:

MSP430F133	8KB +256B Flash, 256B RAM
MSP430F135	16KB +256B Flash, 512B RAM

1.5 14x Devices

The 14x devices contain the following peripherals:

- Basic Clock System (on-chip DCO + one or two crystal oscillators)
- Watchdog Timer/General Purpose Timer
- Timer_A3 (16-bit timer with 3 capture/compare registers and PWM output)
- Timer_B7 (16-bit timer with 7 capture/compare registers and PWM output)
- I/O Port1, 2 (8 I/O's each, all with interrupt)
- I/O Port3, 4, 5, 6 (8 I/O's each)
- Comparator_A (precision analog comparator, ideal for slope A/D conversion)
- ADC12 (12-bit A/D)
- USART0
- USART1
- Hardware Multiplier

Available 14x devices are:

MSP430F147	32KB +256B Flash, 1KB RAM
MSP430F148	48KB +256B Flash, 2KB RAM
MSP430F149	60KB +256B Flash, 2KB RAM

Chapter 2

Architectural Overview

This section describes the basic functions of an MSP430-based system.

The MSP430 devices contain the following main elements:

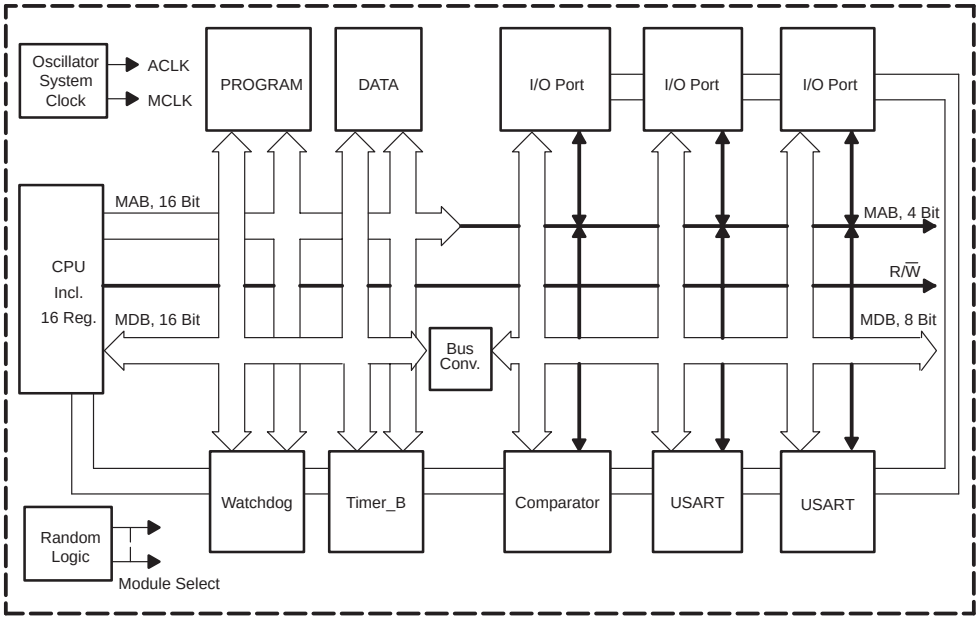
- Central processing unit
- Program memory
- Data memory
- Operation control
- Peripheral modules
- Oscillator and clock generator

Topic **Page**

2.1 Introduction

The architecture of the MSP430 family is based on a memory-to-memory architecture, a common address space for all functional blocks, and a reduced instruction set applicable to all functional blocks as illustrated in Figure 2–1. See specific device data sheets for complete block diagrams of individual devices.

Figure 2–1. MSP430 System Configuration



2.2 Central Processing Unit

The CPU incorporates a reduced and highly transparent instruction set and a highly orthogonal design. It consists of a 16-bit arithmetic logic unit (ALU), 16 registers, and instruction control logic. Four of these registers are used for special purposes. These are the program counter (PC), stack pointer (SP), status register (SR), and constant generator (CGx). All registers, except the constant-generator registers R3/CG2 and part of R2/CG1, can be accessed using the complete instruction set. The constant generator supplies instruction constants, and is not used for data storage. The addressing mode used on CG1 separates the data from the constants.

The CPU control over the program counter, the status register, and the stack pointer (with the reduced instruction set) allows the development of applications with sophisticated addressing modes and software algorithms.

2.3 Program Memory

Instruction fetches from program memory are always 16-bit accesses, whereas data memory can be accessed using word (16-bit) or byte (8-bit) instructions. Any access uses the 16-bit memory data bus (MDB) and as many of the least-significant address lines of the memory address bus (MAB) as required to access the memory locations. Blocks of memory are automatically selected through module-enable signals. This technique reduces overall current consumption. Program memory is integrated as programmable or mask-programmed memory.

In addition to program code, data may also be placed in the ROM section of the memory map and may be accessed using word or byte instructions; this is useful for data tables, for example. This unique feature gives the MSP430 an advantage over other microcontrollers, because the data tables do not have to be copied to RAM for usage.

Sixteen words of memory are reserved for reset and interrupt vectors at the top of the 64-kilobytes address space from 0FFFFh down to 0FFE0h.

2.4 Data Memory

The data memory is connected to the CPU through the same two buses as the program memory (ROM): the memory address bus (MAB) and the memory data bus (MDB). The data memory can be accessed with full (word) data width or with reduced (byte) data width.

Additionally, because the RAM and ROM are connected to the CPU via the same busses, program code can be loaded into and executed from RAM. This is another unique feature of the MSP430 devices, and provides valuable, easy-to-use debugging capability.

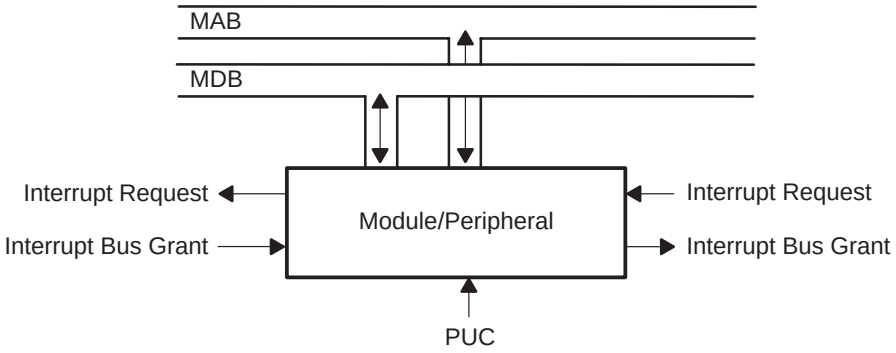
2.5 Operation Control

The operation of the different MSP430 members is controlled mainly by the information stored in the special–function registers (SFRs). The different bits in the SFRs enable interrupts, provide information about the status of interrupt flags, and define the operating modes of the peripherals. By disabling peripherals that are not needed during an operation, total current consumption can be reduced. The individual peripherals are described later in this manual.

2.6 Peripherals

Peripheral modules are connected to the CPU through the MAB, MDB, and interrupt service and request lines. The MAB is usually a 5-bit bus for most of the peripherals. The MDB is an 8-bit or 16-bit bus. Most of the peripherals operate in byte format. Modules with an 8-bit data bus are connected by bus-conversion circuitry to the 16-bit CPU. The data exchange with these modules must be handled with byte instructions. The SFRs are also handled with byte instructions. The operation for 8-bit peripherals follows the order described in Figure 2–2.

Figure 2–2. Bus Connection of Modules/Peripherals



2.7 Oscillator and Clock Generator

The LFXT1 oscillator is designed for the commonly used 32,768 Hz, low-current- consumption clock crystal or to be used with a high-speed crystal. All analog components for the 32,768 Hz oscillator are integrated into the MSP430; only the crystal needs to be connected with no other external components required. When using the LFXT1 oscillator with a high-speed crystal, additional load capacitors are required. Some MSP430 devices have an additional high-speed crystal oscillator (LFXT2). Refer to the clock chapter and the specific device data sheets for details.

In addition to the crystal oscillator(s), all MSP430 devices contain a digitally-controlled RC oscillator (DCO). The DCO is different from RC oscillators found on other microcontrollers because it is digitally controllable and tuneable.

Clock source selection for peripherals and CPU is very flexible. Most peripherals are capable of using the 32768-Hz crystal oscillator clock, the high-speed crystal oscillator clock (where applicable), or the DCO clock. The CPU is capable of executing from the DCO clock or from either of the two crystal oscillator clocks. See Chapter 7 for details on the clock system.

Chapter 3

System Resets, Interrupts, and Operating Modes

This chapter discusses the MSP430x1xx system resets, interrupts, and operating modes.

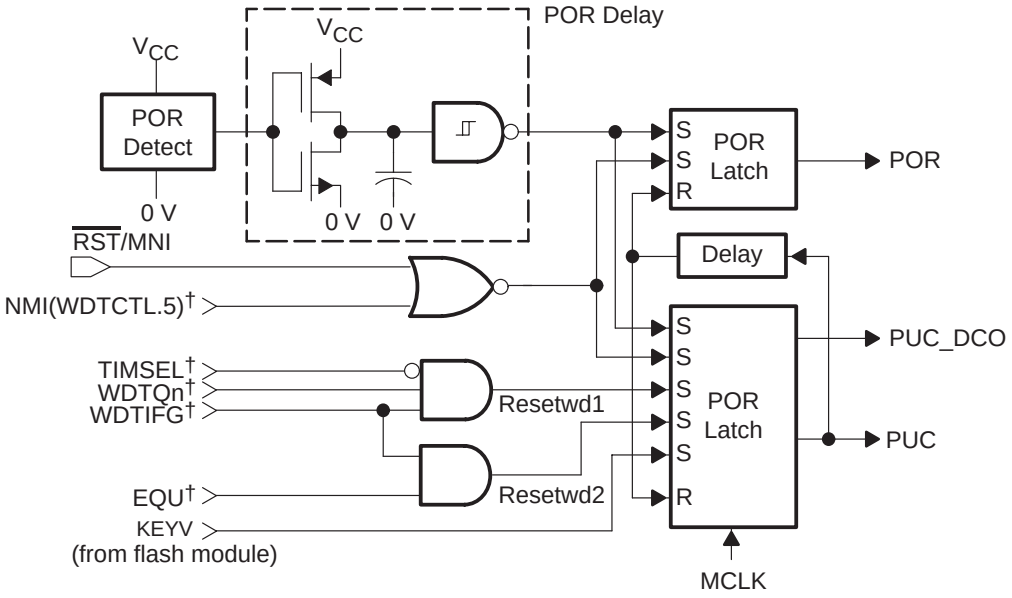
Topic **Page**

3.1 System Reset and Initialization

3.1.1 Introduction

The MSP430 system reset circuitry (shown in Figure 3–1) sources two internal reset signals: power-on reset (POR) and power-up clear (PUC). Different events trigger these reset signals and different initial conditions exist depending on which signal was generated.

Figure 3–1. Power-On Reset and Power-Up Clear Schematic



† From watchdog timer peripheral module

A POR is a device reset. It is only generated by the two following events:

- Powering up the device
- A low signal on the $\overline{\text{RST}}$/NMI pin when configured in the reset mode

A PUC is always generated when a POR is generated, but a POR is not generated by a PUC. The following events trigger a PUC:

- A POR signal
- Watchdog timer expiration (in watchdog mode only)
- Watchdog timer security key violation
- A low signal on the $\overline{\text{RST}}$/NMI pin when configured in the NMI mode
- A Flash memory security key violation

Note:

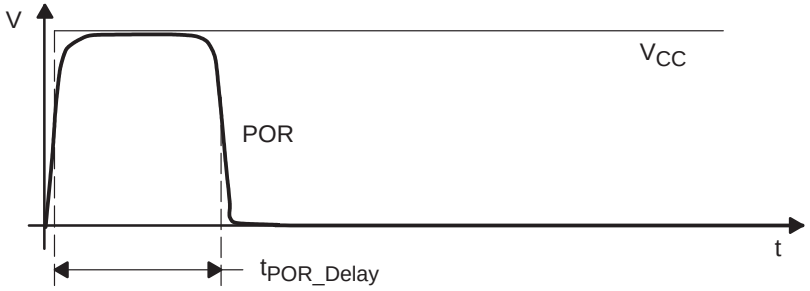
If desired, software can cause a PUC by simply writing to the watchdog timer control register with an incorrect password.

Note:

Generation of the POR/PUC signals does not necessarily generate a system reset interrupt. Anytime a POR is activated, a system reset interrupt is generated. However, when a PUC is activated, a system reset interrupt may or may not be generated. Instead, a lower priority interrupt vector may be generated, depending on what action caused the PUC. Each device data sheet gives a detailed table of what action generates each interrupt. This table should be consulted for the proper handling of all interrupts.

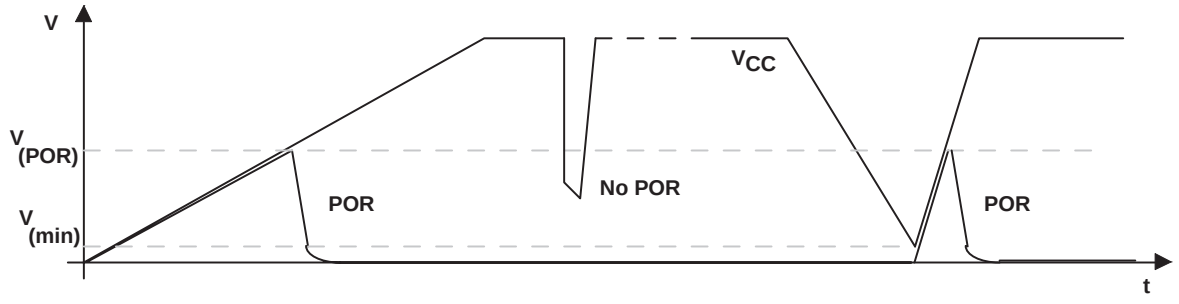
When the V_{CC} supply provides a fast rise time as shown in Figure 3–2, the POR delay provides enough active time on the POR signal to allow the signal to initialize the circuitry correctly after power up. When the V_{CC} rise time is slow, as shown in Figure 3–3, the POR detector holds the POR signal active until V_{cc} has risen above the $V_{(POR)}$ level. This also ensures a correct initialization.

Figure 3–2. Power-On Reset Timing on Fast V_{CC} Rise Time



If power to the chip is cycled, the supply voltage V_{CC} must fall below the $V_{(min)}$ (see Figure 3–3) to ensure that another POR signal occurs when V_{CC} is powered up again. If V_{CC} does not fall below $V_{(min)}$ during a cycle or a glitch, a POR is not generated and power-up conditions do not set correctly.

Figure 3–3. Power-on Reset Timing on Slow V_{CC} Rise Time



3.1.2 Device Initialization After System Reset

After a device reset (POR/PUC combination), the initial system conditions are:

- ❑ I/O pins switched to input mode (see note below).
- ❑ I/O flags are cleared as described in the I/O chapter (see note below).
- ❑ Other peripherals and registers initialized as described in their respective chapters.
- ❑ Status register is reset.
- ❑ Program counter is loaded with address contained at reset vector location (0FFFEh). CPU execution begins at that address.

Note:

I/O pins and flags are only initialized after power-up. After the ’430 is powered and running, if a reset is generated with $\overline{\text{RST}}$/NMI pin (in reset mode), the I/O pins are unaffected.

After a system reset, the user program can evaluate the various flags to determine the source of the reset and take appropriate action.

The initial state of registers and peripherals is discussed in each applicable section of this manual. Each register is shown with a key indicating the accessibility of the register and the initial condition, for example, *rw–(0)*, or *rw–0*. In these examples, the *r* indicates read, the *w* indicates write, and the value after the dash indicates the initial condition. If the value is in parenthesis, the initial condition takes effect only after a POR – a PUC alone will not effect the bit(s). If the value is not in parenthesis, it takes effect after a PUC alone or after a POR/PUC combination. Some examples follow:

Type	Description
rw–(0)	Read/write, reset with POR
rw–0	Read/write, reset with POR or PUC
r–1	Read only, set with POR or PUC
r	Read only, no initial state
w	Write only, no initial state

3.2 Global Interrupt Structure

There are four types of interrupts:

- System reset
- Maskable
- Non-maskable
- (Non)-maskable

System reset (POR/PUC) is discussed in section 3.1.

Maskable interrupts are caused by:

- A watchdog-timer overflow (if timer mode is selected)
- Other modules with interrupt capability

Non-maskable interrupts are not maskable in any way. No individual interrupt enable bit is implemented for them, and the general interrupt enable bit (GIE) has no effect on them.

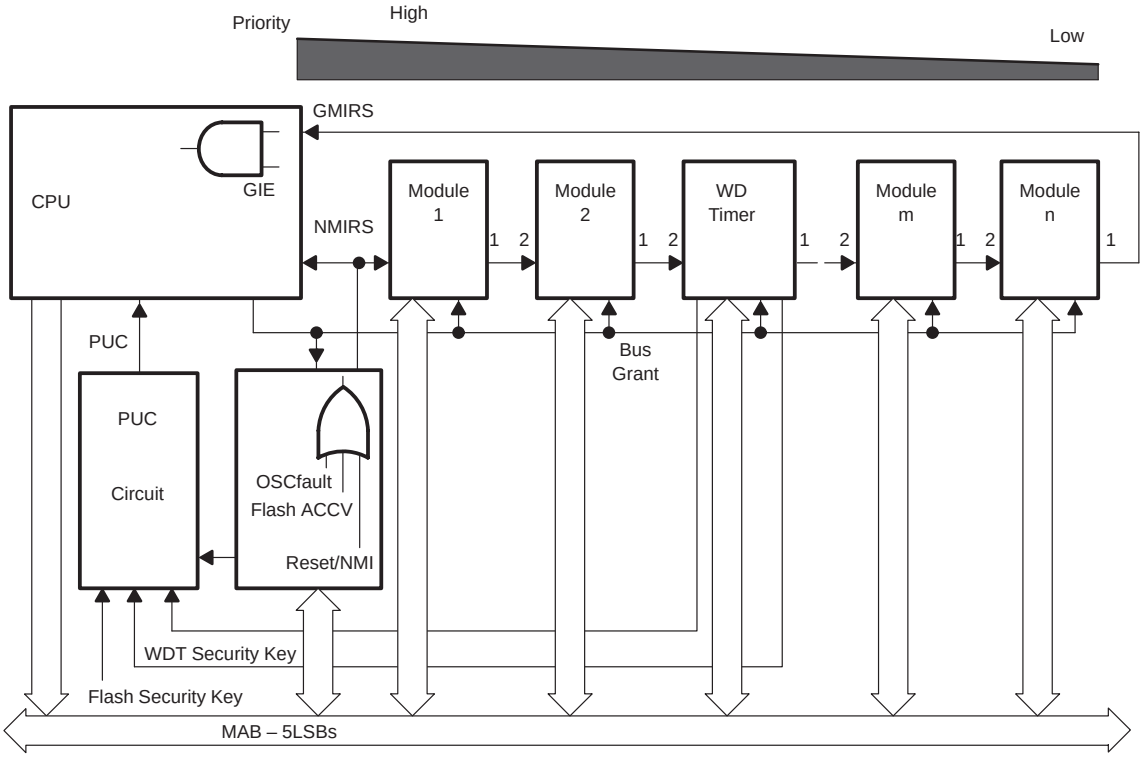
(Non)-maskable interrupts are not masked by the general interrupt enable bit (GIE) but are individually enabled or disabled by an individual interrupt enable bit. When a (non)-maskable interrupt is accepted, the corresponding interrupt enable bit is automatically reset, therefore disabling the interrupt for execution of the interrupt service routine (ISR). The RETI (return from interrupt) instruction has no effect on the individual enable bits of the (non)-maskable interrupts. So the software must set the corresponding interrupt enable bit in the ISR before execution of the RETI instruction for the interrupt to be re-enabled after the ISR.

A (non)-maskable NMI interrupt can be generated by an edge on the RST/NMI pin (if NMI mode is selected), an oscillator fault occurs (if the oscillator fault interrupt is enabled), or an access violation to the flash memory takes place (if the access violation interrupt is enabled).

3.3 MSP430 Interrupt-Priority Scheme

The interrupt priority of the modules, as shown in Figure 3–4, is defined by the arrangement of the modules in the connection chain: the nearer a module is to the CPU/NMIRS, the higher the priority.

Figure 3–4. Interrupt Priority Scheme



Reset and NMI, as shown in Figure 3–5, can only be used as alternative interrupts because they use the same input pin. The associated control bits are located in the watchdog timer control register shown in Figure 3–6, and are password protected.

Figure 3–5. Block Diagram of NMI Interrupt Sources

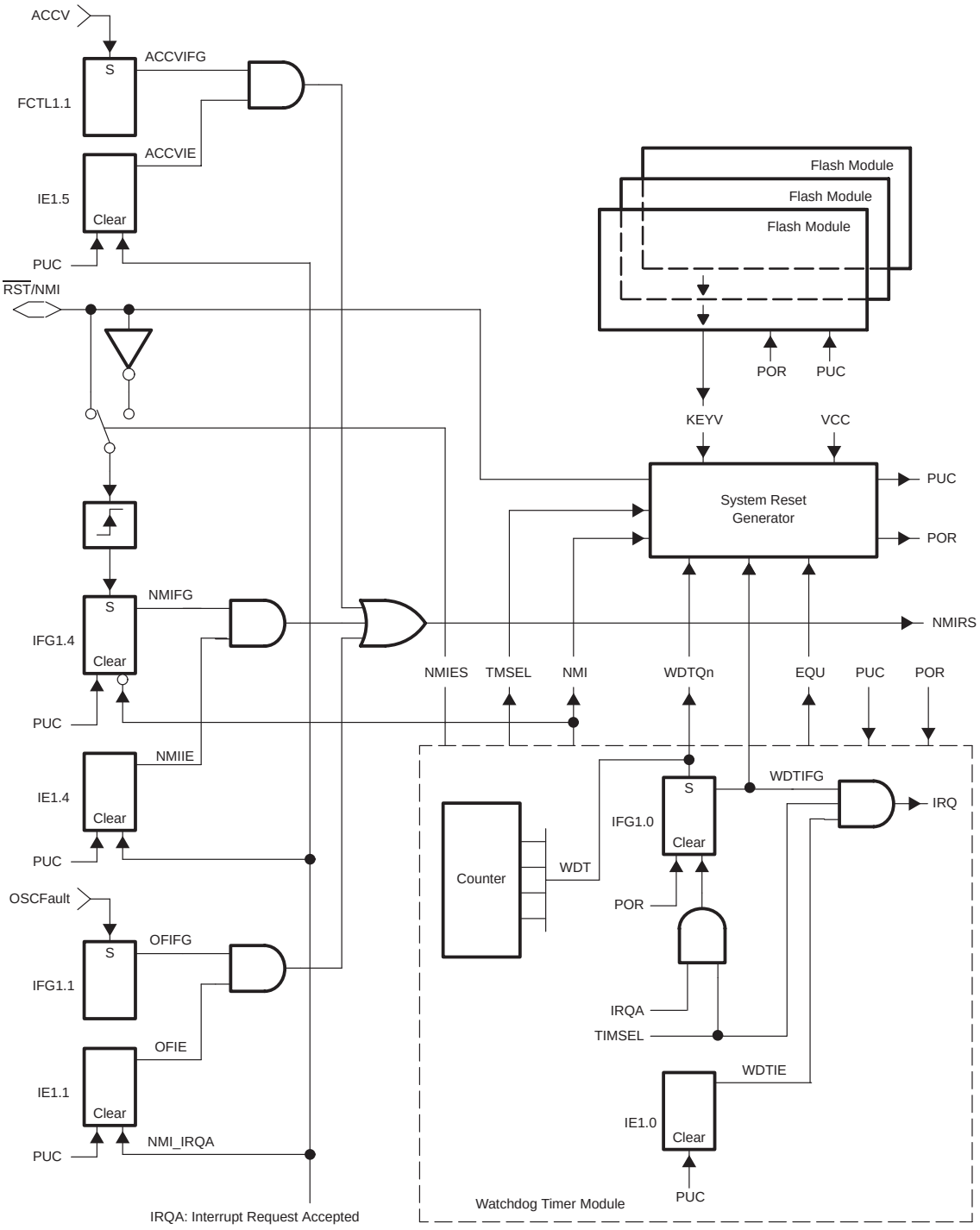


Figure 3–6. $\overline{RST}$/NMI Mode Selection

BITS 0–4,7 See Watchdog Timer chapter.

BIT 5: The NMI bit selects the function of the $\overline{RST}$/NMI input pin. It is cleared after a PUC signal.

NMI = 0: The $\overline{RST}$/NMI input works as reset input. As long as the $\overline{RST}$/NMI pin is held low, the internal PUC signal is active (level-sensitive).

NMI = 1: The $\overline{RST}$/NMI input works as an edge-sensitive, nonmaskable interrupt input.

BIT 6: This bit selects the activating edge of the $\overline{RST}$/NMI input if the NMI function is selected. It is cleared after a PUC signal.

NMIES = 0: A rising edge triggers an NMI interrupt.

NMIES = 1: A falling edge triggers an NMI interrupt.

3.3.1 Operation of Global Interrupt—Reset/NMI

If the $\overline{RST}$/NMI pin is set to the reset function, the CPU is held in the reset state as long as the $\overline{RST}$/NMI pin is held low. After the input changes to a high state, the CPU starts program execution at the word address stored in word location 0FFFEh (reset vector).

If the $\overline{RST}$/NMI pin is set to the NMI function, a signal edge (selected by the NMIES bit) will generate an interrupt if the NMIIE bit is set. When accepted, program execution begins at the address stored in location 0FFFCh. The $\overline{RST}$/NMI flag in the SFR IFG1.4 is also set.

Note:

When configured in the NMI mode, a signal generating an NMI event should not hold the $\overline{RST}$/NMI pin low, unless it is intended to hold the processor in reset. When an NMI event occurs on the pin, the PUC signal is activated, thus resetting the bits in the WDTCTL register. This results in the $\overline{RST}$/NMI pin being configured in the reset mode. If the signal on the $\overline{RST}$/NMI pin that generated the NMI event remains low, the processor will be held in the reset state.

When NMI mode is selected and the NMI edge select bit is changed, an NMI can be generated, depending on the actual level at $\overline{RST}$/NMI pin. When the NMI edge select bit is changed before selecting the NMI mode, no NMI is generated.

The NMI interrupt is maskable by the NMIIE bit.

3.3.2 Operation of Global Interrupt—Oscillator Fault Control

The oscillator fault signal warns of a possible error condition with the crystal oscillator.

3.3.2.1 Oscillator Fault Control in the Basic Clock System

The oscillator-fault signal is triggered when the LFXT1 oscillator is configured to run in HF mode but is not running, stops running after being operational, or is switched off. The oscillator-fault signal is also triggered under the same conditions for the XT2 oscillator, present in some devices. Note that a PUC signal can trigger an oscillator fault, because the PUC switches the LFXT1 to LF mode, therefore switching off the HF mode. The PUC signal also switches off the XT2 oscillator.

The oscillator fault signal can be enabled to generate an NMI by bit OFIE in the SFRs. The interrupt flag OFIFG in the SFRs can then be tested by the interrupt service routine to determine if the NMI was caused by an oscillator fault. See Basic Clock Module chapter for more details on the operation of the crystal oscillators LFXT1 and XT2.

3.4 Interrupt Processing

The MSP430 programmable interrupt structure allows flexible on-chip and external interrupt configurations to meet real-time interrupt-driven system requirements. Interrupts may be initiated by the processor's operating conditions such as watchdog overflow; or by peripheral modules or external events. Each interrupt source can be disabled individually by an interrupt enable bit, or all maskable interrupts can be disabled by the general interrupt enable (GIE) bit in the status register.

Whenever an interrupt is requested and the appropriate interrupt enable bit and general interrupt enable (GIE) bit are set, the interrupt service routine becomes active as follows:

1) CPU active: The currently executing instruction is completed.

2) CPU stopped: The low-power modes are terminated.

3) The program counter pointing to the next instruction is pushed onto the stack.

4) The status register is pushed onto the stack.

5) The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.

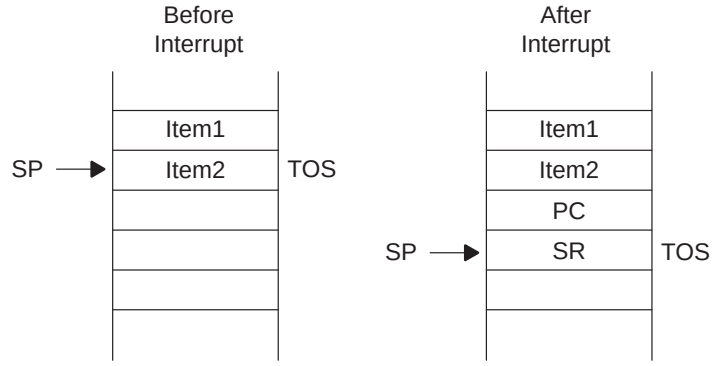
6) The appropriate interrupt request flag resets automatically on single-source flags. Multiple source flags remain set for servicing by software.

7) The GIE bit is reset; the CPUOff bit, the OscOff bit, and the SCG1 bit are cleared; the status bits V, N, Z, and C are reset. SCG0 is left unchanged.

8) The content of the appropriate interrupt vector is loaded into the program counter: the program continues with the interrupt handling routine at that address.

The interrupt latency is six cycles, starting with the acceptance of an interrupt request, and lasting until the start of execution of the appropriate interrupt-service routine first instruction, as shown in Figure 3–7.

Figure 3–7. Interrupt Processing



The interrupt handling routine terminates with the instruction:

`RETI` (return from an interrupt service routine)

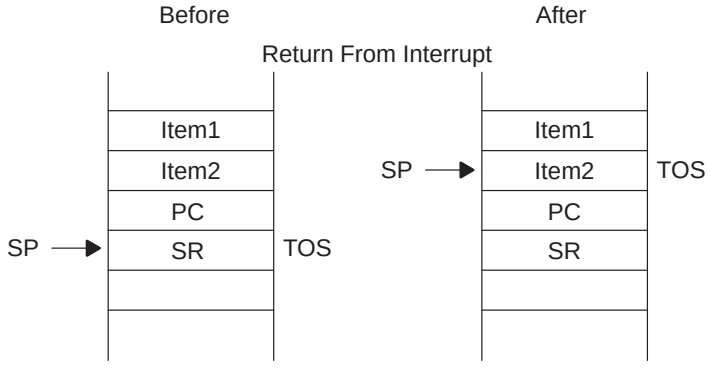
which performs the following actions:

1) The status register with all previous settings pops from the stack. All previous settings of GIE, CPUOFF, etc. are now in effect, regardless of the settings utilized during the interrupt service routine.

2) The program counter pops from the stack and begins execution at the point where it was interrupted.

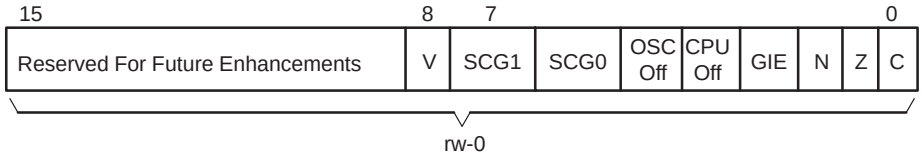
The return from the interrupt is illustrated in Figure 3–8.

Figure 3–8. Return From Interrupt



A RETI instruction takes five cycles. Interrupt nesting is activated if the GIE bit is set inside the interrupt handling routine. The GIE bit is located in status register SR/R2, which is included in the CPU as shown in Figure 3–9.

Figure 3–9. Status Register (SR)



Apart from the GIE bit, other sources of interrupt requests can be enabled/ disabled individually or in groups. The interrupt enable flags are located together within two addresses of the special-function registers (SFRs). The program-flow conditions on interrupt requests can be easily adjusted using the interrupt enable masks. The hardware serves the highest priority within the empowered interrupt source.

3.4.1 Interrupt Control Bits in Special-Function Registers (SFRs)

Most of the interrupt control bits, interrupt flags, and interrupt enable bits are collected in SFRs under a few addresses, as shown in Table 3–1. The SFRs are located in the lower address range and are implemented in byte format. SFRs must be accessed using byte instructions.

Table 3–1. Interrupt Control Bits in SFRs

Address	7 0
000Fh	Not yet defined or implemented
000Eh	Not yet defined or implemented
000Dh	Not yet defined or implemented
000Ch	Not yet defined or implemented
000Bh	Not yet defined or implemented
000Ah	Not yet defined or implemented
0009h	Not yet defined or implemented
0008h	Not yet defined or implemented
0007h	Not yet defined or implemented
0006h	Not yet defined or implemented
0005h	Module enable 2 (ME2.x)
0004h	Module enable 1 (ME1.x)
0003h	Interrupt flag reg. 2 (IFG2.x)
0002h	Interrupt flag reg. 1 (IFG1.x)
0001h	Interrupt enable 2 (IE2.x)
0000h	Interrupt enable 1 (IE1.x)

The MSP430 family supports SFRs by applying the correct logic and functions to each individual module. Each module interrupt source can be individually enabled or disable using the bits described in Table 3–2.

The interrupt-flag registers are described in Table 3–3. The module-enable bits are described in Table 3–4.

Table 3–2. Interrupt Enable Registers 1 and 2

Bit Position	Short Form	Initial State†	Comments
IE1.0	WDTIE	Reset	Watchdog timer enable signal. Inactive if watchdog mode is selected. Active if watchdog timer is configured as general-purpose timer.
IE1.1	OFIE	Reset	Oscillator fault interrupt enable
IE1.2			Not implemented
IE1.3			Not implemented
IE1.4	NMIIE	Reset	NMI interrupt enable
IE1.5	ACCVIE	Reset	Flash access violation enable
IE1.6	URXIE0	Reset	USART0 receive interrupt enable (13x, 14x devices)
IE1.7	UTXIE0	Reset	USART0 transmit interrupt enable (13x, 14x devices)
IE2.0			Not implemented
IE2.1			Not implemented
IE2.2			Not implemented
IE2.3			Not implemented
IE2.4	URXIE1	Reset	USART1 receive interrupt enable (14x devices)
IE2.5	UTXIE1	Reset	USART1 transmit interrupt enable (14x devices)
IE2.6			Not implemented
IE2.7			Not implemented

† The initial state is the logical state after the PUC signal.

Table 3–3. Interrupt Flag Register 1 and 2

Bit Position	Short Form	Initial State	Comments
IFG1.0	WDTIFG	Set	Set on watchdog timer overflow in watchdog mode or security key violation.
		Or reset	Reset with VCC power-up, or a reset condition at the $\overline{\text{RST}}$/NMI pin in reset mode.
IFG1.1	OFIFG	Set	Flag set on oscillator fault
IFG1.2			Not implemented
IFG1.3			Not implemented
IFG1.4	NMIIFG	Reset	Set through the $\overline{\text{RST}}$/NMI pin
IFG1.5			Not implemented
IFG1.6	URXIFG0	Reset	USART0 receive flag (13x, 14x devices)
IFG1.7	UTXIFG0	Set	USART0 transmitter ready (13x, 14x devices)
IFG2.0			Not implemented
IFG2.1			Not implemented
IFG2.2			Not implemented
IFG2.3			Not implemented
IFG2.4	URXIFG1	Reset	USART1 receive flag (14x devices)
IFG2.5	UTXIFG1	Set	USART1 transmitter ready (14x devices)
IFG2.6			Not implemented
IFG2.7			Not implemented

Note: The configuration of some MSP430 devices may differ slightly from those in above table. Refer to specific device data sheets for individual configurations.

Table 3–4. Module Enable Registers 1 and 2

Bit Position	Short Form	Initial State	Comments
ME1.0			Reserved
ME1.1			Reserved
ME1.2			Reserved
ME1.3			Reserved
ME1.4			Reserved
ME1.5			Reserved
ME1.6	URXE0 USPIE0	Reset Reset	USART0 receiver enable (UART mode) USART0 transmit and receive enable (SPI mode)
ME1.7	UTXE0	Reset	USART0 transmit enable (UART mode)
ME2.0			Reserved
ME2.1			Reserved
ME2.2			Reserved
ME2.3			Reserved
ME2.4	URXE1 USPIE1	Reset Reset	USART1 receiver enable (UART mode) USART1 transmit and receive enable (SPI mode)
ME2.5	UTXE1	Reset	USART1 transmit enable (UART mode)
ME2.6			Reserved
ME2.7			Reserved

Note: The configuration of some MSP430 devices may differ slightly from those in above table. Refer to specific device data sheets for individual configurations.

3.4.2 Interrupt Vector Addresses

The interrupt vectors and the power-up starting address are located in the ROM, using the address range 0FFFFh – 0FFE0h as described in Table 3–5. The vector contains the 16-bit address of the appropriate interrupt handler instruction sequence. The interrupt vectors for 14x devices are shown in Table 3–5 in decreasing order of priority. See device data sheet for interrupt vectors for a specific device.

Table 3–5. Interrupt Sources, Flags, and Vectors of 14x Configurations

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-up External Reset Watchdog Flash memory	WDTIFG KEYV (see Note 1)	Reset	0FFFEh	15, highest
NMI Oscillator Fault Flash memory access violation	NMIIFG (see Notes 1 & 3) OFIFG (see Notes 1 & 3) ACCVIFG (see Notes 1 & 3)	(Non)maskable (Non)maskable (Non)maskable	0FFFCh	14
Timer_B7	BCCIFG0 (see Note 2)	Maskable	0FFFAh	13
Timer_B7	BCCIFG1 to BCCIFG6 TBIFG (see Notes 1 & 2)	Maskable	0FFF8h	12
Comparator_A	CMPAIFG	Maskable	0FFF6h	11
Watchdog timer	WDTIFG	Maskable	0FFF4h	10
USART0 receive	URXIFG0	Maskable	0FFF2h	9
USART0 transmit	UTXIFG0	Maskable	0FFF0h	8
ADC	ADCIFG (see Notes 1 & 2)	Maskable	0FFEEh	7
Timer_A3	CCIFG0 (see Note 2)	Maskable	0FFECh	6
Timer_A3	CCIFG1, CCIFG2, TAIFG (see Notes 1 & 2)	Maskable	0FFEAh	5
I/O port P1 (eight flags)	P1IFG.0 (see Notes 1 & 2) To P1IFG.7 (see Notes 1 & 2)	Maskable	0FFE8h	4
USART1 receive	URXIFG1	Maskable	0FFE6h	3
USART1 transmit	UTXIFG1		0FFE4h	2
I/O port P2 (eight flags)	P2IFG.0 (see Notes 1 & 2) To P2IFG.7 (see Notes 1 & 2)	Maskable	0FFE2h	1
			0FFE0h	0, lowest

NOTES: 1. Multiple source flags
2. Interrupt flags are located in the module.
3. (Non)maskable: the individual interrupt-enable bit can disable an interrupt event, but the general-interrupt enable can not disable it.

Note: Some MSP430 devices have different implementations, See device datasheet for details.

3.4.2.1 External Interrupts

All eight bits of ports P1 and P2 are designed for interrupt processing of external events. All individual I/O bits are independently programmable. Any combinations of inputs, outputs, and interrupt conditions are possible. This allows easy adaptation to different I/O configurations. See Chapter I/O Ports for more details on I/O ports.

3.5 Operating Modes

The MSP430 family was developed for ultralow-power applications and uses different levels of operating modes. The MSP430 operating modes, shown in Figure 3–10, give advanced support to various requirements for ultralow power and ultralow energy consumption. This support is combined with an intelligent management of operations during the different module and CPU states. An interrupt event wakes the system from each of the various operating modes and the RETI instruction returns operation to the mode that was selected before the interrupt event.

The ultra-low power system design which uses complementary metal-oxide semiconductor (CMOS) technology, takes into account three different needs:

- The desire for speed and data throughput despite conflicting needs for ultra-low power
- Minimization of individual current consumption
- Limitation of the activity state to the minimum required by the use of low power modes

There are four bits that control the CPU and the main parts of the operation of the system clock generator: CPUOff, OscOff, SCG0, and SCG1. These four bits support discontinuous active mode (AM) requests, to limit the time period of the full operating mode, and are located in the status register. The major advantage of including the operating mode bits in the status register is that the present state of the operating condition is saved onto the stack during an interrupt service request. As long as the stored status register information is not altered, the processor continues (after RETI) with the same operating mode as before the interrupt event. Another program flow may be selected by manipulating the data stored on the stack or the stack pointer. Being able to access the stack and stack pointer with the instruction set allows the program structures to be individually optimized, as illustrated in the following program flow:

- Enter interrupt routine

The interrupt routine is entered and processed if an enabled interrupt awakens the MSP430:

- The SR and PC are stored on the stack, with the content present at the interrupt event.
- Subsequently, the operation mode control bits OscOff, SCG1, and CPUOff are cleared automatically in the status register.

❑ Return from interrupt

Two different modes are available to return from the interrupt service routine and continue the flow of operation:

- Return with low-power mode bits set. When returning from the interrupt, the program counter points to the next instruction. The instruction pointed to is not executed, since the restored low power mode stops CPU activity.
- Return with low-power mode bits reset. When returning from the interrupt, the program continues at the address following the instruction that set the OscOff or CPUOff-bit in the status register. To use this mode, the interrupt service routine must reset the OscOff, CPUOff, SCGO, and SCG1 bits on the stack. Then, when the SR contents are popped from the stack upon RETI, the operating mode will be active mode (AM).

There are six operating modes that the software can configure:

❑ Active mode AM; SCG1=0, SCG0=0, OscOff=0, CPUOff=0:
CPU clocks are active

❑ Low power mode 0 (LPM0); SCG1=0, SCG0=0, OscOff=0, CPUOff=1:
CPU is disabled
MCLK is disabled
SMCLK and ACLK remain active

❑ Low power mode 1 (LPM1); SCG1=0, SCG0=1, OscOff=0, CPUOff=1:
CPU is disabled
MCLK is disabled
DCO's dc generator is disabled if the DCO is not used for MCLK or SMCLK when in active mode. Otherwise, it remains enabled.
SMCLK and ACLK remain active

- Low power mode 2 (LPM2); SCG1=1, SCG0=0, OscOff=0, CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - SMCLK is disabled
  - DCO oscillator automatically disabled because it is not needed for MCLK or SMCLK
  - DCO's dc-generator remains enabled
  - ACLK remains active

- Low power mode 3 (LPM3); SCG1=1, SCG0=1, OscOff=0, CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - SMCLK is disabled
  - DCO oscillator is disabled
  - DCO's dc-generator is disabled
  - ACLK remains active

- Low power mode 4 (LPM4); SCG1=X, SCG0=X, OscOff=1, CPUOff=1:
  - CPU is disabled
  - ACLK is disabled
  - MCLK is disabled
  - SMCLK is disabled
  - DCO oscillator is disabled
  - DCO's dc-generator is disabled
  - Crystal oscillator is stopped

Note:

Peripheral operation is not halted by CPUOff. Peripherals are controlled by their individual control registers.

Table 3–6. Low Power Mode Logic Chart for Basic Clock System

	SCG1	SCG0	OscOff	CPUOff
LPM0	0	0	0	1
LPM1	0	1	0	1
LPM2	1	0	0	1
LPM3	1	1	0	1
LPM4	1	1	1	1

These modes are illustrated in Figure 3–11.

Figure 3–10. MSP430x1xx Operating Modes For Basic Clock System

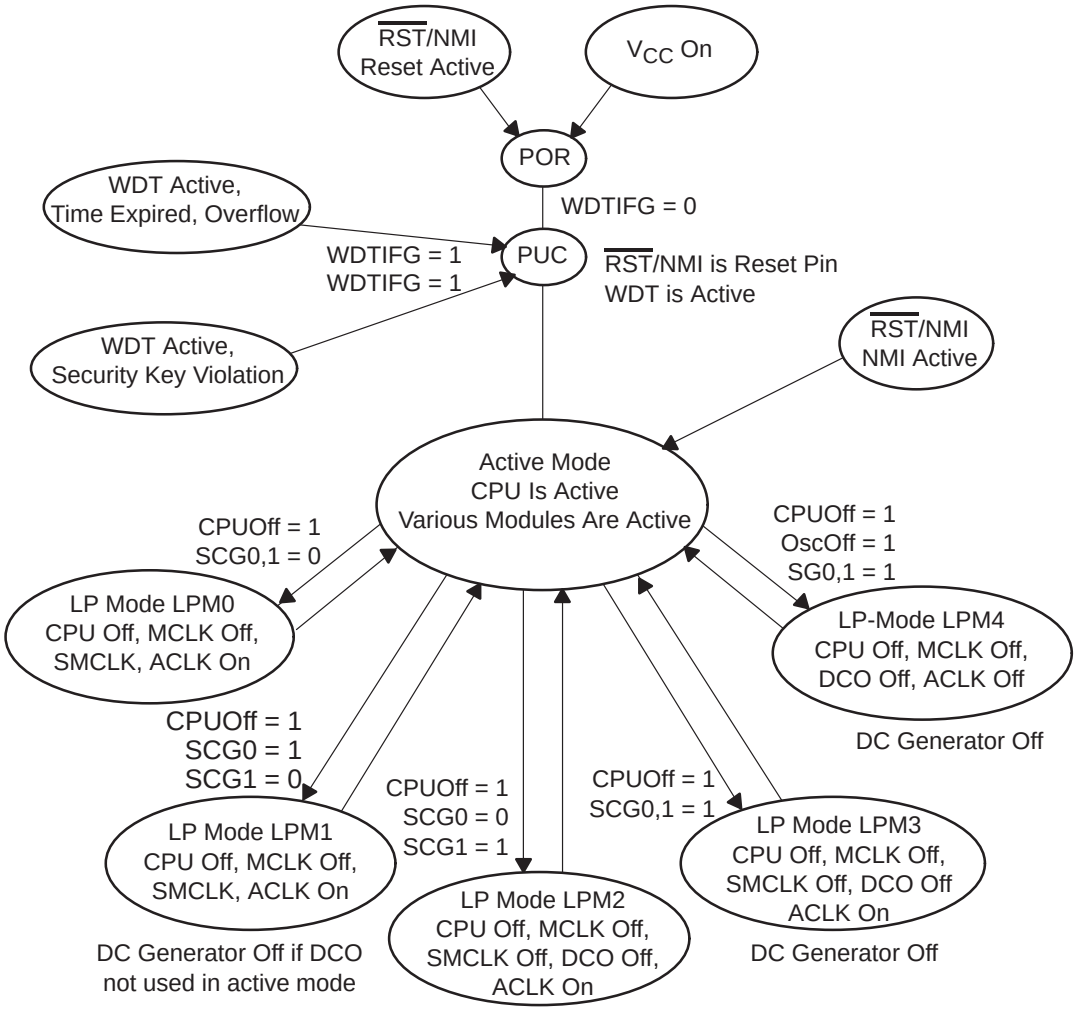
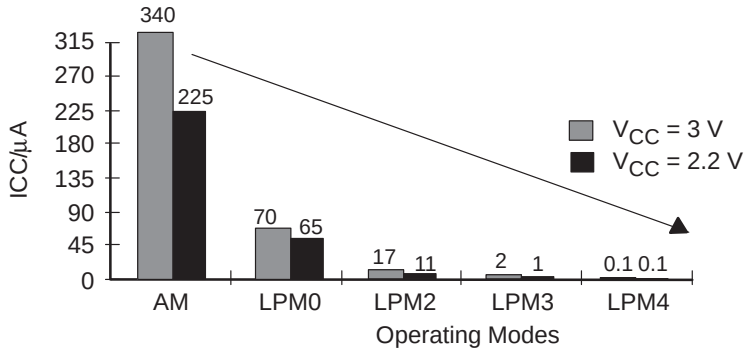


Figure 3–11. Typical Current Consumption of 13x and 14x Devices vs. Operating Modes



The low-power modes 1–4 enable or disable the CPU and the clocks. In addition to the CPU and clocks, enabling or disabling specific peripherals may further reduce total current consumption of the individual modes. The activity state of each peripheral is controlled by the control registers for the individual peripherals. In addition, the SFRs include module enable bits that may be used to enable or disable the operation of specific peripheral modules (see Table 3–4).

3.5.1 Low-Power Mode 0 and 1 (LPM0 and LPM1)

Low power mode 0 or 1 is selected if bit CPUOff in the status register is set. Immediately after the bit is set the CPU stops operation, and the normal operation of the system core stops. The operation of the CPU halts and all internal bus activities stop until an interrupt request or reset occurs. The system clock generator continues operation, and the clock signals MCLK, SMCLK, and ACLK stay active depending on the state of the other three status register bits, SCG0, SCG1, and OscOff.

The peripherals are enabled or disabled with their individual control register settings, and with the module enable registers in the SFRs. All I/O port pins and RAM/registers are unchanged. Wake up is possible through all enabled interrupts.

The following are examples of entering and exiting LPM0. The method shown is applicable to all low-power modes.

The following example describes entering into low-power mode 0.

```
;===Main program flow with switch to CPUOff Mode==============
;
     BIS #18h,SR   ;Enter LPM0 + enable general interrupt GIE
                   ;(CPUOff=1, GIE=1). The PC is incremented
                   ;during execution of this instruction and
                   ;points to the consecutive program step.
     ......        ;The program continues here if the CPUOff
                   ;bit is reset during the interrupt service
                   ;routine. Otherwise, the PC retains its
                   ;value and the processor returns to LPM0.
```

The following example describes clearing low-power mode 0.

```
;===Interrupt service routine================================
    ......          ;CPU is active while handling interrupts
    BIC #10h,0(SP)  ;Clears the CPUOff bit in the SR contents
                    ;that were stored on the stack.

    RETI            ;RETI restores the CPU to the active state
                    ;because the SR values that are stored on
                    ;the stack were manipulated. This occurs
                    ;because the SR is pushed onto the stack
                    ;upon an interrupt, then restored from the
                    ;stack after the RETI instruction.
```

3.5.2 Low-Power Modes 2 and 3 (LPM2 and LPM3)

Low-power mode 2 or 3 is selected if bits CPUOff and SCG1 in the status register are set. Immediately after the bits are set, CPU, MCLK, and SMCLK operations halt and all internal bus activities stop until an interrupt request or reset occurs.

Peripherals that operate with the MCLK or SMCLK signal are inactive because the clock signals are inactive. Peripherals that operate with the ACLK signal are active or inactive according with the individual control registers and the module enable bits in the SFRs. All I/O port pins and the RAM/registers are unchanged. Wake up is possible by enabled interrupts coming from active peripherals or $\overline{\text{RST}}$/NMI.

3.5.3 Low-Power Mode 4 (LPM4)

In low power mode 4 all activities cease; only the RAM contents, I/O ports, and registers are maintained. Wake up is only possible by enabled external interrupts.

Before activating LPM4, the software should consider the system conditions during the low power mode period . The two most important conditions are environmental (that is, temperature effect on the DCO), and the clocked operation conditions.

The environment defines whether the value of the frequency integrator should be held or corrected. A correction should be made when ambient conditions are anticipated to change drastically enough to increase or decrease the system frequency while the device is in LPM4.

3.6 Basic Hints for Low-Power Applications

There are some basic practices to follow when current consumption is a critical part of a system application:

- ❑ Switch off analog circuitry when possible.
- ❑ Select the lowest possible operating frequency for the core and the individual peripheral module.
- ❑ Use the interrupt driven software; the program starts execution rapidly.
- ❑ Tie all unused inputs to an applicable voltage level. The list below defines the correct termination for all unused pins.

Pin	Potential	Comment
❑ **AV_{CC}:**	DV_{CC}	
❑ **AV_{SS}:**	DV_{SS}	
❑ **Xout:**	open	
❑ **XIN**	DV_{SS}	
❑ **XT2IN**	DV_{SS}	13x and 14x devices
❑ **XT2OUT**	open	13x and 14x devices
❑ **Px.0 to Px.7:**	open	Unused ports switched to port function and output direction
❑ **RST/NMI**:	DV_{CC} resp.V_{CC}	Pullup resistor 100k
❑ **Test/V_{PP}**	DV_{SS}	11x devices
❑ **Test**	DV_{SS}	11x1 devices
❑ **TDO:**	Refer to device specific datasheets for the correct termination of these pins.	
❑ **TDI:**		
❑ **TMS:**		
❑ **TCK:**		

Chapter 4

Memory

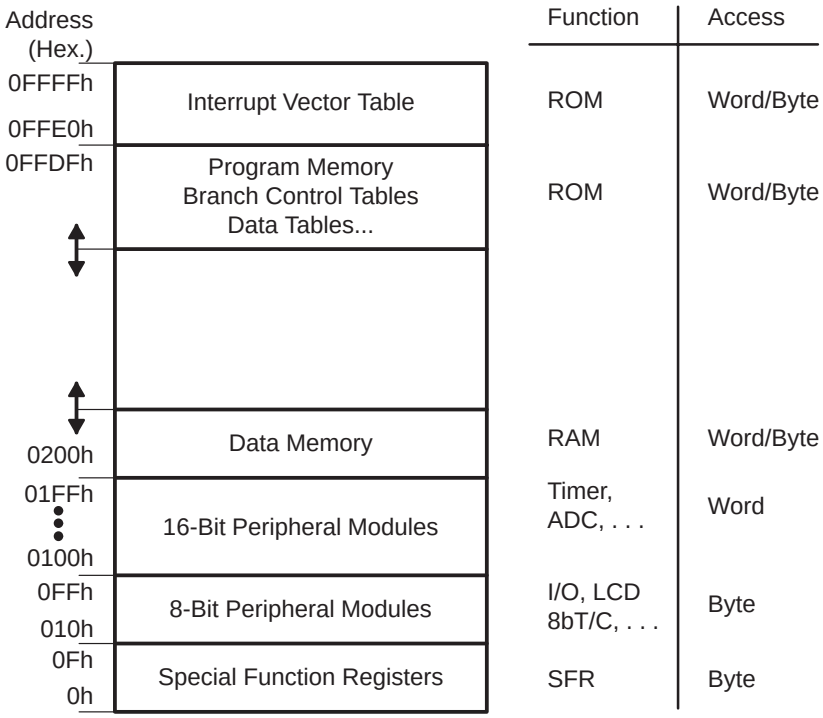
MSP430 devices are configured as a von-Neumann architecture. It has code memory, data memory, and peripherals in one address space. As a result, the same instructions are used for code, data, or peripheral accesses. Also, code may be executed from RAM.

Topic **Page**

4.1 Introduction

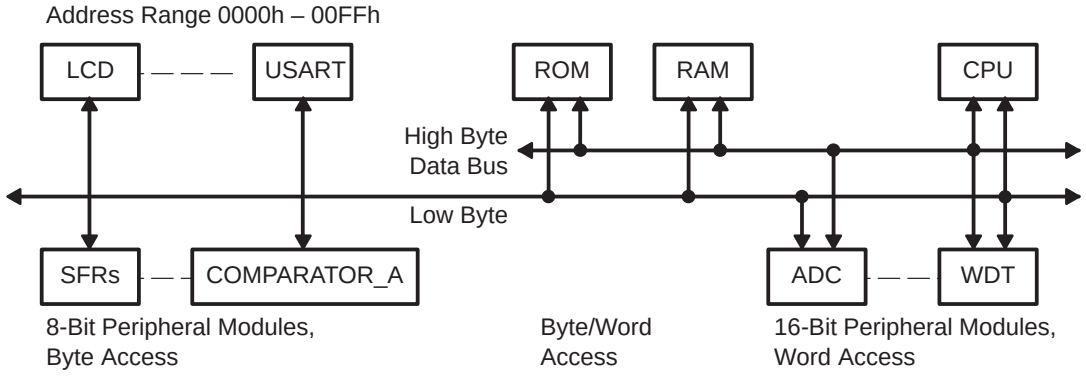
All of the physically separated memory areas (ROM, RAM, SFRs, and peripheral modules) are mapped into the common address space, as shown in Figure 4–1 for the MSP430 family. The addressable memory space is 64KB. Future expansion is possible.

Figure 4–1. Memory Map of Basic Address Space



The memory data bus (MDB) is 16- or 8-bits wide. For those modules that can be accessed with word data the width is always 16 bits. For the other modules, the width is 8 bits, and they must be accessed using byte instructions only. The program memory (ROM) and the data memory (RAM) can be accessed with byte or word instructions.

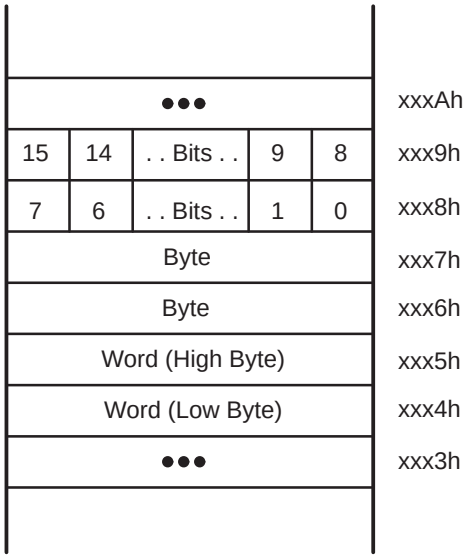
Figure 4–2. Memory Data Bus



4.2 Data in the Memory

Bytes are located at even or odd addresses as shown in Figure 4–3. However, words are only located at even addresses. Therefore, when using word instructions, only even addresses may be used. The low byte of a word is always at an even address. The high byte of a word is at the next odd address after the address of the word. For example, if a data word is located at address xxx2h, then the low byte of that data word is located at address xxx2h, and the high byte of that word is located at address xxx3h.

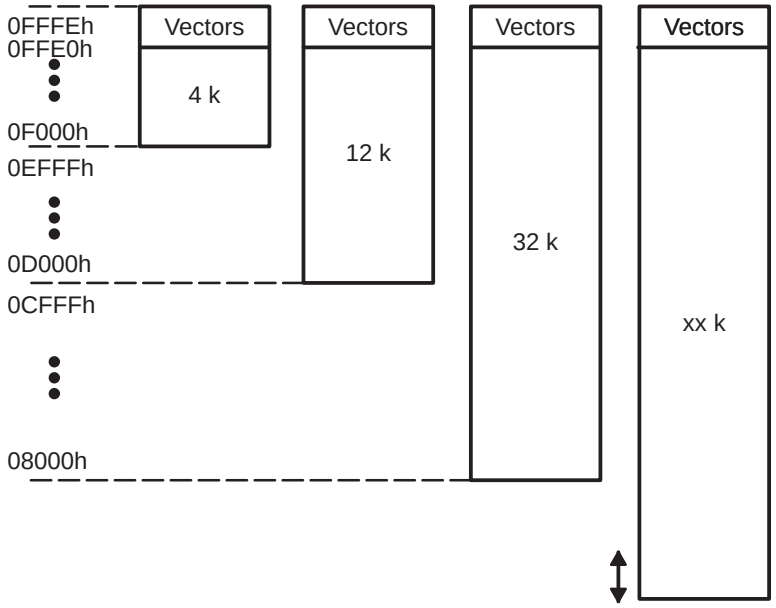
Figure 4–3. Bits, Bytes, and Words in a Byte-Organized Memory



4.3 Internal ROM Organization

Various sizes of ROM (OTP, masked-ROM, EPROM, or FLASH) are available within the 64-kB address space, as shown in Figure 4–4. The common address space is shared with SFRs, peripheral module registers, data and code memory. The SFRs and peripheral modules are mapped into the address range, starting with 0 and ending with 01FFh. The remaining address space, 0200h to 0FFFFh, is shared by data and code memory. The start address for ROM depends on the amount of ROM present. The interrupt vector table is mapped into the the upper 16 words of ROM address space, with the highest priority interrupt vector at the highest ROM word address (0FFFEh). See the individual data sheets for specific memory maps.

Figure 4–4. ROM Organization



4.3.1 Processing of ROM Tables

The MSP430 architecture allows for the storage and usage of large tables in ROM without the need to copy the tables to RAM before using them. This ROM accessing of tables allows fast and clear programming in applications where data tables are necessary. This offers the flexible advantages listed below, and saves on ROM and RAM requirements. To access these tables, all word and byte instructions can be used.

- ROM storage of an output programmable logic array (OPLA) for display character conversion
- The use of as many OPLA terms as needed (no restriction on n terms)
- OTP version automatically includes OPLA programmability
- Computed table accessibility (for example, for a bar graph display)
- Table-supported program flows

4.3.2 Computed Branches and Calls

Computed branches and subroutine calls are possible using standard instructions. The call and branch instructions use the same addressing modes as the other instructions.

The addressing modes allow indirect-indirect addressing that is ideally suited for computed branches and calls. This programming technique permits a program structure that is different from conventional 8- and 16-bit microcontrollers. Most of the routines can be handled easily by using software status handling instead of flag-type program-flow control.

The computed branch and subroutine calls are valid throughout the entire ROM space.

4.4 RAM and Peripheral Organization

The entire RAM can be accessed with byte or word instructions using the appropriate instruction suffix. The peripheral modules, however, are located in two different address spaces and must be accessed with the appropriate instruction length.

- The SFRs are byte-oriented and mapped into the address space from 0h up to 0Fh.
- Peripheral modules that are byte-oriented are mapped into the address space from 010h up to 0FFh.
- Peripheral modules that are word-oriented are mapped into the address space from 100h up to 01FFh.

4.4.1 Random Access Memory

RAM can be used for both code and data memory. Code accesses are always performed on even byte addresses.

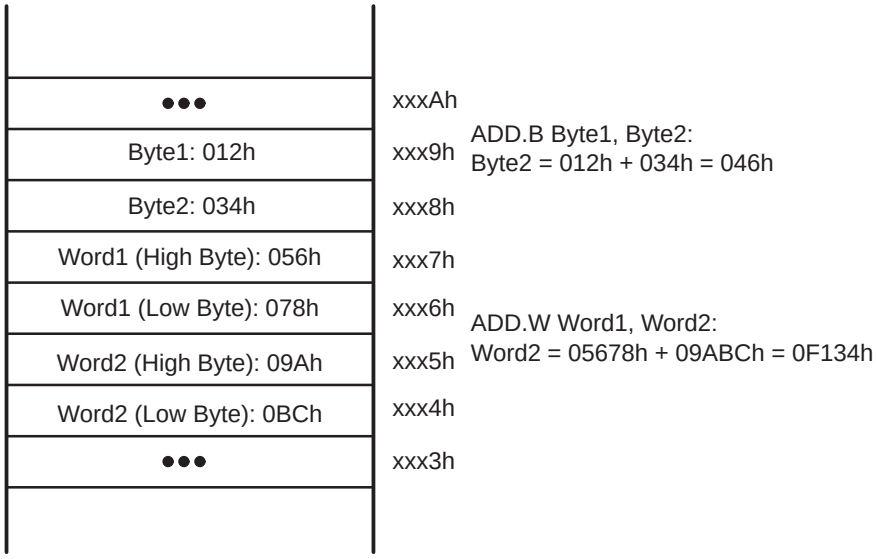
The instruction mnemonic suffix defines the data as being word or byte data.

Example:

```
MOV.B   TXDATA,&UTXBUF0                         ;Byte access
ADD     R5,SUM_A        =   ADD.W   R5,SUM_A    ;Word access
ADDC    SUM_B           =   ADDC.W  SUM_A       ;Word access
```

A word consists of two bytes: a high byte (bit 15 to bit 8), and a low byte (bit 7 to bit 0) as shown in Figure 4–5. It must always align to an even address.

Figure 4–5. Byte and Word Operation



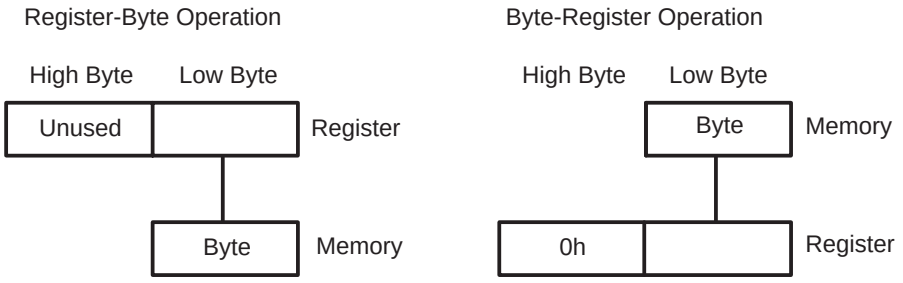
All operations on the stack and PC are word operations and use even-aligned memory addresses.

In the following examples, word-to-word and byte-to-byte operations show the results of the operation and the status bit information.

Example Word-Word Operation	**Example Byte-Byte Operation**
R5 = 0F28Eh	R5 = 0223h
EDE .EQU 0212h	EDE .EQU 0202h
Mem(0F28Eh) = 0FFFEh	Mem(0223h) = 05Fh
Mem(0212h) = 00112h	Mem(0202h) = 043h
ADD @R5,&EDE	ADD.B @R5,&EDE
Mem(0212h) = 00110h	Mem(0202h) = 0A2h
C = 1, Z = 0, N = 0	C = 0, Z = 0, N = 1

Figure 4–6 shows the register-byte and byte-register operations.

Figure 4–6. Register-Byte/Byte-Register Operations



The following examples describe the register-byte and byte-register operations.

Example Register-Byte Operation	**Example Byte-Register Operation**
R5 = 0A28Fh	R5 = 01202h
R6 = 0203h	R6 = 0223h
Mem(0203h) = 012h	Mem(0223h) = 05Fh
ADD.B R5,0(R6)	ADD.B @R6,R5
08Fh	05Fh
+ 012h	+ 002h ;Low byte of R5
0A1h	00061h ;–>Store into R5
	;High byte is 0
Mem (0203h) = 0A1h	R5 = 00061h
C = 0, Z = 0, N = 1	C = 0, Z = 0, N = 0
(Low byte of register)	(Addressed byte)
+ (Addressed byte)	+ (Low byte of register)
–>(Addressed byte)	–>(Low byte of register, zero to High byte)

Note: Word-Byte Operations

Word-byte or byte-word operations on memory data are not supported. Each register-byte or byte-register is performed as a byte operation.

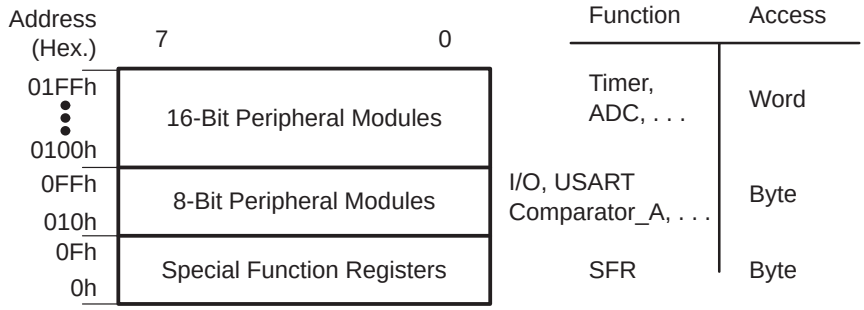
4.4.2 Peripheral Modules—Address Allocation

Some peripheral modules are accessible only with byte instructions, while others are accessible only with word instructions. The address space from 0100 to 01FFh is reserved for word modules, and the address space from 00h to 0FFh is reserved for byte modules.

Peripheral modules that are mapped into the word address space must be accessed using word instructions (for example, MOV R5,&WDTCTL). Peripheral modules that are mapped into the byte address space must be accessed with byte instructions (MOV.B #1,&P1OUT).

The addressing of both is through the absolute addressing mode or the 16-bit working registers using the indexed, indirect, or indirect autoincrement addressing mode. See Figure 4–7 for the RAM/peripheral organization.

Figure 4–7. Example of RAM/Peripheral Organization



4.4.2.1 Word Modules

Word modules are peripherals that are connected to the 16-bit MDB.

Word modules can be accessed with word or byte instructions. If byte instructions are used, only even addresses are permissible, and the high byte of the result is always ’0’.

The peripheral file address space is organized into sixteen frames with each frame representing eight words as described in Table 4–1.

Table 4–1. Peripheral File Address Map—Word Modules

Address	Description
1F0h – 1FFh	Reserved
1E0h – 1EFh	Reserved
1D0h – 1DFH	Reserved
1C0h – 1CFH	Reserved
1B0h – 1BFH	Reserved
1A0h – 1AFH	ADC12 control and interrupt
190h – 19FH	Timer_B
180h – 18FH	Timer_B
170h – 17FH	Timer_A
160h – 16FH	Timer_A
150h – 15FH	ADC12 conversion
140h – 14FH	ADC12 conversion
130h – 13FH	Multiplier
120h – 12FH	Watchdog Timer, Flash control
110h – 11FH	Reserved
100h – 10FH	Reserved

4.4.2.2 Byte Modules

Byte modules are peripherals that are connected to the reduced (eight LSB) MDB. Access to byte modules is always by byte instructions. The hardware in the peripheral byte modules takes the low byte (the LSBs) during a write operation.

Byte instructions operate on byte modules without any restrictions. Read access to peripheral byte modules using word instructions results in unpredictable data in the high byte. Word data is written into a byte module by writing the low byte to the appropriate peripheral register and ignoring the high byte.

The peripheral file address space is organized into sixteen frames as described in Table 4–2.

Table 4–2. Peripheral File Address Map—Byte Modules

Address	Description
00F0h – 00FFh	Reserved
00E0h – 00EFh	Reserved
00D0h – 00DFh	Reserved
00C0h – 00CFh	Reserved
00B0h – 00BFh	Reserved
00A0h – 00AFh	Reserved
0090h – 009Fh	Reserved
0080h – 008Fh	ADC12 memory control
0070h – 007Fh	USART0, USART1
0060h – 006Fh	Reserved
0050h – 005Fh	System clock generator, Comparator A
0040h – 004Fh	Reserved
0030h – 003Fh	Digital I/O port P5, digital I/O port P6
0020h – 002Fh	Digital I/O port P1 and P2 control
0010h – 001Fh	Digital I/O port P3, and P4 control
0000h – 000Fh	Special function

4.4.3 Peripheral Modules-Special Function Registers (SFRs)

The system configuration and the individual reaction of the peripheral modules to the processor operation is configured in the SFRs as described in Table 4–3. The SFRs are located in the lower address range, and are organized by bytes. SFRs must be accessed using byte instructions only.

Table 4–3. Special Function Register Address Map

Address	Data Bus 7 0
000Fh	Not yet defined or implemented
000Eh	Not yet defined or implemented
000Dh	Not yet defined or implemented
000Ch	Not yet defined or implemented
000Bh	Not yet defined or implemented
000Ah	Not yet defined or implemented
0009h	Not yet defined or implemented
0008h	Not yet defined or implemented
0007h	Not yet defined or implemented
0006h	Not yet defined or implemented
0005h	Module enable 2; ME2.2
0004h	Module enable 1; ME1.1
0003h	Interrupt flag reg. 2; IFG2.x
0002h	Interrupt flag reg.1; IFG1.x
0001h	Interrupt enable 2; IE2.x
0000h	Interrupt enable 1; IE1.x

The system power consumption is influenced by the number of enabled modules and their functions. Disabling a module from the actual operation mode reduces power consumption while other parts of the controller remain fully active (unused pins must be tied appropriately or power consumption will increase; see *Basic Hints for Low Power Applications* in section 3.6.

Chapter 5

16-Bit CPU

The MSP430 von-Neumann architecture has RAM, ROM, and peripherals in one address space, both using a single address and data bus. This allows using the same instruction to access either RAM, ROM, or peripherals and also allows code execution from RAM.

Topic **Page**

5.1 CPU Registers

Sixteen 16-bit registers (R0, R1, and R4 to R15) are used for data and addresses and are implemented in the CPU. They can address up to 64 Kbytes (ROM, RAM, peripherals, etc.) without any segmentation. The complete CPU-register set is described in Table 5–1. Registers R0, R1, R2, and R3 have dedicated functions, which are described in detail later.

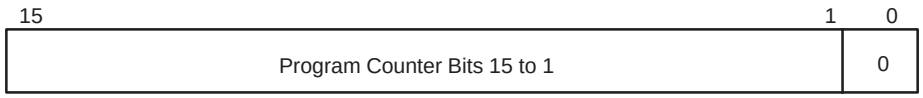
Table 5–1. Register by Functions

Program counter (PC)	R0
Stack pointer (SP)	R1
Status register (SR)	R2
Constant generator (CG1)	
Constant generator (CG2)	R3
Working register R4	R4
Working register R5	R5
: :	: :
Working register R13	R13
Working register R14	R14
Working register R15	R15

5.1.1 The Program Counter (PC)

The 16-bit program counter points to the next instruction to be executed. Each instruction uses an even number of bytes (two, four, or six), and the program counter is incremented accordingly. Instruction accesses are performed on word boundaries, and the program counter is aligned to even addresses. Figure 5–1 shows the program counter bits.

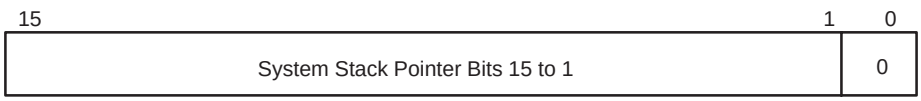
Figure 5–1. Program Counter



5.1.2 The System Stack Pointer (SP)

The system stack pointer must always be aligned to even addresses because the stack is accessed with word data during an interrupt request service. The system SP is used by the CPU to store the return addresses of subroutine calls and interrupts. It uses a predecrement, postincrement scheme. The advantage of this scheme is that the item on the top of the stack is available. The SP can be used by the user software (PUSH and POP instructions), but the user should remember that the CPU also uses the SP. Figure 5–2 shows the system SP bits.

Figure 5–2. System Stack Pointer

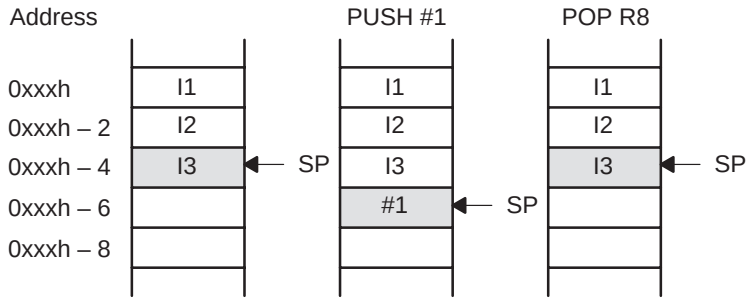


5.1.2.1 Examples for System SP Addressing (Refer to Figure 5–4)

```
MOV     SP,R4       ; SP –> R4
MOV     @SP,R5      ; Item I3 (TOS) –> R5
MOV     2(SP),R6    ; Item I2 –> R6
MOV     R7,0(SP)    ; Overwrite TOS with R7
MOV     R8,4(SP)    ; Modify item I1
PUSH    R12         ; Store R12 in address 0xxxh – 6; SP points to same address
POP     R12         ; Restore R12 from address 0xxxh – 6; SP points to
                      0xxxh – 4
MOV     @SP+,R5     ; Item I3 –> R5 (popped from stack); same as POP
                      instruction
```

Figure 5–3 shows stack usage.

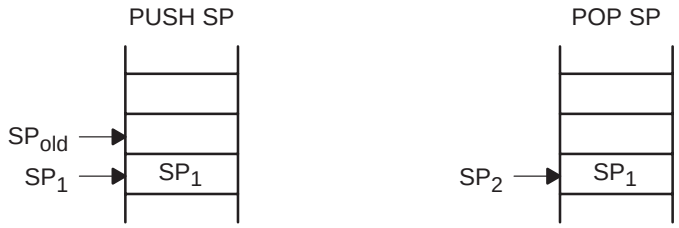
Figure 5–3. Stack Usage



5.1.2.2 Special Cases—PUSH SP and POP SP

The special cases of using the SP as an argument to the PUSH and POP instructions are described below.

Figure 5–4. PUSH SP and POP SP



The stack pointer is changed after a PUSH SP instruction.

The stack pointer is not changed after a POP SP instruction.

After the sequence

```
PUSH SP     ; SP1 is stack pointer after this instruction
  I
  I
POP SP      ; SP2 is stack pointer after this instruction
```

The stack pointer is two bytes lower than before this sequence.

5.1.3 The Status Register (SR)

The status register SR contains the following CPU status bits:

- ❑ V — Overflow bit
- ❑ SCG1 — System clock generator control bit 1
- ❑ SCG0 — System clock generator control bit 0
- ❑ OscOff — Crystal oscillator off bit
- ❑ CPUOff — CPU off bit
- ❑ GIE — General interrupt enable bit
- ❑ N — Negative bit
- ❑ Z — Zero bit
- ❑ C — Carry bit

Figure 5–5 shows the SR bits.

Figure 5–5. Status Register Bits

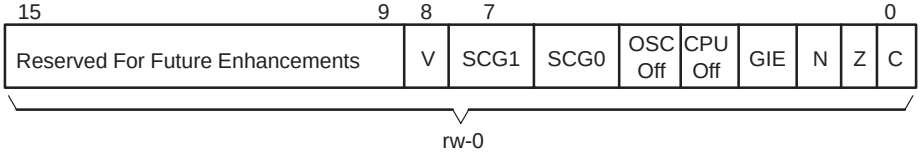


Table 5–2 describes the status register bits.

Table 5–2. Description of Status Register Bits

Bit	Description	
V	Overflow bit. Set if the result of an arithmetic operation overflows the signed-variable range. The bit is valid for both data formats, byte and word:	
	ADD(.B), ADDC(.B)	Set when: Positive + Positive = Negative Negative + Negative = Positive, otherwise reset
	SUB(.B), SUBC(.B), CMP(.B)	Set when: Positive – Negative = Negative Negative – Positive = Positive, otherwise reset
SCG1, SCG0	These bits control four activity states of the system-clock generator and therefore influence the operation of the processor system.	
OscOFF	If set, the crystal oscillator enters off mode: all activities cease; however, the RAM contents, the port, and the registers are maintained. Wake up is possible only through enabled external interrupts when the GIE bit is set and from the NMI.	
CPU Off	If set, the CPU enters off mode: program execution stops. However, the RAM, the port registers, and especially the enabled peripherals (for example, Timer_A, UART, etc.) stay active. Wake up is possible through all enabled interrupts.	
GIE	If set, all enabled maskable interrupts are handled. If reset, all maskable interrupts are disabled. The GIE bit is cleared by interrupts and restored by the RETI instruction as well as by other appropriate instructions.	
N	Set if the result of an operation is negative.	
	Word operation:	Negative bit is set to the value of bit 15 of the result
	Byte operation:	Negative bit is set to the value of bit 7 of the result
Z	Set if the result of byte or word operation is 0; cleared if the result is not 0.	
C	Set if the result of an operation produced a carry; cleared if no carry occurred. Some instructions modify the carry bit using the inverted zero bits.	

Note: Status Register Bits V, N, Z, and C

The status register bits V, N, Z, and C are modified only with the appropriate instruction. For additional information, see the detailed description of the instruction set in Appendix B.

5.1.4 The Constant Generator Registers CG1 and CG2

Commonly-used constants are generated with the constant generator registers R2 and R3, without requiring an additional 16-bit word of program code. The constant used for immediate values is defined by the addressing mode bits (As) as described in Table 5–3. See Section 5.3 for a description of the addressing mode bits (As).

Table 5–3. Values of Constant Generators CG1, CG2

Register	As	Constant	Remarks
R2	00	– – – – –	Register mode
R2	01	(0)	Absolute address mode
R2	10	00004h	+4, bit processing
R2	11	00008h	+8, bit processing
R3	00	00000h	0, word processing
R3	01	00001h	+1
R3	10	00002h	+2, bit processing
R3	11	0FFFFh	–1, word processing

The major advantages of this type of constant generation are:

- No special instructions required
- Reduced code memory requirements: no additional word for the six most used constants
- Reduced instruction cycle time: no code memory access to retrieve the constant

The assembler uses the constant generator automatically if one of the six constants is used as a source operand in the immediate addressing mode. The status register SR/R2, used as a source or destination register, can be used in the register mode only. The remaining combinations of addressing-mode bits are used to support absolute-address modes and bit processing without any additional code. Registers R2 and R3, used in the constant mode, cannot be addressed explicitly; they act like source-only registers.

The RISC instruction set of the MSP430 only has 27 instructions. However, the constant generator allows the MSP430 assembler to support 24 additional, emulated instructions. For example, the single-operand instruction:

```
CLR     dst
```

is emulated by the double-operand instruction with the same length:

```
MOV     R3,dst
or the equivalent
MOV     #0,dst
```

where #0 is replaced by the assembler, and R3 is used with As = 00, which results in:

- One word instruction
- No additional control operation or hardware within the CPU
- Register-addressing mode for source: no additional fetch cycle for the constant (#0)

5.2 Addressing Modes

All seven addressing modes for the source operand and all four addressing modes for the destination operand can address the complete address space. The bit numbers in Table 5–4 describe the contents of the As and Ad mode bits. See Section 5.3 for a description of the source address As and the destination address Ad bits.

Table 5–4. Source/Destination Operand Addressing Modes

As/Ad	Addressing Mode	Syntax	Description
00/0	Register mode	Rn	Register contents are operand
01/1	Indexed mode	X(Rn)	(Rn + X) points to the operand X is stored in the next word
01/1	Symbolic mode	ADDR	(PC + X) points to the operand X is stored in the next word. Indexed mode X(PC) is used.
01/1	Absolute mode	&ADDR	The word following the instruction contains the absolute address.
10/–	Indirect register mode	@Rn	Rn is used as a pointer to the operand.
11/–	Indirect autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards.
11/–	Immediate mode	#N	The word following the instruction contains the immediate constant N. Indirect autoincrement mode @PC+ is used.

The seven addressing modes are explained in detail in the following sections. Most of the examples show the same addressing mode for the source and destination, but any valid combination of source and destination addressing modes is possible in an instruction.

5.2.1 Register Mode

The register mode is described in Table 5–5.

Table 5–5. Register Mode Description

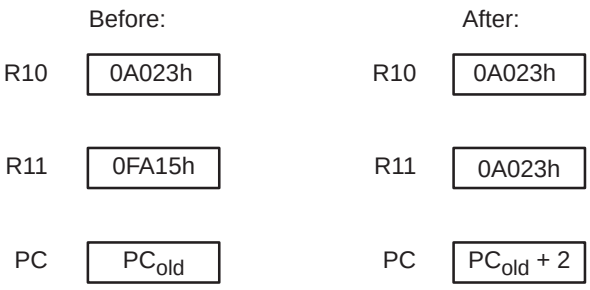
Assembler Code	Content of ROM
MOV R10,R11	MOV R10,R11

Length: One or two words

Operation: Move the content of R10 to R11. R10 is not affected.

Comment: Valid for source and destination

Example: MOV R10,R11



Note: Data in Registers

The data in the register can be accessed using word or byte instructions. If byte instructions are used, the high byte is always 0 in the result. The status bits are handled according to the result of the byte instruction.

5.2.2 Indexed Mode

The indexed mode is described in Table 5–6.

Table 5–6. Indexed Mode Description

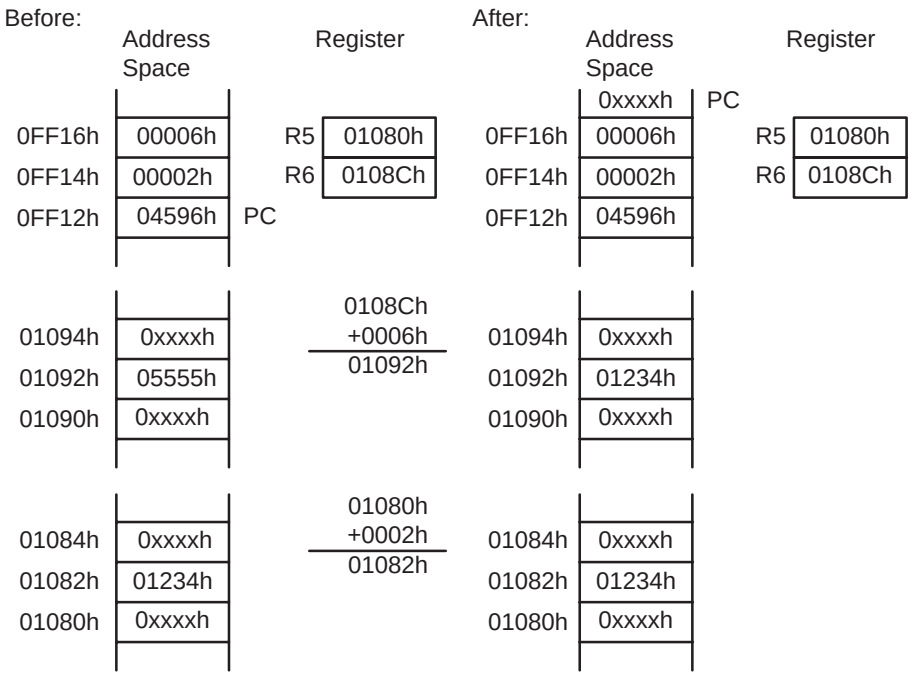
Assembler Code	Content of ROM
MOV 2(R5),6(R6)	MOV X(R5),Y(R6) X = 2 Y = 6

Length: Two or three words

Operation: Move the contents of the source address (contents of R5 + 2) to the destination address (contents of R6 + 6). The source and destination registers (R5 and R6) are not affected. In indexed mode, the program counter is incremented automatically so that program execution continues with the next instruction.

Comment: Valid for source and destination

Example: MOV 2(R5),6(R6):



5.2.3 Symbolic Mode

The symbolic mode is described in Table 5–7.

Table 5–7. Symbolic Mode Description

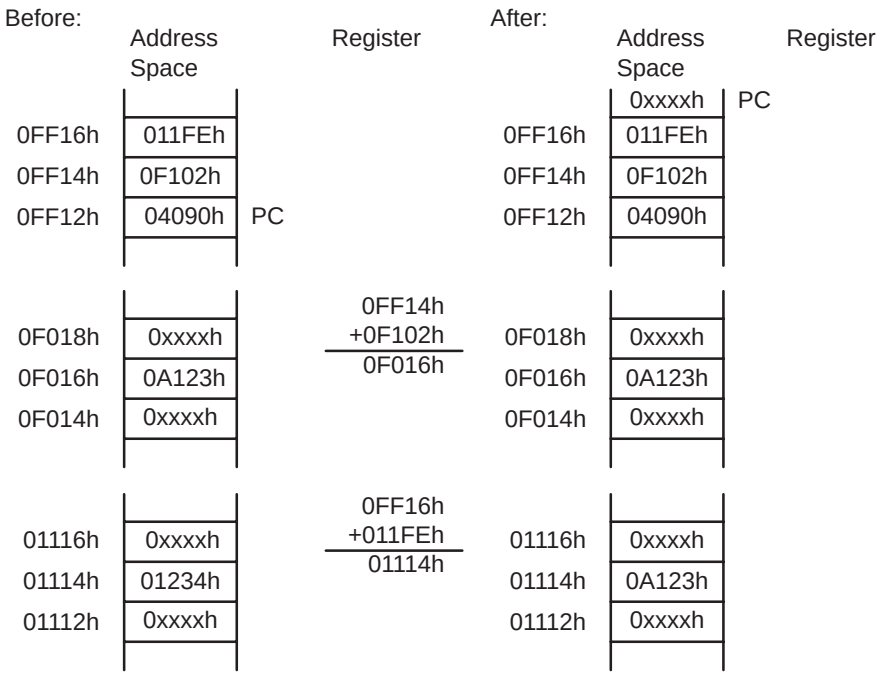
Assembler Code	Content of ROM
MOV EDE,TONI	MOV X(PC),Y(PC) X = EDE – PC Y = TONI – PC

Length: Two or three words

Operation: Move the contents of the source address EDE (contents of PC + X) to the destination address TONI (contents of PC + Y). The words after the instruction contain the differences between the PC and the source or destination addresses. The assembler computes and inserts offsets X and Y automatically. With symbolic mode, the program counter (PC) is incremented automatically so that program execution continues with the next instruction.

Comment: Valid for source and destination

Example: MOV EDE,TONI ;Source address EDE = 0F016h,
;dest. address TONI=01114h



5.2.4 Absolute Mode

The absolute mode is described in Table 5–8.

Table 5–8. Absolute Mode Description

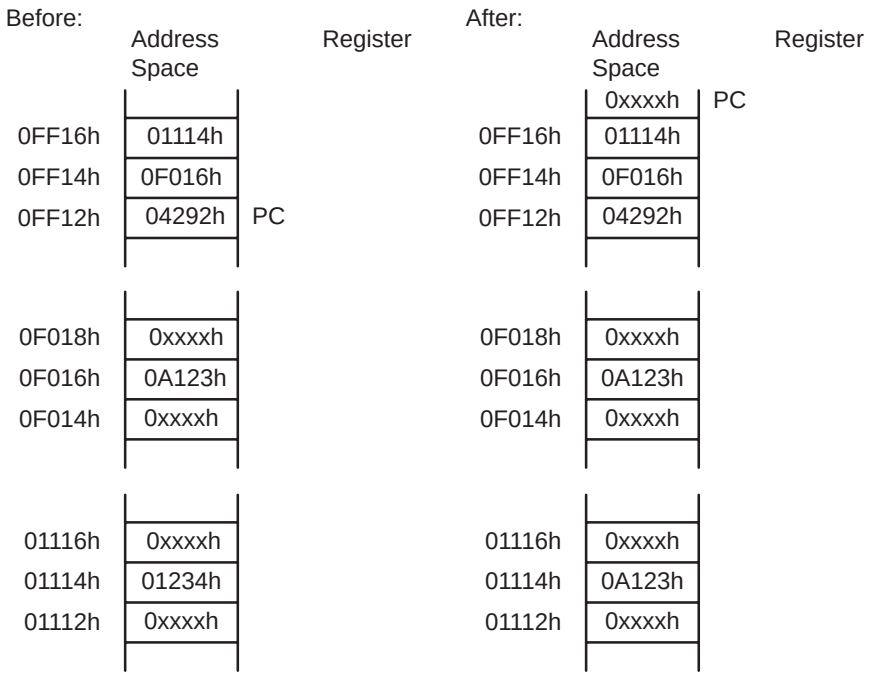
Assembler Code	Content of ROM
MOV &EDE,&TONI	MOV X(0),Y(0) X = EDE Y = TONI

Length: Two or three words

Operation: Move the contents of the source address EDE to the destination address TONI. The words after the instruction contain the absolute address of the source and destination addresses. With absolute mode, the PC is incremented automatically so that program execution continues with the next instruction.

Comment: Valid for source and destination

Example:
```
MOV  &EDE,&TONI     ;Source address EDE = 0F016h,
                    ;dest. address TONI=01114h
```



This address mode is mainly for hardware peripheral modules that are located at an absolute, fixed address. These are addressed with absolute mode to ensure software transportability (for example, position-independent code).

5.2.5 Indirect Mode

The indirect mode is described in table 5–9.

Table 5–9. Indirect Mode Description

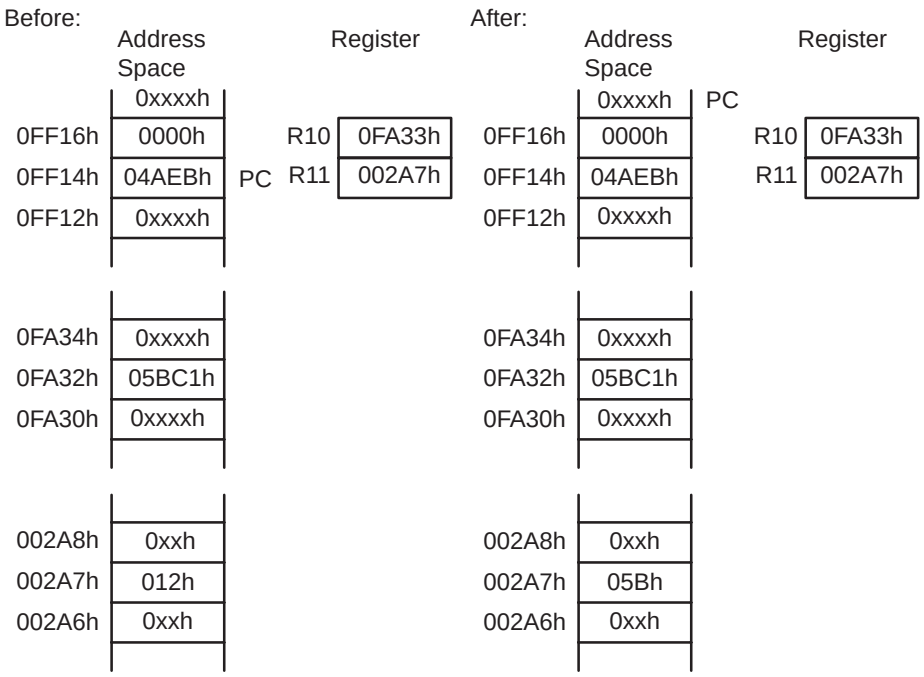
Assembler Code	Content of ROM
MOV @R10,0(R11)	MOV @R10,0(R11)

Length: One or two words

Operation: Move the contents of the source address (contents of R10) to the destination address (contents of R11). The registers are not modified.

Comment: Valid only for source operand. The substitute for destination operand is 0(Rd).

Example: MOV.B @R10,0(R11)



5.2.6 Indirect Autoincrement Mode

The indirect autoincrement mode is described in Table 5–10.

Table 5–10. Indirect Autoincrement Mode Description

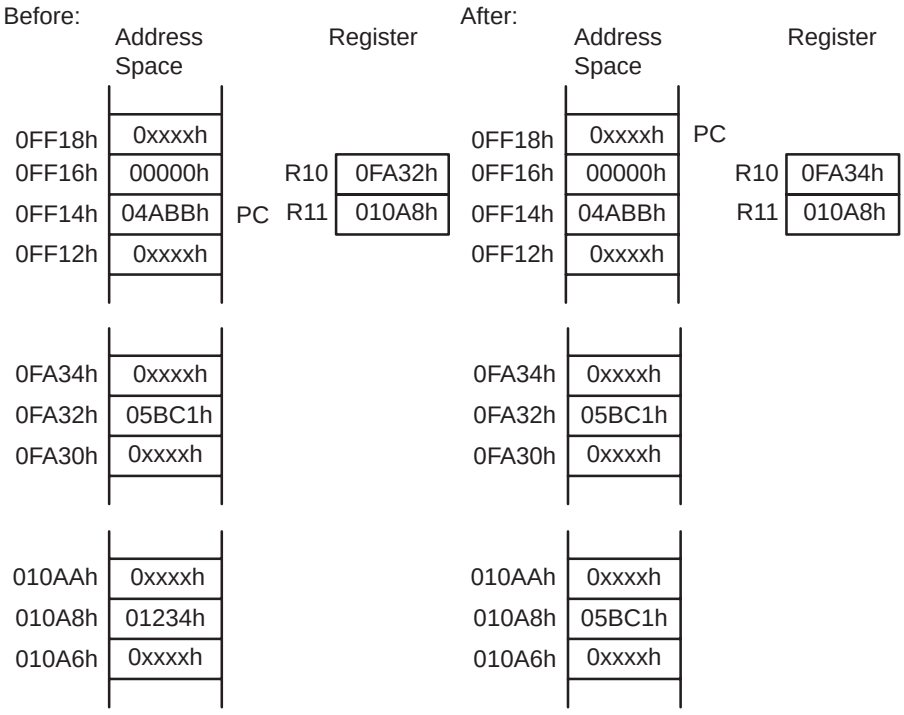
Assembler Code	Content of ROM
MOV @R10+,0(R11)	MOV @R10+,0(R11)

Length: One or two words

Operation: Move the contents of the source address (contents of R10) to the destination address (contents of R11). Register R10 is incremented by 1 for a byte operation, or 2 for a word operation after the fetch; it points to the next address without any overhead. This is useful for table processing.

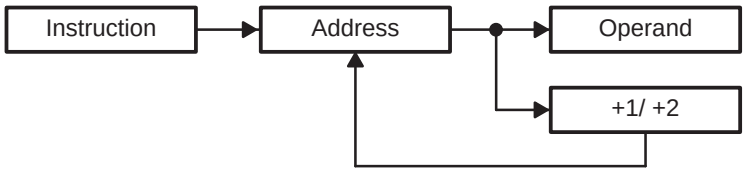
Comment: Valid only for source operand. The substitute for destination operand is 0(Rd) plus second instruction INCD Rd.

Example: MOV @R10+,0(R11)



The autoincrementing of the register contents occurs after the operand is fetched. This is shown in Figure 5–6.

Figure 5–6. Operand Fetch Operation



5.2.7 Immediate Mode

The immediate mode is described in Table 5–11.

Table 5–11. Immediate Mode Description

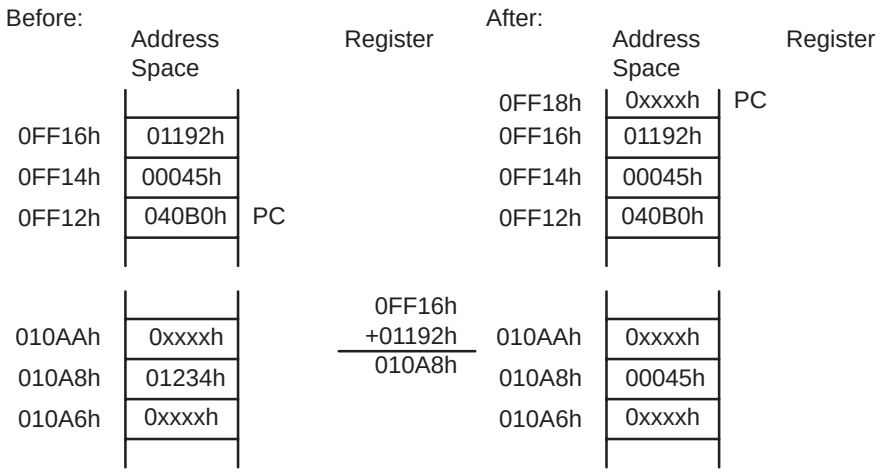
Assembler Code	Content of ROM
MOV #45,TONI	MOV @PC+,X(PC) 45 X = TONI – PC

Length: Two or three words
It is one word less if a constant of CG1 or CG2 can be used.

Operation: Move the immediate constant 45, which is contained in the word following the instruction, to destination address TONI. When fetching the source, the program counter points to the word following the instruction and moves the contents to the destination.

Comment: Valid only for a source operand.

Example: MOV #45,TONI



5.2.8 Clock Cycles, Length of Instruction

The operating speed of the CPU depends on the instruction format and addressing modes. The number of clock cycles refers to the MCLK.

5.2.8.1 Format-I Instructions

Table 5–12 describes the CPU format-I instructions and addressing modes.

Table 5–12. Instruction Format I and Addressing Modes

Address Mode		No. of Cycles	Length of Instruction	Example	
As	Ad				
00, Rn	0, Rm	1	1	MOV	R5,R8
	0, PC	2	1	BR	R9
00, Rn	1, x(Rm)	4	2	ADD	R5,3(R6)
	1, EDE		2	XOR	R8,EDE
	1, &EDE		2	MOR	R5,&EDE
01, x(Rn)	0, Rm	3	2	MOV	2(R5),R7
01, EDE			2	AND	EDE,R6
01, &EDE			2	MOV	&EDE,R8
01, x(Rn)	1, x(Rm)	6	3	ADD	3(R4),6(R9)
01, EDE	1, TONI		3	CMP	EDE,TONI
01, &EDE	1, &TONI		3	MOV	2(R5),&TONI
			3	ADD	EDE,&TONI
10, @Rn	0, Rm	2	1	AND	@R4,R5
10, @Rn	1, x(Rm)	5	2	XOR	@R5,8(R6)
	1, EDE		2	MOV	@R5,EDE
	1, &EDE		2	XOR	@R5,&EDE
11, @Rn+	0, Rm	2	1	ADD	@R5+,R6
	0, PC	3	1	BR	@R9+
11, #N	0, Rm	2	2	MOV	#20,R9
	0, PC	3	2	BR	#2AEh
11, @Rn+	1, x(Rm)	5	2	MOV	@R9+,2(R4)
11, #N	1, EDE		3	ADD	#33,EDE
11, @Rn+	1, &EDE		2	MOV	@R9+,&EDE
11, #N			3	ADD	#33,&EDE

5.2.8.2 Format-II Instructions

Table 5–13 describes the CPU format II instructions and addressing modes.

Table 5–13. Instruction Format-II and Addressing Modes

Address Mode $A_{(s/d)}$	No. of Cycles: RRA RRC SWPB SXT	No. of Cycles: PUSH/ CALL	Length of Instruction (words)	Example
00, Rn	1	3/4	1	SWPB R5
01, X(Rn) 01, EDE 01, &EDE	4 4	5 5	2 2	CALL 2(R7) PUSH EDE SXT &EDE
10, @Rn	3	4	1	RRC @R9
11, @Rn+ (see Note) 11, #N	3	4/5	1 2	SWPB @R10+ CALL #81H

Note: Instruction Format II Immediate Mode

Do not use instructions RRA, RRC, SWPB, and SXT with the immediate mode in the destination field. Use of these in the immediate mode will result in an unpredictable program operation.

5.2.8.3 Format-III Instructions

Format-III instructions are described as follows:

Jxx—all instructions need the same number of cycles, independent of whether a jump is taken or not.

Clock cycle: Two cycles
Length of instruction: One word

5.2.8.4 Miscellaneous-Format Instructions

Table 5–14 describes miscellaneous-format instructions.

Table 5–14. Miscellaneous Instructions or Operations

Activity	Clock Cycle
RETI	5 cycles 1 word†
Interrupt	6 cycles
WDT reset	4 cycles
Reset ($\overline{RST}$/NMI)	4 cycles

† Length of instruction

5.3 Instruction Set Overview

This section gives a short overview of the instruction set. The addressing modes are described in Section 5.2.

Instructions are either single or dual operand or jump.

The source and destination parts of an instruction are defined by the following fields:

src	The source operand defined by As and S-reg
dst	The destination operand defined by Ad and D-reg
As	The addressing bits responsible for the addressing mode used for the source (src)
S-reg	The working register used for the source (src)
Ad	The addressing bits responsible for the addressing mode used for the destination (dst)
D-reg	The working register used for the destination (dst)
B/W	Byte or word operation: 0: word operation 1: byte operation

Note: Destination Address

Destination addresses are valid anywhere in the memory map. However, when using an instruction that modifies the contents of the destination, the user must ensure the destination address is writeable. For example, a masked-ROM location would be a valid destination address, but the contents are not modifiable, so the results of the instruction would be lost.

5.3.1 Double-Operand Instructions

Figure 5–7 illustrates the double-operand instruction format.

Figure 5–7. Double Operand Instruction Format

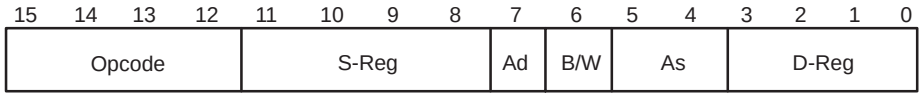


Table 5–15 describes the effects of an instruction on double operand instruction status bits.

Table 5–15. Double Operand Instruction Format Results

Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
MOV	src,dst	src –> dst	–	–	–	–
ADD	src,dst	src + dst –> dst	*	*	*	*
ADDC	src,dst	src + dst + C –> dst	*	*	*	*
SUB	src,dst	dst + .not.src + 1 –> dst	*	*	*	*
SUBC	src,dst	dst + .not.src + C –> dst	*	*	*	*
CMP	src,dst	dst – src	*	*	*	*
DADD	src,dst	src + dst + C –> dst (dec)	*	*	*	*
AND	src,dst	src .and. dst –> dst	0	*	*	*
BIT	src,dst	src .and. dst	0	*	*	*
BIC	src,dst	.not.src .and. dst –> dst	–	–	–	–
BIS	src,dst	src .or. dst –> dst	–	–	–	–
XOR	src,dst	src .xor. dst –> dst	*	*	*	*

* The status bit is affected

– The status bit is not affected

0 The status bit is cleared

1 The status bit is set

Note: Instructions CMP and SUB

The instructions CMP and SUB are identical except for the storage of the result. The same is true for the BIT and AND instructions.

5.3.2 Single-Operand Instructions

Figure 5–8 illustrates the single-operand instruction format.

Figure 5–8. Single Operand Instruction Format

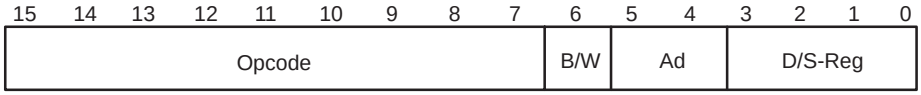


Table 5–16 describes the effects of an instruction on the single operand instruction status bits.

Table 5–16. Single Operand Instruction Format Results

Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
RRC	dst	C –> MSB –>.....LSB –> C	*	*	*	*
RRA	dst	MSB –> MSB –>....LSB –> C	0	*	*	*
PUSH	src	SP – 2 –> SP, src –> @ SP	–	–	–	–
SWPB	dst	swap bytes	–	–	–	–
CALL	dst	SP – 2 –> SP	–	–	–	–
		PC+2 –> stack, dst –> PC				
RETI		TOS –> SR, SP <– SP + 2	X	X	X	X
		TOS –> PC, SP <– SP + 2				
SXT	dst	Bit 7 –> Bit 8.....Bit 15	0	*	*	*

* The status bit is affected

– The status bit is not affected

0 The status bit is cleared

1 The status bit is set

All addressing modes are possible for the CALL instruction. If the symbolic mode (ADDRESS), the immediate mode (#N), the absolute mode (&EDE) or the indexed mode X (RN) is used, the word that follows contains the address information.

5.3.3 Conditional Jumps

Conditional jumps support program branching relative to the program counter. The possible jump range is from –511 to +512 words relative to the program counter state of the jump instruction. The 10-bit program-counter offset value is treated as a signed 10-bit value that is doubled and added to the program counter. None of the jump instructions affect the status bits.

The instruction code fetch and the program counter increment technique end with the formula:

$$PC_{new} = PC_{old} + 2 + PC_{offset} \times 2$$

Figure 5–9 shows the conditional-jump instruction format.

Figure 5–9. Conditional-Jump Instruction Format

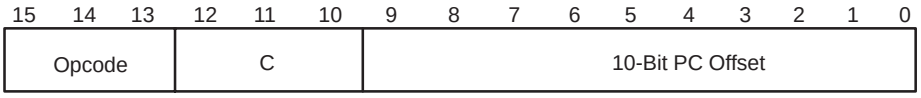


Table 5–17 describes these conditional-jump instructions.

Table 5–17. Conditional-Jump Instructions

Mnemonic	S-Reg, D-Reg	Operation
JEQ/JZ	Label	Jump to label if zero bit is set
JNE/JNZ	Label	Jump to label if zero bit is reset
JC	Label	Jump to label if carry bit is set
JNC	Label	Jump to label if carry bit is reset
JN	Label	Jump to label if negative bit is set
JGE	Label	Jump to label if (N .XOR. V) = 0
JL	Label	Jump to label if (N .XOR. V) = 1
JMP	Label	Jump to label unconditionally

5.3.4 Short Form of Emulated Instructions

The basic instruction set, together with the register implementations of the program counter, stack pointer, status register, and constant generator, form the emulated instruction set; these make up the popular instruction set. The status bits are set according to the result of the execution of the basic instruction that replaces the emulated instruction.

Table 5–18 describes these instructions.

Table 5–18. Emulated Instructions

Mnemonic		Description	Status Bits				Emulation	
			V	N	Z	C		
ArIthmetic Instructions								
ADC[.W]	dst	Add carry to destination	*	*	*	*	ADDC	#0,dst
ADC.B	dst	Add carry to destination	*	*	*	*	ADDC.B	#0,dst
DADC[.W]	dst	Add carry decimal to destination	*	*	*	*	DADD	#0,dst
DADC.B	dst	Add carry decimal to destination	*	*	*	*	DADD.B	#0,dst
DEC[.W]	dst	Decrement destination	*	*	*	*	SUB	#1,dst
DEC.B	dst	Decrement destination	*	*	*	*	SUB.B	#1,dst
DECD[.W]	dst	Double-decrement destination	*	*	*	*	SUB	#2,dst
DECD.B	dst	Double-decrement destination	*	*	*	*	SUB.B	#2,dst
INC[.W]	dst	Increment destination	*	*	*	*	ADD	#1,dst
INC.B	dst	Increment destination	*	*	*	*	ADD.B	#1,dst
INCD[.W]	dst	Increment destination	*	*	*	*	ADD	#2,dst
INCD.B	dst	Increment destination	*	*	*	*	ADD.B	#2,dst
SBC[.W]	dst	Subtract carry from destination	*	*	*	*	SUBC	#0,dst
SBC.B	dst	Subtract carry from destination	*	*	*	*	SUBC.B	#0,dst
Logical Instructions								
INV[.W]	dst	Invert destination	*	*	*	*	XOR	#0FFFFh,dst
INV.B	dst	Invert destination	*	*	*	*	XOR.B	#–1,dst
RLA[.W]	dst	Rotate left arithmetically	*	*	*	*	ADD	dst,dst
RLA.B	dst	Rotate left arithmetically	*	*	*	*	ADD.B	dst,dst
RLC[.W]	dst	Rotate left through carry	*	*	*	*	ADDC	dst,dst
RLC.B	dst	Rotate left through carry	*	*	*	*	ADDC.B	dst,dst
Data Instructions (common use)								
CLR[.W]		Clear destination	–	–	–	–	MOV	#0,dst
CLR.B		Clear destination	–	–	–	–	MOV.B	#0,dst
CLRC		Clear carry bit	–	–	–	0	BIC	#1,SR
CLRN		Clear negative bit	–	0	–	–	BIC	#4,SR
CLRZ		Clear zero bit	–	–	0	–	BIC	#2,SR
POP	dst	Item from stack	–	–	–	–	MOV	@SP+,dst
SETC		Set carry bit	–	–	–	1	BIS	#1,SR
SETN		Set negative bit	–	1	–	–	BIS	#4,SR
SETZ		Set zero bit	–	–	1	–	BIS	#2,SR

Table 5–18. Emulated Instructions (Continued)

Mnemonic		Description	Status Bits				Emulation	
			V	N	Z	C		
Data Instructions (common use) (continued)								
TST[.W]	dst	Test destination	0	*	*	*	CMP	#0,dst
TST.B	dst	Test destination	0	*	*	*	CMP.B	#0,dst
Program Flow Instructions								
BR	dst	Branch to . . .	–	–	–	–	MOV	dst,PC
DINT		Disable interrupt	–	–	–	–	BIC	#8,SR
EINT		Enable interrupt	–	–	–	–	BIS	#8,SR
NOP		No operation	–	–	–	–	MOV	#0h,#0h
RET		Return from subroutine	–	–	–	–	MOV	@SP+,PC

5.3.5 Miscellaneous

Instructions without operands, such as CPUOff, are not provided. Their functions are switched on or off by setting or clearing the function bits in the status register or the appropriate I/O register. Other functions are emulated using dual operand instructions.

Some examples are as follows:

```
BIS     #28h,SR     ; Enter OscOff mode
                    ; + Enable general interrupt (GIE)
BIS     #18h,SR     ; Enter CPUOff mode
                    ; + Enable general interrupt (GIE)
```

5.4 Instruction Map

The instruction map in Figure 5–10 is an example of how to encode instructions. There is room for more instructions, if needed.

Figure 5–10. Core Instruction Map

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0x																
04x																
08x																
0Cx																
10x	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI			
14x																
18x																
1Cx																
20x	JNE/JNZ															
24x	JEQ/JZ															
28x	JNC															
2Cx	JC															
30x	JN															
34x	JGE															
38x	JL															
3Cx	JMP															
40x–4Fx	MOV, MOV.B															
50x–5Fx	ADD, ADD.B															
60x–6Fx	ADDC, ADDC.B															
70x–7Fx	SUBC, SUBC.B															
80x–8Fx	SUB, SUB.B															
90x–9Fx	CMP, CMP.B															
A0x–AFx	DADD, DADD.B															
B0x–BFx	BIT, BIT.B															
C0x–CFx	BIC, BIC.B															
D0x–DFx	BIS, BIS.B															
E0x–EFx	XOR, XOR.B															
F0x–FFx	AND, AND.B															

Chapter 6

Hardware Multiplier

The hardware multiplier is a 16-bit peripheral module. It is not integrated into the CPU. Therefore, it requires no special instructions and operates independent of the CPU. To use the hardware multiplier, the operands are loaded into registers and the results are available the next instruction—no extra cycles are required for a multiplication.

Topic **Page**

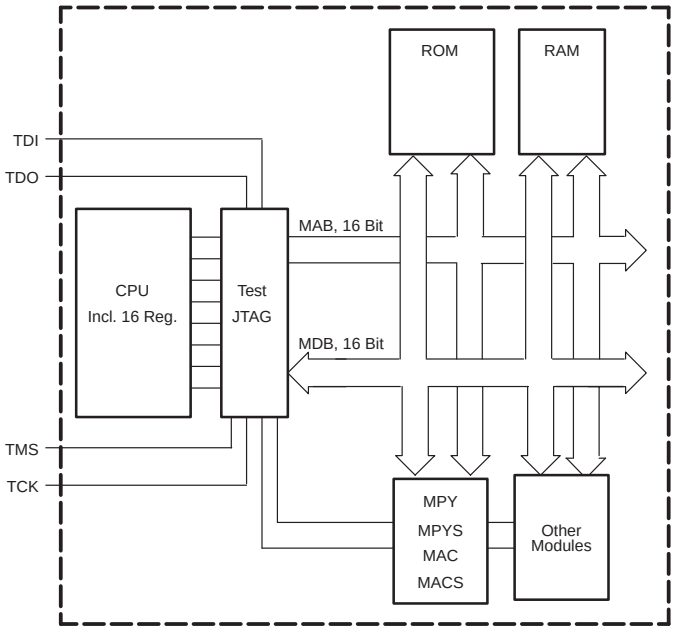
6.1 Hardware Multiplier Module Support

The hardware multiplier module expands the capabilities of the MSP430 family without changing the basic architecture. Multiplication is possible for:

- 16×16 bits
- 16×8 bits
- 8×16 bits
- 8×8 bits

The hardware multiplier module supports four types of multiplication: unsigned multiplication (MPY), signed multiplication (MPYS), unsigned multiplication with accumulation (MAC), and signed multiplication with accumulation (MACS). Figure 6–1 shows how the hardware multiplier module interfaces with the bus system to support multiplication operations.

Figure 6–1. Connection of the Hardware Multiplier Module to the Bus System



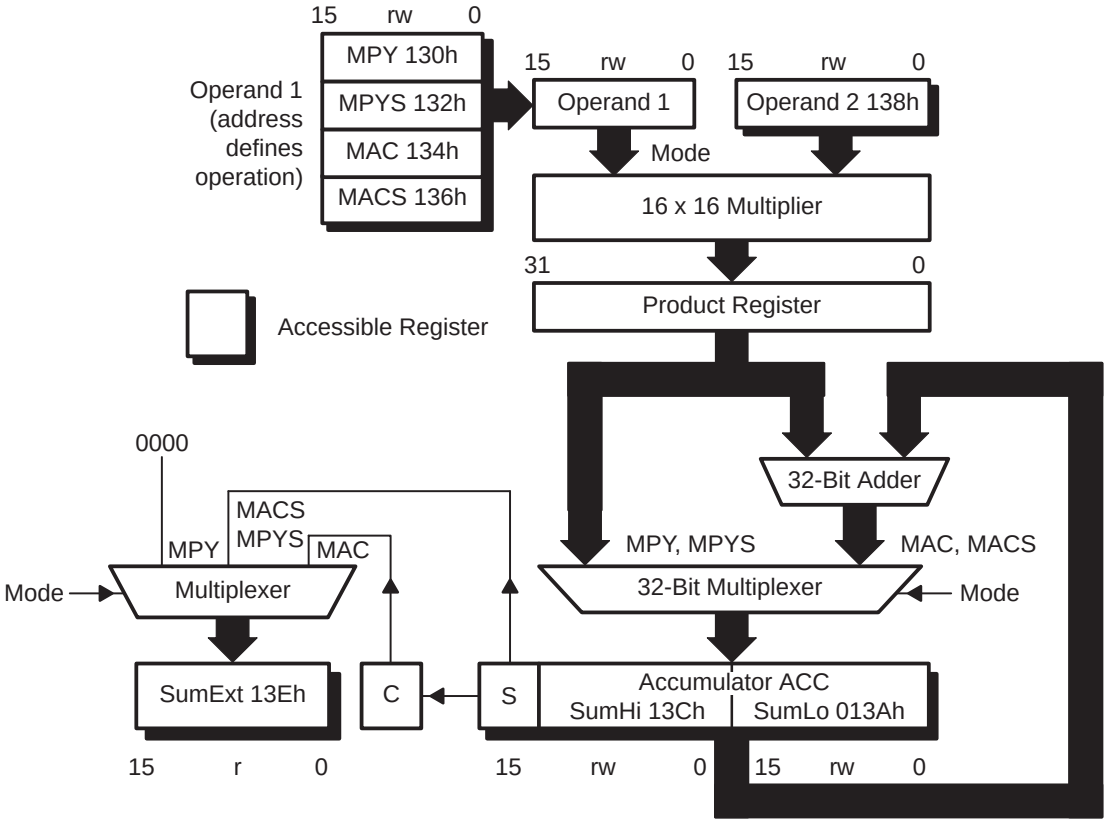
6.2 Hardware Multiplier Operation

The hardware multiplier has two 16-bit registers for both operands and three registers to store the results of the multiplication. The multiplication is executed correctly when the first operand is written to the operand register OP1 prior to writing the second operand to OP2. Writing the first operand to the applicable register selects the type of multiplication. Writing the second operand to OP2 starts the multiplication. Multiplication is completed before the result registers are accessed using the indexed address mode for the source operand. When indirect or indirect autoincrement address modes are used, another instruction is needed between the writing of the second operand and accessing the result registers. Both operands, OP1 and OP2, utilize all seven address mode capabilities.

No instruction is necessary for the multiplication; as a result, the real-time operation does not require additional clock cycles and the interrupt latency is unchanged.

The multiplier architecture is illustrated in Figure 6–2.

Figure 6–2. Block Diagram of the MSP430 16×16-Bit Hardware Multiplier



The sum extension register contents differ, depending on the operation and on the results of the operation.

Table 6–1. Sum Extension Register Contents

Register	MPY	MPYS		MAC		MACS, see Notes	
Operand1	x	+ –	+ +	(OP1×OP2 + ACC) ≤ 0FFFFFFFFh	(OP1×OP2 + ACC) > 0FFFFFFFFh	(OP1×OP2 + ACC) > 07FFFFFFFh	(OP1×OP2 + ACC) ≤ 07FFFFFFFh
Operand2	x	+ –	– –				
SumExt	0000h	0000h	0FFFFh	0000h	0001h	0FFFFh	0000h

Note: The following two overflow conditions may occur when using the MACS function and should be handled by software or avoided.

1) The result of a MACS operation is positive and larger than 07FFF FFFFh. In this case, the SumExt register contains 0FFFFh and the ACC register contains a negative number (8000 0000h .... 0FFFF FFFFh).
2) The result of a MACS operation is negative and less than or equal to 07FFF FFFFh. In this case, the SumExt register contains 0000h and the ACC register contains a positive number (0000 0000h ... 07FFF FFFFh).

6.2.1 Multiply Unsigned, 16×16 bit, 16×8 bit, 8×16 bit, 8×8 bit

The following multiplication operation shows 32 bytes of program code and 32 execution cycles (16×16 bit multiplication).

```
*********************************************************
*      TRANSFER BOTH OPERANDS TO THE REGISTERS IN THE    *
*      HARDWARE MULTIPLIER MODULE                        *
*      USE CONSTANT OPERAND1 AND OPERAND2 TO IDENTIFY    *
*      BYTE DATA                                         *
*********************************************************

OPERAND1 .EQU  0          ; 0: OPERAND1 IS WORD (16BIT)
                          ; 8: OPERAND1 IS BYTE ( 8BIT)
OPERAND2 .EQU  0          ; 0: OPERAND2 IS WORD (16BIT)
                          ; 8: OPERAND2 IS BYTE ( 8BIT)

MPY     .EQU   0130H
MPYS    .EQU   0132H
MAC     .EQU   0134H
MACS    .EQU   0136H
OP2     .EQU   0138H
RESLO   .EQU   013AH
RESHI   .EQU   013CH
SUMEXT  .EQU   013EH
        .BSS   OPER1,2,200H
        .BSS   OPER2,2
        .BSS   RAM,8

        .IF OPERAND1=8
        MOV.B  &OPER1,&MPY ; LOAD 1ST OPERAND,
                           ; DEFINES ADD. UNSIGNED MULTIPLY

        .ELSE
        MOV    &OPER1,&MPY ; LOAD 1ST OPERAND,
                           ; DEFINES ADD. UNSIGNED MULTIPLY

        .ENDIF

        .IF OPERAND1=8
        MOV.B  &OPER2,&OP2 ; LOAD 2ND OPERAND AND START
                           ; MULTIPLICATION

        .ELSE
        MOV    &OPER2,&OP2 ; LOAD 2ND OPERAND AND START
                           ; MULTIPLICATION

        .ENDIF

*********************************************************
*      EXAMPLE TO ADD THE RESULT OF THE HARDWARE         *
*      MULTIPLICATION TO THE RAM DATA, 64BITS            *
*********************************************************

        ADD    &RESLO,&RAM   ; ADD LOW RESULT TO RAM
        ADDC   &RESHI,&RAM+2 ; ADD HIGH RESULT TO RAM+2
        ADC    &RAM+4        ; ADD CARRY TO EXTENSION WORD
        ADC    &RAM+6        ; IF 64 BIT LENGTH IS USED
```

6.2.2 Multiply Signed, 16×16 bit, 16×8 bit, 8×16 bit, 8×8 bit

The following multiplication operation shows 36 bytes of program code and 36 execution cycles (16×16 bit multiplication).

```
*********************************************************
*      TRANSFER BOTH OPERANDS TO THE REGISTERS IN THE    *
*      HARDWARE MULTIPLIER MODULE                        *
*      IF ONE OF THE OPERANDS IS 8 BIT, SIGN EXTENSION   *
*      is NEEDED. USE CONSTANT OPERAND1 AND OPERAND2 TO  *
*      IDENTIFY BYTE DATA                                *
*********************************************************

OPERAND1 .EQU  0          ; 0: OPERAND1 IS WORD (16BIT)
                          ; 8: OPERAND1 IS BYTE ( 8BIT)
OPERAND2 .EQU  0          ; 0: OPERAND2 IS WORD (16BIT)
                          ; 8: OPERAND2 IS BYTE ( 8BIT)

MPY    .EQU  0130H
MPYS   .EQU  0132H
MAC    .EQU  0134H
MACS   .EQU  0136H
OP2    .EQU  0138H
RESLO  .EQU  013AH
RESHI  .EQU  013CH
SUMEXT .EQU  013EH
       .BSS  OPER1,2,200H
       .BSS  OPER2,2
       .BSS  RAM,8

       .IF OPERAND1=0
       MOV   &OPER1,&MPYS ; LOAD 1ST (WORD) OPERAND,
                          ; DEFINES ADD. SIGNED MULTIPLY

       .ELSE
       MOV.B &OPER1,&MPYS ; LOAD 1ST (BYTE) OPERAND,
                          ; DEFINES ADD. SIGNED MULTIPLY
       SXT   &MPYS        ; EXPAND BYTE TO SIGNED WORD DATA
       .ENDIF
       .IF OPERAND2=0
       MOV   &OPER2,&OP2  ; LOAD 2ND (WORD) OPERAND AND
                          ; START SIGNED MULTIPLICATION

       .ELSE
       MOV.B &OPER2,&OP2  ; LOAD 2ND (BYTE) OPERAND,
       SXT   &OP2         ; RE-LOAD 2ND OPERAND AND START
                          ; SIGNED 'FINAL' MULTIPLICATION

       .ENDIF

*********************************************************
*       EXAMPLE TO ADD THE RESULT OF THE HARDWARE       *
*       MULTIPLICATION TO THE RAM DATA, 64 BITS         *
*********************************************************

      ADD  &RESLO,&RAM    ; ADD LOW RESULT TO RAM
      ADDC &RESHI,&RAM+2  ; ADD HIGH RESULT TO RAM+2
      ADDC &SUMEXT,&RAM+4 ; ADD SIGN WORD TO EXTENSION WORD
      ADDC &SUMEXT,&RAM+6 ; IF 64 BIT LENGTH IS USED
```

6.2.3 Multiply Unsigned and Accumulate, 16x16bit, 16x8bit, 8x16bit, 8x8bit

The following multiplication operation shows 32 bytes of program code and 32 execution cycles (16X16-bit multiplication).

```
*********************************************************
*     TRANSFER BOTH OPERANDS TO THE REGISTERS IN THE    *
*     HARDWARE MULTIPLIER MODULE                        *
*     THE RESULT OF THE MULTIPLICATION IS ADDED TO THE  *
*     CONTENT OF BOTH RESULT REGISTERS, RESLO AND RESHI *
*     USE CONSTANT OPERAND1 AND OPERAND2 TO IDENTIFY    *
*     BYTE DATA                                         *
*********************************************************

OPERAND1 .EQU  0          ; 0: OPERAND1 IS WORD (16BIT)
                          ; 8: OPERAND1 IS BYTE ( 8BIT)
OPERAND2 .EQU  0          ; 0: OPERAND2 IS WORD (16BIT)
                          ; 8: OPERAND2 IS BYTE ( 8BIT)

MPY     .EQU   0130H
MPYS    .EQU   0132H
MAC     .EQU   0134H
MACS    .EQU   0136H
OP2     .EQU   0138H
RESLO   .EQU   013AH
RESHI   .EQU   013CH
SUMEXT  .EQU   013EH
        .BSS   OPER1,2,200H
        .BSS   OPER2,2
        .BSS   RAM,8

        .IF OPERAND1=8
        MOV.B  &OPER1,&MAC ; LOAD 1ST OPERAND,
                           ; DEFINES ADD. UNSIGNED MULTIPLY

        .ELSE
        MOV    &OPER1,&MAC ; LOAD 1ST OPERAND,
                           ; DEFINES ADD. UNSIGNED MULTIPLY

        .ENDIF

        .IF OPERAND1=8
        MOV.B  &OPER2,&OP2 ; LOAD 2ND OPERAND AND START
                           ; MULTIPLICATION

        .ELSE
        MOV    &OPER2,&OP2 ; LOAD 2ND OPERAND AND START
                           ; MULTIPLICATION

        .ENDIF

*********************************************************
*      EXAMPLE TO ADD THE RESULT OF THE HARDWARE        *
*      MULTIPLICATION TO THE RAM DATA, 64BITS           *
*      THE RESULT OF THE MULTIPLICATION IS HELD IN RESLO*
*      AND RESHI REGISTERS. THE UPPER TWO WORDS IN THE  *
*      EXAMPLE ARE FURTHER LOCATED IN THEIR RAM LOCATION*
*********************************************************

       ADDC  &SUMEXT,&RAM+4  ; ADD SUMEXTENSTION TO RAM+4
       ADC   &RAM+6          ; IF 64 BIT LENGTH IS USED
```

6.2.4 Multiply Signed and Accumulate, 16x16bit, 16x8bit, 8x16bit, 8x8bit

```
*********************************************************************
*      TRANSFER BOTH OPERANDS TO THE REGISTERS IN THE HARDWARE      *
*      MULTIPLIER MODULE                                            *
*      USE CONSTANT OPERAND1 AND OPERAND 2 TO IDENTIFY BYTE DATA    *
*********************************************************************
OPERAND1    .EQU    0       ; 0:  OPERAND1 IS WORD (16BIT)
                            ; 8:  OPERAND1 IS BYTE ( 8BIT)
OPERAND2    .EQU    0       ; 0:  OPERAND2 IS WORD (16BIT)
                            ; 8:  OPERAND2 IS BYTE ( 8BIT)
MPY         .EQU    0130H
MPYS        .EQU    0132H
MAC         .EQU    0134H
MACS        .EQU    0136H
OP2         .EQU    0138H
RESLO       .EQU    013AH
RESHI       .EQU    013CH
SUMEXT      .EQU    013EH
MAXMACS     .EQU    32H     ;NUMBER OF MACS FUNCTIONS WHICH COULD
                            ;BE EXECUTED TILL AN OVERFLOW OR UNDERFLOW
                            ;COULD OCCUR THE FIRST TIME
            .BSS    OPER1,2,200H
            .BSS    OPER2,2
            .BSS    RAM,8
            .BSS    MCOUNT,2
            .IF OPERAND1=8
            MOV.B   &OPER1,&MACS    ; LOAD 1ST OPERAND,
                                    ; DEFINES ADD. UNSIGNED MULTIPLY
            SXT     &MACS           ; EXPAND BYTE TO SIGNED WORD DATA
            .ELSE
            MOV     &OPER1,&MACS    ; LOAD 1ST OPERAND,
                                    ; DEFINES ADD. UNSIGNED MULTIPLY
            .ENDIF
            .IF OPERAND1=8
            SXT     &OPER2          ; OPER2 MEMORY LOCATION NEEDS
                                    ; 2 BYTES
            MOV.B   &OPER2,&OP2     ; LOAD 2ND OPERAND AND START
                                    ; MULTIPLICATION
            .ELSE
            MOV     &OPER2,&OP2     ; LOAD 2ND OPERAND AND START
                                    ; MULTIPLICATION
            .ENDIF
*********************************************************************
*      EXAMPLE TO ADD THE RESULT OF THE HARDWARE MULTIPLICATION     *
*      TO THE RAM DATA IF NECESSARY                                 *
*      THE RESULT OF THE MULTIPLICATION IS HELD IN RESLO AND        *
*      RESHI REGISTERS. THE UPPER TWO WORDS IN THE EXAMPLE ARE      *
*      FURTHER LOCATED IN THEIR RAM LOCATION                        *
*********************************************************************
            INC     MCOUNT              ; INC MACS COUNTER
            CMP     #MAXMACS,MCOUNT     ; ONLY ADD TO RAM IF NECESSARY
            JNE     NEXTMACS            ;
            ADDC    &RESLO,&RAM+0       ; ADD SUMEXTENSION TO RAM+0
            ADDC    &RESHI,&RAM+2       ; ADD SUMEXTENSION TO RAM+2
            ADDC    &SUMEXT,&RAM+4      ; ADD SUMEXTENSION TO RAM+4
            ADDC    &SUMEXT,&RAM+6      ; IF 64 BIT LENGTH IS USED
            CLR     MCOUNT
NEXTMACS
    . . .
```

6.3 Hardware Multiplier Registers

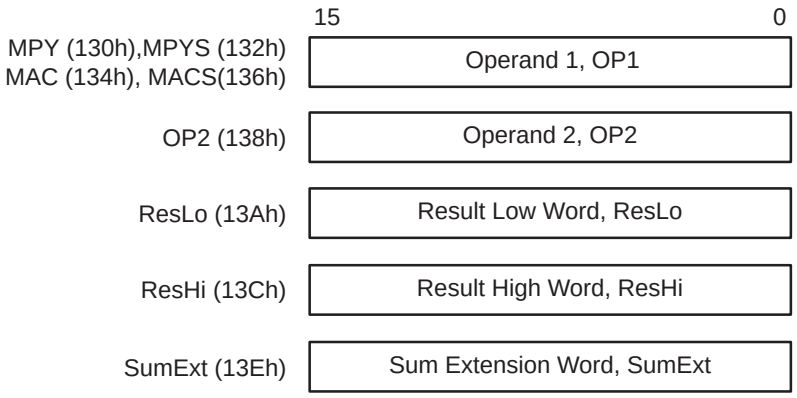
Hardware multiplier registers are word structured, but can be accessed using word or byte processing instructions. Table 6–2 describes the hardware multiplier registers.

Table 6–2. Hardware Multiplier Registers

Register	Short Form	Register Type	Address	Initial State
Multiply Unsigned (Operand1)	MPY	Read/write	0130h	Unchanged
Multiply Signed (Operand1)	MPYS	Read/write	0132h	Unchanged
Multiply+Accumulate (Operand1)	MAC	Read/write	0134h	Unchanged
Multiply Signed+Accumulate (Operand1)	MACS	Read/write	0136h	Unchanged
Second Operand	OP2	Read/write	0138h	Unchanged
Result Low Word	ResLo	Read/write	013Ah	Undefined
Result High Word	ResHi	Read/write	013Ch	Undefined
Sum Extend	SumExt	Read	013Eh	Undefined

Two registers are implemented for both operands, OP1 and OP2, as shown in Figure 6–3. Operand 1 uses four different addresses to address the same register. The different address information is decoded and defines the type of multiplication operation used.

Figure 6–3. Registers of the Hardware Multiplier



The multiplication result is located in two word registers: result high (RESHI) and result low (RESLO). The sum extend register (SumExt) holds the result sign of a signed operation or the overflow of the multiply and accumulate (MAC) operation. See Section 6.5.3 for a description of overflow and underflow when using the MACS operations.

All registers have the least significant bit (LSB) at bit0 and the most significant bit (MSB) at bit7 (byte data) or bit15 (word data).

6.4 Hardware Multiplier Special Function Bits

Because the hardware multiplier module completes all multiplication operations quickly, without interrupt intervention, no special function bits are used.

6.5 Hardware Multiplier Software Restrictions

Two restrictions require attention when the hardware multiplier is used:

- The indirect or indirect autoincrement address mode used to process the result
- The hardware multiplier used in an interrupt routine

6.5.1 Hardware Multiplier Software Restrictions—Address Mode

The result of the multiplication operation can be accessed in indexed, indirect, or indirect autoincrement mode. The result registers may be accessed without any restrictions if you use the indexed address mode including the symbolic and absolute address modes. However, when you use the indirect and indirect autoincrement address modes to access the result registers, you need at least one instruction between loading the second operand and accessing one of the result registers.

```
*********************************************************
*      EXAMPLE: MULTIPLY OPERAND1 AND OPERAND2
*********************************************************

RESLO    .SET   013AH       ; RESLO = ADDRESS OF RESLO
         PUSH   R5          ; R5 WILL HOLD THE ADDRESS OF
         MOV    #RESLO,R5   ; THE RESLO REGISTER

         MOV    &OPER1,&MPY ; LOAD 1ST OPERAND,
                            ; DEFINES ADD. UNSIGNED MULTIPLY
         MOV    &OPER2,&OP2 ; LOAD 2ND OPERAND AND START
                            ; MULTIPLICATION

*********************************************************
*      EXAMPLE TO ADD THE RESULT OF THE HARDWARE        *
*      MULTIPLICATION TO THE RAM DATA, 64BITS           *
*********************************************************

         NOP                ; MIN. ONE CYCLES BETWEEN MOVING
                            ; THE OPERAND2 TO HW-MULTIPLIER
                            ; AND PROCESSING THE RESULT WITH
                            ; INDIRECT ADDRESS MODE
         ADD    @R5+,&RAM   ; ADD LOW RESULT TO RAM
         ADDC   @R5,&RAM+2  ; ADD HIGH RESULT TO RAM+2
         ADC    &RAM+4      ; ADD CARRY TO EXTENSION WORD
         ADC    &RAM+6      ; IF 64 BIT LENGTH IS USED

         POP    R5
```

The previous example shows that the indirect or indirect autoincrement address modes, when used to transfer the result of a multiplication operation to the destination, need more cycles and code than the absolute address mode. There is no need to access the hardware multiplier using the indirect addressing mode.

6.5.2 Hardware Multiplier Software Restrictions—Interrupt Routines

The entire multiplication routine requires only three steps:

1) Move operand OP1 to the hardware multiplier; this defines the type of multiplication.

2) Move operand OP2 to the hardware multiplier; the multiplication starts.

3) Process the result of the multiplication in the RESLO, RESHI, and SUMEXT registers.

The following considerations describe the main routines that use hardware multiplication. If no hardware multiplication is used in the main routine, multiplication in an interrupt routine is protected from further interrupts, because the GIE bit is reset after entering the interrupt service routine. Typically, a multiplication operation that uses the entire data process occurs outside an interrupt routine and the interrupt routines are as short as possible.

A multiplication operation in an interrupt routine has some feedback to the multiplication operation in the main routine.

6.5.2.1 Interrupt Following an OP1 Transfer

The two LSBs of the first operand address define the type of multiplication operation. This information cannot be recovered by any later operation. Therefore an interrupt must not be accepted between the first two steps: move operand OP1 and OP2 to the multiplier.

6.5.2.2 Interrupt Following an OP2 Transfer

After the first two steps, the multiplication result is in the corresponding registers RESLO, RESHI, and SUMEXT. It can be saved on the stack (using the PUSH instruction) and can be restored after completing another multiplication operation (using the POP instruction). However, this operation takes additional code and cycles in the interrupt routine. You can avoid this, by making an entire multiplication routine uninterruptible, by disabling any interrupt (DINT) before entering the multiplication routine, and by enabling interrupts (EINT) after the multiplication routine is completed. The negative aspect of this method is that the critical interrupt latency is increased drastically for events that occur during this period.

6.5.2.3 General Recommendation

In general, one should avoid a hardware multiplication operation within an interrupt routine when a hardware multiplication is already used in the main program. (This will depend upon the application-specific software, applied libraries, and other included software.) The methods previously discussed have some negative implications; therefore, the best practice is to keep interrupt routines as short as possible.

6.5.3 Hardware Multiplier Software Restrictions—MACS

The multiplier does not automatically detect underflow or overflow in the MACS mode. An overflow occurs when the sum of the accumulator register and the result of the signed multiplication exceed the maximum binary range.

The binary range of the accumulator for positive numbers is 0 to $2^{31}-1$ (7FFF FFFFh) and for negative numbers is –1 (0FFFF FFFFh) to -2^{31} (8000 0000h). An overflow occurs when the sum of two negative numbers yields a result that is in the range given above for a positive number. An underflow occurs when the sum of two positive numbers yields a result that is in the range for a negative number.

The maximum number of successive MACS instructions without underflow or overflow is limited by the individual application and should be determined using a worst-case calculation. Care should then be exercised to not exceed the maximum number or to handle the conditions accordingly.

Chapter 7

Basic Clock Module

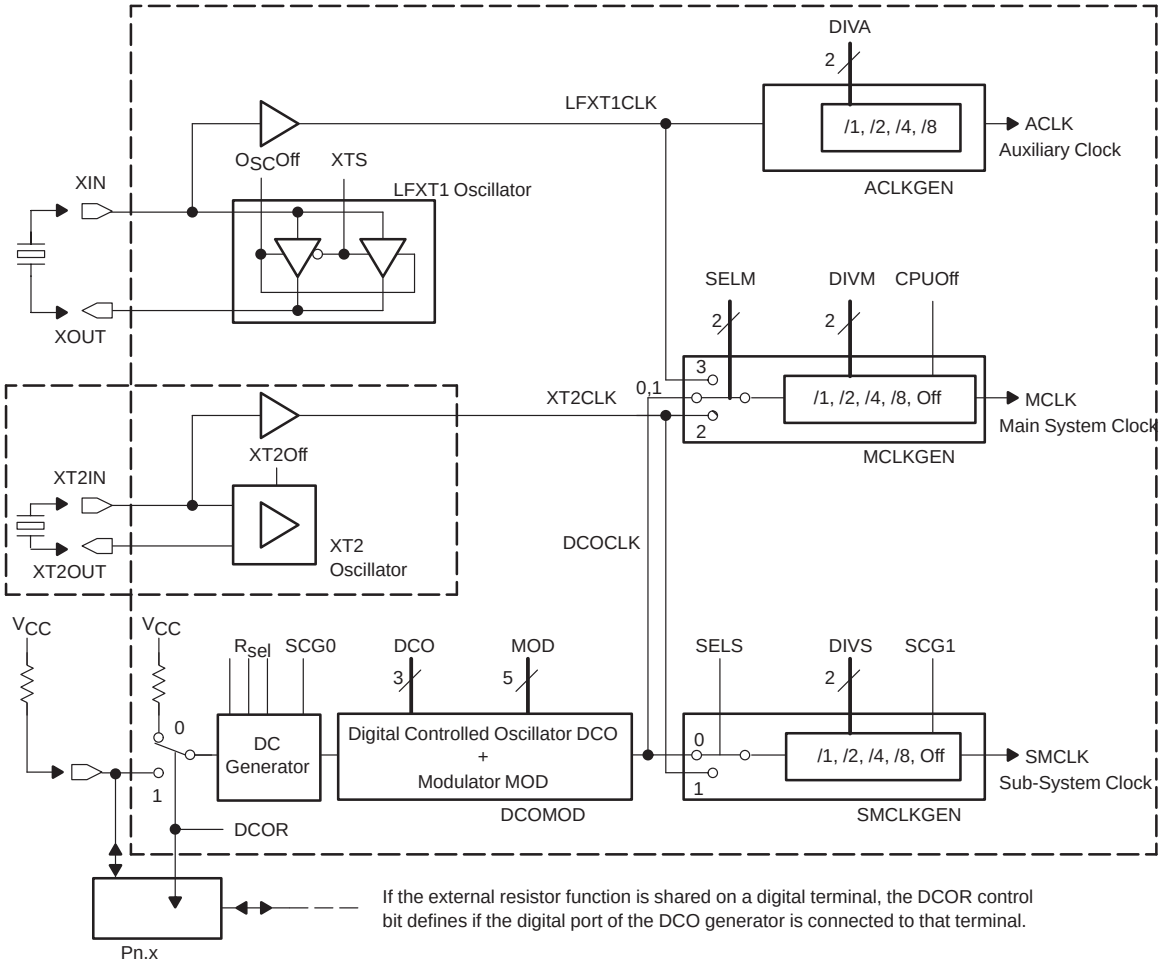
This chapter discusses the Basic Clock Module used in the MSP430x1xx families.

Topic **Page**

7.1 Basic Clock Module

The Basic Clock Module (shown in Figure 7–1) follows the major targets of low system cost and low power consumption. Using three internal clock signals, the design engineer can select the best balance of performance and low power consumption. The Basic Clock Module can be configured to operate without any external components, with one external resistor, with one or two external crystals, with resonators, or with clock sources using any combination of the above. The Basic Clock Module is accessible to the CPU as a byte-wide peripheral module.

Figure 7–1. Basic Clock Schematic



Note: The XT2 oscillator is implemented in MSP430F13x and MSP430F14x devices. The LFXT1 signal is used instead of XT2CLK signal in MSP430x11x and MSP430x11x1 devices without XT2 oscillator.

The Basic Clock Module includes two or three clock sources:

- LFXT1CLK low-frequency/high-frequency clock source. One oscillator that can be used with low-frequency watch crystals, standard crystals, resonators, or external clock sources. See the device data sheet for the exact operating frequency range.
- XT2CLK high-frequency clock source. This optional high-frequency oscillator can also use standard crystals, resonators, or external clock sources in the 450-kHz to 8-MHz range. See the device data sheet for the exact operating frequency range and availability of this optional oscillator.
- DCOCLK clock source. One digitally controlled oscillator (DCO) with RC-type characteristics.

Three clock signals are available from the Basic Clock Module:

- ACLK auxiliary clock. The ACLK is the buffered LFXT1CLK clock source divided by 1, 2, 4, or 8. Software selects the division factor. ACLK is software selectable for individual peripheral modules.
- MCLK master clock. MCLK is software selectable as LFXT1CLK, XT2CLK (if available), or DCOCLK. MCLK is divided by 1, 2, 4, or 8. Software selects the division factor. MCLK is used by the CPU and system.
- SMCLK submain clock. SMCLK is software selectable as LFXT1CLK, XT2CLK (if available), or DCOCLK. SMCLK is divided by 1, 2, 4, or 8. Software selects the division factor. SMCLK is software selectable for individual peripheral modules.

7.2 LFXT1 and XT2 Oscillators

The Basic Clock Module includes the LFXT1 oscillator and, in some configurations, a second XT2 oscillator.

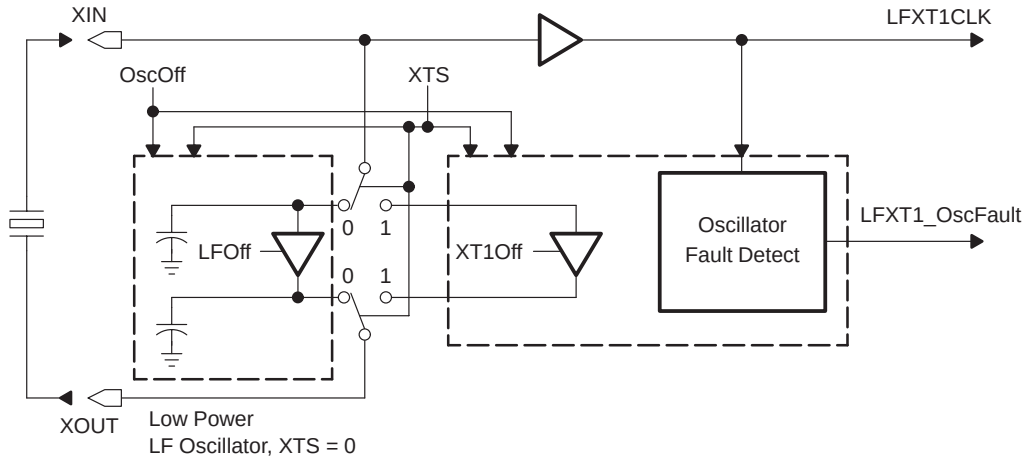
7.2.1 LFXT1 Oscillator

The LFXT1 oscillator starts operating on a valid PUC condition. A valid PUC condition resets the OscOff bit in the status register, which enables LFXT1. Software can disable LFXT1 by setting OscOff, if this signal does not source SMCLK or MCLK.

The design of the LFXT1 oscillator (shown in Figure 7–2) supports the low-current consumption feature and the use of a 32,768-Hz watch crystal when in LF mode (XTS=0). A watch crystal connects to the clock module via two terminals without any other external components. Components necessary to stabilize the clock operation have been integrated into the MSP430.

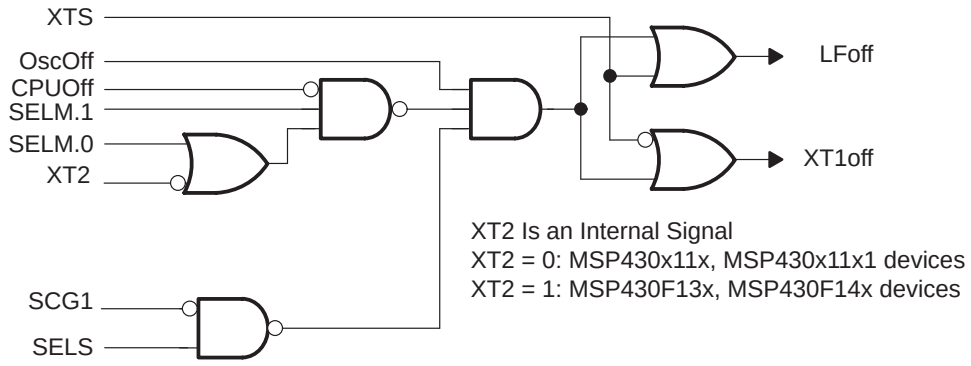
The design of the LFXT1 oscillator also supports high-speed crystals or resonators when in HF mode (XTS = 1). The crystal or resonator connects to the terminals and requires external capacitors on both terminals. These capacitors should be sized according to crystal or resonator specifications.

Figure 7–2. Principle of LFXT1 Oscillator



The OscOff bit in the status register is used to turn off LFXT1CLK if this signal does not source MCLK or SMCLK.

Figure 7–3. Off Signals for the LFXT1 Oscillator



OscOff	CPUOff	SELM.x	SCG1	SELS	LFXT1CLK	Comment
0	x	x	x	x	on	Oscillator Active
1	0	3 or 2+3†	x	x	on‡	Clock Signal Needed For MCLK
1	x	x	0	1	on‡	Clock Signal Needed For SMCLK

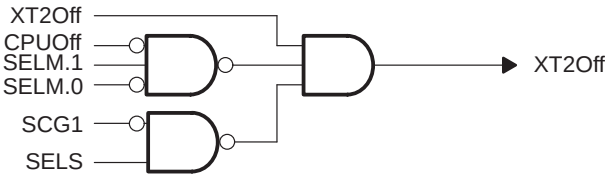
† Two oscillators: SELM.x = 3. Three oscillators: SELM.x = 3 and 2
‡ LFXT1CLK is switched off for all other bit combinations

7.2.2 XT2 Oscillator

A second oscillator, XT2, is available in MSP430F13x and MSP430F14x devices. XT2 sources XT2CLK, and its characteristics are identical to LFXT1 in HF mode.

The XT2Off control bit disables the XT2 oscillator if signal XT2CLK is not used for MCLK or SMCLK. If the CPUOff bit is reset and SELM = 2, XT2CLK sources MCLK . The XT2CLK sources SMCLK if SCG1 is reset and SELS = 1.

Figure 7–4. Off Signals for Oscillator XT2



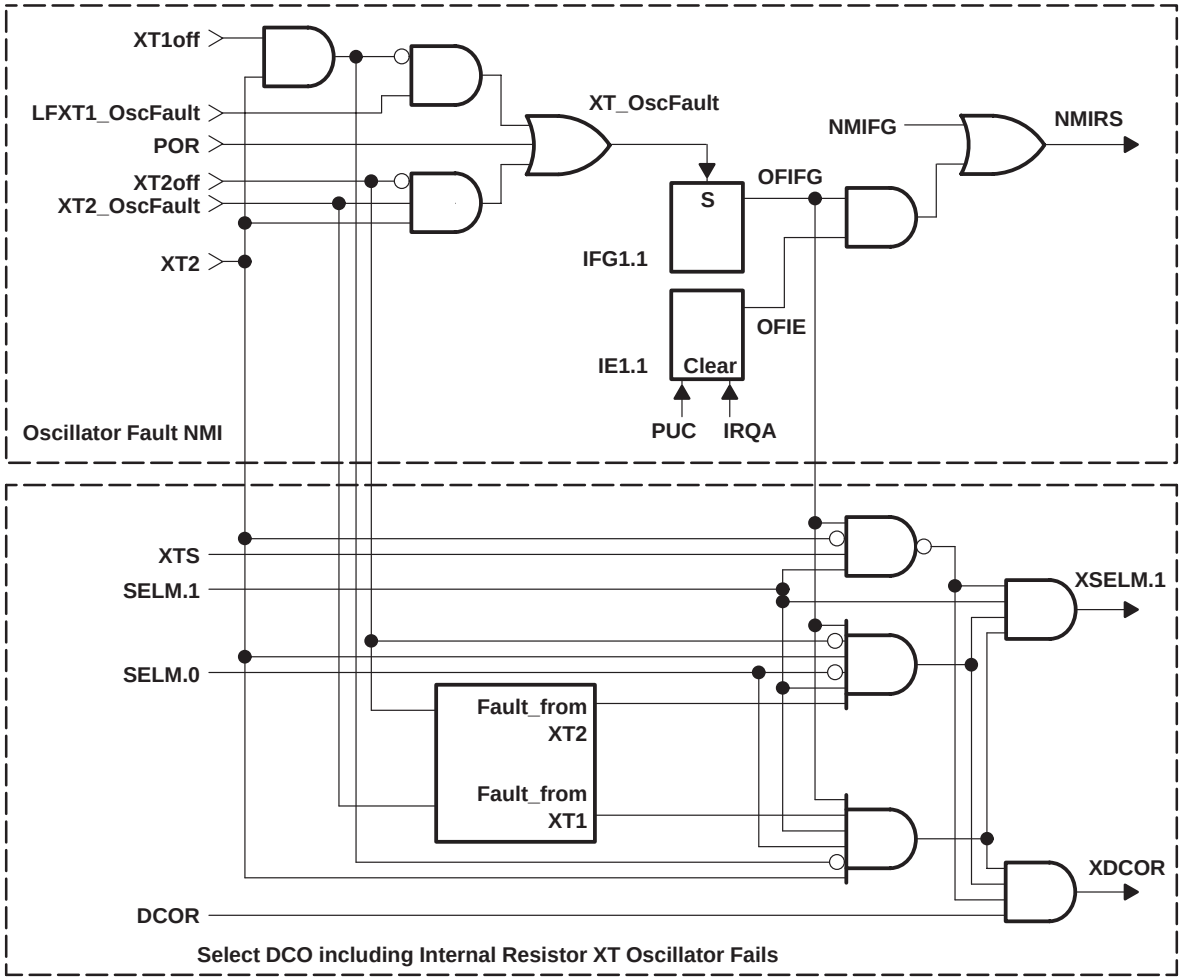
OscOff	CPUOff	SELM.x	SCG1	SELS	XT2CLK	Comment
0	x	x	x	x	on	Oscillator Active
1	0	2	x	x	on†	Clock Signal Needed For MCLK
1	x	x	0	1	on†	Clock Signal Needed For SMCLK

† XT2CLK is switched off in all other bit combinations

7.2.3 Oscillator Fault Detection

An analog circuit controls the operation of oscillators LFXT1 and XT2 and flags an oscillator fault when approximately 100 cycles (at 1 MHz) of the crystal are missed. The active OSCFault signal sets the oscillator-fault-interrupt flag (OFIFG) and requests a non-maskable interrupt when the oscillator-fault interrupt enable bit (OFIE) is set. User software must clear the OFIFG flag.

Figure 7–5. Oscillator-Fault-Interrupt

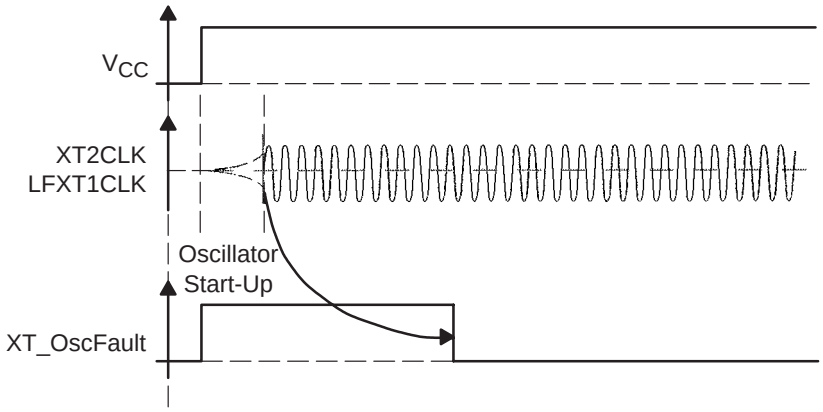


XT2 Is an internal signal
XT2 = 0: Two oscillators – MSP430x11x and MSP430x11x1 devices
XT2 = 1: Three oscillators – MSP430F13x and MSP430F14x devices
IRQA: Interrupt request accepted

The XT2_OscFault signal of the XT2 oscillator, and the LFXT1_OscFault signal of the LFXT1 oscillator will set the oscillator-fault-interrupt flag (OFIFG) independently. During two-oscillator implementation, the oscillator-fault-interrupt flag can be reset by software if the LFXT1_OscFault signal is a logic low. During three-oscillator implementation, the oscillator-fault-interrupt flag can be reset by software only if both oscillator-fault signals, LFXT1_OscFault and XT2_OscFault, are low.

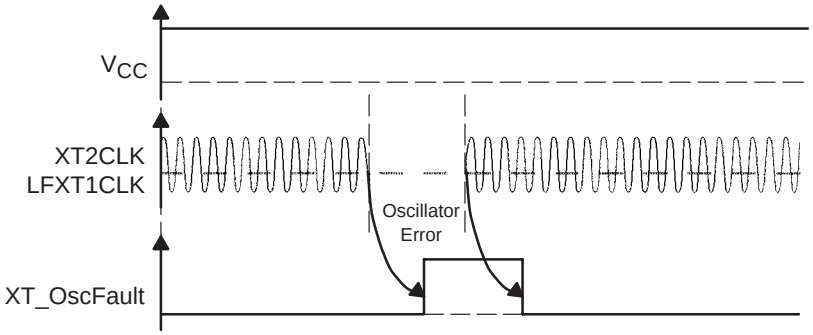
After applying V_{CC} the oscillator fault signal (XT_OscFault) becomes active. The XT_OscFault signal becomes inactive when XT2CLK and/or LFXT1CLK begin oscillating, which takes approximately 50 clock cycles (at 1 MHz).

Figure 7–6. Oscillator-Fault Signal



XT_OscFault becomes active after XT2CLK and/or LFXT1CLK stop oscillating. The delay associated with the XT_OscFault signal is approximately 50 clock cycles (at 1 MHz).

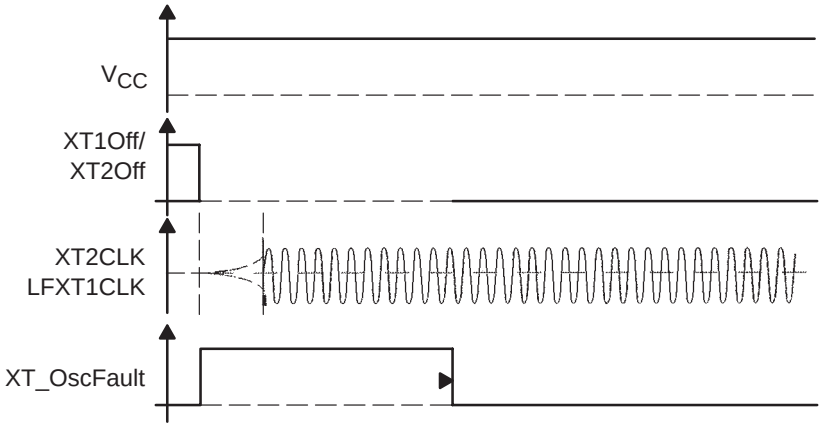
Figure 7–7. Oscillator Fault in Oscillator Error Condition



The oscillator-fault signal returns to a logic low if the oscillator starts operating again. The delay is typically 50 clock cycles (at 1 MHz).

When an XT oscillator has stopped and is then restarted, the oscillator fault signal (XT_OscFault) remains active until the oscillator starts operating, and becomes inactive after a delay of typically 50 clock cycles at (1 MHz).

Figure 7–8. Oscillator Fault in Oscillator Error Condition at Start-Up



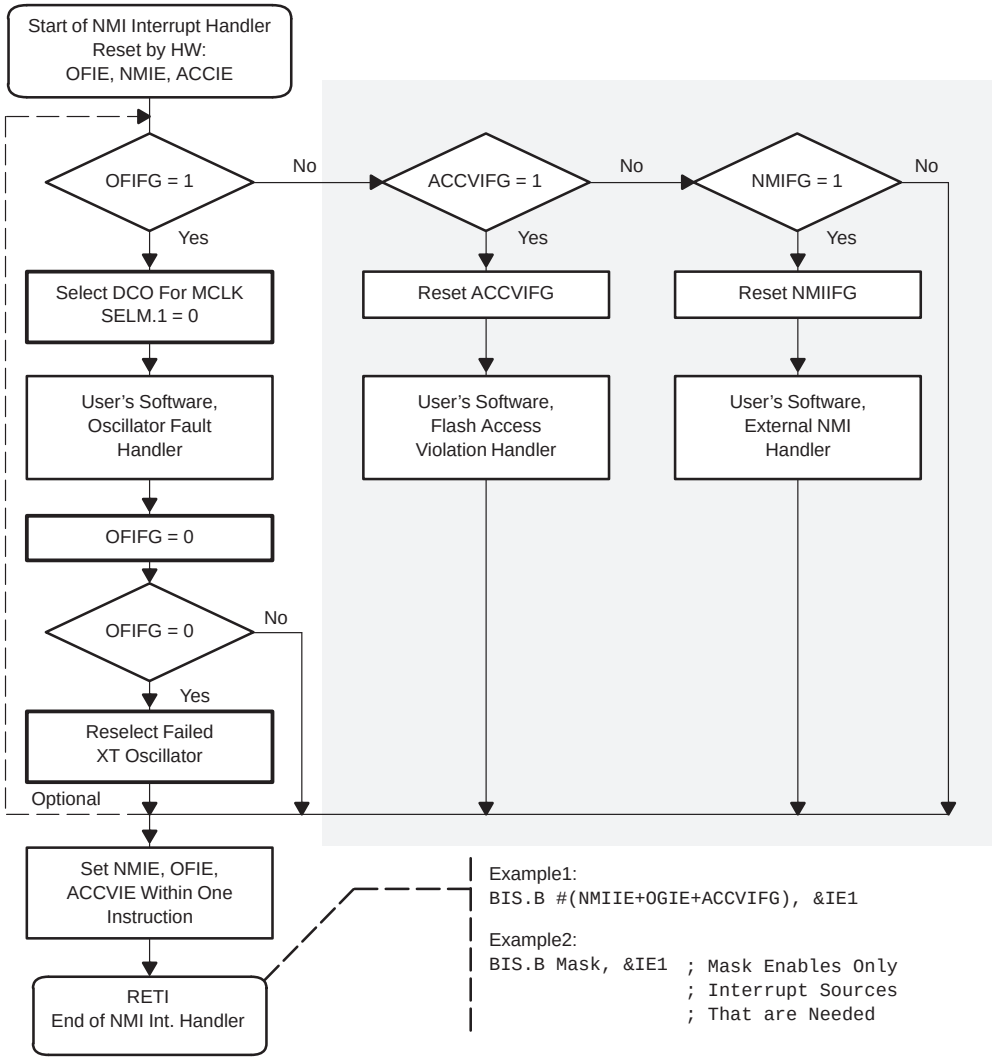
7.2.4 Select DCO Oscillator for MCLK on XT Oscillator Fault

The DCO oscillator is selected automatically for MCLK if either one of the oscillators LFXT1 (in HF mode only), or XT2 is selected for MCLK source and this oscillator fails. Since the DCO oscillator is now selected, the NMI requested by the oscillator fault can be processed. The DCO oscillator is switched on and the DCOCLK is switched to be the source for MCLK. An NMI is processed even if the CPU is switched off (CPUOFF=1).

The MSP430 ultralow-power system allows any enabled interrupt to be serviced from any low-power mode, including LPM4. MCLK automatically becomes active inside of an interrupt service routine.

An NMI interrupt routine written by the user, which handles the oscillator fault, has some important steps that should be used for proper handling of the fault situation. First, if the NMI interrupt routine detects the oscillator fault and selects the DCO as clock source for the system clock MCLK. Second, the user programs a proper algorithm to detect if the XT oscillator is working again and reselects an XT oscillator for the system clock MCLK if desired.

Figure 7–9. NMI/OSCFault Interrupt Handler

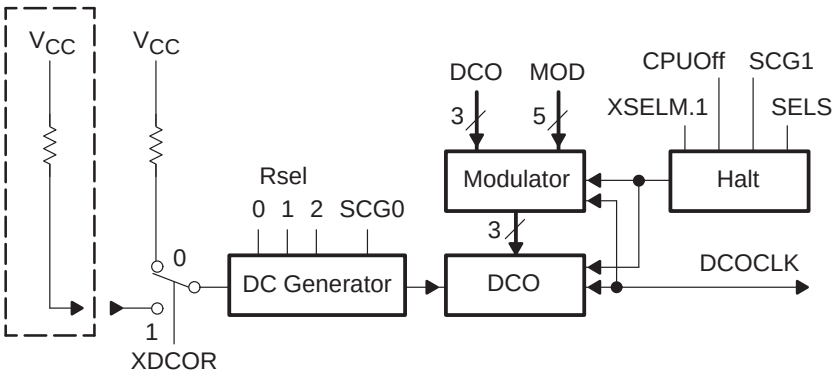


Note: Example for MSP430F1xx device

7.3 Digitally-Controlled Oscillator (DCO)

The DCO is an integrated RC-type oscillator in the Basic Clock Module. The DCO frequency can be tuned by software using the DCO, MOD, and RSEL bits. The DCO is absolutely monotonic. As with any RC-type oscillator, frequency varies with temperature, voltage, and from device to device. The digital control of the oscillator allows frequency stabilization despite its RC-type characteristics.

Figure 7–10. DCO Schematic



The dc generator, when switched off, requires some minimal start-up time (4 microsecond range) due to its low-current design. Once the current is switched on, the resistor injects current in the microampere range into the dc generator; the internal and external parasitic capacitances introduce the delay in the microsecond range. No delay occurs in operating modes that do not require to switch off the dc generator current.

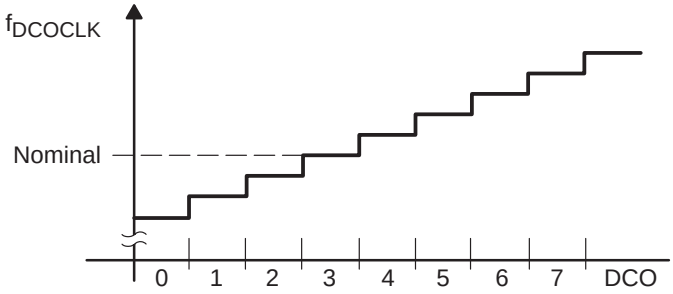
An internal or external resistor is connected to the dc generator, which determines the operating fundamental frequency of the DCOCLK.

The frequency of DCOCLK is set by the following functions:

- The current injected into the dc generator (DCGEN) by either the internal or external resistor defines the fundamental frequency. Control bit DCOR selects the internal or external resistor.
- Control bits Rsel2, Rsel1, and Rsel0 divide the fundamental frequency into eight nominal frequency ranges. These ranges are defined for an individual device in the appropriate data sheet.
- The three control bits, DCO0 to DCO2, adjust the DCOCLK frequency.
- The five modulation bits, MOD0 to MOD4, switch between the frequency selected by the DCO bits and the next higher frequency set by DCO+1.

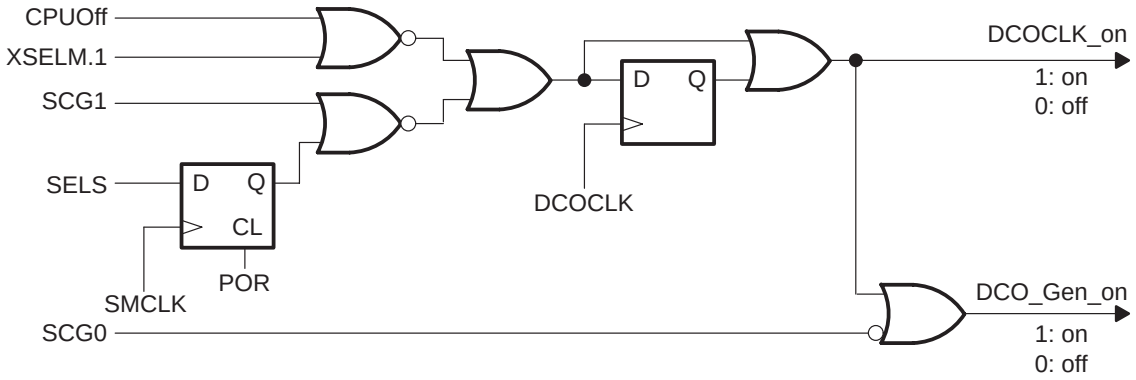
The clock period of the DCOCLK signal changes approximately ten percent for each step of the control bit DCO.

Figure 7–11. Principle Period Steps of the DCO



Five bits (SCG0, CPUOff, SELM.1, SCG1, and SELS) control the operation of the DCO.

Figure 7–12. On/Off Control of DCO



SCG0	CPUOff	SELM.1	SCG1	SELS	DCOCLK	DCOCLK	Comment
x x	0 x	0 x	x 0	x 0	on on	on on	DCO Clock Needed For MCLK DCO Clock Needed For SMCLK
0 0	0 1	1 x	(see Note A)		off off	on on	DCO Clock is Not Needed For MCLK (and SMCLK)
0 0	(see Note B)		0 1	1 x	off off	on on	DCO Clock is Not Needed For SMCLK (and MCLK)
1	(see Note C)				off	off	DCO Clock is Not Needed: SCG0 Bit Switches Off DCOGEN

NOTES: A. SMCLK does not need the DCOCLK signal if:
SMCLK is switched off (SCG1 = 1), or DOCCLK is not selected for SMCLK (SELS = 0).
B. MCLK does not need the DCOCLK signal if:
MCLK is switched off (CPUOff = 1), or DOCCLK is not selected for MCLK (SELM.1 = 1).
C. MCLK and SMCLK does not need the DCOCLK signal if:
The control bit SCG0 in the status register can switch off (SCG0 = 1) the DCOGEN.

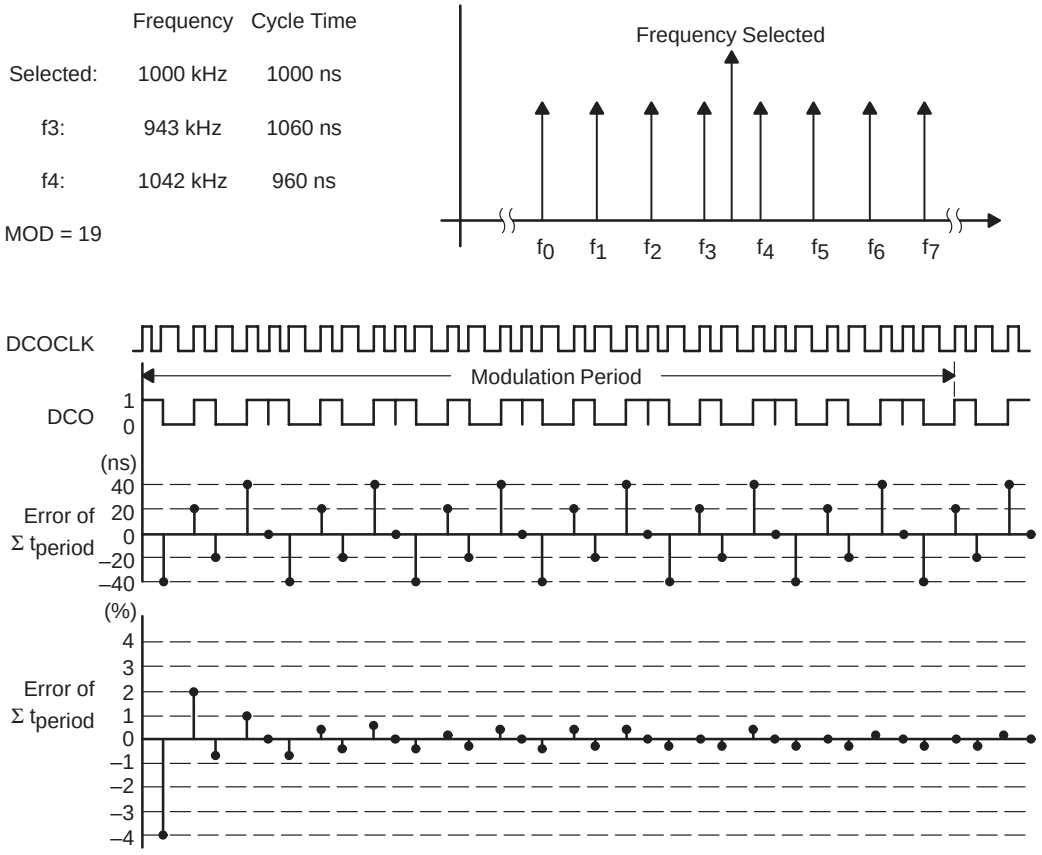
7.3.1 Operation of the DCO Modulator

The modulator is intended to reduce a long accumulating period variation by mixing adjacent DCO periods. On average, a longer period variation can be minimized by mixing DCO periods. The modulator accumulates a period of 32 DCOCLK clock cycles. The MOD control bits define the mixing ratio of the DCO+1 period. The remaining 32-MOD time slots use the DCO period. If the modulation constant is 0, the DCO data in the control register defines the period. The following formula defines the accumulating periods:

$$t = (32 - MOD) \times t_{DCO} + MOD \times t_{DCO+1}$$

The modulator selects f_{dco} or $f_{dco}+1$ individually for each DCO cycle. This is the highest possible rate that can be modulated between two discrete frequency steps. The following example illustrates the main operation of the modulator.

Figure 7–13. Operation of the DCO Modulator



The user should consider two factors when reviewing the timing accuracy generated from the DCOCLK signal:

- Short term accuracy: Each individual cycle is as inaccurate as the DCO steps.
- Long term accuracy: The accumulated average of many individual cycles reduces the relative error down to less than 1/3 percent, assuming a step delta of 10% and a modulation period of 32.

Proper use of the modulation feature on the DCOCLK period increases the accuracy by averaging the periods. The selected frequency set using the control bits in the DCO and the modulation fraction defined by the control bits in MOD sets the DCOCLK periods.

Note: Control of DCOCLK Frequency

The frequency of the digitally-controlled oscillator varies with temperature and voltage and is different for each individual sample. The frequency can be controlled by software if an external reference (such as the ACLK signal) is used to measure the difference and to readjust the DCO frequency.

7.4 Basic Clock Module Operating Modes

Control bits SCG0, SCG1, OscOff, and CPUOff in the status register configure the operating mode, as discussed in Chapter 3, *System Resets, Interrupts and Operating Modes*.

The digitally-controlled oscillator is disabled when not used for MCLK or SMCLK. The dc generator must be switched off separately, but is switched on automatically when the DCOCLK signal is used, either for MCLK or SMCLK.

7.4.1 Starting From Power Up Clear (PUC)

On a valid PUC, the internal resistor is selected for the dc generator, $R_{sel} = 4$, and DCO = 3, allowing the oscillator to operate at a medium frequency and independently from external conditions. ACLK is sourced from LFXT1 in the LF mode and configured to operate with a watch crystal; MCLK and SMCLK are sourced from DCOCLK. Because the CPU executes code from MCLK, which is sourced from the fast-starting DCO, code execution from PUC is fast, typically less than 6 µs. After a PUC, user software selects the best basic clock configuration for the application.

7.4.2 Adjusting the Basic Clock

The control registers of the Basic Clock are under full software control. If clock requirements other than those of the default from PUC are necessary, the Basic Clock can be configured or reconfigured by software at any time during program execution.

- ACLKGEN from LFXT1 crystal, resonator, or external-clock source and divided by 1, 2, 4, or 8. If no LFXTCLK clock signal is needed in the application, the OscOff bit should be set in the status register.
- SCLKGEN from LFXTCLK, DCOCLK, or XT2CLK (x13x and x14x only) and divided by 1, 2, 4, or 8. The SCG1 bit in the status register enables or disables SMCLK.
- MCLKGEN from LFXTCLK, DCOCLK, or XT2CLK (x13x and x14x only) and divided by 1, 2, 4, or 8. When set, the CPUOff bit in the status register enables or disables MCLK.
- DCOCLK frequency is adjusted using the RSEL, DCO, and MOD bits. The DCOCLK clock source is stopped when not used, and the dc generator can be disabled by the SCG0 bit in the status register (when set).
- The XT2 oscillator sources XT2CLK (x13x and x14x only) by clearing the XT2Off bit.

User software can modify the Basic Clock to meet the system requirements at any time using the full MSP430 instruction set. A few examples follow:

```
bis.b  #007h,&BCSCTL1    ; RSEL=7
mov.b  #081h,&BCSCTL1    ; XT2off, RSEL=1
bis.b  #070h,&BCSCTL1    ; ACLK= high-speed XTAL/8
bis.b  #008h,&BCSCTL2    ; SMCLK=LFXT1
inc.b  &DCOCTL           ; Increase DCOCLK
dec.b  &DCOCTL           ; Decrease DCOCLK
```

7.4.3 Basic Clock Features for Low-Power Applications

Conflicting requirements typically exist in battery powered MSP430x1xx applications:

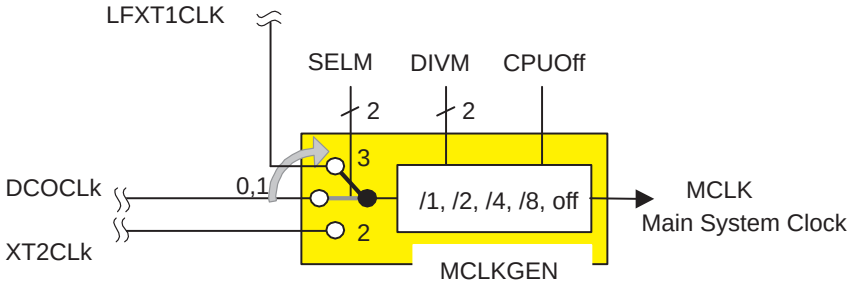
- Low clock frequency for energy conservation and time keeping
- High clock frequency for fast reaction to events and fast burst processing capability

The Basic Clock Module addresses the above conflicting requirements by allowing the design engineer to select from the three available clock signals: ACLK, MCLK and SMCLK. For optimal low-power performance, the ACLK can be configured to oscillate with a low 32,786-Hz watch-crystal frequency, providing a stable time base for the system and low power stand-by operation. The MCLK can be configured to operate from the on-chip DCO which is only activated when requested by events. The SMCLK can be configured to operate from either the watch-crystal or the DCO, depending on peripheral requirements. A flexible clock distribution and divider system is provided to fine tune the individual clock requirements. All basic clock-module configurations are under full software control.

7.4.4 Selecting a Crystal Clock for MCLK

After power up, the basic clock module uses the DCO clock for the system clock MCLK. The LFXT1 oscillator starts in the low-frequency mode (XTS=0). Regardless of the configuration of the clock system the application uses, if all initial conditions are set, the software execution is ensured by the integrated DCO. Finally, the application may use a crystal for further software execution.

Figure 7–14. Select Crystal Oscillator for MCLK, Example Uses LFXT1 for MCLK



The sequence to switch the MCLK source from the DCO clock to the crystal clock (LFXT1CLK or XT2CLK) should be:

1) Switch on the crystal oscillator

   a) OscOff = 0, XTS is reset (LF mode is selected) or set (HF mode is selected)

```
BIS.B   #XTS,&BCSCTL1  ; Set XTS bit to select LFXT1
                       ; oscillator
BIC     #OSCOFF,SR     ; Turn on LFXT1 oscillator
```

b) In x13x and x14x devices, the XT2 oscillator can also be used: XT2Off is reset

```
BIC.B   #XT2Off,&BCSCTL1  ; Reset XT2Off to turn on LFXT1
                          ; oscillator
```

2) Wait until the oscillator, which will be used for MCLK, is settled. Three possibilities are:

a) Always wait enough time so the oscillator is settled under all conditions. The time differs with the crystal type (watch crystal, standard crystal, resonator) and needs to be evaluated with the application.

```
        PUSH    #Wait    ; Load wait time constant onto stack
Loop    DEC     0 (SP)   ; Count wait time down to zero
        JNZ     Loop     ; Wait time not completed > go to loop
                         ; Wait time = t(MCLK) x (4+2)
        BIC.B   #OFIFG,&IFG1           ; Clear osc. fault int. flag
        BIS.B   #(SELM1+SELM0),&BCSCTL2  ; Select now LFXT1 clock
```

b) Wait until the oscillator-fault test (reset OFIFG and then test if OFIFG is reset) indicates that the oscillator has started. Then wait enough time so that the oscillator amplitude is settled.

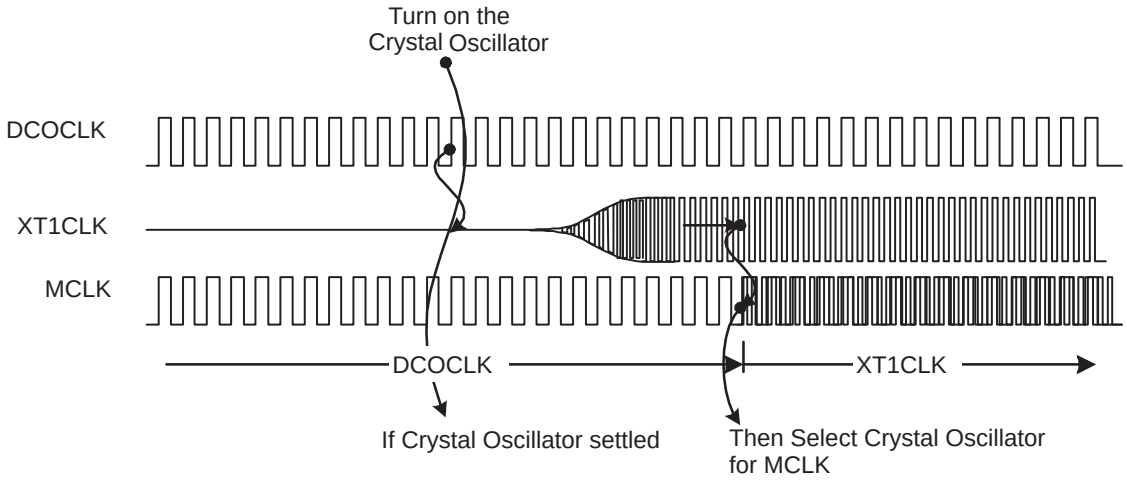
```
TST_OF  BIC.B   #OFIFG,&IFG1 ; Clear osc. fault int. flag
Loop    DEC     0 (SP)  ; Count wait time down to zero
        JNZ     Loop    ; Wait time not completed > go to loop
        BIT.B   #OFIFG,&IFG1   ; Test oscillator fault flag
        JNZ     TST_OF  ; Repeat test until flag remains reset

        PUSH    #Wait   ; Load wait time constant onto stack

        BIC.B   #OFIFG,&IFG1           ; Clear osc. fault int. flag
        BIS.B   #(SELM1+SELM0),&BCSCTL2  ; Select now LFXT1 clock
```

c) Wait until the oscillator-fault test (reset OFIFG and then test if OFIFG is reset) indicates that the oscillator has started. Then count pulses from the oscillator. A typical number is 1024 pulses and needs to be verified under the final application conditions.

```
TST_OF  BIC.B   #OFIFG,&IFG1  ; Clear oscillator fault interrupt
                              ; flag
        BIT_B   #OFIFG,&IFG1  ; Test oscillator fault flag
        JNZ     TST_OF  ; Repeat test until flag remains reset

;* Use Watchdog/Timer with ACLK/2, SSEL=1, IS1=1, IS0=0, RST/NMI is
;* reset pin and Watchdog/Timer in Timer function, counter is
;* cleared.
        BIS.B   #DIVA0,&BCSCTL1 ; Assumes that DIVA.1 bit is reset
        BIC.B   #WDTIE,&IE1     ; No interrupt from Watchdog/Timer
        BIC.B   #WDTIFG,&IFG1   ; Clear WDTIFG: it indicates 1024
                                ; pulses from XT1
        MOV     #(WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTIS1),&WDTCTL
Retest  BIT.B   #WDTIFG,&IFG1   ; Test if 1024 pulses completed
```

```
JNC     Retest
BIC.B   #OFIFG,&IFG1            ; Clear osc. fault int. flag
BIS.B   #(SELM1+SELM0),&BCSCTL2  ; Select now LFXT1 clock
```

Figure 7–15. Timing to Select Crystal Oscillator for MCLK, Example Uses LFXT1 in HF Mode for MCLK



7.4.5 Synchronization of Clock Signals

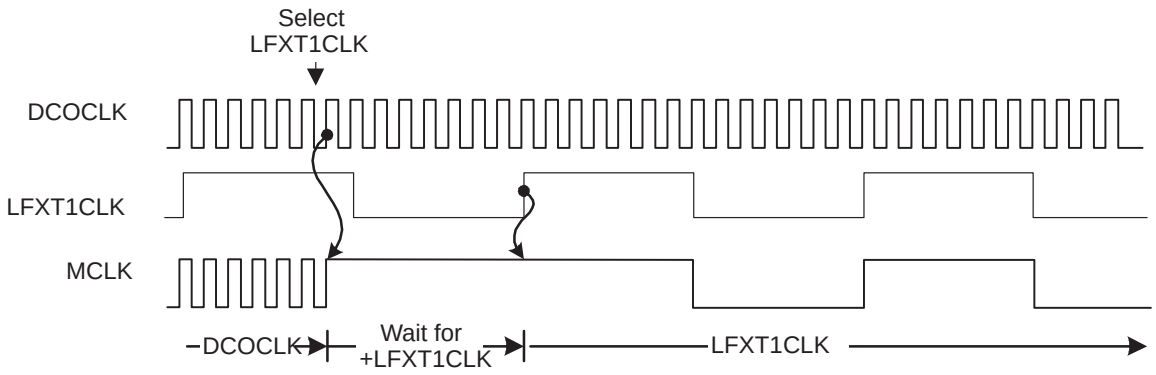
The clock signals MCLK and SMCLK can be supplied by different clock sources. While switching from one clock source to the other, the switch is synchronized to avoid critical race conditions. When another clock source is selected the following occurs in order:

1) The current clock cycle continues until the next rising edge.

2) The clock then remains high until the next rising edge of the newly selected clock.

3) Now the new clock source is selected and continues with a full high period.

Figure 7–16. Select Another Clock Source Signal, Example Switches From DCOCLK to LFXT1CLK for Clock MCLK



7.5 Basic Clock Module Control Registers

The Basic Clock Module is configured using control registers DCOCTL, BCSCTL1, and BCSCTL2, and four bits from the CPU status register: SCG1, SCG0, OscOff, and CPUOFF. User software can modify these control registers from their default condition at any time. The Basic Clock Module control registers are located in the byte-wide peripheral map and should be accessed with byte (.B) instructions.

Register	Short Form	Register Type	Address	Initial State
DCO control register	DCOCTL	Read/write	056h	060h
Basic clock system control 1	BCSCTL1	Read/write	057h	084h
Basic clock system control 1	BCSCTL2	Read/write	058h	reset

7.5.1 Digitally-Controlled Oscillator (DCO) Clock-Frequency Control

DCOCTL is loaded with a value of 060h with a valid PUC condition.

DCOCTL 056h

7							0
DCO.2	DCO.1	DCO.0	MOD.4	MOD.3	MOD.2	MOD.1	MOD.0
rw–0	rw–1	rw–1	rw–0	rw–0	rw–0	rw–0	rw–0

MOD.0 .. MOD.4: The MOD constant defines how often the discrete frequency f_{DCO+1} is used within a period of 32 DCOCLK cycles. During the remaining clock cycles (32–MOD) the discrete frequency f_{DCO} is used. When the DCO constant is set to seven, no modulation is possible since the highest feasible frequency has then been selected.

DCO.0 .. DCO.2: The DCO constant defines which one of the eight discrete frequencies is selected. The frequency is defined by the current injected into the dc generator.

7.5.2 Oscillator and Clock Control Register

BCSCTL1 is affected by a valid PUC or POR condition.

BCSCTL1 057h

7							0
XT2Off	XTS	DIVA.1	DIVA.0	XT5V	Rsel.2	Rsel.1	Rsel.0
rw–(1)	rw–(0)	rw–(0)	rw–(0)	rw–0	rw–1	rw–0	rw–0

Bit0 to Bit2: The internal resistor is selected in eight different steps.
Rsel.0 to Rsel.2 The value of the resistor defines the nominal frequency. The lowest nominal frequency is selected by setting Rsel=0.

Bit3, XT5V: XT5V should always be reset.

Bit4 to Bit5: The selected source for ACLK is divided by:
DIVA = 0: 1
DIVA = 1: 2
DIVA = 2: 4
DIVA = 3: 8

Bit6, XTS: The LFXT1 oscillator operates with a low-frequency clock crystal or with a high-frequency crystal:
XTS = 0: The low-frequency oscillator is selected.
XTS = 1: The high-frequency oscillator is selected.

The oscillator selection must meet the external crystal's operating condition.

Bit7, XT2Off: The XT2 oscillator is switched on or off:
XT2Off = 0: the oscillator is on
XT2Off = 1: the oscillator is off if it is not used for MCLK or SMCLK.

BCSCTL2 is affected by a valid PUC or POR condition.

BCSCTL2
058h

7							0
SELM.1	SELM.0	DIVM.1	DIVM.0	SELS	DIVS.1	DIVS.0	DCOR
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–0	rw–0	rw–0	rw–0

Bit0, DCOR: The DCOR bit selects the resistor for injecting current into the dc generator. Based on this current, the oscillator operates if activated.
DCOR = 0: Internal resistor on, the oscillator can operate. The failsafe mode is on.
DCOR = 1: Internal resistor off, the current must be injected externally if the DCO output drives any clock using the DCOCLK.

Bit1, Bit2: The selected source for SMCLK is divided by:

DIVS.1 .. DIVS.0 DIVS = 0: 1
DIVS = 1: 2
DIVS = 2: 4
DIVS = 3: 8

Bit3, SELS: Selects the source for generating SMCLK:
SELS = 0: Use the DCOCLK
SELS = 1: Use the XT2CLK signal (in three-oscillator systems)
or
LFXT1CLK signal (in two-oscillator systems)

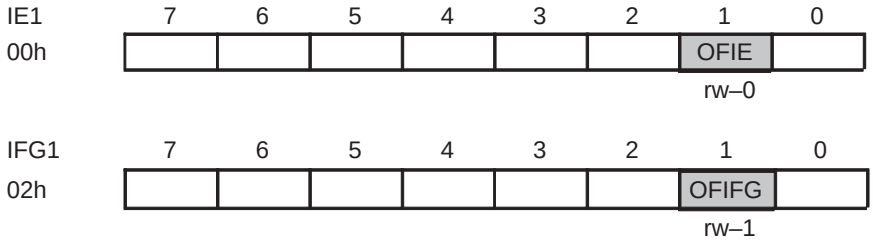
Bit4, Bit5: The selected source for MCLK is divided by:

DIVM.0 .. DIVM.1 DIVM = 0: 1
DIVM = 1: 2
DIVM = 2: 4
DIVM = 3: 8

Bit6, Bit7: Selects the source for generating MCLK:
SELM.0 .. SELM.1 SELM = 0: Use the DCOCLK
SELM = 1: Use the DCOCLK
SELM = 2: Use the XT2CLK (x13x and x14x devices) or
Use the LFXT1CLK (x11x(1) devices)
SELM = 3: Use the LFXT1CLK

7.5.3 Special-Function Register Bits

The Basic Clock Module affects two bits in the special-function registers OFIFG and OFIE. The oscillator fault-interrupt enable bit (OFIE) is located in bit 1 of the interrupt-enable register IE1. The oscillator fault-interrupt flag bit (OFIFG) is located in bit 1 of the interrupt-flag register IFG1.



The oscillator fault signal XT_OscFault sets the OFIFG as long as the oscillator fault condition is active. The detection and effect of the oscillator fault condition is described in section 7.4.1. The oscillator fault interrupt requests a nonmaskable interrupt if the OFIE bit is set. The oscillator interrupt-enable bit is reset automatically if a non-maskable interrupt is accepted. The initial state of the OFIE bit is reset and no oscillator fault requests an interrupt even if a fault condition occurs.

Chapter 8

Digital I/O Configuration

This chapter describes the digital I/O configuration.

Topic **Page**

8.1 Introduction

The general-purpose I/O ports of the MSP430 are designed to give maximum flexibility. Each I/O line is individually configurable, and most have interrupt capability.

There are two different types of I/O port modules in the MSP430x1xx family devices. Ports P1 and P2 are of one type, and ports P3 to P6 are of another type. Both types have the capability to control input/output direction and output level, to read the level applied to a pin, and to control if a port or module function is applied to a pin. The port module for P1 and P2 have interrupt capability; flag, enable, and edge sensitivity are available individually for each bit.

MSP430x11x and MSP430x11x1 devices have ports P1 and P2 implemented; MSP430x13x and MSP430x14x have ports P1 to P6 implemented.

8.2 Ports P1, P2

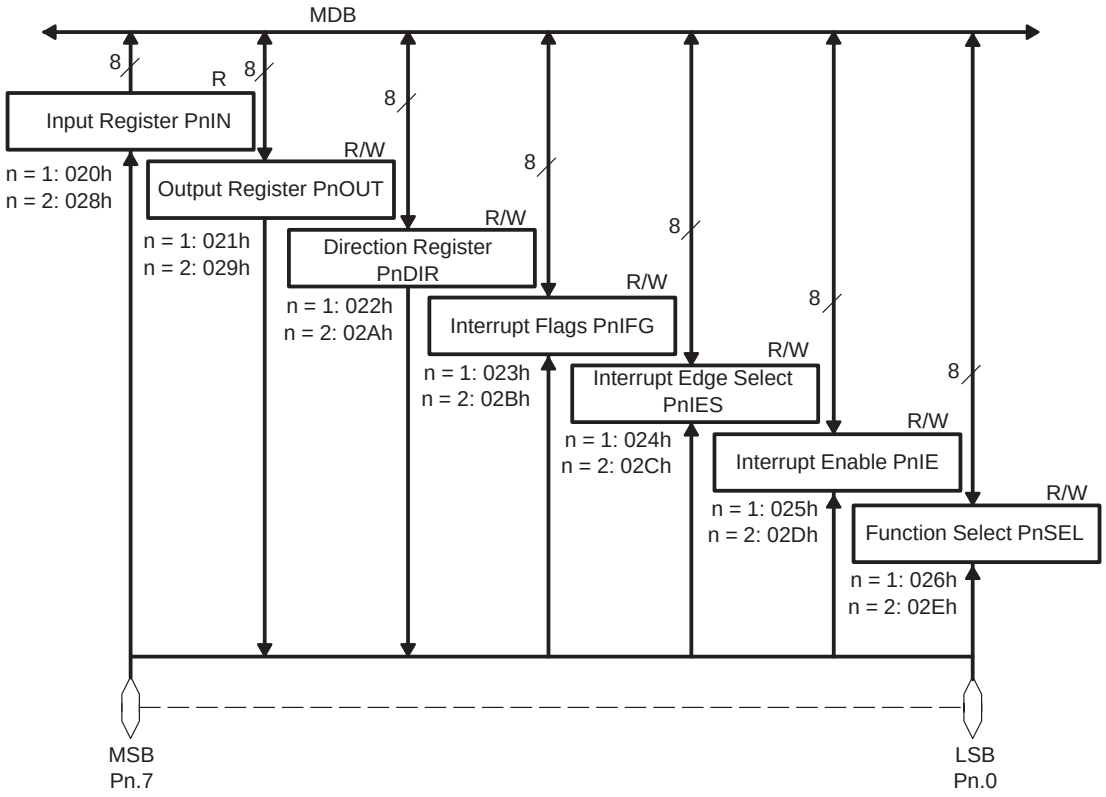
Each of the general-purpose ports P1 and P2 contain 8 general-purpose I/O lines and all of the registers required to control and configure them. Each I/O line is capable of being controlled independently. In addition, each I/O line is capable of producing an interrupt.

Separate vectors are allocated to ports P1 and P2 modules. The pins for port P1 (P1.0–7) source one interrupt, and the pins for port P2 (P2.0–7) source another interrupt.

Seven registers are used to control the port I/O pins (see Section 8.2.1).

Ports P1 and P2 are connected to the processor core through the 8-bit MDB and the MAB. They should be accessed using byte instructions in the absolute address mode.

Figure 8–1. Port P1, Port P2 Configuration



8.2.1 Port P1, Port P2 Control Registers

The seven control registers give maximum digital input/output configuration flexibility:

- ❑ All individual I/O bits are independently programmable.
- ❑ Any combination of input, output, and interrupt condition is possible.
- ❑ Interrupt processing of external events is fully implemented for all eight bits of ports P1 and P2.

The seven registers for port P1 and the seven registers for port P2 are shown in Table 8–1 and Table 8–2, respectively.

Table 8–1. Port P1 Registers

Register	Short Form	Register Type	Address	Initial State
Input	P1IN	Read only	020h	– – – – –
Output	P1OUT	Read/write	021h	Unchanged
Direction	P1DIR	Read/write	022h	Reset
Interrupt flags	P1IFG	Read/write	023h	Reset
Interrupt edge select	P1IES	Read/write	024h	Unchanged
Interrupt enable	P1IE	Read/write	025h	Reset
Function select	P1SEL	Read/write	026h	Reset

Table 8–2. Port P2 Registers

Register	Short Form	Register Type	Address	Initial State
Input	P2IN	Read only	028h	– – – – –
Output	P2OUT	Read/write	029h	Unchanged
Direction	P2DIR	Read/write	02Ah	Reset
Interrupt flags	P2IFG	Read/write	02Bh	Reset
Interrupt edge select	P2IES	Read/write	02Ch	Unchanged
Interrupt enable	P2IE	Read/write	02Dh	Reset
Function select	P2SEL	Read/write	02Eh	Reset

These registers contain eight bits, and should be accessed using byte instructions in absolute-address mode.

8.2.1.1 Input Registers P1IN, P2IN

Both Input registers are read-only registers that reflect the signals at the I/O pins.

Note: Writing to Read-Only Registers P1IN, P2IN

Writing to these read-only registers results in increased current consumption while the write attempt is active.

8.2.1.2 Output Registers P1OUT, P2OUT

Each output register shows the information of the output buffer. The output buffer can be modified by all instructions that write to a destination. If read, the contents of the output buffer are independent of pin direction. A direction change does not modify the output buffer contents.

8.2.1.3 Direction Registers P1DIR, P2DIR

The direction registers contain eight independent bits that define the direction of the I/O pin. All bits are reset by the PUC signal.

When:

Bit = 0: The port pin is switched to input direction (3-state)

Bit = 1: The port pin is switched to output direction

8.2.1.4 Interrupt Flags P1IFG, P2IFG

Each interrupt flag register contains eight flags that reflect whether or not an interrupt is pending for the corresponding I/O pin, if the I/O is interrupt-enabled.

When:

Bit = 0: No interrupt is pending

Bit = 1: An interrupt is pending due to a transition at the I/O pin or from software setting the bit.

Note:

Manipulating P1OUT and P1DIR, as well as P2OUT and P2DIR, can result in setting the P1IFG or P2IFG bits.

Writing a zero to an interrupt flag resets it; writing a one to an interrupt flag sets it and generates an interrupt.

Each group of interrupt flags P1FLG.0 to P1FLG.7 and P2FLG.0 to P2FLG.7 sources its own interrupt vector. Interrupt flags P1IFG.0 to P1IFG.7 and P2IFG.0 to P2IFG.7 are not reset automatically when an interrupt from these events is serviced. The software should determine the origin of the interrupt and reset the appropriate flag(s).

Note:

Any external interrupt event should be at least 1.5 times MCLK or longer, to ensure that it is accepted and the corresponding interrupt flag is set.

8.2.1.5 Interrupt Edge Select P1IES, P2IES

Each interrupt edge select register contains a bit for each corresponding I/O pin to select what type of transition triggers the interrupt flag.

When:

Bit = 0: The interrupt flag is set with a low-to-high transition

Bit = 1: The interrupt flag is set with a high-to-low transition

Note:

Changing the P1IES and P2IES bits can result in setting the associated interrupt flags.

PnIES.x	PnIN.x	PnIFG.x
0 → 1	0	Unchanged
0 → 1	1	May be set
1 → 0	0	May be set
1 → 0	1	Unchanged

8.2.1.6 Interrupt Enable P1IE, P2IE

Each interrupt enable register contains bits to enable the interrupt flag for each I/O pin in the port. Each of the sixteen bits corresponding to pins P1.0 to P1.7 and P2.0 to P2.7 is located in the P1IE and P2IE registers.

When:

Bit = 0: The interrupt request is disabled

Bit = 1: The interrupt request is enabled

Note: Port P1, Port P2 Interrupt Sensitivity

Only transitions, not static levels, cause interrupts.

If an interrupt flag is still set when the RETI instruction is executed (for example, a transition occurs during the interrupt service routine), an interrupt occurs again after RETI is completed. This ensures that each transition is acknowledged by the software.

8.2.1.7 Function Select Registers P1SEL, P2SEL

P1 and P2 port pins are often multiplexed with other peripheral modules to reduce overall pin count on MSP430 devices (see the specific device data sheet to determine which other peripherals also use the device pins). Control registers P1SEL and P2SEL are used to select the desired pin function—I/O port or other peripheral module. Each register contains eight bits corresponding to each pin, and each pin's function is individually selectable. All bits in these registers are reset by the PUC signal. The bit definitions are:

Bit = 0: Port P1 or P2 function is selected for the pin

Bit = 1: Other peripheral module function is selected for the pin

Note: Function Select With P1SEL, P2SEL

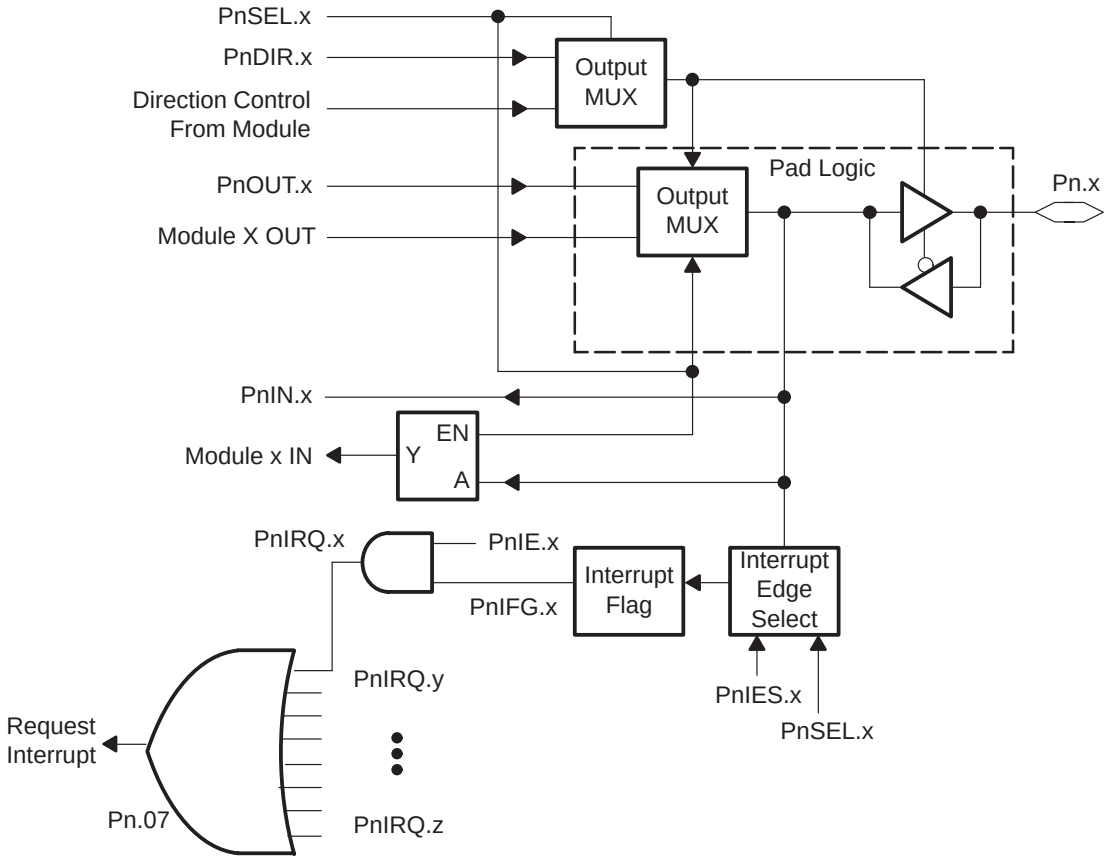
The interrupt-edge-select circuitry is disabled if control bit PnSEL.x is set. Therefore, the input signal can no longer generate an interrupt.

When a port pin is selected to be used as an input to a peripheral module other than the I/O port (PnSEL.x = 1), the actual input signal to the peripheral module is a latched representation of the signal at the device pin (see Figure 8–2 schematic). The latch uses the PnSEL.x bit as its enable, so while PnSEL.x=1, the internal input signal simply follows the signal at the pin. However, if the PnSEL.x bit is reset, then the output of the latch (and therefore the input to the other peripheral module) represents the value of the signal at the device pin just prior to the bit being reset.

8.2.2 Port P1, Port P2 Schematic

The pin logic of each individual port P1 and port P2 signal is identical. Each bit can be read and written to as shown in Figure 8–2.

Figure 8–2. Schematic of One Bit in Port P1, P2



x = 0 to 7, according to bits 0 to 7
n = 1 for Port P1 and 2 for Port P2

8.2.3 Port P1, P2 Interrupt Control Functions

Ports P1 and P2 use eight bits for interrupt flags, eight bits to enable interrupts, eight bits to select the effective edge of an interrupt event, one interrupt vector address for port P1, and one interrupt vector address for port P2.

Each signal uses three bits for configuration and interrupt:

- Interrupt flag, P1IFG.0 to P1IFG.7 and P2IFG.0 to P2IFG.7
- Interrupt enable bit, P1IE.0 to P1IE.7 and P2IE.0 to P2IE.7
- Interrupt edge select bit, P1IES.0 to P1IES.7 and P2IES.0 to P2IES.7

The interrupt flags P1IFG.0 to P1IFG.7 source one interrupt and P2IFG.0 to P2IFG.7 source one interrupt. Any interrupt event on one or more pins of P1.0 to P1.7 or P2.0 to P2.7 requests an interrupt when two conditions are met: the appropriate individual bit PnIE.x is set, and the GIE bit is set. Interrupt flags P1IFG.0 to P1IFG.7 or P2IFG.0 to P2IFG.7 are not automatically reset. The software of the interrupt service routine should handle the detection of the source, and reset the appropriate flag when it is serviced.

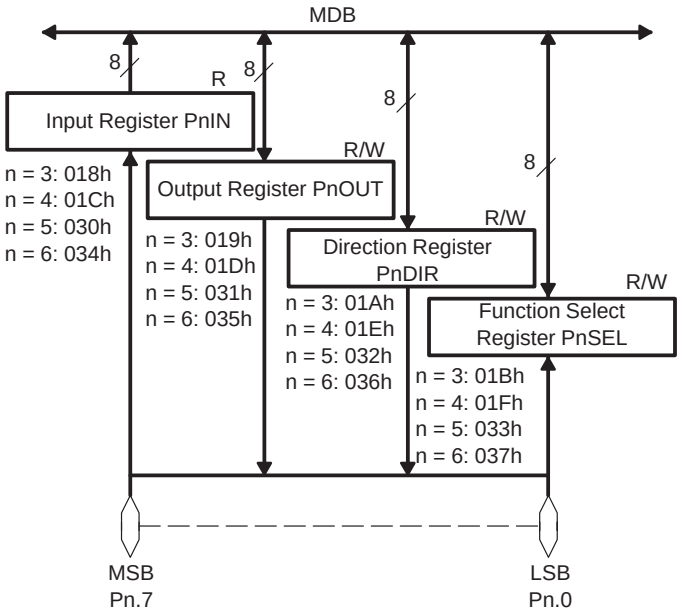
8.3 Ports P3, P4, P5, P6

General-purpose ports P3–P6 function as shown in Figure 8–3. Each pin can be selected to operate with the I/O port function, or to be used with a different peripheral module. This multiplexing of pins allows for a reduced pin count on MSP430 devices.

Four registers control each of the ports (see Section 8.3.1).

Ports P3–P6 are connected to the processor core through the 8-bit MDB and the MAB. They should be accessed with byte instructions using the absolute address mode.

Figure 8–3. Ports P3–P6 Configuration



8.3.1 Port P3–P6 Control Registers

The four control registers of each port give maximum configuration flexibility of digital I/O.

- ❏ All individual I/O bits are programmed independently
- ❏ Any combination of input is possible
- ❏ Any combination of port or module function is possible

The four registers for each port are shown in Table 8–3. They each contain eight bits and should be accessed with byte instructions.

Table 8–3. Port P3–P6 Registers

Register	Short Form	Address	Register Type	Initial State
Input	P3IN	018h	Read only	– – – – –
	P4IN	01Ch	Read only	– – – – –
	P5IN	030h	Read only	– – – – –
	P6IN	034h	Read only	– – – – –
Output	P3OUT	019h	Read/write	Unchanged
	P4OUT	01Dh	Read/write	Unchanged
	P5OUT	031h	Read/write	Unchanged
	P6OUT	035h	Read/write	Unchanged
Direction	P3DIR	01Ah	Read/write	Reset
	P4DIR	01Eh	Read/write	Reset
	P5DIR	032h	Read/write	Reset
	P6DIR	036h	Read/write	Reset
Port Select	P3SEL	01Bh	Read/write	Reset
	P4SEL	01Fh	Read/write	Reset
	P5SEL	033h	Read/write	Reset
	P6SEL	037h	Read/write	Reset

8.3.1.1 Input Registers

The input registers are read-only registers that reflect the signal at the I/O pins.

Note: Writing to Read-Only Register

Any attempt to write to these read-only registers results in an increased current consumption while the write attempt is active.

8.3.1.2 Output Registers

The output registers show the information of the output buffers. The output buffers can be modified by all instructions that write to a destination. If read, the contents of the output buffer are independent of the pin direction. A direction change does not modify the output buffer contents.

8.3.1.3 Direction Registers

The direction registers contain eight independent bits that define the direction of each I/O pin. All bits are reset by the PUC signal.

When:

Bit = 0: The port pin is switched to input direction

Bit = 1: The port pin is switched to output direction

8.3.1.4 Function Select Registers PnSEL

Ports P3–P6 pins are often multiplexed with other peripheral modules to reduce overall pin count on MSP430 devices (see the specific device data sheet to determine which other peripherals also use the device pins). Control registers PnSEL are used to select the desired pin function—I/O port or other peripheral module. Each register contains eight bits corresponding to each pin, and each pin's function is individually selectable. All bits in these registers are reset by the PUC signal. The bit definitions are:

Bit = 0: Port function is selected for the pin

Bit = 1: Other peripheral module function is selected for the pin

Note: Function Select With PnSEL Registers

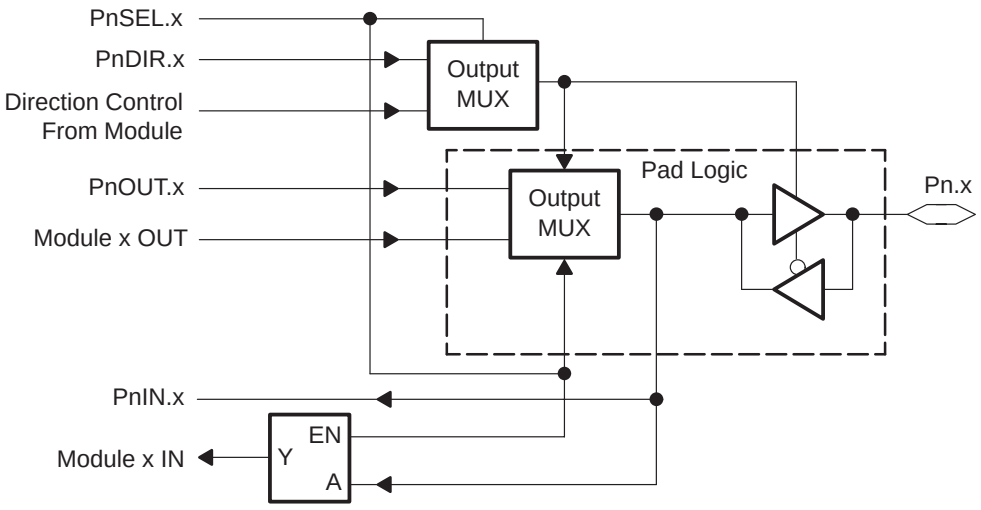
The interrupt-edge-select circuitry is disabled if control bit PnSEL.x is set. Therefore, the input signal can no longer generate an interrupt.

When a port pin is selected to be used as an input to a peripheral module other than the I/O port (PnSEL.x=1), the actual input signal to the peripheral module is a latched representation of the signal at the device pin (see Figure 8–4 schematic). The latch uses the PnSEL.x bit as its enable, so while PnSEL.x=1, the internal input signal simply follows the signal at the pin. However, if the PnSEL.x bit is reset, then the output of the latch (and therefore the input to the other peripheral module) represents the value of signal at the device pin, just prior to the bit being reset.

8.3.2 Port P3–P6 Schematic

The pin logic of each individual port signal is shown in Figure 8–4.

Figure 8–4. Schematic of Bits Pn.x



n = 3 for Port3, 4 for Port P4, 5 for Port P5, and 6 for Port P6
x = 0 to 7, according to bits 0 to 7

Chapter 9

Watchdog Timer

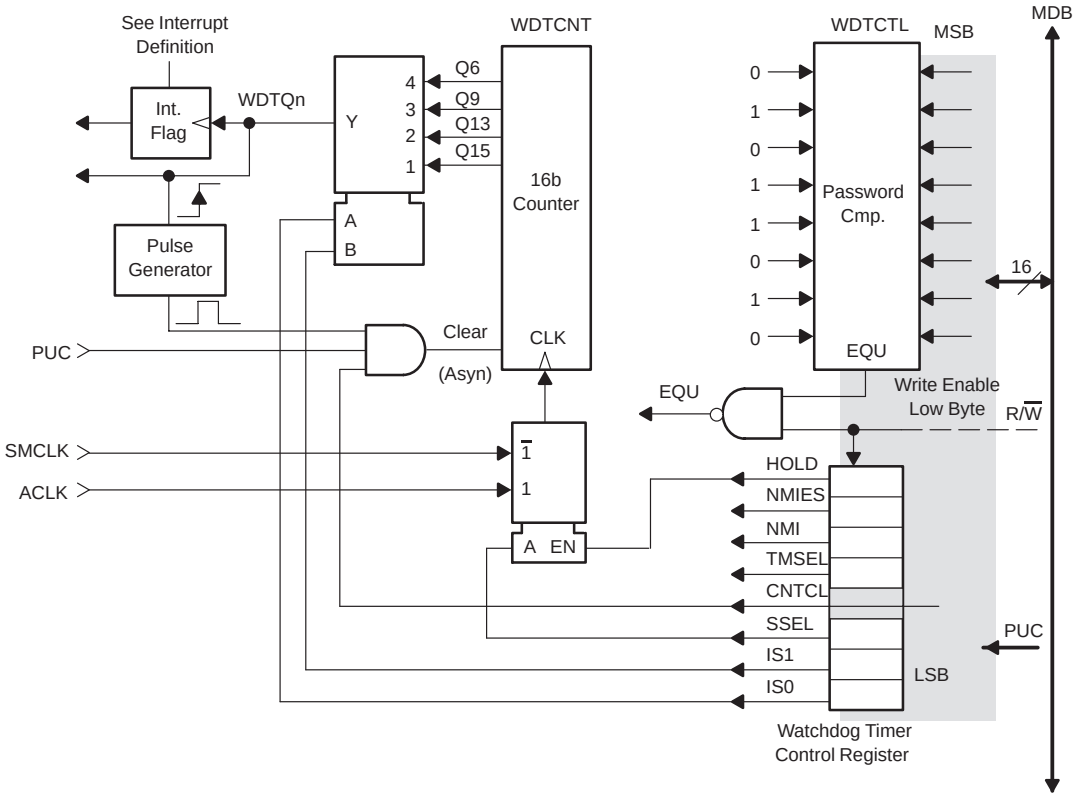
This chapter discusses the Watchdog Timer.

Topic **Page**

9.1 The Watchdog Timer

The primary function of the watchdog-timer module (WDT) is to perform a controlled-system restart after a software problem occurs. If the selected time interval expires, a system reset is generated. If the watchdog function is not needed in an application, the module can work as an interval timer, to generate an interrupt after the selected time interval. The WDT diagram is shown in Figure 9–1.

Figure 9–1. Schematic of Watchdog Timer



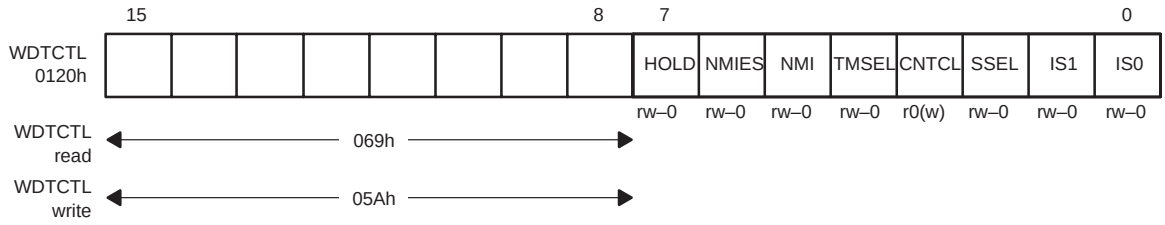
Features of the Watchdog Timer include:

- Eight software-selectable time intervals
- Two operating modes: as watchdog or interval timer
- Expiration of the time interval in watchdog mode, which generates a system reset; or in timer mode, which generates an interrupt request
- Safeguards which ensure that writing to the WDT control register is only possible using a password
- Support of ultralow-power using the hold mode

9.1.1 Watchdog Timer Register

The watchdog-timer counter (WDTCNT) is a 16-bit up-counter that is not directly accessible by software. The WDTCNT is controlled through the watchdog-timer control register (WDTCTL), shown in Figure 9–2, which is a 16-bit read/write register located at the low byte of word address 0120h. Any read or write access must be done using word instructions with no suffix or *.w* suffix. In both operating modes (watchdog or timer), it is only possible to write to WDTCTL using the correct password.

Figure 9–2. Watchdog Timer Control Register



Bits 0, 1: Bits IS0 and IS1 select one of four taps from the WDTCNT, as described in Table 9–1. Assuming $f_{crystal}$ = 32,768 Hz and f_{System} = 1 MHz, the following intervals are possible:

Table 9–1. WDTCNT Taps

SSEL	IS1	IS0	Interval [ms]	
0	1	1	0.064	$t_{SMCLK} \times 2^6$
0	1	0	0.5	$t_{SMCLK} \times 2^9$
1	1	1	1.9	$t_{ACLK} \times 2^6$
0	0	1	8	$t_{SMCLK} \times 2^{13}$
1	1	0	16.0	$t_{ACLK} \times 2^9$
0	0	0	32	$t_{SMCLK} \times 2^{15}$ <– Value after PUC (reset)
1	0	1	250	$t_{ACLK} \times 2^{13}$
1	0	0	1000	$t_{ACLK} \times 2^{15}$

Bit 2: The SSEL bit selects the clock source for WDTCNT.
SSEL = 0: WDTCNT is clocked by SMCLK .
SSEL = 1: WDTCNT is clocked by ACLK.

Bit 3: Counter clear bit. In both operating modes, writing a 1 to this bit restarts the WDTCNT at 00000h. The value read is not defined.

Bit 4: The TMSEL bit selects the operating mode: watchdog or timer.
TMSEL = 0: Watchdog mode
TMSEL = 1: Interval-timer mode

Bit 5: The NMI bit selects the function of the RST/NMI input pin. It is cleared by the PUC signal.
- NMI = 0: The $\overline{\text{RST}}$/NMI input works as reset input. As long as the $\overline{\text{RST}}$/NMI pin is held low, the internal signal is active (level sensitive).
- NMI = 1: The $\overline{\text{RST}}$/NMI input works as an edge-sensitive non-maskable interrupt input.

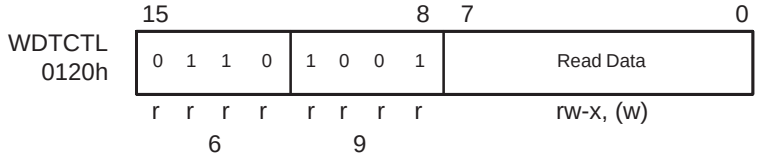
Bit 6: If the NMI function is selected, this bit selects the activating edge of the $\overline{\text{RST}}$/NMI input. It is cleared by the PUC signal.
- NMIES = 0: A rising edge triggers an NMI interrupt.
- NMIES = 1: A falling edge triggers an NMI interrupt.
- CAUTION: Changing the NMIES bit with software can generate an NMI interrupt.

Bit 7: This bit stops the operation of the watchdog counter. The clock multiplexer is disabled and the counter stops incrementing. It holds the last value until the hold bit is reset and the operation continues. It is cleared by the PUC signal.
- HOLD = 0: The WDT is fully active.
- HOLD = 1: The clock multiplexer and counter are stopped.

9.1.1.1 Accessing the WDTCTL (Watchdog Timer Control Register)

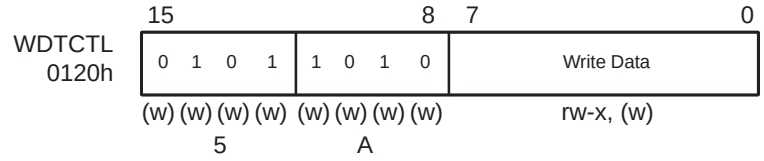
The WDTCTL register can be read or written to. As illustrated in Figure 9–3, WDTCTL can be read without the use of a password. A read access is performed by accessing word address 0120h. The low byte contains the value of WDTCTL. The value of the high byte is always read as 069h.

Figure 9–3. Reading WDTCTL



Write access to WDTCTL, illustrated in Figure 9–4, is only possible using the correct high-byte password. To change register WDTCTL, write to word address 0120h. The low byte contains the data to write to WDTCTL. The high byte is the password, which is 05Ah. A system reset (PUC) is generated if any value other than 05Ah is written to the high byte of address 0120h.

Figure 9–4. Writing to WDTCTL



9.1.2 Watchdog Timer Interrupt Control Functions

The Watchdog Timer (WDT) uses two bits in the SFRs for interrupt control.

- The WDT interrupt flag (WDTIFG) (located in IFG1.0, initial state is reset)
- The WDT interrupt enable (WDTIE) (located in IE1.0, initial state is reset)

When using the watchdog mode, the WDTIFG flag is used by the reset interrupt service routine to determine if the watchdog caused the device to reset. If the flag is set, then the Watchdog Timer initiated the reset condition (either by timing out or by a security key violation). If the flag is cleared, then the PUC was caused by a different source. See chapter 3 for more details on the PUC and POR signals.

When using the Watchdog Timer in interval-timer mode, the WDTIFG flag is set after the selected time interval and a watchdog interval-timer interrupt is requested. The interrupt vector address in interval-timer mode is different from that in watchdog mode. In interval-timer mode, the WDTIFG flag is reset automatically when the interrupt is serviced.

The WDTIE bit is used to enable or disable the interrupt from the Watchdog Timer when it is being used in interval-timer mode. Also, the GIE bit enables or disables the interrupt from the Watchdog Timer when it is being used in interval-timer mode.

9.1.3 Watchdog Timer Operation

The WDT module can be configured in two modes: watchdog and the interval-timer modes.

9.1.3.1 Watchdog Mode

When the WDT is configured to operate in watchdog mode, both a watchdog overflow and a security violation trigger the PUC signal, which automatically clears the appropriate system register bits. This results in a system configuration for the WDTCTL bits where the WDT is set into the watchdog mode and the $\overline{\text{RST}}$/NMI pin is switched to the reset configuration.

After a power-on reset or a system reset, the WDT module automatically enters the watchdog mode and all bits in the WDTCTL register and the watchdog counter (WDTCNT) are cleared. The initial conditions at register WDTCTL cause the WDT to start running at a relatively-low frequency, due to the range of the digitally-controlled oscillator (DCO) automatically being set in these situations. Since the WDTCNT is reset, the user software has ample time to set up or halt the WDT and to adjust the system frequency. Users must refer to the specific data sheets and the clock-system chapter of this manual to determine the details of the clocking circuit on the MSP430 device chosen.

When the module is used in watchdog mode, the software should periodically reset the WDTCNT by writing a 1 to bit CNTCL of WDTCTL to prevent expiration of the selected time interval. If a software problem occurs and the time interval expires because the counter is no longer being reset, a system reset is generated and a system PUC signal is activated. The system restarts at the same program address that follows a power up. The cause of reset can be determined by testing bit 0 of interrupt flag register 1 in the SFRs. The appropriate time interval is selected by setting bits SSEL, IS0, and IS1 accordingly.

9.1.3.2 Timer Mode

Setting WDTCTL register bit TMSEL to 1 selects the timer mode. This mode provides periodic interrupts at the selected time interval. A time interval can also be initiated by writing a 1 to bit CNTCL in the WDTCTL register.

When the WDT is configured to operate in timer mode, the WDTIFG flag is set after the selected time interval, and it requests a standard interrupt service. The WDT interrupt flag is a single-source interrupt flag and is automatically reset when it is serviced. The enable bit remains unchanged. In interval-timer mode, the WDT interrupt-enable bit and the GIE bit must be set to allow the WDT to request an interrupt. The interrupt vector address in timer mode is different from that in watchdog mode.

Note: Watchdog Timer, Changing the Time Interval

Changing the time interval without clearing the WDTCNT may result in an unexpected and immediate system reset or interrupt. The time interval must be changed together with a counter-clear command using a single instruction (for example, MOV #05A0Ah,&WDTCTL).

Changing the clock source during normal operation may result in an incorrect interval. The timer should be halted before changing the clock source.

9.1.3.3 Operation in Low-Power Modes

The MSP430 devices have several low-power modes. Different clock signals are available in different low-power modes. The requirements of the user's application and the type of clocking circuit on the MSP430 device determine how the Watchdog Timer and clocking signals should be configured. Review the device data sheet and clock-system chapter to determine the clocking circuit, clock signals, and low-power modes available. For example, the WDT should not be configured in watchdog mode with SMCLK as its clock source if the user wants to use low-power mode 3 because SMCLK is not active in LPM3, therefore the WDT would not function properly.

The WDT hold condition can also be used to support low power operation. The hold condition can be used in conjunction with low-power modes when needed.

9.1.3.4 Software Example

The following example illustrates the watchdog-reset operation.

```
; After RESET or power–up, the WDTCTL register and WDTCNT
; are cleared and the initial operating conditions are
; watchdog mode with a time interval of 32 ms.
;
;Constant definitions:
;
  WDTCTL  .EQU  0120h  ; Address of Watchdog Timer
  WDTPW   .EQU  05A00h; Password
  T250MS  .EQU  5      ; SSEL, IS0, IS1 set to 250 ms
  T05MS   .EQU  2      ; SSEL, IS0, IS1 set to 0.5 ms
  CNTCL   .EQU  8      ; Bit position to reset WDTCNT
  TMSEL   .EQU  010h   ; Bit position to select timer mode
;
; As long as watchdog mode is selected, watchdog reset has
; to be done periodically through an instruction e.g.:
;
   ........
   ........
   MOV    #WDTPW+CNTCL,&WDTCTL
;
; To change to timer mode and a time interval of 250 ms,
; the following instruction sequence can be used:
;
   MOV    #WDTPW+CNTCL+TMSEL+T250MS,&WDTCTL
             ; Clear WDTCNT and
             ; select 250 ms and timer
             ; mode
   ........
   ........
; Note: The time interval and clear of WDTCNT should be
;         modified within one instruction to avoid
;         unexpected reset or interrupt
;
; Switching back to watchdog mode and a time interval of
; 0.5 ms is performed by:
;
   ........
   ........
   MOV    #WDTPW+CNTCL,&WDTCTL  ; Reset WDT counter
;
   MOV    #WDTPW+T05MS,&WDTCTL  ; Select watchdog mode
                                ; and 0.5 ms
   ........
```

Chapter 10

Timer_A

This section describes the basic functions of the MSP430 general-purpose 16-bit Timer_A. All capture/compare blocks (CCR) are identical. MSP430x11x, MSP430x11x1, and MSP430x13x have three CCRs (Timer_A3) and MSP430x14x has five CCRs (Timer_A5) implemented.

Note:

Throughout this chapter, the word *count* is used in the text. As used in these instances, it refers to the literal act of counting. It means that the counter must be in the process of counting for the action to take place. If a particular value is directly written to the counter, then the associated action will not take place. For example, the CCR0 interrupt flag is set when the timer *counts up to* the value in CCR0. The counter *must count* from CCR0–1 to CCR0. If the CCR0 value were simply written directly to the timer with software, the interrupt flag would *not* be set, even though the values in the timer and the CCR0 registers are the same.

Topic **Page**

10.1 Introduction

Timer_A is an extremely versatile timer made up of :

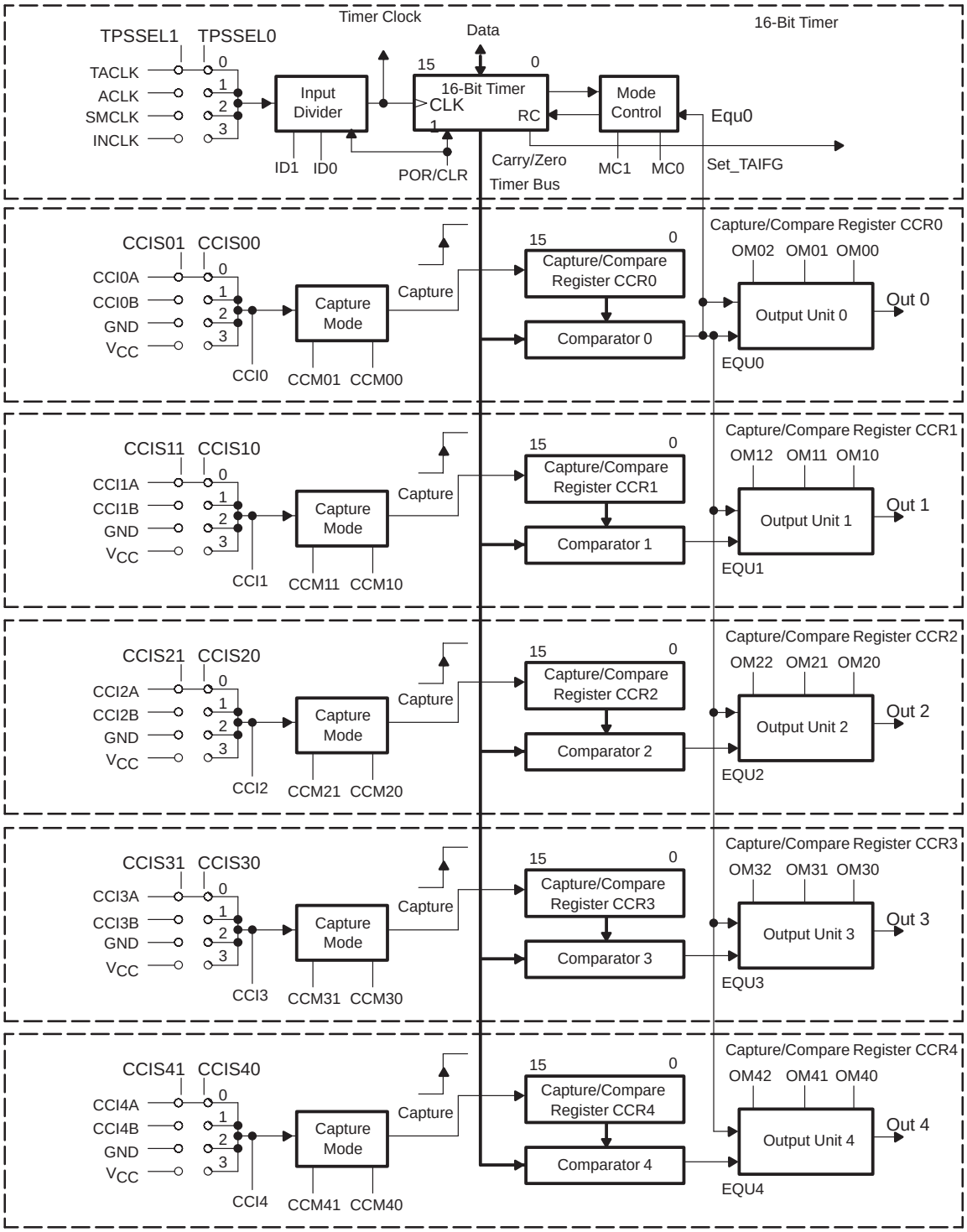
- 16-bit counter with 4 operating modes
- Selectable and configurable clock source
- Three or five independently configurable capture/compare registers with configurable inputs
- Three or five individually configurable output modules with 8 output modes

Timer_A can support multiple, simultaneous, timings; multiple capture/compares; multiple output waveforms such as PWM signals; and any combination of these. Also, each capture/compare register has hardware support for implementing serial communications such as UART protocol (see section 10.7).

Additionally, Timer_A has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers on captures or compares. Each capture/compare block is individually configurable and can produce interrupts on compares or on rising, falling, or both edges of an external capture signal.

The block diagram of Timer_A is shown in Figure 10–1.

Figure 10–1. Timer_A Block Diagram



10.2 Timer_A Operation

The 16-bit timer has 4 modes of operation selectable with the MC0 and MC1 bits in the TACTL register. The timer increments or decrements (depending on mode of operation) with each rising edge of the clock signal. The timer can be read or written to with software. Additionally, the timer can generate an interrupt with its ripple-carry output when it overflows.

10.2.1 Timer Mode Control

The timer has four modes of operation as shown in Figure 10–2 and described in Table 10–1: stop, up, continuous, and up/down. The operating mode is software selectable with the MC0 and MC1 bits in the TACTL register.

Figure 10–2. Mode Control

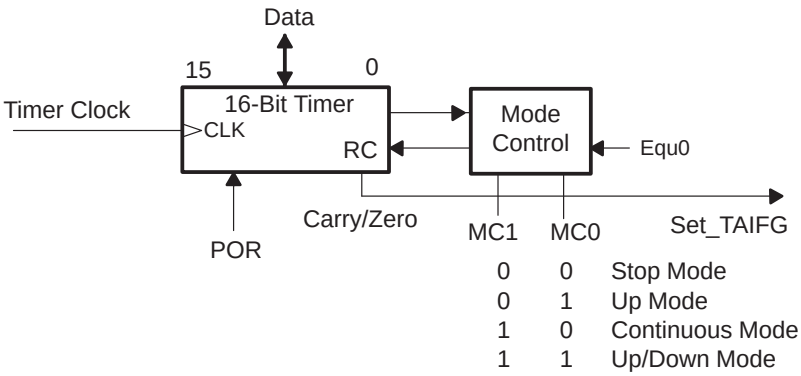


Table 10–1. Timer Modes

Mode Control			
MC1	**MC0**	**Mode**	**Description**
0	0	Stop	The timer is halted.
0	1	Up	The timer counts upward until value is equal to value of compare register CCR0.
1	0	Continuous	The timer counts upward continuously.
1	1	Up/Down	The timer counts up until the timer value is equal to compare register 0 and then it counts down to zero.

10.2.2 Clock Source Select and Divider

The timer clock can be sourced from internal clocks (i.e. ACLK or SMCLK) or from an external source (TACLK) as shown in Figure 10–3. The clock source is selectable with the SSEL0 and SSEL1 bits in the TACTL register. It is important to note that when changing the clock source for the timer, errant timings can occur. For this reason stopping the timer before changing the clock source is recommended.

The selected clock source may be passed directly to the timer or divided by 2,4, or 8, as shown in Figure 10–4. The ID0 and ID1 bits in the TACTL register select the clock division. Note that the input divider is reset by a POR signal (see chapter 3, *System, Resets, Interrupts, and Operating Modes* for more information on the POR signal) or by setting the CLR bit in the TACTL register. Otherwise, the input divider remains unchanged when the timer is modified. The state of the input divider is invisible to software.

Figure 10–3. Schematic of 16-Bit Timer

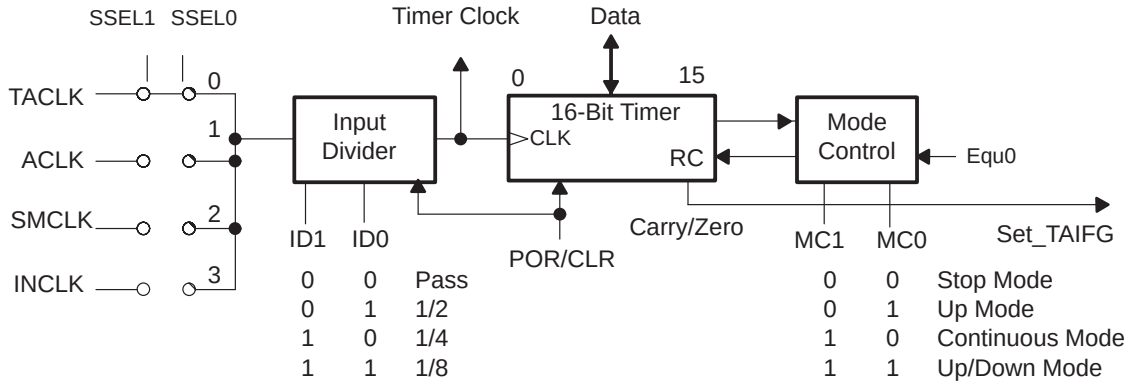
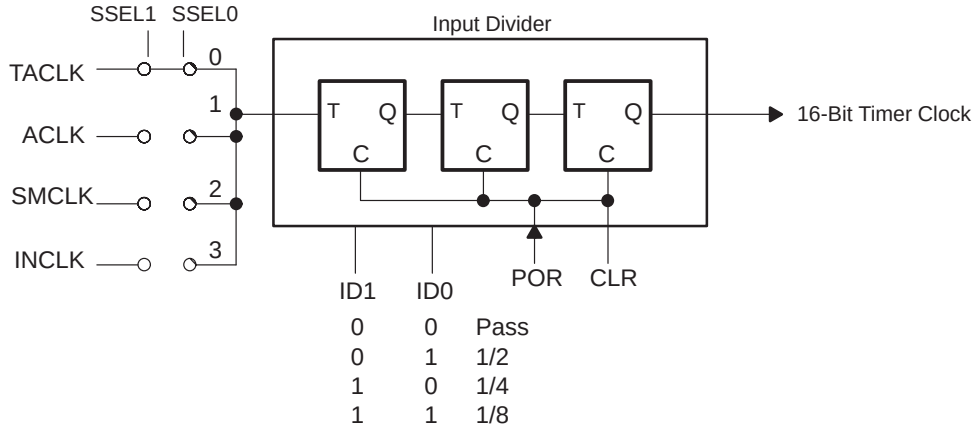


Figure 10–4. Schematic of Clock Source Select and Input Divider



10.2.3 Starting the Timer

The timer may be started or restarted in a variety of ways:

- Release Halt Mode: The timer *counts* in the selected direction when a timer mode other than stop mode is selected with the MCx bits.
- Halted by CCR0 = 0, restarted by CCR0 > 0 when the mode is either up or up/down: When the timer mode is selected to be either up or up/down, the timer may be stopped by writing 0 to capture/compare register 0 (CCR0). The timer may then be restarted by writing a nonzero value to CCR0. In this scenario, the timer starts incrementing in the up direction from zero.
- Setting the CLR bit in TACTL register: Setting the CLR bit in the TACTL register clears the timer value and input clock divider value. The timer increments upward from zero with the next clock cycle as long as stop-mode is not selected with the MCx bits.
- TAR is loaded with 0: When the counter (TAR register) is loaded with zero with a software instruction the timer increments upward from zero with the next clock cycle as long as stop-mode is not selected with the MCx bits.

10.3 Timer Modes

10.3.1 Timer—Stop Mode

Stopping and starting the timer is done simply by changing the mode control bits (MCx). The value of the timer is not affected.

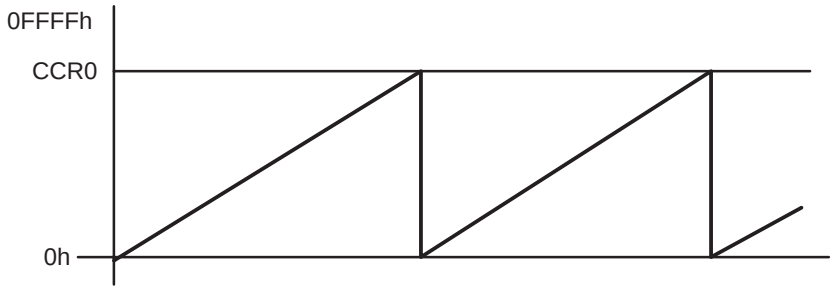
When the timer is stopped from up/down mode and then restarted in up/down mode, the timer counts in the same direction as it was counting before it was stopped. For example, if the timer is in up/down mode and counting in the down direction when the MCx bits are reset, when they are set back to the up/down direction, the timer starts counting in the down direction from its previous value. If this is not desired in an application, the CLR bit in the TACTL register can be used to clear this direction memory feature.

10.3.2 Timer—Up Mode

The up mode is used if the timer period must be different from the 65,536 (16-bit) clock cycles of the continuous mode period. The capture/compare register CCR0 data define the timer period.

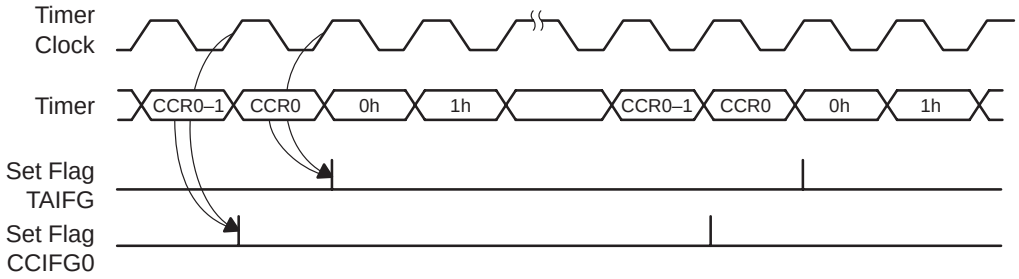
The counter *counts* up to the content of compare register CCR0, as shown in Figure 10–5. When the timer value and the value of compare register CCR0 are equal (or if the timer value is greater than the CCR0 value), the timer restarts counting from zero.

Figure 10–5. Timer Up Mode



Flag CCIFG0 is set when the timer equals the CCR0 value. The TAIFG flag is set when the timer *counts* from CCR0 to zero. All interrupt flags are set independently of the corresponding interrupt enable bit, but an interrupt is requested only if the corresponding interrupt enable bit and the GIE bit are set. Figure 10–6 shows the flag set cycle.

Figure 10–6. Up Mode Flag Setting



10.3.2.1 Timer in Up Mode—Changing the Period Register CCR0 Value

Changing the timer period register CCR0 while the timer is running can be a little tricky. When the new period is greater than or equal to the old period, the timer simply *counts* up to the new period and no special attention is required (see Figure 10–7). However, when the new period is less than the old period, the phase of the timer clock during the CCR0 update affects how the timer reacts to the new period.

If the new, smaller period is written to CCR0 during a high phase of the timer clock, then the timer rolls to zero (or begins counting down when in the up/down mode) on the next rising edge of the timer clock. However, if the new, smaller period is written during a low phase of the timer clock, then the timer continues to increment with the old period for one more clock cycle before adopting the new period and rolling to zero (or beginning counting down). This is shown in Figure 10–8.

Figure 10–7. New Period > Old Period

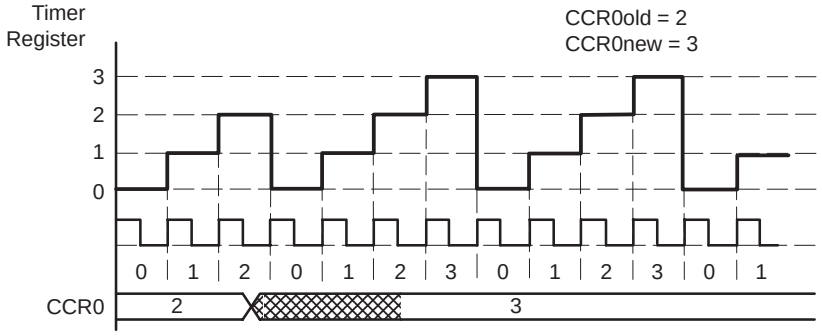
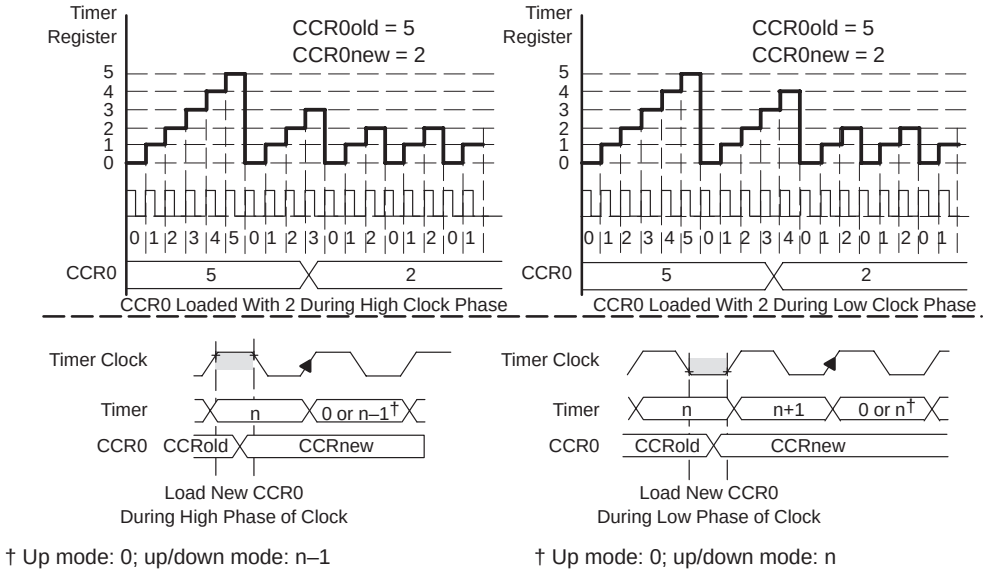


Figure 10–8. New Period < Old Period



† Up mode: 0; up/down mode: n–1

† Up mode: 0; up/down mode: n

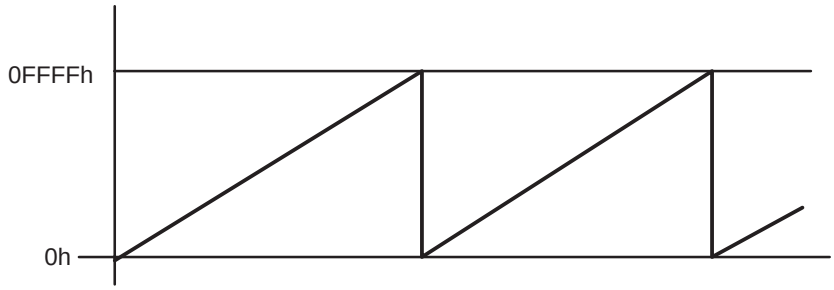
10.3.3 Timer—Continuous Mode

The continuous mode is used if the timer period of 65,536 clock cycles is used for the application. A typical application of the continuous mode is to generate multiple, independent timings. In continuous mode, the capture/compare register CCR0 works in the same way as the other compare registers.

The capture/compare registers and different output modes of each output unit are useful to capture timer data based on external events or to generate various different types of output signals. Examples of the different output modes used with timer-continuous mode are shown in Figure 10–25.

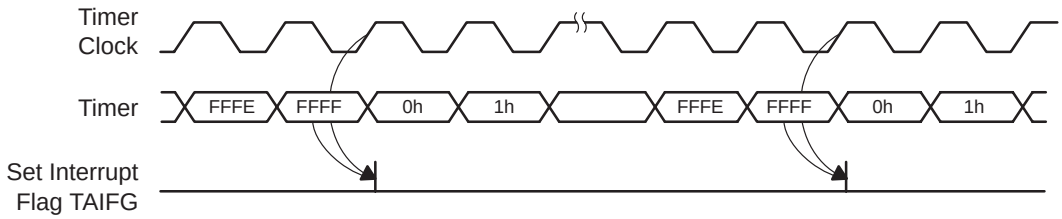
In continuous mode, the timer starts counting from its present value. The counter counts up to 0FFFFh and restarts by counting from zero as shown in Figure 10–9.

Figure 10–9. Timer—Continuous Mode



The TAIFG flag is set when the timer *counts* from 0FFFFh to zero. The interrupt flag is set independently of the corresponding interrupt enable bit, as shown in Figure 10–10. An interrupt is requested if the corresponding interrupt enable bit and the GIE bit are set.

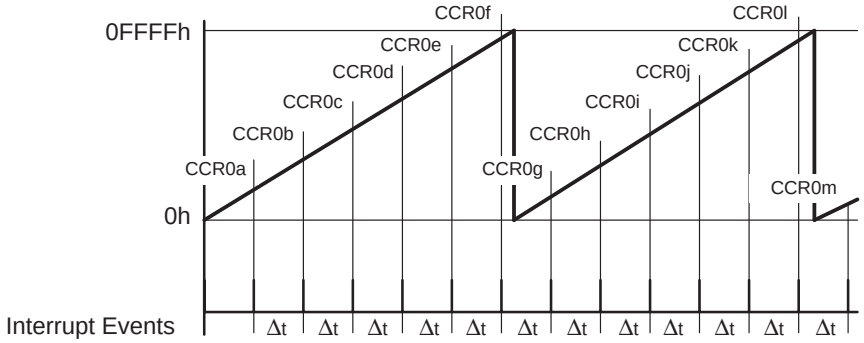
Figure 10–10. Continuous Mode Flag Setting



10.3.3.1 Timer—Use of the Continuous Mode

The continuous mode can be used to generate time intervals for the application software. Each time an interval is completed, an interrupt can be generated. In the interrupt service routine of this event, the time until the next event is added to capture/compare register CCRx as shown in Figure 10–11. Up to five independent time events can be generated using all five capture/compare blocks.

Figure 10–11. Output Unit in Continuous Mode for Time Intervals

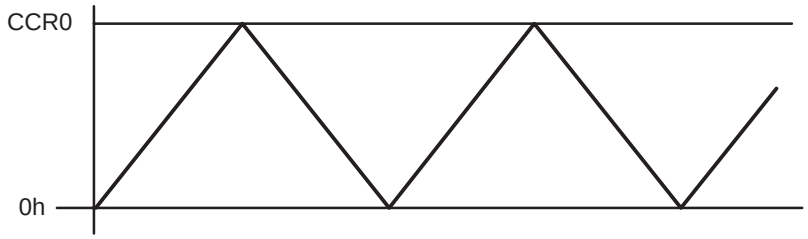


Time intervals can be produced with other modes as well, where CCR0 is used as the period register. Their handling is more complex since the sum of the old CCRx data and the new period can be higher than the CCR0 value. When the sum CCRxold plus Δt is greater than the CCR0 data, the CCR0 value must be subtracted to obtain the correct time interval. The period is twice the value in the CCR0 register.

10.3.4 Timer—Up/Down Mode

The up/down mode is used if the timer period must be different from the 65,536 clock cycles, and if symmetrical pulse waveform generation is needed. In up/down mode, the timer counts up to the content of compare register CCR0, then back down to zero, as shown in Figure 10–12. The period is twice the value in the CCR0 register.

Figure 10–12. Timer Up/Down Mode



The up/down mode also supports applications that require dead times between output signals. For example, to avoid overload conditions, two outputs driving an H-bridge must never be in a high state simultaneously. In the following example (see Figure 10–13), the t_{dead} is:

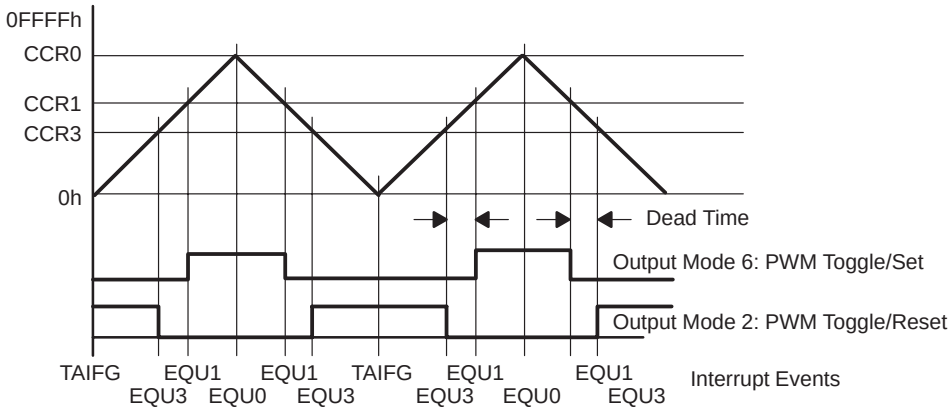
$$t_{dead} = t_{timer} \times (CCR1 - CCR3) =$$

With: t_{dead} Time during which both outputs need to be inactive

t_{timer} Cycle time of the timer clock

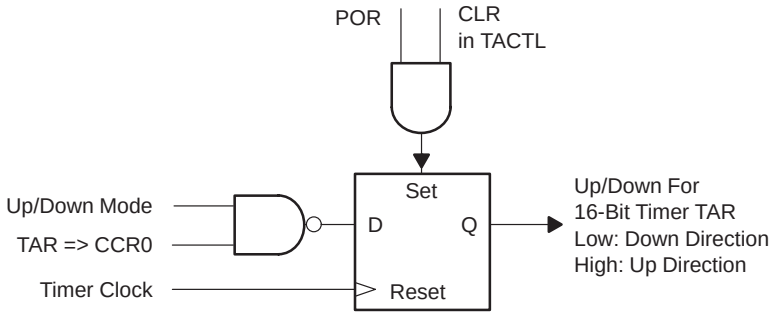
CCRx Content of capture/compare register x

Figure 10–13. Output Unit in Up/Down Mode (II)



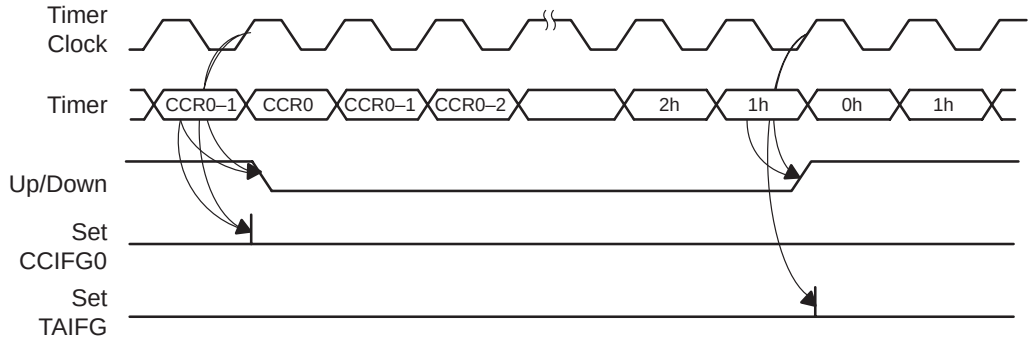
The count direction is always latched with a flip-flop (Figure 10–14). This is useful because it allows the user to stop the timer and then restart it in the same direction it was counting before it was stopped. For example, if the timer was counting down when the MCx bits were reset, then it will continue counting in the down direction if it is restarted in up/down mode. If this is not desired, the CLR bit in the TACTL register must be used to clear the direction. Note that the CLR bit affects other setup conditions of the timer. Refer to Section 10.6 for a discussion of the Timer_A registers.

Figure 10–14. Timer Up/Down Direction Control



In up/down mode, the interrupt flags (CCIFG0 and TAIFG) are set at equal time intervals (Figure 10–15). Each flag is set only once during the period, but they are separated by 1/2 the timer period. CCIFG0 is set when the timer *counts* from CCR0–1 to CCR0, and TAIFG is set when the timer completes *counting* down from 0001h to 0000h. Each flag is capable of producing a CPU interrupt when enabled.

Figure 10–15. Up/Down Mode Flag Setting

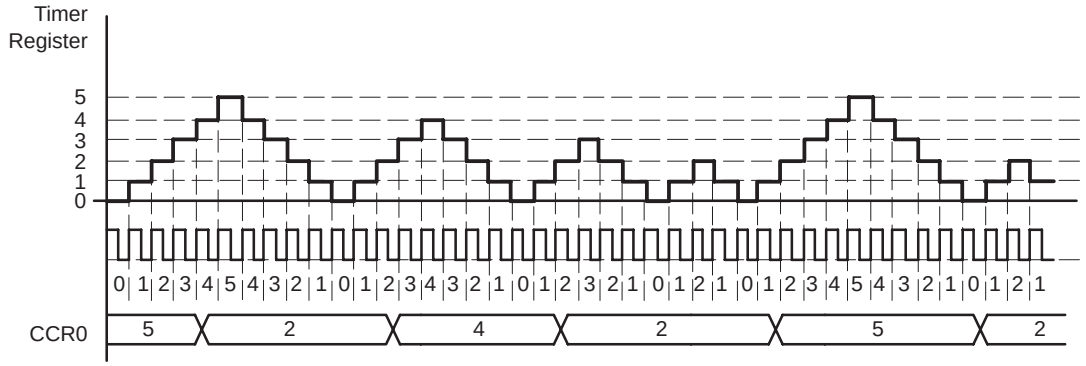


10.3.4.1 Timer In Up/Down Mode—Changing the Value of Period Register CCR0

Changing the period value while the timer is running in up/down mode is even trickier than in up mode. Like in up mode, the phase of the timer clock when CCR0 is changed affects the timer's behavior. Additionally, in up/down mode, the direction of the timer also affects the behavior.

If the timer is counting in the up direction when the new period is written to CCR0, the conditions in the up/down mode are identical to those in the up mode. See Section 10.3.2.1 for details. However, if the timer is counting in the down direction when CCR0 is updated, it continues its descent until it reaches zero. The new period takes effect only after the counter finishes counting down to zero. See Figure 10–16.

Figure 10–16. Altering CCR0—Timer in Up/Down Mode



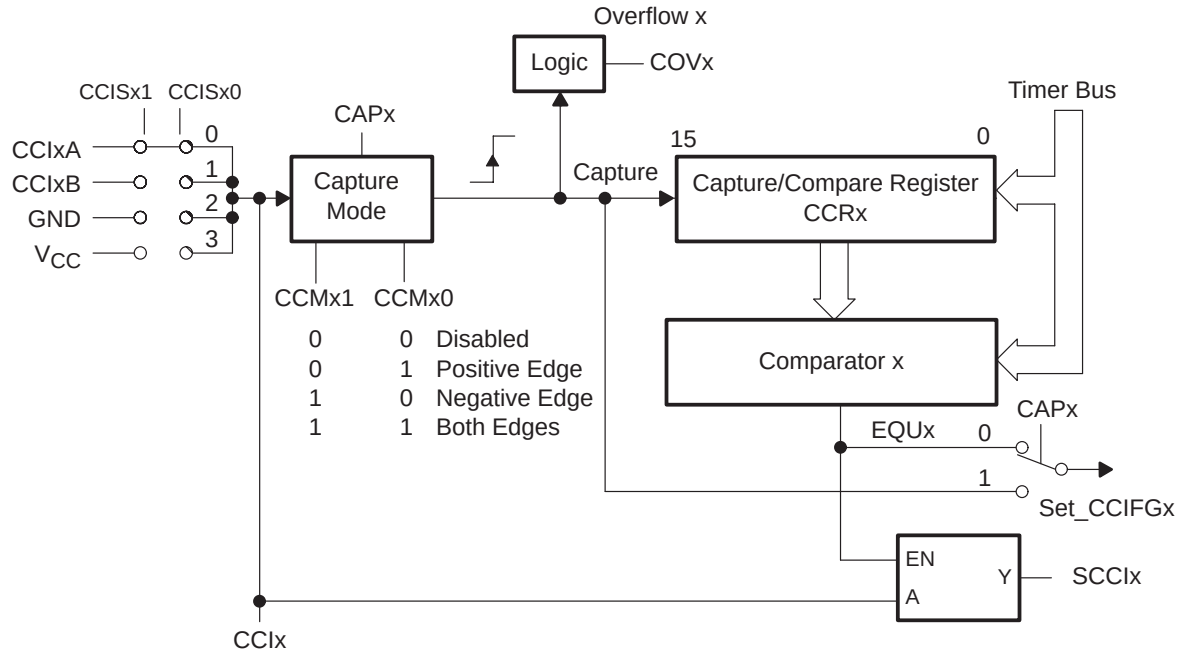
10.4 Capture/Compare Blocks

Three or five (depending on device) identical capture/compare blocks (shown in Figure 10–17) provide flexible control for real-time processing. Any one of the blocks may be used to capture the timer data at an applied event, or to generate time intervals. Each time a capture occurs or a time interval is completed, interrupts can be generated from the applicable capture/compare register. The mode bit CAPx, in control word CCTLx, selects the compare or capture operation and the capture mode bits CCMx1 and CCMx0 in control word CCTLx define the conditions under which the capture function is performed.

Both the interrupt enable bit CCIEx and the interrupt flag CCIFGx are used for capture and compare modes. CCIEx enables the corresponding interrupt. CCIFGx is set on a capture or compare event.

The capture inputs CCIxA and CCIxB are connected to external pins or internal signals. MSP430x1xx devices may have different signals connected to CCIxA and CCIxB. The data sheet should always be consulted to determine the Timer_A connections for a particular device.

Figure 10–17. Capture/Compare Blocks



10.4.1 Capture/Compare Block—Capture Mode

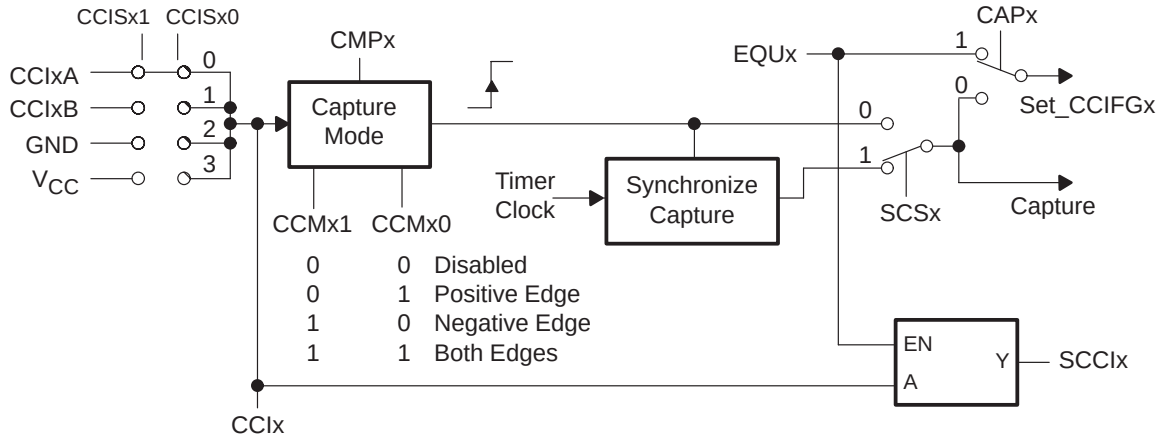
The capture mode is selected if the mode bit CAPx, located in control word CCTLx, is set. The capture mode is used to fix time events. It can be used for speed computations or time measurements. The timer value is copied into the capture register (CCRx) with the selected edge (positive, negative, or both) of the input signal. Captures may also be initiated by software as described in section 10.4.1.1.

If a capture is performed:

- The interrupt flag CCIFGx, located in control word CCTLx, is set.
- An interrupt is requested if both interrupt enable bits CCIEx and GIE are set.

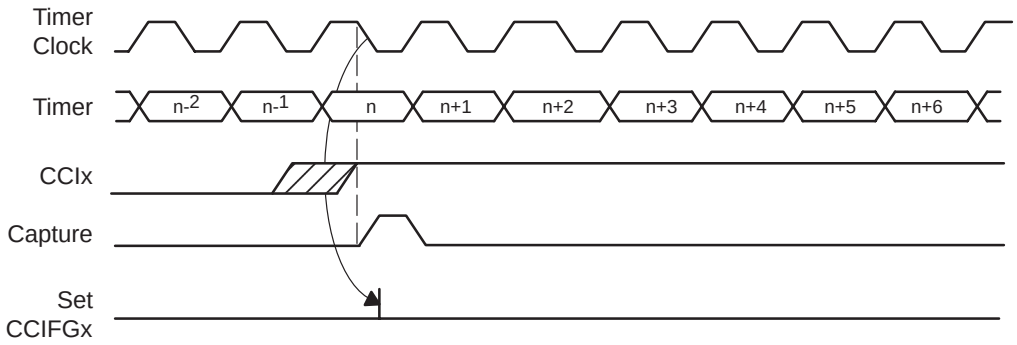
The input signal to the capture/compare block is selected using control bits CCISx1 and CCISx0, as shown in Figure 10–18. The input signal can be read at any time by the software by reading bit CCIx. The input signal may also be latched with compare signal EQUx (see SCCIx bit below) when in compare mode. This feature was designed specifically to support implementing serial communications with Timer_A. See section 10.7 for more details on using Timer_A as a UART.

Figure 10–18. Capture Logic Input Signal



The capture signal can also be synchronized with the timer clock to avoid race conditions between the timer data and the capture signal. This is illustrated in Figure 10–19. The bit SCSx in capture/compare control register CCTLx selects the capture signal synchronization.

Figure 10–19. Capture Signal

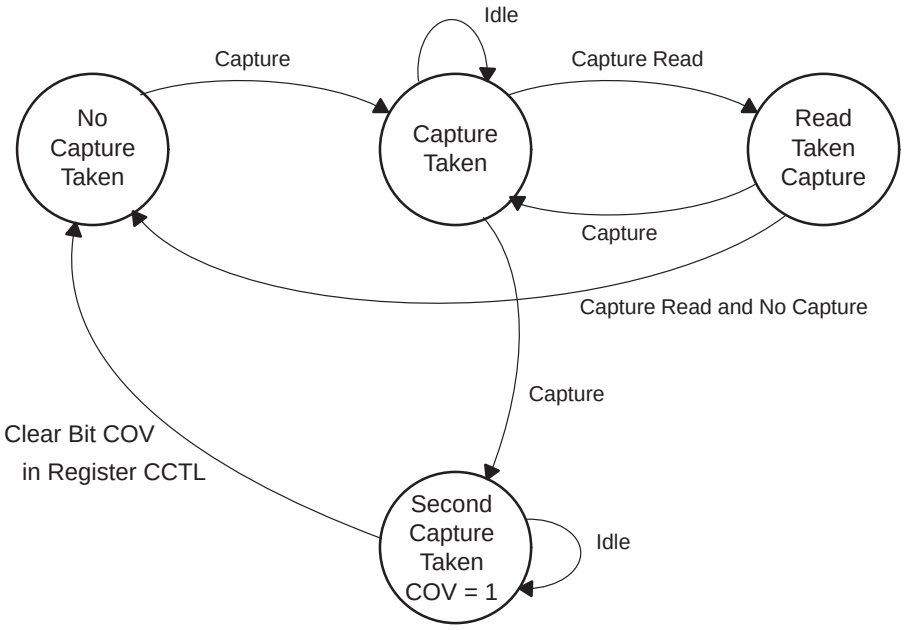


Applications with slow timer clocks can use the nonsynchronized capture signal. In this scenario the software can validate the data and correct it if necessary as shown in the following example:

```
; Software example for the handling of asynchronous
; capture signals
;
; The data of the capture/compare register CCRx are taken
; by the software in the according interrupt routine
; – they are taken only after a CCIFG was set.
; The timer clock is much slower than the system clock
; MCLK.
;
CCRx_Int_hand...                ; Start of interrupt
                                ; handler
            ...
            ...
            CMP   &CCRx,&TAR    ; Test if the data
                                ; CCRX = TAR
            JEQ   Data_Valid
            MOV   &TAR,&CCRx    ; The data in CCRx is
                                ; wrong, use the timer data
Data_Valid  ...   ...           ; The data in CCRx are valid
            ...
            ...
            RETI
;
```

Overflow logic is provided with each capture/compare register to flag the user if a second capture is performed before data from the first capture was read successfully. Bit COVx in register CCTLx is set when this occurs as shown in Figure 10–20.

Figure 10–20. Capture Cycle



Overflow bit COVx is reset by the software as described in the following example:

```
; Software example for the handling of captured data
; looking for overflow condition
;
; The data of the capture/compare register CCRx are taken
; by the software and immediately with the next
; instruction the overflow bit is tested and a decision is
; made to proceed regularly or with an error handler
;
CCRx_Int_hand   ...       ; Start of handler Interrupt
          ...
          ...
                MOV   &CCRx,RAM_Buffer
                BIT   #COV,&CCTLx
                JNZ   Overflow_Hand
          ...
          ...
          ...
                RETI
Overflow_Hand   BIC   #COV,&CCTLx  ; reset capture
                                   ; overflow flag
                                   ; get back to lost
                                   ; synchronization

          ...
          ...
;               RETI
```

Note: Capture With Timer Halted

The capture should be disabled when the timer is halted. The sequence to follow is: stop the capture, then stop the timer. When the capture function is restarted, the sequence should be: start the capture, then start the timer.

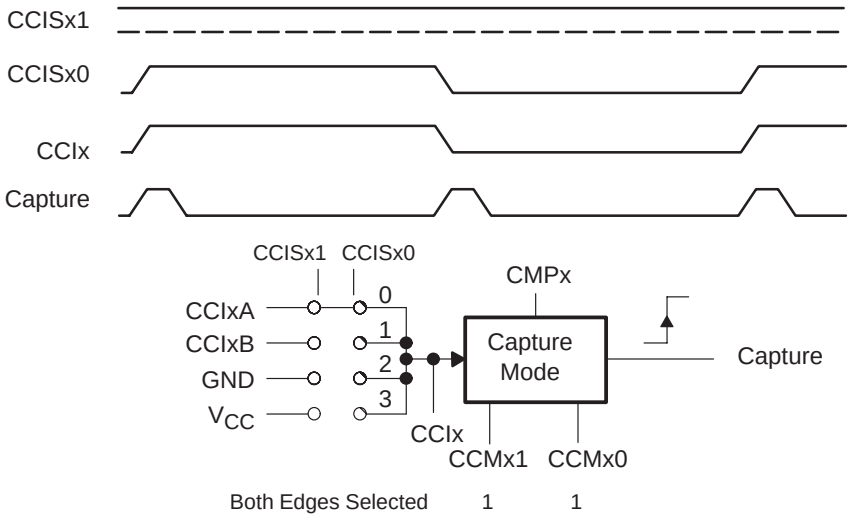
10.4.1.1 Capture/Compare Block, Capture Mode—Capture Initiated by Software

In addition to internal and external signals, captures can be initiated by software. This is useful for various purposes, such as:

- To measure time used by software routines
- To measure time between hardware events
- To measure the system frequency

Two bits, CCISx1 and CCISx0, and the capture mode selected by bits CCMx1 and CCMx0 are used by the software to initiate the capture. The simplest realization is when the capture mode is selected to capture on both edges of CCIx and bit CCISx1 is set. Software then toggles bit CCISx0 to switch the capture signal between V_{CC} and GND, initiating a capture each time the input is toggled, as shown in Figure 10–21.

Figure 10–21. Software Capture Example



The following is a software example of a capture performed by software:

```
; The data of capture/compare register CCRx are taken
; by the software. It is assumed that CCMx1, CCMx0, and
; CCISx1 bits are set. Bit CCIS0 selects the CCIx
; signal to be high or low.
;
          ...
          ...
          XOR   #CCISx0, &CCTLx
          ...
          ...
          ...
```

10.4.2 Capture/Compare Block—Compare Mode

The compare mode is selected if the CAPx bit, located in control word CCTLx, is reset. In compare mode all the capture hardware circuitry is inactive and the capture-mode overflow logic is inactive.

The compare mode is most often used to generate interrupts at specific time intervals or used in conjunction with the output unit to generate output signals such as PWM signals. If the timer becomes equal to the value in compare register x, then:

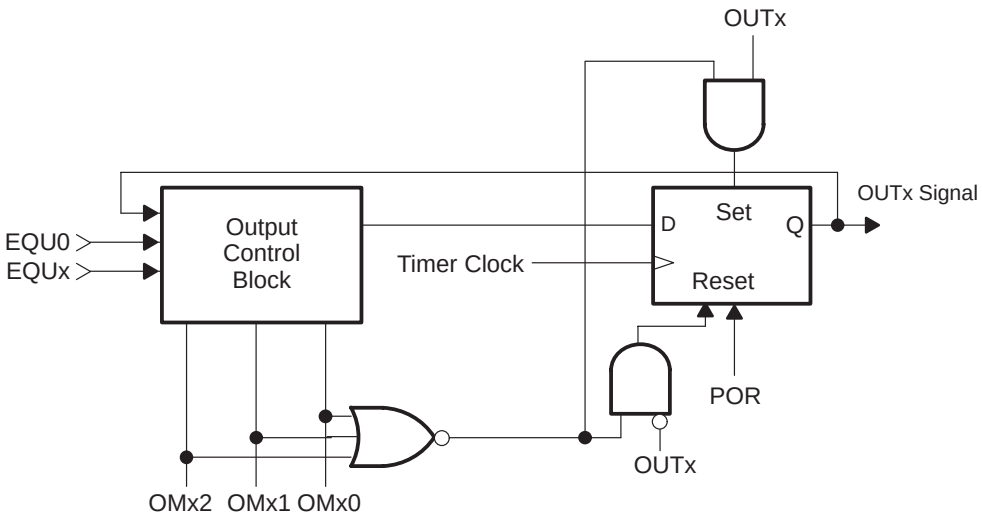
- Interrupt flag CCIFGx, located in control word CCTLx, is set.
- An interrupt is requested if interrupt enable bits CCIEx and GIE are set.
- Signal EQUx is output to the output unit. This signal affects the output OUTx, depending on the selected output mode.

The EQU0 signal is true when the timer value is greater or equal to the CCR0 value. The EQU1 to EQU4 signals are true when the timer value is equal to the corresponding CCR1 to CCR4 values.

10.5 The Output Unit

Each capture/compare block contains an output unit shown in Figure 10–22. The output unit is used to generate output signals such as PWM signals. Each output unit has 8 operating modes that can generate a variety of signals based on the EQU0 and EQUx signals. The output mode is selected with the OMx bits located in the CCTLx register.

Figure 10–22. Output Unit



OMx2	OMx1	OMx0	
0	0	0	Output mode: OUTx signal reflects the value of the OUTx bit
0	0	1	Set mode: OUT x signal reflects the value of signal EQUx
0	1	0	PWM toggle/reset: EQUx toggles OUTx. EQU0 resets OUTx.
0	1	1	PWM set/reset: EQUx sets OUTx. EQU0 resets OUTx
1	0	0	Toggle: EQUx toggles OUTx signal.
1	0	1	Reset: EQUx resets OUTx.
1	1	0	PWM toggle/set: EQUx toggles OUTx. EQU0 sets OUTx.
1	1	1	PWM reset/set: EQUx resets OUTx. EQUx sets OUTx.

Note: OUTx signal updates with rising edge of timer clock for all modes except mode 0.
Modes 2, 3, 6, 7 not useful for output unit 0.

10.5.1 Output Unit—Output Modes

The output modes are defined by the OMx bits and are discussed below. The OUTx signal is changed with the rising edge of the timer clock for all modes except mode 0. Output modes 2, 3, 6, and 7 are not useful for output unit 0.

Output mode 0: Output mode:
The output signal OUTx is defined by the OUTx bit in control register CCTLx. The OUTx signal updates immediately upon completion of writing the bit information.

Output mode 1: Set mode:
The output is set when the timer value becomes equal to capture/compare data CCRx. It remains set until a reset of the timer, or until another output mode is selected that controls the output.

Output mode 2: PWM toggle/reset mode:
The output is toggled when the timer value becomes equal to capture/compare data CCRx. It is reset when the timer value becomes equal to CCR0.

Output mode 3: PWM set/reset mode:
The output is set when the timer value becomes equal to capture/compare data CCRx. It is reset when the timer value becomes equal to CCR0.

Output mode 4: Toggle mode:
The output is toggled when the timer value becomes equal to capture/compare data CCRx. The output period is double the timer period.

Output mode 5: Reset mode:
The output is reset when the timer value becomes equal to capture/compare data CCRx. It remains reset until another output mode is selected that controls the output.

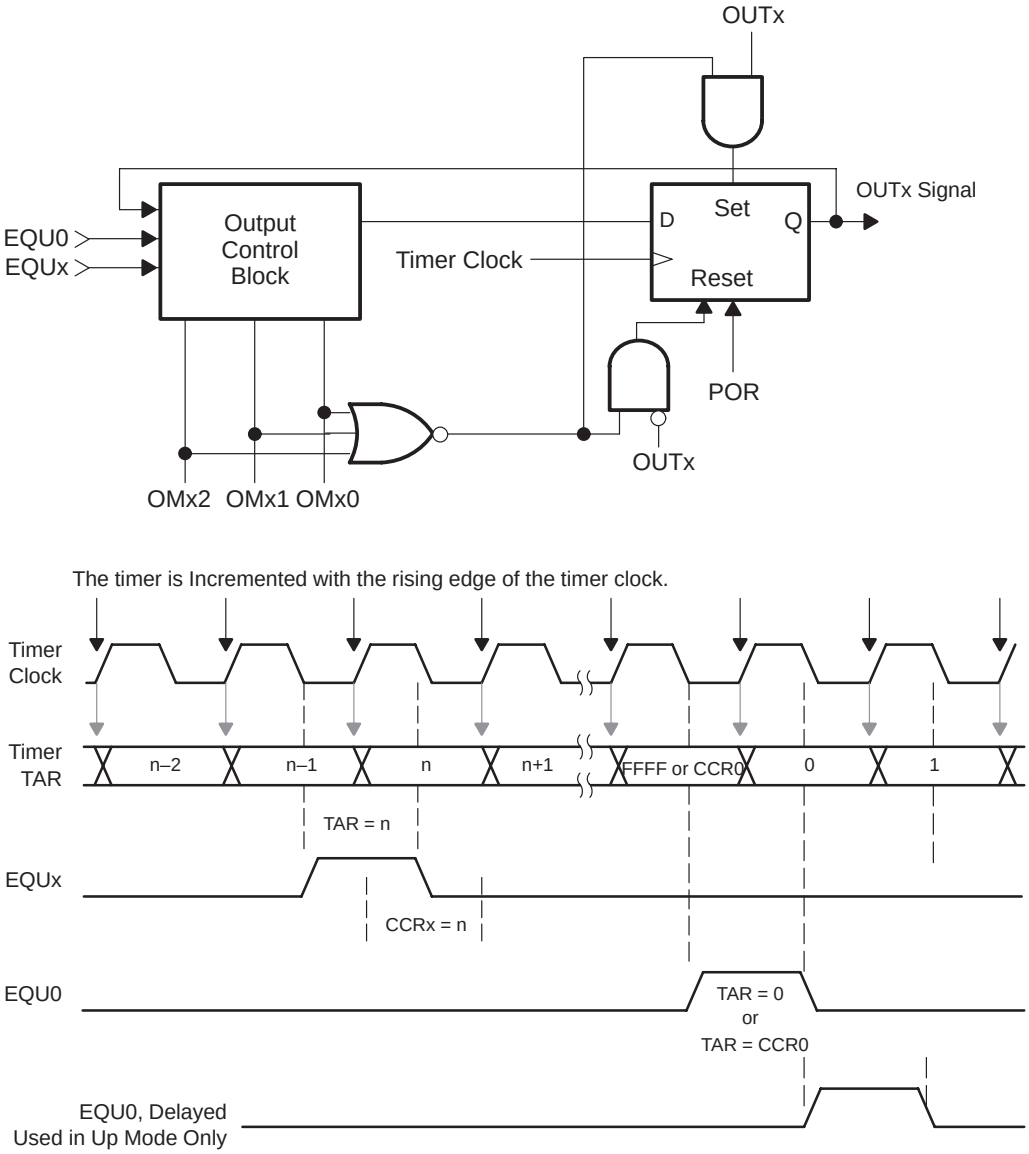
Output mode 6: PWM toggle/set mode:
The output is toggled when the timer value becomes equal to capture/compare data CCRx. It is set when the timer value becomes equal to CCR0.

Output mode 7: PWM toggle/set mode:
The output is reset when the timer value becomes equal to capture/compare data CCRx. It is set when the timer value becomes equal to CCR0.

10.5.2 Output Control Block

The output control block prepares the value of the OUTx signal, which is latched into the OUTx flip-flop with the next positive timer clock edge, as shown in Figure 10–23 and Table 10–2. The equal signals EQUx and EQU0 are sampled during the negative level of the timer clock, as shown in Figure 10–23.

Figure 10–23. Output Control Block



EQU0 delayed is used in up mode, not EQU0. EQU0 is active high when TAR = CCR0. EQU0 delayed is active high when TAR = 0.

Table 10–2. State of OUTx at Next Rising Edge of Timer Clock

Mode	EQU0	EQUx	D
0	x	x	x(OUTx bit)
1	x	0	OUTx (no change)
	x	1	1 (set)
2	0	0	OUTx (no change)
	0	1	OUTx (toggle)
	1	0	0 (reset)
	1	1	1 (set)
3	0	0	OUTx (no change)
	0	1	1 (set)
	1	0	0 (reset)
	1	1	1 (set)
4	x	0	OUTx (no change)
	x	1	OUTx (toggle)
5	x	0	OUTx (no change)
	x	1	0 (reset)
6	0	0	OUTx (no change)
	0	1	OUTx (toggle)
	1	0	1 (set)
	1	1	0 (reset)
7	0	0	OUTx (no change)
	0	1	0 (reset)
	1	0	1 (set)
	1	1	0 (reset)

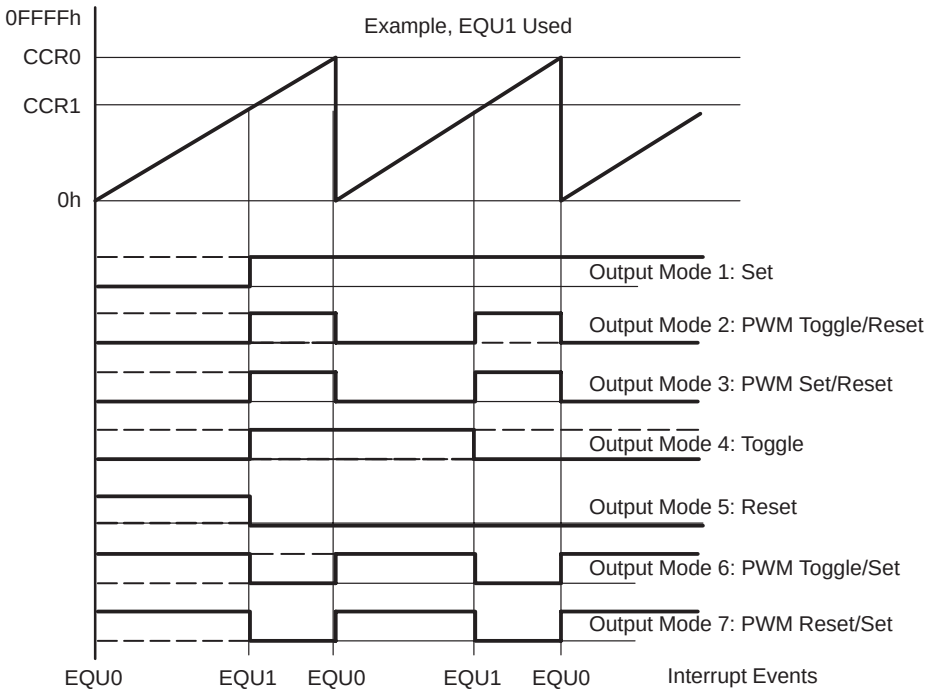
10.5.3 Output Examples

The following are some examples of possible output signals using the various timer and output modes.

10.5.3.1 Output Examples—Timer in Up Mode

The OUTx signal is changed when the timer *counts* up to the CCRx value, and rolls from CCR0 to zero, depending on the output mode, as shown in Figure 10–24.

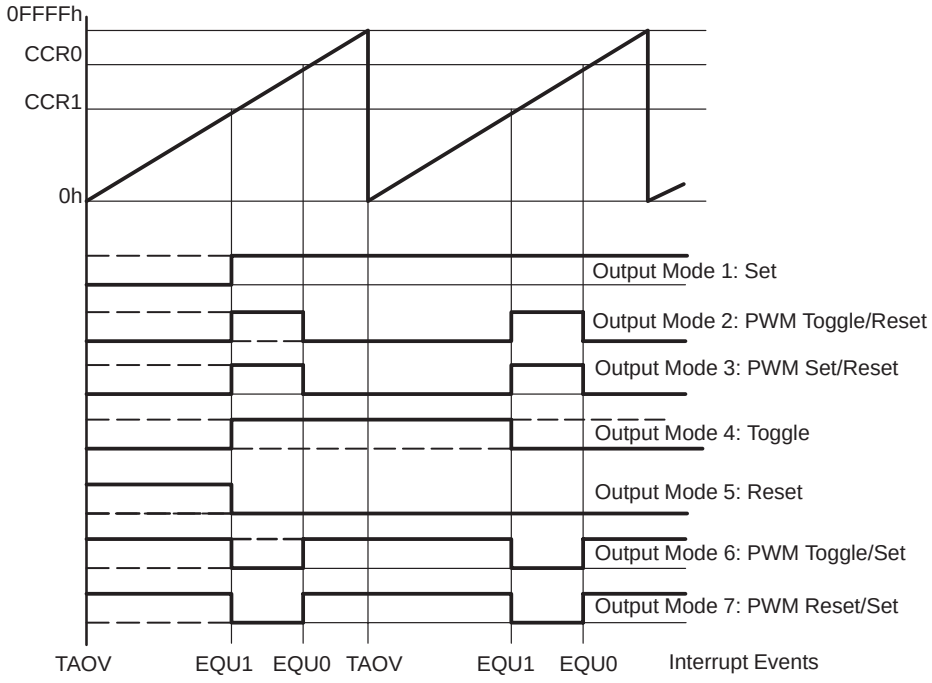
Figure 10–24. Output Examples—Timer in Up Mode



10.5.3.2 Output Examples—Timer in Continuous Mode

The OUTx signal is changed when the timer reaches the CCRx and CCR0 values, depending on the output mode, as shown in Figure 10–25.

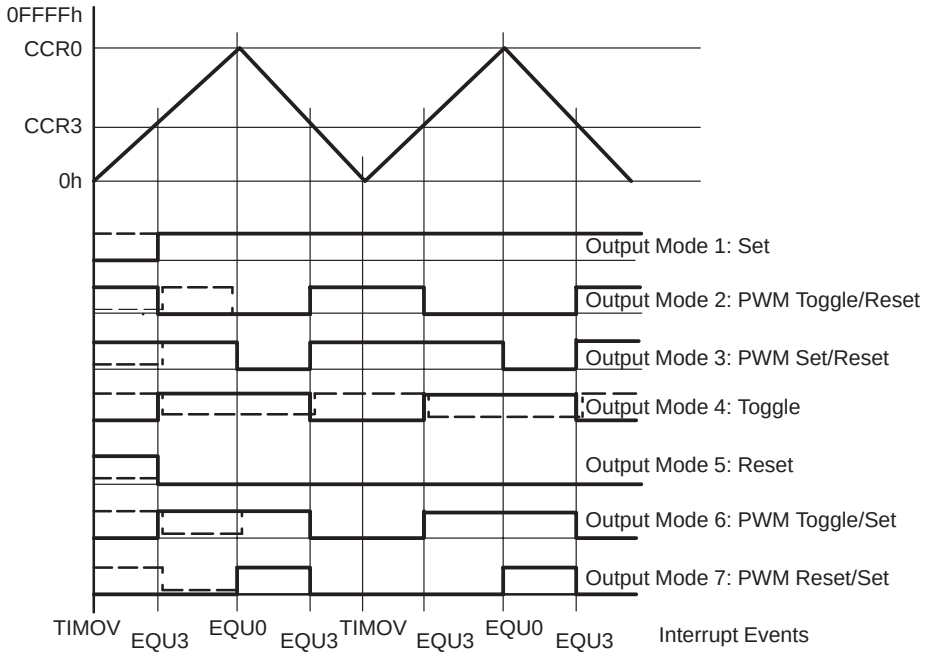
Figure 10–25. Output Examples—Timer in Continuous Mode



10.5.3.3 Output Examples—Timer in Up/Down Mode

The OUTx signal changes when the timer equals CCRx in either count direction and when the timer equals CCR0, depending on the output mode, as shown in Figure 10–26.

Figure 10–26. Output Examples—Timer in Up/Down Mode (I)



10.6 Timer_A Registers

The Timer_A registers, described in Table 10–3, are word-structured and must be accessed using word instructions.

Table 10–3. Timer_A Registers

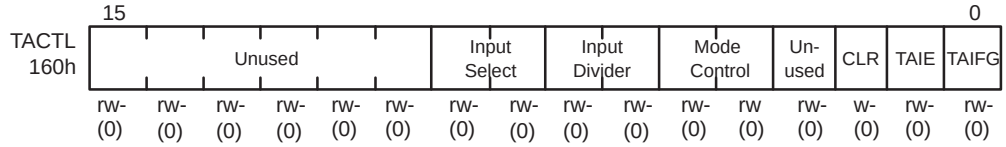
Register	Short Form	Register Type	Address	Initial State
Timer_A control	TACTL	Read/write	160h	POR reset
Timer_A register	TAR	Read/write	170h	POR reset
Cap/com control 0	CCTL0	Read/write	162h	POR reset
Capture/compare 0	CCR0	Read/write	172h	POR reset
Cap/com control 1	CCTL1	Read/write	164h	POR reset
Capture/compare 1	CCR1	Read/write	174h	POR reset
Cap/com control 2	CCTL 2	Read/write	166h	POR reset
Capture/compare 2	CCR2	Read/write	176h	POR reset
Cap/com control 3†	CCTL3	Read/write	168h	POR reset
Capture/compare 3†	CCR3	Read/write	178h	POR reset
Cap/com control 4†	CCTL4	Read/write	16Ah	POR reset
Capture/compare 4†	CCR4	Read/write	17Ah	POR reset
Interrupt vector	TAIV	Read	12Eh	(POR reset)

† 14x devices only

10.6.1 Timer_A Control Register TACTL

The timer and timer operation control bits are located in the timer control register (TACTL) shown in Figure 10–27. All control bits are reset automati–cally by the POR signal, but are not affected by the PUC signal. The control register must be accessed using word instructions.

Figure 10–27. Timer_A Control Register TACTL



Bit 0: TAIFG: This flag indicates a timer overflow event.
- Up mode: TAIFG is set if the timer *counts* from CCR0 value to 0000h.
- Continuous mode: TAIFG is set if the timer *counts* from 0FFFFh to 0000h.
- Up/down mode: TAIFG is set if the timer *counts* down from 0001h to 0000h.

Bit 1: Timer overflow interrupt enable (TAIE) bit. An interrupt request from the timer overflow bit is enabled if this bit is set, and is disabled if reset.

Bit 2: Timer clear (CLR) bit. The timer and input divider are reset with the POR signal, or if bit CLR is set. The CLR bit is automatically reset and is always read as zero. The timer starts in the upward direction with the next valid clock edge, unless halted by cleared mode control bits.

Bit 3: Not used

Bits 4, 5: Mode control: Table 10–4 describes the mode control bits.

Table 10–4. Mode Control

MC1	MC0	Count Mode	Description
0	0	Stop	Timer is halted.
0	1	Up to CCR0	Timer counts up to CCR0 and restarts at 0.
1	0	Continuous up	Timer counts up to 0FFFFh and restarts at 0.
1	1	Up/down	Timer continuously *counts* up to CCR0 and back down to 0.

Bits 6, 7: Input divider control bits. Table 10–5 describes the input divider control bits.

Table 10–5. Input Clock Divider Control Bits

ID1	ID0	Operation	Description
0	0	/1	Input clock source is passed to the timer.
0	1	/2	Input clock source is divided by two.
1	0	/4	Input clock source is divided by four.
1	1	/8	Input clock source is divided by eight.

Bits 8, 9: Clock source selection bits. Table 10–6 describes the clock source selections.

Table 10–6. Clock Source Selection

SSEL1	SSEL0	O/P Signal	Comment
0	0	TACLK	See data sheet device description.
0	1	ACLK	Auxiliary clock ACLK is used.
1	0	SMCLK	System clock SMCLK.
1	1	INCLK	See device description in data sheet.

Bits 10 to 15: Unused

Note: Changing Timer_A Control Bits

If the timer operation is modified by the control bits in the TACTL register, the timer should be halted during this modification. Critical modifications are the input select bits, input divider bits, and the timer clear bit. Asynchronous clocks, input clock, and system clock can result in race conditions where the timer reacts unpredictably.

The recommended instruction flow is:

1) Modify the control register and stop the timer.

2) Start the timer operation.

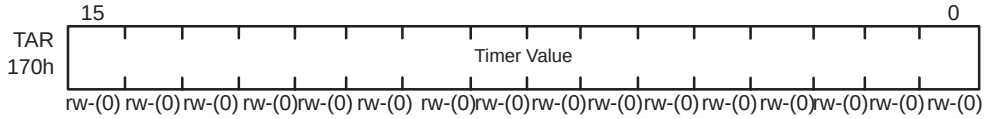
For example:

```
MOV #01C6,&TACTL    ; ACLK/8, timer stopped, timer cleared
BIS  #10h,&TACTL    ; Start timer with up mode
```

10.6.2 Timer_A Register TAR

The TAR register is the value of the timer.

Figure 10–28. TAR Register



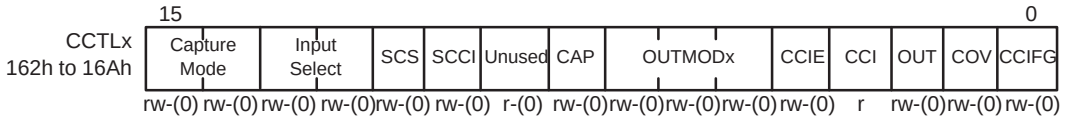
Note: Modifying Timer A Register TAR

When ACLK, SMCLK, or the external clock TACLK or INCLK is selected for the timer clock, any write to timer register TAR should occur while the timer is not operating; otherwise, the results may be unpredictable. In this case, the timer clock is asynchronous to the CPU clock MCLK and critical race conditions exist.

10.6.3 Capture/Compare Control Register CCTLx

Each capture/compare block has its own control word CCTLx, shown in Figure 10–29. The POR signal resets all bits of CCTLx; the PUC signal does not affect these bits.

Figure 10–29. Capture/Compare Control Register CCTLx



Bit 0: Capture/compare interrupt flag CCIFGx

Capture mode:

If set, it indicates that a timer value was captured in the CCRx register.

Compare mode:

If set, it indicates that a timer value was equal to the data in the CCRx register.

CCIFG0 flag:

CCIFG0 is automatically reset when the interrupt request is accepted.

CCIFG1 to CCIFG4 flags:

The flag that caused the interrupt is automatically reset after the TAIV word is accessed. If the TAIV register is not accessed, the flags must be reset with software.

No interrupt is generated if the corresponding interrupt enable bit is reset, but the flag will be set. In this scenario, the flag must be reset by the software.

Setting the CCIFGx flag with software will request an interrupt if the interrupt-enable bit is set.

Bit 1: Capture overflow flag COV

Compare mode selected, CAP = 0:

Capture signal generation is reset. No compare event will set COV bit.

Capture mode selected, CAP = 1:

The overflow flag COV is set if a second capture is performed before the first capture value is read. The overflow flag must be reset with software. It is not reset by reading the capture value.

Bit 2: The OUTx bit determines the value of the OUTx signal if the output mode is 0.

Bit 3: Capture/compare input signal CCIx:
The selected input signal (CCIxA, CCIxB, V_{CC}. or GND) can be read by this bit. See Figure 10–18.

Bit 4: Interrupt enable CCIEx: Enables or disables the interrupt request signal of capture/compare block x. Note that the GIE bit must also be set to enable the interrupt.
0: Interrupt disabled
1: Interrupt enabled

Bits 5 to 7: Output mode select bits:
Table 10–7 describes the output mode selections.

Table 10–7. Capture/Compare Control Register Output Mode

Bit Value	Output Mode	Description
0	Output only	The OUTx signal reflects the value of the OUTx bit
1	Set	EQUx sets OUTx
2	PWM toggle/reset	EQUx toggles OUTx. EQU0 resets OUTx.
3	PWM set/reset	EQUx sets OUTx. EQU0 resets OUTx
4	Toggle	EQUx toggles OUTx signal.
5	Reset	EQUx resets OUTx.
6	PWM toggle/set	EQUx toggles OUTx. EQU0 sets OUTx.
7	PWM reset/set	EQUx resets OUTx. EQU0 sets OUTx.

Note: OUTx updates with rising edge of timer clock for all modes except mode 0. Modes 2, 3, 6, 7 not useful for output unit 0.

Bit 8: CAP sets capture or compare mode.
0: Compare mode
1: Capture mode

Bit 9: Read only, always read as 0.

Bit 10: SCCIx bit:
The selected input signal (CCIxA, CCIxB, V_{CC}, or GND) is latched with the EQUx signal into a transparent latch and can be read via this bit.

Bit 11: SCSx bit:
This bit is used to synchronize the capture input signal with the timer clock.
0: asynchronous capture
1: synchronous capture

Bits 12, 13: Input select, CCIS0 and CCIS1:
These two bits define the capture signal source. These bits are not used in compare mode.
0 Input CCIxA is selected
1 Input CCIxB is selected
2 GND
3 V_{CC}

Bits 14, 15: Capture mode bits:
Table 10–8 describes the capture mode selections.

Table 10–8. Capture/Compare Control Register Capture Mode

Bit Value	Capture Mode	Description
0	Disabled	The capture mode is disabled.
1	Positive Edge	Capture is done with rising edge.
2	Negative Edge	Capture is done with falling edge.
3	Both Edges	Capture is done with both rising and falling edges.

Note: Simultaneous Capture and Capture Mode Selection

Captures must not be performed simultaneously with switching from compare to capture mode. Otherwise, the result in the capture/compare register will be unpredictable.

The recommended instruction flow is:

1) Modify the control register to switch from compare to capture.
2) Capture

For example:

```
BIS  #CAP,&CCTL2        ; Select capture with register CCR2
XOR #CCIS1,&CCTL2       ; Software capture:    CCIS0 = 0
                        ;                      Capture mode = 3
```

10.6.4 Timer_A Interrupt Vector Register

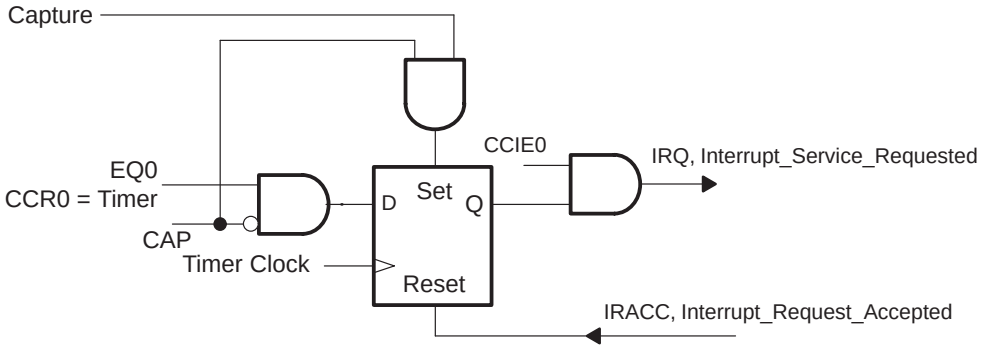
Two interrupt vectors are associated with the 16-bit Timer_A module:

- CCR0 interrupt vector (highest priority)
- TAIV interrupt vector for flags CCIFG1–CCIFGx and TAIFG.

10.6.4.1 CCR0 Interrupt Vector

The interrupt flag associated with capture/compare register CCR0, as shown in Figure 10–30, is set if the timer value is equal to the compare register value.

Figure 10–30. Capture/Compare Interrupt Flag



Capture/compare register 0 has the highest Timer_A interrupt priority, and uses its own interrupt vector.

10.6.4.2 Vector Word, TAIFG, CCIFG1 to CCIFG4 Flags

The CCIFGx (other than CCIFG0) and TAIFG interrupt flags are prioritized and combined to source a single interrupt as shown in Figure 10–31. The interrupt vector register TAIV (shown in Figure 10–32) is used to determine which flag requested an interrupt.

Figure 10–31. Schematic of Capture/Compare Interrupt Vector Word

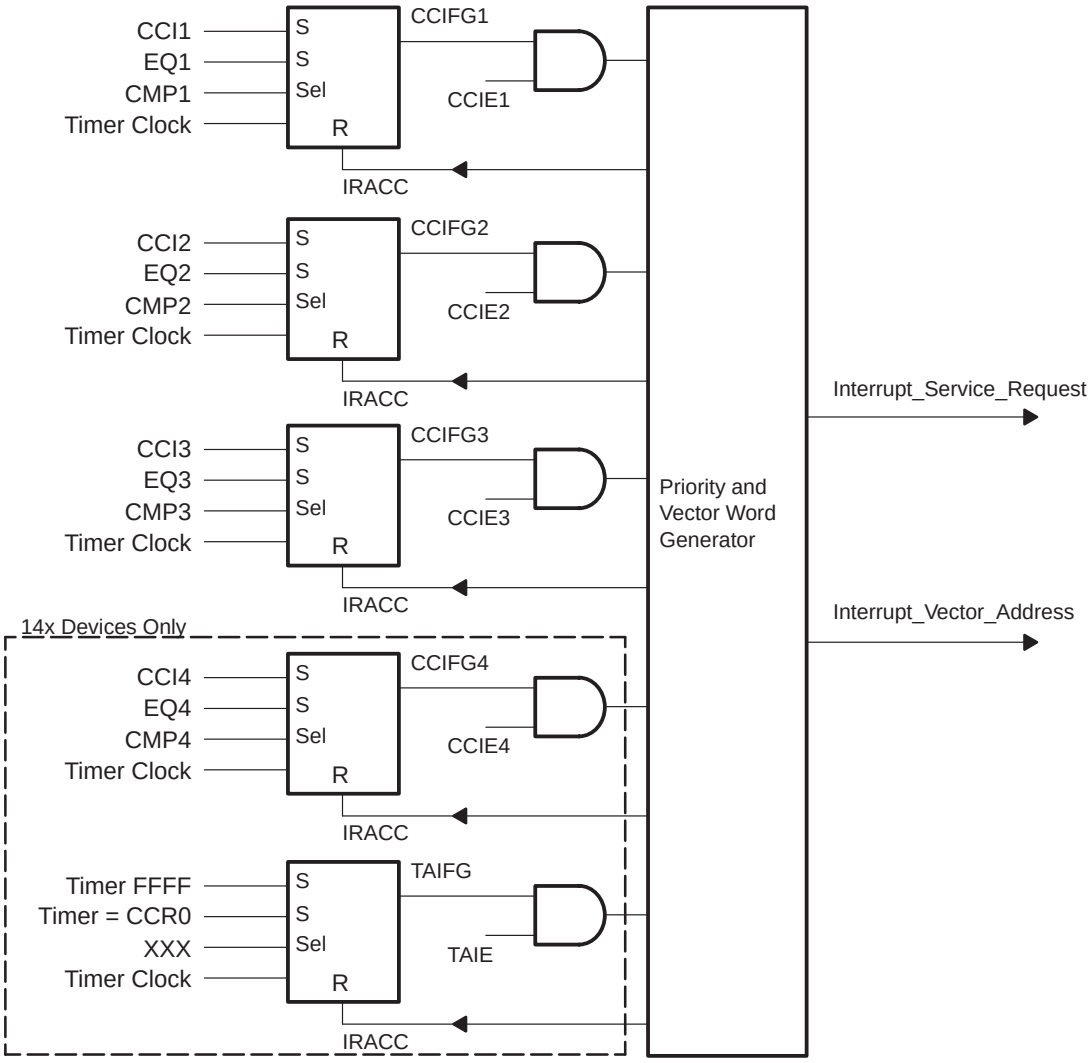
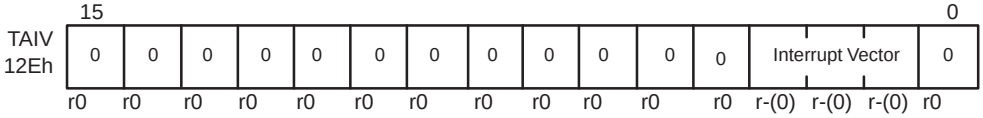


Figure 10–32. Vector Word Register



The flag with the highest priority generates a number from 2 to 12 in the TAIV register as shown in Table 10–9. (If the value of the TAIV register is 0, no interrupt is pending.) This number can be added to the program counter to automatically enter the appropriate software routine without the need for reading and evaluating the interrupt vector. The software example in section 10.6.4.3 shows this technique.

Table 10–9. Vector Register TAIV Description

Interrupt Priority	Interrupt Source	Short Form	Vector Register TAIV Contents
Highest†	Capture/compare 1	CCIFG1	2
	Capture/compare 2	CCIFG2	4
	Capture/compare 3‡	CCIFG3	6
	Capture/compare 4‡	CCIFG4	8
	Timer overflow	TAIFG	10
	Reserved		12
Lowest	Reserved		14
	No interrupt pending		0

† Highest pending interrupt other than CCIFG0. CCIFG0 is always the highest priority Timer_A interrupt.
‡ 14x devices only

Accessing the TAIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, then another interrupt will be immediately generated after servicing the initial interrupt. For example, if both CCIFG2 and CCIFG3 are set, when the interrupt service routine accesses the TAIV register (either by reading it or by adding it directly to the PC), CCIFG2 will be reset automatically. After the RETI instruction of the interrupt service routine is executed, the CCIFG3 flag will generate another interrupt.

Note: Writing to Read-Only Register TAIV

Register TAIV should not be written to. If a write operation to TAIV is performed, the interrupt flag of the highest-pending interrupt is reset. Therefore, the requesting interrupt event is missed. Additionally, writing to this read-only register results in increased current consumption as long as the write operation is active.

10.6.4.3 Timer Interrupt Vector Register, Software Example

The following software example describes the use of vector word TAIV and the handling overhead. The numbers at the right margin show the necessary cycles for every instruction. The example is written for continuous mode: the time difference to the next interrupt is added to the corresponding compare register.

```
; Software example for the interrupt part          Cycles
;
; Interrupt handler for Capture/Compare Module 0.
; The interrupt flag CCIFG0 is reset automatically
;
TIMMOD0   ...       ; Start of handler Interrupt latency  6
          RETI                                            5
;
; Interrupt handler for Capture/Compare Modules 1 to 4.
; The interrupt flags CCIFGx and TAIFG are reset by
; hardware. Only the flag with the highest priority
; responsible for the interrupt vector word is reset.
TIM_HND   $                 ; Interrupt latency         6
          ADD   &TAIV,PC    ; Add offset to Jump table  3
          RETI              ; Vector 0: No interrupt    5
```

```
          JMP    TIMMOD1      ; Vector 2: Module 1        2
          JMP    TIMMOD2      ; Vector 4: Module 2        2
          JMP    TIMMOD3      ; Vector 6: Module 3        2
          JMP    TIMMOD4      ; Vector 8: Module 4        2
;
; Module 5. Timer Overflow Handler: the Timer Register is
; expanded into the RAM location TIMEXT (MSBs)
;
TIMOVH                      ; Vector 10: TIMOV Flag
          INC    TIMEXT       ; Handle Timer Overflow     4
          RETI                                            5
;
TIMMOD2                     ; Vector 4: Module 2
          ADD    #NN,&CCR2    ; Add time difference       5
          ...                 ; Task starts here
          RETI                ; Back to main program      5
;
;
TIMMOD1                     ; Vector 2: Module 1
          ADD    #MM,&CCR1    ; Add time difference       5
          ...                 ; Task starts here
          RETI                ; Back to main program      5
; If all five CCR registers are not implemented on a
; device, the interrupt vectors for the register that are
; present must still be handled.

TIMMOD4
          RETI            ; Simply return                 5

; The Module 3 handler shows a way to look if any other
; interrupt is pending: 5 cycles have to be spent, but
; 9 cycles may be saved if another interrupt is pending
;
TIMMOD3                   ; Vector 6: Module 3
          ADD    #PP,&CCR3 ; Add time difference          5
          ...              ; Task starts here
          JMP    TIM_HND   ; Look for pending interrupts  2
;
          .SECT "VECTORS",0FFF0h   ; Interrupt Vectors
; The vector address may be different for different
; devices.
;
          .WORD  TIM_HND   ; Vector for Capture/Compare
                           ; Module 1..4 and timer overflow
                           ; TAIFG
          .WORD  TIMMOD0   ; Vector for Capture/Compare
                           ; Module 0
```

If the CPU clock MCLK was turned off (CPUOFF=1), then two or three additional cycles need to be added for synchronous start of the CPU. The delta of one clock cycle is caused when clocks are asynchronous to the restart of CPU clock MCLK.

The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles (but not the task handling itself), as described:

- ❏ Capture/compare block CCR0 — 11 cycles
- ❏ Capture/compare blocks CCR1 to CCR4 — 16 cycles
- ❏ Timer overflow TAIFG — 14 cycles

10.6.4.4 Timing Limits

With the TAIV register and the previous software, the shortest repetitive time distance t_{CRmin} between two events using a compare register is:

$$t_{CRmin} = t_{taskmax} + 16 \times t_{cycle}$$

With: $t_{taskmax}$ Maximum (worst case) time to perform the task during the interrupt routine (for example, incrementing a counter)

t_{cycle} Cycle time of the system frequency MCLK

The shortest repetitive time distance t_{CLmin} between two events using a capture register is:

$$t_{CLmin} = t_{taskmax} + 16 \times t_{cycle}$$

10.7 Timer_A UART

The Timer_A is uniquely capable of implementing a UART function, with the following features:

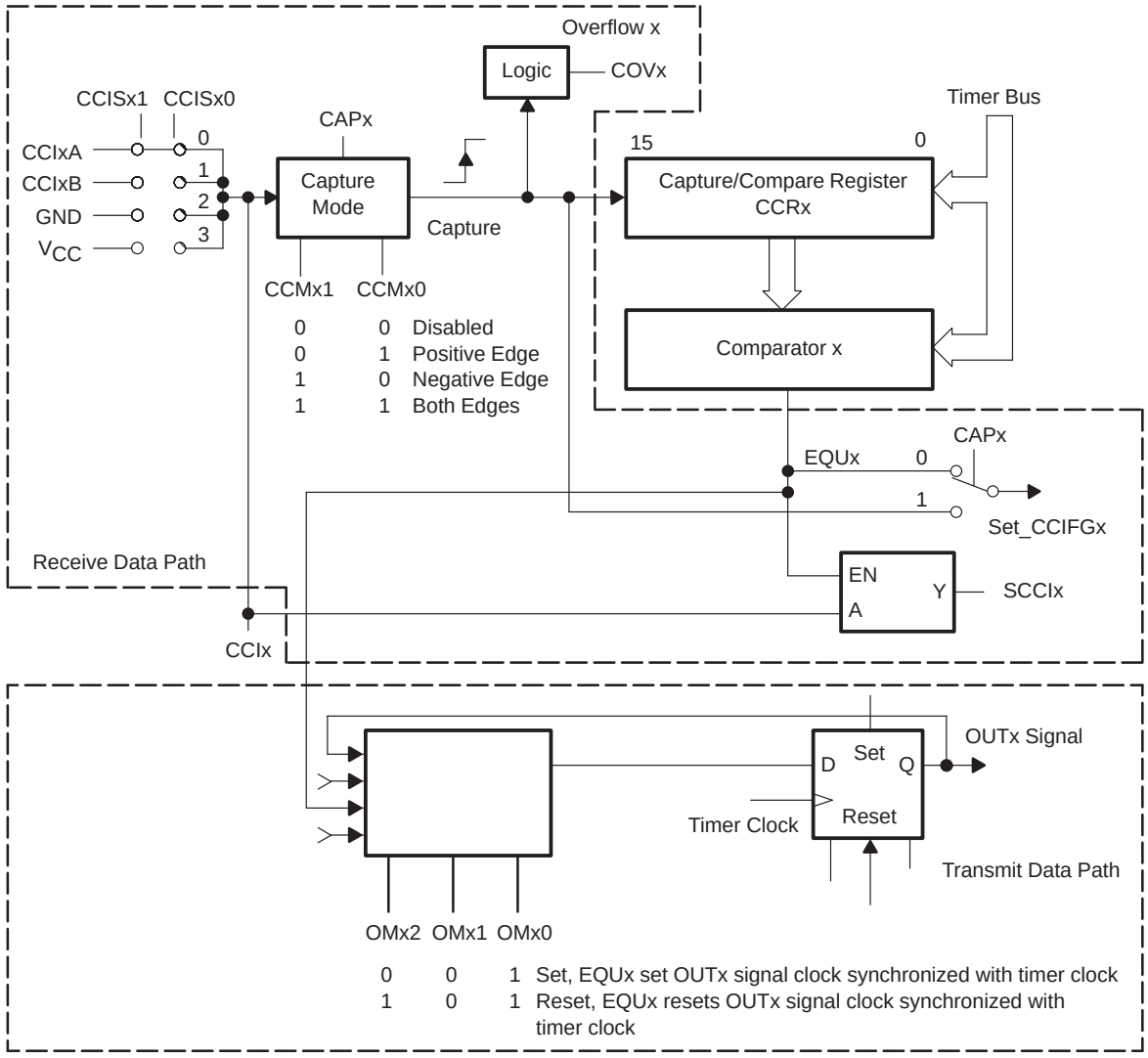
- ❑ Automatic start-bit detection – even from ultralow-power modes
- ❑ Hardware baud-rate generation
- ❑ Hardware latching of RXD and TXD data
- ❑ Baud rates of 75 to 115,200 baud
- ❑ Full-duplex operation

This UART implementation is different from other microcontroller implementations where a UART may be implemented with general-purpose I/O and manual bit manipulation via software polling. Those implementations require great CPU overhead and therefore increase power consumption and decrease the usability of the CPU.

The transmit feature uses one compare function to shift data through the output unit to the selected pin. The baud rate is ensured by reconfiguring the compare data with each interrupt.

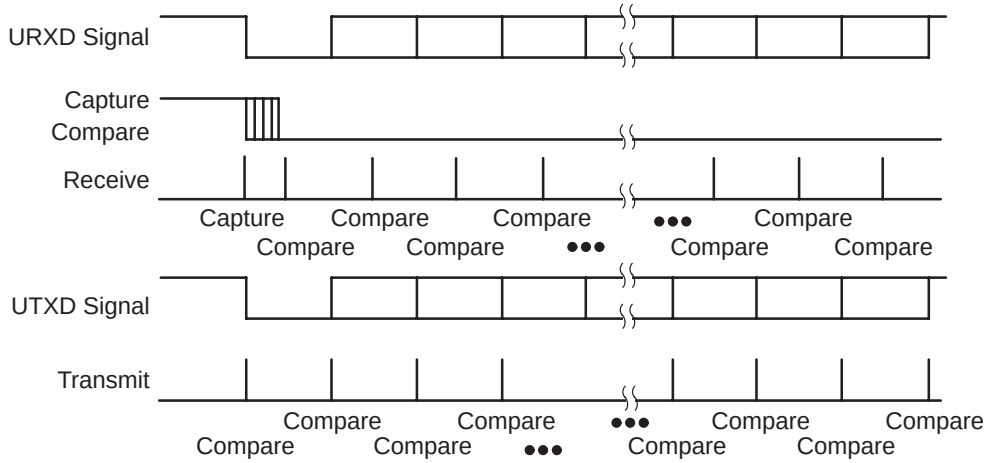
The receive feature uses one capture/compare function to shift pin data into memory through bit SCCIx. The receive start time is recognized by capturing the timer data with the negative edge of the input signal. The same capture/compare block is then switched to compare mode and the receive bits are latched automatically with the EQUx signal. The interrupt routine collects the bits for later software processing. Figure 10–33 illustrates the UART implementation.

Figure 10–33. UART Implementation



One capture/compare block is used when half-duplex communication mode is desired. Two capture/compare blocks are used for full-duplex mode. Figure 10–34 illustrates the capture/compare timing for the UART.

Figure 10–34. Timer_A UART Timing



A complete application note including connection diagrams and complete software listing may be found at *www.ti.com/sc/msp430*.

Chapter 11

Timer_B

This section describes the basic functions of the MSP430 general-purpose 16-bit Timer_B. Timer_B implementation differs among MSP430 devices. Always check the device's data sheet to determine the connections and the number of identical capture/compare registers. Also, the data sheets use additional nomenclature to indicate the number of capture/compare registers implemented for a specific device. For example, if *Timer_B3* is discussed in a data sheet, then that device's implementation of Timer_B contains 3 capture/compare registers.

In its default condition, Timer_B operates identically to Timer_A, except the SCCI bit is not implemented on Timer_B.

Note:

Throughout this chapter, the word *count* is used in the text. As used in these instances, it refers to the literal act of counting. It means that the counter must be in the process of counting for the action to take place. If a particular value is directly written to the counter, then the associated action will not take place. For example, the CCR0 interrupt flag is set when the timer *counts up to* the value in CCR0 compare latch TBCL0. The counter *must count* from TBCL0 –1 to TBCL0. If the TBCL0 value were simply written directly to the timer with software, the interrupt flag would *not* be set, even though the values in the timer and TBCL0 would be the same.

Topic **Page**

11.1 Introduction

Timer_B is an extremely versatile timer made up of :

- ❑ 16-bit counter with 4 operating modes and four selectable lengths (8-bit, 10-bit, 12-bit, or 16-bit)
- ❑ Selectable and configurable clock source
- ❑ Up to seven independently-configurable capture/compare registers with configurable inputs and double-buffered compare registers.
- ❑ Up to seven individually-configurable output modules with eight output modes

Timer_B can support multiple, simultaneous, timings; multiple capture/compares; multiple output waveforms such as PWM signals; and any combination of these. In addition, with the double-buffering of compare data, multiple PWM periods can be updated simultaneously.

Additionally, Timer_B has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers on captures or compares. Each capture/compare block is individually configurable and can produce interrupts on compares or on rising, falling or both edges of an external capture signal.

The block diagram of Timer_B is shown in Figure 11–1.

11.1.1 Similarities and Differences From Timer_A

Timer_B is almost identical to Timer_A (except for a few enhancements noted below) and operates identically to Timer_A in it's default condition.

Timer_B is different from Timer_A in the following ways:

1) The length of Timer_B is programmable to be 8, 10, 12, or 16 bits, whereas Timer_A is only a 16-bit timer.
2) The SCCI bit functionality of the capture/compare registers of Timer_A is not implemented in Timer_B.
3) The function of the capture/compare registers for the compare mode of Timer_B has changed slightly.
4) On some devices a pin is implemented to put all Timer_B outputs into a high-impedance state. Check the device data sheet for the presence of this pin.

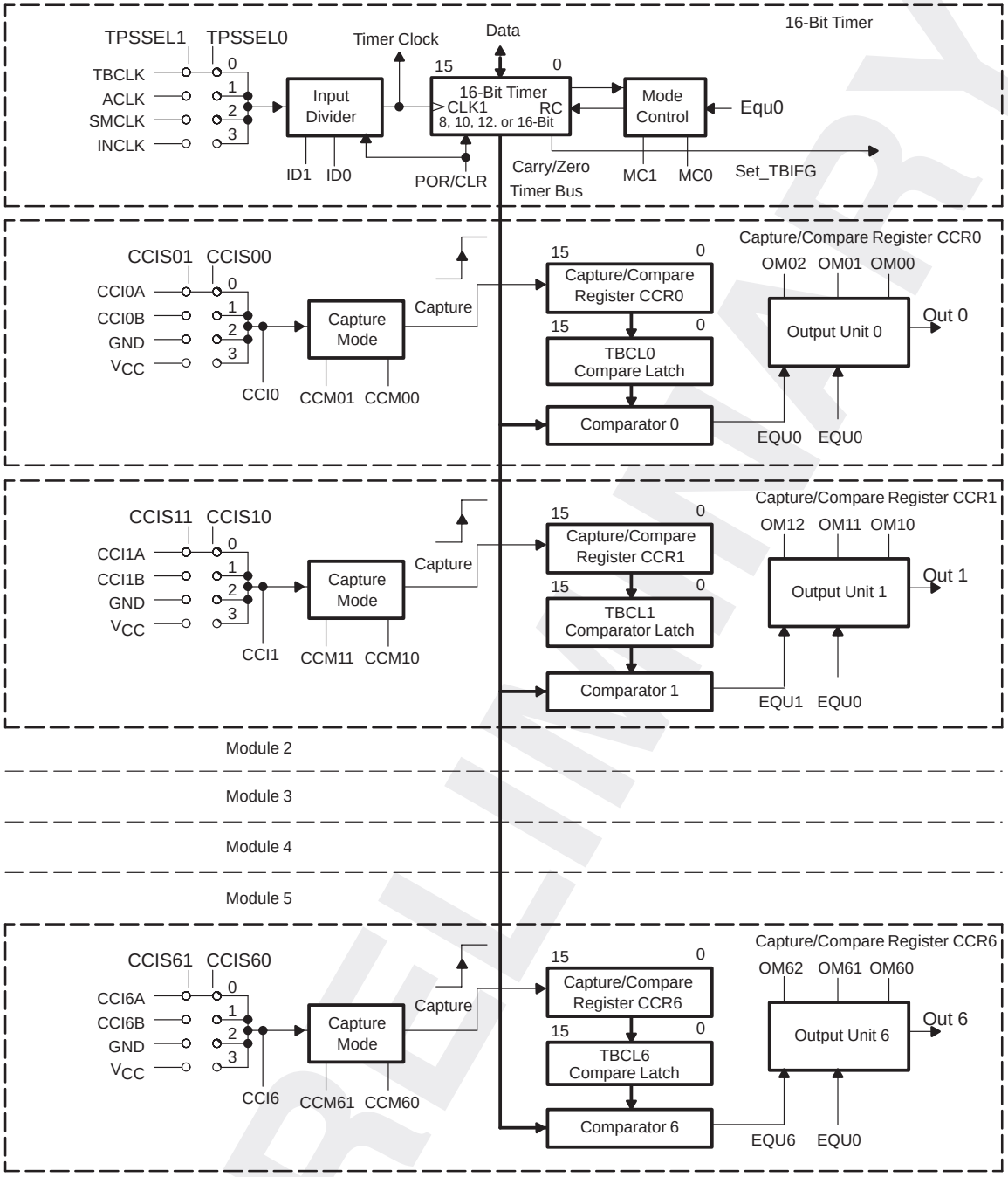
On Timer_A, the capture/compare register CCRx holds the data for the comparison to the timer value. On Timer_B, each CCRx acts as a buffer for a compare latch, and the compare latch holds the data used for the comparison. So, compare data is written to each CCRx in both timers; however, in Timer_B, the compare data is then transferred to the compare latch for the comparison. The timing of the transfer of the compare data from

each CCRx register to the corresponding compare latch (TBCLx) is user-selectable to be either immediate, or on a timer event. See section 11.4.2.1 for a complete discussion on using and configuring the compare latches.

The addition of the compare latch gives the user more control over when exactly a compare period updates. In addition, multiple compare latches may be grouped together allowing the compare period of multiple compare registers to be updated simultaneously. This is useful for example when there is a need to change the period or duty cycle of multiple PWM signals simultaneously.

It is useful to note that in Timer_B's default condition, the compare data is immediately transferred from each CCRx register to the corresponding compare latch. Therefore, in the default condition, the compare mode of Timer_B functions identically to the compare mode of Timer_A.

Figure 11–1. Timer_B Block Diagram



11.2 Timer_B Operation

The 16-bit timer has four modes of operation selectable with the MC0 and MC1 bits in the TBCTL register and four selectable lengths, also configured in the TBCTL register. The timer increments or decrements (depending on mode of operation) with each rising edge of the clock signal. The timer can be read or written to with software. Additionally, the timer can generate an interrupt with its ripple-carry output when it overflows.

11.2.1 Timer Length

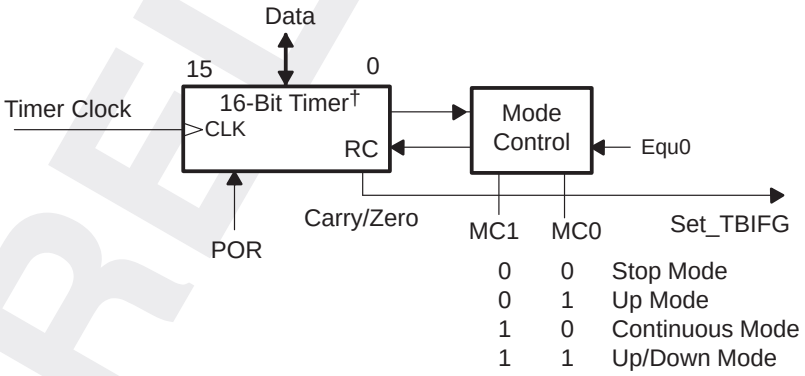
Timer_B is configurable to operate as a true 8-bit, 10-bit, 12-bit, or 16-bit timer. The length of the counter is configured in the TBCTL register. Leading bits are read as zero in 8-bit, 10-bit, and 12-bit mode. Data written to the TBR register in 8-bit, 10-bit, and 12-bit mode will show leading 0s. The maximum count value, $TBR_{(max)}$, for the various lengths is:

Timer_B Configuration	$TBR_{(max)}$
16-bit	0FFFFh
12-bit	0FFFh
10-bit	03FFh
8-bit	0FFh

11.2.2 Timer Mode Control

The timer has four modes of operation as shown in Figure 11–2 and described in Table 11–1: stop, up, continuous, and up/down. The operating mode is software selectable with the MC0 and MC1 bits in the TBCTL register.

Figure 11–2. Mode Control



† Length is selectable for 8-, 10-, 12-, or 16-bit operation.

Table 11–1. Timer Modes

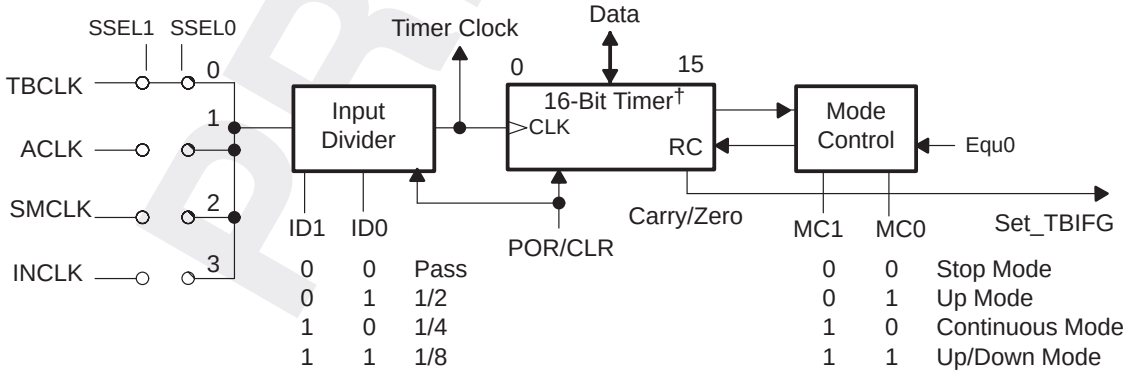
Mode Control			
MC1	**MC0**	**Mode**	**Description**
0	0	Stop	The timer is halted.
0	1	Up	The timer counts upward until its value is equal to the value of compare latch TBCL0. Note: If TBCL0 > $TBR_{(max)}$, the counter counts to zero with the next rising edge of timer clock.
1	0	Continuous	The timer counts upward continuously. The maximum value of TBR [$TBR_{(max)}$] is: 0FFFFh for 16-bit configuration 00FFFh for 12-bit configuration 003FFh for 10-bit configuration 000FFh for 8-bit configuration
1	1	Up/Down	The timer counts up until the timer value is equal to compare latch 0 and then it counts down to zero. Note: If TBCL0 > $TBR_{(max)}$, the counter operates as if it were configured for continuous mode. It will not count down from $TBR_{(max)}$ to zero.

11.2.3 Clock Source Select and Divider

The timer clock can be sourced from internal clocks (i.e. ACLK, MCLK or SMCLK) or from an external source (TBCLK) as shown in Figure 11–3. The clock source is selectable with the SSEL0 and SSEL1 bits in the TBCTL register. It is important to note, that when changing the clock source for the timer, errant timings can occur. For this reason stopping the timer before changing the clock source is recommended.

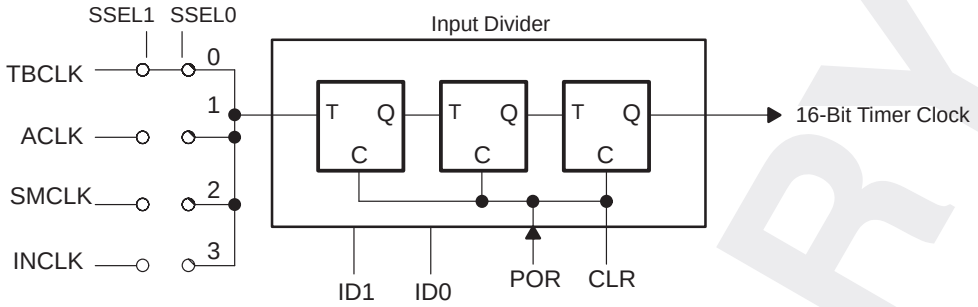
The selected clock source may be passed directly to the timer or divided by 2,4, or 8, as shown in Figure 11–4. The ID0 and ID1 bits in the TBCTL register select the clock division. Note that the input divider is reset by a POR signal or by setting the CLR bit in the TBCTL register (see chapter 3, *System Resets, Interrupts, and Operating Modes*, for more information on the POR signal). Otherwise, the input divider remains unchanged when the timer is modified. The state of the input divider is invisible to software.

Figure 11–3. Schematic of 16-Bit Timer



† Length is selectable for 8-, 10-, 12-, or 16-bit operation.

Figure 11–4. Schematic of Clock Source Select and Input Divider



ID1	ID0	
0	0	Pass
0	1	1/2
1	0	1/4
1	1	1/8

11.2.4 Starting the Timer

The timer may be started or restarted in a variety of ways:

- Release Halt Mode: The timer *counts* in the selected direction when a timer mode other than stop mode is selected with the MCx bits.
- Halted by TBCL0 = 0, restarted by TBCL0 > 0 when the mode is either up or up/down: When the timer mode is selected to be either up or up/down, the timer may be stopped by loading 0 in compare latch 0 (TBCL0) via capture/compare register (CCR0). The timer may then be restarted by loading a nonzero value to TBCL0. In this scenario, the timer starts incrementing in the up direction from zero.
- Setting the CLR bit in TBCTL register: Setting the CLR bit in the TBCTL register clears the timer value and input clock divider value. The timer increments upward from zero with the next clock cycle as long as stop-mode is not selected with the MCx bits.
- TBR is loaded with 0: When the counter (TBR register) is loaded with zero with a software instruction the timer increments upward from zero with the next clock cycle as long as stop-mode is not selected with the MCx bits.

11.3 Timer Modes

11.3.1 Timer—Stop Mode

Stopping and starting the timer is done simply by changing the mode control bits (MCx). The value of the timer is not affected.

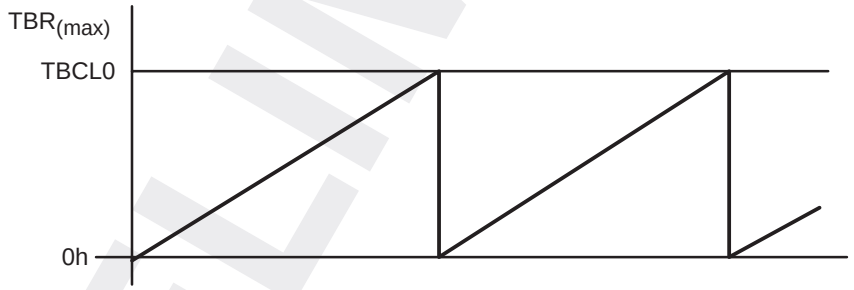
When the timer is stopped from up/down mode and then restarted in up/down mode, the timer counts in the same direction as it was counting before it was stopped. For example, if the timer is in up/down mode and counting in the down direction when the MCx bits are reset, when they are set back to the up/down direction, the timer starts counting in the down direction from its previous value. If this is not desired in an application, the CLR bit in the TBCTL register can be used to clear this direction memory feature.

11.3.2 Timer—Up Mode

The up mode is used if the timer period must be different from the $TBR_{(max)}$ clock cycles of the continuous mode periods. The capture/compare register CCR0 data defines the timer period.

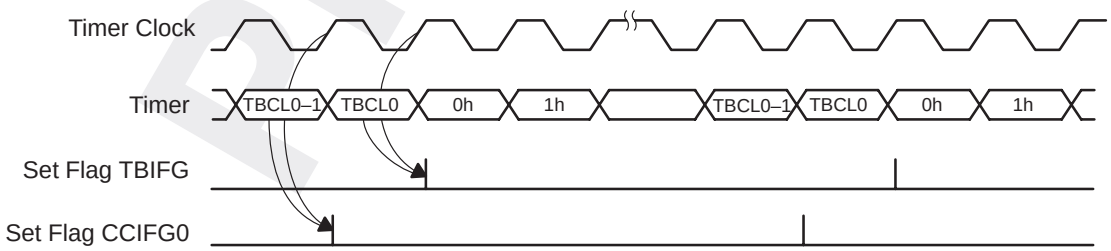
The counter *counts* up to the content of compare latch TBCL0, as shown in Figure 11–5. When the timer value and the value of compare latch TBCL0 are equal (or if the timer value is greater than the TBCL0 value), the timer restarts counting from zero.

Figure 11–5. Timer Up Mode



Flag CCIFG0 is set when the timer equals the TBCL0 value. The TBIFG flag is set when the timer *counts* from TBCL0 to zero. All interrupt flags are set independently of the corresponding interrupt enable bit, but an interrupt is requested only if the corresponding interrupt enable bit and the GIE bit are set. Figure 11–6 shows the flag set cycle.

Figure 11–6. Up Mode Flag Setting



11.3.2.1 Timer in Up Mode—Changing the Period Register TBCL0 Value, Immediate Mode for TBCL0

Changing the timer period register TBCL0 while the timer is running and when the transfer mode from CCR0 is *immediate* can be a little tricky. When the new period is greater than or equal to the old period, the timer simply counts up to the new period and no special attention is required (see Figure 11–7). However, when the new period is less than the old period, the phase of the timer clock during the TBCL0 update affects how the timer reacts to the new period.

If the new, smaller period is transferred from CCR0 to TBCL0 during a high phase of the timer clock, then the timer rolls to zero (or begins counting down when in the up/down mode) on the next rising edge of the timer clock. However, if the new, smaller period is written during a low phase of the timer clock, then the timer continues to increment with the old period for one more clock cycle before adopting the new period and rolling to zero (or beginning counting down). This is shown in Figure 11–8.

Note:

If TBCL0 > $TBR_{(max)}$, the counter rolls to zero with the next rising edge of timer clock.

Figure 11–7. New Period > Old Period

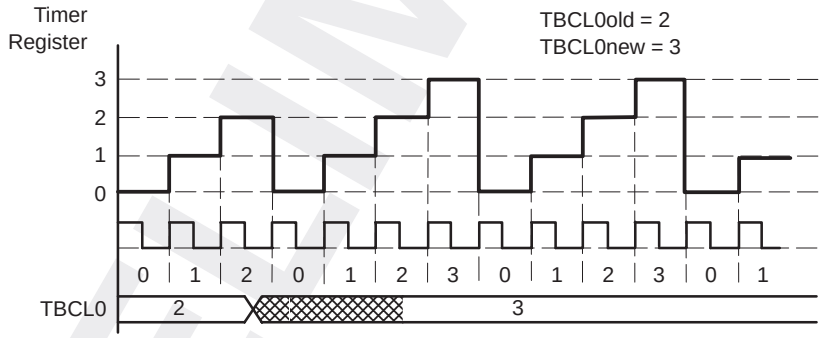
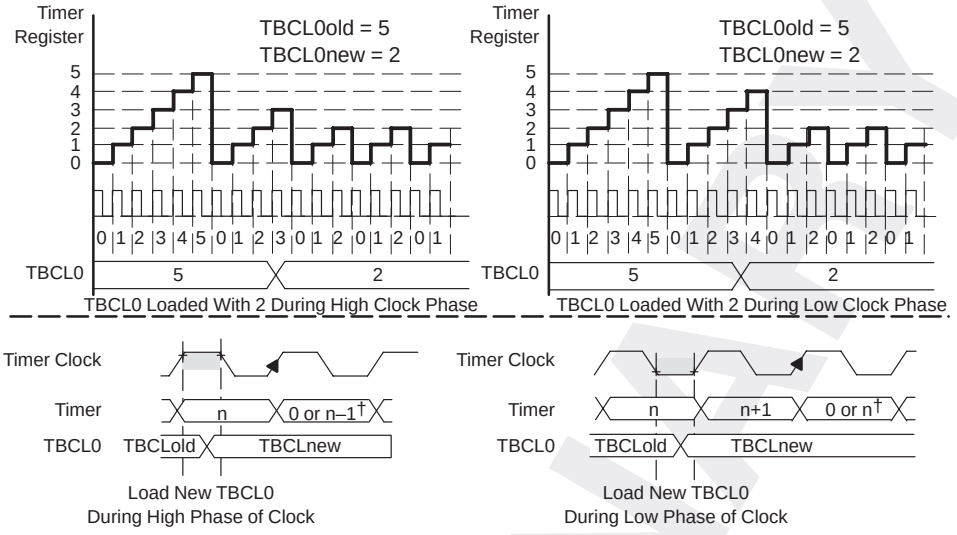


Figure 11–8. New Period < Old Period



† Up mode: 0; up/down mode: n–1

† Up mode: 0; up/down mode: n

11.3.3 Timer—Continuous Mode

The continuous mode is used if the timer period of $TBR_{(max)}$ clock cycles is used for the application. A typical application of the continuous mode is to generate multiple, independent timings. In continuous mode, the capture/compare block 0 works in the same way as the other capture/compare blocks.

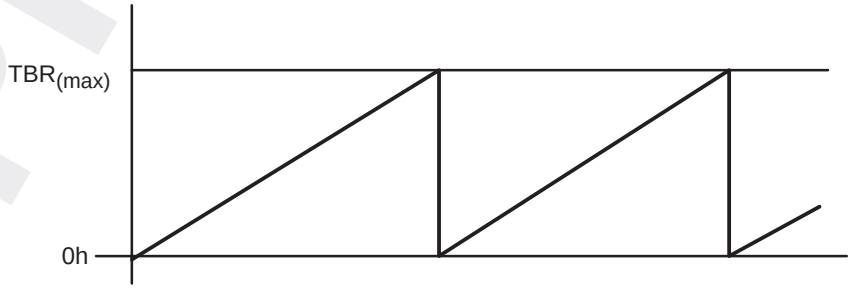
The capture/compare blocks and different output modes of each output unit are useful to capture timer data based on external events or to generate various different types of output signals. Examples of the different output modes used with timer-continuous mode are shown in Figure 11–25.

In continuous mode, the timer starts counting from its present value. The counter counts up to $TBR_{(max)}$ and restarts by counting from zero as shown in Figure 11–9.

The maximum value of TBR [$TBR_{(max)}$] in continuous mode is:

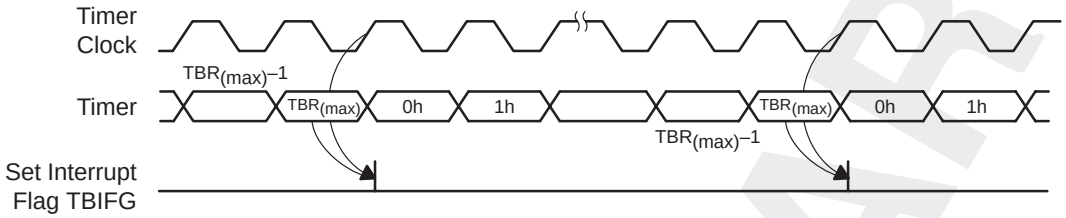
0FFFFh for 16-bit configuration
00FFFh for 12-bit configuration
003FFh for 10-bit configuration
000FFh for 8-bit configuration

Figure 11–9. Timer Continuous Mode



The TBIFG flag is set when the timer *counts* from $TBR_{(max)}$ to zero. The interrupt flag is set independently of the corresponding interrupt enable bit, as shown in Figure 11–10. An interrupt is requested if the corresponding interrupt enable bit and the GIE bit are set.

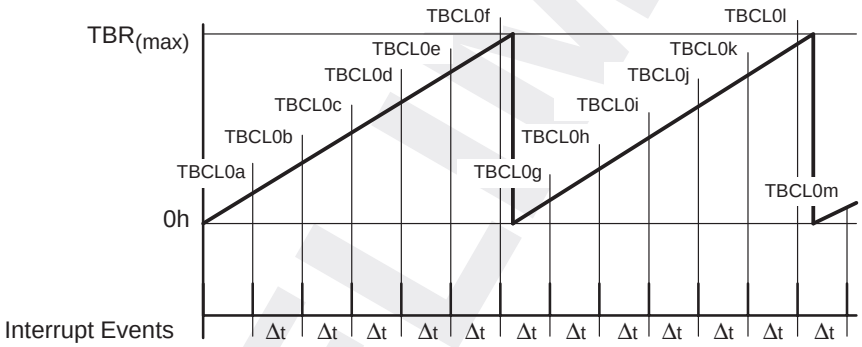
Figure 11–10. Continuous Mode Flag Setting



11.3.3.1 Timer—Use of the Continuous Mode

The continuous mode can be used to generate time intervals for the application software. Each time an interval is completed, an interrupt can be generated. In the interrupt service routine of this event, the time until the next event is added to capture/compare register CCRx (and subsequently compare latch TBCLx) as shown in Figure 11–11. Up to seven independent time events can be generated using all seven capture/compare blocks.

Figure 11–11. Output Unit in Continuous Mode for Time Intervals



Time intervals can be produced with other modes as well, where capture/compare block 0 is used to determine the period. Their handling is more complex since the sum of the old CCRx data and the new period can be higher than the TBCL0 value. When the sum CCRxold plus Δt is greater than the TBCL0 data, the old CCR0 value must be subtracted to obtain the correct time interval.

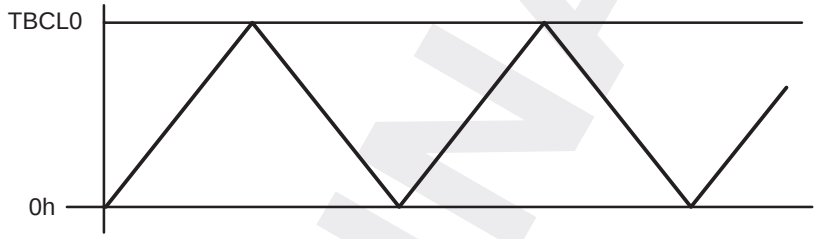
11.3.4 Timer—Up/Down Mode

The up/down mode is used if the timer period must be different from the $TBR_{(max)}$ clock cycles, and if symmetrical pulse waveform generation is needed. In up/down mode, the timer counts up to the content of compare latch TBCL0, then back down to zero, as shown in Figure 11–12. The period is twice the value in the TBCL0 latch.

Note:

If TBCL0 > $TBR_{(max)}$, the counter operates as if it were configured for continuous mode. It will not count down from $TBR_{(max)}$ to zero.

Figure 11–12. Timer Up/Down Mode



The up/down mode also supports applications that require dead times between output signals. For example, to avoid overload conditions, two outputs driving an H-bridge must never be in a high state simultaneously. In the following example (see Figure 11–13), the t_{dead} is:

$$t_{dead} = t_{timer} \times (TBCL1 - TBCL3) =$$

With: t_{dead} Time during which both outputs need to be inactive

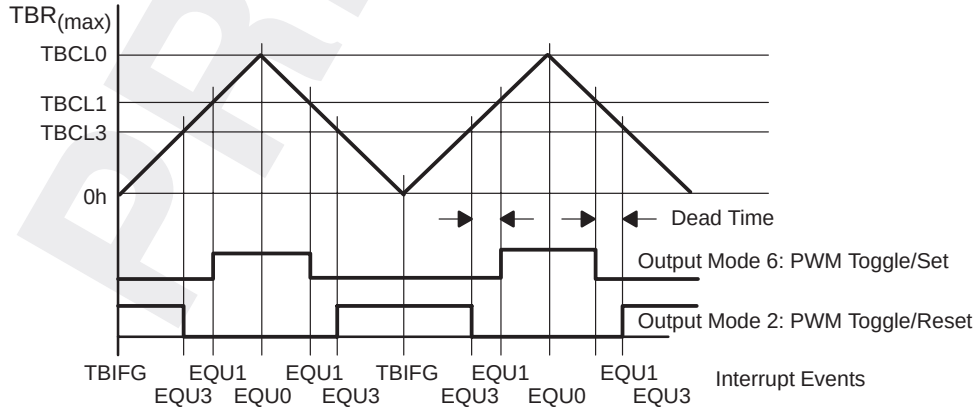
t_{timer} Cycle time of the timer clock

TBCLx Content of compare latch x

Note:

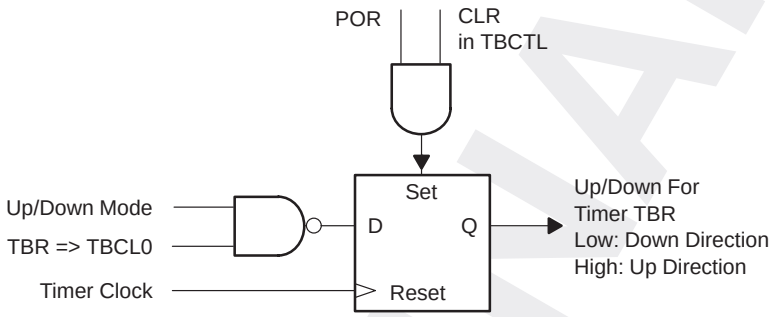
The dead time is ensured by the ability to simultaneously load the compare latches.

Figure 11–13. Output Unit in Up/Down Mode (II)



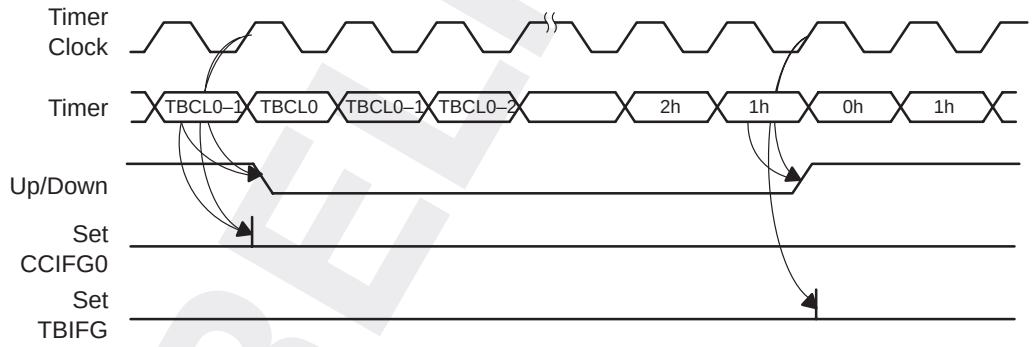
The count direction is always latched with a flip-flop (Figure 11–14). This is useful because it allows the user to stop the timer and then restart it in the same direction it was counting before it was stopped. For example, if the timer was counting down when the MCx bits were reset, then it will continue counting in the down direction if it is restarted in up/down mode. If this is not desired, the CLR bit in the TBCTL register must be used to clear the direction. Note that the CLR bit affects other setup conditions of the timer. Refer to Section 11.6 for a discussion of the Timer_B registers.

Figure 11–14. Timer Up/Down Direction Control



In up/down mode, the interrupt flags (CCIFG0 and TBIFG) are set at equal time intervals (Figure 11–15). Each flag is set only once during the period, but they are separated by 1/2 the timer period. CCIFG0 is set when the timer *counts* from TBCL0–1 to TBCL0, and TBIFG is set when the timer completes *counting* down from 0001h to 0000h. Each flag is capable of producing a CPU interrupt when enabled.

Figure 11–15. Up/Down Mode Flag Setting



11.3.4.1 Timer In Up/Down Mode—Changing the Value of Period Register TBCL0, Immediate Mode for TBCL0

Changing the period value while the timer is running in up/down mode and the transfer mode for TBCL0 is *immediate* is even trickier than in up mode. Like in up mode, the phase of the timer clock when TBCL0 is changed affects the timer's behavior. Additionally, in up/down mode, the direction of the timer also affects the behavior.

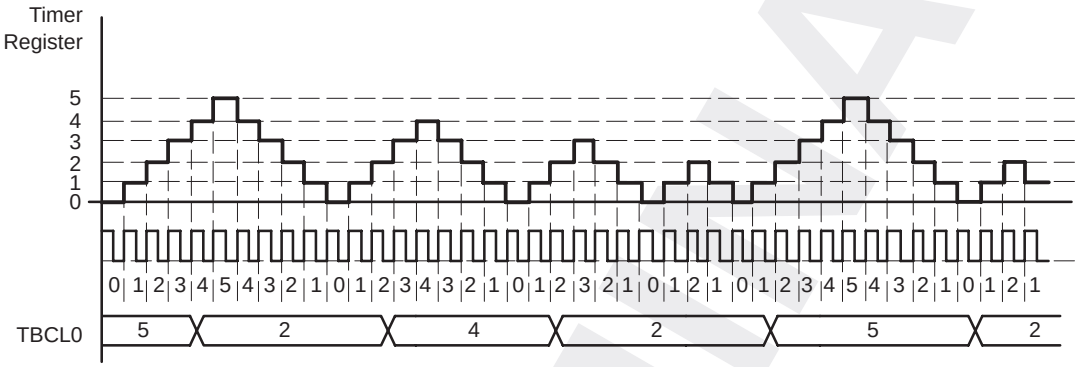
If the timer is counting in the up direction when the new period is transferred from CCR0 to TBCL0, the conditions in the up/down mode are identical to

those in the up mode. See Section 11.3.2.1 for details. However, if the timer is counting in the down direction when TBCL0 is updated, it continues its descent until it reaches zero. The new period takes effect only after the counter finishes counting down to zero. See Figure 11–16.

Note:

If TBCL0 > $TBR_{(max)}$, the counter operates as if it were configured for continuous mode. It will not count down from $TBR_{(max)}$ to zero.

Figure 11–16. Altering TBCL0—Timer in Up/Down Mode



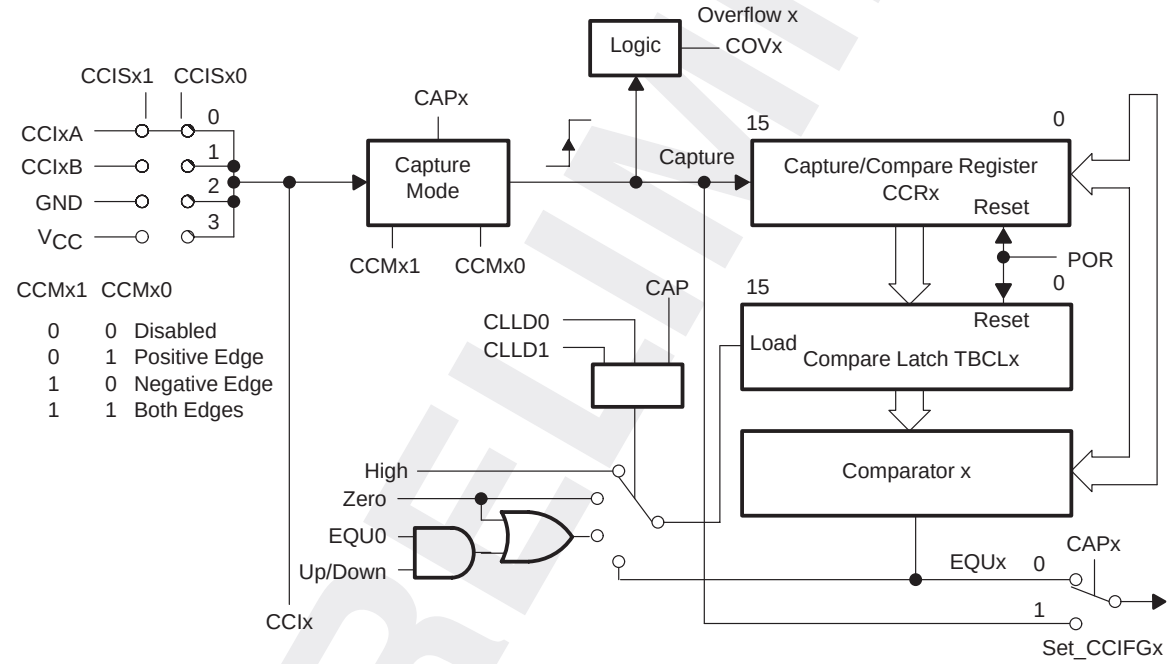
11.4 Capture/Compare Blocks

Seven identical capture/compare blocks (shown in Figure 11–17) provide flexible control for real-time processing. Any one of the blocks may be used to capture the timer data at an applied event, or to generate time intervals. Each time a capture occurs or a time interval is completed, interrupts can be generated from the applicable capture/compare register. The mode bit CAPx, in control word CCTLx, selects the compare or capture operation and the capture mode bits CCMx1 and CCMx0 in control word CCTLx define the conditions under which the capture function is performed.

Both the interrupt enable bit CCIEx and the interrupt flag CCIFGx are used for capture and compare modes. CCIEx enables the corresponding interrupt. CCIFGx is set on a capture or compare event.

The capture inputs CCIxA and CCIxB are connected to external pins or internal signals. Different MSP430 devices may have different signals connected to CCIxA and CCIxB. The data sheet should always be consulted to determine the Timer_B connections for a particular device.

Figure 11–17. Capture/Compare Blocks



11.4.1 Capture/Compare Block—Capture Mode

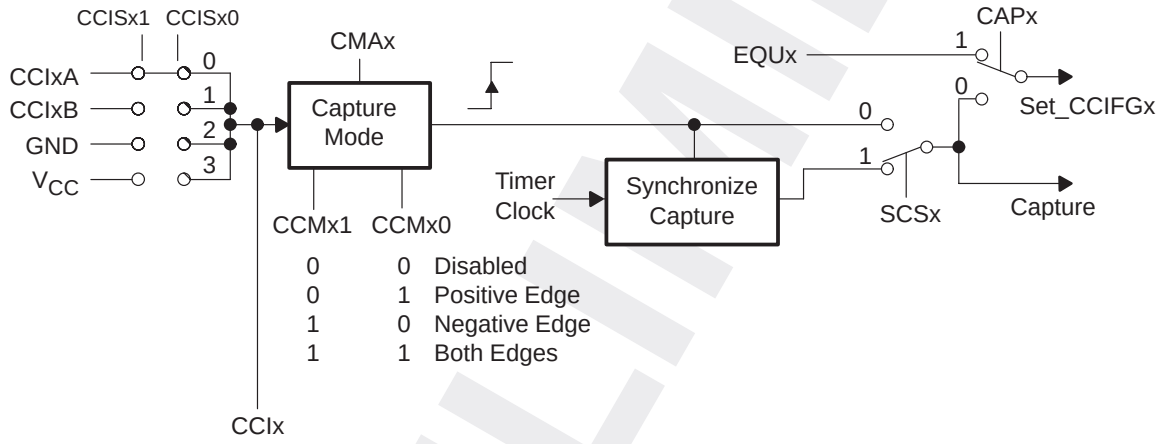
The capture mode is selected if the mode bit CAPx, located in control word CCTLx, is set. The capture mode is used to fix time events. It can be used for speed computations or time measurements. The timer value is copied into the capture register (CCRx) with the selected edge (positive, negative, or both) of the input signal. Captures may also be initiated by software as described in section 11.4.1.1.

If a capture is performed:

- The interrupt flag CCIFGx, located in control word CCTLx, is set.
- An interrupt is requested if both interrupt enable bits CCIEx and GIE are set.

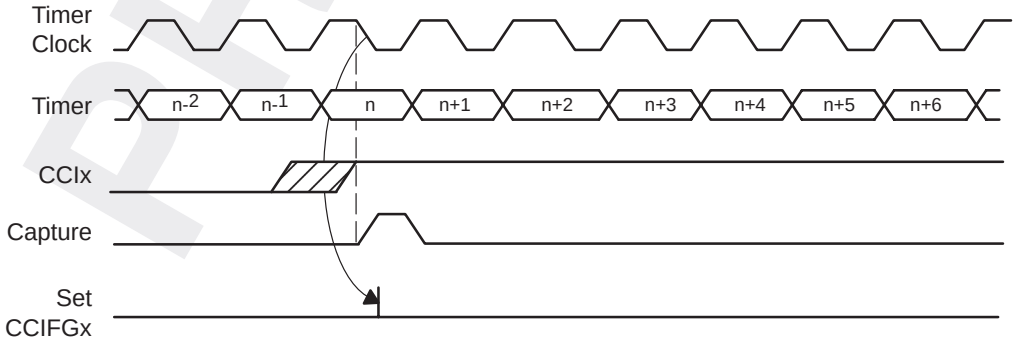
The input signal to the capture/compare block is selected using control bits CCISx1 and CCISx0, as shown in Figure 11–18. The input signal can be read at any time by the software by reading bit CCIx.

Figure 11–18. Capture Logic Input Signal



The capture signal can also be synchronized with the timer clock to avoid race conditions between the timer data and the capture signal. This is illustrated in Figure 11–19. The bit SCSx in capture/compare control register CCTLx selects the capture signal synchronization.

Figure 11–19. Capture Signal

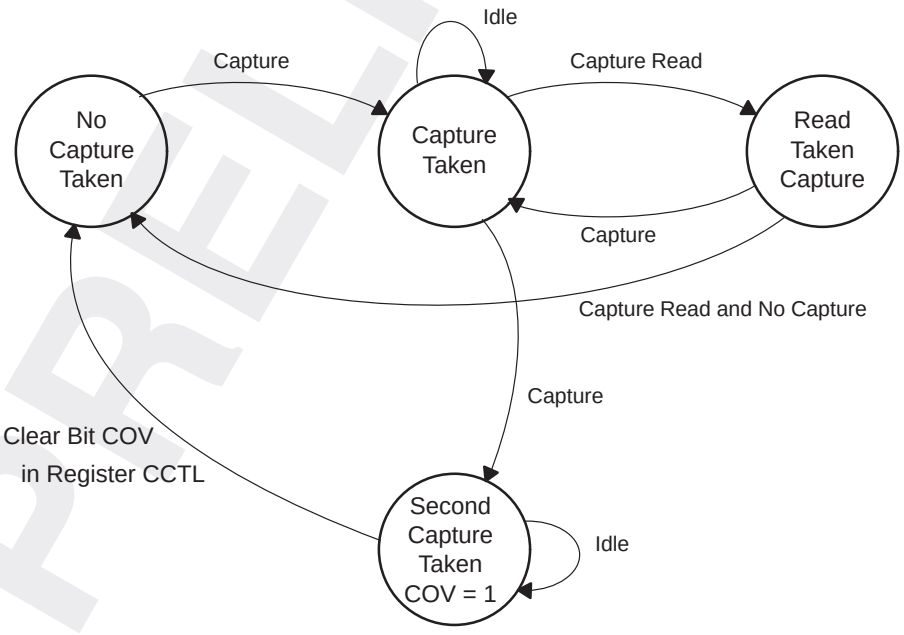


Applications with slow timer clocks can use the nonsynchronized capture signal. In this scenario the software can validate the data and correct it if necessary as shown in the following example:

```
; Software example for the handling of asynchronous
; capture signals
;
; The data of the capture/compare register CCRx are taken
; by the software in the according interrupt routine
; – they are taken only after a CCIFG was set.
; The timer clock is much slower than the system clock
; MCLK.
;
CCRx_Int_hand...                ; Start of interrupt
                                ; handler
            ...
            ...
            CMP   &CCRx,&TBR    ; Test if the data
                                ; CCRX = TBR
            JEQ   Data_Valid
            MOV   &TBR,&CCRx    ; The data in CCRx is
                                ; wrong, use the timer data
Data_Valid  ...   ...           ; The data in CCRx are valid
            ...
            ...
            RETI
;
```

Overflow logic is provided with each capture/compare register to flag the user if a second capture is performed before data from the first capture was read successfully. Bit COVx in register CCTLx is set when this occurs as shown in Figure 11–20.

Figure 11–20. Capture Cycle



Overflow bit COVx is reset by the software as described in the following example:

```
; Software example for the handling of captured data
; looking for overflow condition
;
; The data of the capture/compare register CCRx are taken
; by the software and immediately with the next
; instruction the overflow bit is tested and a decision is
; made to proceed regularly or with an error handler
;
CCRx_Int_hand   ...       ; Start of handler Interrupt
          ...
          ...
                MOV   &CCRx,RAM_Buffer
                BIT   #COV,&CCTLx
                JNZ   Overflow_Hand
          ...
          ...                       ; correct capture data
          ...
                RETI
Overflow_Hand   BIC   #COV,&CCTLx  ; reset capture
                                    ; overflow flag
                                    ; get back to lost
                                    ; synchronization

          ...                       ; Proceed

;               RETI
```

Note: Capture With Timer Halted

The capture should be disabled when the timer is halted. The sequence to follow is: stop the capture, then stop the timer. When the capture function is restarted, the sequence should be: start the capture, then start the timer.

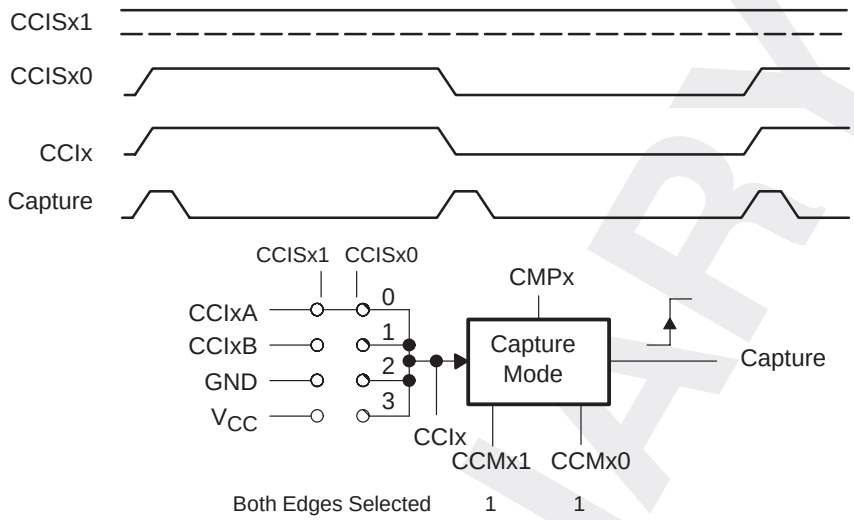
11.4.1.1 Capture/Compare Block, Capture Mode—Capture Initiated by Software

In addition to internal and external signals, captures can be initiated by software. This is useful for various purposes, such as:

- To measure time used by software routines
- To measure time between hardware events
- To measure the system frequency

Two bits, CCISx1 and CCISx0, and the capture mode selected by bits CCMx1 and CCMx0 are used by the software to initiate the capture. The simplest realization is when the capture mode is selected to capture on both edges of CCIx and bit CCISx1 is set. Software then toggles bit CCISx0 to switch the capture signal between V_{CC} and GND, initiating a capture each time the input is toggled, as shown in Figure 11–21.

Figure 11–21. Software Capture Example



The following is a software example of a capture performed by software:

```
; The data of capture/compare register CCRx are taken
; by the software. It is assumed that CCMx1, CCMx0, and
; CCISx1 bits are set. Bit CCIS0 selects the CCIx
; signal to be high or low.
;
;

          ...
          ...
          XOR    #CCISx0, &CCTLx
          ...
          ...
          ...
```

11.4.2 Capture/Compare Block—Compare Mode

The compare mode is selected if the CAPx bit, located in control word CCTLx, is reset. In compare mode all the capture hardware circuitry is inactive and the capture-mode overflow logic is inactive.

The compare mode is most often used to generate interrupts at specific time intervals or used in conjunction with the output unit to generate output signals such as PWM signals.

The compare data is double-buffered. The software writes the compare data to the capture/compare register, but the data is transferred to the compare latch TBCLx to be compared by the compare logic. The transfer of the compare data from the CCRx register to the compare latch is user-selectable to be either immediate or dependent upon a timer event. This double buffering allows the user to update multiple compare values simultaneously. This is useful for example with PWM signals where the period or duty cycle of multiple signals needs to be updated simultaneously. See section 11.4.2.1 for more discussion on how to use and configure the compare latches.

If the timer becomes equal to the value in compare latch TBCLx, then:

- Interrupt flag CCIFGx, located in control word CCTLx, is set.
- An interrupt is requested if interrupt enable bits CCIEx and GIE are set.
- Signal EQUx is output to the output unit. This signal affects the output OUTx, depending on the selected output mode.

The EQU0 signal is true when the timer value is greater or equal to the TBCL0 value. The EQU1 to EQUx signals are true when the timer value is equal to the corresponding TBCL1 to TBCLx values.

11.4.2.1 Capture/Compare Block—Compare Mode—Compare Latch TBCLx

The compare logic uses the data in the compare latch for its comparison with the timer value. The compare data is first written by software to the capture/compare register CCRx and then automatically transferred to the compare latch on a user-selectable load event. The load event is selected with the CLLDx bits in each CCTLx register.

In addition, the compare latches may be grouped together so that each compare latch in a group is updated simultaneously on the load event. All compare latches may be grouped together in a single group, or they may be grouped in groups of two or three compare latches. The grouping is configured with the TBCLGRP bits in the TBCTL register. When using groups, the CLLDx bits of the lowest numbered CCRx register in the group determine the load event for each compare latch of the group except when all 7 compare latches are grouped together (TBCLGRP=3). For example, if a user selects the compare latches to be grouped in groups of three, then there are two groups of three: TBCL1, TBCL2, and TBCL3 form one group, and TBCL4, TBCL5, and TBCL6 form the other group. In this scenario, the CLLDx bits for TBCL1 determine the load event for the first group, and the CLLDx bits for TBCL4 determine the load event for the second group. The CLLDx bits in CCTL2, CCTL3, CCTL5, and CCTL6 are unused. When all compare latches are grouped together (TBCLGRP=3), then the CLLDx bits in TBCL1 determine the load event.

When using groups, two conditions must exist for the compare latches to be loaded. First, all CCRx registers of the group must be updated (except when using immediate mode); second, the load event must occur. This means that if a user intends to retain any CCRx register data of a group when updating the group, the old data must be written to the CCRx register again. Otherwise, the compare latches will not be updated.

The CLLDx bits in the applicable CCTLx register select the load event. There are four choices for the load event:

- Immediate
- When TBR *counts* to 0
- Continuous mode or up mode – when TBR *counts* to 0
  Up/down mode – when TBR *counts* to TBCL0 or *counts* to 0
- When TBR *counts* to TBCLx

The groupings and load conditions are summarized below in Table 11–2

Table 11–2. Shadow Register Operating Modes

TBCLGRP	CLLDx From Lowest CCTLx in Group (see Note 1)	Counter Mode MCx	Load Conditions						
			0	1	2	3	4	5	6
0	0	0–3	Immediate	Immediate	Immediate	Immediate	Immediate	Immediate	Immediate
	1	1–3	TBR counts to 0	TBR counts to 0	TBR counts to 0	TBR counts to 0	TBR counts to 0	TBR counts to 0	TBR counts to 0
	2	1,2	TBR counts to 0	TBR counts to 0	TBR counts to 0	TBR counts to 0	TBR counts to 0	TBR counts to 0	TBR counts to 0
		3	TBR counts to 0 or to TBCL0	TBR counts to 0 or to TBCL0	TBR counts to 0 or to TBCL0	TBR counts to 0 or to TBCL0	TBR counts to 0 or to TBCL0	TBR counts to 0 or to TBCL0	TBR counts to 0 or to TBCL0
	3	1–3	TBR counts to TBCL0	TBR counts to TBCL1	TBR counts to TBCL2	TBR counts to TBCL3	TBR counts to TBCL4	TBR counts to TBCL5	TBR counts to TBCL6
1	0	0–3	Immediate	TBCL1, TBCL2 loaded immediately when the corresponding CCRx register is loaded		TBCL3, TBCL4 loaded immediately when the corresponding CCRx register is loaded		TBCL5, TBCL6 loaded immediately when the corresponding CCRx register is loaded	
	1	1–3	TBR counts to 0	TBCL1, TBCL2 updated simultaneously when TBR counts to 0		TBCL3, TBCL4 updated simultaneously when TBR counts to 0		TBCL5, TBCL6 updated simultaneously when TBR counts to 0	
	2	1,2	TBR counts to 0	TBCL1, TBCL2 updated simultaneously when TBR counts to 0		TBCL3, TBCL4 updated simultaneously when TBR counts to 0		TBCL5, TBCL6 updated simultaneously when TBR counts to 0	
		3	TBR counts to 0 or to TBCL0	TBCL1, TBCL2 updated simultaneously when TBR counts to 0 or to TBCL0		TBCL3, TBCL4 updated simultaneously when TBR counts to 0 or to TBCL0		TBCL5, TBCL6 updated simultaneously when TBR counts to 0 or to TBCL0	
	3	1–3	TBR counts to TBCL0	TBR counts to TBCL1	TBR counts to TBCL2	TBR counts to TBCL3	TBR counts to TBCL4	TBR counts to TBCL5	TBR counts to TBCL6

Table 11–2. Shadow Register Operating Modes (Continued)

TBCLGRP	CLLDx From Lowest CCTLx in Group (see Note 1)	Counter Mode MCx	Load Conditions						
			0	1	2	3	4	5	6
2	0	0–3	Immediate	TBCL1, TBCL2, TBCL3 loaded immediately when the corresponding CCRx register is loaded			TBCL4, TBCL5, TBCL6 loaded immediately when the corresponding CCRx register is loaded		
	1	1–3	TBR counts to 0	TBCL1, TBCL2, TBCL3 updated simultaneously when TBR counts to 0			TBCL4, TBCL5, TBCL6 updated simultaneously when TBR counts to 0		
	2	1,2	TBR counts to 0	TBCL1, TBCL2, TBCL3 updated simultaneously when TBR counts to 0			TBCL4, TBCL5, TBCL6 updated simultaneously when TBR counts to 0		
		3	TBR counts to 0 or to TBCL0	TBCL1, TBCL2, TBCL3 updated simultaneously when TBR counts to 0 or to TBCL0			TBCL4, TBCL5, TBCL6 updated simultaneously when TBR counts to 0 or to TBCL0		
	3	1–3	TBR counts to TBCL0	TBR counts to TBCL1	TBR counts to TBCL2	TBR counts to TBCL3	TBR counts to TBCL4	TBR counts to TBCL5	TBR counts to TBCL6
3	0	0–3	TBCL0, TBCL1, TBCL2, TBCL3, TBCL4, TBCL5, TBCL6 Loaded immediately when the corresponging CCRx register is loaded.						
	1	1–3	TBCL0,TBCL1, TBCL2, TBCL3, TBCL4, TBCL5, TBCL6 updated simultaneously when TBR counts to 0						
	2	1,2	TBCL0,TBCL1, TBCL2, TBCL3, TBCL4, TBCL5, TBCL6 updated simultaneously when TBR counts to 0						
		3	TBCL0,TBCL1, TBCL2, TBCL3, TBCL4, TBCL5, TBCL6 updated simultaneously when TBR counts to 0 or to TBCL0						
	3	1–3	TBR counts to TBCL0	TBR counts to TBCL1	TBR counts to TBCL2	TBR counts to TBCL3	TBR counts to TBCL4	TBR counts to TBCL5	TBR counts to TBCL6

Notes: 1) When using groups, load mode for the group is selected with the CLLDx bits of the lowest numbered CCTLx register in the group (except when TBCLGRP=3). For example, when grouped by 2, the CLLDx bits of CCTL3 determine the load mode for TBCL3 and TBCL4. When grouped by 3, the CLLDx bits of CCTL4 determine the load mode for TBCL4, TBCL5, and TBCL6, etc. When TBCLGRP=3 the CLLDx bits from TBCTL1 are used.

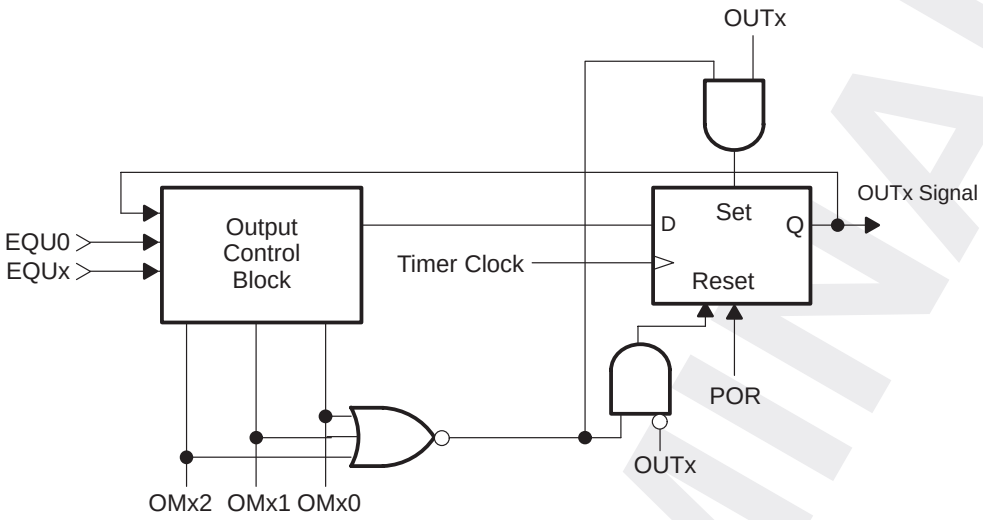
2) When using groups, all CCRx registers must be updated with new data before the load will take place (except when using immediate mode), even if new data = old data. When using immediate mode, each compare latch is updated immediately when its corresponding CCRx register is updated.

3) When using groups, different load modes may be selected for each group. For example, when grouped by 3, immediate mode may be selected (via CLLDx bits in CCTL1) for TBCL1, TBCL2, and TBCL3, and mode 2 may be selected (via CLLDx bits in CCTL4) for TBCL4, TBCL5, and TBCL6.

11.5 The Output Unit

Each capture/compare block contains an output unit shown in Figure 11–22. The output unit is used to generate output signals such as PWM signals. Each output unit has 8 operating modes that can generate a variety of signals based on the EQU0 and EQUx signals. The output mode is selected with the OMx bits located in the CCTLx register.

Figure 11–22. Output Unit



OMx2	OMx1	OMx0	
0	0	0	Output mode: OUTx signal reflects the value of the OUTx bit
0	0	1	Set mode: OUT x signal reflects the value of signal EQUx
0	1	0	PWM toggle/reset: EQUx toggles OUTx. EQU0 resets OUTx.
0	1	1	PWM set/reset: EQUx sets OUTx. EQU0 resets OUTx
1	0	0	Toggle: EQUx toggles OUTx signal.
1	0	1	Reset: EQUx resets OUTx.
1	1	0	PWM toggle/set: EQUx toggles OUTx. EQU0 sets OUTx.
1	1	1	PWM reset/set: EQUx resets OUTx. EQUx sets OUTx.

Note: OUTx signal updates with rising edge of timer clock for all modes except mode 0.
Modes 2, 3, 6, 7 not useful for output unit 0.

11.5.1 Output Unit – Output Modes

The output modes are defined by the OMx bits and are discussed below. The OUTx signal is changed with the rising edge of the timer clock for all modes except mode 0. Output modes 2, 3, 6, and 7 are not useful for output unit 0.

Output mode 0: Output mode:
The output signal OUTx is defined by the OUTx bit in control register CCTLx. The OUTx signal updates immediately upon completion of writing the bit information.

Output mode 1: Set mode:
The output is set when the timer value becomes equal to compare data TBCLx. It remains set until a reset of the timer, or until another output mode is selected and controls the output.

Output mode 2: PWM toggle/reset mode:
The output is toggled when the timer value becomes equal to compare data TBCLx. It is reset when the timer value becomes equal to TBCL0.

Output mode 3: PWM set/reset mode:
The output is set when the timer value becomes equal to compare data TBCLx. It is reset when the timer value becomes equal to TBCL0.

Output mode 4: Toggle mode:
The output is toggled when the timer value becomes equal to compare data TBCLx. The output period is double the timer period.

Output mode 5: Reset mode:
The output is reset when the timer value becomes equal to compare data TBCLx. It remains reset until another output mode is selected and controls the output.

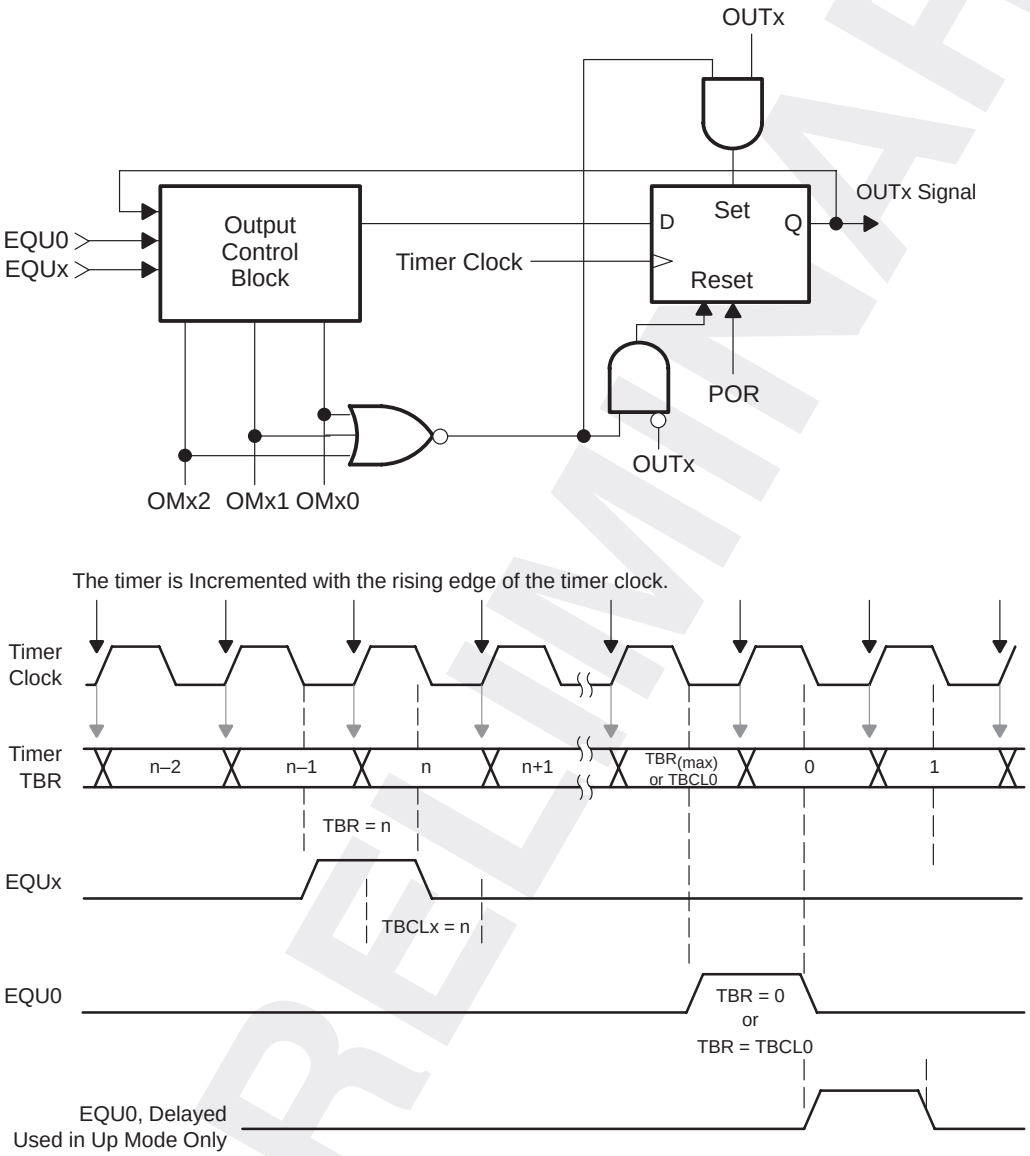
Output mode 6: PWM toggle/set mode:
The output is toggled when the timer value becomes equal to compare data TBCLx. It is set when the timer value becomes equal to TBCL0.

Output mode 7: PWM toggle/set mode:
The output is reset when the timer value becomes equal to compare data TBCLx. It is set when the timer value becomes equal to TBCL0.

11.5.2 Output Control Block

The output control block prepares the value of the OUTx signal, which is latched into the OUTx flip-flop with the next positive timer clock edge, as shown in Figure 11–23 and Table 11–3. The equal signals EQUx and EQU0 are sampled during the negative level of the timer clock, as shown in Figure 11–23.

Figure 11–23. Output Control Block



EQU0 delayed is used in up mode, not EQU0. EQU0 is active high when TBR = TBCL0. EQU0 delayed is active high when TBR = 0.

Table 11–3. State of OUTx at Next Rising Edge of Timer Clock

Mode	EQU0	EQUx	D
0	x	x	x(OUTx bit)
1	x	0	OUTx (no change)
	x	1	1 (set)
2	0	0	$\overline{\text{OUTx}}$ (no change)
	0	1	OUTx (toggle)
	1	0	0 (reset)
	1	1	1 (set)
3	0	0	OUTx (no change)
	0	1	1 (set)
	1	0	0 (reset)
	1	1	1 (set)
4	x	0	$\overline{\text{OUTx}}$ (no change)
	x	1	OUTx (toggle)
5	x	0	OUTx (no change)
	x	1	0 (reset)
6	0	0	$\overline{\text{OUTx}}$ (no change)
	0	1	OUTx (toggle)
	1	0	1 (set)
	1	1	0 (reset)
7	0	0	OUTx (no change)
	0	1	0 (reset)
	1	0	1 (set)
	1	1	0 (reset)

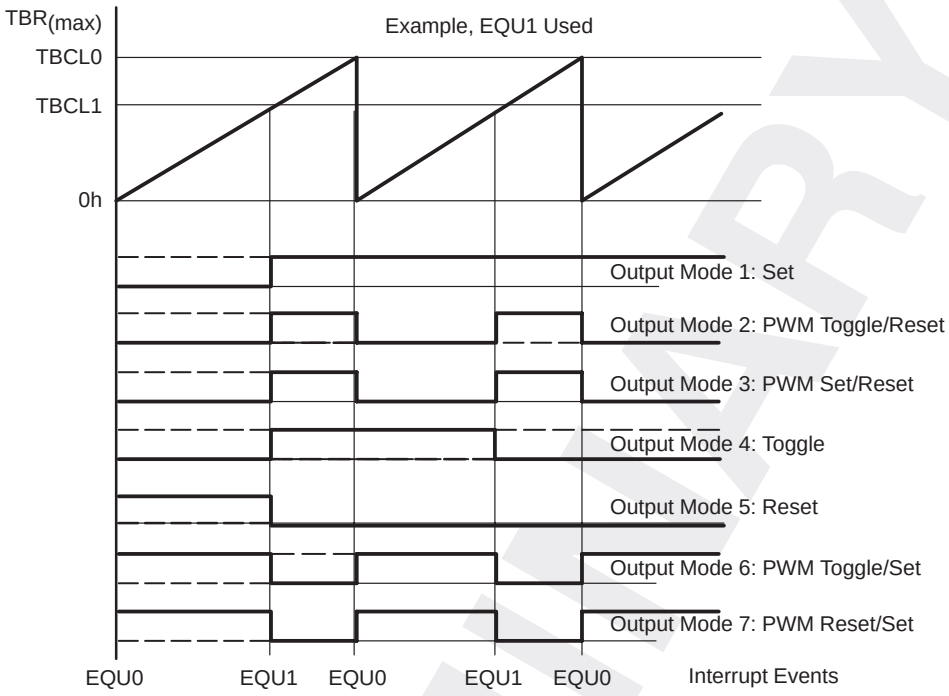
11.5.3 Output Examples

The following are some examples of possible output signals using the various timer and output modes.

11.5.3.1 Output Examples—Timer in Up Mode

The OUTx signal is changed when the timer *counts* up to the TBCLx value, and rolls from TBCL0 to zero, depending on the output mode, as shown in Figure 11–24.

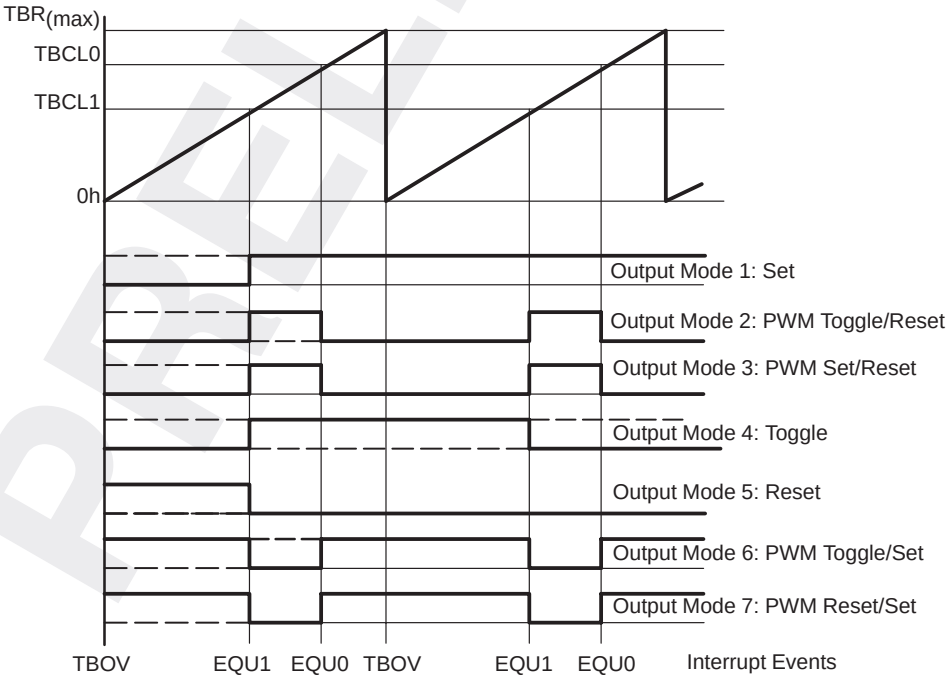
Figure 11–24. Output Examples—Timer in Up Mode



11.5.3.2 Output Examples—Timer in Continuous Mode

The OUTx signal is changed when the timer reaches the TBCLx and TBCL0 values, depending on the output mode, as shown in Figure 11–25.

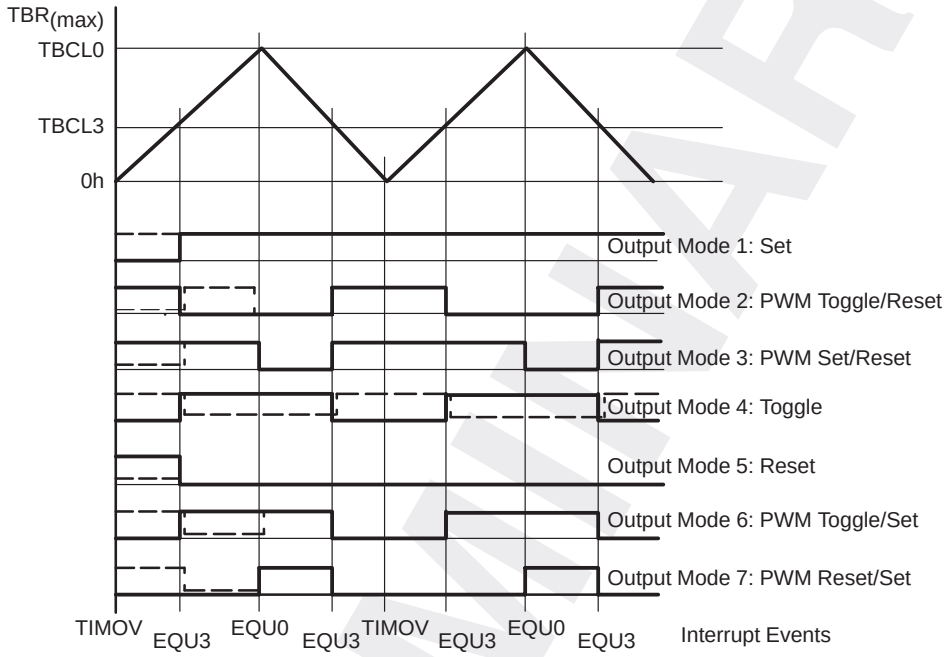
Figure 11–25. Output Examples—Timer in Continuous Mode



11.5.3.3 Output Examples – Timer in Up/Down Mode

The OUTx signal changes when the timer equals TBCLx in either count direction and when the timer equals TBCL0, depending on the output mode, as shown in Figure 11–26.

Figure 11–26. Output Examples – Timer in Up/Down Mode (I)



11.6 Timer_B Registers

The Timer_B registers, described in Table 11–4, are word structured and must be accessed using word instructions.

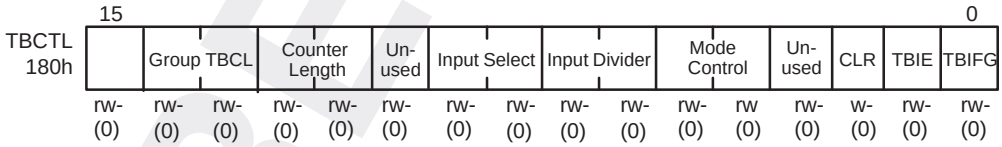
Table 11–4. Timer_B Registers

Register	Short Form	Register Type	Address	Initial State
Timer_B control	TBCTL	Read/write	180h	POR reset
Timer_B register	TBR	Read/write	190h	POR reset
Cap/com control 0	CCTL0	Read/write	182h	POR reset
Capture/compare 0	CCR0	Read/write	192h	POR reset
Cap/com control 1	CCTL1	Read/write	184h	POR reset
Capture/compare 1	CCR1	Read/write	194h	POR reset
Cap/com control 2	CCTL 2	Read/write	186h	POR reset
Capture/compare 2	CCR2	Read/write	196h	POR reset
Cap/com control 3	CCTL3	Read/write	188h	POR reset
Capture/compare 3	CCR3	Read/write	198h	POR reset
Cap/com control 4	CCTL4	Read/write	18Ah	POR reset
Capture/compare 4	CCR4	Read/write	19Ah	POR reset
Capture/compare 5	CCTL5	Read/write	18Ch	POR reset
Capture/compare 5	CCR5	Read/write	19Ch	POR reset
Capture/compare 6	CCTL6	Read/write	18Eh	POR reset
Capture/compare 6	CCR6	Read/write	19Eh	POR reset
Interrupt vector	TBIV	Read	11Eh	(POR reset)

11.6.1 Timer_B Control Register TBCTL

The timer and timer operation control bits are located in the timer control register (TBCTL) shown in Figure 11–27. All control bits are reset automatically by the POR signal, but are not affected by the PUC signal. The control register must be accessed using word instructions.

Figure 11–27. Timer_B Control Register TBCTL



Bit 0: TBIFG: This flag indicates a timer overflow event.

Up mode: TBIFG is set if the timer *counts* from TBCL0 value to 0000h.

Continuous mode: TBIFG is set if the timer *counts* from $TBR_{(max)}$ to 0000h.

Up/down mode: TBIFG is set if the timer *counts* down from 0001h to 0000h.

Bit 1: Timer overflow interrupt enable (TBIE) bit. An interrupt request from the timer overflow bit is enabled if this bit is set, and is disabled if reset.

Bit 2: Timer clear (CLR) bit. The timer and input divider are reset with the POR signal, or if bit CLR is set. The CLR bit is automatically reset and is always read as zero. The timer starts in the upward direction with the next valid clock edge, unless halted by cleared mode control bits.

Bit 3: Not used

Bits 4, 5: Mode control: Table 11–5 describes the mode control bits.

Table 11–5. Mode Control

MC1	MC0	Count Mode	Description
0	0	Stop	Timer is halted.
0	1	Up to CCR0	Timer counts up to TBCL0 and restarts at 0. Note: If TBCL0 > $TBR_{(max)}$, the counter counts to zero with the next rising edge of timer clock.
1	0	Continuous up	Timer counts up to $TBR_{(max)}$ and restarts at 0. The maximum value of TBR [$TBR_{(max)}$] is: 0FFFFh for 16-bit configuration 00FFFh for 12-bit configuration 003FFh for 10-bit configuration 000FFh for 8-bit configuration
1	1	Up/down	Timer continuously counts up to CCR0 and back down to 0. Note: If CCR0 > $TBR_{(max)}$, the counter operates as if it were configured for continuous mode. It will not count down from $TBR_{(max)}$ to zero.

Bits 6, 7: Input divider control bits. Table 11–6 describes the clock-divider bits.

Table 11–6. Input Clock Divider Control Bits

ID1	ID0	Operation	Description
0	0	/1	Input clock source is passed to the timer.
0	1	/2	Input clock source is divided by two.
1	0	/4	Input clock source is divided by four.
1	1	/8	Input clock source is divided by eight.

Bits 8, 9: Clock source selection bits. Table 11–7 describes the clock source selections.

Table 11–7. Clock Source Selection

SSEL1	SSEL0	O/P Signal	Comment
0	0	TBCLK	See data sheet device description
0	1	ACLK	Auxiliary clock ACLK is used
1	0	SMCLK	System clock SMCLK
1	1	INCLK	See device description in data sheet

Bit 10: Unused

Bits 11, 12: Configure 16-bit timer (TBR) for 8-bit, 10-bit, 12-bit, or 16-bit operation

CNTL = 0: 16-bit length, $TBR_{(max)}$ is 0FFFFH
CNTL = 1: 12-bit length, $TBR_{(max)}$ is 0FFFH
CNTL = 2: 10-bit length, $TBR_{(max)}$ is 03FFH
CNTL = 3: 8-bit length, $TBR_{(max)}$ is 0FFH

Bits 13, 14: Load compare latches, individually or in groups. The load signal is controlled via the CLLDx bits located in the appropriate capture/compare control register CCTLx.

TBCLGRP = 0: load individually
Load of the shadow registers is defined in each individual CCTLx register by bits CLLDx. The CLLD bits in each CCTLx register define the operating mode for the shadow registers.

TBCLGRP = 1: Three groups are selected (TBCL1 + TBCL2, TBCL3 + TBCL4, TBCL5 + TBCL6):
TBCL1 + TBCL2: The CLLD bits in CCTL1 define the operating mode.
TBCL3 + TBCL4: The CLLD bits in CCTL3 define the operating mode.
TBCL5 + TBCL6: The CLLD bits in CCTL5 define the operating mode.

TBCLGRP = 2: Two groups are selected (TBCL1 + TBCL2 + TBCL3, TBCL4 + TBCL5 + TBCL6):
TBCL1 + TBCL2 + TBCL3: The CLLD bits in CCTL1 define the operating mode.
TBCL4 + TBCL5 + TBCL6: The CLLD bits in CCTL4 define the operating mode.

TBCLGRP = 3: One group is selected (all TBCLx registers):
The CLLD bits in CCTL1 define the operating mode for all shadow registers.

Bit 15: Unused

Note: Changing Timer_B Control Bits

If the timer operation is modified by the control bits in the TBCTL register, the timer should be halted during this modification. Critical modifications are the input select bits, input divider bits, and the timer clear bit. Asynchronous clocks, input clock, and system clock can result in race conditions where the timer reacts unpredictably.

The recommended instruction flow is:

1) Modify the control register and stop the timer with one instruction.

2) Start the timer operation.

   For example:

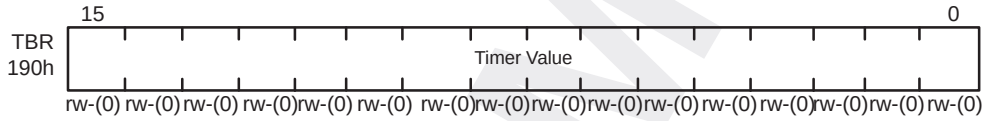
   MOV #0286h,&TBCTL ; ACLK/8, timer stopped, timer cleared

   BIS #10h,&TBCTL ; Start timer with up mode

11.6.2 Timer_B Register TBR

The TBR register is the value of the timer.

Figure 11–28. TBR Register



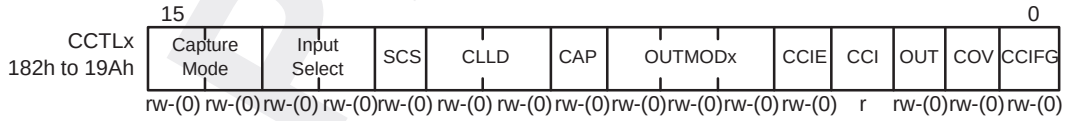
Note: Modifying Timer_B Register TBR

When ACLK, SMCLK, or the external clock TBCLK or INCLK is selected for the timer clock, any write to timer register TBR should occur while the timer is not operating; otherwise, the results may be unpredictable. In this case, the timer clock is asynchronous to the CPU clock MCLK and critical race conditions exist.

11.6.3 Capture/Compare Control Register CCTLx

Each capture/compare block has its own control word CCTLx, shown in Figure 11–29. The POR signal resets all bits of CCTLx; the PUC signal does not affect these bits.

Figure 11–29. Capture/Compare Control Register CCTLx



Bit 0: Capture/compare interrupt flag CCIFGx

Capture mode:

If set, it indicates that a timer value was captured in the CCRx register.

Compare mode:

If set, it indicates that a timer value was equal to the data in the TBCLx latch.

CCIFG0 flag:

CCIFG0 is automatically reset when the interrupt request is accepted.

CCIFG1 to CCIFGx flags:

The flag that caused the interrupt is automatically reset after the TBIV word is accessed. If the TBIV register is not accessed, the flags must be reset with software.

No interrupt is generated if the corresponding interrupt enable bit is reset, but the flag will be set. In this scenario, the flag must be reset by the software.

Setting the CCIFGx flag with software will request an interrupt if the interrupt-enable bit is set.

Bit 1: Capture overflow flag COV

Compare mode selected, CAP = 0:

Capture signal generation is reset. No compare event will set COV bit.

Capture mode selected, CAP = 1:

The overflow flag COV is set if a second capture is performed before the first capture value is read. The overflow flag must be reset with software. It is not reset by reading the capture value.

Bit 2: The OUTx bit determines the value of the OUTx signal if the output mode is 0.

Bit 3: Capture/compare input signal CCIx:

The selected input signal (CCIxA, CCIxB, V_{CC}. or GND) can be read by this bit. See Figure 11–18.

Bit 4: Interrupt enable CCIEx: Enables or disables the interrupt request signal of capture/compare block x. Note that the GIE bit must also be set to enable the interrupt.

0: Interrupt disabled

1: Interrupt enabled

Bits 5 to 7: Output mode select bits:

Table 11–7 describes the output mode selections.

Table 11–8. Capture/Compare Control Register Output Mode

Bit Value	Output Mode	Description
0	Output only	The OUTx signal reflects the value of the OUTx bit
1	Set	EQUx sets OUTx
2	PWM toggle/reset	EQUx toggles OUTx. EQU0 resets OUTx.
3	PWM set/reset	EQUx sets OUTx. EQU0 resets OUTx
4	Toggle	EQUx toggles OUTx signal.
5	Reset	EQUx resets OUTx.
6	PWM toggle/set	EQUx toggles OUTx. EQU0 sets OUTx.
7	PWM reset/set	EQUx resets OUTx. EQUx sets OUTx.

Note: OUTx updates with rising edge of timer clock for all modes except mode 0. Modes 2, 3, 6, 7 not useful for output unit 0.

Bit 8: CAP sets capture or compare mode.
0: Compare mode
1: Capture mode

Bits 9, 10: CLLD: Select load source for compare latch TBCLx
(also see description of bits TBCLGRP, 13 and 14, in TBCTL)
CLLD = 0: Immediate
CLLD = 1: Load CCRx data to TBCLx when TBR counts to 0
CLLD = 2: UP/DOWN mode: load CCRx data to TBCLx when TBR *counts* to TBCL0 or to 0
Continuous mode or UP-mode: load CCRx data to TBCLx when TBR counts to 0
CLLD = 3: CCRx data are loaded to TBCLx when TBR *counts* to TBCLx

Bit 11: SCSx bit:
This bit is used to synchronize the capture input signal with the timer clock.
0: asynchronous capture
1: synchronous capture

Bits 12, 13: Input select, CCIS0 and CCIS1:
These two bits define the capture signal source. These bits are not used in compare mode.
0 Input CCIxA is selected
1 Input CCIxB is selected
2 GND
3 V_{CC}

Bits 14, 15: Capture mode bits:
Table 11–8 describes the capture mode selections.

Table 11–9. Capture/Compare Control Register Capture Mode

Bit Value	Capture Mode	Description
0	Disabled	The capture mode is disabled.
1	Pos. Edge	Capture is done with rising edge.
2	Neg. Edge	Capture is done with falling edge.
3	Both Edges	Capture is done with both rising and falling edges.

Note: Simultaneous Capture and Capture Mode Selection

Captures must not be performed simultaneously with switching from compare to capture mode. Otherwise, the result in the capture/compare register will be unpredictable.

The recommended instruction flow is:

1) Modify the control register to switch from compare to capture.

2) Capture

For example:

```
BIS  #CAP,&CCTL2        ; Select capture with register CCR2
XOR #CCIS1,&CCTL2       ; Software capture:     CCIS0 = 0
                        ;                       Capture mode = 3
```

11.6.4 Timer_B Interrupt Vector Register

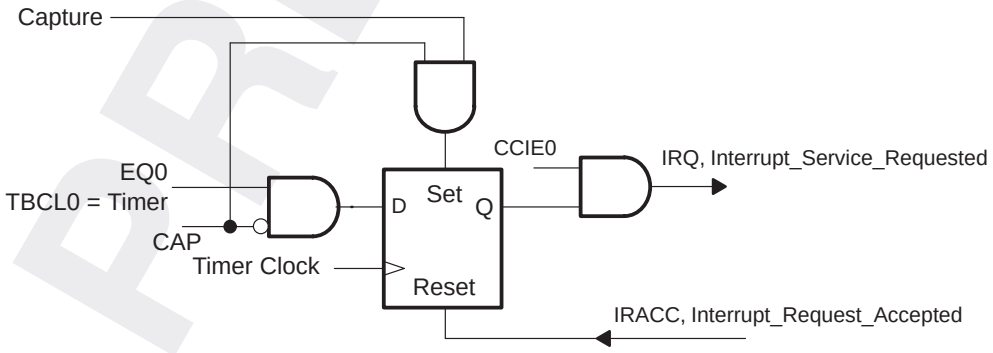
Two interrupt vectors are associated with the 16-bit Timer_B module:

- CCR0 interrupt vector (highest priority)
- TBIV interrupt vector for flags CCIFG1–CCIFGx and TBIFG.

11.6.4.1 CCR0 Interrupt Vector

The interrupt flag associated with capture/compare register CCR0, as shown in Figure 11–30, is set if the timer value is equal to the compare register value.

Figure 11–30. Capture/Compare Interrupt Flag



Capture/compare register 0 has the highest Timer_B interrupt priority, and uses its own interrupt vector.

11.6.4.2 Vector Word, TBIFG, CCIFG1 to CCIFGx Flags

The CCIFGx (other than CCIFG0) and TBIFG interrupt flags are prioritized and combined to source a single interrupt as shown in Figure 11–31. The interrupt vector register TBIV (shown in Figure 11–32) is used to determine which flag requested an interrupt.

Figure 11–31. Schematic of Capture/Compare Interrupt Vector Word

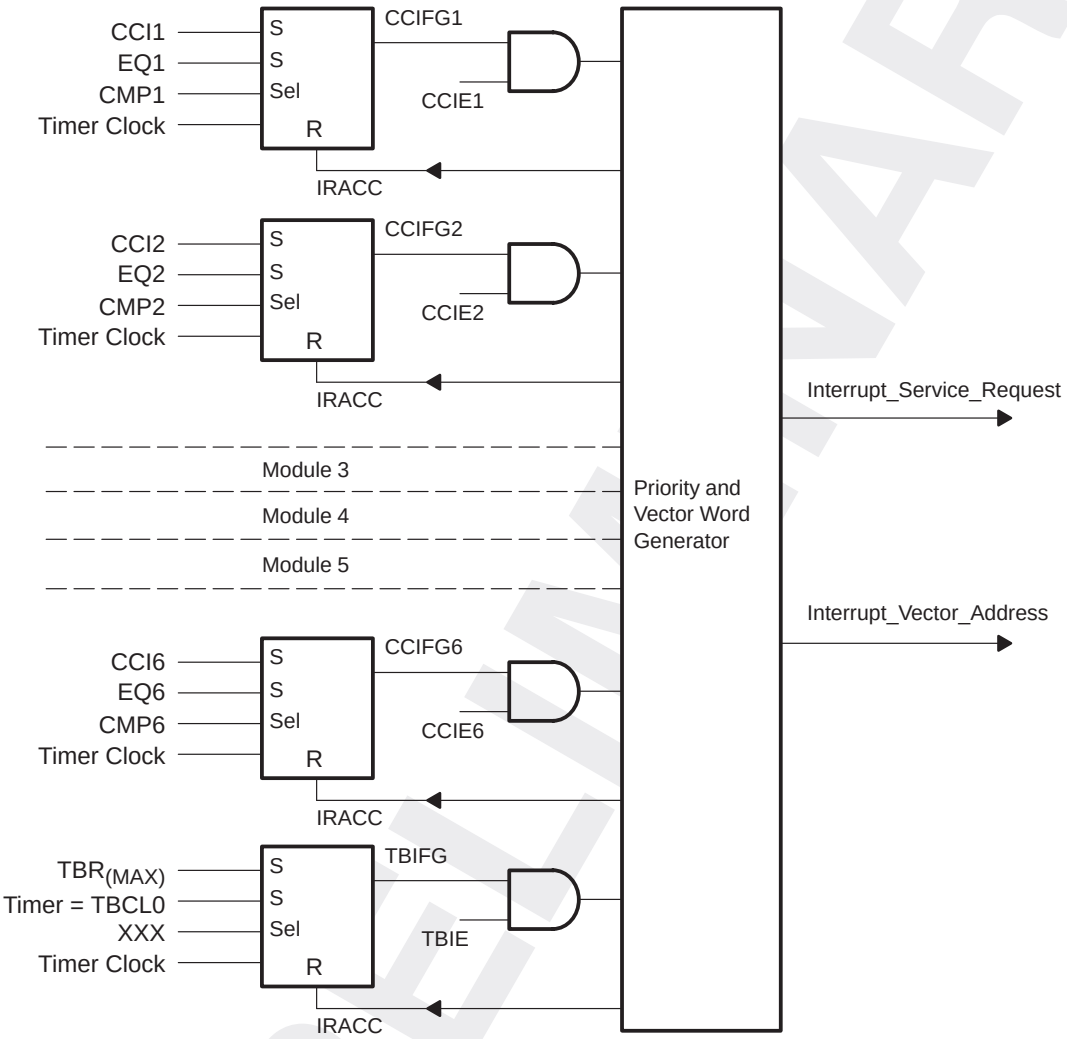
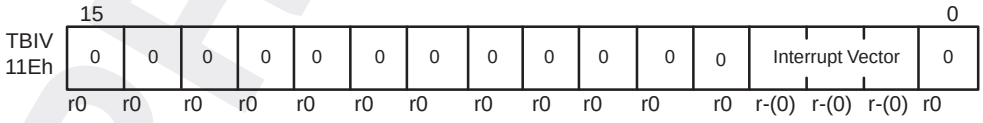


Figure 11–32. Vector Word Register



The flag with the highest priority generates a number from 2 to 14 in the TBIV register as shown in Table 11–9. (If the value of the TBIV register is 0, no interrupt is pending.) This number can be added to the program counter to automatically enter the appropriate software routine without the need for reading and evaluating the interrupt vector. The software example in section 11.6.4.3 shows this technique.

Table 11–10. Vector Register TBIV Description

Interrupt Priority	Interrupt Source	Short Form	Vector Register TBIV Contents
Highest†	Capture/compare 1	CCIFG1	2
	Capture/compare 2	CCIFG2	4
	Capture/compare 3‡	CCIFG3	6
	Capture/compare 4‡	CCIFG4	8
	Capture/compare 5‡	CCIFG5	10
	Capture/compare 6‡	CCIFG6	12
Lowest	Timer overflow	TBIFG	14
	No interrupt pending		0

† Highest pending interrupt other than CCIFG0. CCIFG0 is always the highest priority Timer_B interrupt.
‡ 14x devices only

Accessing the TBIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, then another interrupt will be immediately generated after servicing the initial interrupt. For example, if both CCIFG2 and CCIFG3 are set, when the interrupt service routine accesses the TBIV register (either by reading it or by adding it directly to the PC), CCIFG2 will be reset automatically. After the RETI instruction of the interrupt service routine is executed, the CCIFG3 flag will generate another interrupt.

Note: Writing to Read-Only Register TBIV

Register TBIV should not be written to. If a write operation to TBIV is performed, the interrupt flag of the highest-pending interrupt is reset. Therefore, the requesting interrupt event is missed. Additionally, writing to this read-only register results in increased current consumption as long as the write operation is active.

11.6.4.3 Timer Interrupt Vector Register, Software Example, Timer_B7

The following software example describes the use of vector word TBIV of Timer_B3 and the handling overhead. The numbers at the right margin show the necessary cycles for every instruction. The example is written for continuous mode: the time difference to the next interrupt is added to the corresponding compare register.

```
; Software example for the interrupt part          Cycles
;
; Interrupt handler for Capture/Compare Module 0.
; The interrupt flag CCIFG0 is reset automatically
;
TIMMOD0   ...       ; Start of handler Interrupt latency  6
          RETI                                            5
;
; Interrupt handler for Capture/Compare Modules 1 to 6.
; The interrupt flags CCIFGx and TBIFG are reset by
; hardware. Only the flag with the highest priority
; responsible for the interrupt vector word is reset.
TIM_HND   $                   ; Interrupt latency         6
          ADD    &TBIV,PC     ; Add offset to Jump table  3
          RETI                ; Vector  0: No interrupt   5
          JMP    TIMMOD1      ; Vector  2: Module 1       2
          JMP    TIMMOD2      ; Vector  4: Module 2       2
          JMP    TIMMOD3      ; Vector  6: Module 3       2
          JMP    TIMMOD4      ; Vector  8: Module 4       2
          JMP    TIMMOD5      ; Vector 10: Module 5       2
          JMP    TIMMOD6      ; Vector 12: Module 6       2
;
; Module 7. Timer Overflow Handler: the Timer Register is
; expanded into the RAM location TIMEXT (MSBs)
;
TIMOVH                        ; Vector 14: TIMOV Flag
          INC    TIMEXT       ; Handle Timer Overflow     4
          RETI                                            5
;
TIMMOD2                       ; Vector 4: Module 2
          ADD    #NN,&CCR2    ; Add time difference       5
          ...                 ; Task starts here
          RETI                ; Back to main program      5
;
TIMMOD1                     ; Vector 2: Module 1
          ADD    #MM,&CCR1  ; Add time difference          5
          ...               ; Task starts here
          RETI              ; Back to main program        5

; The Module 3 handler shows a way to look if any other
; interrupt is pending: 5 cycles have to be spent, but
; 9 cycles may be saved if another interrupt is pending
;
TIMMOD3                     ; Vector 6: Module 3
          ADD    #PP,&CCR3  ; Add time difference          5
          ...               ; Task starts here
          JMP    TIM_HND    ; Look for pending interrupts  2
;
          .SECT "VECTORS",0FFF0h   ; Interrupt Vectors
; The vector address may be different for different devices.
;
          .WORD  TIM_HND   ; Vector for Capture/Compare
                           ; Module 1..6 and timer overflow
                           ; TBIFG
          .WORD  TIMMOD0   ; Vector for Capture/Compare
                           ; Module 0
```

11.6.4.4 Timer Interrupt Vector Register, Software Example, Timer_B3

The following software example describes the use of vector word TBIV of Timer_B3 and the handling overhead. The numbers at the right margin show the necessary cycles for every instruction. The example is written for continuous mode: the time difference to the next interrupt is added to the corresponding compare register.

```
; Software example for the interrupt part                Cycles
;
; Interrupt handler for Capture/Compare Module 0.
; The interrupt flag CCIFG0 is reset automatically
;
TIMMOD0   ...       ; Start of handler Interrupt latency  6
          RETI                                            5
;
; Interrupt handler for Capture/Compare Modules 1 to 6.
; The interrupt flags CCIFGx and TBIFG are reset by
; hardware. Only the flag with the highest priority
; responsible for the interrupt vector word is reset.
TIM_HND   $                 ; Interrupt latency           6
          ADD    &TBIV,PC   ; Add offset to Jump table    3
          RETI              ; Vector  0: No interrupt     5
          JMP    TIMMOD1    ; Vector  2: Module 1         2
          JMP    TIMMOD2    ; Vector  4: Module 2         2
          RETI              ; Vector  6
          RETI              ; Vector  8
          RETI              ; Vector 10
          RETI              ; Vector 12
;
; Timer Overflow Handler: the Timer Register is expanded
; into the RAM location TIMEXT (MSBs)
;
TIMOVH                      ; Vector 14: TIMOV Flag
          INC    TIMEXT     ; Handle Timer Overflow       4
          RETI                                            5
;
TIMMOD2                     ; Vector 4: Module 2
          ADD    #NN,&CCR2  ; Add time difference         5
          ...               ; Task starts here
          RETI              ; Back to main program        5

; The Module 1 handler shows a way to look if any other
; interrupt is pending: 5 cycles have to be spent, but
; 9 cycles may be saved if another interrupt is pending
;
TIMMOD1                   ; Vector 6: Module 3
          ADD    #PP,&CCR1; Add time difference           5
          ...             ; Task starts here
          JMP    TIM_HND  ; Look for pending interrupts   2
;
          .SECT "VECTORS",0FFF0h   ; Interrupt Vectors
; The vector address may be different for different devices.
;
          .WORD  TIM_HND   ; Vector for Capture/Compare
                           ; Module 1..6 and timer overflow
                           ; TBIFG
          .WORD  TIMMOD0   ; Vector for Capture/Compare
                           ; Module 0
```

If the FLL (on applicable devices) is turned off, then two additional cycles need to be added for a synchronous start of the CPU and system clock MCLK.

If the CPU clock MCLK was turned off in devices with the Basic Clock Module (CPUOFF=1), then 2 or 3 additional cycles need to be added for synchronous start of the CPU. The delta of one clock cycle is caused when clocks are asynchronous to the restart of CPU clock MCLK.

The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles (but not the task handling itself), as described:

- ❑ Capture/compare block CCR0 — 11 cycles
- ❑ Capture/compare blocks CCR1 to CCR6 — 16 cycles
- ❑ Timer overflow TBIFG — 14 cycles

11.6.4.5 Timing Limits

With the TBIV register and the previous software, the shortest repetitive time distance t_{CRmin} between two events using a compare register is:

$$t_{CRmin} = t_{taskmax} + 16 \times t_{cycle}$$

With: $t_{taskmax}$ Maximum (worst case) time to perform the task during the interrupt routine (for example, incrementing a counter)

t_{cycle} Cycle time of the system frequency MCLK

The shortest repetitive time distance t_{CLmin} between two events using a capture register is:

$$t_{CLmin} = t_{taskmax} + 16 \times t_{cycle}$$

Chapter 12

USART Peripheral Interface, UART Mode

The universal synchronous/asynchronous receive/transmit (USART) serial-communication peripheral supports two serial modes with one hardware configuration. These modes shift a serial bit stream in and out of the MSP430 at a programmed rate or at a rate defined by an external clock. The first mode is the universal asynchronous receive/transmit (UART) communication protocol; the second is the serial peripheral interface (SPI) protocol (discussed in Chapter 13).

Bit SYNC in control register UCTL selects the required mode:

SYNC = 0:	UART – asynchronous mode selected
SYNC = 1:	SPI – synchronous mode selected

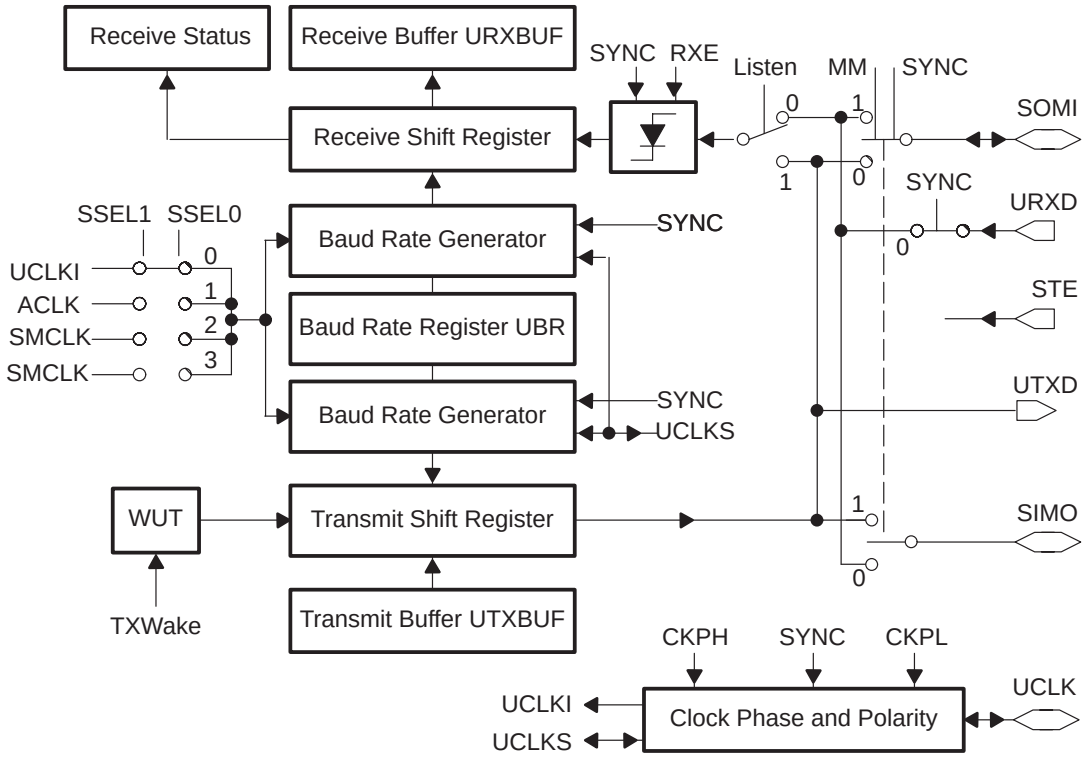
This chapter addresses the UART mode.

Topic **Page**

12.1 USART Peripheral Interface

The USART peripheral interface connects to the CPU as a byte peripheral module. It connects the MSP430 to the external system environment with three or four external pins. Figure 12–1 shows the USART peripheral interface module.

Figure 12–1. Block Diagram of USART



12.2 USART Peripheral Interface, UART Mode

The USART peripheral interface is a serial channel that shifts a serial bit stream of 7 or 8 bits in and out of the MSP430. The UART mode is chosen when control bit SYNC in the USART control register (UCTL) is reset.

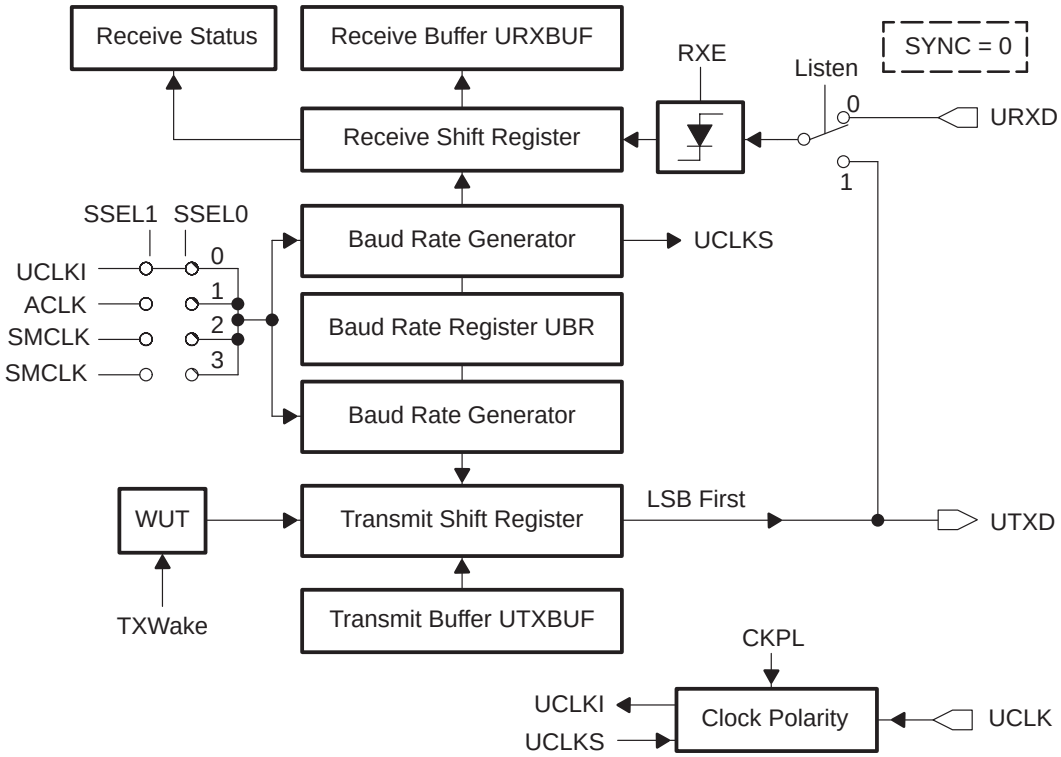
12.2.1 UART Serial Asynchronous Communication Features

Some of the UART features include:

- Asynchronous formats that include idle line/address bit-communication protocols
- Two shift registers that shift a serial data stream into URXD and out of UTXD
- Data that is transmitted/received with the LSB first
- Programmable transmit and receive bit rates
- Status flags

Figure 12–2 shows the USART in UART mode.

Figure 12–2. Block Diagram of USART—UART Mode



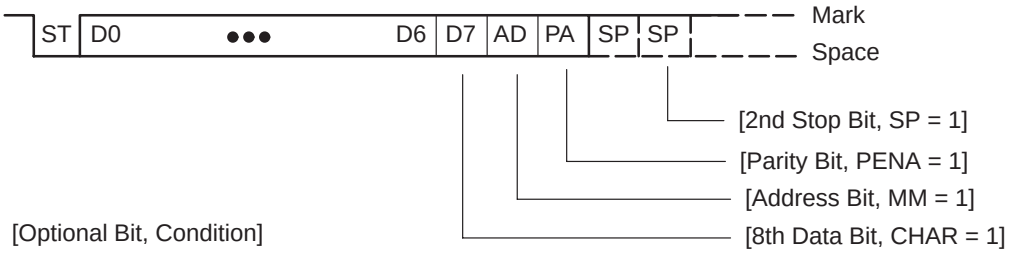
12.3 Asynchronous Operation

In the asynchronous mode, the receiver synchronizes itself to frames but the external transmitting and receiving devices do not use the same clock source; the baud rate is generated locally.

12.3.1 Asynchronous Frame Format

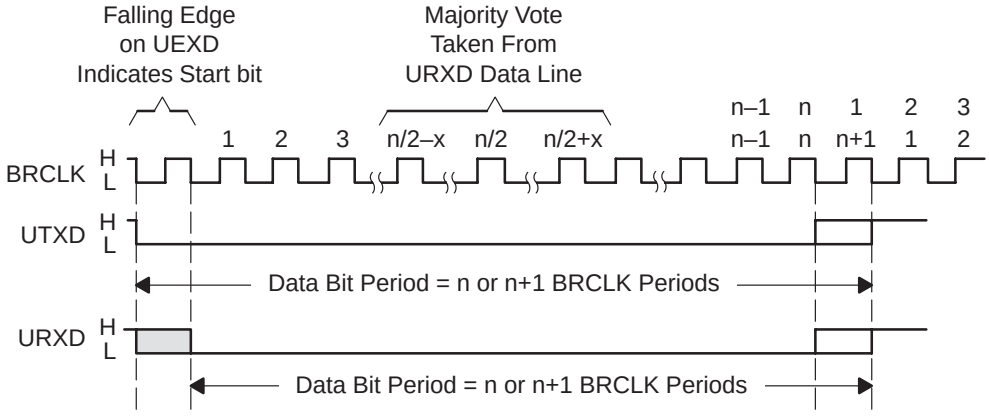
The asynchronous frame format, shown in Figure 12–3, consists of a start bit, seven or eight data bits, an even/odd/no parity bit, an address bit in address bit mode, and one or two stop bits. The bit period is defined by the selected clock source and the data in the baud rate registers.

Figure 12–3. Asynchronous Frame Format



The receive (RX) operation is initiated by the receipt of a valid start bit. It begins with a negative edge at URXD, followed by the taking of a majority vote from three samples where two of the samples must be zero. These samples occur at n/2–X, n/2, and n/2+X of the BRCLK periods following the negative edge. This sequence provides false start-bit rejection, and also locates the center of the bits in the frame, where the bits can be read on a majority basis. The timing of X is $^1/_{32}$ to $^1/_{63}$ times that of the BRCLK, depending on the division rate of the baud rate generator and provides complete coverage of at least two BRCLK periods. Figure 12–4 shows an asynchronous bit format.

Figure 12–4. Asynchronous Bit Format. Example for n or n + 1 Clock Periods



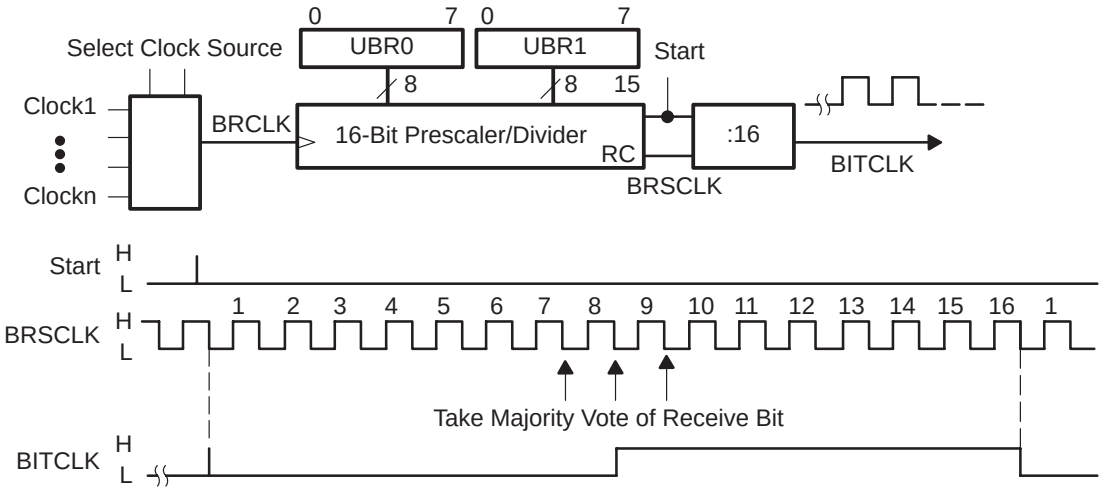
12.3.2 Baud Rate Generation in Asynchronous Communication Format

Baud rate generation in the MSP430 differs from other standard serial-communication interface implementations.

12.3.2.1 Typical Baud Rate Generation

Typical baud-rate generation uses a prescaler from any clock source and a fixed, second-clock divider that is usually divide-by-16. Figure 12–5 shows a typical baud-rate generation.

Figure 12–5. Typical Baud-Rate Generation Other Than MSP430



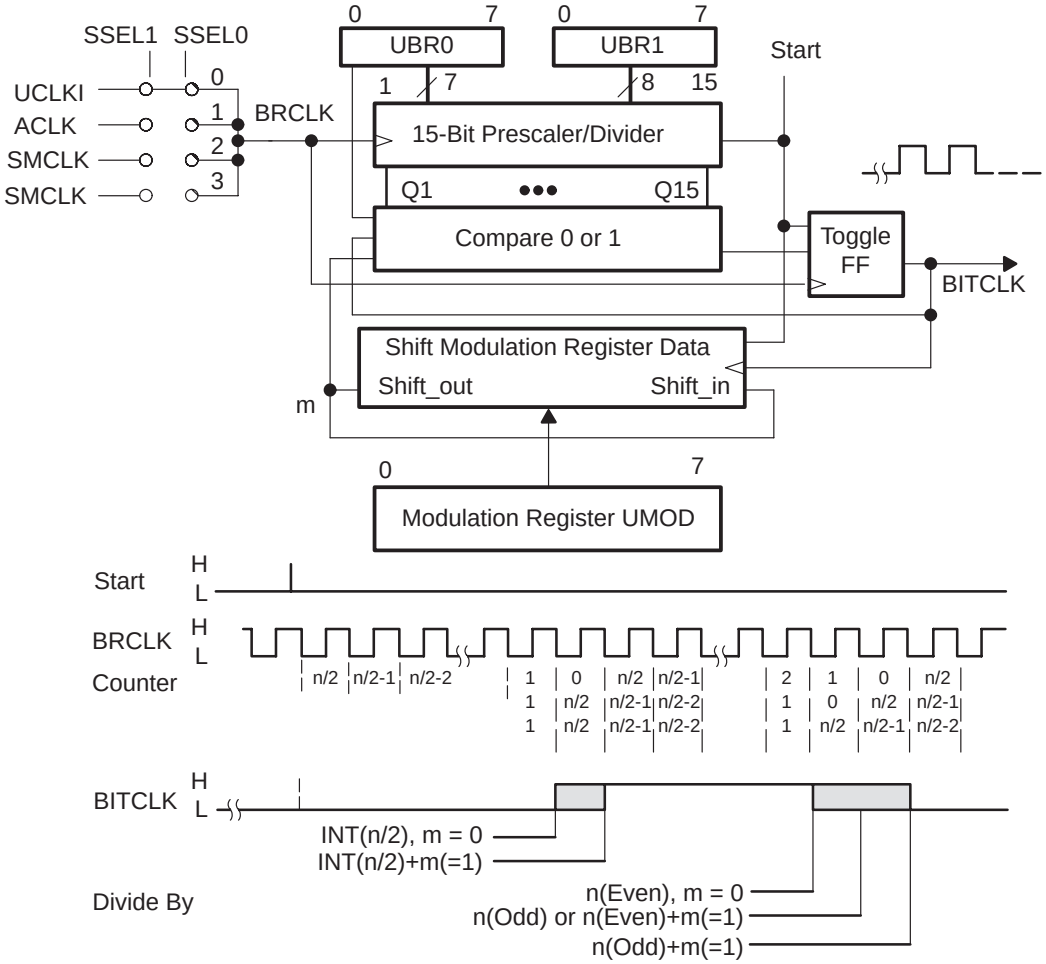
$$\text{Baud rate} = \frac{\text{BRCLK}}{n \times 16}$$

Typical baud-rate schemes often require specific crystal frequencies or cannot generate some baud rates required by some applications. For example, division factors of 18 are not possible, nor are noninteger factors such as 13.67.

12.3.2.2 MSP430 Baud Rate Generation

The MSP430 baud rate generator uses one prescaler/divider and a modulator as shown in Figure 12–6. This combination works with crystals whose frequencies are not multiples of the standard baud rates, allowing the protocol to run at maximum baud rate with a watch crystal (32,768 Hz). This technique results in power advantages because sophisticated, MSP430 low-power operations are possible.

Figure 12–6. MSP430 Baud Rate Generation. Example for n or n + 1 Clock Periods



The modulation register LSB is first used for modulation, which begins with the start bit. A set modulation bit increases the division factor by one.

Example 12–1. 4800 Baud

Assuming a clock frequency of 32,768 Hz for the BRCLK signal and a required baud rate of 4800, the division factor is 6.83. The baud rate generation in the MSP430 USART uses a factor of six plus a modulation register load of 6Fh (0110 1111). The divider runs the following sequence: 7 – 7 – 7 – 7 – 6 – 7 – 7 – 6 and so on. The sequence repeats after all eight bits of the modulator are used.

Example 12–2. 19,200 Baud

Assuming a clock frequency of 1.04 MHz (32 × 32,768 Hz) for the BRCLK signal and a required baud rate of 19,200, the division factor is 54.61. The baud rate generation in the MSP430 USART uses a factor of 54 (36h) plus a modulation register load of 0D5h. The divider runs the following sequence: 55 – 54 – 55 – 54 – 55 – 54 – 55 – 55, and so on. The sequence repeats after all eight bits of the modulator are used.

12.3.3 Asynchronous Communication Formats

The USART module supports two multiprocessor communication formats when asynchronous mode is used. These formats can transfer information between many microcomputers on the same serial link. Information is transferred as a block of frames from a particular source to one or more destinations. The USART has features that identify the start of blocks and suppress interrupts and status information from the receiver until a block start is identified. In both multiprocessor formats, the sequence of data exchanged with the USART module is based on data polling, or on the use of the receive interrupt features.

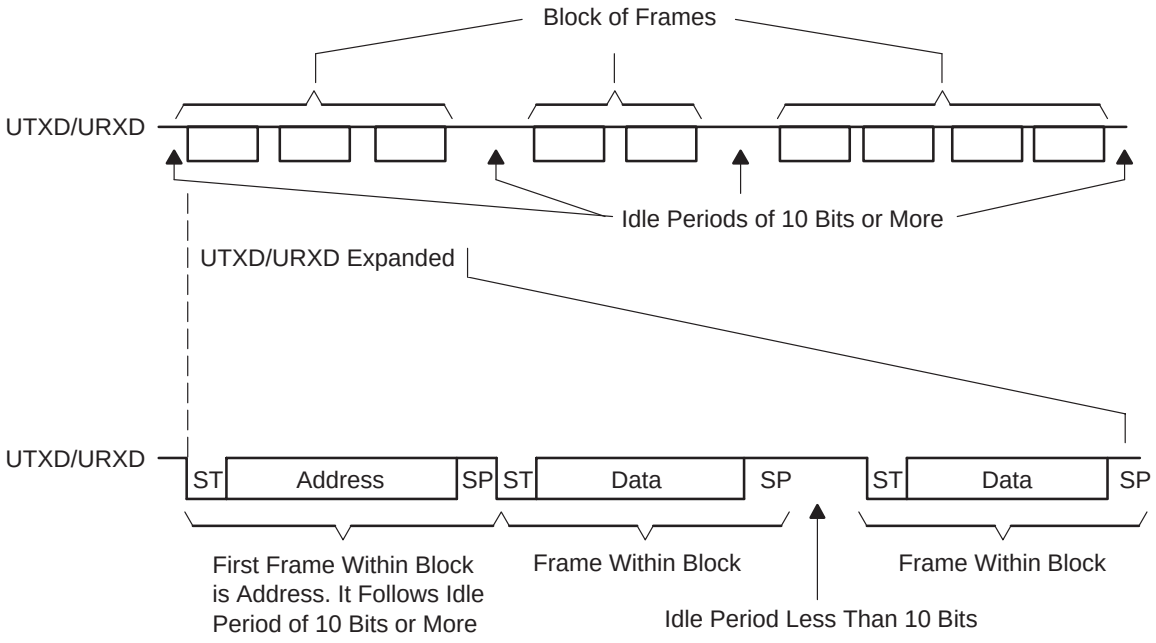
Both of the asynchronous multiprocessor formats—idle-line and address-bit—allow efficient data transfer between multiple communication systems. They can also minimize the activity of the system to save current consumption or processing resources.

The control register bit MM defines the address bit or idle-line multiprocessor format. Both use the wake-up-on-transfer mode by activating the TXWake bit (address feature function) and RXWake bit. The URXWIE and URXIE bits control the transmit and receive features of these asynchronous communication formats.

12.3.4 Idle-Line Multiprocessor Format

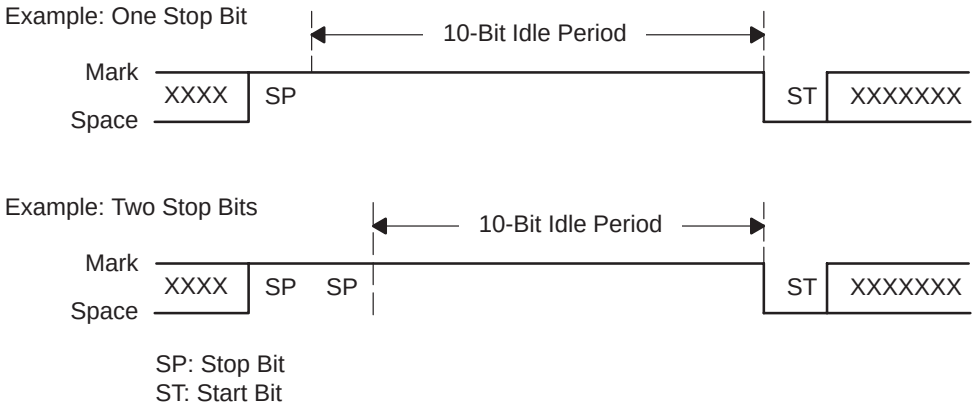
In the idle-line multiprocessor format, shown in Figure 12–7, blocks of data are separated by an idle time. An idle-receive line is detected when ten or more 1s in a row are received after the first stop bit of a character.

Figure 12–7. Idle-Line Multiprocessor Format



When two stop bits are used for the idle line, as shown in Figure 12–8, the second one is counted as the first mark bit of the idle period. The first character received after an idle period is an address character. The RXWake bit can be used as an address tag for the character. In the idle-line multiprocessor format, the RXWake bit is set when a received character is an address character and is transferred into the receive buffer.

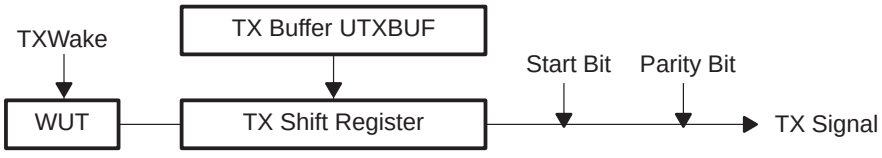
Figure 12–8. USART Receiver Idle Detect



Normally, if the USART URXWIE bit is set in the receive control register, characters are assembled as usual by the receiver. They are not, however, transferred to the receiver buffer, URXBUF, nor are interrupts generated. When an address character is received, the receiver is temporarily activated to transfer the character to URXBUF and to set the URXIFG interrupt flag. Applicable error status flags are set. The application software can validate the received address. If there is a match, the application software further processes the data and executes the operation. If there is no match, the processor waits for the next address character to arrive. The URXWIE bit is not modified by the USART: it must be modified manually to receive nonaddress or address characters.

In idle-line multiprocessor format, a precise idle period can be generated to create efficient address character identifiers. The wake-up temporary (WUT) flag is an internal flag and is double-buffered with TXWake. When the transmitter is loaded from UTXBUF, WUT is loaded from TXWake, and the TXWake bit is reset as shown in Figure 12–9.

Figure 12–9. Double-Buffered WUT and TX Shift Register



The following procedure sends out an idle frame to identify an address character:

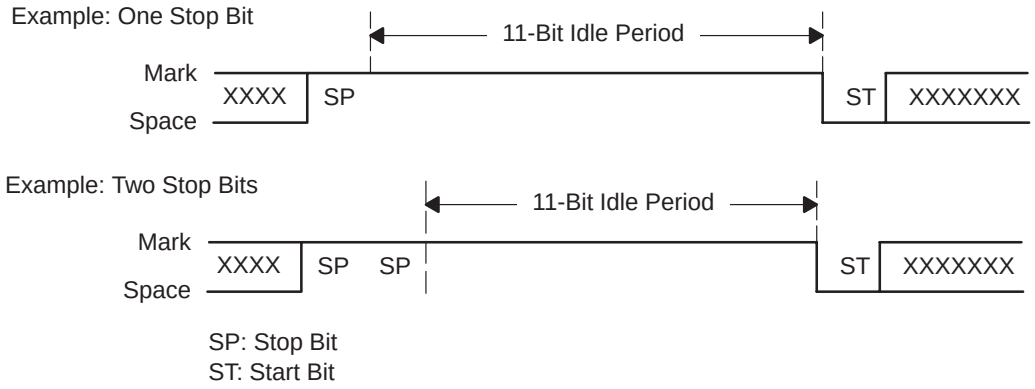
1) Set the TXWake bit and then write any word (don't care) to the UTXBUF (UTXIFG must be set).

   When the transmitter shift register is empty, the contents of UTXBUF are shifted to the transmit shift register and the TXWake value is shifted to WUT.

2) Set bit WUT, which suppresses the start, data, and parity bits and transmits an idle period of exactly 11 bits, as shown in Figure 12–10.

   The next data word, shifted out of the serial port after the address-character identifying idle period, is the second word written to the UTXBUF after the TXWake bit has been set. The first data word written is suppressed while the address identifier is sent out and ignored thereafter. Writing the first don't care word to UTXBUF is necessary to shift the TXWAKE bit to WUT and generate an idle-line condition.

Figure 12–10. USART Transmitter Idle Generation

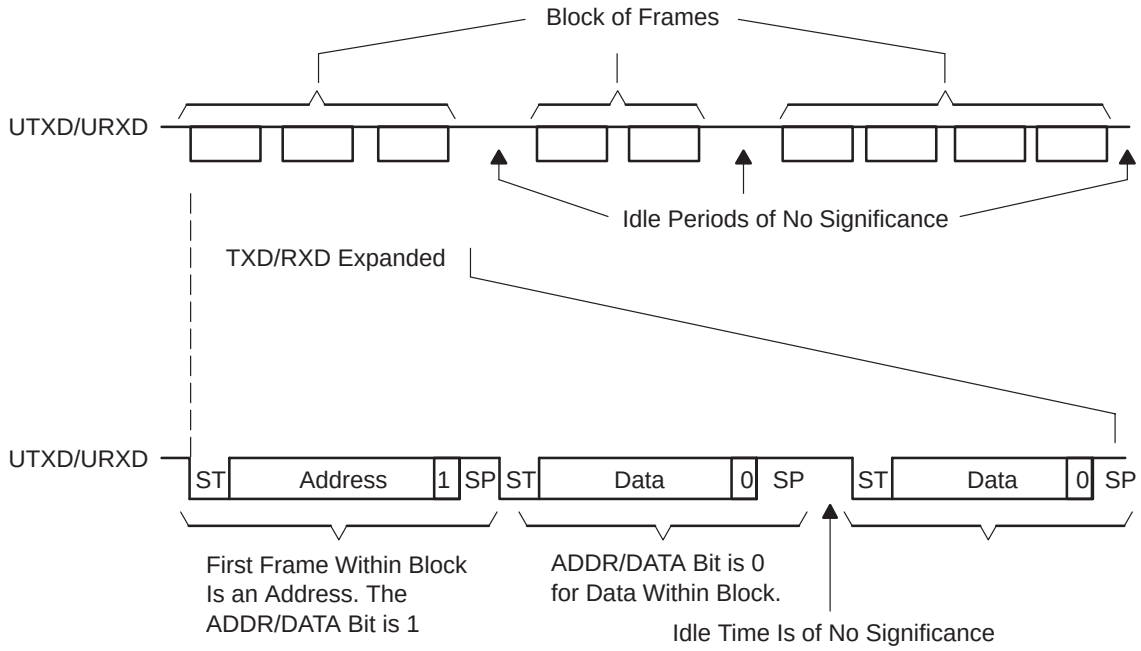


12.3.5 Address-Bit Multiprocessor Format

In the address-bit multiprocessor format shown in Figure 12–11, characters contain an extra bit used as an address indicator. The first character in a block of data carries an address bit which indicates that the character is an address. The RXWake bit is set when a received character is an address character. It is transferred into the receive buffer (receive conditions are true).

Usually, if the USART URXWIE bit is set, data characters are assembled by the receiver but are not transferred to the receiver buffer URXBUF, nor are interrupts generated. When a character that has an address bit set is received, the receiver is temporarily activated to transfer the character to URXBUF and to set the URXIFG. Error status flags are set as applicable. The application software processes the succeeding operation to optimize resource handling or reduce current consumption. The application software can validate the received address. If there is a match, the processor can read the remainder of the data block. If there is not a match, the processor waits for the next address character to arrive.

Figure 12–11. Address-Bit Multiprocessor Format



In the address-bit multiprocessor mode, the address bit of a character can be controlled by writing to the TXWake bit. The value of the TXWake bit is loaded into the address bit of that character each time a character is transferred from transmit buffer UTXBUF to the transmitter. The TXWake bit is then cleared by the USART.

12.4 Interrupt and Enable Functions

The USART peripheral interface serves two main interrupt sources for transmission and reception. Two interrupt vectors serve receive and transmit events.

The interrupt control bits and flags and enable bits of the USART peripheral interface are located in the SFR registers. They are discussed in Table 12–1. See the peripheral file map in Appendix A for the exact bit locations.

Table 12–1. USART Interrupt Control and Enable Bits – UART Mode

Receive interrupt flag	URXIFG	Initial state reset (by PUC/SWRST)
Receive interrupt enable	URXIE	Initial state reset (by PUC/SWRST)
Receive enable (see note)	URXE	Initial state reset (by PUC)
Transmit interrupt flag	UTXIFG	Initial state set (by PUC/SWRST)
Transmit interrupt enable	UTXIE	Initial state reset (by PUC/SWRST)
Transmit enable.(see note)	UTXE	Initial state reset (by PUC)

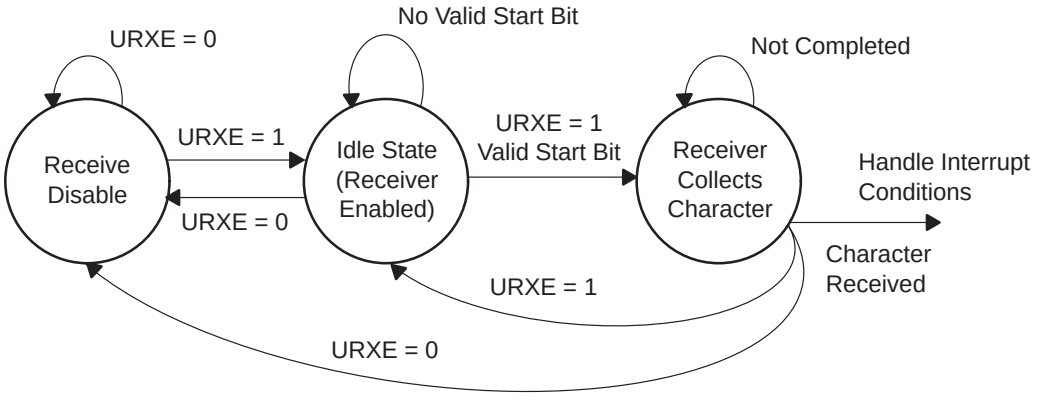
Note: Different for SPI mode, see Chapter 13.

The USART receiver and transmitter operate independently, but use the same baud rate.

12.4.1 USART Receive Enable Bit

The receive enable bit URXE, shown in Figure 12–12, enables or disables receipt of the bit stream on the URXD data line. Disabling the USART receiver stops the receive operation after completion of receiving the character, or stops immediately if no receive operation is active. Start-bit detection is also disabled.

Figure 12–12. State Diagram of Receiver Enable



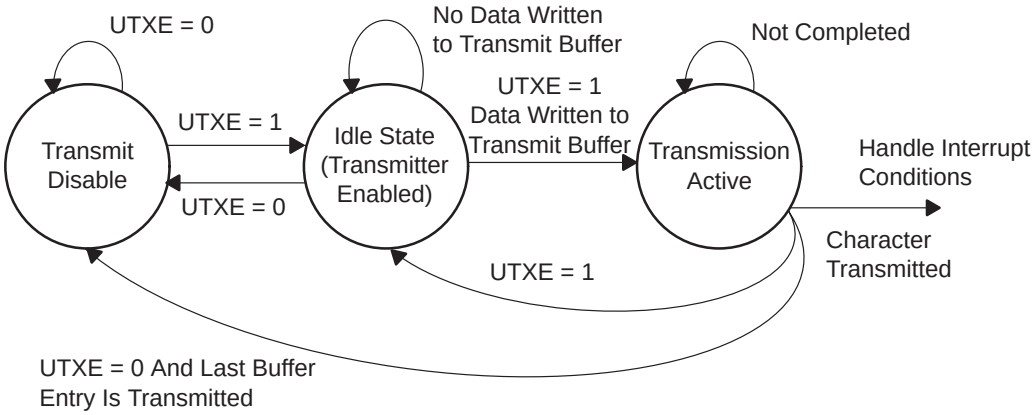
Note: URXE Reenabled, UART Mode

Because the receiver is completely disabled, reenabling the receiver is asynchronous to any data stream on the communication line. Synchronization can be performed by looking for an idle line condition before receiving a character.

12.4.2 USART Transmit Enable Bit

The transmit enable bit UTXE, shown in Figure 12–13, enables or disables a character transmission on the serial-data line. If this bit is reset, the transmitter is disabled but any active transmission does not halt until the data in the transmit shift register and the transmit buffer are transmitted. Data written to the transmit buffer before UTXE has been reset may be modified or overwritten—even after UTXE is reset—until it is shifted to the transmit shift register. For example, if software writes a byte to the transmit buffer and then resets UTXE, the byte written to the transmit buffer will be transmitted and may be modified or overwritten until it is transferred into the transmit shift register. However, after the byte is transferred to the transmit shift register, any subsequent writes to UTXBUF while UTXE is reset will not result in transmission, but UTXBUF will be updated with the new value.

Figure 12–13. State Diagram of Transmitter Enable



When UTXE is reset and the current transmission is completed, new data written to the transmit buffer will not be transmitted. Once the UTXE bit is set, the data in the transmit buffer are immediately loaded into the transmit shift register and character transmission is started.

Note: Writing to UTXBUF, UART Mode

Data should never be written to transmit buffer UTXBUF when the buffer is not ready and when the transmitter is enabled (UTXE is set). Otherwise, the transmission will have errors.

Note: Write to UTXBUF/Reset of Transmitter, UART Mode

Disabling the transmitter should be done only if all data to be transmitted has been moved to the transmit shift register.

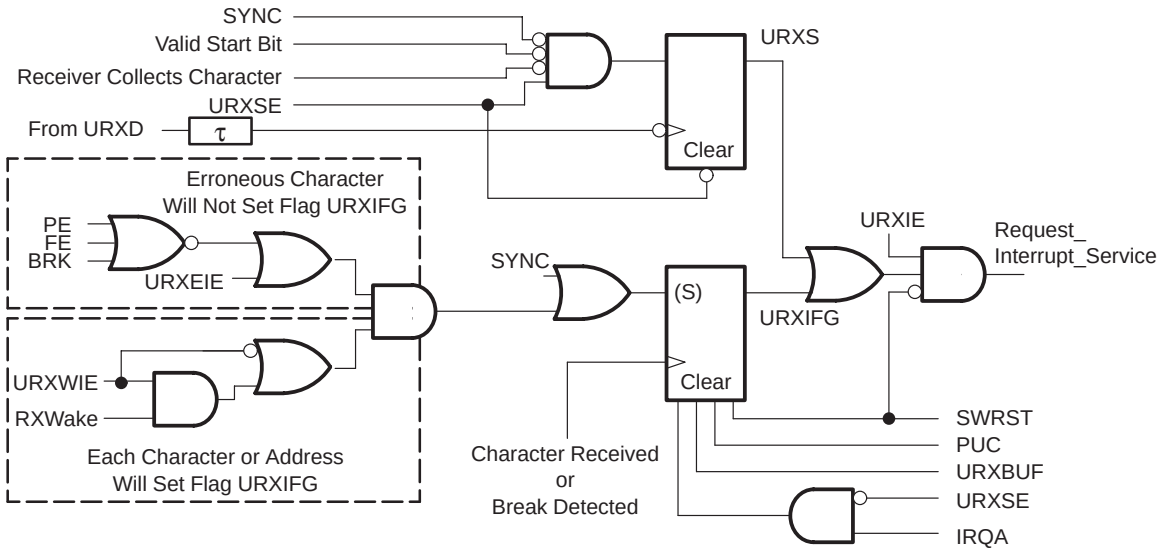
```
MOV.B  #....,&UTXBUF
BIC.B  #UTXE,&ME2        ; If BITCLK < MCLK then the
                         ; transmitter might be stopped
                         ; before the buffer is loaded
                         ; into the transmitter shift
                         ; register
```

12.4.3 USART Receive Interrupt Operation

In the receive interrupt operation, shown in Figure 12–14, the receive interrupt flag URXIFG is set or is unchanged each time a character is received and loaded into the receive buffer:

- Erroneous characters (parity, frame, or break error) do not set interrupt flag URXIFG when URXEIE is reset: URXIFG is unchanged.
- All types of characters (URXWIE = 0), or only address characters (URXWIE = 1), set the interrupt flag URXIFG. When URXEIE is set, erroneous characters can also set the interrupt flag URXIFG.

Figure 12–14. Receive Interrupt Operation



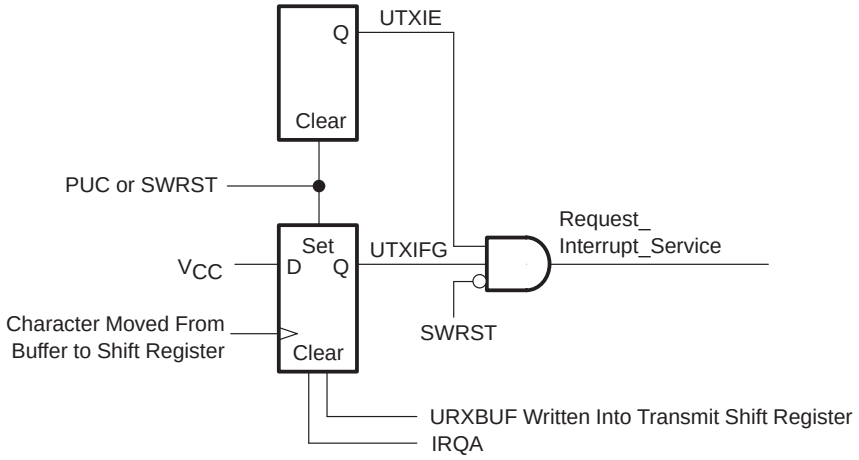
URXIFG is reset by a system reset PUC signal, or with a software reset (SWRST). URXIFG is reset automatically if the interrupt is served (URXSE = 0) or the receive buffer URXBUF is read. A set receive interrupt flag URXIFG indicates that an interrupt event is waiting to be served. A set receive interrupt enable bit URXIE enables serving a waiting interrupt request. Both the receive interrupt flag URXIFG and the receive interrupt enable bit URXIE are reset with the PUC signal and a SWRST.

Signal URXIFG can be accessed by the software, whereas signal URXS cannot. When both interrupt events—character receive action and receive start detection—are enabled by the software, the flag URXIFG indicates that a character was received but the start-detect interrupt was not. Because the interrupt software handler for the receive start detection resets the URXSE bit, this clears the URXS bit and prevents further interrupt requests from URXS. The URXIFG should already be reset since no set condition was active during URXIFG latch time.

12.4.4 USART Transmit Interrupt Operation

In the transmit interrupt operation, shown in Figure 12–15, the transmit interrupt flag UTXIFG is set by the transmitter to indicate that the transmitter buffer UTXBUF is ready to accept another character. This bit is automatically reset if the interrupt request service is started or a character is written into the UTXBUF. This flag asserts a transmitter interrupt if the local (UTXIE) and general interrupt enable (GIE) bits are set. The UTXIFG is set after a system reset PUC signal, or removal of a SWRST.

Figure 12–15. Transmit Interrupt Operation



The transmit interrupt enable UTXIE bit controls the ability of the UTXIFG to request an interrupt, but does not prevent the flag UTXIFG from being set. The UTXIE is reset with a PUC signal or a software reset (SWRST) bit. The UTXIFG bit is set after a system reset PUC signal or software reset (SWRST), but the UTXIE bit is reset to ensure full interrupt-control capability.

12.5 Control and Status Registers

The USART control and status registers are byte-structured and should be accessed using byte processing instructions (suffix B). Tables 12–2 and 12–3 list the registers and their access modes.

Table 12–2. USART0 Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USART control	UCTL0	Read/write	070h	See section 12.5.1.
Transmit control	UTCTL0	Read/write	071h	See section 12.5.2.
Receive control	URCTL0	Read/write	072h	See section 12.5.3.
Modulation control	UMCTL0	Read/write	073h	Unchanged
Baud rate 0	UBR00	Read/write	074h	Unchanged
Baud rate 1	UBR10	Read/write	075h	Unchanged
Receive buffer	URXBUF0	Read/write	076h	Unchanged
Transmit buffer	UTXBUF0	Read	077h	Unchanged

Table 12–3. USART1 Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USART control	UCTL1	Read/write	078h	See section 12.5.1.
Transmit control	UTCTL1	Read/write	079h	See section 12.5.2.
Receive control	URCTL1	Read/write	07Ah	See section 12.5.3.
Modulation control	UMCTL1	Read/write	07Bh	Unchanged
Baud rate 0	UBR01	Read/write	07Ch	Unchanged
Baud rate 1	UBR11	Read/write	07Dh	Unchanged
Receive buffer	URXBUF1	Read/write	07Eh	Unchanged
Transmit buffer	UTXBUF1	Read	07Fh	Unchanged

All bits are random after a PUC signal, unless otherwise noted by the detailed functional description.

The reset of the USART peripheral interface is performed by a PUC signal or a SWRST. After a PUC signal, the SWRST bit remains set and the USART interface remains in the reset condition until it is disabled by resetting the SWRST bit.

The USART module operates in asynchronous or synchronous mode as defined by the SYNC bit. The bits in the control registers can have different functions in the two modes. All bits in this section are described with their functions in the asynchronous mode (SYNC = 0). Their functions in the synchronous mode are described in Chapter 13, *USART Peripheral Interface, SPI Mode*.

12.5.1 USART Control Register UCTL

The information stored in the USART control register (UCTL), shown in Figure 12–16, determines the basic operation of the USART module. The register bits select the communications protocol, communication format, and parity bit. All bits must be programmed according to the selected mode before resetting the SWRST bit to disable the reset.

Figure 12–16. USART Control Register UCTL

UCTL0, 070h
UCTL1, 078h

7							0
PENA	PEV	SP	CHAR	Listen	SYNC	MM	SWRST
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–1

Bit 0: The USART state machines and operating flags are initialized to the reset condition (URXIFG = URXIE = UTXIE = 0, UTXIFG = 1) if the software reset bit is set. Until the SWRST bit is reset, all affected logic is held in the reset state. This implies that after a system reset the USART must be reenabled by resetting this bit. The receive and transmit enable flags URXE and UTXE are not altered by SWRST.

The SWRST bit resets the following bits and flags: URXIE, UTXIE, URXIFG, RXWAKE, TXWAKE, RXERR, BRK, PE, OE, and FE

The SWRST bit sets the following bits: UTXIFG, TXEPT

Bit 1: Multiprocessor mode (address/idle-line wake up)
Two multiprocessor protocols, idle-line and address-bit, are supported by the USART module. The choice of multiprocessor mode affects the operation of the automatic address decoding functions.
MM = 0: Idle-line multiprocessor protocol
MM = 1: Address-bit multiprocessor protocol
The conventional asynchronous protocol uses MM-bit reset.

Bit 2: Mode or function of USART module selected
The SYNC bit selects the function of the USART peripheral interface module. Some of the USART control bits have different functions in UART and SPI mode.
SYNC = 0: UART function is selected
SYNC = 1: SPI function is selected

Bit 3: The listen bit selects if the transmitted data is fed back internally to the receiver.
Listen = 0: No feedback
Listen = 1: Transmit signal is internally fed back to the receiver. This is commonly known as loopback mode.

Bit 4: Character length
This register bit selects the length of the character to be transmitted as either 7 or 8 bits. 7-bit characters do not use the eighth bit in URXBUF and UTXBUF. This bit is padded with *0*.

CHAR = 0: 7-bit data
CHAR = 1: 8-bit data

Bit 5: Number of stop bits
This bit determines the number of stop bits transmitted. The receiver checks for one stop bit only.
SP = 0: one stop bit
SP = 1: two stop bits

Bit 6: Parity odd/even
If the PENA bit is set (parity bit is enabled), the PEV bit defines odd or even parity according to the number of odd or even 1 bits (in both the transmitted and received characters), the address bit (address-bit multiprocessor mode), and the parity bit.
PEV = 0: odd parity
PEV = 1: even parity

Bit 7: Parity enable
If parity is disabled, no parity bit is generated during transmission or expected during reception. A received parity bit is not transferred to the URXBUF with the received data as it is not considered one of the data bits. In address-bit multiprocessor mode, the address bit is included in the parity calculation.
PEN = 0: Parity disable
PEN = 1: Parity enable

Note: Mark and Space Definitions

The mark condition is identical to the signal level in the idle state. Space is the opposite signal level: the start bit is always space.

12.5.2 Transmit Control Register UTCTL

The transmit control register (UTCTL), shown in Figure 12–17, controls the USART hardware associated with the transmit operation.

Figure 12–17. Transmitter Control Register UTCTL

UTCTL1, 07Bh
UTCTL0, 071h

7							0
Unused	CKPL	SSEL1	SSEL0	URXSE	TXWake	Unused	TXEPT
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–1

Bit 0: The transmitter empty (TXEPT) flag is set when the transmitter shift register and UTXBUF are empty, and is reset when data is written to UTXBUF. It is set by a SWRST.

Bit 1: Unused

Bit 2: The TXWake bit controls the transmit features of the multiprocessor communication modes. Each transmission —started by loading the UTXBUF—uses the state of the TXWake bit to initialize the address-identification feature. It must not be cleared—the USART hardware clears this bit once it is transferred to the WUT; a SWRST also clears the TXWake bit.

Bit 3: The receive-start edge-control bit, if set, requests a receive interrupt service. For a successful interrupt service, the corresponding enable bits URXIE and GIE must be set. The advantage of this bit is that it starts the controller clock system, including MCLK, along with the interrupt service, and keeps it running by modifying the mode control bits.

Bits 4, 5: Source select 0 and 1
The source select bit defines which clock source is used for baud-rate generation:

SSEL1, SSEL0	0	External clock, UCLKI
	1	ACLK
	2, 3	SMCLK

Bit 6: Clock polarity CKPL
The CKPL bit controls the polarity of the UCLKI signal.
CKPL = 0: The UCLKI signal has the same polarity as the UCLK signal.
CKPL = 1: The UCLKI signal has an inverted polarity to the UCLK signal.

Bit 7: Unused

12.5.3 Receiver Control Register URCTL

The receiver-control register (URCTL), shown in Figure 12–18, controls the USART hardware associated with the receiver operation and holds error and wake-up conditions modified by the latest character written to the receive buffer (URXBUF). Once any one of the bits FE, PE, OE, BRK, RXERR, or RXWake is set, none are reset by receiving another character. The bits are reset by accessing the receive buffer, by a USART software reset (SWRST), by a system reset PUC signal, or by an instruction.

Figure 12–18. Receiver-Control Register URCTL

URCTL0, 072h
URCTL1, 07Ah

7							0
FE	PE	OE	BRK	URXEIE	URXWIE	RXWake	RXERR
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

Bit 0: The receive error bit (RXERR) indicates that one or more error flags (FE, PE, OE, or BRK) is set. It is not reset when the error flags are cleared by instruction.

Bit 1: Receiver wake-up detect
The RXWake bit is set when a received character is an address character and is transferred into the receive buffer.
Address-bit multiprocessor mode: RXWake is set when the address bit is set in the character received.
Idle-line multiprocessor mode: RXWake is set if an idle URXD line is detected (11 bits of mark level) in front of the received character.

RXWake is reset by accessing the receive buffer (URXBUF), by a USART software reset, or by a system-reset PUC signal.

Bit 2: The receive wake-up interrupt-enable bit (URXWIE) selects the type of character to set the interrupt flag (URXIFG):
URXWIE = 0: Each character received sets the URXIFG
URXWIE = 1: Only characters that are marked as address characters set the interrupt flag URXIFG. It operates identically in both multiprocessor modes.
The wake-up interrupt enable feature depends on the receive erroneous-character feature. See also Bit 3, URXEIE.

Bit 3: The receive erroneous-character interrupt-enable bit URXEIE selects whether an erroneous character is to set the interrupt flag URXIFG.
URXEIE = 0: Each erroneous character received does not alter the interrupt flag URXIFG.
URXEIE = 1: All characters can set the interrupt flag URXIFG as described in Table 12–4, depending on the conditions set by the URXWIE bit.

Table 12–4. Interrupt Flag Set Conditions

URXEIE	URXWIE	Char. w/Error	Char. Address	Description Flag URXIFG After a Character is Received
0	X	1	X	Unchanged
0	0	0	X	Set
0	1	0	0	Unchanged
0	1	0	1	Set
1	0	X	X	Set (Receives all characters)
1	1	X	0	Unchanged
1	1	X	1	Set

Bit 4: The break detect bit (BRK) is set when a break condition occurs and the URXEIE bit is set. The break condition is recognized if the RXD line remains continuously low for at least 10 bits, beginning after a missing first stop bit. It is not cleared by receipt of a character after the break is detected, but is reset by a SWRST, a system reset, or by reading the URXBUF. The receive interrupt flag URXIFG is set if a break is detected.

Bit 5: The overrun error flag bit OE is set when a character is transferred into the URXBUF before the previous character is read out. The previous character is overwritten and lost. OE is reset by a SWRST, a system reset, or by reading the URXBUF.

Bit 6: The parity error flag bit PE is set when a character is received with a mismatch between the number of 1s and its parity bit, and is loaded into the receive buffer. The parity checker includes the address bit, used in the address-bit multiprocessor mode, in the

calculation. The flag is disabled if parity generation and detection are not enabled. In this case the flag is read as 0. It is reset by a SWRST, a system reset, or by reading the URXBUF.

Bit 7: The framing error flag bit FE is set when a character is received with a 0 stop bit and is loaded into the receive buffer. Only the first stop bit is checked when more than one is used. The missing stop bit indicates that the start-bit synchronization is lost and the character is incorrectly framed. FE is reset by a SWRST, a system reset, or by reading the URXBUF.

Note: Receive Status Control Bits

The receive status control bits FE, PE, OE, BRK, and RXWake are set by the hardware according to the conditions of the characters received. Once the bits are set, they remain set until the software resets them directly, or there is a reading of the receive buffer. False character interpretation or missing-interrupt capability can result in uncleared error bits.

12.5.4 Baud Rate Select and Modulation Control Registers

The baud-rate generator uses the content of the baud-rate select registers UBR0 and UBR1 shown in Figure 12–19, with the modulation control register to generate the serial data-stream bit timing.

Figure 12–19. USART Baud Rate Select Register

UBR00, 074h
UBR01, 07Ch

7							0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
rw	rw	rw	rw	rw	rw	rw	rw

UBR10, 075h
UBR11, 07Dh

7							0
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
rw	rw	rw	rw	rw	rw	rw	rw

$$\text{Baud rate} = \frac{BRCLK}{UBR + \frac{1}{n}\sum_{i=0}^{n-1} mi}$$ with UBR= [UBR1,UBR0]

The baud-rate control register range is: $3 \le UBR < 0FFFFh$

Note:

Unpredictable receive and transmission occur if UBR <3.

The modulation control register, shown in Figure 12–20, ensures proper timing generation with the UBR0 and UBR01, even with crystal frequencies that are not integer multiples of the required baud rate.

Figure 12–20. USART Modulation Control Register

UMCTL0, 073h
UMCTL1, 07Bh

7							0
m7	m6	m5	m4	m3	m2	m1	m0
rw	rw	rw	rw	rw	rw	rw	rw

The timing of the running bit is expanded by one clock cycle of the baud-rate-divider input clock if bit m_i is set.

Each time a bit is received or transmitted, the next bit in the modulation control register determines the present bit timing. The first bit time in the protocol—the start bit time—is determined by UBR plus m0; the next bit is determined by UBR plus m1, and so on.

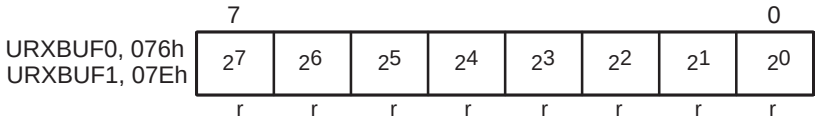
The modulation sequence is:

m0 – m1 – m2 – m3 – m4 – m5 – m6 – m7 – m0 – m1 – m2 – .....

12.5.5 Receive-Data Buffer URXBUF

The receive-data buffer (URXBUF), shown in Figure 12–21, contains previous data from the receiver shift register. Reading URXBUF resets the receive-error bits, the RXWake bit, and the interrupt flag (URXIFG).

Figure 12–21. USART0 Receive Data Buffer URXBUF



In seven-bit length mode, the MSB of the URXBUF is always reset.

The receive data buffer is loaded with the recently-received character as described in Table 12–5, when receive and control conditions are true.

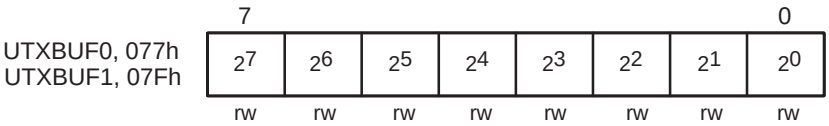
Table 12–5. Receive Data Buffer Characters

URXEIE	URXWIE	**Load URXBUF With:**	PE	FE	BRK
0	1	Error-free address characters	0	0	0
1	1	All address characters	X	X	X
0	0	Error-free characters	0	0	0
1	0	All characters	X	X	X

12.5.6 Transmit Data Buffer UTXBUF

The transmit data buffer (UTXBUF), shown in Figure 12–22, contains current data to be transmitted.

Figure 12–22. Transmit Data Buffer UTXBUF



The UTXIFG flag indicates that the UTXBUF buffer is ready to accept another character for transmission.

The transmission is initialized by writing data to UTXBUF. The transmission of this data is started immediately if the transmitter shift register is empty or is going to be empty.

Note: Writing to UTXBUF

Writing data to the transmit-data buffer must only be done if buffer UTXBUF is empty; otherwise, an unpredictable character can be transmitted.

12.6 Utilizing Features of Low-Power Modes

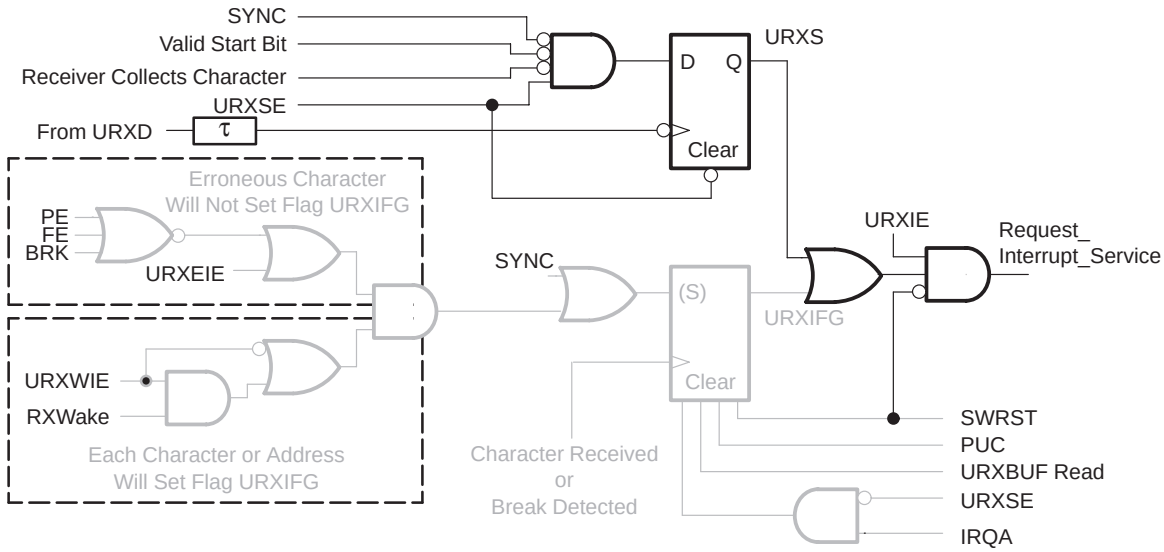
There are several functions or features of the USART that support the ultralow power architecture of the MSP430. These include:

- Support system start up from any processor mode by sensing of UART frame-start condition
- Use the lowest input clock frequency for the required baud rate
- Support multiprocessor modes to reduce use of MSP430 resources

12.6.1 Receive-Start Operation From UART Frame

The most effective use of start detection in the receive path is achieved when the baud-rate clock runs from SMCLK. In this configuration, the MSP430 can be put into a low-power mode with SMCLK disabled. The receive-start condition is the negative edge from the signal on pin URXD. Each time the negative edge triggers the interrupt flag URXS, it requests a service when enable bits URXIE and GIE are set. This wakes the MSP430 and the system returns to active mode, supporting the USART transfer.

Figure 12–23. Receive-Start Conditions



Three character streams do not set the interrupt flag (URXIFG):

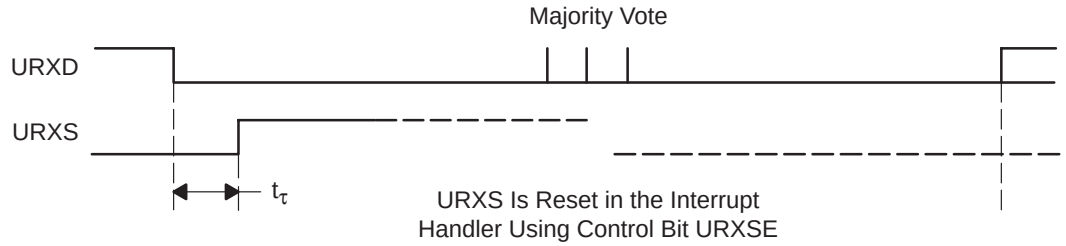
- Erroneous characters (URXEIE = 0)
- Address characters (URXWIE = 1)
- Invalid start-bit detection

The interrupt software should handle these conditions.

12.6.1.1 Start Conditions

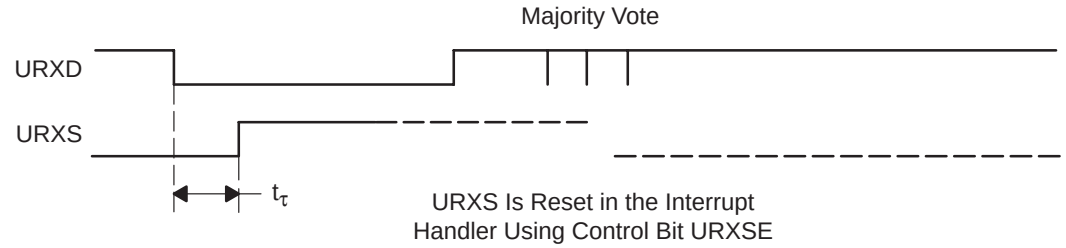
The URXD signal feeds into the USART module by first going into a deglitch circuit. Glitches cannot trigger the receive-start condition flag URXS, which prevents the module from being started from small glitches on the URXD line. Because glitches do not start the system or the USART module, current consumption is reduced in noisy environments. Figure 12–24 shows the accepted receive-start timing condition.

Figure 12–24. Receive-Start Timing Using URXS Flag, Start Bit Accepted



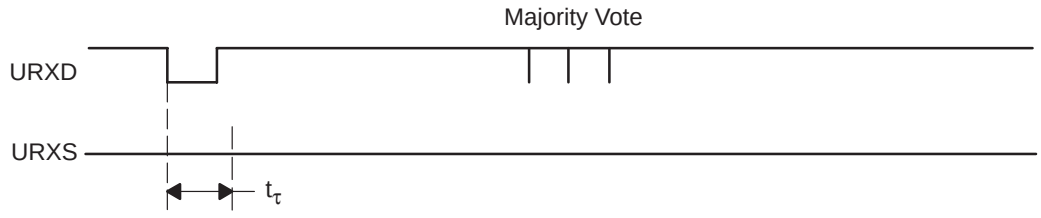
The UART stops receiving a character when the URXD signal exceeds the deglitch time t_τ but the majority vote on the signal fails to detect a start bit, as shown in Figure 12–25. The software should handle this condition and return the system to the appropriate low-power mode. The interrupt flag URXIFG is not set.

Figure 12–25. Receive-Start Timing Using URXS Flag, Start Bit Not Accepted



Glitches at the URXD line are suppressed automatically and no further activity occurs in the MSP430 as shown in Figure 12–26. The data for the deglitch time t_τ is noted in the corresponding device specification.

Figure 12–26. Receive-Start Timing Using URXS Flag, Glitch Suppression



The interrupt handler must reset the URXSE bit in control register UCTL to prevent further interrupt service requests from the URXS signal and to enable the basic function of the receive interrupt flag URXIFG.

```
*************************************************************
*    Interrupt handler for frame start condition and        *
*    Character receive                                      *

*************************************************************

IFG2      .EQU  3               ; URXIFG and UTXIFG in
                                ; address 3
UTCTL     .EQU  71h             ;
UTXIFG    .EQU  0               ;
URXSE     .EQU  8               ;
                                ;
URX_Int   BIT.B #URXIFG,&IFG2   ; test URXIFG signal to
          JNE   ST_COND         ; check if frame start
                                ; condition
          .....
          .....
ST_COND   BIC.B #URXSE,&UTCTL   ; clear ff/signal URXS,
                                ; stop further interrupt
                                ; requests
          BIS.B #URXSE,&UTCTl   ; Prepare FF_URXS for next
                                ; frame start bits and set
          .....                 ; the conditions to run the
          .....                 ; clock needed for UART RX
```

Note: Break Detect (BRK) Bit With Halted UART Clock

If the UART operates with the wake-up-on-start-condition mode and switches off the UCLK whenever a character is completely received, a communication line break cannot be detected automatically by the UART hardware. The break detection requires the baud-rate generator BRSCLK, but it is stopped upon the missing UCLK.

12.6.2 Maximum Utilization of Clock Frequency vs Baud Rate UART Mode

The current consumption increases linearly with the clock frequency. It should be kept to the minimum required to meet application conditions. Fast communication speed is needed for calibration and testing in manufacturing processes, alarm responses in critical applications, and response time to human requests for information.

The MSP430 USART can generate baud rates up to one third of the clock frequency. An additional modulation of the baud-rate timing adjusts timing for individual bits within a frame. The timing is adjusted from bit to bit to meet timing requirements even when a noninteger division is needed. Baud rates up to 4800 baud can be generated from a 32,768 Hz crystal with maximum errors of 11 percent. Standard UARTs—even with the worst maximum error (–14.6 percent)—can obtain maximum baud rates of 75 baud.

12.6.3 Support of Multiprocessor Modes for Reduced Use of MSP430 Resources

Communication systems can use multiprocessor modes with multiple-character idle-line or address-bit protocols. The first character can be a target address, a message identifier, or can have another definition. This character is interpreted by the software and, if it is of any significance to the application, the succeeding characters are collected and further activities are defined. An insignificant first character would stop activity for the processing device. This application is supported by the wake-up interrupt feature in the receive operation, and sends wake-up conditions along with a transmission. Avoiding activity on insignificant characters reduces consumption of MSP430 resources and the system can remain in the most efficient power-conserving mode.

In addition to the multiprocessor modes, rejecting erroneous characters saves MSP430 resources. This practice prevents interrupt handling of the erroneous characters. The processor waits in the most efficient power-conserving mode until a character is processed.

12.7 Baud Rate Considerations

The MSP430 baud-rate generator uses a divider and a modulator. A given crystal frequency and a required baud rate determines the required division factor N:

$$N = \frac{\text{BRCLK}}{\text{baud rate}}$$

The required division factor N usually has an integer part and a fraction. The divider in the baud rate generator realizes the integer portion of the division factor N, and the modulator meets the fractional part as closely as possible. The factor N is defined as:

$$N = UBR + \frac{1}{n}\sum_{i=0}^{n-1} mi$$

Where:

N: Target division factor
UBR: 16-bit representation of registers UBR1 and UBR0
i: Actual bit in the frame
n: Number of bits in the frame
m_i: Data of the actual modulation bit

$$Baud\ rate = \frac{BRCLK}{N} = \frac{BRCLK}{UBR + \frac{1}{n}\sum_{i=0}^{n-1} m_i}$$

12.7.1 Bit Timing in Transmit Operation

The timing for each individual bit in one frame or character is the sum of the actual bit timings as shown in Figure 12–27. The baud-rate generation error shown in Figure 12–28 in relation to the required ideal timing, is calculated for each individual bit. The relevant error information is the error relative to the actual bit, not the overall relative error.

Figure 12–27. MSP430 Transmit Bit Timing

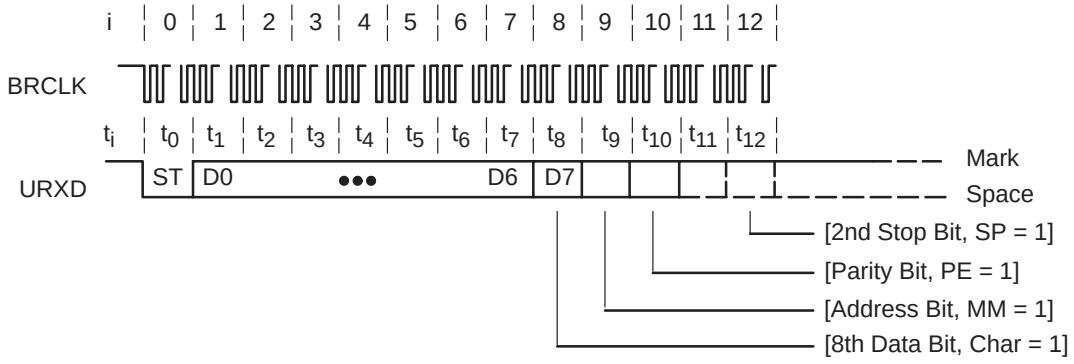
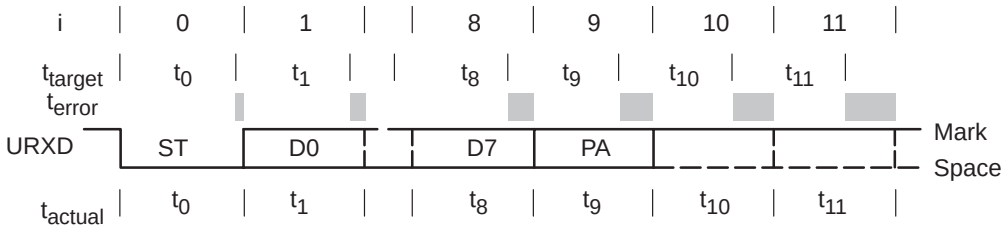


Figure 12–28. MSP430 Transmit Bit Timing Errors



Even small errors per bit (relative errors) can result in large cumulative errors. They must be considered to be cumulative, not relative. The error of an individual bit can be calculated by:

$$Error[\%] = \frac{\sum_{i=0}^{n-1} t_{actual_i} - \sum_{i=0}^{n-1} t_{target_i}}{t_{baud\ rate}} \times 100\%$$

or,

$$Error\ [\%] = \left\{ \frac{baud\ rate}{BRCLK} \times \left[(i+1) \times UBR + \sum_{i=0}^{n-1} m_i \right] - (i+1) \right\} \times 100\%$$

With:

baud rate: Required baud rate
BRCLK: Input frequency – selected for UCLK, ACLK, or SMCLK
i = 0 for the start bit, 1 for the data bit D0, and so on
UBR: Division factor in registers UBR1 and UBR0

Example 12–3. Error Example for 2400 Baud

The following data are assumed:

Baud rate =	2400
BRCLK =	32,768 Hz (ACLK)
UBR =	13, since the ideal division factor is 13.67
m = 6Bh:	m7=0, m6=1, m5=1, m4=0, m3=1, m2=0, m1=1 and m0=1 The LSB (m0) of the modulation register is used first.

$$\text{Start bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((0+1)\times \text{UBR} + 1) - 1\right) \times 100\% = 2.54\%$$

$$\text{Data bit D0 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((1+1)\times \text{UBR} + 2) - 2\right) \times 100\% = 5.08\%$$

$$\text{Data bit D1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((2+1)\times \text{UBR} + 2) - 3\right) \times 100\% = 0.29\%$$

$$\text{Data bit D2 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((3+1)\times \text{UBR} + 3) - 4\right) \times 100\% = 2.83\%$$

$$\text{Data bit D3 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((4+1)\times \text{UBR} + 3) - 5\right) \times 100\% = -1.95\%$$

$$\text{Data bit D4 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((5+1)\times \text{UBR} + 4) - 6\right) \times 100\% = 0.59\%$$

$$\text{Data bit D5 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((6+1)\times \text{UBR} + 5) - 7\right) \times 100\% = 3.13\%$$

$$\text{Data bit D6 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((7+1)\times \text{UBR} + 5) - 8\right) \times 100\% = -1.66\%$$

$$\text{Data bit D7 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((8+1)\times \text{UBR} + 6) - 9\right) \times 100\% = 0.88\%$$

$$\text{Parity bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((9+1)\times \text{UBR} + 7) - 10\right) \times 100\% = 3.42\%$$

$$\text{Stop bit 1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((10+1)\times \text{UBR} + 7) - 11\right) \times 100\% = -1.37\%$$

$$\text{Stop bit 2 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((11+1)\times \text{UBR} + 8) - 12\right) \times 100\% = 1.17\%$$

12.7.2 Typical Baud Rates and Errors

The standard baud rate data needed for the baud rate registers and the modulation register are listed in Table 12–6 for the 32,768-Hz watch crystal (ACLK) and SMCLK, assumed to be 32 times the ACLK frequency. The error listed is calculated for the transmit and receive paths. In addition to the error for the receive operation, the synchronization error must be considered.

Table 12–6. Commonly Used Baud Rates, Baud Rate Data, and Errors

Baud Rate	Divide by		ACLK (32,768 Hz)						MCLK (1,048,576 Hz)				
	ACLK	MCLK	UBR1	UBR0	UMOD	Max. TX Error %	Max. RX Error %	Synchr. RX Error %	UBR1	UBR0	UMOD	Max. TX Error%	Max. RX Error %
75	436.91	13,981	1	B4	FF	–0.1/0.3	–0.1/0.3	±2	36	9D	FF	0/0.1	±2
110	297.89	9532.51	1	29	FF	0/0.5	0/0.5	±3	25	3C	FF	0/0.1	±3
150	218.45	6990.5	0	DA	55	0/0.4	0/0.4	±2	1B	4E	FF	0/0.1	±2
300	109.23	3495.25	0	6D	22	–0.3/0.7	–0.3/0.7	±2	0D	A7	00	–0.1/0	±2
600	54.61	1747.63	0	36	D5	–1/1	–1/1	±2	06	D3	FF	0/0.3	±2
1200	27.31	873.81	0	1B	03	–4/3	–4/3	±2	03	69	FF	0/0.3	±2
2400	13.65	436.91	0	0D	6B	6/3	–6/3	±4	01	B4	FF	0/0.3	±2
4800	6.83	218.45	0	06	6F	–9/11	–9/11	±7	0	DA	55	0/0.4	±2
9600	3.41	109.23	0	03	4A	–21/12	–21/12	±15	0	6D	03	–0.4/1	±2
19,200		54.61							0	36	6B	–0.2/2	±2
38,400		27.31							0	1B	03	–4/3	±2
76,800		13.65							0	0D	6B	–6/3	±4
115,200		9.10							0	09	08	–5/7	±7

The maximum error is calculated for the receive and transmit modes. The receive-mode error is the accumulated time versus the ideal scanning time in the middle of each bit. The transmit error is the accumulated timing error versus the ideal time of the bit period.

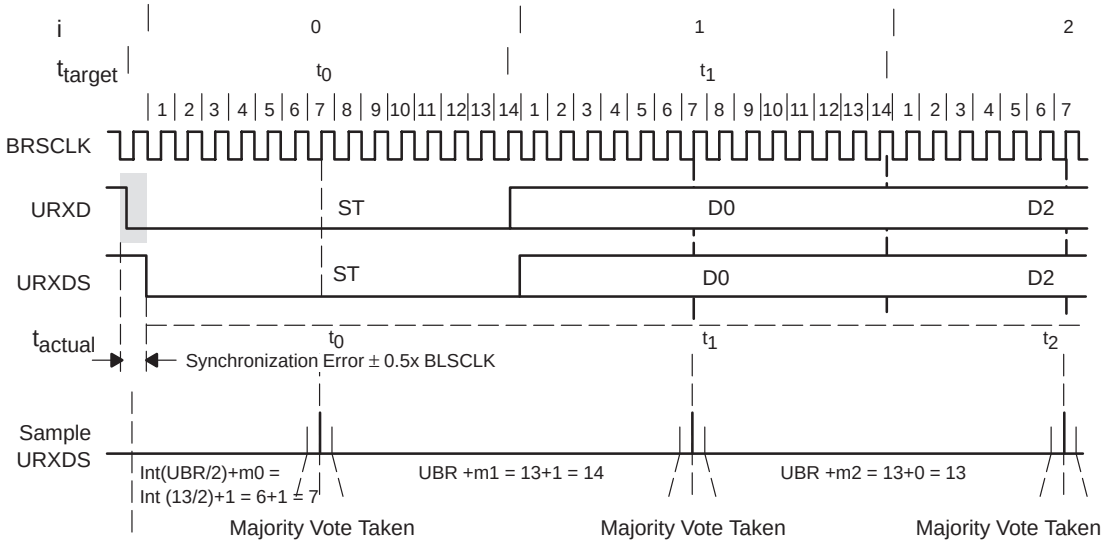
The MSP430 USART peripheral interface allows baud rates nearly as high as the clock rate. It has a low error accumulation as a result of modulating the individual bit timing. In practice, an error margin of 20% to 30% supports standard serial communication.

12.7.3 Synchronization Error

The synchronization error, shown in Figure 12–29, results from the asynchronous timing between the URXD pin data signal and the internal clock system. The receive signal is synchronized with the BRSCLK clock. The BRSCLK clock is sixteen to thirty-one times faster than the bit timing, as described.

BRSCLK = BRCLK	for		N	≤ 1F
BRSCLK = BRCLK/2	for	20h	≤ N	≤ 3Fh
BRSCLK = BRCLK/4	for	40h	≤ N	≤ 7Fh
BRSCLK = BRCLK/8	for	80h	≤ N	≤ FFh
BRSCLK = BRCLK/16	for	100	≤ N	≤ 1FF
BRSCLK = BRCLK/32	for	200	≤ N	≤ 3FFh
BRSCLK = BRCLK/64	for	400	≤ N	≤ 7FFh
BRSCLK = BRCLK/128	for	800h	≤ N	≤ FFFh
BRSCLK = BRCLK/256	for	1000h	≤ N	≤ 1FFFh
BRSCLK = BRCLK/512	for	2000h	≤ N	≤ 3FFFh
BRSCLK = BRCLK/1024	for	4000h	≤ N	≤ 7FFFh
BRSCLK = BRCLK/2048	for	8000h	≤ N	≤ FFFFh

Figure 12–29. Synchronization Error



The target start-bit detection-baud-rate timing $t_{target(0)}$ is half the baud-rate timing $t_{baud\ rate}$ because the bit is tested in the middle of its period. The target baud rate timing $t_{targeti}$ for all of the other succeeding bits is the baud rate timing $t_{baud\ rate}$.

$$Error\ [\%] = \frac{t_{actual_0} + t_{target_0}}{0.5 \times t_{target_0}} + \frac{\sum_{i=1}^{n-1} t_{actual_i} - \sum_{i=1}^{n-1} t_{target_i}}{t_{target_i}} \times 100\%$$

OR

$$Error\ [\%] = \left(\frac{baud\ rate}{BRCLK} \times \left\{ 2 \times \left[m0 + int\left(UBR/2\right) \right] + \left(i \times UBR + \sum_{i=1}^{n-1} m_i \right) \right\} - 1 - i \right) \times 100\%$$

Where:

baud rate is the required baud rate
BRCLK is the input frequency—selected for UCLK, ACLK, or SMCLK
i = 0 for the start bit, 1 for data bit D0, and so on
UBR is the division factor in registers UBR1 and BRB0

Example 12–4. Synchronization Error—2400 Baud

The following data are assumed:

Baud rate =	2400
BRCLK =	32,768 Hz (ACLK)
UBR =	13, since the ideal division factor is 13.67
m = 6Bh:	m7=0, m6=1, m5=1, m4=0, m3=1, m2=0, m1=1 and m0=1 The LSB (m0) of the modulation register is used first.

$$\text{Start bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (0 \times \text{UBR} + 0 - 0)] - 1 \right) \times 100\% = 2.54\%$$

$$\text{Data bit D0 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (1 \times \text{UBR} + 1)] - 1 - 1 \right) \times 100\% = 5.08\%$$

$$\text{Data bit D1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (2 \times \text{UBR} + 1)]–1–2\right) \times 100\% = 0.29\%$$

$$\text{Data bit D2 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (3 \times \text{UBR} + 2)]–1–3\right) \times 100\% = 2.83\%$$

$$\text{Data bit D3 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (4 \times \text{UBR} + 2)]–1–4\right) \times 100\% = –1.95\%$$

$$\text{Data bit D4 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (5 \times \text{UBR} + 3)]–1–5\right) \times 100\% = 0.59\%$$

$$\text{Data bit D5 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (6 \times \text{UBR} + 4)]–1–6\right) \times 100\% = 3.13\%$$

$$\text{Data bit D6 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (7 \times \text{UBR} + 4)]–1–7\right) \times 100\% = –1.66\%$$

$$\text{Data bit D7 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (8 \times \text{UBR} + 5)]–1–8\right) \times 100\% = 0.88\%$$

$$\text{Parity bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (9 \times \text{UBR} + 6)]–1–9\right) \times 100\% = 3.42\%$$

$$\text{Stop bit 1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (10 \times \text{UBR} + 6)]–1–10\right) \times 100\% = –1.37\%$$

$$\text{Stop bit 2 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (11 \times \text{UBR} + 7)]–1–11\right) \times 100\% = –1.17\%$$

Chapter 13

USART Peripheral Interface, SPI Mode

The universal synchronous/asynchronous receive/transmit (USART) serial-communication peripheral supports two serial modes with one hardware configuration. These modes shift a serial-bit stream in and out of the MSP430 at a programmed rate or at a rate defined by an external clock. The first mode is the universal asynchronous-receive/transmit (UART) communication protocol (discussed in Chapter 12); the second is the serial peripheral-interface (SPI) protocol.

Bit SYNC in control register UCTL selects the required mode:

SYNC = 0:	UART—asynchronous mode selected
SYNC = 1:	SPI—synchronous mode selected

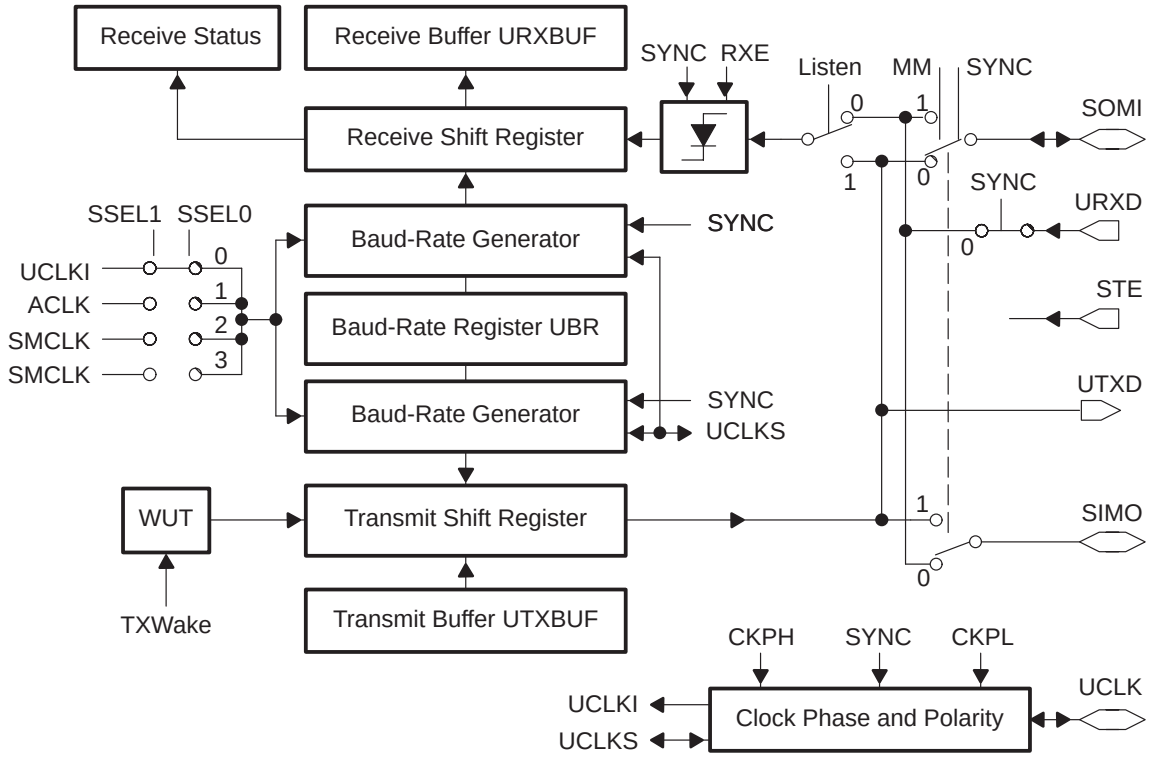
This chapter describes the SPI mode.

Topic **Page**

13.1 USART Peripheral Interface

The USART peripheral interface connects to the CPU as a byte-peripheral module. It connects the MSP430 to the external system environment with three or four external pins. Figure 13–1 shows the USART peripheral-interface module

Figure 13–1. Block Diagram of USART



13.2 USART Peripheral Interface, SPI Mode

The USART peripheral interface is a serial channel that shifts a serial bit stream of 7 or 8 bits in and out of the MSP430. The SPI mode is chosen when control bit SYNC in the USART control register (UCTL) is set.

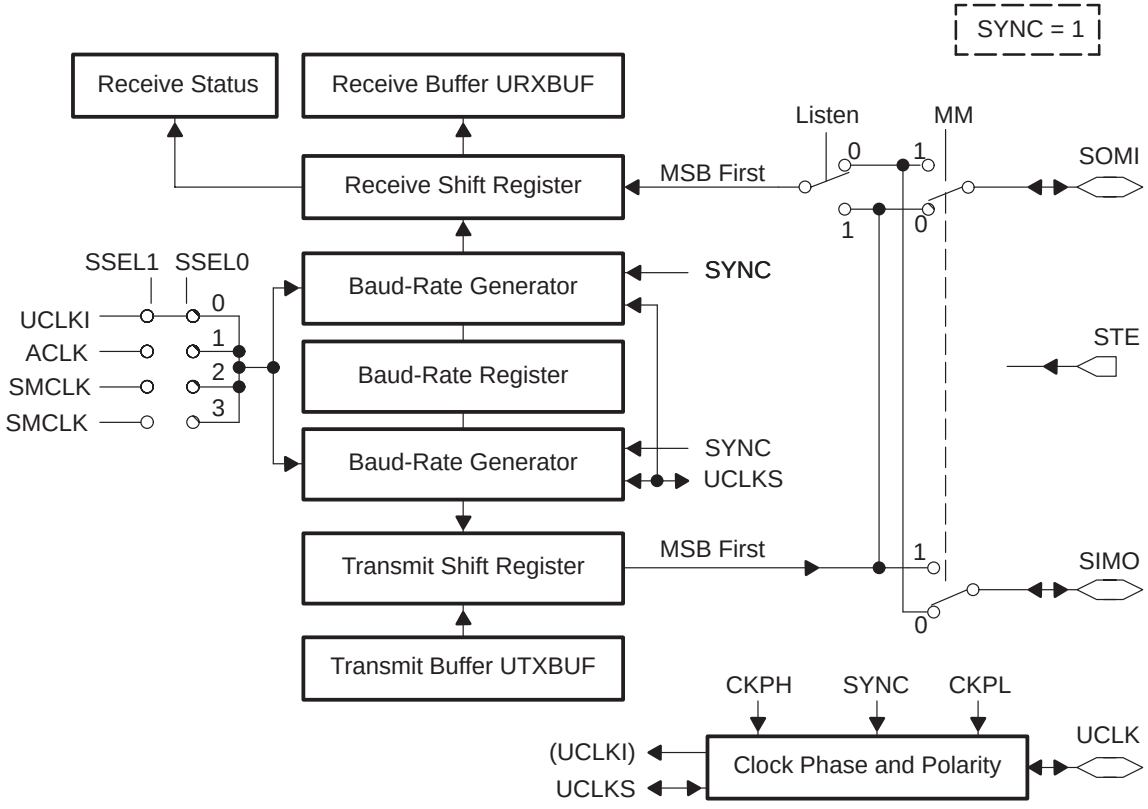
13.2.1 SPI Mode Features

The features of the SPI mode are:

- ❑ Supports three-pin and four-pin SPI operations via SOMI, SIMO, UCLK, and STE
- ❑ Master or slave mode
- ❑ Separate shift registers for receive (URXBUF) and transmit (UTXBUF)
- ❑ Double buffers for receiving and transmitting
- ❑ Has clock-polarity and clock-phase control
- ❑ Has clock-frequency control in master mode
- ❑ Supports a character length of seven or eight bits per character

Figure 13–2 shows the USART module in SPI mode.

Figure 13–2. Block Diagram of USART—SPI Mode



13.3 Synchronous Operation

In USART synchronous mode, data and clock signals transmit and receive serial data. The master supplies the clock and data. The slaves use this clock to shift serial information in and out.

The four-pin SPI mode also uses a control line to enable a slave to receive and transmit data. The line is controlled by the master.

Three or four signals are used for data exchange:

- SIMO Slave in, master out
  The direction is defined by SIMODIR (SIMODIR=0, input direction) SIMODIR = [SYNC .and. MM .and. (STC .or. STE)]
  Output direction is selected when SPI + Master Mode is selected.
  When 4-pin SPI is selected (STC=0) input direction is forced by a low level on external STE pin.

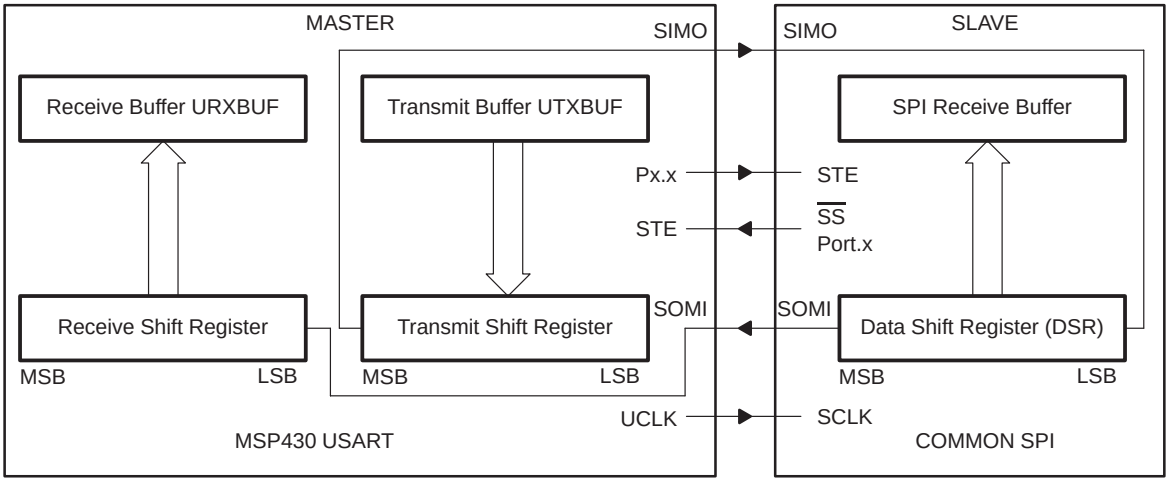
- SOMI Slave out, master in
  The direction is defined by SOMIDIR (SIMODIR=0 input direction) SOMIDIR = [SYNC .and. .not.(MM)] .or. [STC .or. .not.(STE)]
  Output direction is selected when SPI + Slave Mode is selected.
  When 4-pin SPI is selected (STC=0) input direction is forced by a low level on external STE pin.

- UCLK USART clock. The master drives this signal and the slave uses it to receive and transmit data.
  The direction is defined by UCLKDIR (UCLKDIR=0 input direction) UCLKDIR = [SYNC .and. MM .and. (STC .or. STE)]
  Output direction is selected when SPI + Master Mode is selected.
  When 4-pin SPI is selected (STC=0) input direction is forced by a low level on external STE pin.

- STE Slave transmit enable. Used in four-pin mode to control more than one slave in a multiple master and slave system.

The interconnection of the USART in synchronous mode to another device's serial port with one common transmit/receive shift register is shown in Figure 13–3, where MSP430 is master or slave. The operation of both devices is identical.

Figure 13–3. MSP430 USART as Master, External Device With SPI as Slave



The master initiates the transfer by sending the UCLK signal. For the master, data is shifted out of the transmit shift register on one clock edge, and shifted into the receive shift register on the opposite edge. For the slave, the data shifting operation is the same and uses one common register for transmitting and receiving data. Master and slave send and receive data at the same time.

Whether the data is meaningful or dummy data depends on the application software:

- Master sends data and slave sends dummy data
- Master sends data and slave sends data
- Master sends dummy data and slave sends data

Figures 13–4 and 13–5 show an example of a serial synchronous data transfer for a character length of seven bits. The initial content of the receive shift register is 00. The following events occur in order:

A) Slave writes 98h to the data shift register (DSR) and waits for the master to shift data out.

B) Master writes B0h to UTXBUF, which is immediately transferred to the transmit shift register, and starts the transmission.

C) First character is finished and sets the interrupt flags.

D) Slave reads 58h from the receive buffer (right justified).

E) Slave writes 54h to the DSR and waits for the master to shift out data.

F) Master reads 4Ch from the receive buffer URXBUF (right justified).

G) Master writes E8h to the transmit buffer UTXBUF and starts the transmission.

Note: If USART is in slave mode, no UCLK is needed after D), until G). However, in master mode, two clocks are used internally (not on UCLK signal) to end transmit/receive of first character and prepare the transmit/receive of the next character.

H) Second character is finished and sets the interrupt flag.

I) Master receives 2Ah and slave receives 74h (right justified).

Figure 13–4. Serial Synchronous Data Transfer

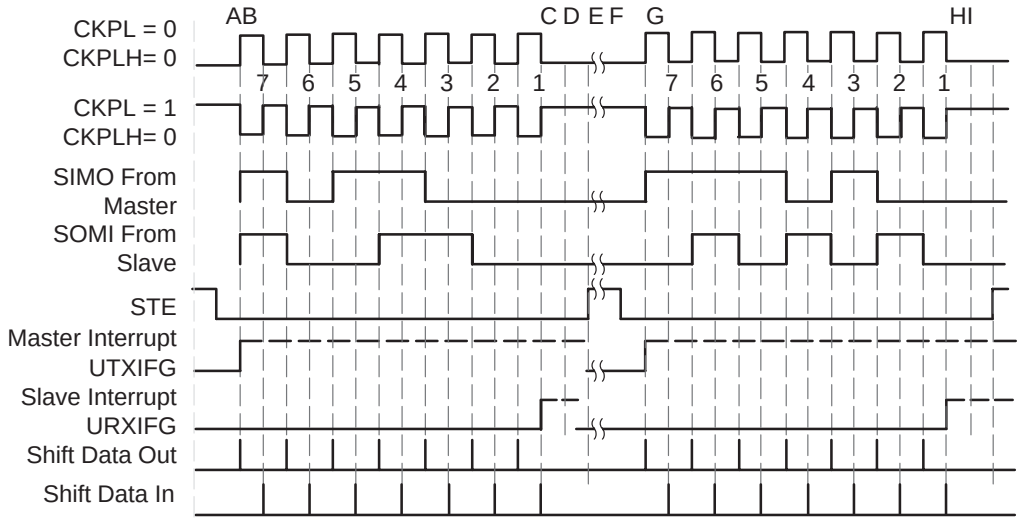
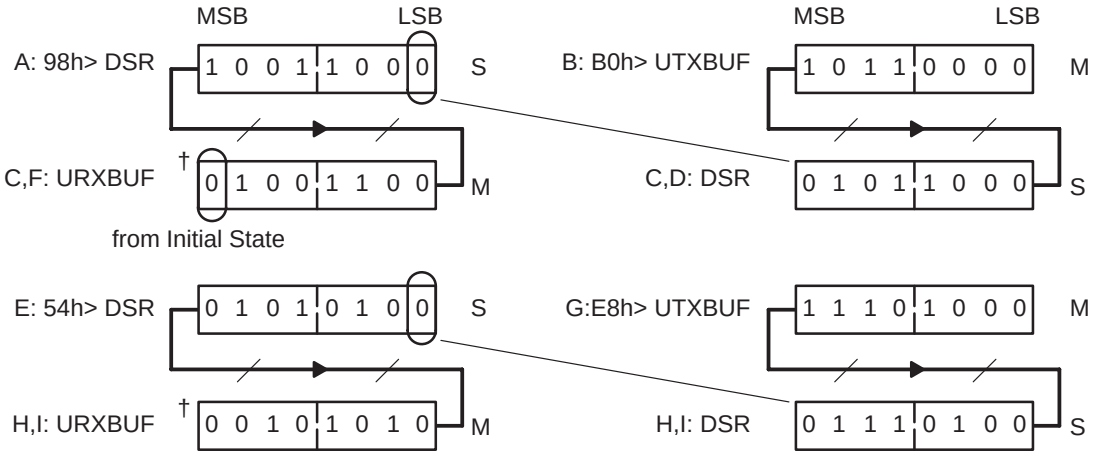


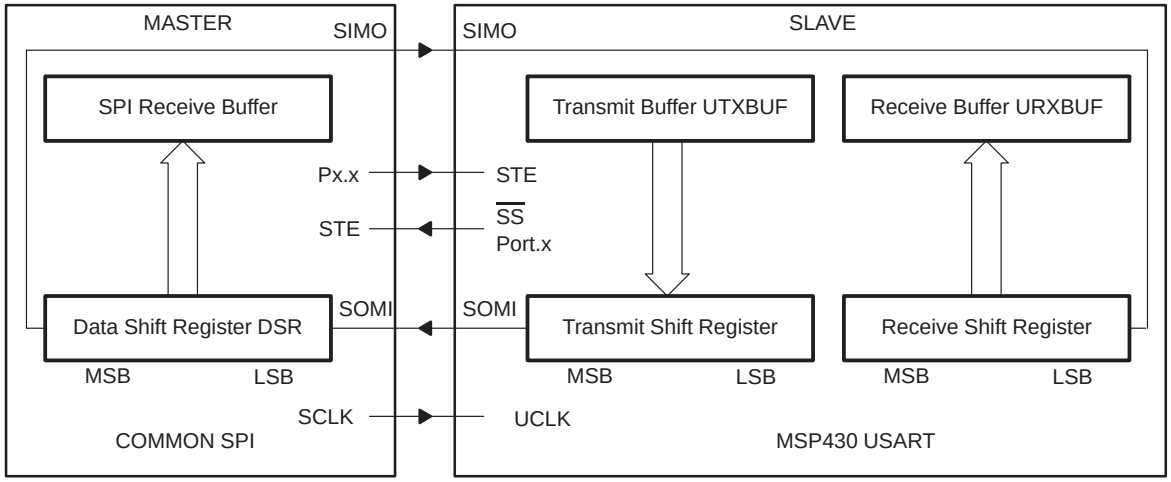
Figure 13–5. Data Transfer Cycle



† In 7 bit mode, the MSB of RXBUF is always read as 0.
S: Slave M: Master

Figure 13–6 illustrates the USART module functioning as a slave in a three or four-pin SPI configuration.

Figure 13–6. MSP430 USART as Slave in Three-Pin or Four-Pin Configuration



13.3.1 Master SPI Mode

The master mode is selected when the master-mode bit (MM) in control register UCTL is set. The USART module controls the serial-communication network by providing UCLK at the UCLK pin. Data is output on the SIMO pin during the first UCLK period and latched from the SOMI pin in the middle of the corresponding UCLK period.

The data written to the transmit buffer (UTXBUF) is moved to the transmit shift register as soon as the shift register is empty. This initiates the data transfer on the SIMO pin starting with the most-significant bit. At the same time, received data is shifted into the receive shift register and, upon receiving the selected number of bits, the data is transferred to the receive buffer (URXBUF) setting the receive interrupt flag (URXIFG). Data is shifted into the receive shift register starting with the most-significant bit. It is stored and right-justified in the receive buffer (URXBUF). When previous data is not read from the receive buffer (URXBUF), the overrun error bit (OE) is set.

Note: USART Synchronous Master Mode, Receive Initiation

The master writes data to the transmit buffer UTXBUF to receive a character. The receive starts when the transmit shift register is empty and the data is transferred to it. Receive and transmit operations always take place together, at opposite clock edges.

The protocol can be controlled using the transmit-interrupt flag UTXIFG, or the receive-interrupt flag URXIFG. By using UTXIFG immediately after sending the shift-register data to the slave, the buffer data is transferred to the shift register and the transmission starts. The slave receive timing should ensure that there is a timely pick-up of the data. The URXIFG flag indicates when the data shifts out and in completely. The master can use URXIFG to ensure that the slave is ready to correctly receive the next data.

13.3.1.1 Four-Pin SPI Master Mode

The signal on STE is used by the active master to prevent bus conflicts with another master. The STE pin is an input when the corresponding PnSEL bit (in the I/O registers) selects the module function. The master operates normally while the STE signal is high. Whenever the STE signal is low, for example, when another device makes a request to become master, the actual master reacts such that:

- The pins that drive the SPI bus lines SIMO and UCLK are set to inputs.
- The error bit FE and the interrupt flag URXIFG in register URCTL are set.

The bus conflict is then removed: SIMO and UCLK do not drive the bus lines, and the error flag indicates the system integrity violation to the software. Pins SIMO and UCLK are forced to the input state while STE is in a low state, and they return to the conditions defined by the corresponding control bits when STE returns to a high state.

In the three-pin mode, the STE input signal is not relevant.

13.3.2 Slave SPI Mode

The slave mode is selected when bit MM of the control register is reset and synchronous mode is selected.

The UCLK pin is used as the input for the serial-shift clock supplied by an external master. The data-transfer rate is determined by this clock and not by the internal bit-rate generator. The data, loaded into the transmit shift register through the transmit buffer (UTXBUF) before the start of UCLK, is transmitted on the SOMI pin using the UCLK supplied from the master. Simultaneously, the serial data applied to the SIMO pin are shifted into the receive shift register on the opposite edge of the clock.

The receive-interrupt flag URXIFG indicates when the data is received and transferred into the receive buffer. The overrun-error bit is set when the previously-received data is not read before the new data is written to the receive buffer.

13.3.2.1 Four-Pin SPI Slave Mode

In the four-pin SPI mode, the STE signal is used by the slave to enable the transmit and receive operations. It is applied from the SPI master. The receive and transmit operations are disabled when the STE signal is high, and enabled when it is low. Whenever the STE signal becomes high, any receive operation in progress is halted and then continues when the STE signal is low again. The STE signal enables one slave to access the data lines. The SOMI is input if STE is set high.

13.4 Interrupt and Control Functions

The USART peripheral interface serves two main interrupt sources for transmission and reception. Two interrupt vectors serve receive and transmit interrupt events.

The interrupt control bits and flags and enable bits of the USART peripheral interface are located in the SFR address range. The bit functions are described below in Table 13–1. See the peripheral-file map in Appendix A for the exact bit locations.

Table 13–1. USART Interrupt Control and Enable Bits—SPI Mode

Receive interrupt flag	URXIFG	Initial state reset (by PUC/SWRST)
Receive interrupt enable	URXIE	Initial state reset (by PUC/SWRST)
Receive/transmit enable (see Note)	USPIE	Initial state reset (by PUC)
Transmit interrupt flag	UTXIFG	Initial state set (by PUC/SWRST)
Transmit interrupt enable	UTXIE	Initial state reset (by PUC/SWRST)

Note: Different for UART mode, see Chapter 12.

The USART receiver and transmitter operate in parallel and use the same baud-rate generator in synchronous master mode. In synchronous slave mode, the external clock applied to UCLK is used for the receiver and the transmitter. The receiver and transmitter are enabled and disabled together with the USPIE bit.

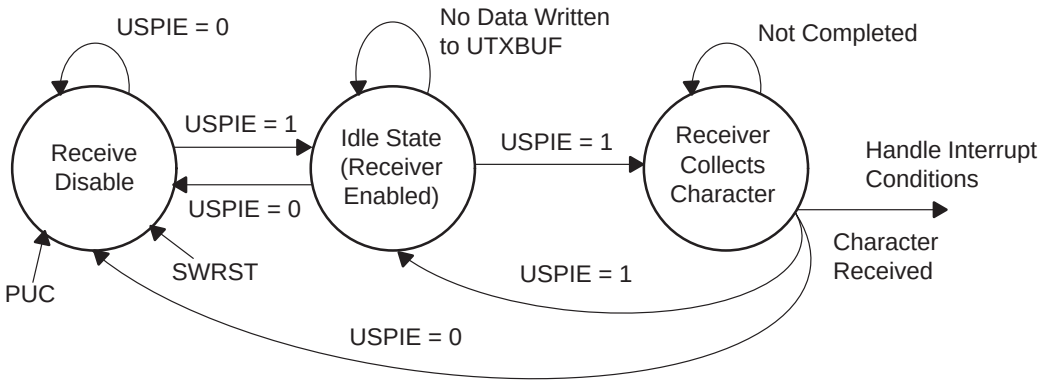
13.4.1 USART Receive/Transmit Enable Bit, Receive Operation

The receive/transmit enable bit (USPIE) enables or disables collection of the bit stream on the URXD/SOMI data line. Disabling the USART receiver (USPIE = 0) stops the receive operation after completion, or stops a pending operation if no receive operation is active. In synchronous mode, UCLK does not shift any data into the receiver shift register.

13.4.1.1 Receive/Transmit Enable Bit—MSP430 as Master

The receive operation functions identically for three-pin and four-pin modes, as shown in Figure 13–7, when the MSP430 USART is selected to be the SPI master.

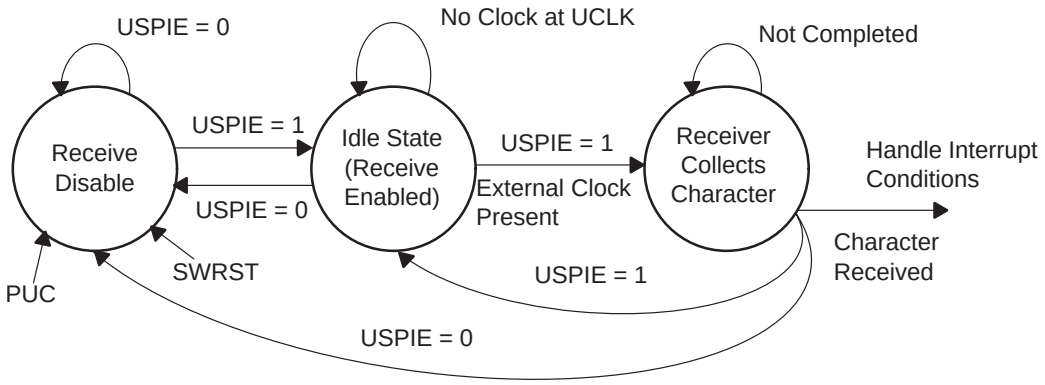
Figure 13–7. State Diagram of Receiver Enable Operation—MSP430 as Master



13.4.1.2 Receive/Transmit Enable Bit—MSP430 as Slave, Three-Pin Mode

The receive operation functions differently for three-pin and four-pin modes when the MSP430 USART module is selected to be the SPI slave. In the three-pin mode, shown in Figure 13–8, no external SPI receive-control signal stops an active receive operation. A PUC signal, a software reset (SWRST), or a receive/transmit enable (USPIE) signal can stop a receive operation and reset the USART.

Figure 13–8. State Diagram of Receive/Transmit Enable—MSP430 as Slave, Three-Pin Mode



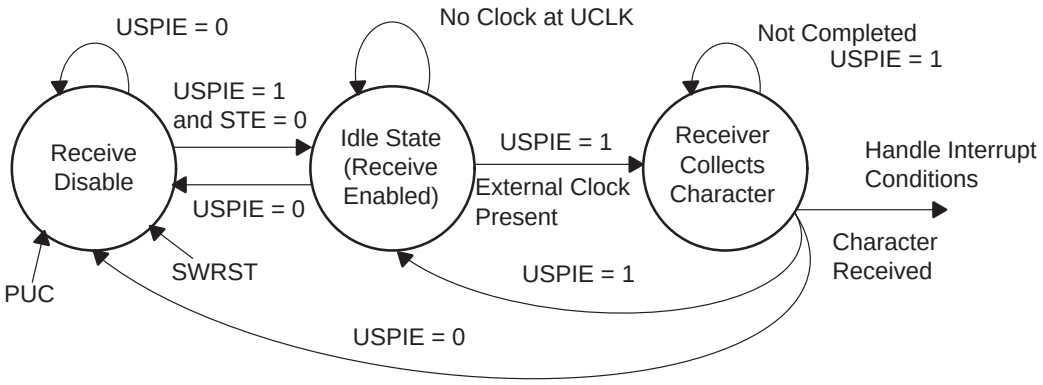
Note: USPIE Re-enabled, SPI Mode

After the receiver is completely disabled, a reenabling of the receiver is asynchronous to any data stream on the communication line. Synchronization to the data stream is handled by the software protocol in three-pin SPI mode.

13.4.1.3 Receive/Transmit Enable Bit—MSP430 as Slave, Four-Pin Mode

In the four-pin mode, shown in Figure 13–9, the external SPI receive-control signal applied to pin STE stops a started receive operation. A PUC signal, a software reset (SWRST), or a receive/transmit enable (USPIE) can stop a receive operation and reset the operation-control state machine. Whenever the STE signal is set to high, the receive operation is halted.

Figure 13–9. State Diagram of Receive Enable—MSP430 as Slave, Four-Pin Mode



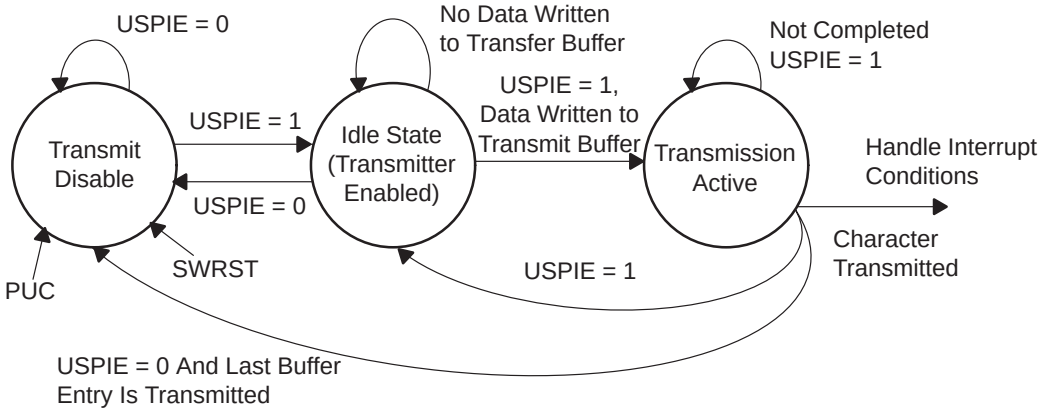
13.4.2 USART Receive/Transmit Enable Bit, Transmit Operation

The receive/transmit enable bit USPIE, shown in Figures 13–10 and 13–11, enables or disables the shifting of a character on the serial data line. If this bit is reset, the transmitter is disabled, but any active transmission does not halt until all data previously written to the transmit buffer is transmitted. If the transmission is completed, any further write operation to the transmitter buffer does not transmit. When the UTXBUF is ready, any pending request for transmission remains, which results in an immediate start of transmission when USPIE is set and the transmitter is empty. A low state on the STE signal removes the active master (four-pin mode) from the bus. It also indicates that another master is requesting the active-master function.

13.4.2.1 Receive/Transmit Enable—MSP430 as Master

Figure 13–10 shows the transmit-enable activity when the MSP430 is master.

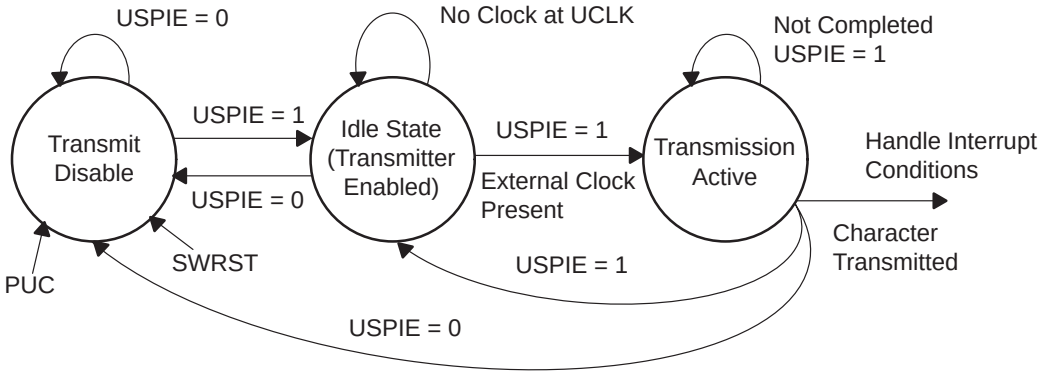
Figure 13–10. State Diagram of Transmit Enable—MSP430 as Master



13.4.2.2 Receive/Transmit Enable, MSP430 is Slave

Figure 13–11 shows the receive/transmit-enable-bit activity when the MSP430 is slave.

Figure 13–11. State Diagram of Transmit Enable—MSP430 as Slave



When USPIE is reset, any data can be written regularly into the transmit buffer, but no transmission is started. Once the USPIE bit is set, the data in the transmit buffer are immediately loaded into the transmit shift register and character transmission is started.

Note: Writing to UTXBUF, SPI Mode

Data should never be written to transmit buffer UTXBUF when the buffer is not ready (UTXIFG = 0) and the transmitter is enabled (USPIE is set). If data is written, character shifting can be random.

Note: Write to UTXBUF/Reset of Transmitter, SPI Mode

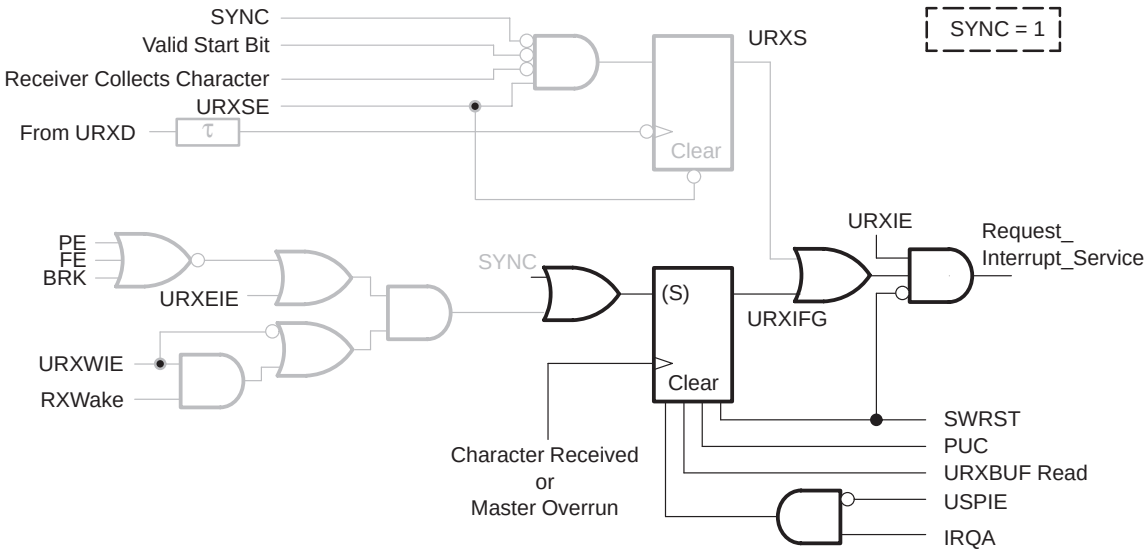
Disabling of the transmitter should be done only if all data to be transmitted have been moved to the transmit shift register.

```
MOV.B  #....,&UTXBUF
BIC.B  #USPIE,&ME2       ; If BITCLK < MCLK then the
                         ; transmitter might be stopped
                         ; before the buffer is loaded
                         ; into the transmitter
                         ; shift register
```

13.4.3 USART Receive-Interrupt Operation

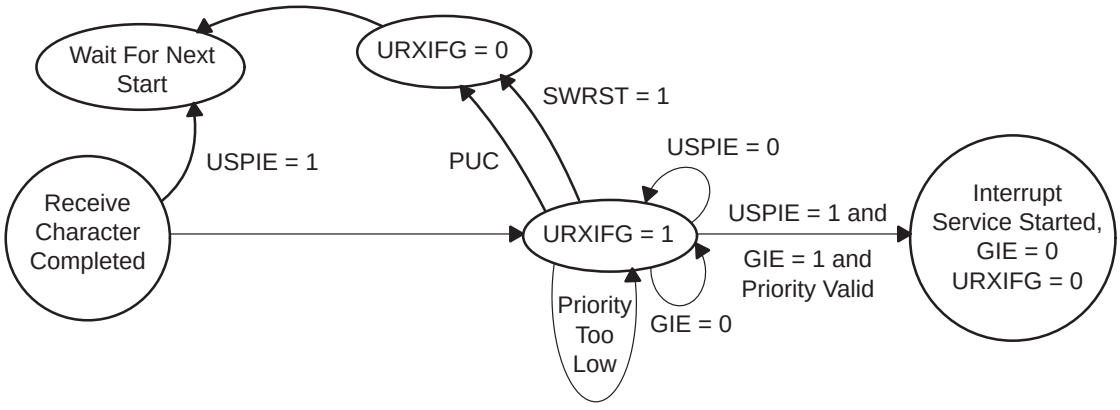
In the receive-interrupt operation shown in Figure 13–12, the receive-interrupt flag URXIFG is set each time a character is received and loaded into the receive buffer.

Figure 13–12. Receive Interrupt Operation



URXIFG is reset by a system reset PUC signal, or by a software reset (SWRST). URXIFG is reset automatically if the interrupt is served or the receive buffer URXBUF is read. The receive interrupt enable bit (USPIE), if set, enables a CPU interrupt request as shown in Figure 13–13. The receive interrupt flag bits URXIFG and USPIE are reset with a PUC signal or a SWRST.

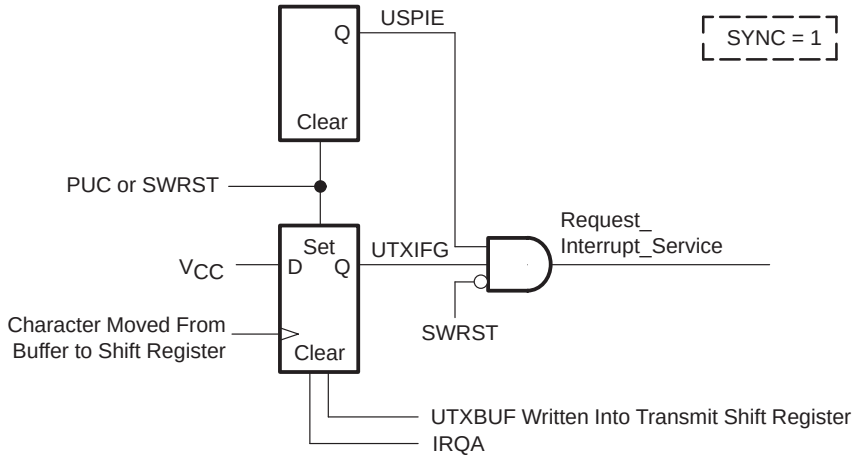
Figure 13–13. Receive Interrupt State Diagram



13.4.4 Transmit-Interrupt Operation

In the transmit-interrupt operation shown in Figure 13–14, the transmit-interrupt flag UTXIFG is set by the transmitter to indicate that the transmitter buffer UTXBUF is ready to accept another character. This bit is automatically reset if the interrupt-request service is started or a character is written to the UTXBUF. This flag activates a transmitter interrupt if bits USPIE and GIE are set. The UTXIFG is set after a system reset PUC signal, or removal of SWRST.

Figure 13–14. Transmit-Interrupt Operation



The transmit-interrupt enable bit UTXIE controls the ability of the UTXIFG to request an interrupt, but does not prevent the UTXIFG flag from being set. The USPIE is reset with a PUC signal or a SWRST. The UTXIFG bit is set after a system reset PUC signal or a SWRST, but the USPIE bit is reset to ensure full interrupt-control capability.

13.5 Control and Status Registers

The USART registers, shown in Tables 13–2 and 13–3, are byte structured and should be accessed using byte instructions.

Table 13–2. USART0 Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USART control	UCTL0	Read/write	070h	See Section 13.5.1
Transmit control	UTCTL0	Read/write	071h	See Section 13.5.2
Receive control	URCTL0	Read/write	072h	See Section 13.5.3
Modulation control	UMCTL0	Read/write	073h	Unchanged
Baud rate 0	UBR00	Read/write	074h	Unchanged
Baud rate 1	UBR10	Read/write	075h	Unchanged
Receive buffer	URXBUF0	Read/write	076h	Unchanged
Transmit buffer	UTXBUF0	Read	077h	Unchanged

Table 13–3. USART1 Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USART control	UCTL1	Read/write	078h	See Section 13.5.1
Transmit control	UTCTL1	Read/write	079h	See Section 13.5.2
Receive control	URCTL1	Read/write	07Ah	See Section 13.5.3
Modulation control	UMCTL1	Read/write	07Bh	Unchanged
Baud rate 0	UBR01	Read/write	07Ch	Unchanged
Baud rate 1	UBR11	Read/write	07Dh	Unchanged
Receive buffer	URXBUF1	Read/write	07Eh	Unchanged
Transmit buffer	UTXBUF1	Read	07Fh	Unchanged

All bits are random following the PUC signal, unless otherwise noted by the detailed functional description.

Reset of the USART module is performed by the PUC signal or a SWRST. After a PUC signal, the SWRST bit remains set and the USART module remains in the reset condition. It is disabled by resetting the SWRST bit. The SPI mode is disabled after the PUC signal.

The USART module operates in asynchronous or synchronous mode as defined by the SYNC bit. The bits in the control registers can have different functions in the two modes. All bits are described with their function in the synchronous mode—SYNC = 1. Their function in the asynchronous mode is described in Chapter 12.

13.5.1 USART Control Register

The information stored in the control register, shown in Figure 13–15, determines the basic operation of the USART module. The register bits select the communication mode and the number of bits per character. All bits should be programmed to the desired mode before resetting the SWRST bit.

Figure 13–15. USART Control Register

UCTL0, 070h
UCTL1, 078h

7							0
Unused	Unused	Unused	CHAR	Listen	SYNC	MM	SWRST
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–1

Bit 0: The USART state machines and operating flags are initialized to the reset condition (URXIFG=USPIE=0, UTXIFG=1) if the software reset bit is set. Until the SWRST bit is reset, all affected logic is held in the reset state. This implies that after a system reset the USART must be reenabled by resetting this bit.

Bit 1: Master mode is selected when the MM bit is set. The USART module slave mode is selected when the MM bit is reset.

Bit 2: Peripheral module mode select
The SYNC bit sets the function of the USART peripheral-interface module. Some of the USART control bits have different functions in UART and SPI modes.
SYNC = 0: UART function is selected
SYNC = 1: SPI function is selected

Bit 3: The listen bit determines the transmitted data to feed back internally to the receiver. This is commonly called loopback mode.

Bit 4: Character length
This register bit sets the length of the character to be transmitted as either seven or eight bits.
CHAR = 0: 7-bit data
CHAR = 1: 8-bit data

Bit 5: Unused

Bit 6: Unused

Bit 7: Unused

13.5.2 Transmit Control Register UTCTL

The transmit control register (UTCTL), shown in Figure 13–16, controls the USART hardware associated with transmitter operations.

Figure 13–16. Transmit Control Register UTCTL

UTCTL0, 071h
UTCTL1, 079h

7							0
CKPH	CKPL	SSEL1	SSEL0	Unused	Unused	STC	TXEPT
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–1

Bit 0: Master mode:
The transmitter-empty flag TXEPT is set when the transmitter shift register and UTXBUF are empty, and reset when data are written to UTXBUF. It is set again by a SWRST.

Slave mode:
The transmitter-empty flag TXEPT is not set when the transmitter shift register and UTXBUF are empty.

Bit 1: The slave transmit-control bit STC selects if the STE pin is used for master and slave mode:

STC = 0: The four-pin mode of SPI is selected. The STE signal is used by the master to avoid bus conflicts, or is used in slave mode to control transmit and receive enable.

STC = 1: The three-pin SPI mode is selected. STE is not used in master or slave mode.

Bit 2: Unused

Bit 3: Unused

Bits 4, 5: Source select 0 and 1
The source-select bits define which clock source is used for baud-rate generation only when master mode is selected:

SSEL1,SSEL0	0	External clock UCLK selected
	1	Auxiliary clock ACLK selected
	2, 3	SMCLK selected

In master mode (MM = 1), an external clock at UCLK cannot be selected since the master supplies the UCLK signal for any slave. In slave mode, bits SSEL1 and SSEL0 are not relevant. The external clock UCLK is always used.

Bits 6, 7: Clock polarity CKPL and clock phase CKPH
The CKPL bit controls the polarity of the SPICLK signal.

CKPL = 0: The inactive level is low; data is output with the rising edge of UCLK; input data is latched with the falling edge of UCLK.

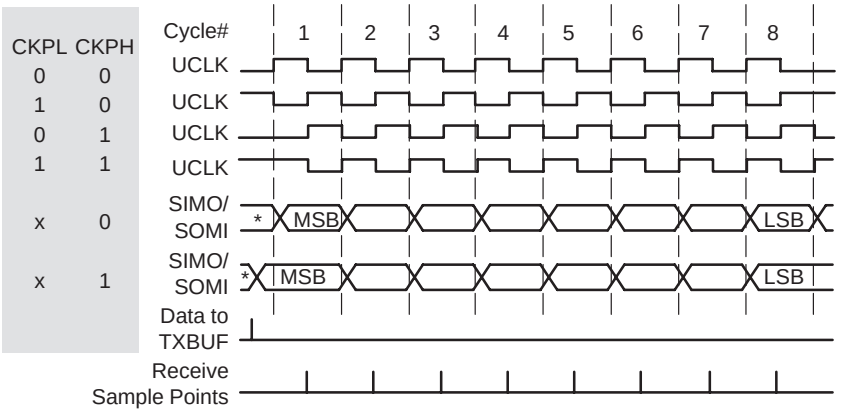
CKPL = 1: The inactive level is high; data is output with the falling edge of UCLK; input data is latched with the rising edge of SPICLK.

The CKPH bit controls the polarity of the SPICLK signal as shown in Figure 13–17.

CKPH = 0: Normal UCLK clocking scheme

CKPH = 1: UCLK is delayed by one half cycle

Figure 13–17. USART Clock Phase and Polarity



When operating with the CKPH bit set, the USART (synchronous mode) makes the first bit of data available after the transmit shift register is loaded and before the first edge of the UCLK. In this mode, data is latched on the first edge of UCLK and transmitted on the second edge.

13.5.3 Receive Control Register URCTL

The receive control register (URCTL), shown in Figure 13–18, controls the USART hardware associated with the receiver operation and holds error conditions.

Figure 13–18. Receive Control Register URCTL

URCTL0, 072h
URCTL1, 07Ah

7							0
FE	Undef.	OE	Undef.	Unused	Unused	Undef.	Undef.
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

Bit 0: Undefined, driven by USART hardware

Bit 1: Undefined, driven by USART hardware

Bit 2: Unused

Bit 3: Unused

Bit 4: Undefined, driven by USART hardware

Bit 5: The overrun-error-flag bit (OE) is set when a character is transferred to URXBUF before the previous character is read. The previous character is overwritten and lost. OE is reset by a SWRST, a system reset, by reading the URXBUF, or by an instruction.

Bit 6: Undefined, driven by USART hardware

Bit 7: Frame error. The FE bit is set when four-pin mode is selected and a bus conflict stops an active master by applying a negative

transition signal to pin STE. FE is reset by a SWRST, a system reset, by reading the URXBUF, or by an instruction.

13.5.4 Baud-Rate Select and Modulation Control Registers

The baud-rate generator uses the content of baud-rate select registers UBR1 and UBR0, shown in Figure 13–19, to generate the serial-data-stream bit timing. The smallest division factor is two.

Figure 13–19. USART Baud-Rate Select Register

UBR00, 074h
UBR01, 07Ch

7							0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
rw	rw	rw	rw	rw	rw	rw	rw

UBR10, 075h
UBR11, 07Dh

7							0
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
rw	rw	rw	rw	rw	rw	rw	rw

$$\text{Baud rate} = \frac{BRCLK}{UBR + \frac{1}{n}\sum_{i}^{n} mi} \quad \text{with UBR= [UBR1,UBR0]}$$

The maximum baud rate that can be selected for transmission in master mode is half of the clock-input frequency of the baud-rate generator. In slave mode, the rate is determined by the external clock applied to UCLK.

The modulation control register, shown in Figure 13–20, is not used for serial synchronous communication. It is best kept in reset mode (bits m0 to m7 = 0).

Figure 13–20. USART Modulation Control Register

UMCTL0, 073h
UMCTL1, 07Bh

7							0
m7	m6	m5	m4	m3	m2	m1	m0
rw	rw	rw	rw	rw	rw	rw	rw

13.5.5 Receive Data Buffer URXBUF

The receive data buffer (URXBUF), shown in Figure 13–21, contains previous data from the receiver shift register. URXBUF is cleared with a SWRST or a PUC signal. Reading URXBUF resets the receive-error bits and the receive-interrupt flag URXIFG.

Figure 13–21. Receive Data Buffer URXBUF

URXBUF0, 076h
URXBUF1, 07Eh

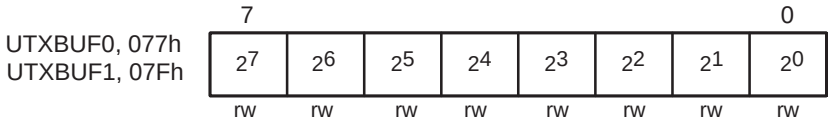
7							0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
rw	rw	rw	rw	rw	rw	rw	rw

The MSB of the URXBUF is always reset in seven-bit-length mode.

13.5.6 Transmit Data Buffer UTXBUF

The transmit data buffer (UTXBUF), shown in Figure 13–22, contains current data for the transmitter to transmit.

Figure 13–22. Transmit Data Buffer UTXBUF



The UTXIFG bit indicates that UTXBUF is ready to accept another character for transmission. In master mode, the transmission is initialized by writing data to UTXBUF. The transmission of this data is started immediately if the transmit shift register is empty.

When seven-bit character-length is used, the data moved into the transmit buffer must be left-justified since the MSB is shifted out first.

Chapter 14

Comparator_A

The Comparator_A peripheral module is used to compare analog signals to support various forms of analog-to-digital conversion.

The Comparator_A module includes:

- Comparator with on/off capability and no input hysteresis
- Internal analog voltage reference generator
- Internal reference levels available externally
- Input multiplexer to exchange the comparator terminals
- Software-selectable RC-filter at the comparator output
- One interrupt vector

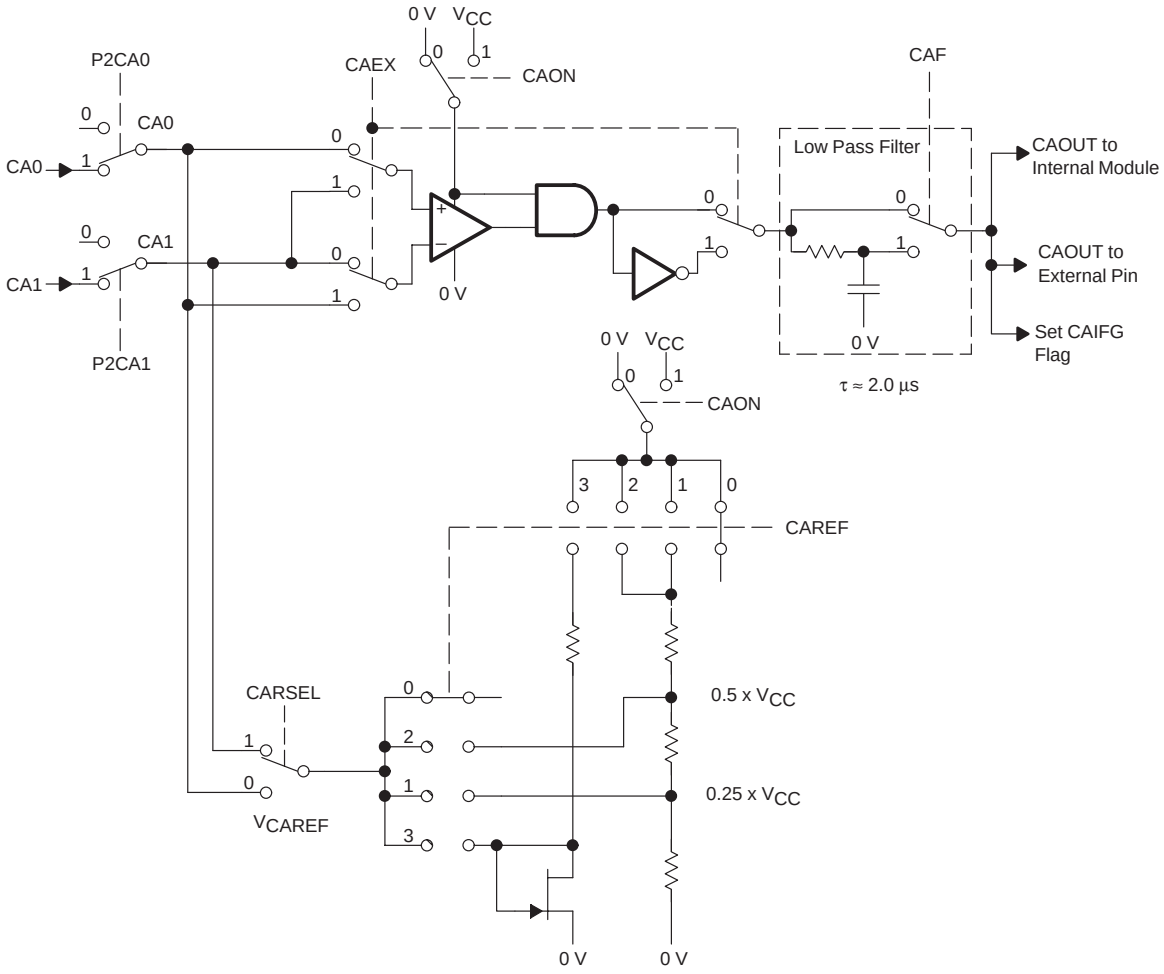
The Comparator_A is implemented in MSP430x11x1, MSP430x13x, and MSP430x14x devices.

Topic **Page**

14.1 Comparator_A Overview

The primary function of the comparator module is to support precision A/D slope-conversion applications, battery-voltage supervision, and monitoring of external analog signals. The comparator is controlled via twelve control bits in registers CACTL1 and CACTL2.

Figure 14–1. Schematic of Comparator_A



The input and output pins of Comparator_A are often multiplexed with other pin functions on the MSP430. Additionally, the internal connections to Comparator_A can differ among MSP430 devices. The data sheet of a desired device should always be consulted to determine the specific connection implementations.

14.2 Comparator_A Description

The comparator_A peripheral module is comprised of several major blocks. These blocks are described in this section.

14.2.1 Input Analog Switches

The input analog switches connect or disconnect the comparator input terminals to associated port pins using the P2CA0 and P2CA1 control bits. Both terminal inputs can be controlled individually. P2CA0 and P2CA1 allow:

- Application of an external signal to the + and – terminals of the comparator, or
- Routing of an internal reference voltage (if applied) to a comparator-input terminal as an output on an associated port pin. In this way, the internal reference voltage can be used to bias external circuitry.

Internally, the input switch is constructed as a T-switch to suppress distortion in the signal path. When a comparator terminal is not connected to an external pin, it should be connected to an internal reference-voltage level.

Note:

Ensure that the comparator input terminals are connected to signal, power, or ground level. Otherwise, floating levels may cause unexpected interrupts and current consumption may increase.

14.2.2 Input Multiplexer

Control bit CAEX controls the input multiplexer to select which input signals are connected to the comparator's + and – terminals. Additionally, when the comparator terminals are exchanged, the output signal from the comparator is inverted. This allows the user to determine or compensate for the comparator offset.

14.2.3 The Comparator

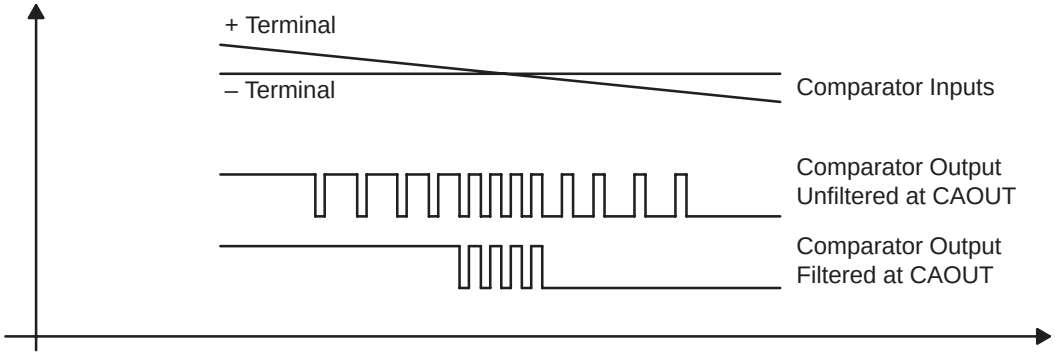
The comparator compares the analog voltages at the + and – input terminals. If the + terminal is more positive than the – terminal, the comparator output will be high (note that the value of signal CAOUT also depends on the value of CAEX). The comparator can be switched on or off using control bit CAON. The comparator should be switched off when not in use to stop its current consumption. When the comparator is switched off, the output is low (note that the value of CAOUT still depends on the value of CAEX, even when the comparator is off).

14.2.4 The Output Filter

The output of the comparator can be used with or without internal filtering. When control bit CAF is set, the output is filtered with an-chip RC-filter. The filter is bypassed when CAF is reset.

A comparator output will oscillate if the voltage difference across the input terminals is small. Internal and external parasitic effects and cross coupling on and between signal lines, power-supply lines, and other parts of the system are responsible for this behavior (see Figure 14–2). The comparator output oscillation reduces accuracy and resolution of the comparison result. Selecting the output filter can reduce errors associated with comparator oscillation.

Figure 14–2. RC-Filter Response at the Output of the Comparator



14.2.5 The Voltage Reference Generator

The voltage reference generator is used to generate V_{CAREF}. V_{CAREF} can be applied to either of the comparator input terminals. Control bits CAREF0 and CAREF1 control the output of the voltage generator. Control bit CARSEL selects the comparator terminal to which V_{CAREF} is applied. If external signals are applied to the comparator input terminals, the internal reference generator should be shut off to reduce current consumption. The divider in the voltage reference generator can generate a fraction of the device's V_{CC}, or a fixed transistor-threshold voltage. This threshold voltage tolerance is specified in the specific device's data sheet.

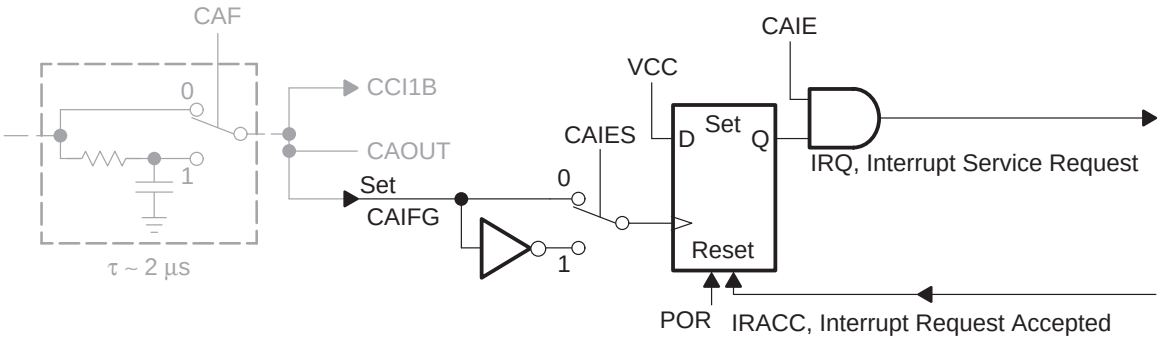
Ratiometric measurement principles that compare unknown values, such as resistive or capacitive sensors, with a known value such as a precision resistor or capacitor, can use an internal reference and achieve accurate results without an absolute V_{CC}. V_{CC} needs to be stable, but not necessarily known. The accuracy of ratiometric measurements is determined by the accuracy of the known resistor or capacitor value.

Absolute measurement principles require a stable V_{CC} to ensure that the voltage reference generated produces accurate reference-voltage levels.

14.2.6 Comparator_A Interrupt Circuitry

One interrupt and one interrupt vector are associated with the Comparator_A (see Figure 14–3). The interrupt flag CAIFG is set on either the rising or falling edge of the comparator output. The interrupt edge-select bit, CAIES, determines which edge of the output signal sets the CAIFG flag. The interrupt-enable bit, CAIE, along with the general interrupt-enable bit (GIE) control if the CAIFG bit generates a CPU interrupt. If both the CAIE and the GIE bits are set, then the CAIFG flag will generate a CPU interrupt request. The CAIFG flag is automatically reset when the CPU interrupt request is serviced. The CAIFG, CAIES, and CAIE bits are all located in the CACTL1 register.

Figure 14–3. Comparator_A Interrupt System



14.3 Comparator_A Control Registers

The Comparator_A module is configured with three module registers as shown in Table 14–1. The module registers are mapped into the lower peripheral file address range where all byte modules are located and should be accessed with byte instructions (suffix B).

Table 14–1. Comparator_A Control Registers

Register	Short Form	Register Type	Address	Initial State
• C_A control register 1:	CACTL1	Read/write	059h	Reset
• C_A control register 2:	CACTL2	Read/write	05Ah	Reset
• C_A port dissipation reg:	CAPD	Read/write	05Bh	Reset

14.3.1 Comparator_A, Control Register CACTL1

The control register CACTL1 is shown and described below.

	7							0
CACTL1 059h	CAEX	CA RSEL	CA REF1	CA REF0	CAON	CAIES	CAIE	CAIFG
	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

CAIFG, bit0: The Comparator_A interrupt flag

CAIE, bit1: The Comparator_A interrupt enable

CAIES, bit2: The Comparator_A interrupt edge select bit

0: The rising edge of the comparator output sets the Comparator_A interrupt flag CAIFG

1: The falling edge of the comparator output sets the Comparator_A interrupt flag CAIFG

CAON, bit3: The comparator is switched on or off. When off, the current consumption of the comparator is stopped. The current consumption of the reference circuitry is enabled or disabled independently.

0: The comparator is disabled, current consumption is stopped and the output of the comparator is low.

1: The comparator is enabled and active.

CAREF, bit4,5: 0: Internal reference is switched off. An external reference can be applied.

1: 0.25*Vcc reference is selected

2: 0.50*Vcc reference is selected

3: Diode reference is selected.
The diode reference varies with each individual device, temperature and supply voltage. See device data sheet.

CARSEL, bit6: The internal reference V_{CAREF}, selected by CAREF bits, is applied to the +terminal or –terminal.

0: Reference is selected to the +terminal (CAEX = 0) or –terminal (CAEX = 1)

1: Reference is selected to the –terminal (CAEX = 0) or +terminal (CAEX = 1)

CAEX, bit7: The inputs of the comparator are exchanged. This is used to measure or compensate for the offset of the comparator.

14.3.2 Comparator_A, Control Register CACTL2

The control register CACTL2 is shown and described below.

	7							0
CACTL2 05Ah	CACTL 2.7	CACTL 2.6	CACTL 2.5	CACTL 2.4	P2CA1	P2CA0	CAF	CAOUT
	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)

CAOUT, bit0: The comparator output. Writing to this bit, for example when writing a new register value, has no affect or negative impact.

CAF, bit1: The comparator output filter is bypassed (CAF=0) or switched into the output path (CAF=1).

P2CA0, bit2: Pin to CA0

0: The external, pin signal is not connected to the Comparator_A

1: The external, pin signal is connected to the Comparator_A

P2CA1, bit3: Pin to CA1

0: The external, pin signal is not connected to the Comparator_A

1: The external, pin signal is connected to the Comparator_A

Bits 4–7: See device data sheet for implementation.

14.3.3 Comparator_A, Port Disable Register CAPD

Typically, the comparator input and output functions are multiplexed with digital I/O port pins to save pin count on a device. Also, slope A/D applications often utilize multiple digital I/O ports (for charging and discharging) to provide multiple channels of slope A/D conversion. In these multichannel applications, a useful feature of the digital I/O pins is the ability to disable the input buffer. Typically, all channels except the one being converted are disabled, providing a high-impedance input and avoiding current consumption caused by through-put current. (See section 14.4.1)

The typical digital I/O ports on MSP430 do not have the ability to disable the input buffer. However, on devices with the Comparator_A, the capability has been added and is controlled with the CAPD.x bits.

The control bits CAPD.0 to CAPD.7 are initially reset, enabling all the input buffers for the associated port. The port input buffer is disabled if the according CAPD.x bit is set. See device data sheet for port associations.

The ability to disable the input buffer for the device pin applies to up to eight inputs of the associated digital I/O port (check device data sheet for implementation details). For example, the x11x1 devices have CA1 multiplexed on pin P2.4 and CA0 multiplexed on pin P2.3, so the Comparator_A is associated with port P2. On this device, all input buffers associated with all P2 pins (P2.x) may have the capability to be disabled with the CAPD register.

	7							0
CAPD 05Bh	**CAPD.7**	**CAPD.6**	**CAPD.5**	**CAPD.4**	**CAPD.3**	**CAPD.2**	**CAPD.1**	**CAPD.0**
	rw-(0)	**rw-(0)**	**rw-(0)**	**rw-(0)**	**rw-(0)**	**rw-(0)**	**rw-(0)**	**rw-(0)**

CAPD.x: 0: The input buffer for the pin enabled.

1: The input buffer for the pin is disabled.

14.4 Comparator_A in Applications

The Comparator_A can be used to:

- Measure resistive elements
- Detect external voltage or current levels
- Measure external voltage and current sources
- Measure the voltage of a battery used in the system

14.4.1 Analog Signals at Digital Inputs

Typically, Comparator_A inputs are multiplexed with digital I/O pins. When analog signals are applied to these digital CMOS gates, parasitic current can flow from the positive terminal (V_{DD}, V_{CC}) to the negative terminal (V_{SS}, GND). See Figure 14–4. This parasitic current occurs if the input voltage is around the transition level of the input gate.

Figure 14–4. Transfer Characteristic and Power Dissipation in a CMOS Inverter/Buffer

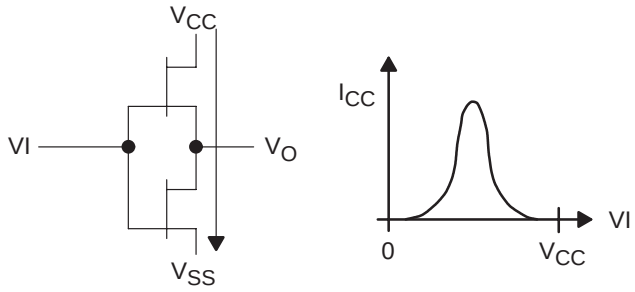
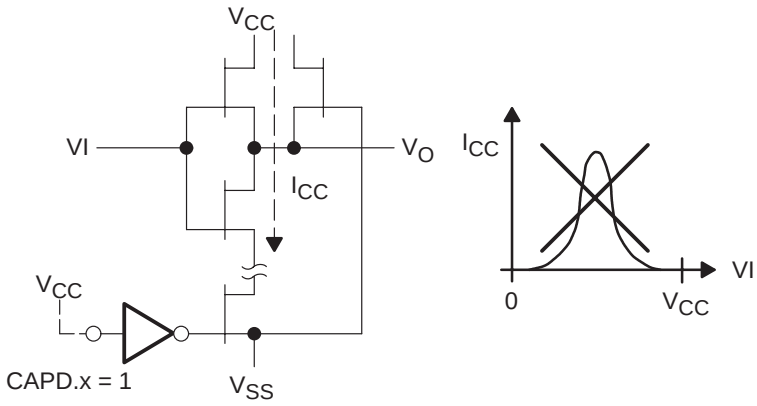


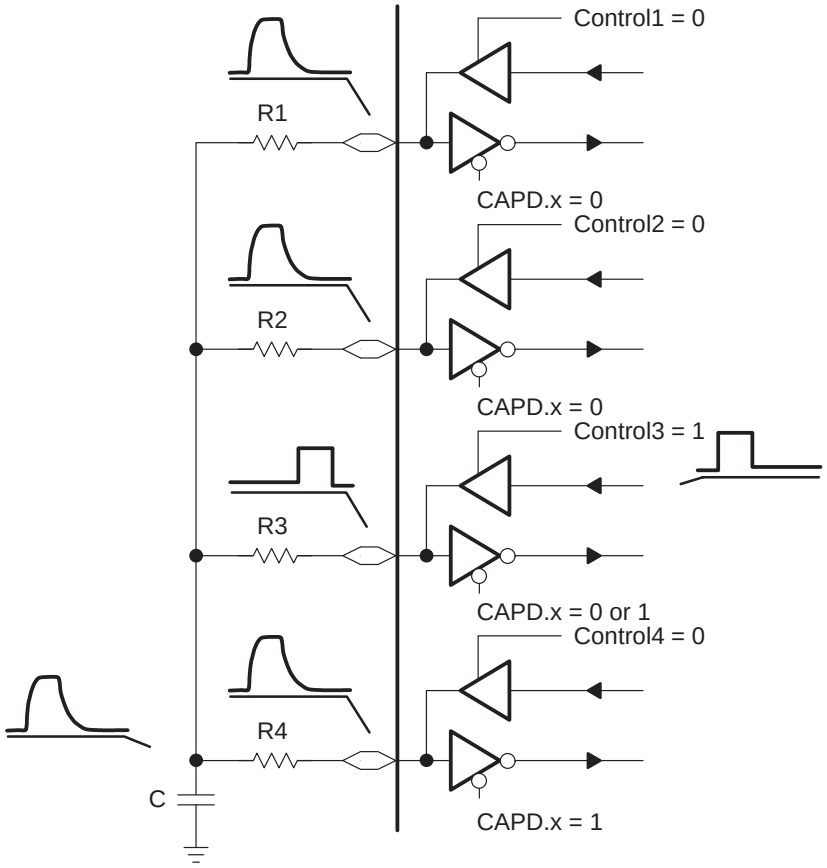
Figure 14–5. Transfer Characteristic and Power Dissipation in a CMOS Gate



MSP430 devices with the Comparator_A module have additional circuitry on the associated digital I/O port pins to allow the input buffers to be disabled (see Figure 14–5). The buffers are enabled or disabled with the CAPD.x bits (see section 14.3.3). Note that the circuitry is added to all pins of the associated I/O port, not just the pins for the Comparator_A inputs.

Disabling the input buffer for a specific pin will disable the parasitic current flow and therefore reduce overall current consumption. It is important to disable the buffer for any I/O pin that is not being actively driven if current consumption is critical (see Figure 14–6).

Figure 14–6. Application Example With One Active(Driving R3) and Three Passive Pins With Applied Analog Signals

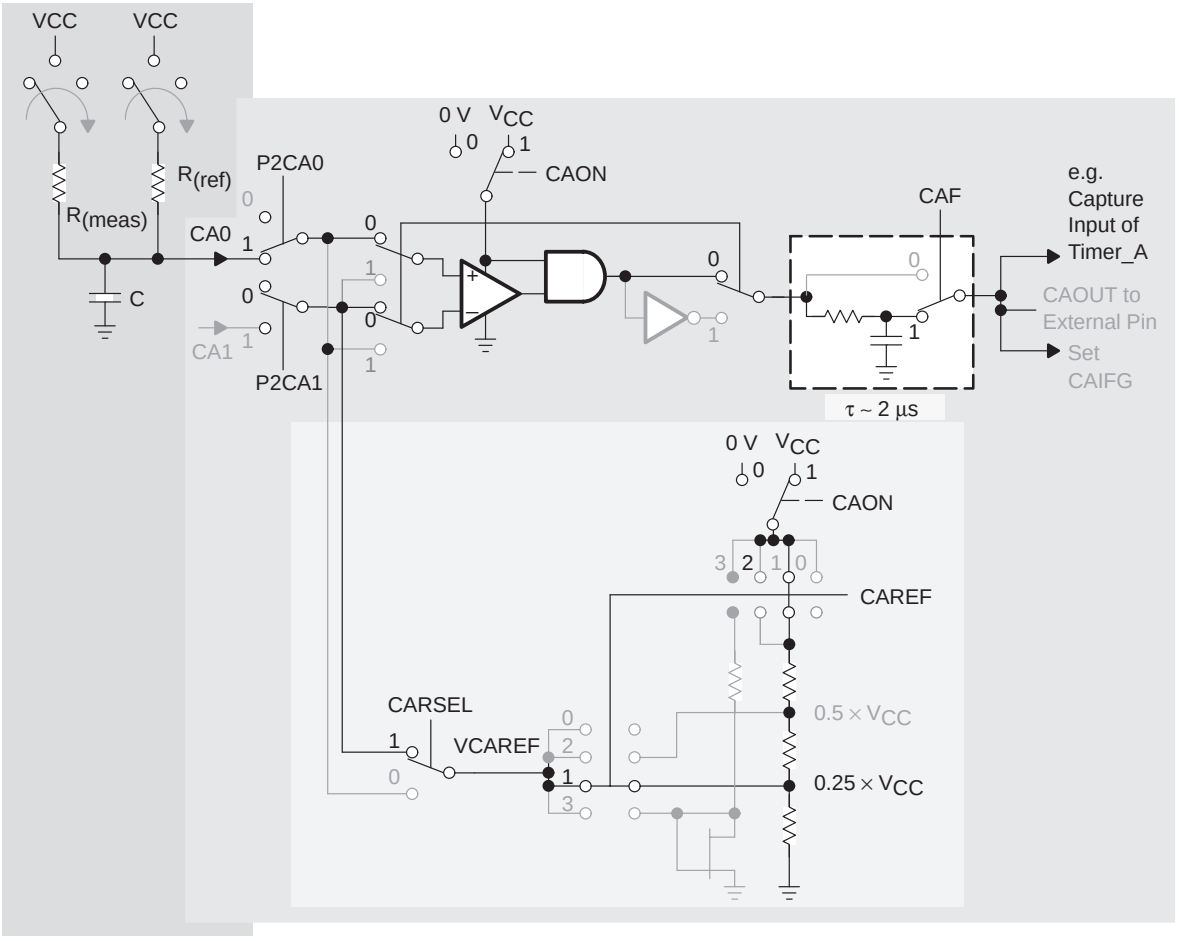


The specific implementation (which digital inputs/outputs can be controlled by CAPD.x) varies with each MSP430 device configuration. Refer to the specific device's data sheet to see which I/O port is associated with Comparator_A.

14.4.2 Comparator_A Used to Measure Resistive Elements

The Comparator_A can be used to measure resistive elements. For example, temperature can be converted into digital data via a thermistor, by comparing the thermistor's discharge time to that of a reference resistor. See Figure 14–7.

Figure 14–7. Temperature Measurement Systems



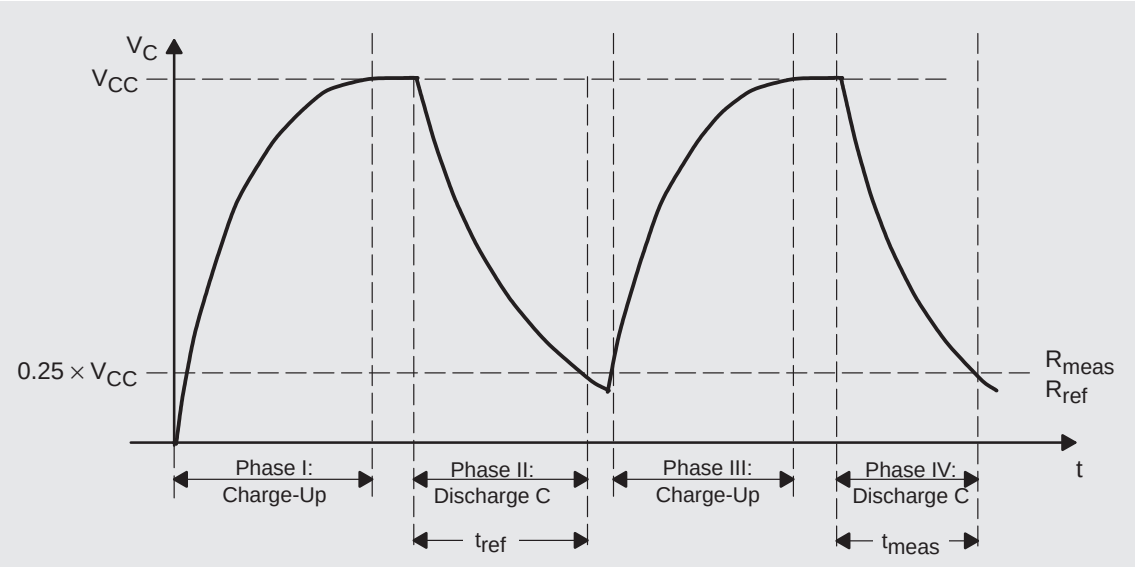
The resistive elements are compared using a capacitor charge-discharge cycle, as shown in Figure 14–8. This is based on a ratiometric conversion principle, as the ratio of two capacitor-discharge times is compared. Absolute V_{CC} and the actual capacitor value are not critical, as the ratiometric principle cancels these values out. V_{CC} and the capacitor value should simply remain constant during the conversion.

$$\frac{N_{meas}}{N_{ref}} = \frac{-R_{meas} \times C \times ln \frac{V_{ref}}{V_{CC}}}{-R_{ref} \times C \times ln \frac{V_{ref}}{V_{CC}}}$$

$$\frac{N_{meas}}{N_{ref}} = \frac{R_{meas}}{R_{ref}}$$

$$R_{meas} = R_{ref} \times \frac{N_{meas}}{N_{ref}}$$

Figure 14–8. Timing for Temperature Measurement Systems



MSP430 resources used to calculate the temperature sensed by R(meas):

Digital I/O:

- Two digital outputs to charge and discharge the capacitor. Port pins are set to provide a V_{CC} output (charge a capacitor), reset to discharge a capacitor, and switched to high-impedance (including correct state of CAPD.x bit) when not in use. One output discharges the capacitor via reference resistor R(ref), the other output discharges it via R(meas).

Comparator_A:

- The – terminal is connected to a reference level, for example 0.25 x V_{CC}.
- The + terminal is connected to the positive terminal of the capacitor.
- CAOUT or CAIFG utilized to measure the discharge time.
- The output filter should be used to minimize multiple switching when the voltages at the comparator inputs are close together.

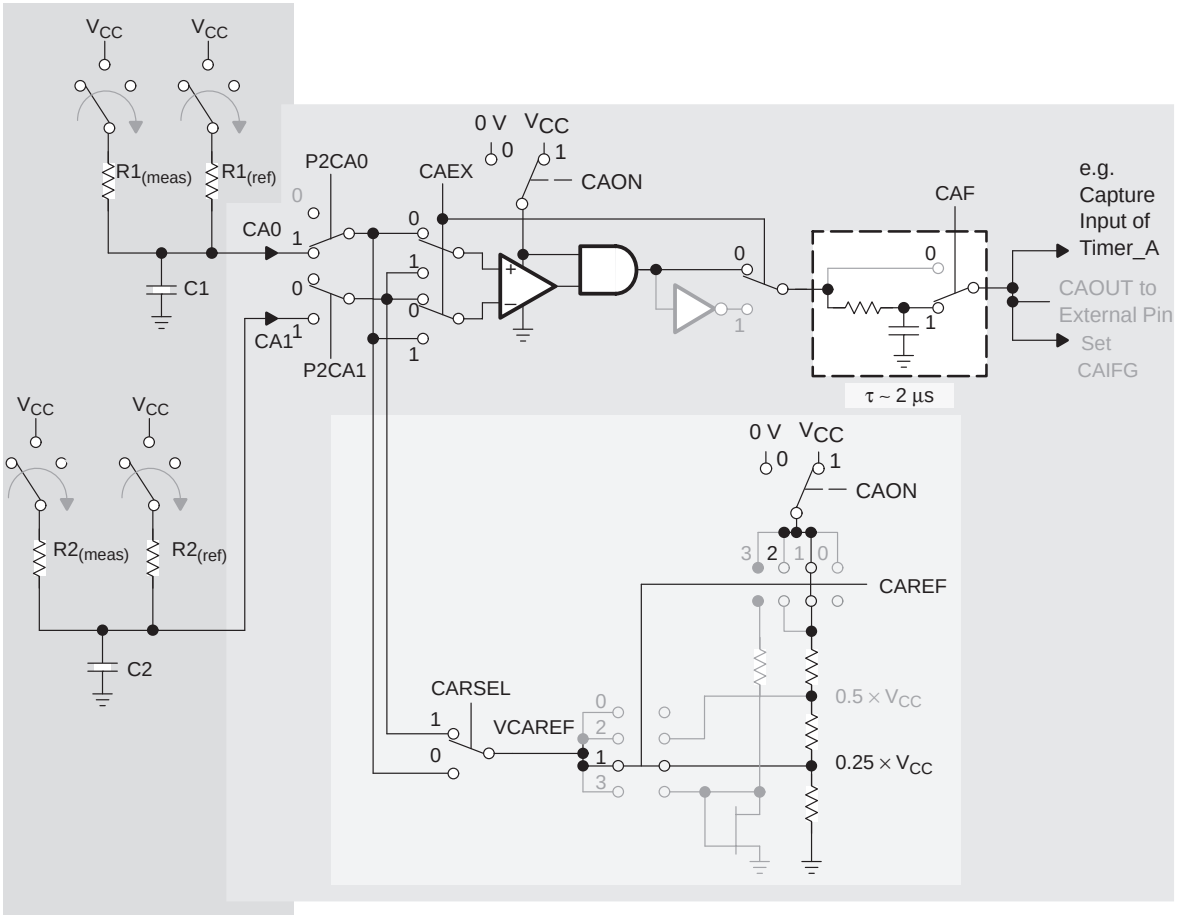
If CAOUT is available as an input to a timer-capture register such as Timer_A, the capacitor discharge time can be measured very precisely, without software polling for a change of CAOUT, by using the timer capture function.

$$R_{meas} = R_{ref} \times \frac{N_{meas}}{N_{ref}} \qquad T_{meas} = f\left(R_{meas}\right)$$

14.4.3 Measuring Two Independent Resistive Element Systems

It is possible to measure two independent systems with one comparator. The input multiplexer, which is controlled via CAEX, allows the two independent systems to be isolated. See Figure 14–9. An example could be if one temperature sensor has a resistor range of 10 kΩ to 200 kΩ. The other sensor is in the range of 1 kΩ to 1.5 kΩ. Two independent measurement paths are used to optimize individual measurement performance. The conversion principle is identical to the one described in the previous section.

Figure 14–9. Two Independent Temperature Measurement Systems



In Figure 14–10, the active signal paths are shown when the upper independent system is selected for conversion. This example uses the 0.25×V_{CC} internal reference, and shows the software selectable RC-filter as active.

Figure 14–10. Temperature Measurement Via Temperature Sensor $R1_{(meas)}$

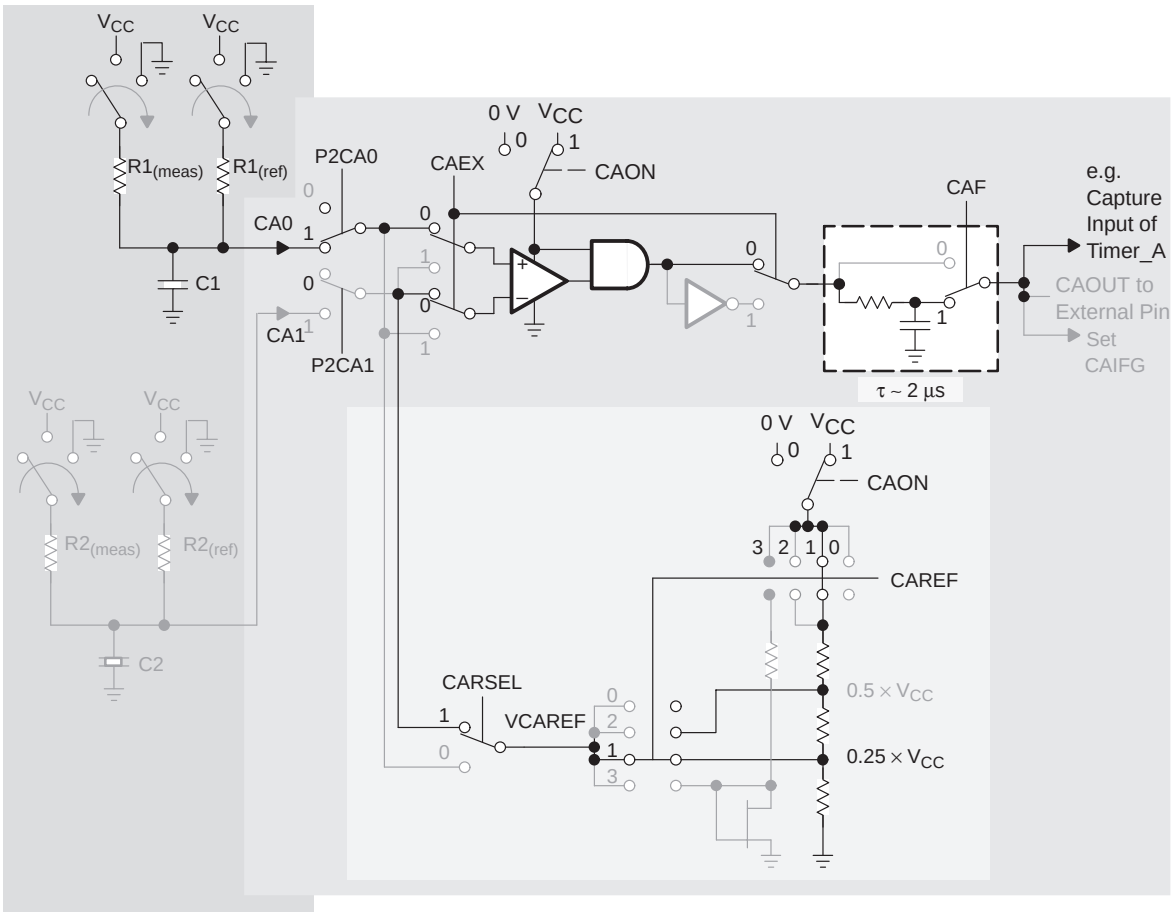
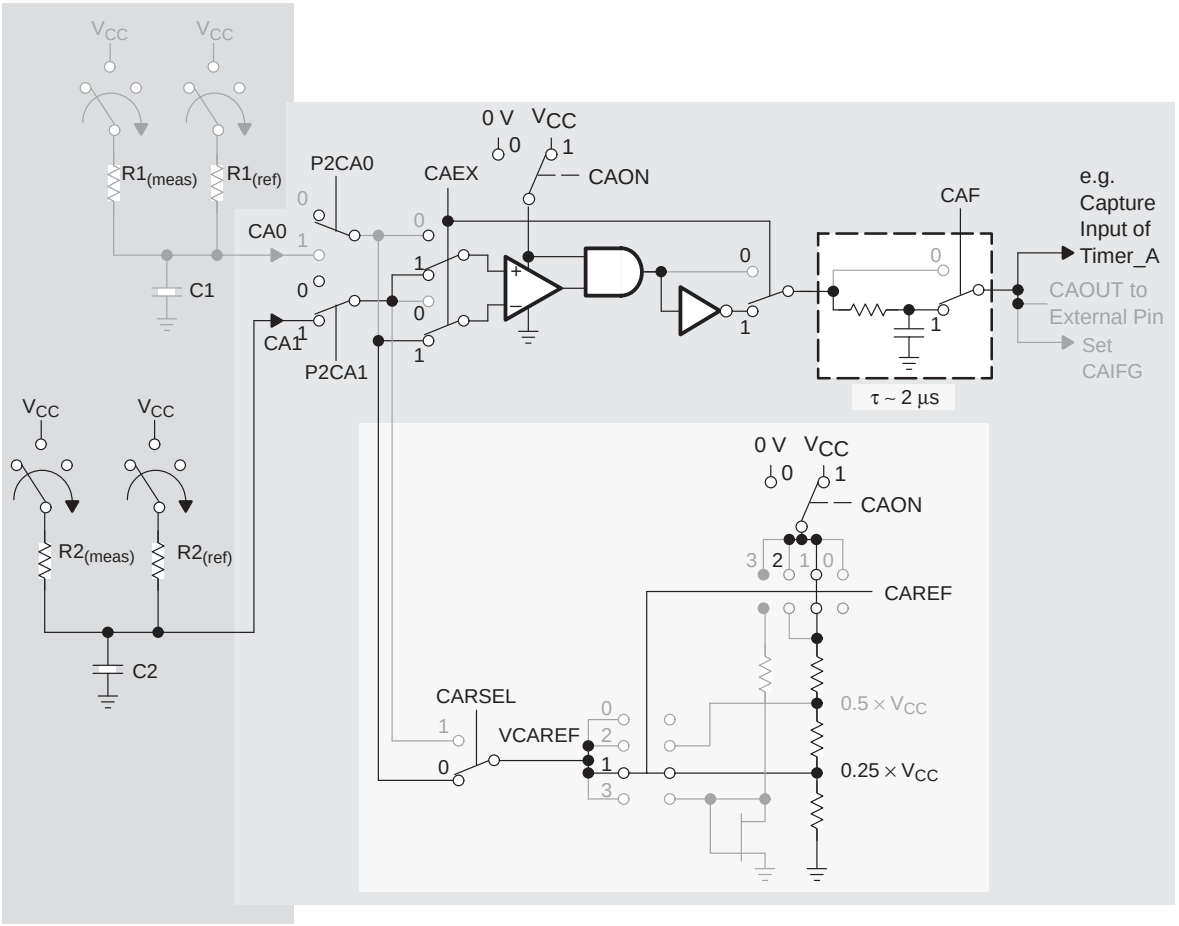


Figure 14–11 shows the active signal paths for the lower independent system. This example uses the 0.25×V_{CC} internal reference and shows the software selectable RC-filter as active.

Figure 14–11. Temperature Measurement Via Temperature Sensor $R2_{(meas)}$



14.4.4 Comparator_A Used to Detect a Current or Voltage Level

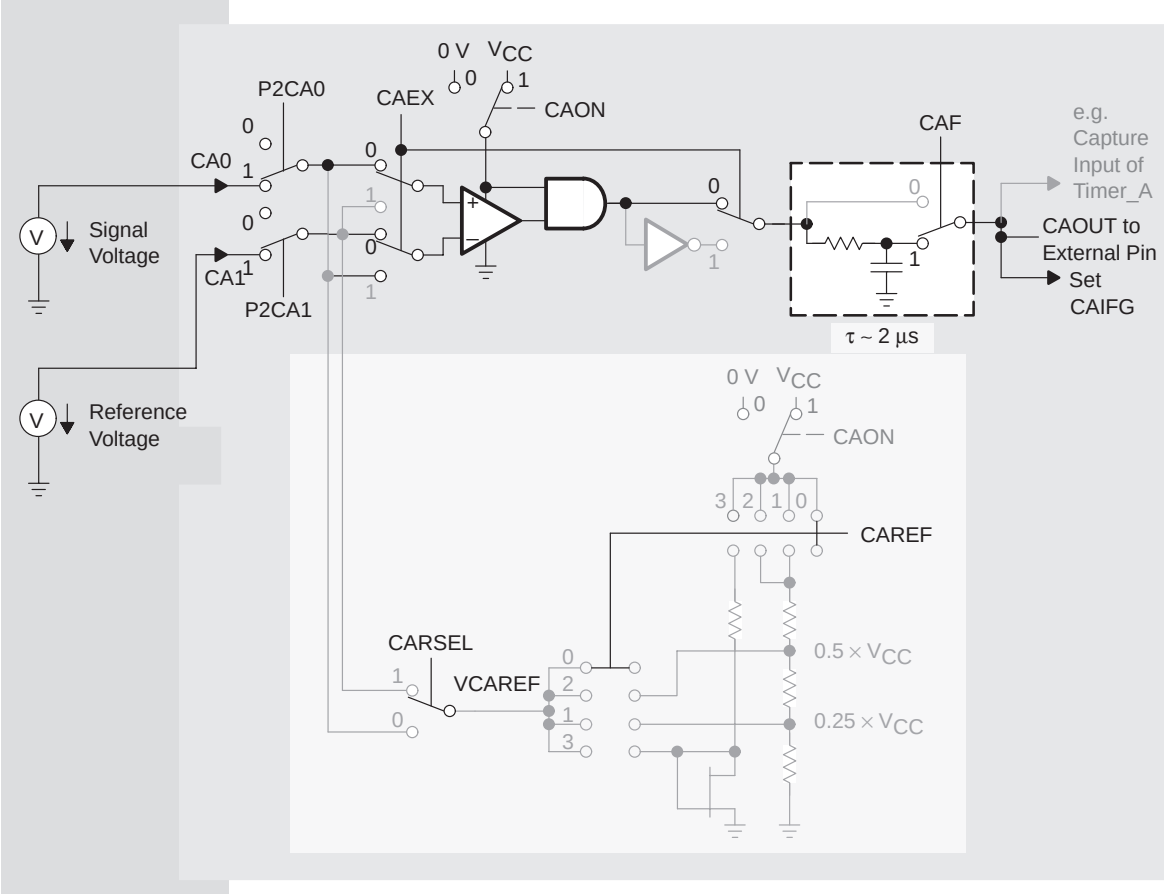
Comparator_A can be used to detect current or voltage levels if they are below or above a reference level (shown in Figure 14–12). The reference level can be selected from the internal reference-voltage generator, or by applying an external reference level. Application software can poll the CAOUT bit for the status of the comparator, or use the interrupt flag CAIFG to determine if the level of the current or voltage source has crossed the comparator threshold.

In Figure 14–12, two external voltages are compared. Application software can poll the CAOUT bit:

CAOUT = 0: V(signal < V(ref)

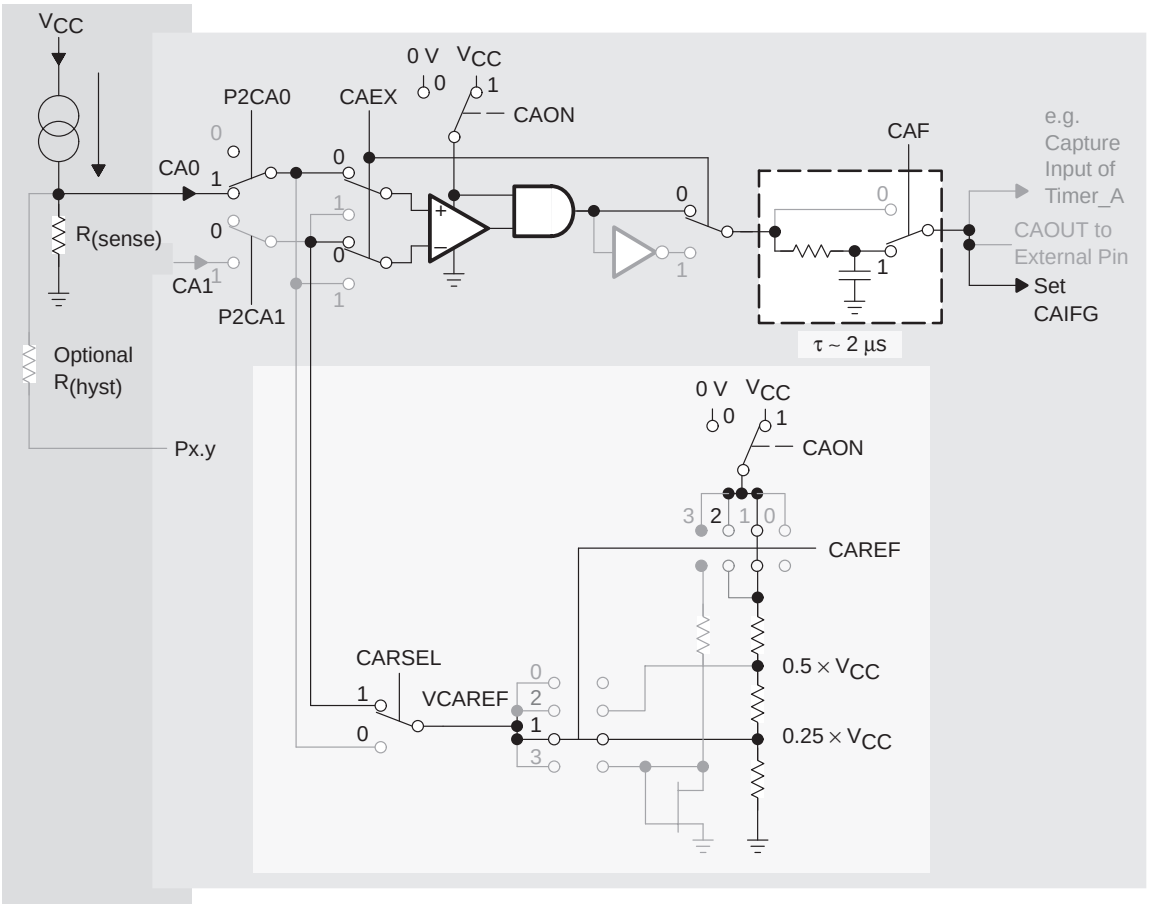
CAOUT = 1: V(signal > V(ref)

Figure 14–12. Detect a Voltage Level Using an External Reference Level



In Figure 14–13 current is transferred to an input voltage by $I \times R_{(sense)}$. The current limit is set for example to $0.25 \times V_{CC}$. The current is below the limit as long as CAOUT is reset.

Figure 14–13. Detect a Current Level Using an Internal Reference Level

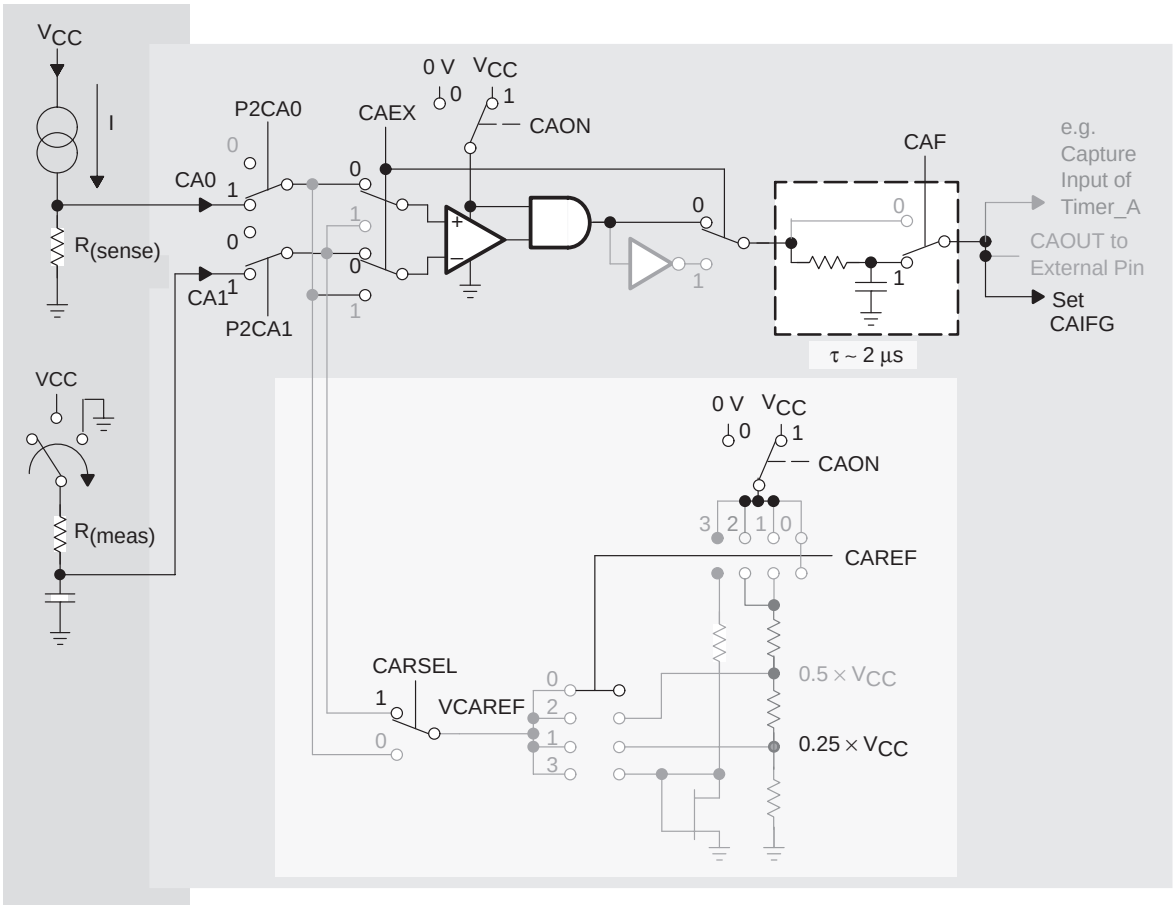


14.4.5 Comparator_A Used to Measure a Current or Voltage Level

In addition to detecting levels, the comparator can be used to measure currents or voltages. To measure a voltage, a known, stable voltage source is used to charge up an RC combination. The time required to charge the combination to a threshold value set by the voltage to be measured is then used to calculate the voltage level (see Figure 14–16). V_{CAREF} can be used as the known stable voltage source if V_{CC} in the user's system meets the required stability and accuracy.

A similar approach is used to measure a current. A known, stable voltage source is again used to charge an RC combination to a threshold value. In this case, the threshold voltage is created by passing the current to be measured through a known resistance (see Figure 14–14).

Figure 14–14. Measuring a Current Source



The equation for the current *I* is:

$$I = \frac{1}{R(sense)} \times V_{CC} \times \left[1 - e^{\frac{t_1 \times ln\,0.3}{t_1 + t_2}}\right]$$

Figure 14–15. Timing for Measuring a Current Source

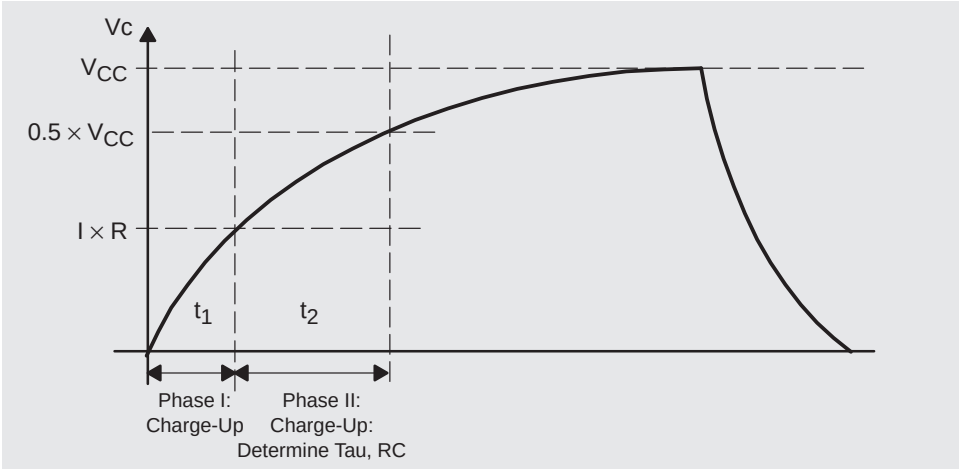
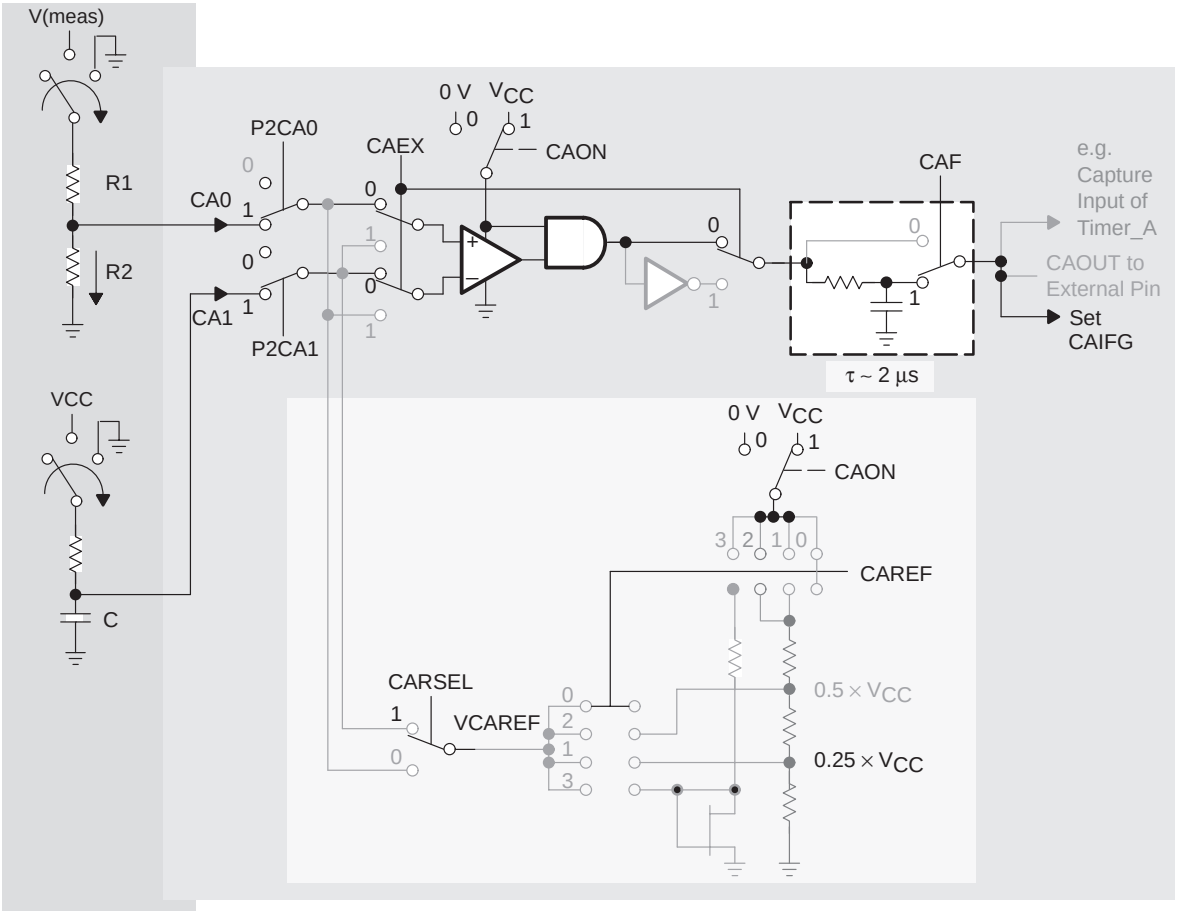


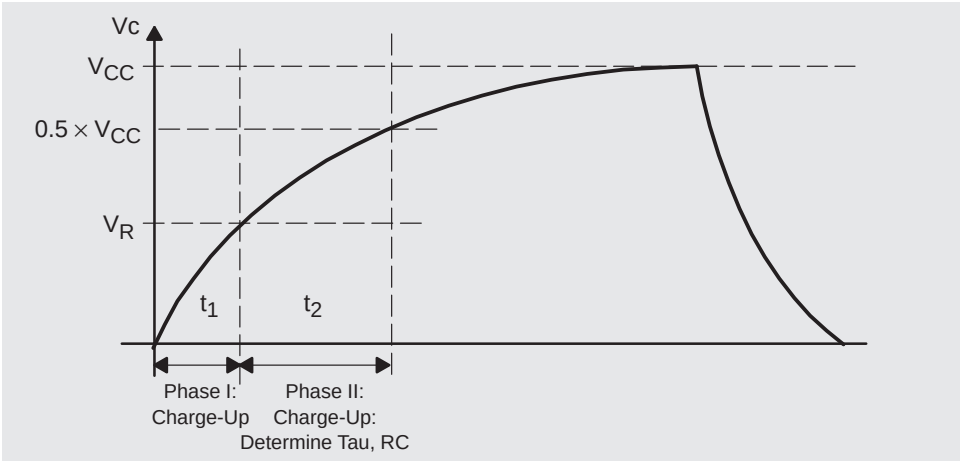
Figure 14–16. A/D Converter for Voltage Sources



The equation for the voltage V_{meas} is:

$$V(meas) = \frac{R1 + R2}{R2} \times V_{CC} \times \left[1 - e^{\frac{t_1 \times ln\,0.3}{t_1 + t_2}}\right]$$

Figure 14–17. A/D Converter for Voltage Sources, Conversion Timing



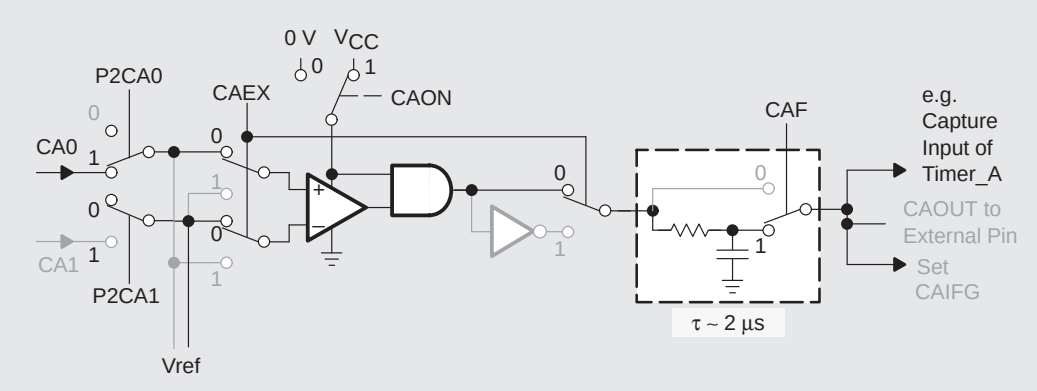
Note: During phase I, control bit P2CA0 = 1 and CAREF = 0.
During phase II, control bit P2CA0 = 0, CARSEL = 0, and CAREF= 1.

14.4.6 Measuring the Offset Voltage of Comparator_A

The input offset voltage of the comparator varies with each device and also with temperature, supply voltage, and input voltage. If the input voltage is stable (*reference voltage*), it will not influence the offset voltage significantly. To increase the precision of voltage measurements, the comparator offset voltage can be measured by the following steps. To simply compensate for the offset without measuring it, see section 14.4.7

First, execute a conversion with CAEX = 0. V_{CA0} is applied to the + terminal of the comparator, and V_{ref} is applied to the – terminal of the comparator as shown in Figure 14–18.

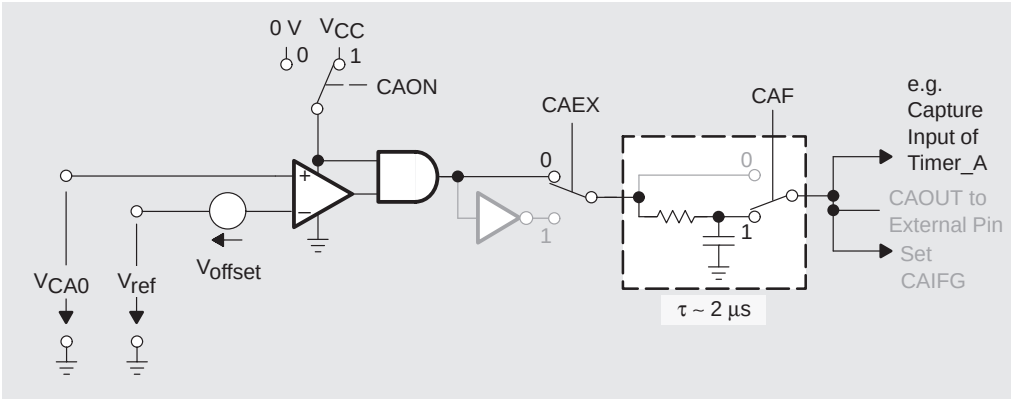
Figure 14–18. Measuring the Offset Voltage of the Comparator, CAEX = 0



The V_{offset} in this configuration is in series with V_{ref} as shown in Figure 14–19.

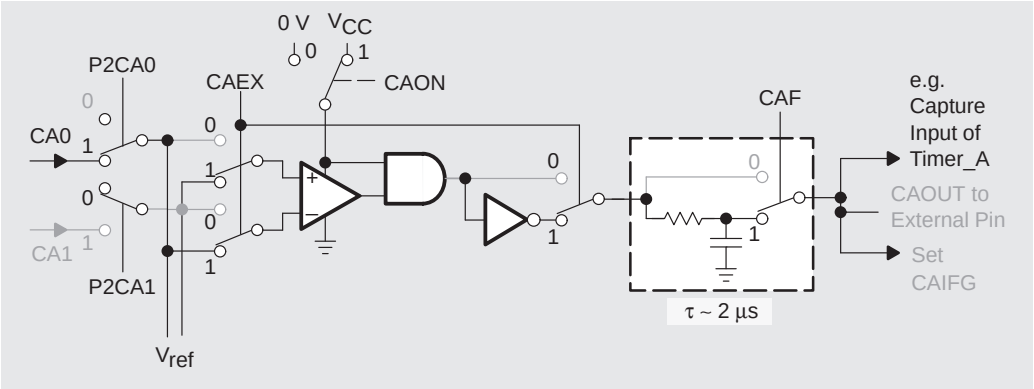
$V_{CA0} = V_{ref} + V_{offset}$

Figure 14–19. Offset Voltage of the Comparator, CAEX = 0



Next, execute a conversion with CAEX = 1. V_{CA0} is applied to the – terminal of the comparator, and V_{ref} is applied to the + terminal of the comparator as shown in Figure 14–20.

Figure 14–20. Measuring the Offset Voltage of the Comparator, CAEX = 1

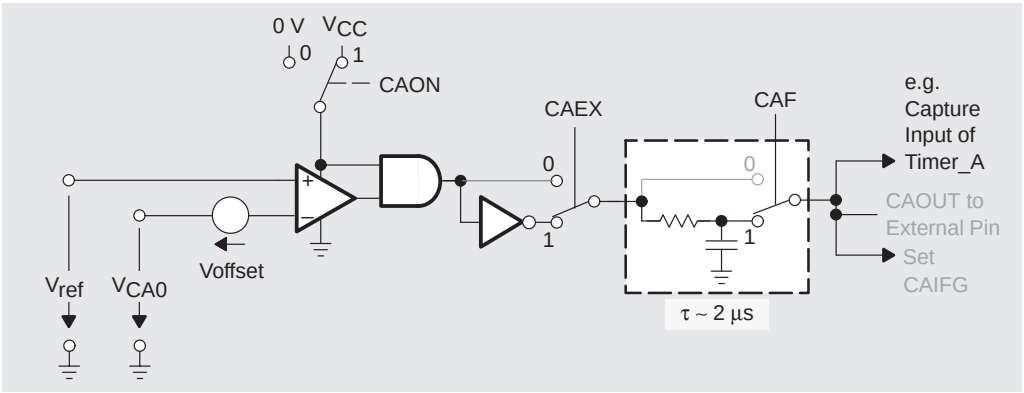


The V_{offset} in this configuration is in series with V_{CA0} as shown in Figure 14–21.

$$V_{ref} = V_{CA0} + V_{offset}$$

$$V_{CA0} = V_{ref} - V_{offset}$$

Figure 14–21. Offset Voltage of the Comparator, CAEX = 1



Finally, calculate V_{ref} from the below formulas.

$$N1 = -R_{VCA0} \times C \times ln \frac{V_{ref} + V_{offset}}{V_{CC}} \times f_{osc}$$

$$N2 = -R_{VCA0} \times C \times ln \frac{V_{ref} - V_{offset}}{V_{CC}} \times f_{osc}$$

This leads to:

$$V_{offset} = V_{CC} \times e^{\left(\frac{N1+N2}{2N1} \times ln \frac{V_{ref}}{V_{CC}}\right)} - V_{ref}$$

N = timer counts

14.4.7 Compensating for the Offset Voltage of Comparator_A

Another way to improve the accuracy is to compensate for the effect of input offset voltage without actually measuring it.

When CAEX = 0, the V_{offset} is in series with V_{ref}:
$V_{CA0} = V_{ref} + V_{offset}$

When CAEX = 1, the V_{offset} is in series with V_{CA0}:
$V_{ref} = V_{CA0} + V_{offset} \quad \Rightarrow \quad V_{CA0} = V_{ref} - V_{offset}$

Adding the result of two conversions (one with each input configuration) and dividing by two will cancel the effect of the offset voltage.

$$\begin{array}{r} V_{CA0} = V_{ref} + V_{offset} \\ + \; V_{CA0} = V_{ref} - V_{offset} \\ \hline 2 \times V_{CA0} = 2 \times V_{ref} \end{array} \quad \Rightarrow \quad \frac{N1 + N2}{2} = \textit{Conversion without offset}$$

N = Timer count

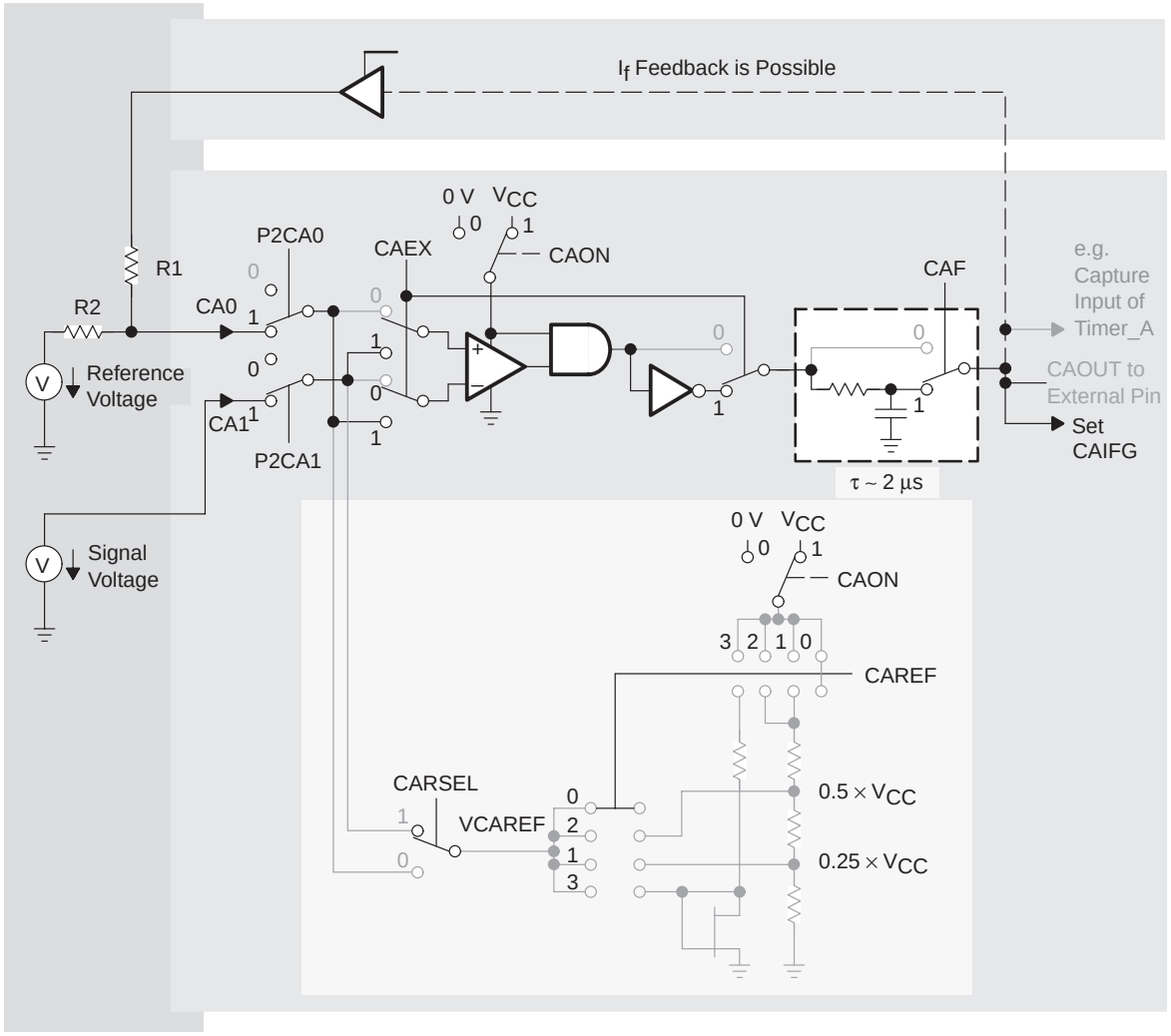
14.4.8 Adding Hysteresis to Comparator_A

When the voltage level applied to the + terminal is close to the voltage level at the – terminal, the output of the comparator may oscillate. This can cause the following two situations:

- The current consumption increases, since the signal path driven by the comparator output is constantly charged and discharged.
- The software receives constant requests for service either via interrupt service requests, or after successful polling of CAOUT or CAIFG.

Figure 14–22 shows how to add hysteresis to the comparator to prevent output oscillation.

Figure 14–22. Use CAOUT at an External Pin to Add Hysteresis to the Reference Level



Adding hysteresis can only be done if CAOUT is available externally. Refer to the device's data sheet to determine if CAOUT is available at an external pin.

The hysteresis can be calculated as follows:

$$V_{(hyst)} = \pm \frac{R2}{R1 + R2} \times V_{CC}$$

Chapter 15

ADC12

The ADC12 12-bit analog-to-digital converter is a high-speed, extremely versatile analog-to-digital converter implemented on MSP430x13x and MSP430x14x devices. This chapter discusses the ADC12 and how to use it.

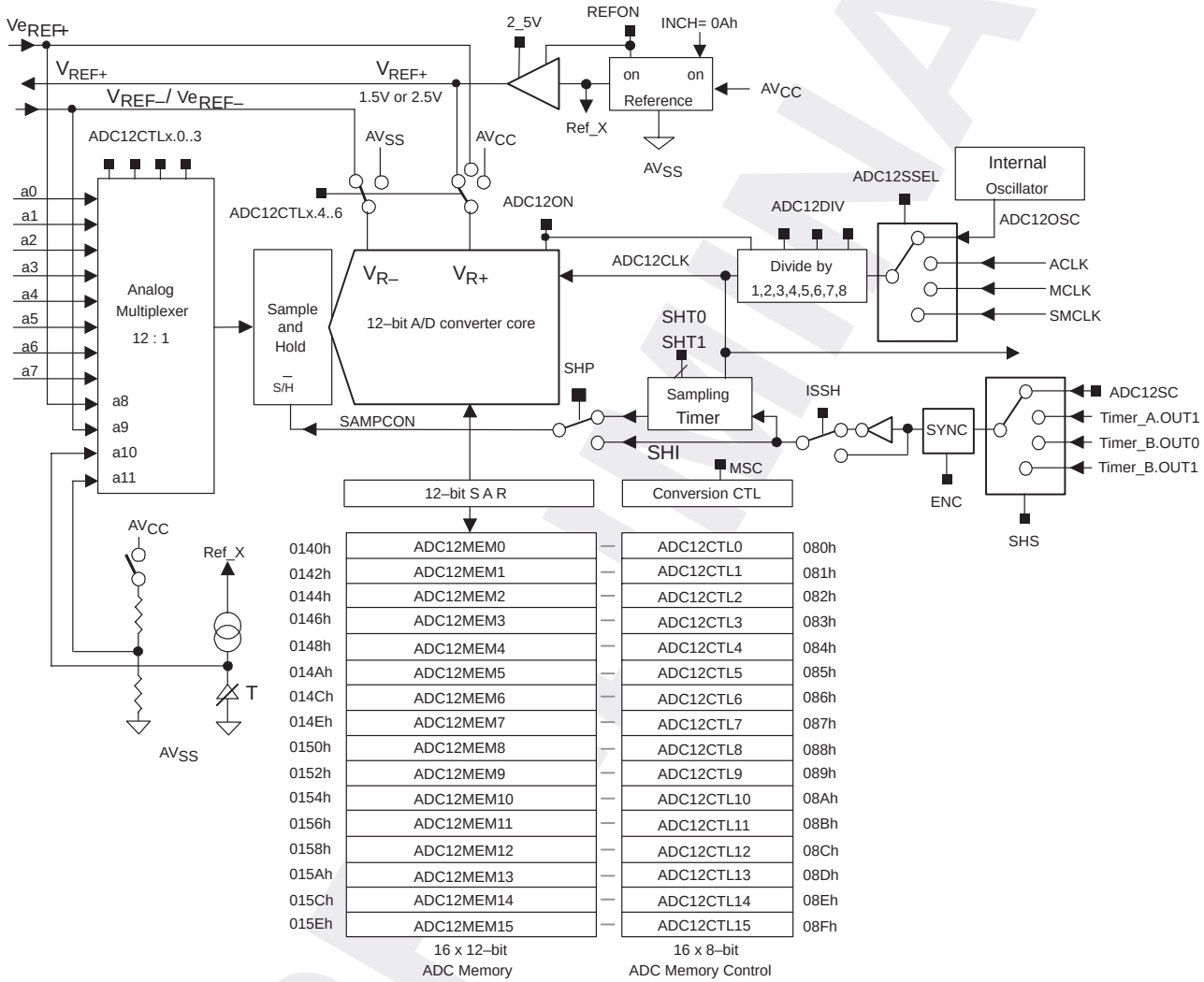
Topic **Page**

15.1 Introduction

The ADC12 12-bit analog-to-digital converter (shown in Figure 15–1) has five main functional blocks that can be individually configured and optimized:

- ADC core with sample-and-hold
- Conversion memory and configuration
- Reference voltage and configuration
- Conversion clock source select and control
- Sample timing and conversion control

Figure 15–1. ADC12 Schematic



The ADC12 can convert one of eight external analog inputs, or one of four internal voltages. The four internal channels are used for temperature measurement (via on-chip temperature diode), and for measurement of Vcc (via Vcc/2) and the positive and negative references applied on Ve_{REF+} and $V_{REF–}/Ve_{REF–}$.

The ADC12 can use its internal reference, or it can use external reference(s) or a combination of internal and external reference-voltage levels.

The ADC12 has versatile sample-and-hold circuitry giving the user many options for control of the sample timing. The sample timing may be directly controlled by software (via a control bit), or any one of three internal or external signals (depending on device configuration – see *Sampling* section and check the data sheet for details). Typically, the internal timing signals come from other MSP430 timers such as Timer_A. Additionally, the sample timing may be programmed as a multiple of the ADC12 conversion clock.

As with sample timing, the user has several choices for the ADC12 conversion clock. The ADC12 conversion clock may be chosen from any available internal MSP430 clock, or may be selected from a dedicated oscillator contained in the ADC12 peripheral. Additionally, the chosen clock source may be divided by any factor from 1 to 8.

The ADC12 has four operating modes. It can be configured to perform a single conversion on a single channel, or multiple conversions on a single channel. The ADC12 can also be configured to perform conversions on a sequence-of-channels, running through the sequence once, or repeatedly. When performing conversions on a sequence-of-channels, the sequence is completely definable by the user. For example, a possible sequence-of-channels could be a1–a3–a1–a6–a2, etc. In addition, each channel may be individually configured for which reference(s) are to be used for the conversion.

Conversion results are stored in 16 conversion-memory registers. Each of these registers has its own configuration and control register allowing the user to select the input channel and the reference(s) used for the conversion result that is stored in that register.

Some key and unique features of the ADC12 are:

- 200-ksps maximum conversion rate
- 12-bit converter with ±1LSB differential nonlinearity (DNL) and ±1LSB integral nonlinearity (INL)
- Built-in sample-and-hold with selectable sampling periods controlled by software (via a control bit), a sampling timer, or by other MSP430 timers
- On-chip, dedicated RC oscillator – used as an option for sample-and-conversion timing
- Integrated diode for temperature measurement
- Eight individually configurable channels for conversion of external signals
- Four internal channels for conversion of temperature, AV_{CC}, and external references
- On-chip reference voltages – 1.5 V or 2.5 V, selected by software
- Selectable internal or external sources for both positive and negative reference-voltage levels (selectable for each channel independently)
- Selectable conversion clock source

- Versatile conversion modes including single-channel, repeated single-channel, sequence, and repeated sequence.
- Sixteen 12-bit registers for storage of conversion results. Each register is individually accessible by software and individually configurable to define the channel and references for its conversion result.
- ADC core and reference voltage powered down separately

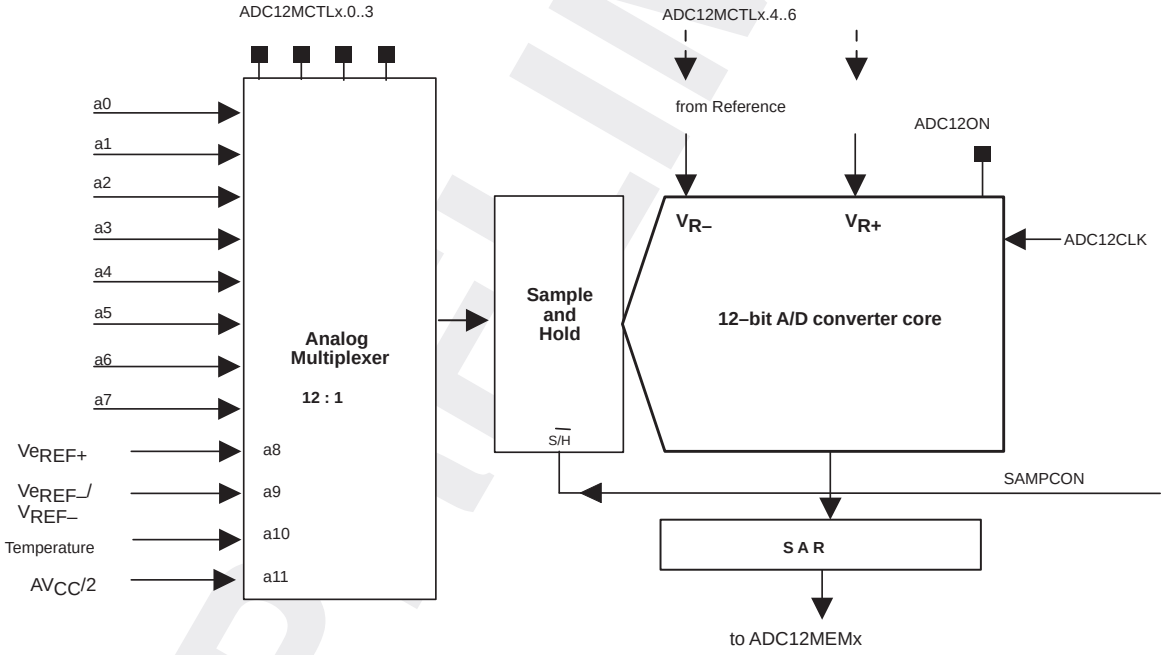
15.2 ADC12 Description and Operation

15.2.1 ADC Core

The ADC core (shown in Figure 15–2) converts the analog input to its 12-bit representation and stores the results in the conversion memory. The core uses two programmable/selectable voltage levels (V_{R+} and $V_{R–}$) to define the upper and lower limits of the conversion range, and to define the full-scale and zero-scale readings. The digital output is full scale when the input signal is equal to or higher than V_{R+}, and zero when the input signal is equal to or lower than $V_{R–}$. The input channel and the reference voltage levels (V_{R+} and $V_{R–}$) are defined in the conversion-control memory. The conversion formula is:

$$\text{NADC} = 4095 \times \frac{\text{Vin} - V_{R–}}{V_{R+} - V_{R–}}$$

Figure 15–2. ADC Core, Input Multiplexer, and Sample-and-Hold



It is important to note that the 3 LSBs of the conversion are resolved resistively. Therefore, when the 3 LSBs are being resolved during a conversion, approximately 200 μA will be required from the reference. The user should keep this in mind when choosing and decoupling an external reference. Refer to the device data sheet for more details on ADC12 specifications.

Caution! ADC12 Turnon Time

When the ADC12 is turned on with the ADC12ON bit, the turnon time noted in the data sheet ($t_{ADC12ON}$) must be observed before a conversion is started. Otherwise, the results will be false.

15.2.2 Reference

The ADC12 A/D converter contains a built-in reference with two selectable reference-voltage levels (1.5 V and 2.5 V). Either of these reference voltages may be applied to V_{R+} of the A/D core and also may be available externally on pin V_{REF+} (check device data sheet for availability of V_{REF+} pin). Additionally, an external reference may be supplied for V_{R+} through pin Ve_{REF+} (check data sheet for availability of Ve_{REF+} pin).

The reference-voltage level for V_{R-} can be selected to be AV_{SS} or may be supplied externally through the V_{REF-}/Ve_{REF-} pin (check device data sheet for V_{REF-}/Ve_{REF-} pin). If the V_{REF-}/Ve_{REF-} pin is not available, then V_R- is connected to AV_{SS}.

Configuration of the reference voltage(s) is done with the Sref bits (bits 4, 5, and 6) in the ADC12MCTLx registers. Up to six combinations of positive and negative reference voltages are supported as described in Table 15–1.

If only external references are used, the internal-reference generator can be turned off with the REFON bit to conserve power.

Table 15–1. Reference Voltage Configurations

Sref	Voltage at V_{R+}	Voltage at V_{R-}
0	AV_{CC}	AV_{SS}
1	V_{REF+} (internal)	AVSS
2,3	Ve_{REF+} (external)	AV_{SS}
4	AV_{CC}	V_{REF-}/ Ve_{REF-} (internal or external)
5	V_{REF+} (internal)	V_{REF-}/ Ve_{REF-} (internal or external)
6,7	Ve_{REF+} (external)	V_{REF-}/ Ve_{REF-} (internal or external)

The voltage levels V_{R+} and V_{R-} establish the upper and lower limits of the analog inputs to produce a full-scale and zero-scale reading, respectively. The values of V_{R+}, V_{R-}, and the analog input should not exceed the positive supply or be lower than AV_{SS}, consistent with the absolute maximum ratings specified in the device data sheet. The digital output is full scale when the input signal is equal to or higher than V_{R+}, and zero when the input signal is equal to or lower than V_{R-}.

Warning ! Reference Voltage Settling Time

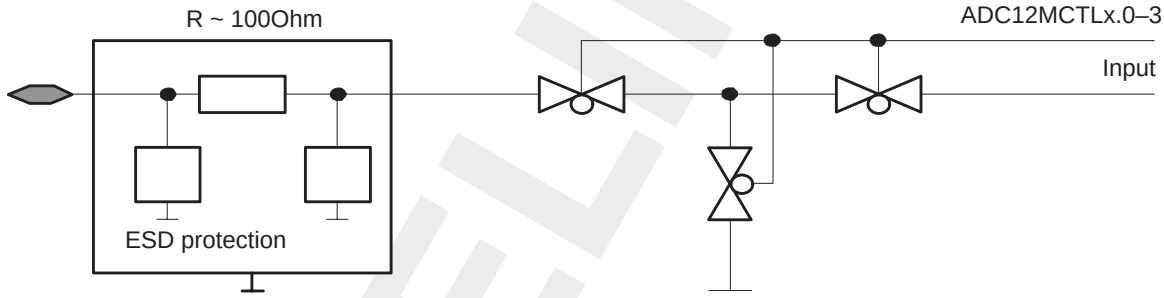
When the built-in reference is turned on with the VREFON bit, the settling timing noted in the data sheet must be observed before starting a conversion. Otherwise, the results will be false until the reference settles. Once all internal and external references have settled, no additional settling time is required when selecting or changing the conversion range for each channel.

15.3 Analog Inputs and Multiplexer

15.3.1 Analog Multiplexer

The eight external analog input channels and four internal signals are selected as the channel for conversion by the analog multiplexer. Channel selection is made for each conversion-memory register with the corresponding ADC12MCTLx register. The input multiplexer is a break-before-make type (shown in Figure 15–3) to reduce input-to-input noise injection resulting from channel switching. The input multiplexer is also a T-switch to minimize the coupling between channels. Channels that are not selected are isolated from the A/D and the intermediate node is connected to analog ground (AV_{SS}) so that the stray capacitance is grounded to help eliminate crosstalk.

Figure 15–3. Analog Multiplexer Channel



Crosstalk can exist because there is always some parasitic coupling capacitance across the switch and between switches. This can take several forms, such as coupling from the input to the output of an off switch, or coupling from an off analog input channel to the output of an adjacent on channel. For high-accuracy conversions, crosstalk interference should be minimized by shielding and other well-known printed-circuit board (PCB) layout techniques.

15.3.2 Input Signal Considerations

During sampling, the analog input signal is applied to the internal capacitor array of the A/D core. Therefore, the charge of the capacitor array is supplied directly by the source. The capacitor array has to be charged completely during the sampling period. Therefore the external source resistances, dynamic impedances, and capacitance of the capacitor array must be matched with the sampling period so the analog signal can settle to within 12-bit accuracy.

Additionally, source impedances also affect the accuracy of the converter. The source signal can drop at the input of the device due to leakage current or averaged dc-input currents (due to input switching currents). For a 12-bit converter, the error in LSBs due to leakage current is:

Error(LSBs) = 4.096 x (μA of leakage current) $\times$ (kΩ of source resistance)/($V_{R+} - V_{R-}$)

For example, a 50-nA leakage current with a 10-kΩ source resistance and a 1.5-V V_{REF} gives 1.4 LSBs of error.

These errors due to source impedance also apply to the output impedance of any external-voltage reference source applied to Ve_{REF+}. The output impedance must be low enough to enable the transients to settle within 0.2/ADCLK and generate leakage current induced errors of <<1LSB.

See the *Sampling* section for more details on sample timing and sampling considerations.

15.3.3 Using the Temperature Diode

To use the on-chip temperature diode, the user simply selects the analog input channel to 10. Any other configuration is done as if an external channel was selected, including reference selection, conversion-memory selection, etc.

Selecting the diode channel automatically turns on the on-chip reference generator (see Figure 15–1) as a voltage source for the temperature diode. However, it does not enable the V_{REF+} output or affect the reference selections for the conversion; so, reference selections are the same as with any other channel.

See the device data sheet for the temperature diode specifications.

15.4 Conversion Memory

A typical approach in single-channel converters uses an interrupt request to signal the end of the conversion and requires the conversion data to be moved to another location before another conversion can be performed. However, the ADC12 incorporates 16 conversion-memory registers (ADC12MEMx, see Figure 15–1) allowing the A/D converter to run multiple conversions without software intervention. This increases the system performance by reducing software overhead.

Additionally, each of the 16 conversion-memory registers has an associated control register (ADC12MCTLx) allowing total flexibility for each conversion. The memory-control registers allow the user to specify the channel and reference(s) used for each individual conversion. All other control bits that configure the other operating conditions of the ADC12, such as conversion modes, sample and conversion control signal, ADC clock, and sample timing, are located in control registers ADC12CTL0 and ADC12CTL1. Each conversion-memory register is individually accessible by software in the address range 0140h – 015Eh.

Using the conversion memory involves control bits in two places. First, the CStartAdd bits located in ADC12CTL1 point to the conversion-memory register to be used for single-channel conversions or the first conversion-memory register to be used for a sequence. The conversion-start address (CStartAdd) can be any value from 0h – 0Fh and points to ADC12MEM0 – ADC12MEM15, respectively. Second, the end-of-sequence (EOS) bit in each conversion-memory control register marks the end of an automatic-conversion sequence.

The EOS bit, when set, defines the end of a conversion sequence. When cleared, an internal conversion-memory pointer (not visible to software) is incremented after the current conversion is completed and the conversion result is stored in the conversion memory. The conversion-memory pointer is then prepared to use the next conversion-memory register to store the results of the next conversion. The internal conversion-memory pointer is incremented with each conversion until a set EOS bit is encountered. Note that defining the end of a sequence is independent from defining the mode of operation (see the *Conversion Modes* section), and that the EOS bits are ignored when using single-conversion mode or repeated conversion of a single-channel mode.

Conversion sequences always use sequential conversion-memory registers, can start with any conversion-memory register, and do not necessarily require any EOS bit to be set. For example, if the CONSEQ bits define the mode of operation to be conversion of a sequence (single or repeated), the CStartAdd bits point to conversion-memory register 14, and no EOS bits are set for any of the conversion-memory registers, then the conversion-memory registers will be used in sequential order (14, 15, 0, 1, 2, ... , 14, 15, 0, 1, 2, … etc.) for each consecutive conversion, and the sequence of conversions will continue until stopped by software. This is useful, for example, in an application that must take advantage of the buffering supplied by the conversion memory but requires more than 16 repeated conversions of a single channel. In this instance the user should set up each memory-control register identically, specifying the same channel and reference(s) for each conversion, and all EOS bits must be cleared. Once the converter is started, it will continue to run until stopped by software.

15.5 Conversion Modes

The ADC12 has four conversion modes:

- ❑ Single-channel, single-conversion
- ❑ Single-channel, repeated-conversions
- ❑ Sequence-of-channels, single-sequence
- ❑ Sequence-of-channels, repeated-sequence

Each mode is summarized in Table 15–2 and described in detail in the following sections.

Table 15–2. Conversion-Modes Summary

CONVERSION MODE	CONSEQ	OPERATION
Single channel	00	Single conversion from a selected channel X = CstartAdd, points to the conversion start address Result is in ADC12MEMx; interrupt flag is ADC12IFG.x Channel (INCH) and reference voltage (Sref) are selected in ADC12MCTLx
Sequence-of-channels	01	A sequence-of-channels is converted x = CstartAdd; points to the conversion start address The last channel in a sequence (y) is marked with EOS=1 (ADC12MCTLx.7), all other EOS bits in ADC12MCTLx, ADC12MCTL(x+1)..., ADC12MCTL(y–1) are reset. Result is in ADC12MEMx, ADC12MEM(x+1),…, ADC12MEMy Interrupt flags are ADC12IFG.x, ADC12IFG.(x+1),…, ADC12IFG.y More than one sequence is possible Channel (INCH) and reference voltage (Sref) are selected in ADC12MCTLx
Repeat single channel	10	The conversion of one single channel is permanently repeated until repeat is off or ENC is reset x = CstartAdd; points to the conversion start address Result is in ADC12MEMx; interrupt flag is ADC12IFG.x Channel (INCH) and reference voltage (Sref) are selected in ADC12MCTLx
Repeat sequence-of-channels	11	The conversion of a sequence-of-channels is permanently repeated until repeat is off or ENC is reset x = CstartAdd; points to the conversion start address The last channel in a sequence (y) is marked with EOS=1 (ADC12MCTL.7), all other EOS bits in ADC12MCTLx, ADC12MCTL(x+1),…, ADC12MCTL(y–1) are reset Result is in ADC12MEMx, ADC12MEMx+1,….; interrupt flag is ADC12IFG.x, ADC12IFG.x+1,…. More than one sequence is possible Channel (INCH) and reference voltage (Sref) are selected in ADC12MCTLx

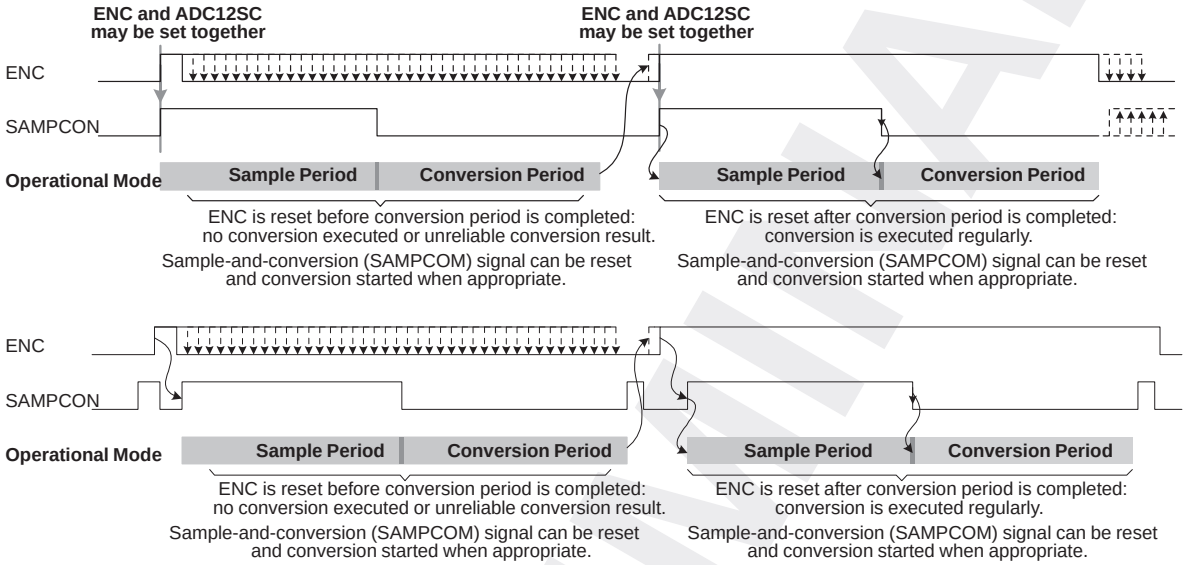
15.5.1 Single-Channel, Single-Conversion Mode

The single-channel mode converts a single channel once. The channel to be converted is selected by the INCH bits in the conversion-memory control register (ADC12MCTLx) associated with the conversion-memory register pointed to by the CStartAdd bits (located in ADC12CTL1x. The conversion range (V_{R+},

V_{R-}) is configured in the same conversion-memory control register by the Sref bits. The conversion result is stored in conversion-memory register ADC12MEMx pointed to by the CStartAdd bits.

The conversion may be stopped immediately by resetting the enable-conversion bit (ENC, located in ADC12CTL0), but the conversion results will be unreliable, or the conversion may not be performed. This is illustrated in Figure 15–4.

Figure 15–4. Stopping Conversion With ENC Bit

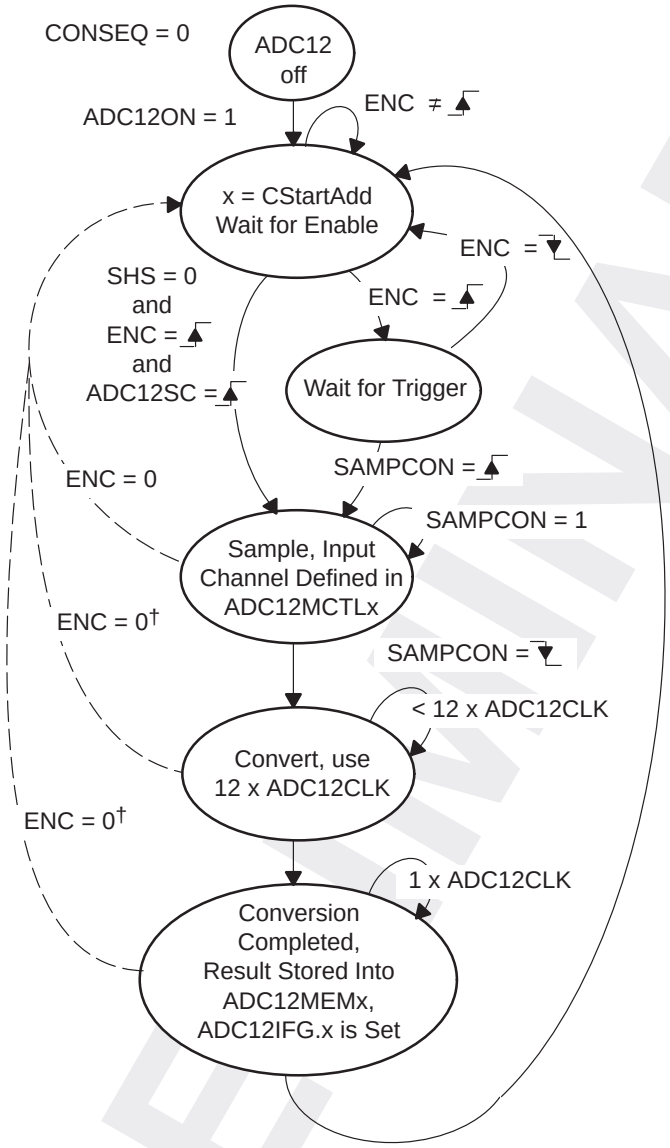


When the conversion is complete and the results are written to the selected conversion-memory register, the corresponding interrupt flag ADC12IFG.x is set, and, if the appropriate interrupt enables are set, an interrupt request is generated (see the *ADC12 Interrupt Vector Register ADC12IV* section).

Once the conversion is completed, the ENC bit must be reset and then set again to prepare for another conversion. All additional incoming sample-input signals will be ignored until the ENC bit is reset and set again. Also, the conversion mode may be changed after the conversion begins but before it has completed, and the new mode will take effect after the current conversion has completed. See also the *Switching between Conversion Modes* section.

An illustration of single-channel, single-conversion mode is shown in Figure 15–5.

Figure 15–5. Single-Channel, Single-Conversion Mode



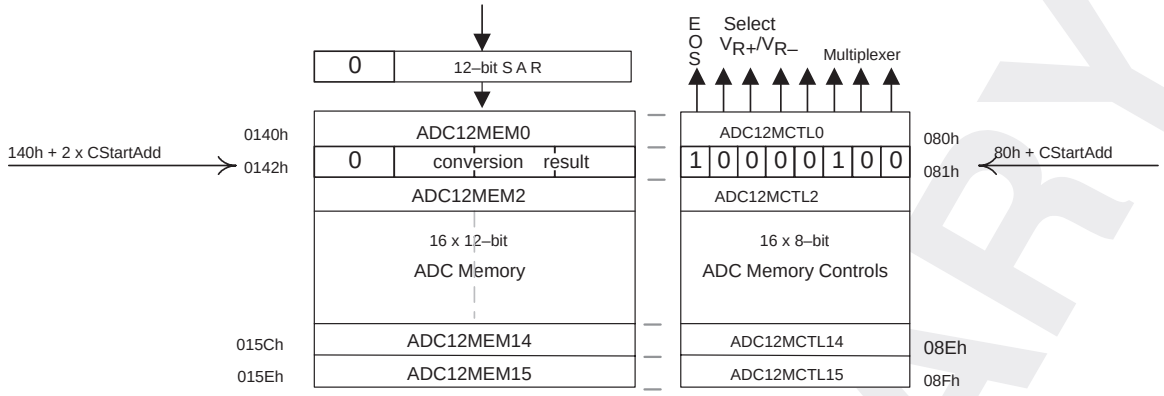
† Conversion result is unpredictable

An example of the conversion-memory setup is shown in Figure 15–6 for single-channel conversion. The example uses the following conditions:

- Single conversion of channel a4
- Internal reference voltage with V_{R+} at AV_{CC} and with V_{R-} at AV_{SS}
- Conversion result to be stored in conversion-memory register ADC12MEM1

This means that control bit CStartAdd in ADC12CTL0 is assigned a value of 1. The channel (INCH=4) and reference voltages (Sref=0) are selected via ADC12MCTL1.

Figure 15–6. Example Conversion-Memory Setup



15.5.2 Sequence-of-Channels Mode

The sequence-of-channels mode converts a sequence of channels. The CStartAdd bits in ADC12CTL1 point to the first conversion-memory register used for the sequence. The results of the remaining conversions in the sequence are stored in sequential conversion-memory registers. For example, if a sequence is three-conversions long and the CStartAdd bits point to conversion-memory register 4, then when the sequence is started, the first conversion result isstored in ADC12MEM4, the second result is stored in ADC12MEM5, and the third result is stored in ADC12MEM6.

When performing sequences of conversions, the channels and references for each conversion are individually configurable via the conversion-memory control register associated with the each conversion-memory register used in the sequence. For example, if a sequence of conversions uses ADC12MEM3 – ADC12MEM6, then the channel and reference(s) for each conversion are individually configurable with ADC12MCTL3 – ADC12MCTL6.

The end of a sequence is marked by the end-of-sequence bit (EOS) in the last conversion-memory control register used in the sequence. Each conversion-memory control register contains an EOS bit. All EOS bits of the conversion-memory control registers used in a sequence must be reset, except for the last one in the sequence. For example, if a sequence starts with ADC12MEM7 and ends with ADC12MEM12, then the EOS bit of registers ADC12MCTL7 – ADC12MCTL11 must be reset and the EOS bit of ADC12MCTL12 must be set. Conversions stop when the end of a sequence is reached.

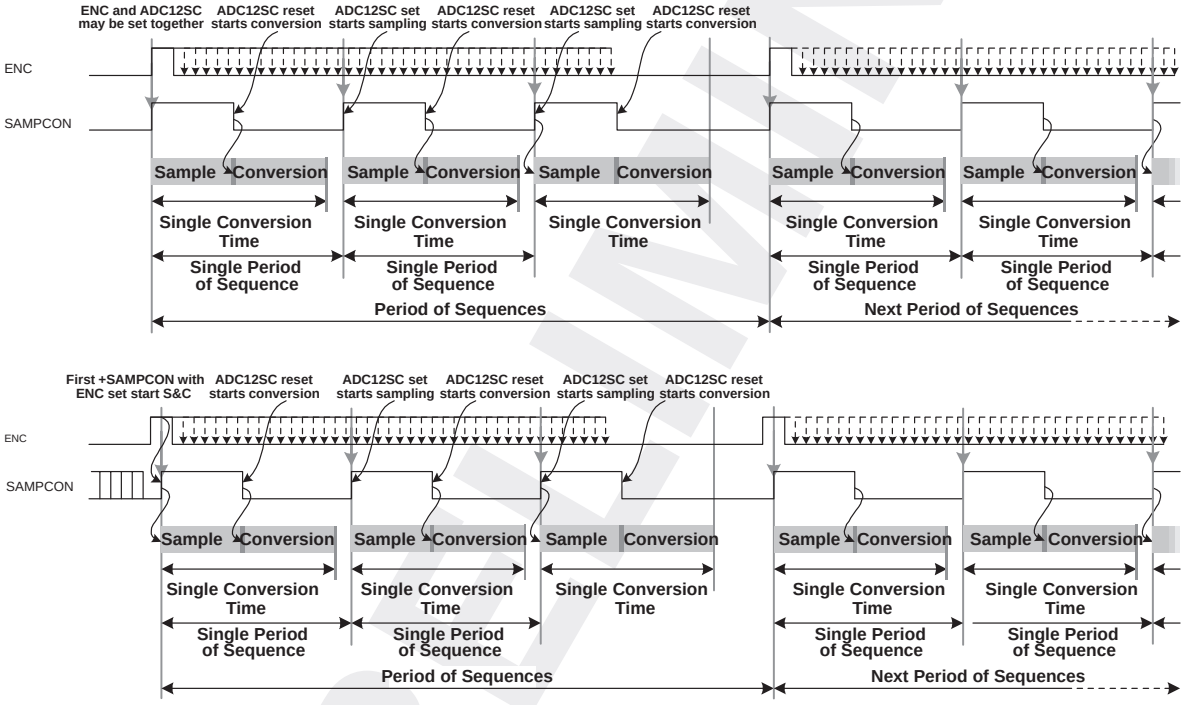
To execute the same sequence, or a new sequence, the ENC bit must be reset and then set again. All additional sample-input signals are ignored until this happens. Note that the ENC bit may be reset any time after the sequence begins, and the sequence still completes normally (see Figure 15–7).

If the conversion mode is changed after the sequence begins but before it has completed and the ENC bit is left high, the sequence completes normally, and the new mode takes effect after the sequence completes, unless the new mode is *single-channel single-conversion.* If the new mode is *single-channel single-conversion,* the current sequence-of-channels stops proceeding when

no sample-and-conversion is active, or after an active sample-and-conversion is completed. The original sequence may not be completed, but all completed conversion results are valid. See also the *Switching Between Conversion Modes* section.

If the conversion mode is changed after the sequence begins, but before it has completed, and the ENC bit is toggled, then the original sequence completes normally and the new mode takes effect and is started after the original sequence completes – unless the new mode is *single-channel single-conversion*. If the new mode is *single-channel single-conversion,* then the original sequence stops when no sample-and-conversion is active, or after an active sample-and-conversion is completed, or when the ENC bit is reset, whichever comes first. Then, the single conversion begins when the ENC bit is set again. See also the *Switching Between Conversion Modes* section.

Figure 15–7. ENC Does Not Effect Active Sequence

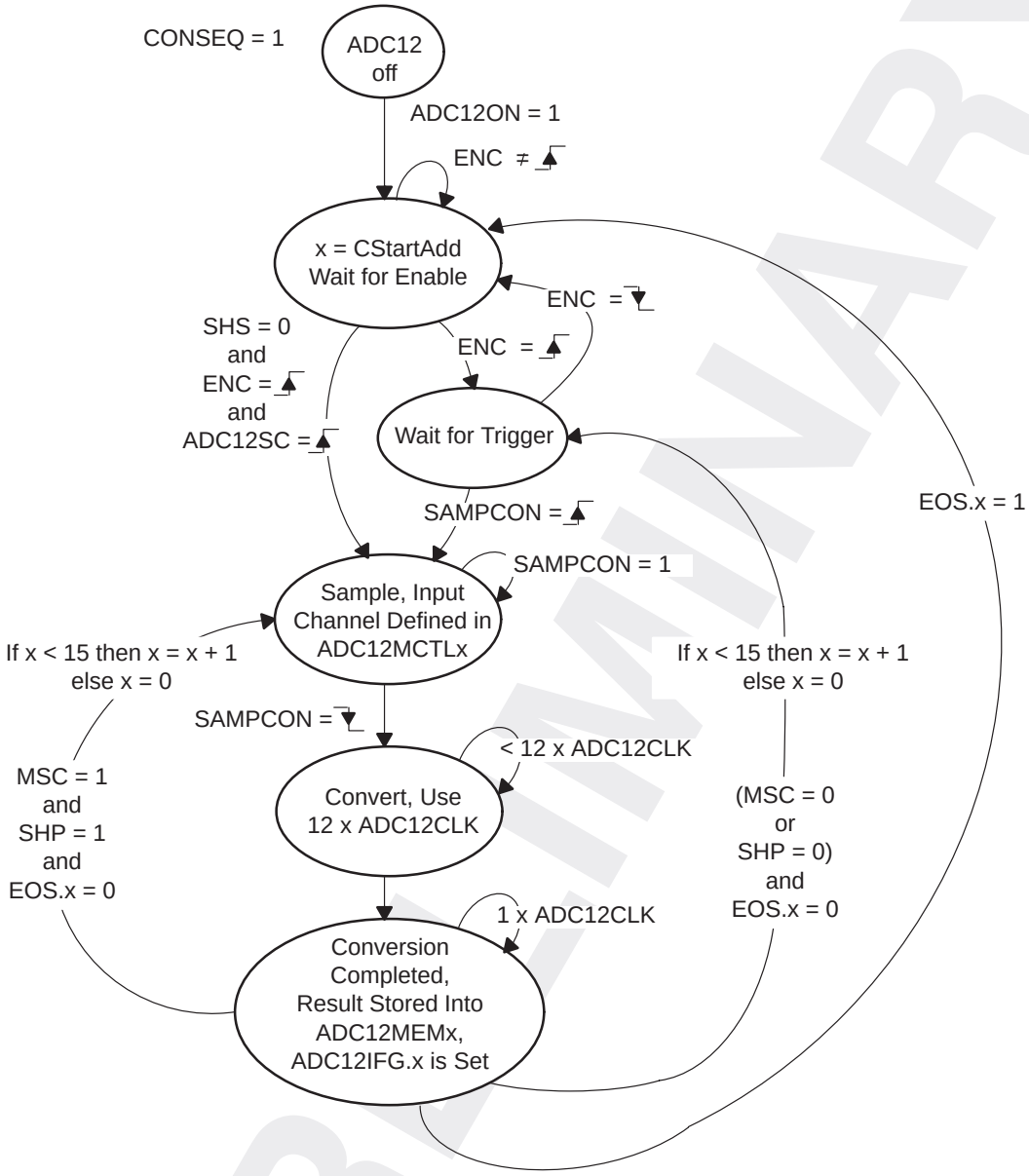


An active sequence may be stopped immediately by selecting *single-channel single-conversion* mode (reset CONSEQ.1 bit) and then resetting the enable-conversion bit ENC. The data in memory register ADC12MEMx is unpredictable and the interrupt flag ADC12IFG.x may or may not be set. This is generally not recommended but may be used as an emergency exit.

Each time a conversion is completed, the results are loaded into the appropriate ADC12MEMx register and the corresponding interrupt flag ADC12IFG.x is set to indicate completion of the conversion. Additionally, If the appropriate interrupt-enable flags are set, an interrupt request is generated (see the *ADC12 Interrupt Vector Register ADC12IV* section).

An illustration of sequence of channels mode is shown in Figure 15–8.

Figure 15–8. Sequence-of-Channels Mode



An example showing a sequence of conversions is shown and flow-charted in Figures 15–9 and 15–10. The example shows the sequence a0, a5, a7, a0, a0, a3, and uses ADC12MEM6 for storing the first conversion results. The set-up of each conversion in the sequence is:

- a0, using reference voltages $V_{R+} = AV_{CC}$ and $V_{R-} = AV_{SS}$
- a5, using reference voltages V_{R+} at V_{REF+} and $V_{R-} = AV_{SS}$
- a7, using reference voltages V_{R+} at V_{REF+} and $V_{R-} = Ve_{REF-}/ V_{REF-}$
- a0, using reference voltages $V_{R+} = AV_{CC}$ and $V_{R-} = AV_{SS}$
- a0, using reference voltages $V_{R+} = AV_{CC}$ and $V_{R-} = AV_{SS}$
- a3, using reference voltages $V_{R+} = AV_{CC}$ and $V_{R-} = Ve_{REF-}/ V_{REF-}$

Figure 15–9. Sequence-of-Channels Mode Flow

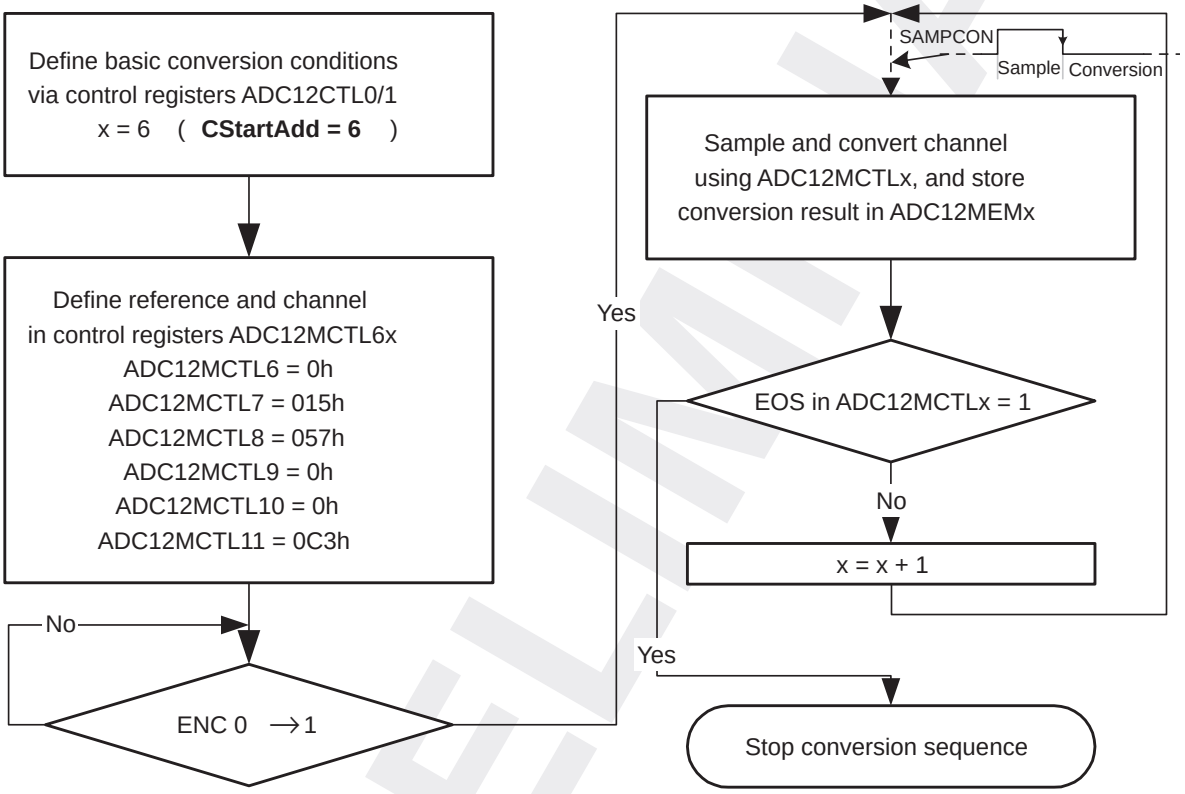
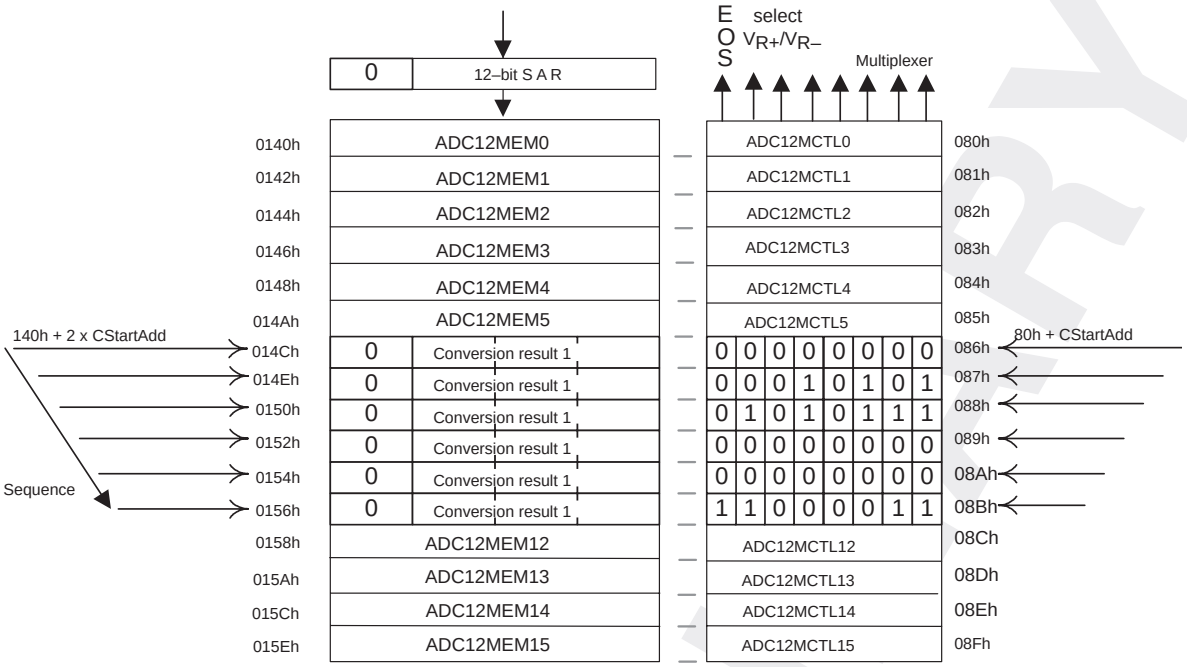


Figure 15–10. Sequence-of-Channels Mode Example



15.5.3 Repeat-Single-Channel Mode

The repeat-single-channel mode is identical to the single-channel mode, except that conversions are repeated on the chosen channel until stopped by software. Each time a conversion is completed, the results are loaded into the appropriate ADC12MEMx register and the corresponding interrupt flag ADC12IFG.x is set to indicate completion of the conversion. Additionally, If the appropriate interrupt-enable flags are set, an interrupt request is generated (see the *ADC12 Interrupt Vector Register ADC12IV* section).

The conversion mode may be changed without first stopping the conversions. When this is done, the new mode takes effect after the current conversion completes (see also the *Switching Between Conversion Modes* section).

There are three ways to stop repeated conversions on a single channel:

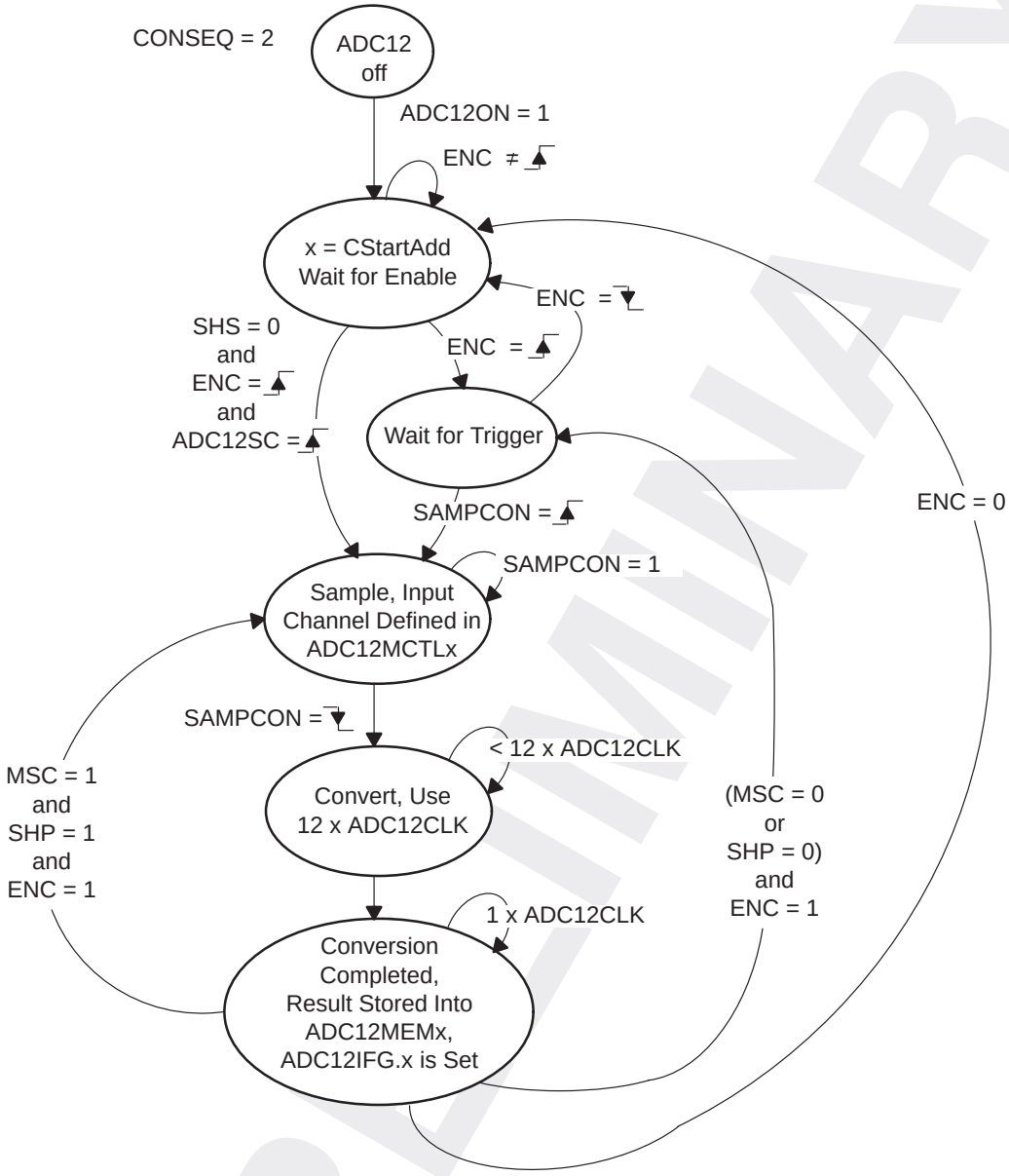
1) Select single-channel mode instead of repeat-single-channel mode with the CONSEQ bits. When this is done, the current conversion is completed normally, the result is loaded into ADC12MEMx, and the associated interrupt flag ADC12IFG.x is set.

2) Reset the ENC bit (ADC12CTL0.1) to stop conversions after the current conversion is completed. Again, the result is loaded into ADC12MEMx and the associated interrupt flag ADC12IFG.x is set.

3) Select single-channel mode instead of repeat-single-channel mode and then reset the enable-conversion bit (ENC). When this is done, the current conversion stops immediately. However, the data in memory register ADC12MEMx is unpredictable and the associated interrupt flag ADC12IFG.x may or may not be set. This method is generally not recommended.

An illustration of repeat-single-channel mode is shown in Figure 15–11.

Figure 15–11. Repeat-Single-Channel Mode



15.5.4 Repeat-Sequence-of-Channels Mode

The repeat-sequence-of-channel mode is identical to the sequence-of-channel mode, except the sequence is repeated continuously until stopped by software. Each time a conversion is completed, the results are loaded into the appropriate ADC12MEMx register and the corresponding interrupt flag ADC12IFG.x is set to indicate completion of the conversion. Additionally, If the appropriate interrupt-enable flags are set, an interrupt request is generated (see the *ADC12 Interrupt Vector Register ADC12IV* section).

The conversion mode may be changed without first stopping the conversions. When this is done, the new mode takes effect after the current sequence com-

pletes, except when the new mode is repeat-single-channel. In this case, the sequence does not complete and the new mode takes effect immediately (see also the *Switching Between Conversion Modes* section).

There are four ways to stop repeat-sequence-of-channels conversions:

1) Select sequence-of-channels mode (CONSEQ = 1) instead of repeated sequence-of-channels mode (CONSEQ = 3). When this is done, the current sequence of conversions is completed normally and no further conversions take place. The conversion results are loaded into registers ADC12MEMx and the corresponding interrupt flags ADC12IFG.x are set.

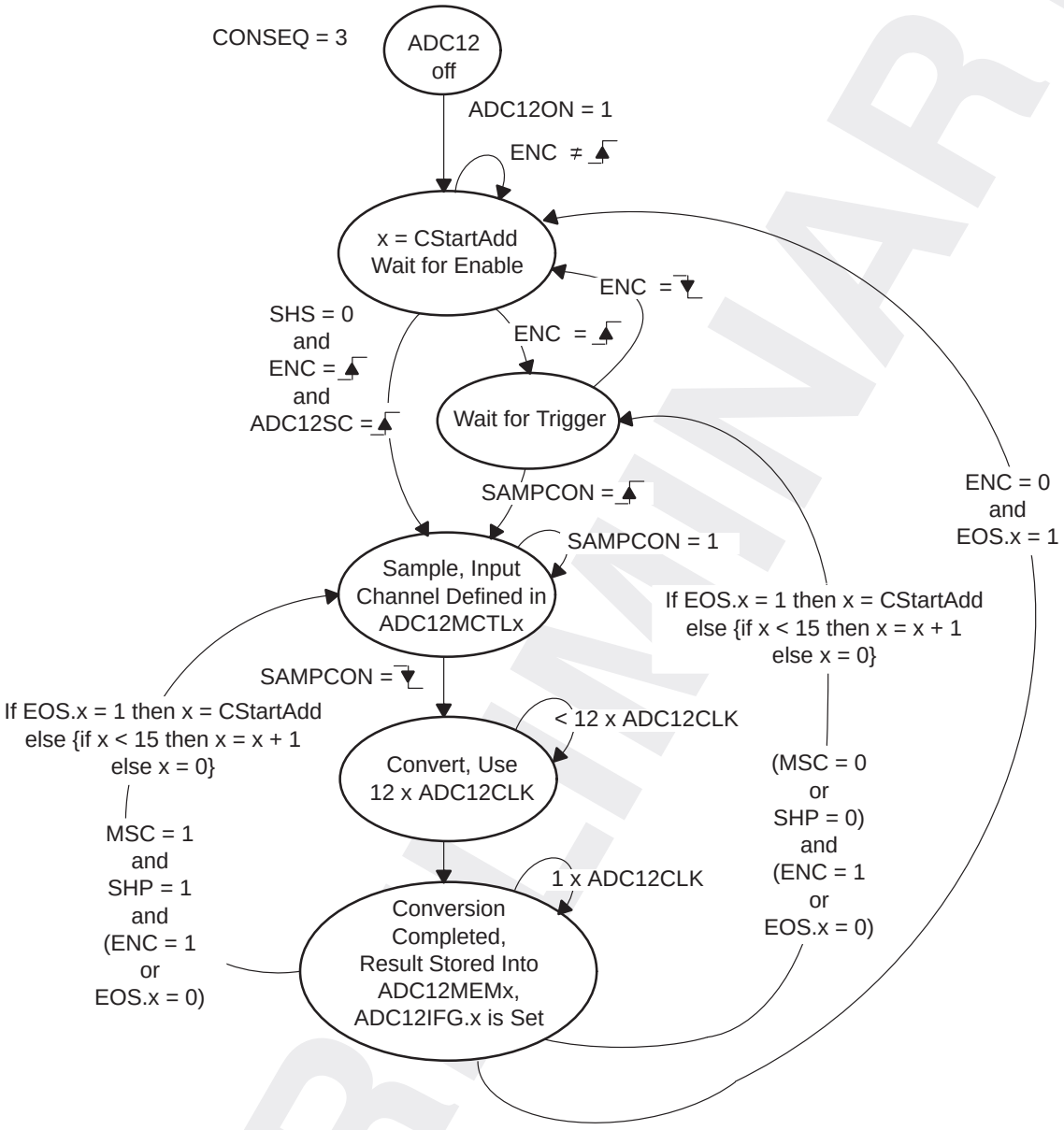
2) Reset ENC bit (ADC12CTL0.1). This stops the conversions after the current sequence is completed. The conversion results of all conversions in the sequence are stored in their appropriate ADC12MEMx register and the associated interrupt flags ADC12IFG.x are set.

3) Select repeat-single-channel mode (CONSEQ = 2) instead of the repeat-sequence-of-channel mode, and then select single-channel mode. The current conversion is completed normally. The current conversion result is loaded into register ADC12MEMx and the associated interrupt flags ADC12IFG.x are set. The data for x is somewhere between CStartAdd and the last register of the sequence.

4) Select single-channel mode (CONSEQ = 0) and reset enable-conversion bit ENC. The current conversion is stopped immediately. The data in memory register ADC12MEMx is unpredictable and the interrupt flag ADC12IFG.x may or may not be set. This method is generally not recommended.

An illustration of repeat-sequence-of-channels mode is shown in Figure 15–12.

Figure 15–12. Repeat-Sequence-of-Channels Mode



15.5.5 Switching Between Conversion Modes

Changing the mode of operation of the ADC12 while the converter is not actively running is done simply by selecting the new mode of operation with the CONSEQ bits. However, if the conversion mode is changed while the converter is actively running, intermediate and undesirable modes can be accidentally selected if both CONSEQ bits are changed in a single instruction. Therefore, the following mode changes should be avoided while the converter is running: $0 \rightarrow 3$, $1 \rightarrow 2$, $2 \rightarrow 1$, and $3 \rightarrow 0$.

The intermediate modes are caused by the asynchronous clocks for the CPU and the ADC12. These intermediate modes can be avoided simply by changing only one CONSEQ bit per instruction. For example, to change from mode 0 to mode 3 while the converter is actively running, the following instructions could be used:

```
BIS #CONSEQ_0,&ADC12CTL1    ; Example: 0 → 3, first
 ..............................; step is 0 → 1
BIS #CONSEQ_1,&ADC12CTL1 ...; second step is 1 → 3
```

Acceptable sequence modifications are: 0 → 1, 0 → 2, 1 → 0, 1 → 3, 2 → 0, 2 → 3, 3 → 1, and 3 → 2.

15.5.6 Power Down

The ADC12 incorporates two bits (ADC12ON and REFON) for power savings. ADC12ON turns on the A/D core and REFON turns on the reference generator. Each bit is individually controllable by software. The ADC12 is turned off completely if both bits are reset. The ADC12 registers are not affected by either of these bits and can be accessed and modified at any time (see the *ADC12 Control Registers* section). Note, however, that ADC12ON and REFON may only be modified if ENC=0.

Additionally, other ADC12 functions are automatically switched on and off as needed—if possible—to realize additional power savings – even while the ADC12 is running.

Caution! Do not power-down the converter or the reference generator while the converter is active. Conversion results will be false.
It is possible to disable the reference generator and the ADC12 by resetting bits ADC12ON and REFON before an active conversion or sequence of conversions has completed. For example, if the conversion mode is set to sequence-of-channels, and software resets the ENC bit immediately after the sequence begins, the ADC12ON and REFON bits can then be reset before the sequence completes. If this occurs, the ADC12 will be powered down immediately and the conversion results will be false.

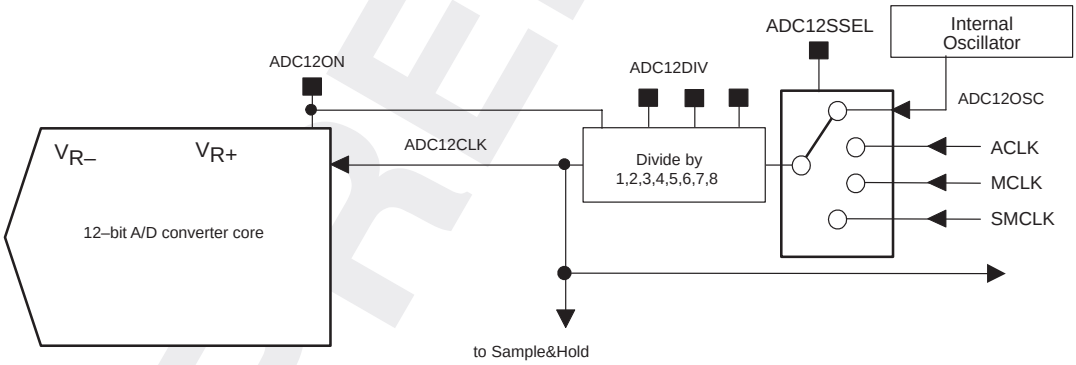
Caution! The following must be considered when turning the ADC12 and voltage reference on or off.
ADC12 turnon time: when the ADC12 is turned on with the ADC12ON bit, the turnon time noted in the data sheet ($t_{ADC12ON}$) must be observed before a conversion is started. Otherwise, the results will be false.
Reference-voltage settling Time: When the built-in reference is turned on with the VREFON bit, the settling timing noted in the data sheet must be observed before a conversion is started. Otherwise, the results will be false until the reference settles. Once all internal and external references have settled, no additional settling time is required when selecting or changing the conversion range for each channel.
Settling time of external signals: external signals must be settled before performing the first conversion after turning on the ADC12. Otherwise, the conversion results will be false.

15.6 Conversion Clock and Conversion Speed

The conversion clock for the ADC12 (ADC12CLK shown in Figure 15–13) can be selected from several sources and can be divided by any factor from 1 – 8. The ADC12CLK is used for the A/D conversion and to generate the sampling period [if pulse-sampling mode is selected (SHP=1)]. Possible clock sources are the internal oscillator (ADC12OSC), ACLK, MCLK, and SMCLK.

The internal oscillator generates the ADC12OSC signal and is in the 5-MHz range (see device data sheet for specifications). The internal-oscillator frequency will vary with individual devices, supply voltage, and temperature. A stable clock source should be used for the conversion clock when accurate conversion timing is required.

Figure 15–13. The Conversion Clock ADC12CLK



The conversion starts with the falling edge of the sample signal SAMPCON (see the *Sampling* section and Figure 15–14). Thirteen conversion clocks (ADC12CLK) are required to complete a conversion. The conversion time is:

$$t_{conversion} = 13 \times (ADC12DIV/f_{ADC12CLK})$$

Where ADC12DIV is any integer from 1 to 8. The ADC12CLK frequency must not exceed the maximum and minimum frequencies specified in the data sheet. Either violation may result in inaccurate conversion results.

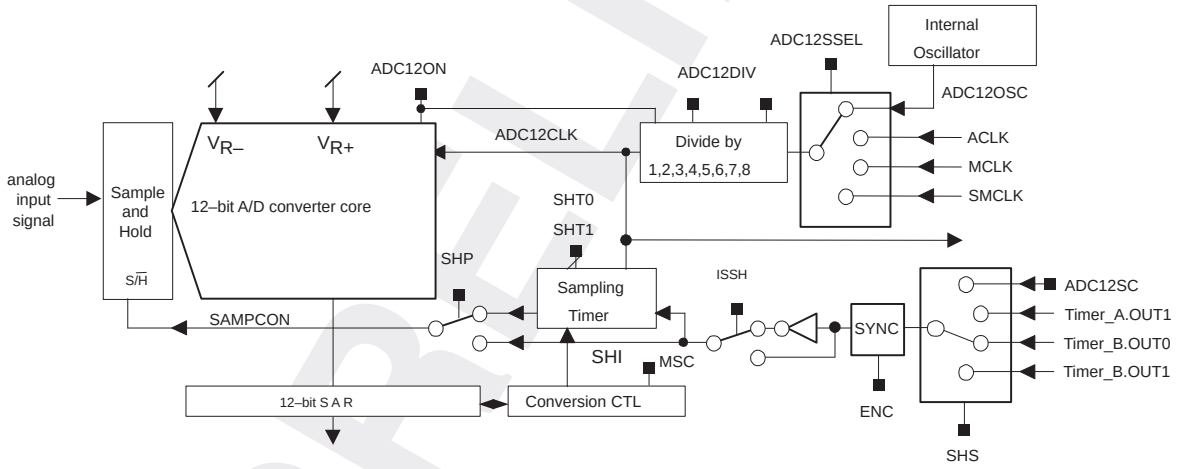
Note: Availability of ADC12CLK During Conversion

Users must ensure that the clock chosen for ADC12CLK remains active until the ADC12 can complete its operation. If the clock is removed while the ADC12 is active, the operation can not be completed and the end-of-conversion feedback to the program is not possible.

15.7 Sampling

The ADC12 sample-and-hold (S/H) circuitry (shown in Figure 15–14) is flexible and configurable. The configuration is done by software via control bits in the ADC12CTL0 and ADC12CTL1 registers. Configuration and operation of the S/H circuitry is discussed in this section.

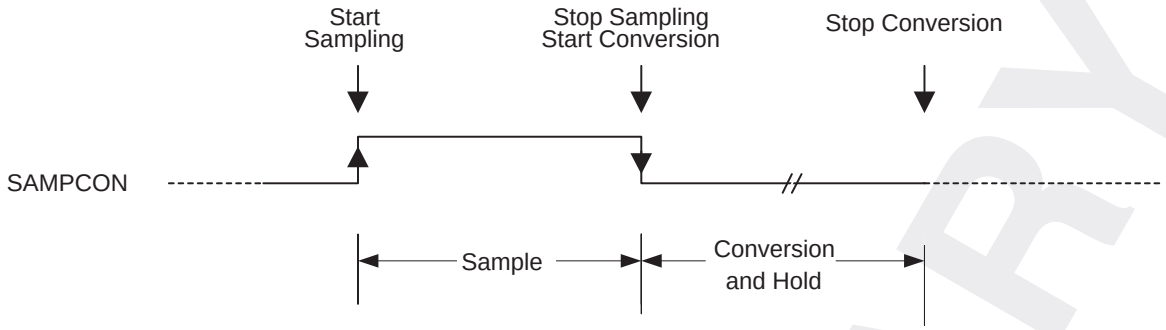
Figure 15–14. The Sample-and-Hold Function



15.7.1 Sampling Operation

The sample-and-hold circuitry samples the analog signal when the sampling signal SAMPCON (see Figure 15–14) is high. Conversion starts immediately with the falling edge of SAMPCON. The sample-and-hold holds the signal value when SAMPCON is low. Conversion takes 13 ADC12CLK cycles (see Figure 15–15).

Figure 15–15. Sample and Conversion, Basic Signal Timing



The analog input signal must be valid and steady during the sampling period in order to obtain an accurate conversion. It is also desirable not to have any digital activity on any adjacent channels during the whole conversion period to ensure that errors due to supply glitching, ground bounce, or crosstalk do not corrupt the conversion results.

In addition, gains and losses in internal charge limit the hold time. The user should ensure that the data sheet limits are not violated. Otherwise, the sampled analog voltage may increase or decrease, resulting in false conversion values.

15.7.2 Sample Signal Input Selection

The SAMPCON signal, which controls sample timing and the start of a conversion, may be sourced by one of several signals. SAMPCON may be sourced directly from one of the signals available at the input selection switch (see Figure 15–16), further called sample-signal input, or from the integrated sampling timer. When the sampling timer is used to source SAMPCON, the sample-signal input is used to trigger the sampling timer.

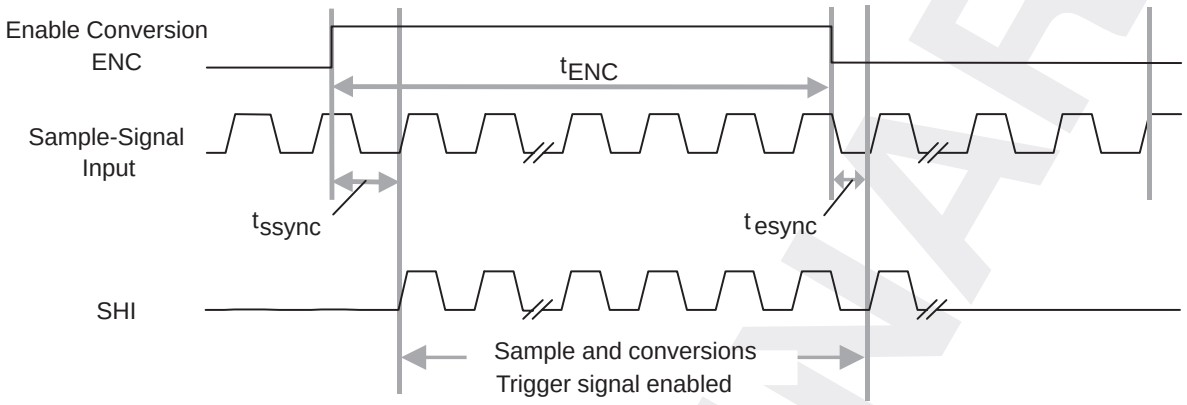
The sample-signal input is selected by the SHS bits in ADC12CTL1. There are four choices for the sample-signal input: ADC12SC, Timer_A.OUT1, Timer_B.OUT0, and Timer_B.OUT1. The polarity of the sample-signal input may be selected by the ISSH bit (see Figure 15–16). Also, the sample-signal input is passed to the sampling timer or to the SAMPCON signal under control of the ENC bit. This is discussed in detail further ahead.

ADC12SC is a control bit located in ADC12CTL0. Its value is set by software. Depending on the selected sampling mode, this bit allows the software to either start a sample-and-conversion (S/C) cycle (SHP=1), or to completely control the sampling period (SHP=0).

The sample-signal input can be asynchronous to a conversion-enable, and is synchronized and enabled by the ENC bit. Without synchronization, the first sampling period after the ENC bit is set could be erroneous, depending on where the ENC bit is set within the cycle of the input signal. In Figure 15–18, for example, note that the ENC bit is set in the middle of a high pulse from the sample-signal input. If the sample-input signal were simply passed directly to the S/H, the first conversion of the example would be erroneous because the first sampling period is too short.

To prevent this problem, synchronization logic is implemented in the sample input selection switch. This ensures that the first sample-and-conversion cycle begins with the first rising edge of the sample-input signal applied *after* the ENC bit is set. Additionally, the last sample-and-conversion begins with the first rising edge of the sample-input signal *after* ENC has been reset.

Figure 15–16. Synchronized Sample and Conversion Signal With Enable Conversion



15.7.3 Sampling Modes

The sampling circuitry has two modes of operation: pulse-sampling mode and extended-sampling mode. In pulse-sampling mode, the sample-signal input (selected by the SHS bits in ADC12CTL1) is used to trigger the internal sampling timer, and the actual sample timing signal (SAMPCON) is then generated by the sampling timer and is an integer multiple of the ADC12CLK signal.

In extended-sampling mode, the sampling-signal input bypasses the sample timer and is used to source SAMPCON directly, therefore completely controlling the sample timing—asynchronously to ADC12CLK. Note that 13 ADC12CLK cycles are still required to complete one conversion.

15.7.3.1 Pulse-Sample Mode

In the pulse-sample mode, the sample-input signal, selected by the SHS bits, triggers the sampling timer with its rising edge. The sampling timer then generates the sample timing. The sampling time is programmable by the SHT0 or SHT1 bits located in ADC12CTL0. When conversion-memory registers ADC12MEM0 to ADC12MEM7 are selected to store the conversion result(s), the SHT0 bits are used to program the sampling time. When conversion-memory registers ADC12MEM8 to ADC12MEM15 are selected for the conversion data, the SHT1 bits are used to program the sampling timing. Therefore, it is possible to program two different sampling times for a sequence of conversions by using both upper and lower conversion-memory registers in the sequence. This feature is useful when different external-source impedance conditions exist and require different sample timings.

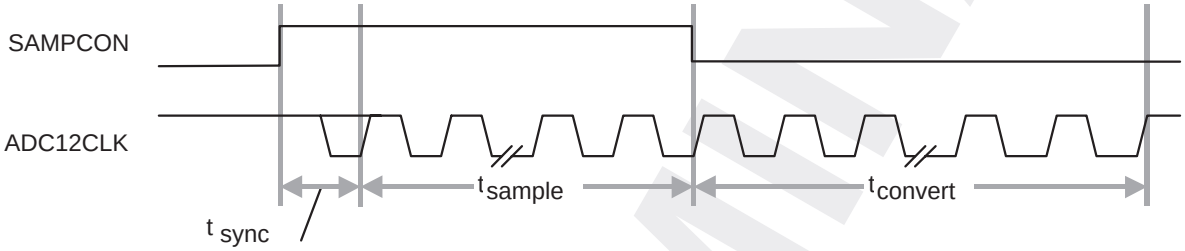
In pulse-sampling mode, sampling time is a multiple of the ADC12CLK x 4, and is calculated by:

tsample = 4 x $t_{ADC12CLK}$ x SHTx

SHTx is determined by bits SHT0 or SHT1 (see table in *Control Registers ADC12CTL0 and ADC12CTL1* section*).*

The sampling signal SAMPCON remains in the sampling state (high) for the synchronization time t_{sync} and the selected sample time t_{sample}, as shown in Figure 15–17. The conversion takes 13 × ADC12CLK cycles ($t_{convert}$). It is important to note that after a sample-and-conversion cycle has been triggered by the sample-input signal, additional triggers (via a rising edge on the sample-input signal) will be missed/ignored until the prior sample-and-conversion cycle is completed.

Figure 15–17. Conversion Timing, Pulse-Sample Mode



An example of the pulse-sample mode configuration is shown in Figure 15–18. The selected input signal source is Timer_B.OUT0. The timing for the example is shown in Figure 15–19.

Figure 15–18. Pulse-Sample Mode Example Configuration

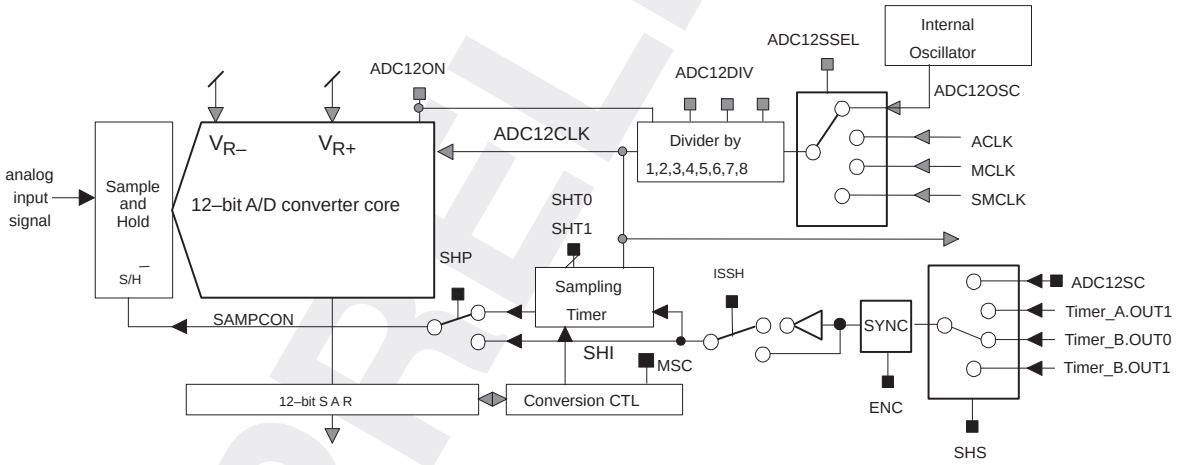
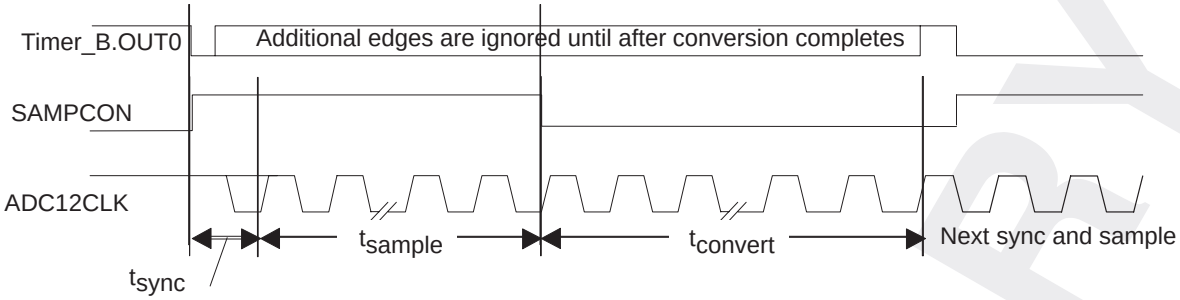


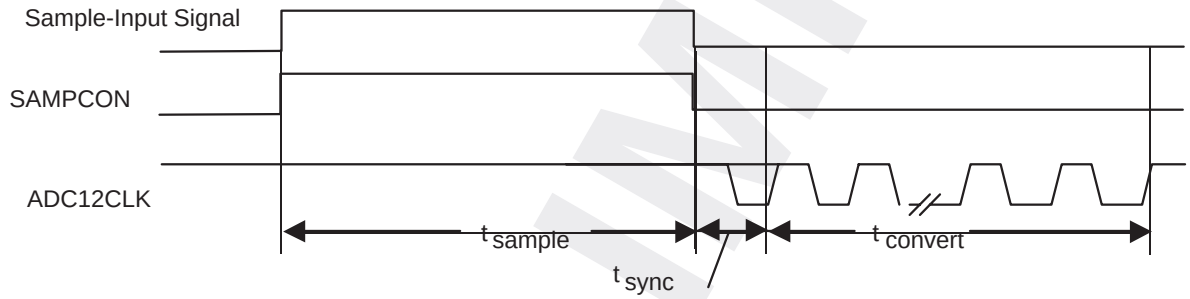
Figure 15–19. Pulse-Sample Mode Example Timing



15.7.3.2 Extended-Sample Mode

In extended-sample mode, the input signal selected by the SHS bits is used to control the sampling (SAMPCON signal) directly. The internal sampling timer is not used. As shown in Figure 15–20, the sampling period is active while SAMPCON is high. Hold mode is active when SAMPCON is low. The conversion starts with the falling edge of SAMPCON after a synchronization time t_{sync}. The conversion takes 13 × ADC12CLK ($t_{convert}$).

Figure 15–20. Conversion Timing for Extended-Sample Mode



The extended-sample mode allows total control of the sampling period and the start of a conversion. The extended-sample mode is useful in applications that require an extended sampling period to accommodate different input-source impedances, or in applications where the maximum sampling period supplied by the internal sampling timer is insufficient.

An example of the extended-sample-mode configuration is shown in Figure 15–21. The selected input signal source is Timer_B.OUT0. The timing for the example is shown in Figure 15–22.

Figure 15–21. Extended-Sample Mode Example Configuration

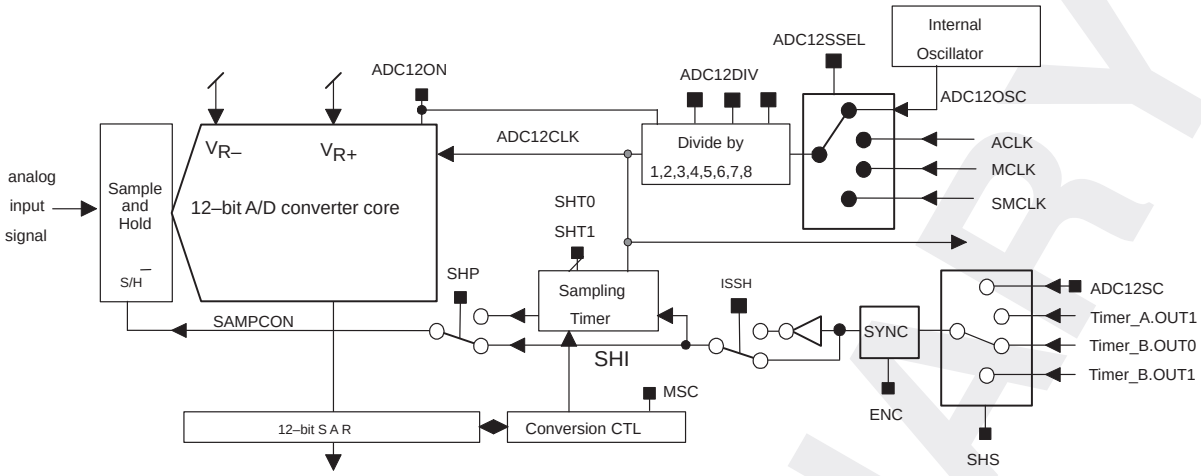
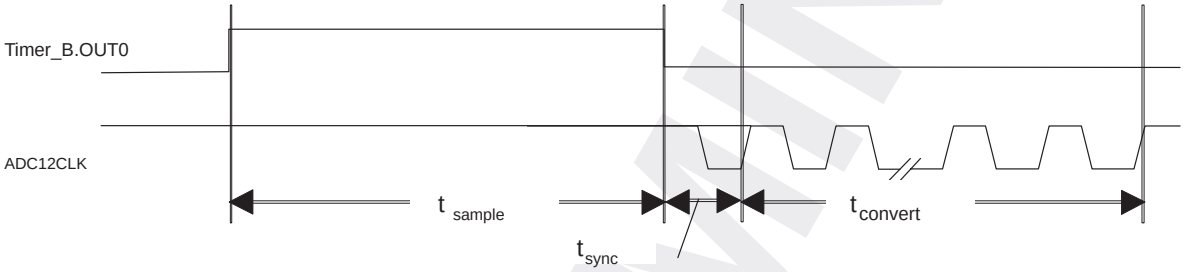


Figure 15–22. Extended-Sample Mode Example Timing



15.7.4 Using the MSC Bit

The multiple-sample-and-conversion (MSC) control bit is not used if the sample signal SAMPCON is generated without the sampling timer. However, when the sampling timer is used to generate the SAMPCON signal and the operating mode is other than single-channel single-conversion (CONSEQ > 0), the MSC bit can be used to configure the converter to perform the successive conversions automatically and as quickly as possible.

If MSC = 0, then a rising edge of the SHI signal is required to trigger each sample-and-conversion, regardless of what mode the converter is in. When MSC = 1 and CONSEQ > 0, the first rising edge of the SHI signal triggers the first conversion, but successive conversions are triggered automatically as soon as the prior conversion is completed. Additional rising edges on SHI are ignored until the sequence is completed or until the ENC bit is toggled (depending on mode). The function of the ENC bit is unchanged when using the MSC bit. See Figures 15–23 and 15–24.

Figure 15–23. Use of MSC Bit With Nonrepeated Modes

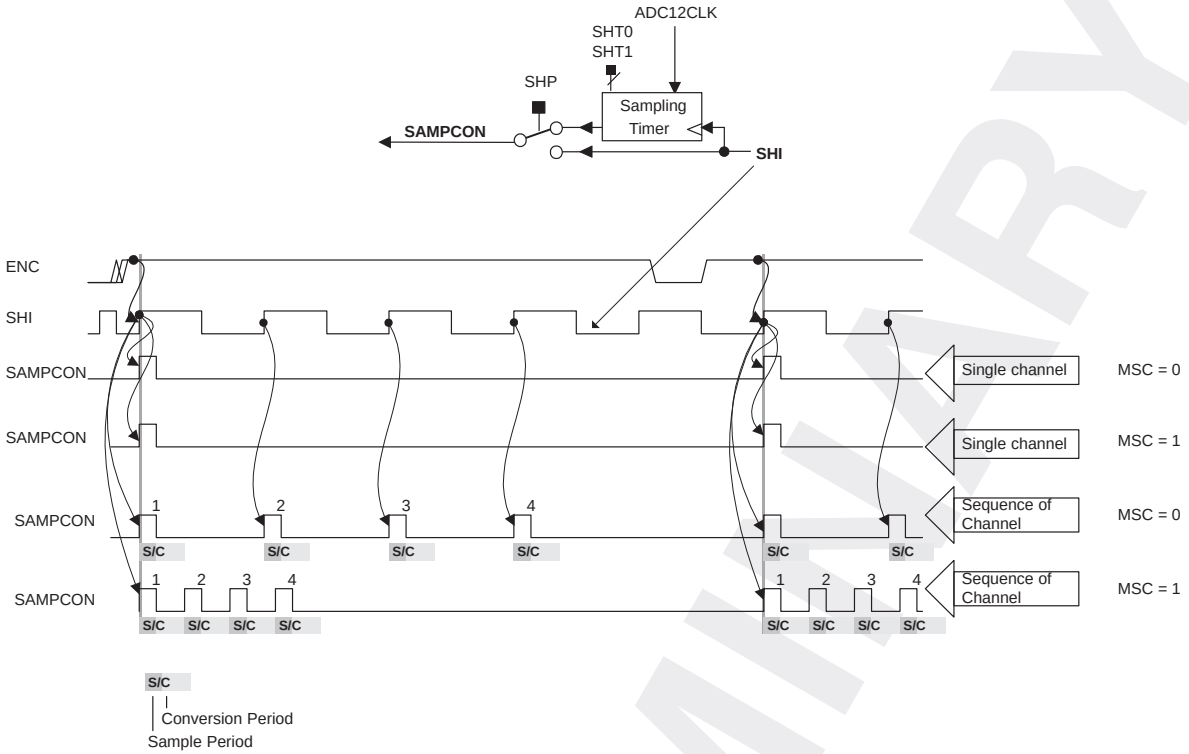
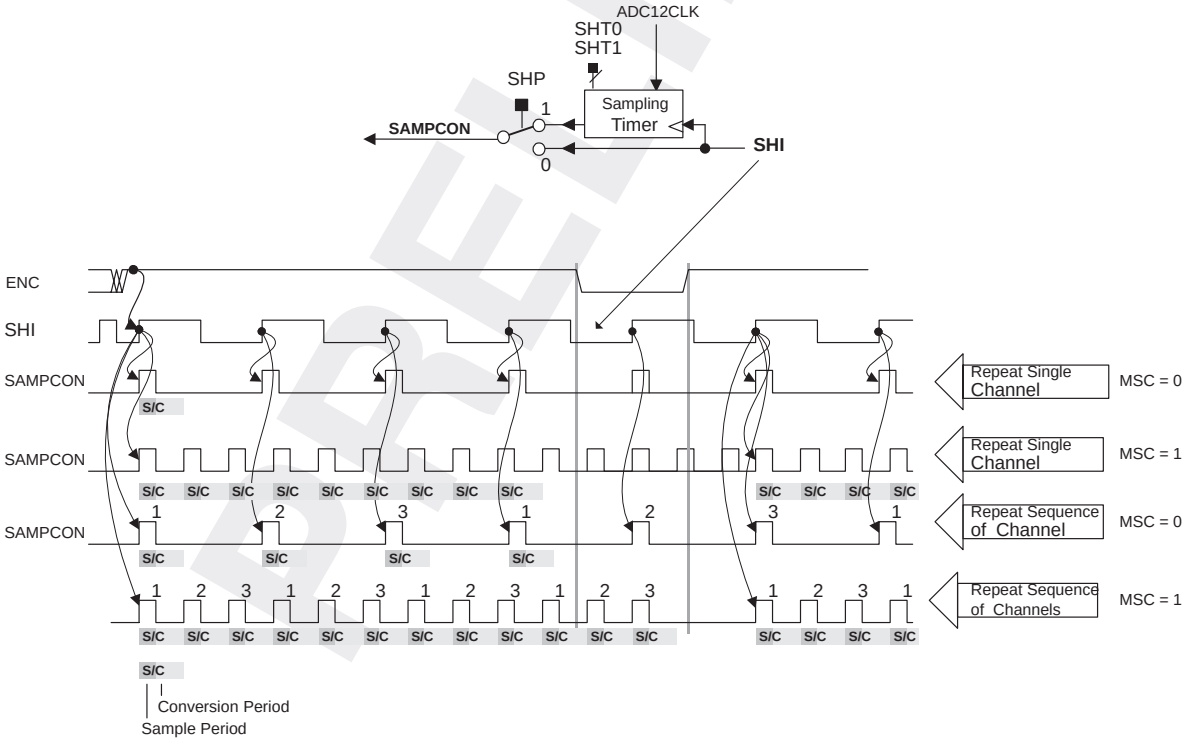


Figure 15–24. Use of MSC Bit With Repeated Modes



15.7.5 Sample Timing Considerations

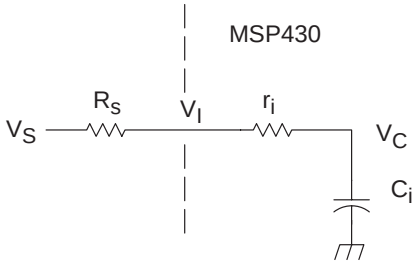
The A/D converter uses the charge redistribution method. Thus, when the inputs are internally switched to sample the input analog signal, the switching action causes displacement currents to flow into and out of the analog inputs. These current spikes or transients occur at the leading and falling edges of the sample pulse, and usually decay and settle before causing any problems because typically the external time constant is less than that presented by the internal effective RC. Internally, the analog inputs see an effective maximum nominal RC of a 30 pF (C-array) capacitor in series with a 2-kΩ resistor (R_{on} of switches). However, if the external dynamic-source impedance is large, then these transients may not settle within the allocated sampling time to ensure 12 bits of accuracy.

It is imperative that the proper sample timing be used for accurate conversions. The next section discusses how to calculate the sample timing.

15.7.5.1 Simplified Sample-Timing Analysis

Using the equivalent circuit shown in Figure 15–25, the time required to charge the analog-input capacitance from 0 to VS within 1/2 LSB can be derived as follows.

Figure 15–25. Equivalent Circuit



V_I = Input voltage at pin Ax
V_S = External driving-source voltage
R_s = Source resistance (must be real at input frequency)
r_i = Input resistance (MUX-on resistance)
C_i = Input capacitance
V_C = Capacitance-charging voltage

The capacitance-charging voltage is given by:

$$V_C = V_S\left(1–EXP\left(\frac{–tc}{R_t \times C_i}\right)\right) \quad (1)$$

Where:

$R_t = R_S + Z_i$
t_c = Cycle time

The input impedance Z_i is ~1 kΩ at 3.0 V, and is higher (~ 2 kΩ) at 1.8 V. The final voltage to 1/2 LSB is given by:

$$V_C\ (1/2LSB) = V_S–\left(\frac{V_S}{8192}\right) \quad (2)$$

Equating equation 1 to equation 2 and solving for cycle time tc gives:

$$V_S–\left(\frac{V_S}{8192}\right) = V_S\left(1–EXP\left(\frac{–tc}{R_t \times C_i}\right)\right) \quad (3)$$

and the time to charge to 1/2 LSB (minimum sampling time) is:

$$t_{ch}(1/2\ LSB) = R_t \times C_i \times In(8192)$$

Where:

$$In(8192) = 9.011$$

Therefore, with the values given, the time for the analog input signal to settle is:

$$t_{ch}(1/2\ LSB) = (R_S + 1\ k\Omega) \times C_i \times 9.011 \quad (4)$$

This time must be less than the sampling time.

If the pulse-sampling mode is used, the maximum ADC12CLK frequency is:

$$\max[f(ADC12CLK)] = \frac{SHTx}{t_{ch}(1/2LSB)} \quad (5)$$

This frequency must not exceed the maximum ADC12CLK frequency specified in the data sheet.

15.8 ADC12 Control Registers

Five control registers, sixteen conversion-memory registers, and sixteen conversion-memory control registers are used to configure the ADC12:

Register	Short Form	Register Type	Address	Initial State
ADC control register 0	ADC12CTL0	Read/write	01A0h	Reset with POR
ADC control register 1	ADC12CTL1	Read/write	01A2h	Reset with POR
ADC interrupt flag register	ADC12IFG	Read/write	01A4h	Reset with POR
ADC interrupt enable register	ADC12IE	Read/write	01A6h	Reset with POR
ADC interrupt vector word	ADC12IV	Read	01A8h	Reset with POR
ADC memory 0 to ADC memory 15	ADC12MEM0 ADC12MEM15	Read	0140h 015Eh	Unchanged
ADC memory control 0 to ADC memory control 15	ADC12MCTL0 ADC12MCTL15	Read	080h 08Fh	Reset with POR

Note: All registers may be accessed by any instruction subject to register-access restrictions.

15.8.1 Control Registers ADC12CTL0 and ADC12CTL1

All control bits of ADC12CTLx are reset during POR. Most of the control bits in registers ADC12CTL0, ADC12CTL1, and ADC12MCTLx can only be modified if ENC is reset. These bits are marked . All other bits can be modified at any time.

The control bits of control register ADC12CTL0 and ADC12CTL1 are:

ADC12CTL0 01A0h

15							8	7							0
SHT1				SHT0				MSC	2_5V	REF ON	ADC12 ON	ADC12 OVIE	ADC12 TOVIE	ENC	ADC12 SC
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

ADC12SC bit0 Sample and convert. The ADC12SC bit can be used to control the conversion with software if ENC is set. It is recommended to have ISSH=0. If the sampling signal SAMPCON is generated by the sampling timer (SHP=1), changing the ADC12SC bit from 0 to 1 starts the sample-and-conversion operation. When the A/D conversion is complete (BUSY=0) the ADC12SC bit is automatically reset.

If the sample signal is directly controlled by ADC12SC (SHP=0), then the high level of the ADC12SC bit defines the sample time. The conversion starts once it is reset.

All automatic-sequence functions (CONSEQ={1,2,3}) and multiple sample-and-conversion functions (MSC=1) are executed normally. Therefore, when using ADC12SC the software must ensure that the frequency of the timing of the ADC12SC bit meets the applicable timing requirements.

NOTE: The start of a conversion by software (SHS=0, in ADC12CTL1) is possible by setting both ENC and ADC12SC control bits within one instruction.

ENC bit1 Enable conversion. The software (via ADC12SC) or external signals can start a conversion only if the enable-conversion bit ENC is high. Most of the control bits in ADC12CTL0 and ADC12CTL 1, and all bits in ADCMCTL.x may be changed only if ENC is low.

0 : No conversion can be started. This is the initial state.
1 : The first sample and conversion starts with the first rising edge of the SAMPCON signal. The selected operation proceeds as long as ENC is set.

CONSEQ=0, ADC12BUSY=1, ENC= 1→ 0:
In this mode, if ENC is reset, the current conversion is immediately stopped. The conversion results are unpredictable.

CONSEQ≠0, ADC12BUSY=x, ENC= 1→ 0:
In these modes, if ENC is reset, the current conversion or sequence is completed and the conversion results are valid. The conversion activities are stopped after the current conversion or sequence is completed.

ADC12TOVIE bit2 Conversion-time-overflow interrupt enable.
The timing overflow happens if another sample-and-conversion is requested while the current conversion is not completed. This is independent of the conversion modes selected by CONSEQ. If the timing overflow vector is generated and the timing overflow interrupt enable flag ADC12TOVIE and the general interrupt enable bit GIE are set, an interrupt service is requested. There is no individual interrupt flag. See the *ADC12 Interrupt Vector Register ADC12IV* section for more information on ADC12 interrupts.

ADC12OVIE bit3 Overflow-interrupt enable. Individual enable for the overflow-interrupt vector.
The overflow happens if a conversion result is written into an ADC memory ADC12MEMx but the previous result was not read. An interrupt service is requested if the overflow vector is generated, the overflow-interrupt-enable flag ADC12OVIE is set, and the general-interrupt-enable bit GIE is set. There is no individual interrupt flag. See the *ADC12 Interrupt Vector Register ADC12IV* section for more information on ADC12 interrupts.

ADC12ON bit4 Turn on the 12-bit ADC core. Settling-time constraints must be met when the ADC12 core is powered up.

0: Power consumption of the core is off. No conversion will be started.
1: ADC core is supplied with power. If no A/D conversion is needed, ADC12ON can be reset to conserve power.

REFON bit5 Reference voltage ON.

0: The internal-reference voltage is switched off. No power is consumed from the reference-voltage generator.
1: The internal-reference voltage is switched on. The reference-voltage generator consumes power. When the reference generator is switched on, the settling time of the reference voltage must be completed before the first sampling and conversion is started.

2_5V bit6 Reference-voltage level.

0: The internal reference voltage is 1.5V, if REFON = 1.
1: The internal reference voltage is 2.5V, if REFON = 1.

MSC bit7 Multiple sample and conversion. Valid only when the sample timer is selected to generate the SAMPCON signal (SHP=1) and the A/D mode is chosen as repeat-single-channel, sequence-of-channel or repeat-sequence-of-channels (CONSEQ≠0).

0: The sampling timer requires a rising edge of the SHI signal to trigger each sample-and-conversion.
1: The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversion are performed automatically as soon as the prior conversion is completed—without additional rising edges of SHI. Additional rising edges of SHI are ignored until the sequence has completed or the ENC bit has been toggled (depending on mode).

SHT0 bits 8–11 Sample-and-hold Time0. These bits define the sample timing for conversions whose results are stored in conversion-memory registers ADC12MEM0 to ADC12MEM7.

The sample time is a multiple of the ADC12CLK × 4:

$t_{sample} = 4 \times ADC12CLK \times n$

SHT0	0	1	2	3	4	5	6	7	8	9	10	11	12–15
n	1	2	4	8	16	24	32	48	64	96	128	192	256

SHT1 bits 12–15 Sample-and-hold Time1. These bits define the sample timing for conversions whose results are stored in conversion-memory registers ADC12MEM8 to ADC12MEM15.

The sample time is a multiple of the ADC12CLK × 4:

$t_{sample} = 4 \times ADC12CLK \times n$

SHT1	0	1	2	3	4	5	6	7	8	9	10	11	12–15
n	1	2	4	8	16	24	32	48	64	96	128	192	256

ADC12CTL1
01A2h

15								8	7						0
CSStartAdd				SHS		SHP	ISSH	ADC12DIV		ADC12SSEL		CONSEQ		ADC12 BUSY	
rw –(0)	rw –(0)	rw –(0)	rw –(0)	rw –(0)	rw –(0)	rw –(0)	rw –(0)	rw –(0)	rw –(0)	rw –(0)	rw –(0)	rw –(0)	rw –(0)	rw –(0)	r –(0)

ADC12BUSY bit0 The ADC12BUSY bit indicates an active sample or conversion operation. It is used specifically when the conversion mode is single-channel-single-conversion, because if the ENC bit is reset in this mode, the conversion stops immediately and the results are invalid. Therefore, the ADC12BUSY bit should be tested to verify that it is 0 before resetting the ENC bit when in single-channel-single-conversion mode.

The busy bit is not useful in all other operating modes because resetting the ENC bit does not immediately affect any other mode.

0: No operation is active.
1: A sample period, conversion or conversion sequence is active.

CONSEQ bits 1–2 The CONSEQ bits select the conversion mode. Repeat mode is on if the CONSEQ.1 is set.

0: Single-channel-single-conversion mode. One single channel is converted once.
1: Sequence-of-channels mode. A sequence of conversions is executed once.
2: Repeat-single-channel mode. Conversions on a single channel are repeated until CONSEQ is set to 0 or 1.
3: Repeat-sequence-of-channels. A sequence of conversions is repeated until CONSEQ is set to 0 or 1.

NOTE: See also section *Conversion Modes.*

ADC12SSEL	bits 3–4	Select the clock source for the converter core 0: ADC12 internal oscillator, ADC12OSC 1: ACLK 2: MCLK 3: SMCLK
ADC12DIV	bits 5–7	Select the division rate for the clock source selected by ADC12SSEL bits. 0 to 7: Divide selected clock source by 1 to 8 The divider's output signal name is ADC12CLK. Thirteen of these clocks are required for a conversion.
ISSH	bit8	Invert sample-input signal. 0: The sample-input signal is not inverted. 1: The sample-input signal is inverted.
SHP	bit9	The SHP bit selects the source of the sampling signal (SAMPCON) to be either the output of the sampling timer or the sample-input signal directly. 0: SAMPCOM signal is sourced directly from the sample-input signal. 1: SAMPCON signal is sourced from the sampling timer. The rising edge of the sample-input signal triggers the sampling timer.
SHS	bits 10–11	Source select for the sample-input signal. 0: Control bit ADC12SC is selected. 1: Timer_A.OUT1 2: Timer_B.OUT0 3: Timer_B.OUT1
CStartAdd	bits 12–15	Conversion start address CStartAdd is used to define which ADC12 conversion-memory register (ADC12MEMx) is used for a single conversion or for the first conversion in a sequence of conversions. The value of CStartAdd is 0 to 0Fh, corresponding to ADC12MEM0 to ADC12MEM15. Since there is one corresponding conversion-memory control register (ADC12MCTLx) for each conversion-memory register (ADC12MEMx), CStartAdd also points to the corresponding ADC12MCTLx register.

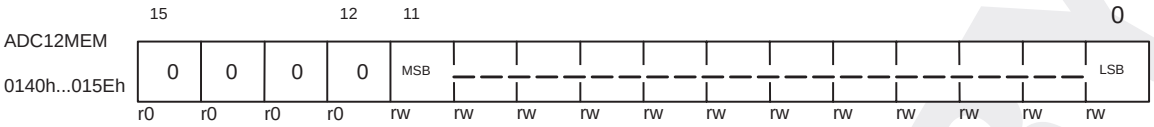
Warning : Modifying ADC control register during active conversion

The enable conversion control bit (ENC) in the ADC12CTL0 register protects most bits from modification during an active conversion. However, some bits that are necessary for proper completion of active conversions and interrupt enable bits can be modified independently of ENC. The user must use caution when modifying theses bits to ensure an active conversion is not corrupted, or to not use corrupted data.

To avoid corrupting any active conversions, stop the conversion, wait for the busy bit to be reset, reset the ENC bit, then modify the control bits.

15.8.2 Conversion-Memory Registers ADC12MEMx

There are sixteen conversion-memory registers ADC12MEMx as follows:



ADC12MEM0, to ADC12MEM15 — bits 0–15

Conversion results. The 12-bit conversion results are right-justified and the four MSBs are always read as 0.

The ADC12OV interrupt flag will be set in time to indicate that a overflow situation occurred. Software can detect it if it reads the conversion result and then tests for overflow condition. The corresponding interrupt flag is reset if ADC12MEMx is accessed.

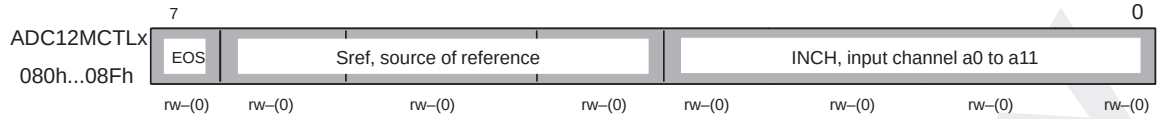
Warning : SOFTWARE WRITE TO REGISTER ADC12MEMx

TYPICALLY, SOFTWARE SHOULD NOT WRITE TO THE CONVERSION RESULT REGISTERS ADC12MEMX. IF SOFTWARE WRITES TO ONE OF THESE REGISTERS WHILE THE ADC12 IS ATTEMPTING TO WRITE TO THE SAME REGISTER, THE DATA IN THE REGISTER WILL BE UNPREDICTABLE. IF SOFTWARE ENSURES THAT IT IS WRITING TO A CONVERSION RESULT REGISTER THAT IS NOT BEING ACCESSED BY THE ADC12, THEN THE WRITE COMPLETES NORMALLY AND THE DATA IS WRITTEN CORRECTLY. THE ASSOCIATED INTERRUPT FLAG IS RESET.

15.8.3 Control Registers ADC12MCTLx

Each conversion-memory register ADC12MEMx has its own control register ADC12CTLx. The conversion-memory registers hold the conversion results, and the control register for each conversion-memory register selects basic conversion conditions such as selecting the analog channel, the reference voltage sources for V_{R+} and V_{R-}, and indicating the end of a sequence.

All control bits in ADC12CTLx are reset during POR (see Chapter 3 for POR details). The control registers ADC12MCTL.x can be modified only if the enable conversion control bit ENC is reset. Any instruction that writes to an ADC12MCTL register while the ENC bit is set will have no effect.



INCH, bits 0–3 The INCH (input channel) bits select one of eight external, or one of four internal analog signals for conversion.

0–7: a0 to a7
8: Ve_{REF+}
9: V_{REF-}/Ve_{REF-}
10: Temperature diode
11–15: $(AV_{CC} – AV_{SS}) / 2$

Note: Selecting channel 10 automatically turns on the on-chip reference generator for a voltage source for the temperature diode. However, it does not enable the V_{REF+} output or effect the reference selections for the conversion.

Sref, bits4–6 The Sref bits select one of six reference voltage combinations used for conversion. The conversion is done between the selected voltage range V_{R+} and V_{R-}.

0: $V_{R+} = AV_{CC}$ and $V_{R-} = AV_{SS}$
1: $V_{R+} = V_{REF+}$ and $V_{R-} = AV_{SS}$
2,3: $V_{R+} = Ve_{REF+}$ and $V_{R-} = AV_{SS}$
4: $V_{R+} = AV_{CC}$ and $V_{R-} = V_{REF-}$/ Ve_{REF-}
5: $V_{R+} = V_{REF+}$ and $V_{R-} = V_{REF-}$/ Ve_{REF-}
6,7: $V_{R+} = Ve_{REF+}$ and $V_{R-} = V_{REF-}$/ Ve_{REF-}

EOS, bit7 The end-of-sequence bit, when set, indicates the last conversion in a sequence of conversions.

Note: A sequence will roll over from ADC12MEM15/ADC12MCTL15 to ADC12MEM0/ADC12MCTL0 if the EOS bit in ADC12MCTL15 is not set.

Note: If none of the EOS bits is set and a sequence-of-channels (CONSEQ={1,3}) is selected, resetting the ENC bit will not stop the sequence. To stop the sequence, first select a single-channel mode (CONSEQ={0,2}) and then reset ENC. See also the ENC bit description.

15.8.4 ADC12 Interrupt Flags ADC12IFG.x and Interrupt-Enable Registers ADC12IEN.x

There are 16 ADC12IFG.x interrupt flags, 16 ADC12IE.x interrupt enable bits, and one interrupt vector word. The interrupt flags and enable bits are associated with the 16 ADC12MEMx registers.

All interrupt flags and interrupt-enable bits are reset during POR.

ADC12IFG

0184h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC IFG.15	ADC IFG.14	ADC IFG.13	ADC IFG.12	ADC IFG.11	ADC IFG.10	ADC IFG.9	ADC IFG.8	ADC IFG.7	ADC IFG.6	ADC IFG.5	ADC IFG.4	ADC IFG.3	ADC IFG.2	ADC IFG.1	ADC IFG.0
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

ADC12IFG.x, bits 0–15 The ADC12IFG.x interrupt flag is set if a conversion-result register ADC12MEMx is loaded with the result of a conversion. The range for x is 0 to 15.

The interrupt flags are reset if their corresponding ADC12MEMx conversion-result register is accessed. To enable correct handling of overflow conditions, they are not reset by accessing the interrupt vector word ADC12IV. The overflow condition exists if another conversion result is written to ADC12MEMx and the corresponding ADC12IFG.x is not reset.

ADC12IE

01A6h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC IE.15	ADC IE.14	ADC IE.13	ADC IE.12	ADC IE.11	ADC IE.10	ADC IE.9	ADC IE.8	ADC IE.7	ADC IE.6	ADC IE.5	ADC IE.4	ADC IE.3	ADC IE.2	ADC IE.1	ADC IE.0
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

ADC12IE.x, bits 0–15 The ADC12IE.x interrupt-enable bit enables or disables the interrupt-request service generated if the corresponding interrupt flag ADC12IFG.x is set. The range for x is 0 to 15.

15.8.5 ADC12 Interrupt Vector Register ADC12IV

The 12-bit ADC has one interrupt vector to assist the handling of the 18 possible interrupt flags. Each of the 18 interrupt flags is prioritized and a unique vector word is generated according to the highest-pending interrupt. The priorities and corresponding vector-word values are shown in Table 15–3. Overflow flag ADC12OVIFG has the highest priority, followed by timing-overflow flag ADC12TOVIFG, and then by the interrupt flags for each conversion-memory register ADC12IFG.0 to ADC12IFG.15.

The highest-pending interrupt flag generates a number from 0 (no interrupt is pending) to 36. This encoded number can be added to the program counter to automatically enter the software routine for handling each specific interrupt (see software example, section 15.8.5.1.

An interrupt request is immediately generated if an interrupt flag is pending (ADC12IV≠0), if the corresponding interrupt enable bit (ADC12OVIE, ADC12TOVIE, or ADC12IE.x) is set, and if the general interrupt enable bit GIE is set. When an interrupt request is generated, the service is requested by the highest-priority interrupt that is enabled.

It is important to note that ADC12OVIFG and ADC12TOVIFG are reset automatically when either is the highest pending interrupt and the ADC12IV register is accessed. For example, if both are pending simultaneously, ADC12OVIFG will be reset automatically with the first access of ADC12IV, and ADC12TOVIFG will be reset automatically with the next access to the ADC12IV (assuming ADC12OVIFG was not set again). However, flags ADC12IFG.x must be reset by software or reset by accessing the corresponding conversion-memory register ADC12MEMx.

Also note that the flags ADC12OVIFG and ADC12TOVIFG can not be accessed by software. They are visible only via the interrupt vector word ADC12IV data.

Table 15–3. ADC12IV Interrupt-Vector Values

ADC Interrupt Flags ADC12IFG																ADC12TOV	ADC12OV	ADC12IV
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	2
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	0	4
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	0	0	6
x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	0	0	0	8
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
x	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36

Note: Writing to Read Only Register ADC12IV

When a write to vector word register ADC12IV occurs, the highest-pending interrupt flag is reset. Therefore, the interrupt event is missed.

15.8.5.1 ADC Interrupt Vector Register, Software Example

The following software example shows the use of vector word ADC12IV and the associated software overhead. The numbers at the right margin show the cycles required for every instruction. The example shows a basic interrupt handler structure that can be adopted to individual application requirements.

The software overhead for the different interrupt sources, including interrupt latency and return-from-interrupt cycles (but not the task handling itself), is:

- ADC12IFG.0 to ADC12IFG.14, ADC12OV 16 cycles
- ADC12IFG.15 14 cycles

```
; Interrupt handler for the 12-bit ADC.
; The flag, which is enabled and has the highest priority,
; determines the interrupt vector word and is reset by
; hardware after accessing (instruction
; ADD  &TADC12IV,PC). Flags ADC12OV, ADC12TOV, and
; ADC12IFG.x are reset by hardware.
ADC_HND   $         ; Interrupt latency                    6
   ADD &ADC12IV,PC  ; Add offset to Jump table             3
   RETI             ; Vector 0: No interrupt               5
   JMP ADC12OV      ; Vector 2: ADC overflow               2
   JMP ADC12TOV     ; Vector 4: ADC timing overflow        2
   JMP ADC12MOD0    ; Vector 6: ADC12MEM0 was loaded
..................  ; (ADC12IFG.0)                         2
   JMP ADC12MOD1    ; Vector 8: ADC12MEM1 was loaded
..................  ; (ADC12IFG.1)                         2
   :
   :
   JMP ADC12MOD13   ; Vector 34: ADC12MEM14 was loaded
                    ; (ADC12IFG.14)                       2
   JMP ADC12MOD14   ; Vector 36: ADC12MEM15 was loaded
                    ; (ADC12IFG.15)                       2
;
; Module 15. Handler for ADC12IFG.15 starts here. Note a JMP
; instruction is not needed to get here because the PC is
; already here after the ADD&ADC12IV,PC instruction.
;
ADC12OV   ...       ; Vector 2: ADC12OV Flag
          ...       ; First instruction to handle ADC12
                    ; overflow condition
          RETI                                          5
;
ADC12TOV  ...       ; Vector 4: ADC12OV Flag
          ...       ; First instruction to handle ADC12 timing
                    ; overflow condition
          RETI                                          5
;
ADC12MOD2           ; Vector 10: ADC12MEM2 was loaded
                    ; (ADC12IFG.2)
          MOV &ADC12MEM2,R6 ; ADC12IFG2 is reset due to access
                            ; of ADC memory
          ...       ; Task starts here
```

```
          RETI      ; Back to main program                5
;
ADC12MOD1           ; Vector 8: ADC12MEM1 was loaded
                    ; (ADC12IFG.1)
          ADD &ADC12MEM1,R6 ; ADC12IFG1 is reset due to access
                            ; of ADC memory
          ...       ; Task starts here
          RETI      ; Back to main program                5
; The Module 3 handler shows a way to look if any other
;interrupt is pending: 5 cycles have to be spent, but 9 cycles
; may be saved if another interrupt is pending
;
ADC12MOD0             ; Vector 6: ADC12MEM0 was loaded
                      ; (ADC12IFG.0)
          ...         ; First instruction to be executed   x
          ...         ; Task starts here
          JMP ADC12_HND ; With this instruction the software
                        ; does not leave the handler; it looks
                        ; for pending ADC12 interrupts      2
;
          .SECT  ”VECTORS”,0FFF4h ; Interrupt Vectors
          .WORD ADC12_HND  ; Vector for 12-bit ADC interrupt
                           ; flags and overflow
```

Note: Basic Clock System

If the CPU clock MCLK is turned off (CPUOff=1), then two or three additional cycles need to be added for synchronous start of the CPU system. The delta of one clock cycle is caused when clocks are asynchronous to the restart of CPU clock MCLK.

15.9 A/D Grounding and Noise Considerations

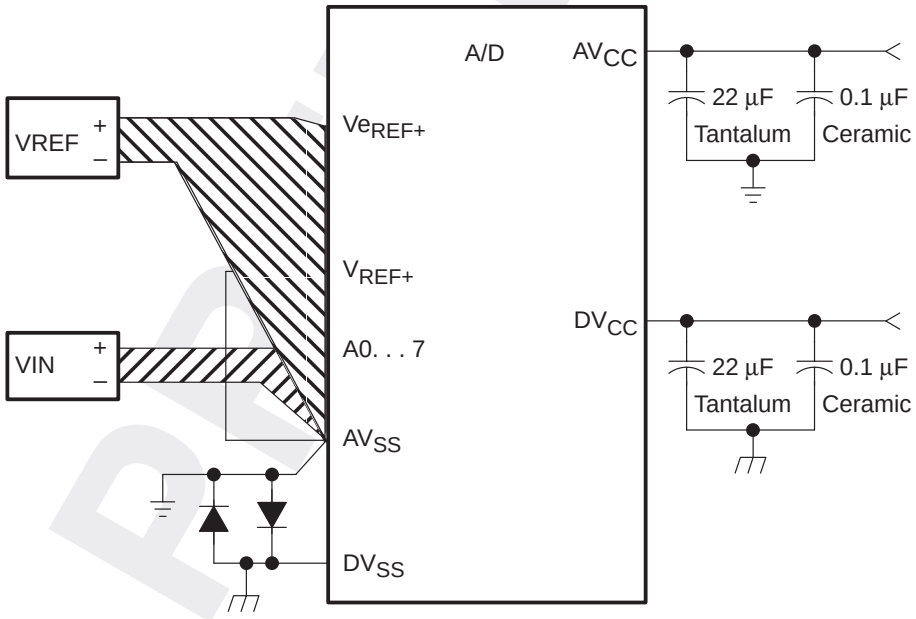
As with any high-resolution converter, care and special attention must be paid to the printed-circuit-board layout and the grounding scheme to eliminate ground loops and any unwanted parasitic components/effects and noise. Industry-standard grounding and layout techniques should be followed to reduce these unwanted effects.

Ground Loops are formed when return current from the A/D flows through paths that are common with other analog or digital circuitry. If care is not taken, this current can generate small, unwanted offset voltages that can add to or subtract from the reference or input voltages of the A/D converter. One way to avoid ground loops is to use a star connection scheme for AV_{SS} (shown in Figure 15–26). This way the ground current or reference currents do not flow through any common input leads, eliminating any error voltages.

The digital ground DV_{SS} and analog ground AV_{SS} can also be star-connected together. If separate supplies are used, two reverse biased diodes can be used to limit the voltage difference to less than ±700mV.

In addition to grounding, ripple and noise spikes on the power supply lines due to digital switching or switching power supplies can corrupt the conversion result. The ripple can become more dominant by reducing the value of the conversion voltage range ($V_{R+} - V_{R-}$), therefore reducing the value of the LSB and the noise margin. Thus a clean, noise-free setup becomes even more important to achieve the desired accuracy. Adding carefully placed bypass capacitors returned to the respective ground planes can help in reducing ripple in the supply current and minimizing these effects.

Figure 15–26. A/D Grounding and Noise Considerations



Appendix A

Peripheral File Map

This appendix summarizes the peripheral file (PF) and control-bit information into a single location for reference.

Each PF register is presented as a row of boxes containing the control or status bits belonging to the register. The register symbol (e.g. P0IN) and the PF hex address are to the left of each register.

Topic **Page**

A.1 Overview

Bit accessibility and/or hardware definitions are indicated following each bit symbol:

- rw: Read/write
- r: Read only
- r0: Read as 0
- r1: Read as 1
- w: Write only
- w0: Write as 0
- w1: Write as 1
- (w): No register bit implemented; writing a 1 results in a pulse. The register bit is always read as 0.
- h0: Cleared by hardware
- h1: Set by hardware
- –0,–1: Condition after PUC signal active
- –(0),–(1): Condition after POR signal active

The tables in the following sections describe byte access to each peripheral file according to the previously-described definitions.

A.2 Special Function Register of MSP430x1xx Family, Byte Access

000Fh								
Module enable 2, ME2 0005h			UTXE1 rw-0	URXE1 USPIE1 rw-0				
Module enable 1, ME1 0004h	UTXE0 rw-0	URXE0 USPIE0 rw-0						
Interrupt flag 2, IFG2 0003h			UTXIFG1 rw-1	URXIFG1 rw-0				
Interrupt flag 1, IFG1 0002h	UXIFG0 rw-1	URXIFG0 rw-0		NMIIFG rw-0			OFIFG rw-1	WDTIFG rw-0
Interrupt enable 2, IE2 0001h			UTXIE1 rw-0	URXIE1 rw-0				
Interrupt enable 1, IE1 0000h	UTXIE0 rw-0	URXIE0 rw-0	ACCVIE rw-0	NMIIE rw-0			OFIE rw-0	WDTIE rw-0

Note: SFR bits are not implemented on devices without the corresponding peripheral.

A.3 Digital I/O, Byte Access

Bit # -	7	6	5	4	3	2	1	0
Function select, P4SEL 001Fh	P4SEL.7 rw-0	P4SEL.6 rw-0	P4SEL.5 rw-0	P4SEL.4 rw-0	P4SEL.3 rw-0	P4SEL.2 rw-0	P4SEL.1 rw-0	P4SEL.0 rw-0
Direction register, P4DIR 001Eh	P4DIR.7 rw-0	P4DIR.6 rw-0	P4DIR.5 rw-0	P4DIR.4 rw-0	P4DIR.3 rw-0	P4DIR.2 rw-0	P4DIR.1 rw-0	P4DIR.0 rw-0
Output register, P4OUT 001Dh	P4OUT.7 rw	P4OUT.6 rw	P4OUT.5 rw	P4OUT.4 rw	P4OUT.3 rw	P4OUT.2 rw	P4OUT.1 rw	P4OUT.0 rw
Input register, P4IN 001Ch	P4IN.7 r	P4IN.6 r	P4IN.5 r	P4IN.4 r	P4IN.3 r	P4IN.2 r	P4IN.1 r	P4IN.0 r
Function select, P3SEL 001Bh	P3SEL.7 rw-0	P3SEL.6 rw-0	P3SEL.5 rw-0	P3SEL.4 rw-0	P3SEL.3 rw-0	P3SEL.2 rw-0	P3SEL.1 rw-0	P3SEL.0 rw-0
Direction register, P3DIR 001Ah	P3DIR.7 rw-0	P3DIR.6 rw-0	P3DIR.5 rw-0	P3DIR.4 rw-0	P3DIR.3 rw-0	P3DIR.2 rw-0	P3DIR.1 rw-0	P3DIR.0 rw-0
Output register, P3OUT 0019h	P3OUT.7 rw	P3OUT.6 rw	P3OUT.5 rw	P3OUT.4 rw	P3OUT.3 rw	P3OUT.2 rw	P3OUT.1 rw	P3OUT.0 rw
Input register, P3IN 0018h	P3IN.7 r	P3IN.6 r	P3IN.5 r	P3IN.4 r	P3IN.3 r	P3IN.2 r	P3IN.1 r	P3IN.0 r
0017h								
0016h								
0010h								

A.3 Digital I/O, Byte Access (Continued)

Bit # -	7	6	5	4	3	2	1	0
Function select, P6SEL 0037h	P6SEL.7 rw-0	P6SEL.6 rw-0	P6SEL.5 rw-0	P6SEL.4 rw-0	P6SEL.3 rw-0	P6SEL.2 rw-0	P6SEL.1 rw-0	P6SEL.0 rw-0
Direction register, P6DIR 0036h	P6DIR.7 rw-0	P6DIR.6 rw-0	P6DIR.5 rw-0	P6DIR.4 rw-0	P6DIR.3 rw-0	P6DIR.2 rw-0	P6DIR.1 rw-0	P6DIR.0 rw-0
Output register, P6OUT 0035h	P6OUT.7 rw	P6OUT.6 rw	P6OUT.5 rw	P6OUT.4 rw	P6OUT.3 rw	P6OUT.2 rw	P6OUT.1 rw	P6OUT.0 rw
Input register, P6IN 0034h	P6IN.7 r	P6IN.6 r	P6IN.5 r	P6IN.4 r	P6IN.3 r	P6IN.2 r	P6IN.1 r	P6IN.0 r
Function select, P5SEL 0033h	P5SEL.7 rw-0	P5SEL.6 rw-0	P6SEL.5 rw-0	P5SEL.4 rw-0	P5SEL.3 rw-0	P5SEL.2 rw-0	P5SEL.1 rw-0	P5SEL.0 rw-0
Direction register, P5DIR 0032h	P5DIR.7 rw-0	P5DIR.6 rw-0	P5DIR.5 rw-0	P5DIR.4 rw-0	P5DIR.3 rw-0	P5DIR.2 rw-0	P5DIR.1 rw-0	P5DIR.0 rw-0
Output register, P5OUT 0031h	P5OUT.7 rw	P5OUT.6 rw	P5OUT.5 rw	P5OUT.4 rw	P5OUT.3 rw	P5OUT.2 rw	P5OUT.1 rw	P5OUT.0 rw
Input register, P5IN 0030h	P5IN.7 r	P5IN.6 r	P5IN.5 r	P5IN.4 r	P5IN.3 r	P5IN.2 r	P5IN.1 r	P5IN.0 r
002Fh								
Function select, P2SEL 002Eh	P2SEL.7 rw-0	P2SEL.6 rw-0	P2SEL.5 rw-0	P2SEL.4 rw-0	P2SEL.3 rw-0	P2SEL.2 rw-0	P2SEL.1 rw-0	P2SEL.0 rw-0
Interrupt enable, P2IE 002Dh	P2IE.7 rw-0	P2IE.6 rw-0	P2IE.5 rw-0	P2IE.4 rw-0	P2IE.3 rw-0	P2IE.2 rw-0	P2IE.1 rw-0	P2IE.0 rw-0
Interrupt edge select, P2IES 002Ch	P2IES.7 rw	P2IES.6 rw	P2IES.5 rw	P2IES.4 rw	P2IES.3 rw	P2IES.2 rw	P2IES.1 rw	P2IES.0 rw
Interrupt flags, P2IFG 002Bh	P2IFG.7 rw-0	P2IFG.6 rw-0	P2IFG.5 rw-0	P2IFG.4 rw-0	P2IFG.3 rw-0	P2IFG.2 rw-0	P2IFG.1 rw-0	P2IFG.0 rw-0
Direction register, P2DIR 002Ah	P2DIR.7 rw-0	P2DIR.6 rw-0	P2DIR.5 rw-0	P2DIR.4 rw-0	P2DIR.3 rw-0	P2DIR.2 rw-0	P2DIR.1 rw-0	P2DIR.0 rw-0
Output register, P2OUT 0029h	P2OUT.7 rw	P2OUT.6 rw	P2OUT.5 rw	P2OUT.4 rw	P2OUT.3 rw	P2OUT.2 rw	P2OUT.1 rw	P2OUT.0 rw
Input register, P2IN 0028h	P2IN.7 r	P2IN.6 r	P2IN.5 r	P2IN.4 r	P2IN.3 r	P2IN.2 r	P2IN.1 r	P2IN.0 r
0027h								
Function select, P1SEL 0026h	P1SEL.7 rw-0	P1SEL.6 rw-0	P1SEL.5 rw-0	P1SEL.4 rw-0	P1SEL.3 rw-0	P1SEL.2 rw-0	P1SEL.1 rw-0	P1SEL.0 rw-0
Interrupt enable, P1IE 0025h	P1IE.7 rw-0	P1IE.6 rw-0	P1IE.5 rw-0	P1IE.4 rw-0	P1IE.3 rw-0	P1IE.2 rw-0	P1IE.1 rw-0	P1IE.0 rw-0
Int. edge select, P1IES 0024h	P1IES.7 rw	P1IES.6 rw	P1IES.5 rw	P1IES.4 rw	P1IES.3 rw	P1IES.2 rw	P1IES.1 rw	P1IES.0 rw
Interrupt flags, P1IFG 0023h	P1IFG.7 rw-0	P1IFG.6 rw-0	P1IFG.5 rw-0	P1IFG.4 rw-0	P1IFG.3 rw-0	P1IFG.2 rw-0	P1IFG.1 rw-0	P1IFG.0 rw-0
Direction register, P1DIR 0022h	P1DIR.7 rw-0	P1DIR.6 rw-0	P1DIR.5 rw-0	P1DIR.4 rw-0	P1DIR.3 rw-0	P1DIR.2 rw-0	P1DIR.1 rw-0	P1DIR.0 rw-0
Output register, P1OUT 0021h	P1OUT.7 rw	P1OUT.6 rw	P1OUT.5 rw	P1OUT.4 rw	P1OUT.3 rw	P1OUT.2 rw	P1OUT.1 rw	P1OUT.0 rw
Input register, P1IN 0020h	P1IN.7 r	P1IN.6 r	P1IN.5 r	P1IN.4 r	P1IN.3 r	P1IN.2 r	P1IN.1 r	P1IN.0 r

A.4 Basic Clock Registers, Byte Access

Bit # –	7	6	5	4	3	2	1	0
BCSCTL2 0058h	SELM.1 rw–0	SELM.0 rw–0	DIVM.1 rw–0	DIVM.0 rw–0	SELS rw–0	DIVS.1 rw–0	DIVS.0 rw–0	DCOR rw–0
BCSCTL1 0057h	XT2OFF rw–(1)	XTS rw–(0)	DIVA.1 rw–(0)	DIVA.0 rw–(0)	XT5V rw–0	Rsel.2 rw–1	Rsel.1 rw–0	Rsel.0 rw–0
DCOCTL 0056h	DCO.2 rw–0	DCO.1 rw–1	DCO.0 rw–1	MOD.4 rw–0	MOD.3 rw–0	MOD.2 rw–0	MOD.1 rw–0	MOD.0 rw–0

A.5 EPROM Control Register Byte Access

Bit # –	7	6	5	4	3	2	1	0
EPROM control register† EPCTL 0054h	r-0	r-0	r-0	r-0	r-0	r-0	VPPS rw-0	EXE rw-0

† Non-EPROM devices may use this register for other control purposes.

A.6 Comparator_A Registers, Byte Access

Bit # –	7	6	5	4	3	2	1	0
Comparator_A Port Disable, CAPD 005Bh	CAPD.7 rw–(0)	CAPD.6 rw–(0)	CAPD.5 rw–(0)	CAPD.4 rw–(0)	CAPD.3 rw–(0)	CAPD.2 rw–(0)	CAPD.1 rw–(0)	CAPD.0 rw–(0)
Comparator_A control reg. 2, CACTL2 005Ah	CACTL2.7 rw–(0)	CACTL2.6 rw–(0)	CACTL2.5 rw–(0)	CACTL2.4 rw–(0)	CA1 rw–(0)	CA0 rw–(0)	CAF rw–(0)	CAOUT r–(0)
Comparator_A control reg. 1, CACTL1 0059h	CAEX rw–(0)	CARSEL rw–(0)	CAREF1 rw–(0)	CAREF0 rw–(0)	CAON rw–(0)	CAIES rw–(0)	CAIE rw–(0)	CAIFG rw–(0)

A.7 USART0, USART1, UART Mode (Sync=0), Byte Access

Bit # –	7	6	5	4	3	2	1	0
USART1 Transmit buffer UTXBUF1 07Fh	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
USART1 Receive buffer URXBUF1 07Eh	2^7 r	2^6 r	2^5 r	2^4 r	2^3 r	2^2 r	2^1 r	2^0 r
USART1 Baud rate UBR11 07Dh	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^9 rw	2^8 rw
USART1 Baud rate UBR01 07Ch	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
USART1 Modulation control UMCTL1 07Bh	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
USART1 Receive control URCTL1 07Ah	FE rw-0	PE rw-0	OE rw-0	BRK rw-0	URXEIE rw-0	URXWIE rw-0	RXWake rw-0	RXERR rw-0
USART1 Transmit control UTCTL1 079h	Unused rw-0	CKPL rw-0	SSEL1 rw-0	SSEL0 rw-0	URXSE rw-0	TXWAKE rw-0	Unused rw-0	TXEPT rw-1
USART1 USART control UCTL1 078h	PENA rw-0	PEV rw-0	SP rw-0	CHAR rw-0	Listen rw-0	SYNC rw-0	MM rw-0	SWRST rw-1
USART0 Transmit buffer UTXBUF0 077h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
USART0 Receive buffer URXBUF0 076h	2^7 r	2^6 r	2^5 r	2^4 r	2^3 r	2^2 r	2^1 r	2^0 r
USART0 Baud rate UBR10 075h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^9 rw	2^8 rw
USART0 Baud rate UBR00 074h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
USART0 Modulation control UMCTL0 073h	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
USART0 Receive control URCTL0 072h	FE rw-0	PE rw-0	OE rw-0	BRK rw-0	URXEIE rw-0	URXWIE rw-0	RXWake rw-0	RXERR rw-0
USART0 Transmit control UTCTL0 071h	Unused rw-0	CKPL rw-0	SSEL1 rw-0	SSEL0 rw-0	URXSE rw-0	TXWAKE rw-0	Unused rw-0	TXEPT rw-1
USART0 USART control UCTL0 070h	PENA rw-0	PEV rw-0	SP rw-0	CHAR rw-0	Listen rw-0	SYNC rw-0	MM rw-0	SWRST rw-1

A.8 USART0, USART1, SPI Mode (Sync=1), Byte Access

Bit # –	7	6	5	4	3	2	1	0
USART1 Transmit buffer UTXBUF1 07Fh	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
USART1 Receive buffer URXBUF1 07Eh	2^7 r	2^6 r	2^5 r	2^4 r	2^3 r	2^2 r	2^1 r	2^0 r
USART1 Baud rate UBR11 07Dh	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^9 rw	2^8 rw
USART1 Baud rate UBR01 07Ch	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
USART1 Modulation control UMCTL1 07Bh	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
USART1 Receive control URCTL1 07Ah	FE rw-0	Undef. rw-0	OE rw-0	Undef. rw-0	Unused rw-0	Unused rw-0	Undef. rw-0	Undef. rw-0
USART1 Transmit control UTCTL1 079h	CKPH rw-0	CKPL rw-0	SSEL1 rw-0	SSEL0 rw-0	Unused rw-0	Unused rw-0	STC rw-0	TXEPT rw-1
USART1 USART control UCTL1 078h	Unused rw-0	Unused rw-0	Unused rw-0	CHAR rw-0	Listen rw-0	SYNC rw-0	MM rw-0	SWRST rw-1
USART0 Transmit buffer UTXBUF0 077h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
USART0 Receive buffer URXBUF0 076h	2^7 r	2^6 r	2^5 r	2^4 r	2^3 r	2^2 r	2^1 r	2^0 r
USART0 Baud rate UBR10 075h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^9 rw	2^8 rw
USART0 Baud rate UBR00 074h	2^7 rw	2^6 rw	2^5 rw	2^4 rw	2^3 rw	2^2 rw	2^1 rw	2^0 rw
USART0 Modulation control UMCTL0 073h	m7 rw	m6 rw	m5 rw	m4 rw	m3 rw	m2 rw	m1 rw	m0 rw
USART0 Receive control URCTL0 072h	FE rw-0	Undef. rw-0	OE rw-0	Undef. rw-0	Unused rw-0	Unused rw-0	Undef. rw-0	Undef. rw-0
USART0 Transmit control UTCTL0 071h	CKPH rw-0	CKPL rw-0	SSEL1 rw-0	SSEL0 rw-0	Unused rw-0	Unused rw-0	STC rw-0	TXEPT rw-1
USART0 USART control UCTL0 070h	Unused rw-0	Unused rw-0	Unused rw-0	CHAR rw-0	Listen rw-0	SYNC rw-0	MM rw-0	SWRST rw-1

A.9 ADC12 Registers, Byte and Word Access

Bit # –	7	6	5	4	3	2	1	0
ADC12MCTL15† 008Fh	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)
ADC12MCTL14† 008Eh	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)
ADC12MCTL13† 008Dh	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)
ADC12MCTL12† 008Ch	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)
ADC12MCTL11† 008Bh	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)
ADC12MCTL10† 008Ah	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)
ADC12MCTL9† 0089h	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)
ADC12MCTL8† 0088h	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)
ADC12MCTL7† 0087h	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)
ADC12MCTL6† 0086h	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)
ADC12MCTL5† 0085h	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)
ADC12MCTL4† 0084h	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)
ADC12MCTL3† 0083h	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)
ADC12MCTL2† 0082h	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)
ADC12MCTL1† 0081h	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)
ADC12MCTL0† 0080h	EOS rw–(0)	Sref.2 rw–(0)	Sref.1 rw–(0)	Sref.0 rw–(0)	INCH.3 rw–(0)	INCH.2 rw–(0)	INCH.1 rw–(0)	INCH.0 rw–(0)

† All bits of ADC12MCTLx registers are only modifiable when ENC=0.

A.9 ADC12 Registers, Byte and Word Access (Continued)

Bit # –	15	14	13	12	11	0
ADC12MEM15 015Eh	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	
ADC12MEM14 015Ch	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	
ADC12MEM13 015Ah	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	
ADC12MEM12 0158h	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	
ADC12MEM11 0156h	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	
ADC12MEM10 0154h	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	
ADC12MEM9 0152h	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	
ADC12MEM8 0150h	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	
ADC12MEM7 014Eh	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	
ADC12MEM6 014Ch	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	
ADC12MEM5 014Ah	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	
ADC12MEM4 0148h	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	
ADC12MEM3 0146h	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	
ADC12MEM2 0144h	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	
ADC12MEM1 0142h	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	
ADC12MEM0 0140h	Unused r0	Unused r0	Unused r0	Unused r0	MSB <———— Conversion Result ————> LSB All conversion-result bits of type rw	

A.9 ADC12 Registers, Byte and Word Access (Continued)

Bit # –	15	14	13	12	11	10	9	8
ADC12IE 01A6h	ADC12IE.15 rw–(0)	ADC12IE.14 rw–(0)	ADC12IE.13 rw–(0)	ADC12IE.12 rw–(0)	ADC12IE.11 rw–(0)	ADC12IE.10 rw–(0)	ADC12IE.9 rw–(0)	ADC12IE.8 rw–(0)
ADC12IFG 01A4h	ADC12IFG.15 rw–(0)	ADC12IFG.14 rw–(0)	ADC12IFG.13 rw–(0)	ADC12IFG.12 rw–(0)	ADC12IFG.11 rw–(0)	ADC12IFG.10 rw–(0)	ADC12IFG.9 rw–(0)	ADC12IFG.8 rw–(0)
ADC12CTL1 01A2h	CStartAdd.3† rw–(0)	CStartAdd.2† rw–(0)	CStartadd.1† rw–(0)	CStartAdd.0† rw–(0)	SHS.1† rw–(0)	SHS.0† rw–(0)	SHP† rw–(0)	ISSH† rw–(0)
ADC12CTL0 01A0h	SHT1.3† rw–(0)	SHT1.2† rw–(0)	SHT1.1† rw–(0)	SHT1.0† rw–(0)	SHT0.3† rw–(0)	SHT0.2† rw–(0)	SHT0.1† rw–(0)	SHT0.0† rw–(0)

† Only modifiable when ENC=0.

Bit # –	7	6	5	4	3	2	1	0
ADC12IE 01A6h	ADC12IE.7 rw–(0)	ADC12IE.6 rw–(0)	ADC12IE.5 rw–(0)	ADC12IE.4 rw–(0)	ADC12IE.3 rw–(0)	ADC12IE.2 rw–(0)	ADC12IE.1 rw–(0)	ADC12IE.0 rw–(0)
ADC12IFG 01A4h	ADC12IFG.7 rw–(0)	ADC12IFG.6 rw–(0)	ADC12IFG.5 rw–(0)	ADC12IFG.4 rw–(0)	ADC12IFG.3 rw–(0)	ADC12IFG.2 rw–(0)	ADC12IFG.1 rw–(0)	ADC12IFG.0 rw–(0)
ADC12CTL1 01A2h	ADC12DIV.2† rw–(0)	ADC12DIV.1† rw–(0)	ADC12DIV.0† rw–(0)	ADC12SSEL.1† rw–(0)	ADC12SSEL.0† rw–(0)	CONSEQ.1 rw–(0)	CONSEQ.0 rw–(0)	ADC12BUSY r–(0)
ADC12CTL0 01A0h	MSC† rw–(0)	2_5V† rw–(0)	REFON† rw–(0)	ADC12ON† rw–(0)	ADC12OVIE rw–(0)	ADC12TOVIE rw–(0)	ENC rw–(0)	ADC12SC rw–(0)

† Only modifiable when ENC=0.

A.10 Watchdog/Timer, Word Access

Bit # –	15							8
Watchdog Timer, Control register WDTCTL 120h	<———			Read as 069h Written as 05Ah				———> ———>

Bit # –	7	6	5	4	3	2	1	0
Watchdog Timer, Control register WDTCTL 120h	HOLD rw-0	NMIES rw-0	NMI rw-0	TMSEL rw-0	CNTCL (w),r0	SSEL rw-0	IS1 rw-0	IS0 rw-0

A.11 Flash Control Registers, Word Access

Bit # –	15	14	13	12	11	10	9	8
FCTL3 012Ch	<——— <———			Read as 096h Written as 0A5h				———> ———>
FCTL2 012Ah	<——— <———			Read as 096h Written as 0A5h				———> ———>
FCTL1 0128H	<——— <———			Read as 096h Written as 0A5h				———> ———>

Bit # –	7	6	5	4	3	2	1	0
FCTL3 012Ch	Reserved r0	Reserved r0	EMEX rw–0	Lock rw–1	WAIT r–1	ACCVIFG rw–0	KEYV rw–(0)	Busy r(w)–0
FCTL2 012Ah	SSEL1 rw–0	SSEL0 rw–1	FN5 rw–0	FN4 rw–0	FN3 rw–0	FN2 rw–0	FN1 rw–1	FN0 rw–0
FCTL1 0128H	SEGWRT rw–0	WRT rw–0	Reserved r0	Reserved r0	Reserved r0	MEras rw–0	Erase rw–0	Reserved r0

A.12 Hardware Multiplier, Word Access

Bit # –	15	14	13	12	11	10	9	8
Sum extend, SumExt 013Eh	† r	† r	† r	† r	† r	† r	† r	† r
Result-high word ResHI 013Ch	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
Result-low word ResLO 013Ah	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
Second operand OP2 0138h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
MPYS+ACC MACS 0136h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
MPY+ACC MAC 0134h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
Multiply signed MPYS 0132h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw
Multiply unsigned MPY 0130h	2^{15} rw	2^{14} rw	2^{13} rw	2^{12} rw	2^{11} rw	2^{10} rw	2^{9} rw	2^{8} rw

Bit # –	7	6	5	4	3	2	1	0
Sum extend, SumExt 013Eh	† r	† r	† r	† r	† r	† r	† r	† r
Result-high word ResHI 013Ch	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
Result-low word ResLO 013Ah	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
Second operand OP2 0138h	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
MPYS+ACC MACS 0136h	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
MPY+ACC MAC 0134h	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
Multiply signed MPYS 0132h	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw
Multiply unsigned MPY 0130h	2^{7} rw	2^{6} rw	2^{5} rw	2^{4} rw	2^{3} rw	2^{2} rw	2^{1} rw	2^{0} rw

† The Sum Extend register SumExt holds a 16×16-bit multiplication (MPYS) sign result, or the overflow of the multiply and accumulate (MAC) operation, or the sign of the signed multiply and accumulate (MACS) operation. Overflow and underflow of the MACS operation must be handled by software.

A.13 Timer_A Registers, Word Access

Bit # –	15	14	13	12	11	10	9	8
017Eh								
017Ch								
Cap/com register CCR4† 017Ah	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/com register CCR3† 0178h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/com register CCR2 0176h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/com register CCR1 0174h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/com register CCR0 0172h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Timer_A register TAR 0170h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
016Eh								
016Ch								
Cap/com control CCTL4†, 016Ah	CM41 rw-(0)	CM40 rw-(0)	CCIS41 rw-(0)	CCIS40 rw-(0)	SCS4 rw-(0)	SCCI4 rw-(0)	Unused r0	CAP4 rw-(0)
Cap/com control CCTL3†, 0168h	CM31 rw-(0)	CM30 rw-(0)	CCIS31 rw-(0)	CCIS30 rw-(0)	SCS3 rw-(0)	SCCI3 rw-(0)	Unused r0	CAP3 rw-(0)
Cap/com control CCTL2, 0166h	CM21 rw-(0)	CM20 rw-(0)	CCIS21 rw-(0)	CCIS20 rw-(0)	SCS2 rw-(0)	SCCI2 rw-(0)	Unused r0	CAP2 rw-(0)
Cap/com control CCTL1, 0164h	CM11 rw-(0)	CM10 rw-(0)	CCIS11 rw-(0)	CCIS10 rw-(0)	SCS1 rw-(0)	SCCI1 rw-(0)	Unused r0	CAP1 rw-(0)
Cap/com control CCTL0, 0162h	CM01 rw-(0)	CM00 rw-(0)	CCIS01 rw-(0)	CCIS00 rw-(0)	SCS0 rw-(0)	SCCI0 rw-(0)	Unused r0	CAP0 rw-(0)
Timer_A control TACTL 0160h	Unused rw-(0)	Unused rw-(0)	Unused rw-(0)	Unused rw-(0)	Unused rw-(0)	SSEL2 rw-(0)	SSEL1 rw-(0)	SSEL0 rw-(0)

† Registers are reserved on devices with Timer_A3.

A.13 Timer_A Registers, Word Access (Continued)

Bit # –	7	6	5	4	3	2	1	0
017Eh								
017Ch								
Cap/com register CCR4† 017Ah	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/com register CCR3† 0178h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/com register CCR2 0176h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/com register CCR1 0174h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/com register CCR0 0172h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Timer_A register TAR 0170h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
016Eh								
016Ch								
Cap/com control CCTL4†, 016Ah	OutMod42 rw-(0)	OutMod41 rw-(0)	OutMod40 rw-(0)	CCIE4 rw-(0)	CCI4 r	OUT4 rw-(0)	COV4 rw-(0)	CCIFG4 rw-(0)
Cap/com control CCTL3†, 0168h	OutMod32 rw-(0)	OutMod31 rw-(0)	OutMod30 rw-(0)	CCIE3 rw-(0)	CCI3 r	OUT3 rw-(0)	COV3 rw-(0)	CCIFG3 rw-(0)
Cap/com control CCTL2, 0166h	OutMod22 rw-(0)	OutMod21 rw-(0)	OutMod20 rw-(0)	CCIE2 rw-(0)	CCI2 r	OUT2 rw-(0)	COV2 rw-(0)	CCIFG2 rw-(0)
Cap/com control CCTL1, 0164h	OutMod12 rw-(0)	OutMod11 rw-(0)	OutMod10 rw-(0)	CCIE1 rw-(0)	CCI1 r	OUT1 rw-(0)	COV1 rw-(0)	CCIFG1 rw-(0)
Cap/com control CCTL0, 0162h	OutMod02 rw-(0)	OutMod01 rw-(0)	OutMod00 rw-(0)	CCIE0 rw-(0)	CCI0 r	OUT0 rw-(0)	COV0 rw-(0)	CCIFG0 rw-(0)
Timer_A control TACTL 0160h	ID1 rw-(0)	ID0 rw-(0)	MC1 rw-(0)	MC0 rw-(0)	Unused rw-(0)	CLR rw-(0)	TAIE rw-(0)	TAIFG rw-(0)

† Registers are reserved on devices with Timer_A3.

Bit # –	15	14	13	12	11	10	9	8
Timer_A interrupt vector TAIV 12Eh	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0

Bit # –	7	6	5	4	3	2	1	0
Timer_A interrupt vector TAIV 12Eh	0 r0	0 r0	0 r0	0 r0	TAIV r-(0)	r-(0)	r-(0)	0 r0

TAIV Vector, Timer_A5 (five capture/compare blocks integrated)
- 0: No interrupt pending
- 2: CCIFG1 flag set, interrupt flag of capture/compare block 1
- 4: CCIFG2 flag set, interrupt flag of capture/compare block 2 (CCIFG1=0)
- 6: CCIFG3 flag set, interrupt flag of capture/compare block 3 (CCIFG1=CCIFG2=0)
- 8: CCIFG3 flag set, interrupt flag of capture/compare block 3 (CCIFG1=CCIFG2=CCIFG3=0)
- 10: TAIFG flag set, interrupt flag of Timer_A register/counter (CCIFG1=CCIFG2=CCIFG3=CCIFG4=0)

TAIV Vector, Timer_A3 (three capture/compare blocks integrated)
- 0: No interrupt pending
- 2: CCIFG1 flag set, interrupt flag of capture/compare block 1
- 4: CCIFG2 flag set, interrupt flag of capture/compare block 2 (CCIFG1=0)
- 6: Reserved
- 8: Reserved
- 10: TAIFG flag set, interrupt flag of Timer_A register/counter (CCIFG1=CCIFG2=CCIFG3=CCIFG4=0)

A.14 Timer_B Registers, Word Access

Bit # –	15	14	13	12	11	10	9	8
Cap/com register CCR6† 019Eh	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/com register CCR5† 019Ch	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/com register CCR4† 019Ah	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/com register CCR3† 0198h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/com register CCR2 0196h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/com register CCR1 0194h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/com register CCR0 0192h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Timer_B register TBR 0190h	2^{15} rw-(0)	2^{14} rw-(0)	2^{13} rw-(0)	2^{12} rw-(0)	2^{11} rw-(0)	2^{10} rw-(0)	2^{9} rw-(0)	2^{8} rw-(0)
Cap/com control CCTL6†, 018Eh	CM61 rw-(0)	CM60 rw-(0)	CCIS61 rw-(0)	CCIS60 rw-(0)	SCS6 rw-(0)	CLLD6.1 rw-(0)	CLLD6.0 rw–(0)	CAP6 rw-(0)
Cap/com control CCTL5†, 018Ch	CM51 rw-(0)	CM50 rw-(0)	CCIS51 rw-(0)	CCIS50 rw-(0)	SCS5 rw-(0)	CLLD5.1 rw-(0)	CLLD5.0 rw–(0)	CAP5 rw-(0)
Cap/com control CCTL4†, 018Ah	CM41 rw-(0)	CM40 rw-(0)	CCIS41 rw-(0)	CCIS40 rw-(0)	SCS4 rw-(0)	CCLD4.1 rw-(0)	CCLD4.0 r0	CAP4 rw-(0)
Cap/com control CCTL3†, 0188h	CM31 rw-(0)	CM30 rw-(0)	CCIS31 rw-(0)	CCIS30 rw-(0)	SCS3 rw-(0)	CCLD3.1 rw-(0)	CCLD3.0 r0	CAP3 rw-(0)
Cap/com control CCTL2, 0186h	CM21 rw-(0)	CM20 rw-(0)	CCIS21 rw-(0)	CCIS20 rw-(0)	SCS2 rw-(0)	CCLD2.1 rw-(0)	CCLD2.0 r0	CAP2 rw-(0)
Cap/com control CCTL1, 0184h	CM11 rw-(0)	CM10 rw-(0)	CCIS11 rw-(0)	CCIS10 rw-(0)	SCS1 rw-(0)	CCLD1.1 rw-(0)	CCLD1.0 r0	CAP1 rw-(0)
Cap/com control CCTL0, 0182h	CM01 rw-(0)	CM00 rw-(0)	CCIS01 rw-(0)	CCIS00 rw-(0)	SCS0 rw-(0)	CCLD0.1 rw-(0)	CCLD0.0 r0	CAP0 rw-(0)
Timer_B control TBCTL, 0180h	Unused rw-(0)	TBCLGRP1 rw-(0)	TBCLGRP0 rw-(0)	TBCNTL1 rw-(0)	TBCNTL0 rw-(0)	Unused rw-(0)	TBSSEL1 rw-(0)	TBSSEL0 rw-(0)

† Registers are reserved on devices with Timer_B3.

A.14 Timer_B Registers, Word Access (Continued)

Bit # –	7	6	5	4	3	2	1	0
Cap/com register CCR6† 019Eh	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/com register CCR5† 019Ch	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/com register CCR4† 019Ah	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/com register CCR3† 0198h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/com register CCR2 0196h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/com register CCR1 0194h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/com register CCR0 0192h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Timer_B register TBR 0190h	2^7 rw-(0)	2^6 rw-(0)	2^5 rw-(0)	2^4 rw-(0)	2^3 rw-(0)	2^2 rw-(0)	2^1 rw-(0)	2^0 rw-(0)
Cap/com control CCTL6†, 018Eh	OutMod62 rw-(0)	OutMod61 rw-(0)	OutMod60 rw-(0)	CCIE6 rw-(0)	CCI6 r	OUT6 rw-(0)	COV6 rw-(0)	CCIFG6 rw-(0)
Cap/com control CCTL5†, 018Ch	OutMod52 rw-(0)	OutMod51 rw-(0)	OutMod50 rw-(0)	CCIE5 rw-(0)	CCI5 r	OUT5 rw-(0)	COV5 rw-(0)	CCIFG5 rw-(0)
Cap/com control CCTL4†, 018Ah	OutMod42 rw-(0)	OutMod41 rw-(0)	OutMod40 rw-(0)	CCIE4 rw-(0)	CCI4 r	OUT4 rw-(0)	COV4 rw-(0)	CCIFG4 rw-(0)
Cap/com control CCTL3†, 0188h	OutMod32 rw-(0)	OutMod31 rw-(0)	OutMod30 rw-(0)	CCIE3 rw-(0)	CCI3 r	OUT3 rw-(0)	COV3 rw-(0)	CCIFG3 rw-(0)
Cap/com control CCTL2, 0186h	OutMod22 rw-(0)	OutMod21 rw-(0)	OutMod20 rw-(0)	CCIE2 rw-(0)	CCI2 r	OUT2 rw-(0)	COV2 rw-(0)	CCIFG2 rw-(0)
Cap/com control CCTL1, 0184h	OutMod12 rw-(0)	OutMod11 rw-(0)	OutMod10 rw-(0)	CCIE1 rw-(0)	CCI1 r	OUT1 rw-(0)	COV1 rw-(0)	CCIFG1 rw-(0)
Cap/com control CCTL0, 0182h	OutMod02 rw-(0)	OutMod01 rw-(0)	OutMod00 rw-(0)	CCIE0 rw-(0)	CCI0 r	OUT0 rw-(0)	COV0 rw-(0)	CCIFG0 rw-(0)
Timer_B control TACTL 0180h	TBID1 rw-(0)	TBID0 rw-(0)	TBMC1 rw-(0)	TBMC0 rw-(0)	Unused rw-(0)	TBCLR rw-(0)	TBIE rw-(0)	TBIFG rw-(0)

† Registers are reserved on devices with Timer_B3.

A.14 Timer_B Registers, Word Access (Continued)

Bit # –	15	14	13	12	11	10	9	8
Timer_B interrupt vector TBIV 11Eh	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0	0 r0

Bit # –	7	6	5	4	3	2	1	0
Timer_B interrupt vector TBIV 11Eh	0 r0	0 r0	0 r0	0 r0	TBIV r-(0)	r-(0)	r-(0)	0 r0

TBIV Vector, Timer_B5 (five capture/compare blocks integrated)

0: No interrupt pending
2: CCIFG1 flag set, interrupt flag of capture/compare block 1
4: CCIFG2 flag set, interrupt flag of capture/compare block 2 (CCIFG1=0)
6: CCIFG3 flag set, interrupt flag of capture/compare block 3 (CCIFG1=CCIFG2=0)
8: CCIFG4 flag set, interrupt flag of capture/compare block 4 (CCIFG1=CCIFG2=CCIFG3=0)
10: CCIFG5 flag set, interrupt flag of capture/compare block 5 (CCIFG1=CCIFG2=CCIFG3=CCIFG4=0)
12: CCIFG6 flag set, interrupt flag of capture/compare block 6 (CCIFG1=CCIFG2=CCIFG3=CCIFG4=CCIFG5=0)
14: TBIFG flag set, interrupt flag of Timer_B register/counter (CCIFG1=CCIFG2=CCIFG3=CCIFG4=CCIFG5=CCIFG6=0)

TBIV Vector, Timer_B3 (three capture/compare blocks integrated)

0: No interrupt pending
2: CCIFG1 flag set, interrupt flag of capture/compare block 1
4: CCIFG2 flag set, interrupt flag of capture/compare block 2 (CCIFG1=0)
6: Reserved
8: Reserved
10: Reserved
12: Reserved
14: TBIFG flag set, interrupt flag of Timer_B register/counter (CCIFG1=CCIFG2=0)

Appendix B

Instruction Set Description

The MSP430 core CPU architecture evolved from a reduced instruction set with highly-transparent instruction formats. Using these formats, core instructions are implemented into the hardware. Emulated instructions are also supported by the assembler. Emulated instructions use the core instructions with the built-in constant generators CG1 and CG2 and/or the program counter (PC). The core and emulated instructions are described in detail in this section. The emulated instruction mnemonics are listed with examples.

Program memory words used by an instruction vary from one to three words, depending on the combination of addressing modes.

Topic **Page**

B.1 Instruction Set Overview

The following list gives an overview of the instruction set.

				Status Bits			
				V	N	Z	C
*	ADC[.W];ADC.B	dst	dst + C –> dst	*	*	*	*
	ADD[.W];ADD.B	src,dst	src + dst –> dst	*	*	*	*
	ADDC[.W];ADDC.B	src,dst	src + dst + C –> dst	*	*	*	*
	AND[.W];AND.B	src,dst	src .and. dst –> dst	0	*	*	*
	BIC[.W];BIC.B	src,dst	.not.src .and. dst –> dst	–	–	–	–
	BIS[.W];BIS.B	src,dst	src .or. dst –> dst	–	–	–	–
	BIT[.W];BIT.B	src,dst	src .and. dst	0	*	*	*
*	BR	dst	Branch to	–	–	–	–
	CALL	dst	PC+2 –> stack, dst –> PC	–	–	–	–
*	CLR[.W];CLR.B	dst	Clear destination	–	–	–	–
*	CLRC	Clear carry bit		–	–	–	0
*	CLRN	Clear negative bit		–	0	–	–
*	CLRZ	Clear zero bit		–	–	0	–
	CMP[.W];CMP.B	src,dst	dst – src	*	*	*	*
*	DADC[.W];DADC.B	dst	dst + C –> dst (decimal)	*	*	*	*
	DADD[.W];DADD.B	src,dst	src + dst + C –> dst (decimal)	*	*	*	*
*	DEC[.W];DEC.B	dst	dst – 1 –> dst	*	*	*	*
*	DECD[.W];DECD.B	dst	dst – 2 –> dst	*	*	*	*
*	DINT	Disable interrupt		–	–	–	–
*	EINT	Enable interrupt		–	–	–	–
*	INC[.W];INC.B	dst	Increment destination, dst +1 –> dst	*	*	*	*
*	INCD[.W];INCD.B	dst	Double-Increment destination, dst+2–>dst	*	*	*	*
*	INV[.W];INV.B	dst	Invert destination	*	*	*	*
	JC/JHS	Label	Jump to Label if Carry-bit is set	–	–	–	–
	JEQ/JZ	Label	Jump to Label if Zero-bit is set	–	–	–	–
	JGE	Label	Jump to Label if (N .XOR. V) = 0	–	–	–	–
	JL	Label	Jump to Label if (N .XOR. V) = 1	–	–	–	–
	JMP	Label	Jump to Label unconditionally	–	–	–	–
	JN	Label	Jump to Label if Negative-bit is set	–	–	–	–
	JNC/JLO	Label	Jump to Label if Carry-bit is reset	–	–	–	–
	JNE/JNZ	Label	Jump to Label if Zero-bit is reset	–	–	–	–

				Status Bits			
				V	N	Z	C
	MOV[.W];MOV.B	src,dst	src –> dst	–	–	–	–
*	NOP		No operation	–	–	–	–
*	POP[.W];POP.B	dst	Item from stack, SP+2 → SP	–	–	–	–
	PUSH[.W];PUSH.B	src	SP – 2 → SP, src → @SP	–	–	–	–
	RETI		Return from interrupt TOS → SR, SP + 2 → SP TOS → PC, SP + 2 → SZP	*	*	*	*
*	RET		Return from subroutine TOS → PC, SP + 2 → SP	–	–	–	–
*	RLA[.W];RLA.B	dst	Rotate left arithmetically	*	*	*	*
*	RLC[.W];RLC.B	dst	Rotate left through carry	*	*	*	*
	RRA[.W];RRA.B	dst	MSB → MSB → ....LSB → C	0	*	*	*
	RRC[.W];RRC.B	dst	C → MSB → .....LSB → C	*	*	*	*
*	SBC[.W];SBC.B	dst	Subtract carry from destination	*	*	*	*
*	SETC		Set carry bit	–	–	–	1
*	SETN		Set negative bit	–	1	–	–
*	SETZ		Set zero bit	–	–	1	–
	SUB[.W];SUB.B	src,dst	dst + .not.src + 1 → dst	*	*	*	*
	SUBC[.W];SUBC.B	src,dst	dst + .not.src + C → dst	*	*	*	*
	SWPB	dst	swap bytes	–	–	–	–
	SXT	dst	Bit7 → Bit8 ..... Bit15	0	*	*	*
*	TST[.W];TST.B	dst	Test destination	0	*	*	1
	XOR[.W];XOR.B	src,dst	src .xor. dst → dst	*	*	*	*

Note: Asterisked Instructions

Asterisked (*) instructions are emulated. They are replaced with core instructions by the assembler.

B.1.1 Instruction Formats

The following sections describe the instruction formats.

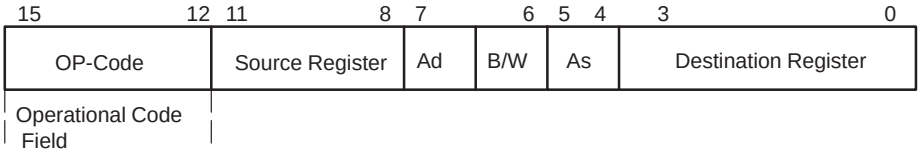
B.1.1.1 Double-Operand Instructions (Core Instructions)

The instruction format using double operands, as shown in Figure B–1, consists of four main fields to form a 16-bit code:

- operational code field, four bits [op-code]
- source field, six bits [source register + As]
- byte operation identifier, one bit [BW]
- destination field, five bits [dest. register + Ad]

The source field is composed of two addressing bits and a four-bit register number (0....15). The destination field is composed of one addressing bit and a four-bit register number (0....15). The byte identifier B/W indicates whether the instruction is executed as a byte (B/W = 1) or as a word instruction (B/W = 0).

Figure B–1. Double-Operand Instructions



				Status Bits			
				V	N	Z	C
ADD[.W];	ADD.B	src,dst	src + dst –> dst	*	*	*	*
ADDC[.W];	ADDC.B	src,dst	src + dst + C –> dst	*	*	*	*
AND[.W];	AND.B	src,dst	src .and. dst –> dst	0	*	*	*
BIC[.W];	BIC.B	src,dst	.not.src .and. dst –> dst	–	–	–	–
BIS[.W];	BIS.B	src,dst	src .or. dst –> dst	–	–	–	–
BIT[.W];	BIT.B	src,dst	src .and. dst	0	*	*	*
CMP[.W];	CMP.B	src,dst	dst – src	*	*	*	*
DADD[.W];	DADD.B	src,dst	src + dst + C –> dst (dec)	*	*	*	*
MOV[.W];	MOV.B	src,dst	src –> dst	–	–	–	–
SUB[.W];	SUB.B	src,dst	dst + .not.src + 1 –> dst	*	*	*	*
SUBC[.W];	SUBC.B	src,dst	dst + .not.src + C –> dst	*	*	*	*
XOR[.W];	XOR.B	src,dst	src .xor. dst –> dst	*	*	*	*

Note: Operations Using the Status Register (SR) for Destination

All operations using status register SR for destination overwrite the SR contents with the operation result; as described in that operation, the status bits are not affected.

Example: ADD #3,SR ; Operation: (SR) + 3 —> SR

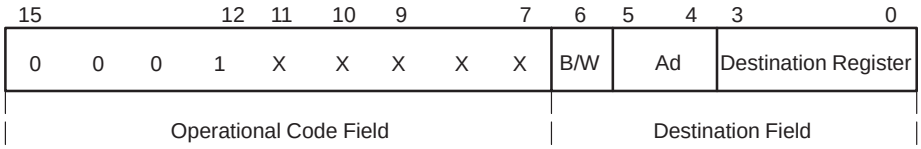
B.1.1.2 Single Operand Instructions (Core Instructions)

The instruction format using a single operand, as shown in Figure B–2, consists of two main fields to form a 16-bit code:

- operational code field, nine bits with four MSBs equal to 1h
- byte operation identifier, one bit [B/W]
- destination field, six bits [destination register + Ad]

The destination field is composed of two addressing bits and the four-bit register number (0....15). The destination field bit position is the same as that of the two operand instructions. The byte identifier (B/W) indicates whether the instruction is executed as a byte (B/W = 1) or as a word (B/W = 0).

Figure B–2. Single-Operand Instructions



				Status Bits			
				V	N	Z	C
RRA[.W];	RRA.B	dst	MSB → MSB ...LSB → C	0	*	*	*
RRC[.W];	RRC.B	dst	C → MSB .....LSB → C	*	*	*	*
PUSH[.W];	PUSH.B	dst	SP – 2 → SP, src → @SP	–	–	–	–
SWPB		dst	swap bytes	–	–	–	–
CALL		dst	PC→2 + @SP, dst → PC	–	–	–	–
RETI		dst	TOS → SR, SP + 2 → SP TOS → PC, SP + 2 → SP	*	*	*	*
SXT		dst	Bit 7 → Bit 8 ..... Bit 15	0	*	*	*

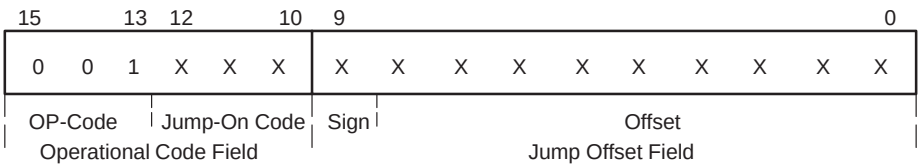
B.1.2 Conditional and Unconditional Jumps (Core Instructions)

The instruction format for conditional and unconditional jumps, as shown in Figure B–3, consists of two main fields to form a 16-bit code:

- operational code (op-code) field, six bits
- jump offset field, ten bits

The operational-code field is composed of the op-code (three bits), and three bits according to the following conditions.

Figure B–3. Conditional and Unconditional Jump Instructions



Conditional jumps jump to addresses in the range of –511 to +512 words relative to the current address. The assembler computes the signed offsets and inserts them into the op-code.

JC/JHS	Label	Jump to label if carry bit is set
JEQ/JZ	Label	Jump to label if zero bit is set
JGE	Label	Jump to label if (N .XOR. V) = 0
JL	Label	Jump to label if (N .XOR. V) = 1
JMP	Label	Jump to label unconditionally
JN	Label	Jump to label if negative bit is set
JNC/JLO	Label	Jump to label if carry bit is reset
JNE/JNZ	Label	Jump to label if zero bit is reset

Note: Conditional and Unconditional Jumps

Conditional and unconditional jumps do not affect the status bits.

A jump that is taken alters the PC with the offset:

PCnew = PCold + 2 + 2*offset

A jump that is not taken continues the program with the ascending instruction.

B.1.3 Emulated Instructions

The following instructions can be emulated with the reduced instruction set without additional code words. The assembler accepts the emulated instruction mnemonic, and inserts the applicable core instruction op-code.

The following list describes the emulated instruction short form.

Mnemonic		Description	Status Bits V	N	Z	C	Emulation	
Arithmetical instructions								
ADC[.W]	dst	Add carry to destination	*	*	*	*	ADDC	#0,dst
ADC.B	dst	Add carry to destination	*	*	*	*	ADDC.B	#0,dst
DADC[.W]	dst	Add carry decimal to destination	*	*	*	*	DADD	#0,dst
DADC.B	dst	Add carry decimal to destination	*	*	*	*	DADD.B	#0,dst
DEC[.W]	dst	Decrement destination	*	*	*	*	SUB	#1,dst
DEC.B	dst	Decrement destination	*	*	*	*	SUB.B	#1,dst
DECD[.W]	dst	Double-decrement destination	*	*	*	*	SUB	#2,dst
DECD.B	dst	Double-decrement destination	*	*	*	*	SUB.B	#2,dst
INC[.W]	dst	Increment destination	*	*	*	*	ADD	#1,dst
INC.B	dst	Increment destination	*	*	*	*	ADD.B	#1,dst
INCD[.W]	dst	Increment destination	*	*	*	*	ADD	#2,dst
INCD.B	dst	Increment destination	*	*	*	*	ADD.B	#2,dst
SBC[.W]	dst	Subtract carry from destination	*	*	*	*	SUBC	#0,dst
SBC.B	dst	Subtract carry from destination	*	*	*	*	SUBC.B	#0,dst
Logical instructions								
INV[.W]	dst	Invert destination	*	*	*	*	XOR	#0FFFFh,dst
INV.B	dst	Invert destination	*	*	*	*	XOR.B	#0FFFFh,dst
RLA[.W]	dst	Rotate left arithmetically	*	*	*	*	ADD	dst,dst
RLA.B	dst	Rotate left arithmetically	*	*	*	*	ADD.B	dst,dst
RLC[.W]	dst	Rotate left through carry	*	*	*	*	ADDC	dst,dst
RLC.B	dst	Rotate left through carry	*	*	*	*	ADDC.B	dst,dst
Data instructions (common use)								
CLR[.W]		Clear destination	–	–	–	–	MOV	#0,dst
CLR.B		Clear destination	–	–	–	–	MOV.B	#0,dst
CLRC		Clear carry bit	–	–	–	0	BIC	#1,SR
CLRN		Clear negative bit	–	0	–	–	BIC	#4,SR
CLRZ		Clear zero bit	–	–	0	–	BIC	#2,SR
POP	dst	Item from stack	–	–	–	–	MOV	@SP+,dst
SETC		Set carry bit	–	–	–	1	BIS	#1,SR
SETN		Set negative bit	–	1	–	–	BIS	#4,SR
SETZ		Set zero bit	–	–	1	–	BIS	#2,SR
TST[.W]	dst	Test destination	0	*	*	1	CMP	#0,dst
TST.B	dst	Test destination	0	*	*	1	CMP.B	#0,dst
Program flow instructions								
BR	dst	Branch to	–	–	–	–	MOV	dst,PC
DINT		Disable interrupt	–	–	–	–	BIC	#8,SR
EINT		Enable interrupt	–	–	–	–	BIS	#8,SR
NOP		No operation	–	–	–	–	MOV	#0h,#0h
RET		Return from subroutine	–	–	–	–	MOV	@SP+,PC

B.2 Instruction Set Description

This section catalogues and describes all core and emulated instructions in alphabetical order. Some examples serve as explanations and others as application hints.

The suffix .W or no suffix in the instruction mnemonic results in a word operation.

The suffix .B at the instruction mnemonic results in a byte operation.

ADC[.W] Add carry to destination
ADC.B Add carry to destination

Syntax

```
ADC     dst   or   ADC.W   dst
ADC.B   dst
```

Operation dst + C –> dst

Emulation

```
ADDC    #0,dst
ADDC.B  #0,dst
```

Description The carry bit (C) is added to the destination operand. The previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if dst was incremented from 0FFFFh to 0000, reset otherwise
Set if dst was incremented from 0FFh to 00, reset otherwise
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The 16-bit counter pointed to by R13 is added to a 32-bit counter pointed to by R12.

```
ADD     @R13,0(R12)     ; Add LSDs
ADC     2(R12)          ; Add carry to MSD
```

Example The 8-bit counter pointed to by R13 is added to a 16-bit counter pointed to by R12.

```
ADD.B   @R13,0(R12)     ; Add LSDs
ADC.B   1(R12)          ; Add carry to MSD
```

ADD[.W] Add source to destination
ADD.B Add source to destination

Syntax

```
ADD     src,dst     or      ADD.W   src,dst
ADD.B   src,dst
```

Operation src + dst –> dst

Description The source operand is added to the destination operand. The source operand is not affected. The previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the result, cleared if not
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example R5 is increased by 10. The jump to TONI is performed on a carry.

```
ADD     #10,R5
JC      TONI        ; Carry occurred
.....               ; No carry
```

Example R5 is increased by 10. The jump to TONI is performed on a carry.

```
ADD.B   #10,R5      ; Add 10 to Lowbyte of R5
JC      TONI        ; Carry occurred, if (R5) ≥ 246 [0Ah+0F6h]
.....               ; No carry
```

ADDC[.W] Add source and carry to destination
ADDC.B Add source and carry to destination

Syntax

```
ADDC     src,dst    or    ADDC.W    src,dst
ADDC.B   src,dst
```

Operation src + dst + C –> dst

Description The source operand and the carry bit (C) are added to the destination operand. The source operand is not affected. The previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the MSB of the result, reset otherwise
V**:** Set if an arithmetic overflow occurs, otherwise reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The 32-bit counter pointed to by R13 is added to a 32-bit counter, eleven words (20/2 + 2/2) above the pointer in R13.

```
ADD      @R13+,20(R13)   ; ADD LSDs with no carry in
ADDC     @R13+,20(R13)   ; ADD MSDs with carry
...                      ; resulting from the LSDs
```

Example The 24-bit counter pointed to by R13 is added to a 24-bit counter, eleven words above the pointer in R13.

```
ADD.B    @R13+,10(R13)   ; ADD LSDs with no carry in
ADDC.B   @R13+,10(R13)   ; ADD medium Bits with carry
ADDC.B   @R13+,10(R13)   ; ADD MSDs with carry
...                      ; resulting from the LSDs
```

AND[.W] Source AND destination
AND.B Source AND destination

Syntax

```
AND     src,dst     or   AND.W src,dst
AND.B   src,dst
```

Operation src .AND. dst –> dst

Description The source operand and the destination operand are logically ANDed. The result is placed into the destination.

Status Bits

N: Set if result MSB is set, reset if not set
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (= .NOT. Zero)
V: Reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The bits set in R5 are used as a mask (#0AA55h) for the word addressed by TOM. If the result is zero, a branch is taken to label TONI.

```
MOV     #0AA55h,R5      ; Load mask into register R5
AND     R5,TOM          ; mask word addressed by TOM with R5
JZ      TONI            ;
.....                   ; Result is not zero
;
;
;       or
;
;
AND     #0AA55h,TOM
JZ      TONI
```

Example The bits of mask #0A5h are logically ANDed with the low byte TOM. If the result is zero, a branch is taken to label TONI.

```
AND.B   #0A5h,TOM       ; mask Lowbyte TOM with R5
JZ      TONI            ;
.....                   ; Result is not zero
```

BIC[.W] Clear bits in destination
BIC.B Clear bits in destination

Syntax

```
BIC     src,dst    or  BIC.W  src,dst
BIC.B   src,dst
```

Operation .NOT.src .AND. dst –> dst

Description The inverted source operand and the destination operand are logically ANDed. The result is placed into the destination. The source operand is not affected.

Status Bits

N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The six MSBs of the RAM word LEO are cleared.

```
BIC     #0FC00h,LEO          ; Clear 6 MSBs in MEM(LEO)
```

Example The five MSBs of the RAM byte LEO are cleared.

```
BIC.B   #0F8h,LEO            ; Clear 5 MSBs in Ram location LEO
```

Example The port pins P0 and P1 are cleared.

```
P0OUT   .equ   011h;         Definition of port address
P0_0    .equ                 01h
P0_1    .equ                 02h

BIC.B   #P0_0+P0_1,&P0OUT    ;Set P0.0 and P0.1 to low
```

BIS[.W] Set bits in destination
BIS.B Set bits in destination

Syntax

```
BIS     src,dst    or  BIS.W    src,dst
BIS.B   src,dst
```

Operation src .OR. dst –> dst

Description The source operand and the destination operand are logically ORed. The result is placed into the destination. The source operand is not affected.

Status Bits

N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The six LSBs of the RAM word TOM are set.

```
BIS     #003Fh,TOM; set the six LSBs in RAM location TOM
```

Example Start an A/D- conversion

```
ASOC    .equ    1       ; Start of conversion bit
ACTL    .equ    114h    ; ADC control register

BIS     #ASOC,&ACTL     ; Start A/D-conversion
```

Example The three MSBs of RAM byte TOM are set.

```
BIS.B   #0E0h,TOM       ; set the 3 MSBs in RAM location TOM
```

Example Port pins P0 and P1 are set to high.

```
P0OUT   .equ    011h
P0      .equ    01h
P1      .equ    02h

BIS.B   #P0+P1,&P0OUT
```

BIT[.W] Test bits in destination
BIT.B Test bits in destination

Syntax BIT src,dst or BIT.W src,dst

Operation src .AND. dst

Description The source and destination operands are logically ANDed. The result affects only the status bits. The source and destination operands are not affected.

Status Bits
N: Set if MSB of result is set, reset otherwise
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (.NOT. Zero)
V: Reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example If bit 9 of R8 is set, a branch is taken to label TOM.

```
BIT     #0200h,R8       ; bit 9 of R8 set?
JNZ     TOM             ; Yes, branch to TOM
...                     ; No, proceed
```

Example Determine which A/D channel is configured by the MUX.

```
ACTL    .equ            114h            ; ADC control register
BIT     #4,&ACTL                        ; Is channel 0 selected?
jnz     END                             ; Yes, branch to END
```

Example If bit 3 of R8 is set, a branch is taken to label TOM.

```
BIT.B   #8,R8
JC      TOM
```

Example A serial communication receive bit (RCV) is tested. Because the carry bit is equal to the state of the tested bit while using the BIT instruction to test a single bit, the carry bit is used by the subsequent instruction; the read information is shifted into register RECBUF.

```
;
; Serial communication with LSB is shifted first:
                                ; xxxx  xxxx  xxxx  xxxx
BIT.B   #RCV,RCCTL      ; Bit info into carry
RRC     RECBUF          ; Carry –> MSB of RECBUF
                        ; cxxx  xxxx
.....                   ; repeat previous two instructions
.....                   ; 8 times
                        ; cccc  cccc
                        ;  ^     ^
                        ; MSB   LSB
; Serial communication with MSB is shifted first:
BIT.B   #RCV,RCCTL      ; Bit info into carry
RLC.B   RECBUF          ; Carry –> LSB of RECBUF
                        ; xxxx  xxxc
.....                   ; repeat previous two instructions
.....                   ; 8 times
                        ; cccc  cccc
                        ; |     LSB
                        ; MSB
```

* **BR, BRANCH** Branch to ..... destination

Syntax BR dst

Operation dst –> PC

Emulation MOV dst,PC

Description An unconditional branch is taken to an address anywhere in the 64K address space. All source addressing modes can be used. The branch instruction is a word instruction.

Status Bits Status bits are not affected.

Example Examples for all addressing modes are given.

```
BR      #EXEC   ;Branch to label EXEC or direct branch (e.g. #0A4h)
                ; Core instruction MOV @PC+,PC

BR      EXEC    ; Branch to the address contained in EXEC
                ; Core instruction MOV X(PC),PC
                ; Indirect address

BR      &EXEC   ; Branch to the address contained in absolute
                ; address EXEC
                ; Core instruction MOV X(0),PC
                ; Indirect address

BR      R5      ; Branch to the address contained in R5
                ; Core instruction MOV R5,PC
                ; Indirect R5

BR      @R5     ; Branch to the address contained in the word
                ; pointed to by R5.
                ; Core instruction MOV @R5,PC
                ; Indirect, indirect R5

BR      @R5+    ; Branch to the address contained in the word pointed
                ; to by R5 and increment pointer in R5 afterwards.
                ; The next time—S/W flow uses R5 pointer—it can
                ; alter program execution due to access to
                ; next address in a table pointed to by R5
                ; Core instruction MOV @R5,PC
                ; Indirect, indirect R5 with autoincrement

BR      X(R5)   ; Branch to the address contained in the address
                ; pointed to by R5 + X (e.g. table with address
                ; starting at X). X can be an address or a label
                ; Core instruction MOV X(R5),PC
                ; Indirect, indirect R5 + X
```

CALL Subroutine

Syntax CALL dst

Operation

dst	–> tmp	dst is evaluated and stored
SP – 2	–> SP	
PC	–> @SP	PC updated to TOS
tmp	–> PC	dst saved to PC

Description A subroutine call is made to an address anywhere in the 64K address space. All addressing modes can be used. The return address (the address of the following instruction) is stored on the stack. The call instruction is a word instruction.

Status Bits Status bits are not affected.

Example Examples for all addressing modes are given.

```
CALL    #EXEC   ; Call on label EXEC or immediate address (e.g. #0A4h)
                ; SP–2 → SP, PC+2 → @SP, @PC+ → PC

CALL    EXEC    ; Call on the address contained in EXEC
                ; SP–2 → SP, PC+2 → @SP, X(PC) → PC
                ; Indirect address

CALL    &EXEC   ; Call on the address contained in absolute address
                ; EXEC
                ; SP–2 → SP, PC+2 → @SP, X(PC) → PC
                ; Indirect address

CALL    R5      ; Call on the address contained in R5
                ; SP–2 → SP, PC+2 → @SP, R5 → PC
                ; Indirect R5

CALL    @R5     ; Call on the address contained in the word
                ; pointed to by R5
                ; SP–2 → SP, PC+2 → @SP, @R5 → PC
                ; Indirect, indirect R5

CALL    @R5+    ; Call on the address contained in the word
                ; pointed to by R5 and increment pointer in R5.
                ;  The next time—S/W flow uses R5 pointer—
                ; it can alter the program execution due to
                ;  access to next address in a table pointed to by R5
                ; SP–2 → SP, PC+2 → @SP, @R5 → PC
                ; Indirect, indirect R5 with autoincrement

CALL    X(R5)   ; Call on the address contained in the address pointed
                ; to by R5 + X (e.g. table with address starting at X)
                ; X can be an address or a label
                ; SP–2 → SP, PC+2 → @SP, X(R5) → PC
                ; Indirect indirect R5 + X
```

*** CLR[.W]** Clear destination
* **CLR.B** Clear destination

Syntax
```
CLR     dst     or   CLR.W dst
CLR.B   dst
```

Operation 0 –> dst

Emulation
```
MOV     #0,dst
MOV.B   #0,dst
```

Description The destination operand is cleared.

Status Bits Status bits are not affected.

Example RAM word TONI is cleared.

```
CLR     TONI        ; 0 –> TONI
```

Example Register R5 is cleared.

```
CLR     R5
```

Example RAM byte TONI is cleared.

```
CLR.B   TONI        ; 0 –> TONI
```

*** CLRC** Clear carry bit

Syntax CLRC

Operation 0 –> C

Emulation BIC #1,SR

Description The carry bit (C) is cleared. The clear carry instruction is a word instruction.

Status Bits
N: Not affected
Z: Not affected
C: Cleared
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The 16-bit decimal counter pointed to by R13 is added to a 32-bit counter pointed to by R12.

```
CLRC                ; C=0: defines start
DADD  @R13,0(R12)   ; add 16-bit counter to low word of 32-bit counter
DADC  2(R12)        ; add carry to high word of 32-bit counter
```

*** CLRN** Clear negative bit

Syntax CLRN

Operation 0 → N
or
(.NOT.src .AND. dst –> dst)

Emulation BIC #4,SR

Description The constant 04h is inverted (0FFFBh) and is logically ANDed with the destination operand. The result is placed into the destination. The clear negative bit instruction is a word instruction.

Status Bits
N: Reset to 0
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The Negative bit in the status register is cleared. This avoids special treatment with negative numbers of the subroutine called.

```
        CLRN
        CALL    SUBR
        ......
        ......
SUBR    JN      SUBRET  ; If input is negative: do nothing and return
        ......
        ......
        ......
SUBRET  RET
```

*** CLRZ** Clear zero bit

Syntax CLRZ

Operation 0 → Z
or
(.NOT.src .AND. dst –> dst)

Emulation BIC #2,SR

Description The constant 02h is inverted (0FFFDh) and logically ANDed with the destination operand. The result is placed into the destination. The clear zero bit instruction is a word instruction.

Status Bits
N: Not affected
Z: Reset to 0
C: Not affected
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The zero bit in the status register is cleared.

```
CLRZ
```

CMP[.W] Compare source and destination
CMP.B Compare source and destination

Syntax

```
CMP     src,dst    or    CMP.W    src,dst
CMP.B   src,dst
```

Operation

dst + .NOT.src + 1
or
(dst – src)

Description The source operand is subtracted from the destination operand. This is accomplished by adding the 1s complement of the source operand plus 1. The two operands are not affected and the result is not stored; only the status bits are affected.

Status Bits

N: Set if result is negative, reset if positive (src >= dst)
Z: Set if result is zero, reset otherwise (src = dst)
C: Set if there is a carry from the MSB of the result, reset otherwise
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example R5 and R6 are compared. If they are equal, the program continues at the label EQUAL.

```
CMP     R5,R6      ; R5 = R6?
JEQ     EQUAL      ; YES, JUMP
```

Example Two RAM blocks are compared. If they are not equal, the program branches to the label ERROR.

```
        MOV   #NUM,R5            ; number of words to be compared
L$1     CMP   &BLOCK1,&BLOCK2    ; Are Words equal?
        JNZ   ERROR              ; No, branch to ERROR
        DEC   R5                 ; Are all words compared?
        JNZ   L$1                ; No, another compare
```

Example The RAM bytes addressed by EDE and TONI are compared. If they are equal, the program continues at the label EQUAL.

```
        CMP.B  EDE,TONI          ; MEM(EDE) = MEM(TONI)?
        JEQ    EQUAL             ; YES, JUMP
```

Example Check two keys connected to port pins P0 and P1. If key1 is pressed, the program branches to label MENU1; if key2 is pressed, the program branches to MENU2.

```
P0IN    .EQU   010h
KEY1    .EQU   01h
KEY2    .EQU   02h

        CMP.B  #KEY1,&P0IN
        JEQ    MENU1
        CMP.B  #KEY2,&P0IN
        JEQ    MENU2
```

*** DADC[.W]** Add carry decimally to destination
* **DADC.B** Add carry decimally to destination

Syntax

```
DADC      dst   or   DADC.W   src,dst
DADC.B    dst
```

Operation dst + C –> dst (decimally)

Emulation

```
DADD      #0,dst
DADD.B    #0,dst
```

Description The carry bit (C) is added decimally to the destination.

Status Bits

N: Set if MSB is 1
Z: Set if dst is 0, reset otherwise
C: Set if destination increments from 9999 to 0000, reset otherwise
Set if destination increments from 99 to 00, reset otherwise
V: Undefined

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The four-digit decimal number contained in R5 is added to an eight-digit decimal number pointed to by R8.

```
CLRC                ; Reset carry
                    ; next instruction's start condition is defined
DADD      R5,0(R8)  ; Add LSDs + C
DADC      2(R8)     ; Add carry to MSD
```

Example The two-digit decimal number contained in R5 is added to a four-digit decimal number pointed to by R8.

```
CLRC                ; Reset carry
                    ; next instruction's start condition is defined
DADD.B    R5,0(R8)  ; Add LSDs + C
DADC      1(R8)     ; Add carry to MSDs
```

DADD[.W] Source and carry added decimally to destination
DADD.B Source and carry added decimally to destination

Syntax

```
DADD    src,dst    or  DADD.W  src,dst
DADD.B  src,dst
```

Operation src + dst + C –> dst (decimally)

Description The source operand and the destination operand are treated as four binary coded decimals (BCD) with positive signs. The source operand and the carry bit (C) are added decimally to the destination operand. The source operand is not affected. The previous contents of the destination are lost. The result is not defined for non-BCD numbers.

Status Bits

N: Set if the MSB is 1, reset otherwise
Z: Set if result is zero, reset otherwise
C: Set if the result is greater than 9999
Set if the result is greater than 99
V: Undefined

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The eight-digit BCD number contained in R5 and R6 is added decimally to an eight-digit BCD number contained in R3 and R4 (R6 and R4 contain the MSDs).

```
CLRC            ; CLEAR CARRY
DADD    R5,R3   ; add LSDs
DADD    R6,R4   ; add MSDs with carry
JC      OVERFLOW ; If carry occurs go to error handling routine
```

Example The two-digit decimal counter in the RAM byte CNT is incremented by one.

```
CLRC            ; clear Carry
DADD.B  #1,CNT  ; increment decimal counter
```

or

```
SETC
DADD.B  #0,CNT  ; ≡ DADC.B  CNT
```

*** DEC[.W]** Decrement destination
*** DEC.B** Decrement destination

Syntax

DEC dst or DEC.W dst
DEC.B dst

Operation

dst – 1 –> dst

Emulation

SUB #1,dst

Emulation

SUB.B #1,dst

Description

The destination operand is decremented by one. The original contents are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if dst contained 1, reset otherwise
C: Reset if dst contained 0, set otherwise
V: Set if an arithmetic overflow occurs, otherwise reset.
Set if initial value of destination was 08000h, otherwise reset.
Set if initial value of destination was 080h, otherwise reset.

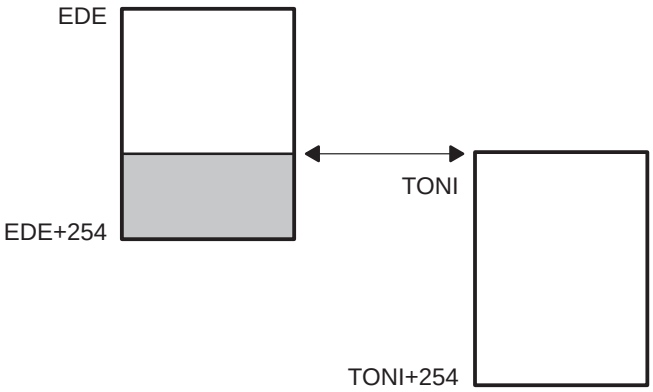
Mode Bits

OscOff**,** CPUOff, and GIE are not affected.

Example R10 is decremented by 1

```
                DEC     R10     ; Decrement R10

; Move a block of 255 bytes from memory location starting with EDE to memory location starting with
;TONI. Tables should not overlap: start of destination address TONI must not be within the range EDE
; to EDE+0FEh
;
                MOV     #EDE,R6
                MOV     #255,R10
L$1             MOV.B   @R6+,TONI–EDE–1(R6)
                DEC     R10
                JNZ     L$1

; Do not transfer tables using the routine above with the overlap shown in Figure B–4.
```

Figure B–4. Decrement Overlap



Example Memory byte at address LEO is decremented by one.

```
                DEC.B   LEO     ; Decrement MEM(LEO)

; Move a block of 255 bytes from memory location starting with EDE to memory location starting with
; TONI. Tables should not overlap: start of destination address TONI must not be within the range EDE
; to EDE+0FEh
;
                MOV     #EDE,R6
                MOV.B   #255,LEO
L$1             MOV.B   @R6+,TONI–EDE–1(R6)
                DEC.B   LEO
                JNZ     L$1
```

*** DECD[.W]** Double-decrement destination
*** DECD.B** Double-decrement destination

Syntax
DECD dst or DECD.W dst
DECD.B dst

Operation
dst – 2 –> dst

Emulation
SUB #2,dst
Emulation
SUB.B #2,dst

Description
The destination operand is decremented by two. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 2, reset otherwise
C: Reset if dst contained 0 or 1, set otherwise
V: Set if an arithmetic overflow occurs, otherwise reset.
Set if initial value of destination was 08001 or 08000h, otherwise reset.
Set if initial value of destination was 081 or 080h, otherwise reset.

Mode Bits
OscOff, CPUOff, and GIE are not affected.

Example
R10 is decremented by 2.

```
        DECD    R10         ; Decrement R10 by two
```

```
; Move a block of 255 words from memory location starting with EDE to memory location
; starting with TONI
; Tables should not overlap: start of destination address TONI must not be within the
; range EDE to EDE+0FEh
;
        MOV     #EDE,R6
        MOV     #510,R10
L$1     MOV     @R6+,TONI–EDE–2(R6)
        DECD    R10
        JNZ     L$1
```

Example
Memory at location LEO is decremented by two.

```
        DECD.B  LEO         ; Decrement MEM(LEO)
```

Decrement status byte STATUS by two.

```
        DECD.B  STATUS
```

*** DINT** Disable (general) interrupts

Syntax DINT

Operation 0 → GIE
or
(0FFF7h .AND. SR → SR / .NOT.src .AND. dst –> dst)

Emulation BIC #8,SR

Description All interrupts are disabled.
The constant 08h is inverted and logically ANDed with the status register (SR). The result is placed into the SR.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits GIE is reset. OscOff and CPUOff are not affected.

Example The general interrupt enable (GIE) bit in the status register is cleared to allow a nondisrupted move of a 32-bit counter. This ensures that the counter is not modified during the move by any interrupt.

```
DINT                    ; All interrupt events using the GIE bit are disabled
NOP
MOV     COUNTHI,R5      ; Copy counter
MOV     COUNTLO,R6
EINT                    ; All interrupt events using the GIE bit are enabled
```

Note: Disable Interrupt

If any code sequence needs to be protected from interruption, the DINT should be executed at least one instruction before the beginning of the uninterruptible sequence, or should be followed by an NOP.

*** EINT** Enable (general) interrupts

Syntax EINT

Operation 1 → GIE
or
(0008h .OR. SR –> SR / .NOT.src .OR. dst –> dst)

Emulation BIS #8,SR

Description All interrupts are enabled.
The constant #08h and the status register SR are logically ORed. The result is placed into the SR.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits GIE is set. OscOff and CPUOff are not affected.

Example The general interrupt enable (GIE) bit in the status register is set.

```
; Interrupt routine of port P0.2 to P0.7
; The interrupt level is the lowest in the system
; P0IN is the address of the register where all port bits are read. P0IFG is the address of
; the register where all interrupt events are latched.
;
                PUSH.B  &P0IN
                BIC.B   @SP,&P0IFG  ; Reset only accepted flags
                EINT                ; Preset port 0 interrupt flags stored on stack
                                    ; other interrupts are allowed
                BIT     #Mask,@SP
                JEQ     MaskOK      ; Flags are present identically to mask: jump
                ......
MaskOK          BIC     #Mask,@SP
                ......
                INCD    SP          ; Housekeeping: inverse to PUSH instruction
                                    ; at the start of interrupt subroutine. Corrects
                                    ; the stack pointer.
                RETI
```

Note: Enable Interrupt

The instruction following the enable interrupt instruction (EINT) is always executed, even if an interrupt service request is pending when the interrupts are enable.

*** INC[.W]** Increment destination
*** INC.B** Increment destination

Syntax

```
INC     dst    or   INC.W  dst
INC.B   dst
```

Operation dst + 1 –> dst

Emulation

```
ADD     #1,dst
```

Description The destination operand is incremented by one. The original contents are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise
C: Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise
V: Set if dst contained 07FFFh, reset otherwise
Set if dst contained 07Fh, reset otherwise

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The item on the top of a software stack (not the system stack) for byte data is removed.

```
SSP     .EQU    R4
;
        INC     SSP     ; Remove TOSS (top of SW stack) by increment
                        ; Do not use INC.B since SSP is a word register
```

Example The status byte of a process STATUS is incremented. When it is equal to 11, a branch to OVFL is taken.

```
        INC.B   STATUS
        CMP.B   #11,STATUS
        JEQ     OVFL
```

*** INCD[.W]** Double-increment destination
*** INCD.B** Double-increment destination

Syntax
```
INCD    dst   or  INCD.W   dst
INCD.B  dst
```

Operation dst + 2 –> dst

Emulation
```
ADD     #2,dst
```
Emulation
```
ADD.B   #2,dst
```

Example The destination operand is incremented by two. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 0FFFEh, reset otherwise
Set if dst contained 0FEh, reset otherwise
C: Set if dst contained 0FFFEh or 0FFFFh, reset otherwise
Set if dst contained 0FEh or 0FFh, reset otherwise
V: Set if dst contained 07FFEh or 07FFFh, reset otherwise
Set if dst contained 07Eh or 07Fh, reset otherwise

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The item on the top of the stack (TOS) is removed without using a register.

```
.......
PUSH    R5      ; R5 is the result of a calculation, which is stored
                ; in the system stack
INCD    SP      ; Remove TOS by double-increment from stack
                ; Do not use INCD.B, SP is a word-aligned
                ; register
RET
```

Example The byte on the top of the stack is incremented by two.

```
INCD.B  0(SP)   ; Byte on TOS is increment by two
```

*** INV[.W]** Invert destination
*** INV.B** Invert destination

Syntax
```
INV     dst
INV.B   dst
```

Operation .NOT.dst –> dst

Emulation
```
XOR     #0FFFFh,dst
```
Emulation
```
XOR.B   #0FFh,dst
```

Description The destination operand is inverted. The original contents are lost.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if dst contained 0FFFFh, reset otherwise
Set if dst contained 0FFh, reset otherwise
C: Set if result is not zero, reset otherwise (= .NOT. Zero)
Set if result is not zero, reset otherwise (= .NOT. Zero)
V: Set if initial destination operand was negative, otherwise reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example Content of R5 is negated (twos complement).
```
MOV     #00Aeh,R5   ;                       R5 = 000AEh
INV     R5          ; Invert R5,            R5 = 0FF51h
INC     R5          ; R5 is now negated,    R5 = 0FF52h
```

Example Content of memory byte LEO is negated.
```
MOV.B   #0AEh,LEO   ;                       MEM(LEO) = 0AEh
INV.B   LEO         ; Invert LEO,           MEM(LEO) = 051h
INC.B   LEO         ; MEM(LEO) is negated,  MEM(LEO) = 052h
```

JC Jump if carry set
JHS Jump if higher or same

Syntax

```
JC      label
JHS     label
```

Operation

If C = 1: PC + 2 × offset –> PC
If C = 0: execute following instruction

Description

The status register carry bit (C) is tested. If it is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If C is reset, the next instruction following the jump is executed. JC (jump if carry/higher or same) is used for the comparison of unsigned numbers (0 to 65536).

Status Bits

Status bits are not affected.

Example

The P0IN.1 signal is used to define or control the program flow.

```
BIT     #10h,&P0IN      ; State of signal –> Carry
JC      PROGA           ; If carry=1 then execute program routine A
.....                   ; Carry=0, execute program here
```

Example

R5 is compared to 15. If the content is higher or the same, branch to LABEL.

```
CMP     #15,R5
JHS     LABEL           ; Jump is taken if R5 ≥ 15
.....                   ; Continue here if R5 < 15
```

JEQ, JZ Jump if equal, jump if zero

Syntax JEQ label, JZ label

Operation If Z = 1: PC + 2 × offset –> PC
If Z = 0: execute following instruction

Description The status register zero bit (Z) is tested. If it is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If Z is not set, the instruction following the jump is executed.

Status Bits Status bits are not affected.

Example Jump to address TONI if R7 contains zero.

```
TST     R7          ; Test R7
JZ      TONI        ; if zero: JUMP
```

Example Jump to address LEO if R6 is equal to the table contents.

```
CMP     R6,Table(R5)  ; Compare content of R6 with content of
                      ; MEM (table address + content of R5)
JEQ     LEO           ; Jump if both data are equal
.....                 ; No, data are not equal, continue here
```

Example Branch to LABEL if R5 is 0.

```
TST     R5
JZ      LABEL
.....
```

JGE Jump if greater or equal

Syntax JGE label

Operation If (N .XOR. V) = 0 then jump to label: PC + 2 × offset –> PC
If (N .XOR. V) = 1 then execute the following instruction

Description The status register negative bit (N) and overflow bit (V) are tested. If both N and V are set or reset, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If only one is set, the instruction following the jump is executed.

This allows comparison of signed integers.

Status Bits Status bits are not affected.

Example When the content of R6 is greater or equal to the memory pointed to by R7, the program continues at label EDE.

```
CMP     @R7,R6      ; R6 ≥ (R7)?, compare on signed numbers
JGE     EDE         ; Yes, R6 ≥ (R7)
.....               ; No, proceed
.....
.....
```

JL Jump if less

Syntax JL label

Operation If (N .XOR. V) = 1 then jump to label: PC + 2 × offset –> PC
If (N .XOR. V) = 0 then execute following instruction

Description The status register negative bit (N) and overflow bit (V) are tested. If only one is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If both N and V are set or reset, the instruction following the jump is executed.

This allows comparison of signed integers.

Status Bits Status bits are not affected.

Example When the content of R6 is less than the memory pointed to by R7, the program continues at label EDE.

```
CMP     @R7,R6      ; R6 < (R7)?,  compare on signed numbers
JL      EDE         ; Yes, R6 < (R7)
.....               ; No, proceed
.....
.....
```

JMP	Jump unconditionally
Syntax	MP label
Operation	PC + 2 × offset –> PC
Description	The 10-bit signed offset contained in the instruction LSBs is added to the program counter.
Status Bits	Status bits are not affected.
Hint:	This one-word instruction replaces the BRANCH instruction in the range of –511 to +512 words relative to the current program counter.

JN Jump if negative

Syntax JN label

Operation if N = 1: PC + 2 × offset –> PC
if N = 0: execute following instruction

Description The negative bit (N) of the status register is tested. If it is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If N is reset, the next instruction following the jump is executed.

Status Bits Status bits are not affected.

Example The result of a computation in R5 is to be subtracted from COUNT. If the result is negative, COUNT is to be cleared and the program continues execution in another path.

```
        SUB     R5,COUNT    ; COUNT – R5 –> COUNT
        JN      L$1         ; If negative continue with COUNT=0 at PC=L$1
        ......              ; Continue with COUNT≥0
        ......
        ......
        ......
L$1     CLR     COUNT
        ......
        ......
        ......
```

JNC Jump if carry not set
JLO Jump if lower

Syntax

```
JNC     label
JNC     label
```

Operation

if C = 0: PC + 2 × offset –> PC
if C = 1: execute following instruction

Description

The status register carry bit (C) is tested. If it is reset, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If C is set, the next instruction following the jump is executed. JNC (jump if no carry/lower) is used for the comparison of unsigned numbers (0 to 65536).

Status Bits

Status bits are not affected.

Example

The result in R6 is added in BUFFER. If an overflow occurs, an error handling routine at address ERROR is used.

```
        ADD     R6,BUFFER   ; BUFFER + R6 –> BUFFER
        JNC     CONT        ; No carry, jump to CONT
ERROR   .....               ; Error handler start
        .....
        .....
        .....
CONT    .....               ; Continue with normal program flow
        .....
        .....
```

Example

Branch to STL2 if byte STATUS contains 1 or 0.

```
        CMP.B   #2,STATUS
        JLO     STL2        ; STATUS < 2
        .....               ; STATUS ≥ 2, continue here
```

JNE, JNZ Jump if not equal, jump if not zero

Syntax JNE label, JNZ label

Operation If Z = 0: PC + 2 × offset –> PC
If Z = 1: execute following instruction

Description The status register zero bit (Z) is tested. If it is reset, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If Z is set, the next instruction following the jump is executed.

Status Bits Status bits are not affected.

Example Jump to address TONI if R7 and R8 have different contents.

```
CMP     R7,R8     ; COMPARE R7 WITH R8
JNE     TONI      ; if different: jump
.....             ; if equal, continue
```

MOV[.W] Move source to destination
MOV.B Move source to destination

Syntax

MOV src,dst or MOV.W src,dst
MOV.B src,dst

Operation

src –> dst

Description

The source operand is moved to the destination.
The source operand is not affected. The previous contents of the destination are lost.

Status Bits

Status bits are not affected.

Mode Bits

OscOff, CPUOff, and GIE are not affected.

Example

The contents of table EDE (word data) are copied to table TOM. The length of the tables must be 020h locations.

```
        MOV   #EDE,R10              ; Prepare pointer
        MOV   #020h,R9              ; Prepare counter
Loop    MOV   @R10+,TOM–EDE–2(R10)  ; Use pointer in R10 for both tables
        DEC   R9                    ; Decrement counter
        JNZ   Loop                  ; Counter ≠ 0, continue copying
        ......                      ; Copying completed
        ......
        ......
```

Example

The contents of table EDE (byte data) are copied to table TOM. The length of the tables should be 020h locations

```
        MOV   #EDE,R10              ; Prepare pointer
        MOV   #020h,R9              ; Prepare counter
Loop    MOV.B @R10+,TOM–EDE–1(R10)  ; Use pointer in R10 for
                                    ; both tables
        DEC   R9                    ; Decrement counter
        JNZ   Loop                  ; Counter ≠ 0, continue
                                    ; copying
        ......                      ; Copying completed
        ......
        ......
```

*** NOP** No operation

Syntax NOP

Operation None

Emulation MOV #0,#0

Description No operation is performed. The instruction may be used for the elimination of instructions during the software check or for defined waiting times.

Status Bits Status bits are not affected.

The NOP instruction is mainly used for two purposes:

- To hold one, two or three memory words
- To adjust software timing

Note: Emulating No-Operation Instruction

Other instructions can emulate no-operation instruction using different numbers of cycles and code words.

Examples:

```
MOV     0(R4),0(R4)     ; 6 cycles, 3 words
MOV     @R4,0(R4)       ; 5 cycles, 2 words
BIC     #0,EDE(R4)      ; 4 cycles, 2 words
JMP     $+2             ; 2 cycles, 1 word
BIC     #0,R5           ; 1 cycle, 1 word
```

*** POP[.W]** Pop word from stack to destination
*** POP.B** Pop byte from stack to destination

Syntax

```
POP     dst
POP.B   dst
```

Operation

```
@SP   –> dst
SP + 2 –> SP
```

Emulation

```
MOV     @SP+,dst    or    MOV.W    @SP+,dst
```

Emulation

```
MOV.B   @SP+,dst
```

Description The stack location pointed to by the stack pointer (TOS) is moved to the destination. The stack pointer is incremented by two afterwards.

Status Bits Status bits are not affected.

Example The contents of R7 and the status register are restored from the stack.

```
POP     R7      ; Restore R7
POP     SR      ; Restore status register
```

Example The contents of RAM byte LEO is restored from the stack.

```
POP.B   LEO     ; The low byte of the stack is moved to LEO.
```

Example The contents of R7 is restored from the stack.

```
POP.B   R7      ; The low byte of the stack is moved to R7,
                ; the high byte of R7 is 00h
```

Example The contents of the memory pointed to by R7 and the status register are restored from the stack.

```
POP.B   0(R7)   ; The low byte of the stack is moved to the
                ; the byte which is pointed to by R7
                : Example:  R7 = 203h
                ;           Mem(R7) = low byte of system stack
                : Example:  R7 = 20Ah
                ;           Mem(R7) = low byte of system stack
POP     SR
```

Note: The System Stack Pointer

The system stack pointer (SP) is always incremented by two, independent of the byte suffix.

PUSH[.W] Push word onto stack
PUSH.B Push byte onto stack

Syntax

```
PUSH    src    or    PUSH.W    src
PUSH.B  src
```

Operation

SP – 2 → SP
src → @SP

Description The stack pointer is decremented by two, then the source operand is moved to the RAM word addressed by the stack pointer (TOS).

Status Bits

N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The contents of the status register and R8 are saved on the stack.

```
PUSH    SR      ; save status register
PUSH    R8      ; save R8
```

Example The contents of the peripheral TCDAT is saved on the stack.

```
PUSH.B  &TCDAT  ; save data from 8-bit peripheral module,
                ; address TCDAT, onto stack
```

Note: The System Stack Pointer

The system stack pointer (SP) is always decremented by two, independent of the byte suffix.

*** RET**	Return from subroutine
Syntax	RET
Operation	@SP→ PC SP + 2 → SP
Emulation	MOV @SP+,PC
Description	The return address pushed onto the stack by a CALL instruction is moved to the program counter. The program continues at the code address following the subroutine call.
Status Bits	Status bits are not affected.

RETI Return from interrupt

Syntax RETI

Operation

TOS → SR
SP + 2 → SP
TOS → PC
SP + 2 → SP

Description The status register is restored to the value at the beginning of the interrupt service routine by replacing the present SR contents with the TOS contents. The stack pointer (SP) is incremented by two.

The program counter is restored to the value at the beginning of interrupt service. This is the consecutive step after the interrupted program flow. Restoration is performed by replacing the present PC contents with the TOS memory contents. The stack pointer (SP) is incremented.

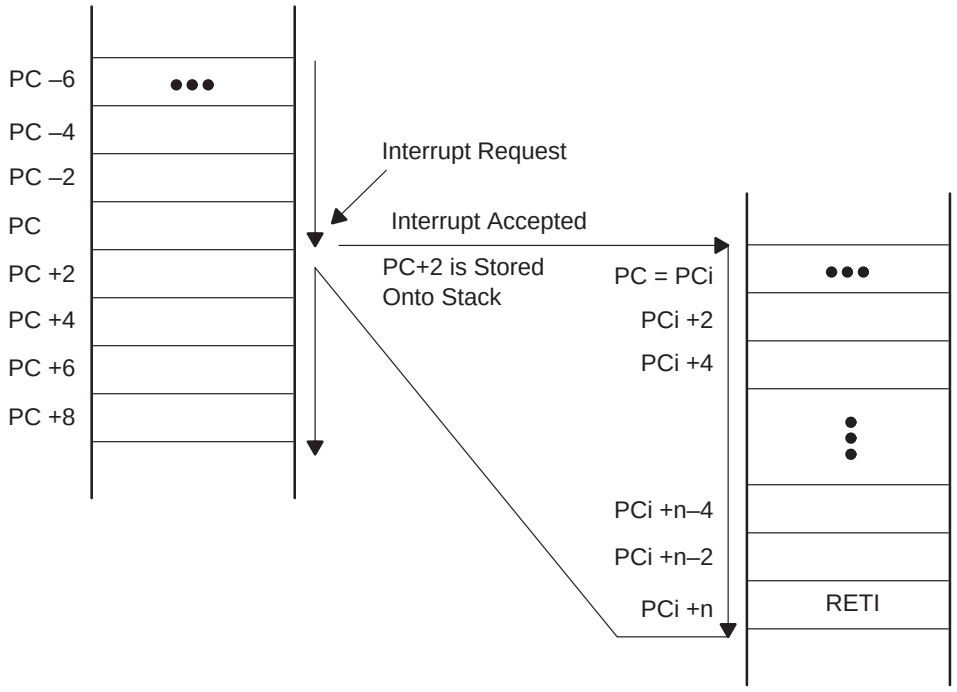
Status Bits

N: restored from system stack
Z: restored from system stack
C: restored from system stack
V: restored from system stack

Mode Bits OscOff, CPUOff, and GIE are restored from system stack.

Example Figure B–5 illustrates the main program interrupt.

Figure B–5. Main Program Interrupt



*** RLA[.W]** Rotate left arithmetically
*** RLA.B** Rotate left arithmetically

Syntax

RLA dst or RLA.W dst
RLA.B dst

Operation

C <– MSB <– MSB–1 .... LSB+1 <– LSB <– 0

Emulation

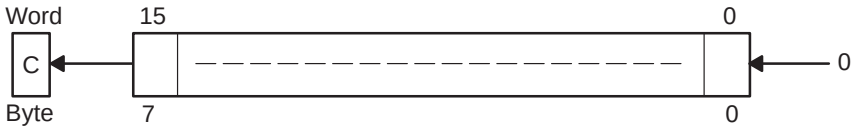
ADD dst,dst
ADD.B dst,dst

Description

The destination operand is shifted left one position as shown in Figure B–6. The MSB is shifted into the carry bit (C) and the LSB is filled with 0. The RLA instruction acts as a signed multiplication by 2.

An overflow occurs if dst ≥ 04000h and dst < 0C000h before operation is performed: the result has changed sign.

Figure B–6. Destination Operand—Arithmetic Shift Left



An overflow occurs if dst ≥ 040h and dst < 0C0h before the operation is performed: the result has changed sign.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the MSB
V: Set if an arithmetic overflow occurs:
the initial value is 04000h ≤ dst < 0C000h; otherwise it is reset
Set if an arithmetic overflow occurs:
the initial value is 040h ≤ dst < 0C0h; otherwise it is reset

Mode Bits

OscOff, CPUOff, and GIE are not affected.

Example

R7 is multiplied by 4.

```
RLA     R7      ; Shift left R7 (× 2) – emulated by  ADD  R7,R7
RLA     R7      ; Shift left R7 (× 4) – emulated by  ADD  R7,R7
```

Example

The low byte of R7 is multiplied by 4.

```
RLA.B   R7      ; Shift left low byte of R7 (× 2) – emulated by
                ; ADD.B  R7,R7
RLA.B   R7      ; Shift left low byte of R7 (× 4) – emulated by
                ; ADD.B  R7,R7
```

Note: RLA Substitution

The assembler does not recognize the instruction:

RLA @R5+ nor RLA.B @R5+.

It must be substituted by:

ADD @R5+,–2(R5) or ADD.B @R5+,–1(R5).

*** RLC[.W]** Rotate left through carry
*** RLC.B** Rotate left through carry

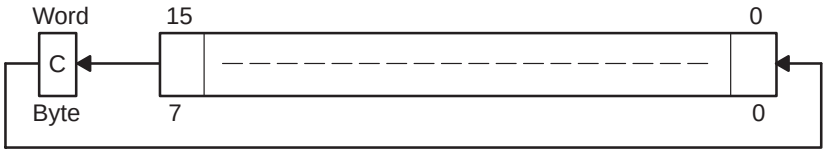
Syntax
RLC dst or RLC.W dst
RLC.B dst

Operation
C <– MSB <– MSB–1 .... LSB+1 <– LSB <– C

Emulation
ADDC dst,dst

Description
The destination operand is shifted left one position as shown in Figure B–7. The carry bit (C) is shifted into the LSB and the MSB is shifted into the carry bit (C).

Figure B–7. Destination Operand—Carry Left Shift



Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the MSB
V: Set if arithmetic overflow occurs, reset otherwise
Set if 03FFFh < $dst_{initial}$ < 0C000h, reset otherwise
Set if 03Fh < $dst_{initial}$ < 0C0h, reset otherwise

Mode Bits
OscOff, CPUOff, and GIE are not affected.

Example
R5 is shifted left one position.

```
RLC     R5          ; (R5 x 2) + C –> R5
```

Example
The input P0IN.1 information is shifted into the LSB of R5.

```
BIT.B   #2,&P0IN    ; Information –> Carry
RLC     R5          ; Carry=P0in.1 –> LSB of R5
```

Example
The MEM(LEO) content is shifted left one position.

```
RLC.B   LEO         ; Mem(LEO) x 2 + C –> Mem(LEO)
```

Example
The input P0IN.1 information is to be shifted into the LSB of R5.

```
BIT.B   #2,&P0IN    ; Information –> Carry
RLC.B   R5          ; Carry = P0in.1 –> LSB of R5
                    ; High byte of R5 is reset
```

Note: RLC and RLC.B Emulation

The assembler does not recognize the instruction:

```
RLC     @R5+.
```

It must be substituted by:

```
ADDC    @R5+,–2(R5).
```

RRA[.W] Rotate right arithmetically
RRA.B Rotate right arithmetically

Syntax

RRA dst or RRA.W dst
RRA.B dst

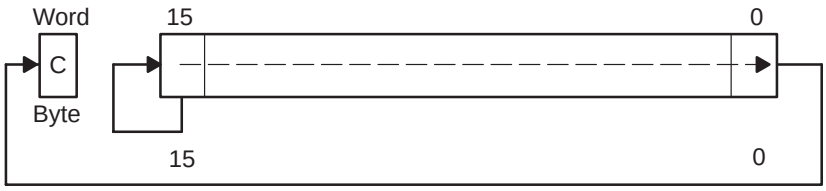
Operation

MSB –> MSB, MSB –> MSB–1, ... LSB+1 –> LSB, LSB –> C

Description

The destination operand is shifted right one position as shown in Figure B–8. The MSB is shifted into the MSB, the MSB is shifted into the MSB–1, and the LSB+1 is shifted into the LSB.

Figure B–8. Destination Operand—Arithmetic Right Shift



Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the LSB
V: Reset

Mode Bits

OscOff, CPUOff, and GIE are not affected.

Example R5 is shifted right one position. The MSB retains the old value. It operates equal to an arithmetic division by 2.

```
                RRA     R5          ; R5/2 –> R5

;               The value in R5 is multiplied by 0.75 (0.5 + 0.25).
;
                PUSH    R5          ; hold R5 temporarily using stack
                RRA     R5          ; R5 × 0.5  –>  R5
                ADD     @SP+,R5     ; R5 × 0.5 + R5 = 1.5 × R5  –> R5
                RRA     R5          ; (1.5 × R5) × 0.5 = 0.75 × R5  –> R5
                ......
                ......
; OR
;
                RRA     R5          ; R5 × 0.5  –>  R5
                PUSH    R5          ; R5 × 0.5  –>  TOS
                RRA     @SP         ; TOS × 0.5 = 0.5 × R5 × 0.5 = 0.25 × R5  –> TOS
                ADD     @SP+,R5     ; R5 × 0.5 + R5 × 0.25 = 0.75 × R5  –> R5
                ......
```

Example The low byte of R5 is shifted right one position. The MSB retains the old value. It operates equal to an arithmetic division by 2.

```
                RRA.B   R5          ; R5/2 –> R5: operation is on low byte only
                                    ; High byte of R5 is reset

;               The value in R5 (low byte only) is multiplied by 0.75 (0.5 + 0.25).
;
                PUSH.B  R5          ; hold low byte of R5 temporarily using stack
                RRA.B   R5          ; R5 × 0.5  –>  R5
                ADD.B   @SP+,R5     ; R5 × 0.5 + R5 = 1.5 × R5  –> R5
                RRA.B   R5          ; (1.5 × R5) × 0.5 = 0.75 × R5  –> R5
                ......
; OR
;
                RRA.B   R5          ; R5 × 0.5  –>  R5
                PUSH.B  R5          ; R5 × 0.5  –>  TOS
                RRA.B   @SP         ;TOS × 0.5 = 0.5 × R5 × 0.5 = 0.25 × R5  –> TOS
                ADD.B   @SP+,R5     ; R5 × 0.5 + R5 × 0.25 = 0.75 × R5  –> R5
                ......
```

RRC[.W] Rotate right through carry
RRC.B Rotate right through carry

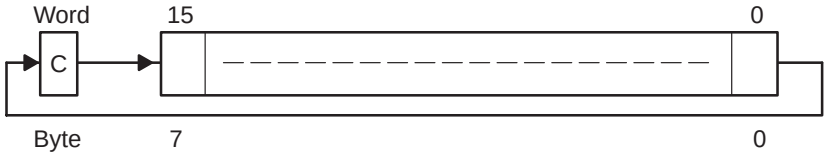
Syntax

```
RRC     dst     or      RRC.W   dst
RRC     dst
```

Operation C –> MSB –> MSB–1 .... LSB+1 –> LSB –> C

Description The destination operand is shifted right one position as shown in Figure B–6. The carry bit (C) is shifted into the MSB, the LSB is shifted into the carry bit (C).

Figure B–9. Destination Operand—Carry Right Shift



Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Loaded from the LSB
V: Set if initial destination is positive and initial carry is set, otherwise reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example R5 is shifted right one position. The MSB is loaded with 1.

```
SETC                ; Prepare carry for MSB
RRC     R5          ; R5/2 + 8000h –> R5
```

Example R5 is shifted right one position. The MSB is loaded with 1.

```
SETC                ; Prepare carry for MSB
RRC.B   R5          ; R5/2 + 80h –> R5; low byte of R5 is used
```

*** SBC[.W]** Subtract (borrow*) from destination
*** SBC.B** Subtract (borrow*) from destination

Syntax

```
SBC     dst     or      SBC.W   dst
SBC.B   dst
```

Operation

dst + 0FFFFh + C –> dst
dst + 0FFh + C –> dst

Emulation

```
SUBC    #0,dst
SUBC.B  #0,dst
```

Description The carry bit (C) is added to the destination operand minus one. The previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Reset if dst was decremented from 0000 to 0FFFFh, set otherwise
Reset if dst was decremented from 00 to 0FFh, set otherwise
V: Set if initially C = 0 and dst = 08000h
Set if initially C = 0 and dst = 080h

Mode Bits OscOff**,** CPUOff, and GIE are not affected.

Example The 16-bit counter pointed to by R13 is subtracted from a 32-bit counter pointed to by R12.

```
SUB     @R13,0(R12)     ; Subtract LSDs
SBC     2(R12)          ; Subtract carry from MSD
```

Example The 8-bit counter pointed to by R13 is subtracted from a 16-bit counter pointed to by R12.

```
SUB.B   @R13,0(R12)     ; Subtract LSDs
SBC.B   1(R12)          ; Subtract carry from MSD
```

Note: Borrow Is Treated as a .NOT.

The borrow is treated as a .NOT. carry :

Borrow	Carry bit
Yes	0
No	1

*** SETC** Set carry bit

Syntax SETC

Operation 1 –> C

Emulation BIS #1,SR

Description The carry bit (C) is set.

Status Bits
N: Not affected
Z: Not affected
C: Set
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example Emulation of the decimal subtraction:
Subtract R5 from R6 decimally
Assume that R5 = 3987 and R6 = 4137

```
DSUB    ADD     #6666h,R5       ; Move content R5 from 0–9 to 6–0Fh
                                ; R5 = 03987 + 6666 = 09FEDh
        INV     R5              ; Invert this (result back to 0–9)
                                ; R5 = .NOT. R5 = 06012h
        SETC                    ; Prepare carry = 1
        DADD    R5,R6           ; Emulate subtraction by addition of:
                                ; (10000 – R5 – 1)
                                ; R6 = R6 + R5 + 1
                                ; R6 = 4137 + 06012 + 1 = 1 0150 = 0150
```

*** SETN**	Set negative bit
Syntax	SETN
Operation	1 –> N
Emulation	BIS #4,SR
Description	The negative bit (N) is set.
Status Bits	N: Set Z: Not affected C: Not affected V: Not affected
Mode Bits	OscOff, CPUOff, and GIE are not affected.

*** SETZ**	Set zero bit
Syntax	SETZ
Operation	1 –> Z
Emulation	BIS #2,SR
Description	The zero bit (Z) is set.
Status Bits	N: Not affected Z: Set C: Not affected V: Not affected
Mode Bits	OscOff, CPUOff, and GIE are not affected.

SUB[.W] Subtract source from destination
SUB.B Subtract source from destination

Syntax

SUB src,dst or SUB.W src,dst
SUB.B src,dst

Operation

dst + .NOT.src + 1 –> dst
or
[(dst – src –> dst)]

Description

The source operand is subtracted from the destination operand by adding the source operand's 1s complement and the constant 1. The source operand is not affected. The previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the MSB of the result, reset otherwise.
Set to 1 if no borrow, reset if borrow.
V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits

OscOff, CPUOff, and GIE are not affected.

Example

See example at the SBC instruction.

Example

See example at the SBC.B instruction.

Note: Borrow Is Treated as a .NOT.

The borrow is treated as a .NOT. carry :

Borrow	Carry bit
Yes	0
No	1

SUBC[.W]SBB[.W] Subtract source and borrow/.NOT. carry from destination
SUBC.B,SBB.B Subtract source and borrow/.NOT. carry from destination

Syntax

SUBC	src,dst	or	SUBC.W	src,dst	or
SBB	src,dst	or	SBB.W	src,dst	
SUBC.B	src,dst	or	SBB.B	src,dst	

Operation

dst + .NOT.src + C –> dst
or
(dst – src – 1 + C –> dst)

Description

The source operand is subtracted from the destination operand by adding the source operand's 1s complement and the carry bit (C). The source operand is not affected. The previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive.
Z: Set if result is zero, reset otherwise.
C: Set if there is a carry from the MSB of the result, reset otherwise.
Set to 1 if no borrow, reset if borrow.
V: Set if an arithmetic overflow occurs, reset otherwise.

Mode Bits

OscOff, CPUOff, and GIE are not affected.

Example

Two floating point mantissas (24 bits) are subtracted.
LSBs are in R13 and R10, MSBs are in R12 and R9.

```
SUB.W    R13,R10   ; 16-bit part, LSBs
SUBC.B   R12,R9    ;  8-bit part, MSBs
```

Example

The 16-bit counter pointed to by R13 is subtracted from a 16-bit counter in R10 and R11(MSD).

```
SUB.B    @R13+,R10      ; Subtract LSDs without carry
SUBC.B   @R13,R11       ; Subtract MSDs with carry
...                     ; resulting from the LSDs
```

Note: Borrow Is Treated as a .NOT. Carry

The borrow is treated as a .NOT. carry :

Borrow	Carry bit
Yes	0
No	1

SWPB Swap bytes

Syntax SWPB dst

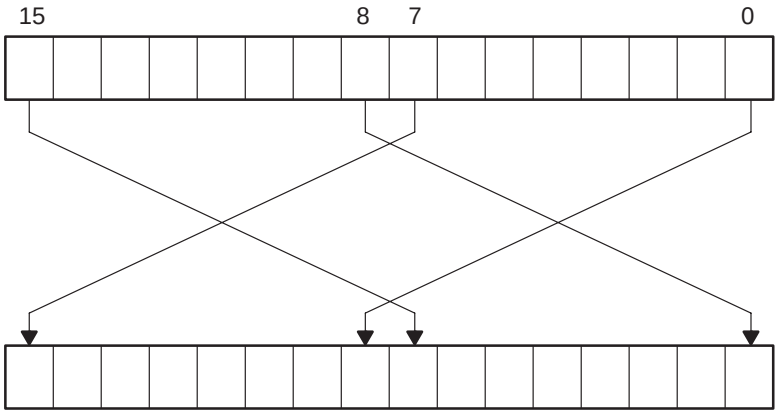
Operation Bits 15 to 8 <–> bits 7 to 0

Description The destination operand high and low bytes are exchanged as shown in Figure B–10.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

Mode Bits OscOff, CPUOff, and GIE are not affected.

Figure B–10. Destination Operand Byte Swap



Example

```
MOV     #040BFh,R7      ; 0100000010111111 –> R7
SWPB    R7              ; 1011111101000000 in R7
```

Example The value in R5 is multiplied by 256. The result is stored in R5,R4.

```
SWPB    R5              ;
MOV     R5,R4           ;Copy the swapped value to R4
BIC     #0FF00h,R5      ;Correct the result
BIC     #00FFh,R4       ;Correct the result
```

SXT Extend Sign

Syntax SXT dst

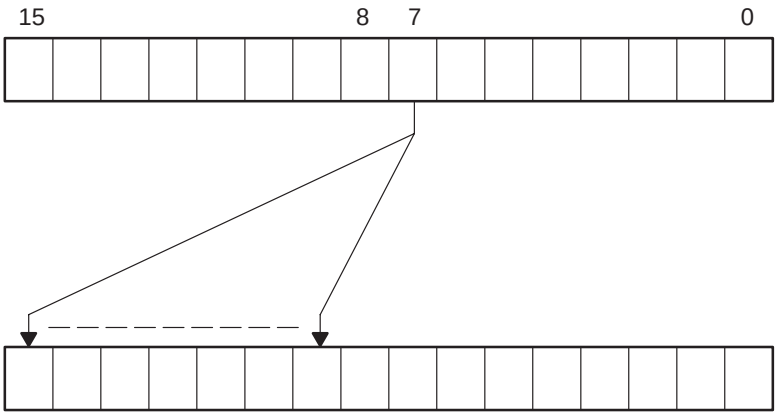
Operation Bit 7 –> Bit 8 ..... Bit 15

Description The sign of the low byte is extended into the high byte as shown in Figure B–11.

Status Bits
N: Set if result is negative, reset if positive
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (.NOT. Zero)
V: Reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Figure B–11. Destination Operand Sign Extension



Example R7 is loaded with the Timer/Counter value. The operation of the sign-extend instruction expands bit 8 to bit 15 with the value of bit 7.
R7 is then added to R6.

```
MOV.B   &TCDAT,R7   ; TCDAT = 080h:     . . . .  . . . . 1000 0000
SXT     R7          ; R7 = 0FF80h:      1111 1111 1000 0000
ADD     R7,R6       ; add value of EDE to 16-bit ACCU
```

*** TST[.W]** Test destination
*** TST.B** Test destination

Syntax

```
TST     dst    or   TST.W  dst
TST.B   dst
```

Operation

dst + 0FFFFh + 1
dst + 0FFh + 1

Emulation

```
CMP     #0,dst
CMP.B   #0,dst
```

Description The destination operand is compared with zero. The status bits are set according to the result. The destination is not affected.

Status Bits

N: Set if destination is negative, reset if positive
Z: Set if destination contains zero, reset otherwise
C: Set
V: Reset

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example R7 is tested. If it is negative, continue at R7NEG; if it is positive but not zero, continue at R7POS.

```
        TST     R7      ; Test R7
        JN      R7NEG   ; R7 is negative
        JZ      R7ZERO  ; R7 is zero
R7POS   .....           ; R7 is positive but not zero
R7NEG   .....           ; R7 is negative
R7ZERO  .....           ; R7 is zero
```

Example The low byte of R7 is tested. If it is negative, continue at R7NEG; if it is positive but not zero, continue at R7POS.

```
        TST.B   R7      ; Test low byte of R7
        JN      R7NEG   ; Low byte of R7 is negative
        JZ      R7ZERO  ; Low byte of R7 is zero
R7POS   .....           ; Low byte of R7 is positive but not zero
R7NEG   .....           ; Low byte of R7 is negative
R7ZERO  .....           ; Low byte of R7 is zero
```

XOR[.W] Exclusive OR of source with destination
XOR.B Exclusive OR of source with destination

Syntax

```
XOR     src,dst    or    XOR.W    src,dst
XOR.B   src,dst
```

Operation src .XOR. dst –> dst

Description The source and destination operands are exclusive ORed. The result is placed into the destination. The source operand is not affected.

Status Bits

N: Set if result MSB is set, reset if not set
Z: Set if result is zero, reset otherwise
C: Set if result is not zero, reset otherwise (= .NOT. Zero)
V: Set if both operands are negative

Mode Bits OscOff, CPUOff, and GIE are not affected.

Example The bits set in R6 toggle the bits in the RAM word TONI.

```
XOR     R6,TONI     ; Toggle bits of word TONI on the bits set in R6
```

Example The bits set in R6 toggle the bits in the RAM byte TONI.

```
XOR.B R6,TONI       ; Toggle bits in word TONI on bits
                    ; set in low byte of R6,
```

Example Reset to 0 those bits in low byte of R7 that are different from bits in RAM byte EDE.

```
XOR.B   EDE,R7      ; Set different bit to 1s
INV.B   R7          ; Invert Lowbyte, Highbyte is 0h
```

Appendix C

Flash Memory

This chapter describes the MSP430 flash memory module. The flash memory module is electrically erasable and programmable. Devices with a flash memory module are multiple-time programmable devices (MTP). They can be erased and programmed off-board, or in a system via the MSP430's JTAG peripheral module or via the processor's resources.

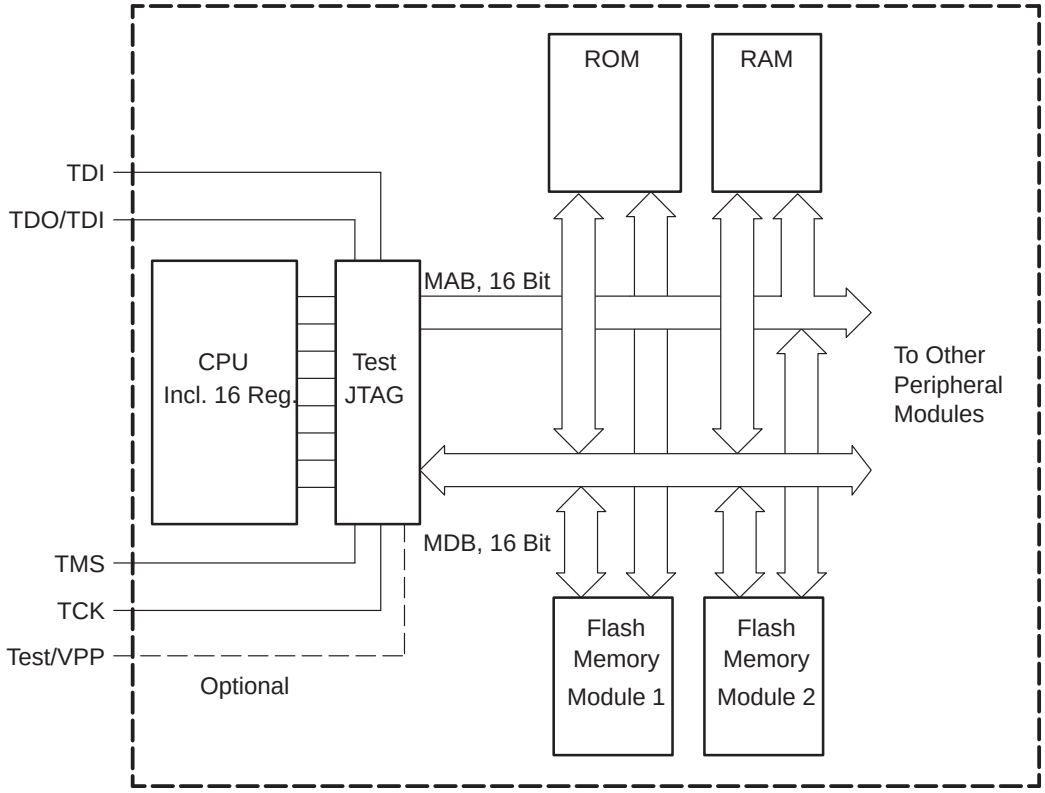
Software running on an MSP430 device can erase and program the flash memory module. This active software may run in RAM, in ROM, or in the flash memory. The flash memory may be a different memory module or the same memory module. The active software may not be in a memory location which is actively erased.

Topic **Page**

C.1 Flash Memory Organization

The flash memory may have one or more modules of different sizes as shown in Figure C–1. A module is a physical memory unit that operates independent from other modules. In an MSP430 configuration with more than one flash memory module, all modules are located in one linear-address range.

Figure C–1.Interconnection of Flash Memory Module(s)



Independent modules, such as Module1 and Module2, are intended to execute software code from one module while simultaneously programming or erasing another module.

Note: Flash Memory Module(s) in MSP430 Devices

Different devices may have one or more flash memory modules.

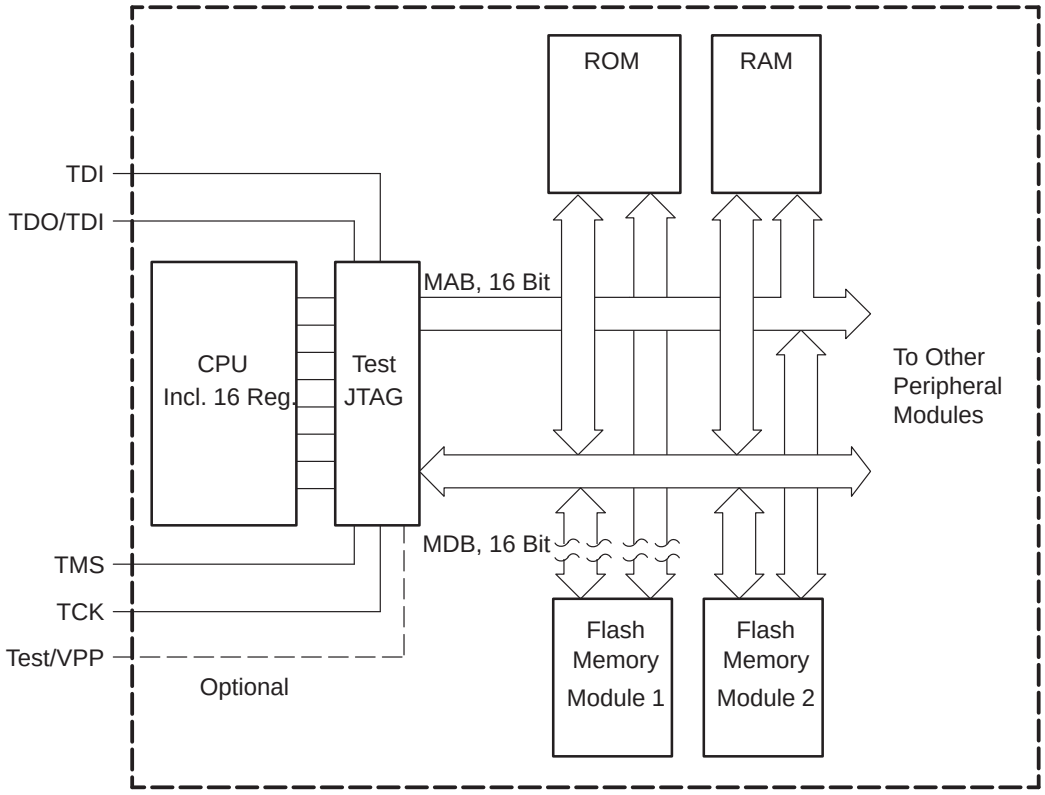
If the active software and the target programming location are in the same flash memory module, the program execution is halted (flag BUSY=1) until the programming cycle is completed (flag BUSY=0). Then it proceeds with the next instruction. The active software may also erase segments of the flash memory module. The user should be careful not to erase memory locations that are necessary to execute the software correctly.

A flash memory module, being programmed or erased, can not be accessed.

Figure C–2 shows the flash memory Module1 in program or erase operation. During this operation the module is disconnected from the memory address

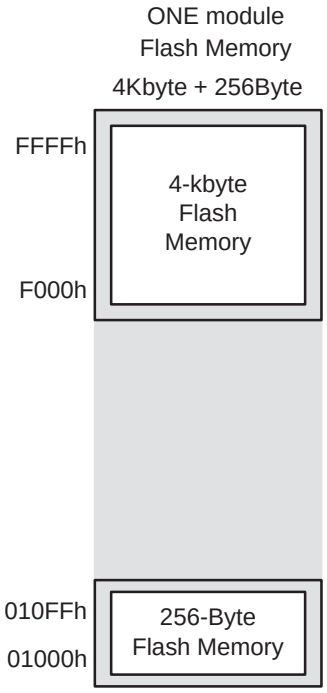
bus and memory data bus. When a second module (here Module2) is implemented, program code in this module can be executed while Module1 is disconnected.

Figure C–2. Flash Memory Module1 Disabled, Module2 Can Execute Code Simultaneously



One MSP430 flash memory module will have, in addition to its code segments, extra flash memory called *information memory*.

Figure C–3.Flash Memory Module Example



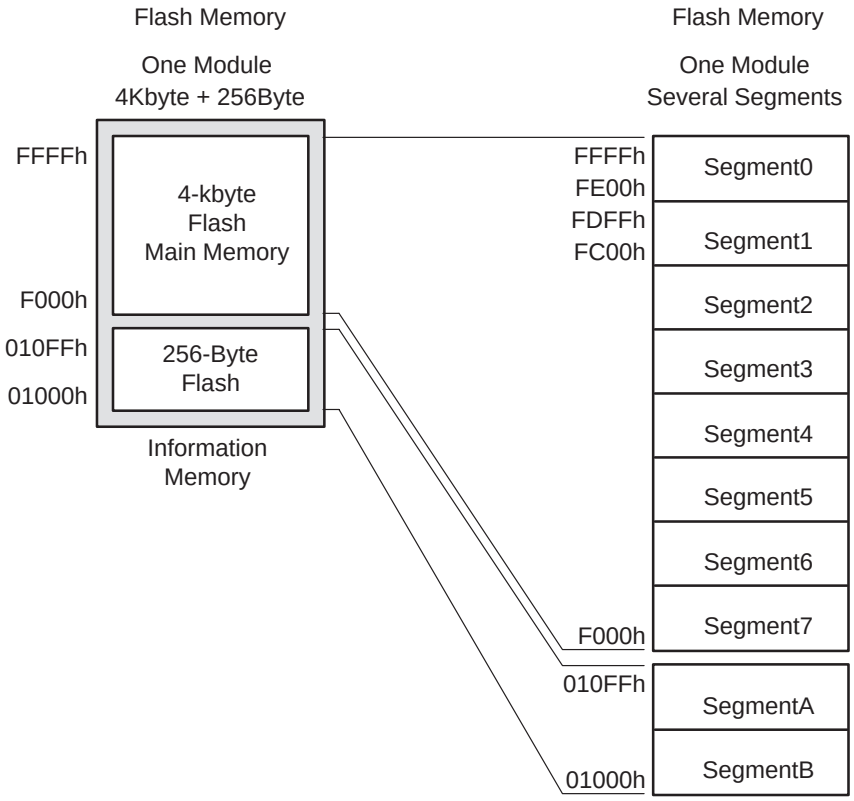
A module has several segments. The information memory has two segments of 128 bytes each. In the example in Figure C–4, the 4-kB module has eight segments of 512 bytes (Segment0 to Segment7), and two 128-byte segments (SegmentA and SegmentB). Segment0 to Segment7 can be erased individually or as a group. SegmentA and SegmentB can be erased individually or as a group with segments 0 to 7.

The segment structure is described in the device's data sheet. The information memory can be located directly below the main memory's address, or at a different address but will be in the same module.

Note:

Flash memory modules may have different numbers of segments. Segment are numbered from 0 up to n, e.g., segment 0 to segment n.

Figure C–4. Segments in Flash Memory Module, 4K-Byte Example



C.1.1 Why Is a Flash Memory Module Divided Into Several Segments?

Once a bit in flash memory has been programmed, it cannot be erased without erasing a whole segment. For this reason, the MSP430 flash memory modules have been heavily segmented to allow erasing and reprogramming of smaller memory segments.

C.2 Flash Memory Data Structure and Operation

The flash memory can be read and written (programmed) in bytes or words. Bits can be written as 0s once between erase cycles. The read access does not differ from access to masked ROM or RAM. Flash memory has restrictions in write operation:

- The default (erased) level for all bits is 1. Bits that are not programmed to 0s can be programmed to 0s at any time.
- The smallest memory portion to be erased is a segment. No single byte or word erase is possible.
- Access to a flash memory module is only possible when the module is not in a write or erase operation. For example, program code can not be executed in a module while it is processing a write or erase operation. The access limitation has no critical impact on program execution, but an access violation can be flagged in some situations (see flash memory register section in this chapter).

C.2.1 Flash Memory Basic Functions

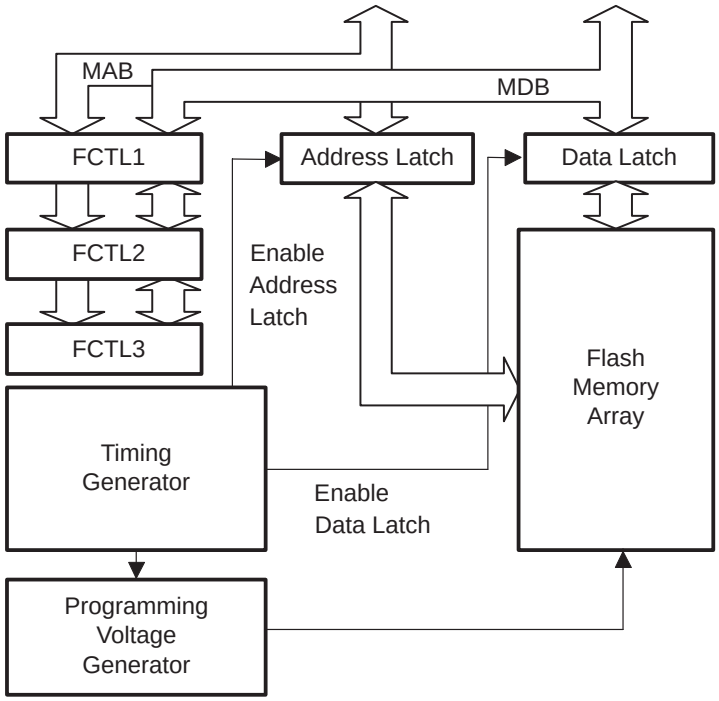
The basic functions of flash memory are to:

- Supply program code and data during program execution
- Erase, under software or JTAG control, parts of a module (one segment), multiple segments, or an entire module.
- Write data to a memory location under software or JTAG control. A double-speed programming sequence is implemented within a 64-byte section of the address range xx00h to xx3fh.

C.2.2 Flash Memory Block Diagram

The flash memory module has a minimum of three control registers, a timing generator, a voltage generator to supply program and erase voltages, and the flash memory itself. Data and address are latched when execution of a write (program) or erase operation is in progress.

Figure C–5.Flash Memory Module Block Diagram



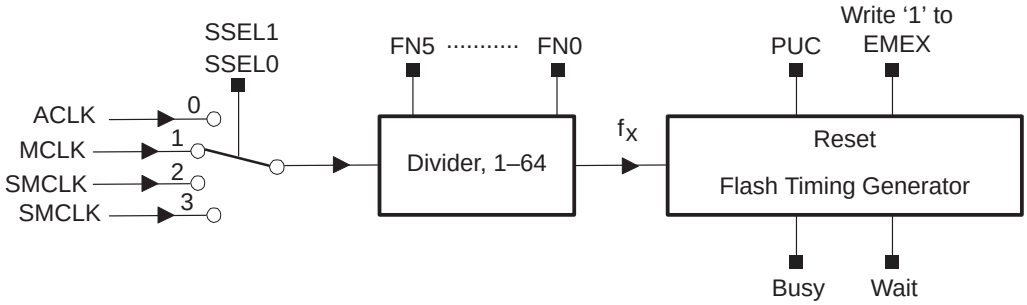
C.2.3 Flash Memory, Basic Operation

The flash memory module works in read mode most of the time, the address and data latch are transparent, and the timing generator and programming voltage generator are off. The flash memory module changes its mode of operation when data is written (programmed) to the module, or when the flash memory, or parts of it, are erased. In these situations, flash control registers FCTL1, FCTL2, and FCTL3 need to be set up properly to ensure correct write

or erase operation. Once these registers are set up and write or erase is started, the timing generator controls the entire operation and applies all signals internally. If the BUSY control signal is set, it indicates that the timing generator is active and a write or erase cycle is active. The segment write mode also uses a second control bit WAIT. There are three basic parts to a write or erase cycle: preparation of program/erase voltage, control timing for the program or erase operation, and the switch-off sequence of the program/erase voltage. Once a write or erase function is started, the software should not access the flash memory until the BUSY signal indicates, with *0*, that it can be accessed again. In critical situations where flash programming or erase should be immediately stopped, the *emergency exit* bit EMEX can be set. The current operation may be incomplete or the result may be incorrect.

Two different clock sources (ACLK, MCLK, or SMCLK) can be selected to clock the timing generator. The connected clock sources applied to the timing generator may vary with the device, see data sheet for details. The clock source selected should be active from the beginning of write or erase until the operation is fully completed.

Figure C–6.Block Diagram of the Timing Generator in the Flash Memory Module



The selected clock source should be divided to meet the frequency requirement f_x of the flash timing generator.

If the clock signals are not available throughout the duration of the write or erase operation, or their frequencies change drastically, the result of the write or erase may be marginal, or the flash memory module may be stressed above the limits of reliable operation.

Table C–1 shows all useful combinations of control bits for proper write and erase operation:

Table C–1. Control Bits for Write or Erase Operation

FUNCTION PERFORMED	SEGWRT	WRT	Meras	Erase	BUSY	WAIT	Lock
Write word or byte	0	1	0	0	0	0	0
Write word or byte in same segment, segment write mode	1	1	0	0	0	1	0
Erase one segment by writing to any address in the target segment (0 to n or A or B)	0	0	0	1	0	0	0
Erase all segments (0 to n) but not the information memory (SegmentA and SegmentB)	0	0	1	0	0	0	0
Erase all segments (0 to n and A and B) by writing to any address in the flash memory module	0	0	1	1	0	0	0

Note: A write to flash memory performed with any other combination of bits SEGWRT, WRT, Meras, Eras, BUSY, WAIT, and Lock will result in an access violation. ACCVIFG is set and an NMI is requested if ACCVIE=1.

C.2.4 Flash Memory Status During Code Execution

The flash memory module delivers data for code execution in the same manner as any masked ROM or RAM. The flash memory module should be in read mode, with no write (programming) or erase operation active. By default, power-on reset (POR) puts the flash memory into read mode. No control bits need to be defined in the flash memory control registers after POR for code execution.

C.2.5 Flash Memory Status During Erase

The default bit level of the flash memory is 1. Any successful erase sets all bits of a segment or a block to this default level. Once a bit is programmed to the 0-level, only the erase function can reset it back to 1. Erase can be performed for one segment, a group of segments, or for an entire module. This can vary for each device configuration, and the exact implementation should be noted in the data sheet.

The erase operation starts with the following sequence:

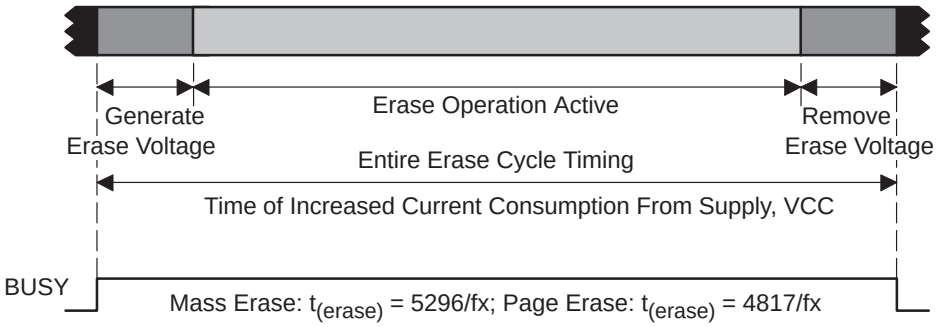
1) Set the correct input-clock frequency of the timing generator by selecting the clock source and predivider.
2) Reset the LOCK control bit, if set.
3) Watch the BUSY bit. Continue to the next steps only if the BUSY bit is reset.
4) Set the erase control bit *Erase* to erase a segment, or
5) Set the *mass-erase* control bit MEras to erase a group of segments or
6) Set the *mass-erase* (MEras) and erase (Erase) control bits to erase the entire flash memory
7) Execute a dummy write to any address in the range to be erased.

The *dummy write* starts the erase cycle. An example of dummy write is *CLR &0F012h*.

Note that a dummy write is ignored in a segment where the selected operation can not be executed successfully.

An example of such a situation can take place when Segment 1 is to be erased: the control bits are set properly, but the *dummy write* is sent to the information memory. No flag indicates this unsuccessful erase situation.

Figure C–7.Basic Flash EEPROM Module Timing During the Erase Cycle



The erase cycle completes successfully when none of the following restrictions is violated:

- The selected clock source is available until the cycle is completed.
- The predivider should not be modified during the operation.
- No further access to the flash memory module is performed while BUSY is set.
  - No read of data from this block
  - No write into this block
  - No further erase of this block

  An access will result in setting the KEYV bit and requesting an NMI interrupt. The NMI interrupt routine should handle such violations.
- The supply voltage should be within the devices' electrical specifications defined in the respective data sheet; however, slight variations can be tolerated.

Control bit BUSY indicates an active erase cycle. It is set immediately after a dummy write starts the timing generator. It remains set until the entire erase cycle is completed and the erased segment or block is ready to be accessed again. The BUSY bit can not be set by software. But it can be reset. In case of emergency, set the emergency exit (EMEX) bit and the erase operation will be stopped immediately; BUSY bit is reset. One example of stop erase by software is when the supply voltage drops drastically and the operating conditions of the controller are exceeded. Another example is when the timing of the erase cycle gets out of control, for example, when the clock-source signal is lost.

Note:

When the erase cycle is stopped before its normal completion by the hardware, the timing generator is stopped and erasure of the flash memory can be marginal. An incomplete erasure can be verified. But an erase level of 1 can be inconsistently read as valid when supply voltage, temperature, access time (instruction execution, data read), and frequency vary.

C.2.6 Flash Memory Status During Write (Programming)

The flash memory erase bit level is 1. Bits can only be written (programmed) to a 0-level. Once a bit is programmed, only the erase function can reset it back to the 1-level. The byte or word 0-level can not be written (programmed) in one cycle. Any bit can be programmed from 1 to 0 at any time, but not from 0 to 1.

Two slightly different write operations can be performed: write a single byte or word of data, or write a sequence of bytes or words. A write sequence of bytes or words can be performed as multiple sequential, or as a segment write. The segment write is approximately twice as fast as a multiple sequential write algorithm.

The write (program) operation starts with the following sequence:

- ❑ Set the correct input clock frequency of the timing generator by selecting the clock source and predivider.
- ❑ Reset the LOCK control bit, if set.
- ❑ Watch the BUSY bit. Continue with the next steps only if the BUSY bit is reset.
- ❑ Set the write-control bit WRT when a single byte of word data is to be written.
- ❑ Set the write WRT and SEGWRT control bits when segment write is chosen to write multiple bytes or words to the flash memory module.
- ❑ Writing the data to the selected address starts the timing generator. The data is written (programmed) while the timing generator proceeds.

Note:

Whenever the write cycle is stopped before its normal ending by the hardware, the timing generator is stopped and the data written to the flash memory can be marginal. The data may be incorrect, which can be verified, or the data are verified to be correct but the programming is marginal. Reading of the data may be inconsistently valid when varying the supply voltage, the temperature, the access time (instruction execution, data read), or the time.

Figure C–8.Basic Flash Memory Module Timing During Write (Single Byte or Word) Cycle

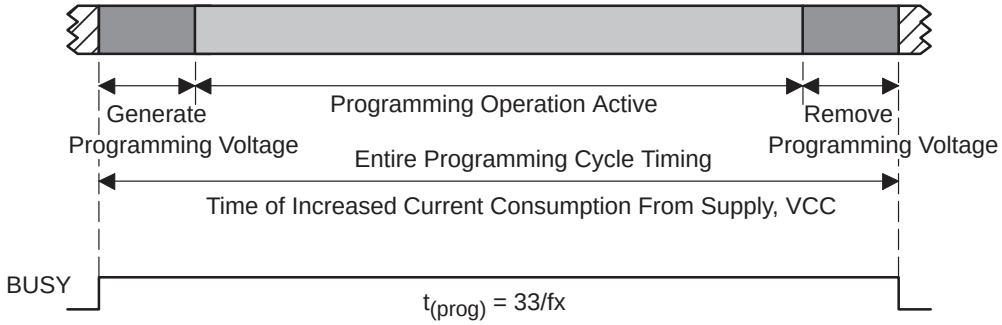
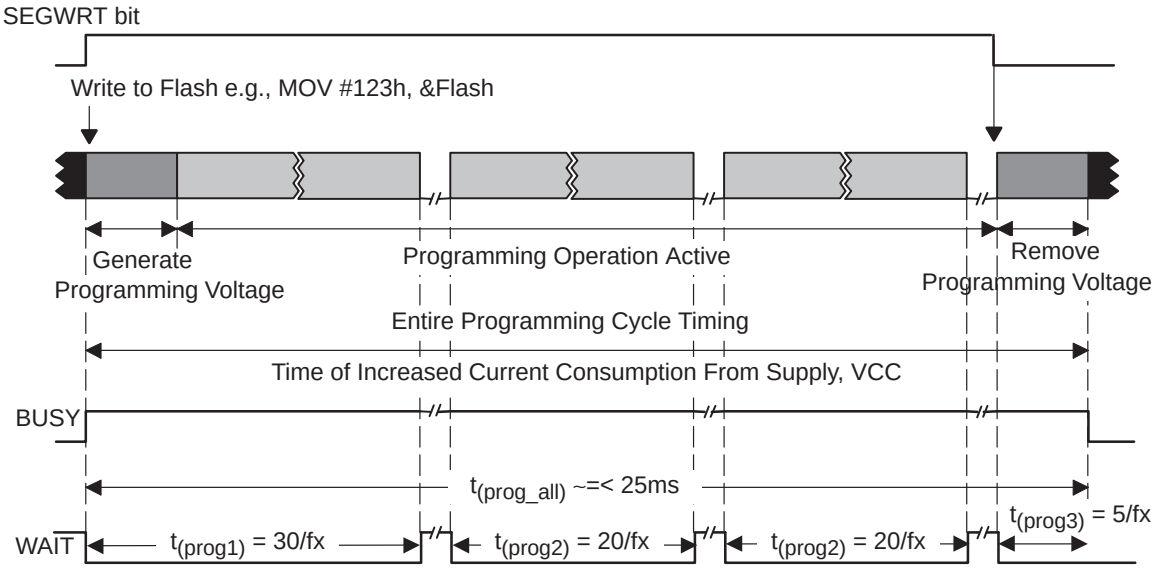


Figure C–9.Basic Flash Memory Module Timing During a Segment-Write Cycle



The segment write can be used on sequential addresses of the memory module. One segment is 64 bytes long, starting at 0xx00h, 0xx40h, 0xx80h, or 0xxC0h, and ending at 0xx3Fh, 0xx7Fh, 0xxBFh, or 0xxFFh. Examples of sequential segment addresses are:

```
0F000h to 0F03Fh, 0F040h to 0F07Fh,0F080h to 0F0BFh,
0F0C0h to 0F0FFh, 0F100 to 0F13Fh,
```

The segment-write (program) operation at the 64-byte boundaries needs special software support (test of address 0xx3Fh, 0xx7Fh, 0xxBFh, or 0xxFFh was successful):

- ❑ Wait until the WAIT bit is set, indicating that the write of the last byte or word was completed.
- ❑ Reset control bit SEGWRT.
- ❑ The BUSY bit remains set until the programming voltage is removed from the flash memory module and overstress is avoided.
- ❑ Wait the recovery time $t_{(rcv)}$ before another segment write is started.

The write cycle is successfully completed if none of the following restrictions is violated:

- ❑ The selected clock source is available until the cycle is completed.
- ❑ The predivider is not modified.
- ❑ The access to the flash memory module is restricted as long as BUSY is set.

The conditions to read data from the flash memory with and without access violation are listed in Table C–2.

Table C–2. Conditions to Read Data From Flash Memory

Flash Operation	Instruction Fetch (see Note 1)	BUSY	WAIT	Data on Memory Data Bus (MDB)	Action
Byte/word program cycle (see Note 2)	No	1	0	3FFF	Access violation
	Yes	1	0	3FFF → JMP $	Nothing
Flash read mode		0	0	Memory contents from applied address	PC = PC + 2
Page erase cycle (see Note 3)	No	1	0	3FFF	Access violation
	Yes	1	0	3FFF → JMP $	Nothing
Mass-erase cycle (see Note 3)	No	1	0	3FFF	Access violation
	Yes	1	0	3FFF → JMP $	Nothing
All erase (mass and information memory)	No	1	0	3FFF	Access violation
	Yes	1	0	3FFF → JMP $	Nothing
Segment write (see Note 4)	N.A.	1	0	3FFF	Access violation and LOCK (see Note 5)
	No	1	1	3FFF	Nothing
	Yes	1	1	3FFF	Access violation and LOCK (see Note 5)

Notes:
1) Instruction fetch refers to the fetch part of an instruction, and reads one word. The instruction fetch reads the first word of instructions with more than one word. The JMP instruction has one word. The data fetched (3FFFh) is used by the CPU as an instruction.
2) Ensure that the programmed data does not result in unpredictable program execution, such as destruction of executable code sequences.
3) If the PC points to the memory location being erased, no access violation indicates this situation. After erase, no executable code is available and an unpredictable situation occurs.
4) Any software located in a flash memory module can not use the SEGWRT mode to program the same flash memory module. Using the byte or word programming mode allows programming data in the flash memory module holding the software code currently executing.
5) The access violation sets the LOCK bit to 1. Setting the LOCK bit allows completion of the active segment write operation in the normal manner.

- The supply voltage should be within the devices' electrical conditions and can only vary slightly, as specified in the applicable data sheet

The control bit BUSY indicates that the write or segment-write cycle is active. It is set by the instruction that writes data to the flash memory module and starts the timing generator. It remains set until the write cycle is completed and the programming voltage is removed. In the write mode the BUSY bit indicates if the flash memory is ready for another write operation. In segment write mode the WAIT bit indicates if the flash memory is ready for another write operation and the BUSY bit indicates the segment write operation is completed. In case of emergency, the emergency exit bit EMEX is set and stops the write cycle immediately. The programming voltage is switched off. One situation where the write cycle should be stopped by software is when the supply voltage drops drastically and the controller's operating conditions may be exceeded. Another case is when the flash memory timing gets out of control, as when the clock-source signal is lost.

Note:

Whenever the write cycle is stopped before its normal ending by the hardware, the timing generator is stopped and the data written in flash memory may be marginal. Data reading may be inconsistently valid when varying the supply voltage, the temperature, the access time (instruction execution, data read), or the time.

C.3 Flash Memory Control Registers

Defining the correct control bits of three control registers enables write (program), erase, or mass-erase. All three registers should be accessed using word instructions only. The control registers are protected against false write or erase cycles via a key word. Any violation of this keyword sets the KEYV bit and requests a nonmaskable interrupt (NMI). The keyword is different to the keyword used with the Watchdog Timer.

All control bits are reset during PUC. PUC is activated after V_{CC} is applied, a reset condition is applied to the RST/NMI pin or watchdog, or a flash operation was not performed normally.

C.3.1 Flash Memory Control Register FCTL1

Any write to control register FCTL1 during erase, mass-erase, or write (programming) will end in an access violation with ACCVIFG=1. In an active segment-write mode, the control register can be written if wait mode is active (WAIT=1). In an active segment write mode and while WAIT=0, writing to control register FCTL1 will also end in an access violation with ACCVIFG=1.

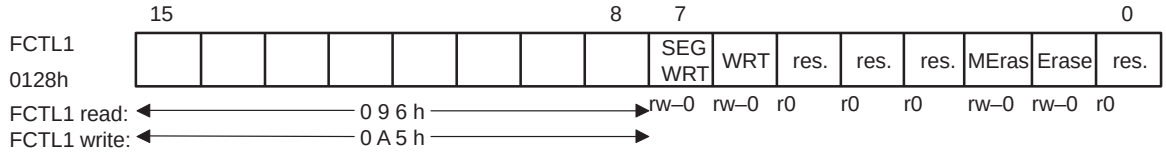
Read access is possible at any time without restrictions.

Any write to control register FCTL1 during erase, mass-erase, or write (programming) will end in an access violation with ACCVIFG=1. In an active seg-

ment write mode, the control register can be written if wait mode is active (WAIT=1). In an active segment write mode and while WAIT=0, writing to control register FCTL1 will also end in an access violation with ACCVIFG=1.

Read access is possible at any time without restrictions.

The control bits of control register FCTL1 are:



Erase 0128h, bit1, Erase a segment

0: No segment erase is started.

1: Erase of one segment is enabled. The segment n to be erased is defined by a *dummy* write into any address within the segment. The Erase bit is automatically reset when the erase operation is completed.

Note: Instruction fetch access during erase is allowed. Any other access to the flash memory during erase results in setting the ACCVIFG bit, and an NMI interrupt is requested. The NMI interrupt routine should handle such violations.

MEras 0128h, bit2, Mass-erase, Segment0 to Segmentn are erased together.

0: No erase is started

1: Erase of Segment0 to Segmentn is enabled. When a *dummy* write into any address in Segment0 to Segmentn is executed, mass-erase is started. The MEras bit is automatically reset when the erase operation is completed.

Note: Instruction fetch access during mass-erase is allowed. Any other access to the flash memory during erase results in setting the ACCVIFG bit, and an NMI interrupt is requested. The NMI interrupt routine should handle such violations.

WRT 0128h, bit6, The bit WRT should be set to get a successful write execution.

If bit WRT is reset and write access to the flash memory is performed, an access violation occurs and ACCVIFG is set.

Note: Instruction fetch access during erase is allowed. Any other access to the flash memory during erase results in setting the ACCVIFG bit, and an NMI interrupt is requested. The NMI interrupt routine should handle such violations.

SEGWRT 0128h, bit7, Bit SEGWRT can be used to reduce total programming time.

The segment-write bit SEGWRT is useful if larger sequences of data have to be programmed. If programming of one segment is completed, a reset and set sequence should be performed to enable access to the next segment. The WAIT bit should be high before the next write instruction is executed. See also paragraph C.1.1 and Figure C–9.

0: No segment write accelerate is selected.

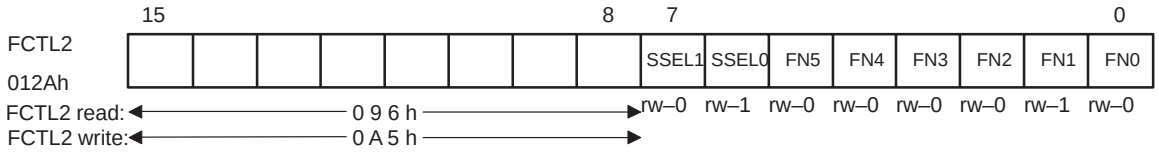
1: Segment write is used. This bit needs to be reset and set between segment borders.

C.3.2 Flash Memory Control Register FCTL2

A PUC resets the flash timing generator. The generator is also reset if the emergency exit bit EMEX is set.

The timing generator generates the timing necessary to write, erase, and mass-erase from a selected clock source. Two control bits SSEL0 and SSEL1 in control register FCTL2 can select one of three clock sources. The clock source selected should be divided to meet the frequency requirements for f_X, as specified in the device's data sheet.

Writing to control register FCTL2 should not be attempted if the BUSY bit is set; otherwise an access violation will occur (ACCVIFG=1). Read access to FCTL2 is possible at any time without restrictions.

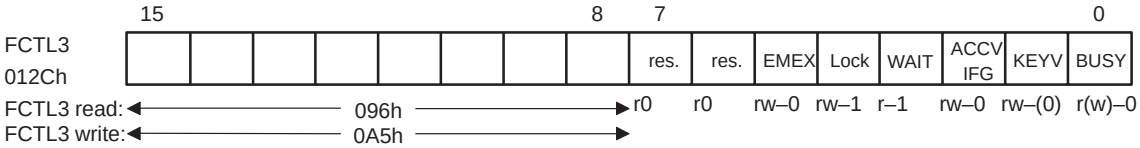


The control bits are:

FN0 to	012Ah, bit0,	These six bits define the division rate of the clock signal. The division rate can be 1 to 64, depending on the digital value of FN5 to FN0
FN5	012Ah, bit5,	plus one
SSEL0	012Ah, bit0,	Determine the clock source.
SSEL1		0: ACLK
		1: MCLK
		2,3 SMCLK

C.3.3 Flash Memory Control Register FCTL3

There are no restrictions on modifying this control register. The control bits are reset or set (WAIT) by a PUC, but key violation bit KEYV is reset by POR.



BUSY 0128h, bit0, The bit BUSY shows if an access to the flash memory is possible (BUSY=0), or if an access violation can occur. The BUSY bit is read only, but a write operation is allowed. The BUSY bit should be tested before each write and erase cycle. The flash-timing generator hardware immediately sets the BUSY bit after the start of a write operation, a segment-write operation, a segment erase, or a mass-erase. Once the timing generator has completed its function, the BUSY bit is reset by hardware.

The program and erase timing are shown in Figures C–7, C–8, and C–9.

0: Flash memory is not busy. Read, write, erase and mass-erase are possible without any violation of the internal flash timing. The BUSY bit is reset by POR and by the flash timing generator.

1: Flash memory is busy. Remains in busy state if segment write function is in *wait* mode.

The conditions for access to the flash memory during BUSY=1 are described in paragraph C.2.6.

KEYV, 012Ch, bit1, Key Violated.

0: Key 0A5h (high byte) was not violated.

1: Key 0A5h (high byte) was violated. Violation occurs when a write access to register FCTL1, FCTL2 or FCTL3 is executed and the *high byte* is not equal to 0A5h. If the security key is violated, bit KEYV is set and a PUC is performed. The KEYV bit can be used to determine the source that forced a start of the program at the reset vector's address. The KEYV bit is not automatically reset and should reset by software.

Note: Any key violation results in a PUC, independent of the state of the KEYV bit. To avoid endless software loops, the flash memory control registers should not be written during a key violation service routine.

Note: The software can set the KEYV bit. A PUC is also performed if it is set by software.

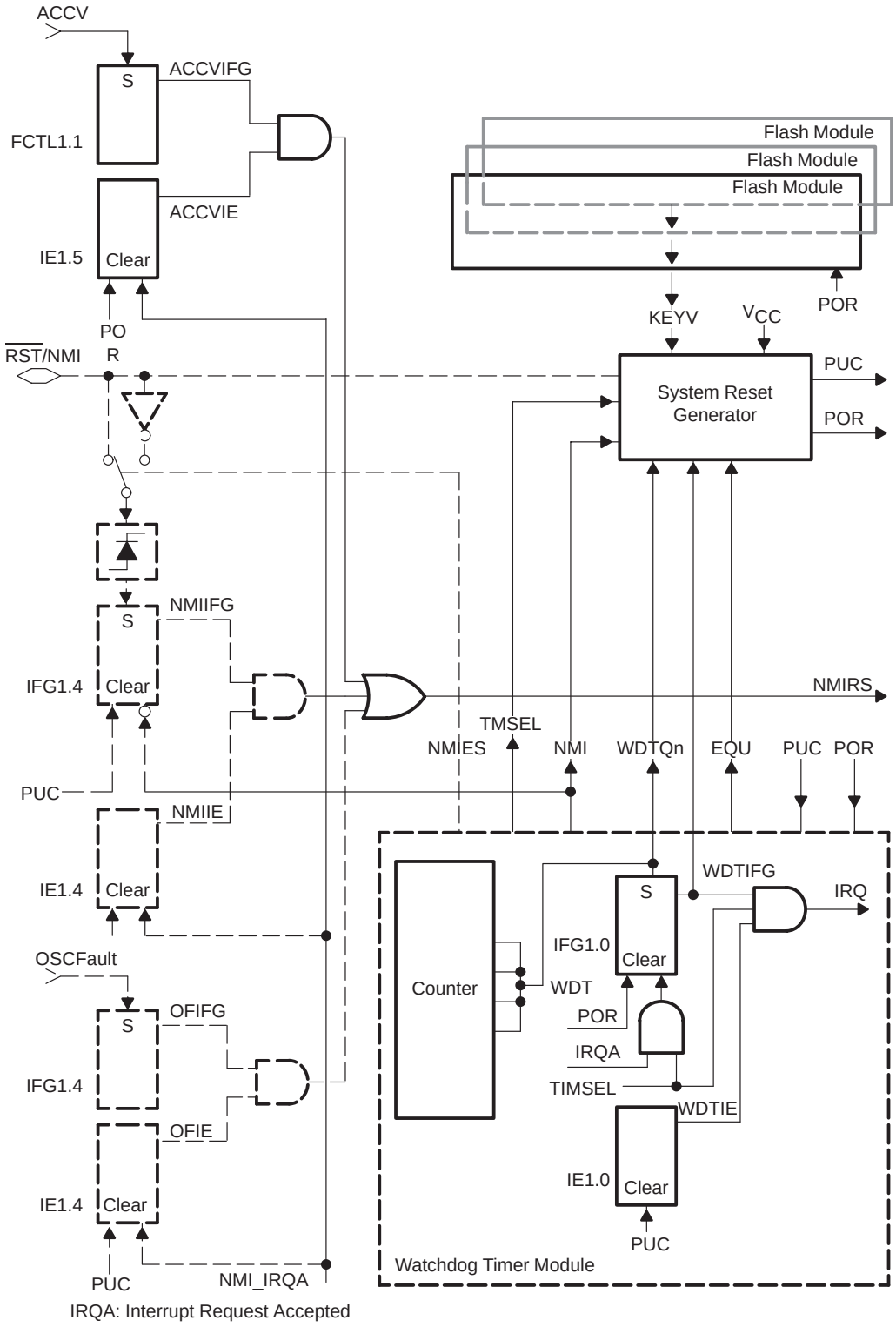
ACCVIFG	bit2,	Access violation interrupt flag The access-violation interrupt flag is set when the flash memory module is improperly accessed while a write or erase operation is active. The violation situations are described in section C.2. When the access-violation interrupt-enable bit is set, the interrupt-service request is accepted and the program continues at the NMI interrupt-vector address. Reading the control registers will not set the ACCVIFG bit. Note: The proper interrupt-enable bit ACCVIE is located in interrupt-enable register IE1 of the special-function register. Software can set the ACCVIFG bit; in this case, an NMI is also executed.
WAIT	012Ch, bit3,	Wait. In the segment write mode the WAIT bit indicates that the flash memory is ready to receive the (next) data for programming. The WAIT bit is read only, but a write to the WAIT bit is allowed. The WAIT bit is automatically reset if the SEGWRT bit is reset or the LOCK bit is set. Segment-write operation is completed, and then the WAIT bit returns to 1. Condition, SEGWRT=1 (see Figure C-9): After each successful write operation, the BUSY bit is reset to indicate that another byte or word can be written (programmed). The BUSY bit does not indicate the condition when the timing generator has completed the entire programming. The high-voltage portion and voltage generator remain active. The maximum time $t_{(CPT)}$ should not be violated. 0: Segment-write operation has started and programming is in progress. 1: Segment-write operation is active and programming of data is completed. Waiting for the next data to be programmed.
Lock	012Ch, bit4,	The Lock bit can be set during any write, erase of a segment, or mass-erase request. The active sequence is completed normally. In segment-write mode, if the Lock bit is set and SEGWRT and WAIT are set, the SEGWRT and WAIT bits are reset and the mode ends normally. The WAIT bit is 1 after segment-write mode has ended. Software or hardware can control the Lock bit. If an access violation occurs (see conditions described in paragraph C.1.1), the ACCVIFG and the Lock bit are set. 0: Flash memory can be read, programmed, erased, and mass-erased. 0: Flash memory can be read but not programmed, erased, or mass-erased. A current program, erase, or mass-erase operation is completed normally. The access-violation interrupt flag ACCVIFG is set when the flash memory module is accessed while the Lock bit is set.

EMEX	012Ch, bit5,	Emergency exit. The emergency exit should only be used when a flash memory write or erase operation is out-of-control. 0: No function. 0: Stops the active operation immediately and shuts down all internal parts of the flash memory controller. Current consumption immediately drops back to the active mode. All bits in control register FCTL1 are reset. Since the EMEX bit is automatically reset by hardware, the software always reads EMEX as 0.

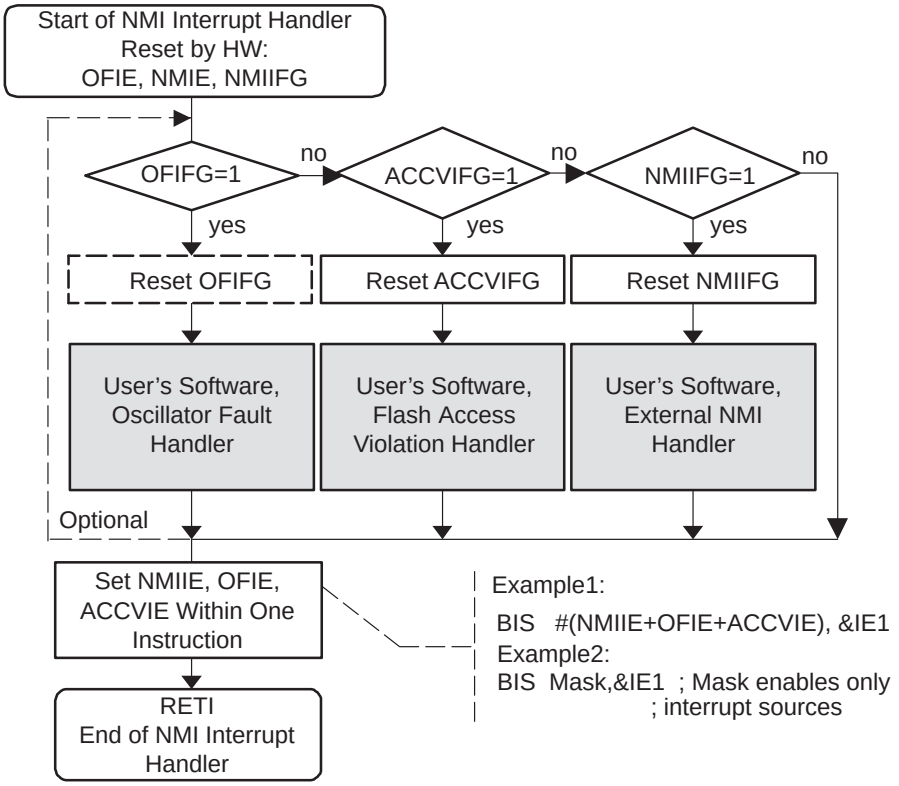
C.4 Flash Memory, Interrupt and Security Key Violation

One NMI vector is used for three non-maskable interrupt (NMI) events, RST/NMI, oscillator fault (OFIFG), and flash-access violation (ACCVIFG). The software can determine the source of the interrupt request by testing interrupt flags NMIIFG, OFIFG, and ACCVIFG. They remain set until reset by software.

Figure C–10. Basic Flash Memory Module Timing During Segment Write Cycle



C.4.1 Example of an NMI Interrupt Handler



The NMI handler takes care of all sources requesting a nonmaskable interrupt. The NMI interrupt is a multiple-source interrupt per MSP430 definition. The hardware resets the interrupt-enable flags: the external nonmaskable interrupt enable NMIIE, the oscillator fault interrupt enable OFIE, and the flash memory access-violation interrupt enable. The individual software handlers reset the interrupt flags and reenables the interrupt enable bits according to the application needs. After all software is processed, the interrupt enable bits have to be set if another NMI event is to be accepted. Setting the interrupt enable bits should be the last instruction before the return-from-interrupt instruction RETI. If this rule is violated, the stack can grow out of control while other NMI requests are already pending. Setting the interrupt enable bits can be accomplished by using a bit-set-instruction BIS using immediate data or a mask. The mask data can be modified anywhere via software (for example in RAM); this constitutes the nonmaskable interrupt processing.

C.4.2 Protecting One-Flash Memory-Module Systems From Corruption

MSP430 configurations having one flash memory module use this module for program code and interrupt vectors. When the flash memory module is in a write, erase, or mass-erase operation and the program accesses it, an access violation occurs. This violation will request an interrupt service – but when the interrupt vector is read from the flash memory, 03FFFh will be read independent of the data in the flash memory at the vector's memory location.

To protect the software from this error situation, all interrupt sources have to be disabled since all interrupt requests will fail. The flash memory returns the vector 03FFFh. Before the interrupt enable bits are modified, they can be stored in RAM to be restored when the flash memory is ready for access again.

The following interrupt enable bits should be reset to stop all interrupt service requests:

- GIE = 0
- NMIIE = ACCVIE = OFIE = 0

Additionally the watchdog should be halted to prevent its expiration when flash memory is busy:

- WDTHOLD = 1

When the flash memory is ready, the interrupt sources can be enabled again. Before they are enabled, critical interrupt flags should be checked and, if necessary, served or reset by software.

- GIE = 1 or left disabled, or be restored to the previous level
- NMIIE = ACCVIE = OFIE = 1 or left disabled, or be restored to the previous level
- WDTHOLD = 0 or left disabled, or be restored to the previous level

C.5 Flash Memory Access via JTAG and Software

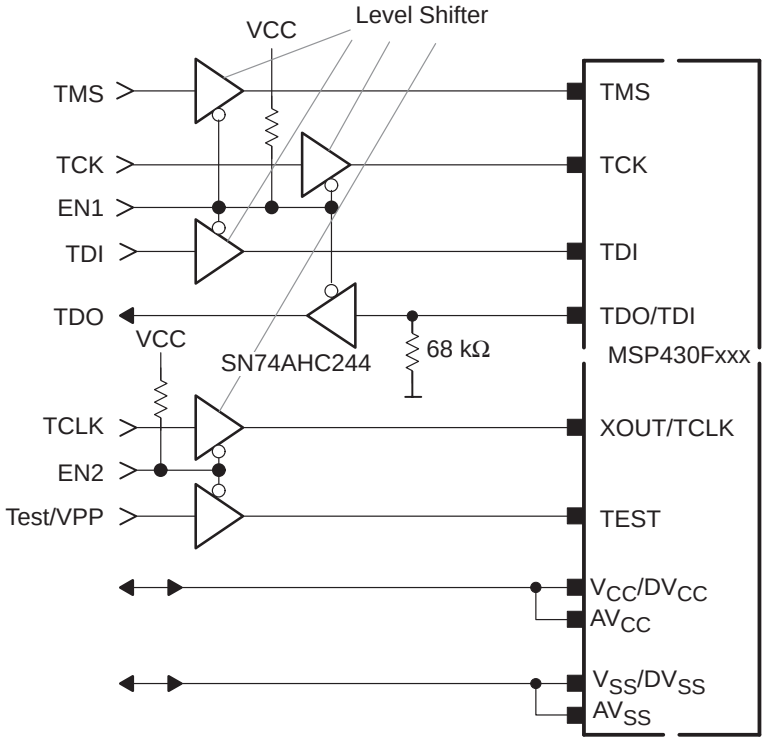
C.5.1 Flash Memory Protection

Flash memory access via the serial test and programming interface JTAG can be inhibited when the *security fuse* is activated. The *security fuse* is activated via serial instructions shifted into the JTAG. Activating the fuse is not reversible, and any access to the internal system is disrupted. The bypass function described in the IEEE1149.1 standard is active.

C.5.2 Program Flash Memory Module via Serial Data Link Using JTAG Feature

The hardware interconnection to the JTAG pins is done via four separate pins, plus the ground or V_{SS} reference level. The JTAG pins are TMS, TCK, TDI (/VPP), and TDO (/TDI).

Figure C–11. Signal Connections to MSP430 JTAG Pins

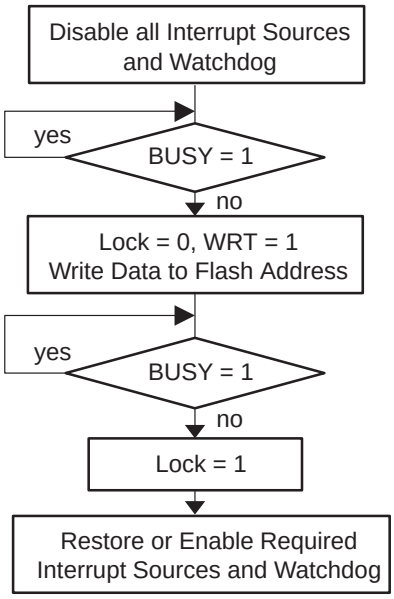


C.5.3 Programming a Flash Memory Module via Controller Software

No special external hardware is required to program a flash memory module. The power supply at pin V_{CC} should supply the higher current during write (program) and erase modes. The software algorithm is simple. The embedded timing generator in the flash memory module controls the program and erase cycles. Software can not run in the same flash memory module where data is to be written. Such background software needs to run on other memory device, such as a ROM module, a RAM module, or another flash memory module.

C.5.3.1 Example: Programming One Word Into a Flash Memory Module via Software Execution Outside this Module

This example assumes that the code to program the flash location is not executed from the target flash memory module.




```
Fxkey   .set    03300h
Fwkey   .set    0A500h
; No interrupt request may happen while the flash is programmed

Test_Busy1
        BIT     #BUSY,&FCTL3
        JNZ     Test_Busy1

        MOV     #Fwkey,&FCTL3           : Clear lock bit
        MOV     #(Fwkey+WRT),&FCTL1:    : Enable write to flash
        MOV     #123h,&0FF1Eh           : Write a word to flash

Test_Busy2
        BIT     #BUSY,&FCTL3            ; still busy?
        JNZ     Test_Busy2              ; yes, repeat busy test
        MOV     #Fwkey,&FCTL1           : Reset write bit
        XOR     #(Fxkey+Lock),&FCTL3    : Change lock bit to 1

; Enable those interrupt sources that should be accepted
```

The BUSY bit can be tested before the write to the flash memory module is done, or after a write (program) starts:

- For flash memory locations that hold data, it is a good practice to test the BUSY bit before the write is executed. This has some time benefits, since the write process is executed via the flash memory timing generator without further CPU intervention. It is important that the clock source remains active until BUSY is reset by the flash memory hardware.

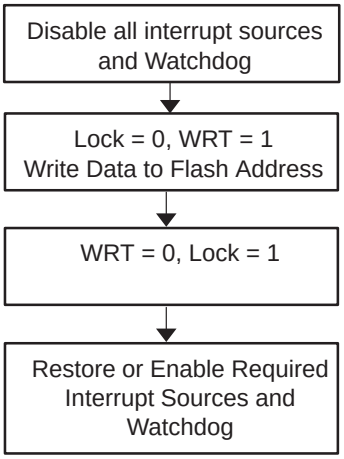
  The power or clock management, responsible for entering low-power modes, has to make sure that it does not switch off the clock source used by the flash controller.

- For flash memory blocks that hold program code, it is a good practice to test the BUSY bit after the write is executed. The program can only proceed if the module can be accessed again. No special attention is needed during execution of software code. Every write to the flash memory module has to leave the programming cycle with the BUSY bit reset.

  Testing the BUSY bit before writing to a flash memory block that holds program code ensures that the active program will not access the flash memory module. Two types of access are visible: execute program code, or read and write data on this flash memory module.

C.5.3.2 Example: Programming One Word Into the Same Flash Memory Module via Software

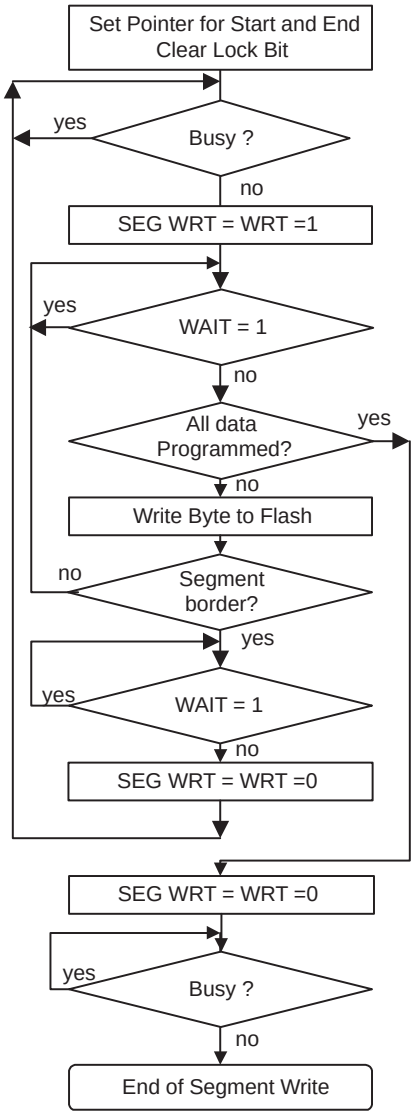
The program execution waits after the write-to-flash instruction (MOV #123h,&0FF1Eh) until the busy bit is reset again. If no other write-to-flash instruction method is used the BUSY bit test may not be needed to ensure correct flash-write handling.




```
Fxkey   .set    03300h
Fwkey   .set    0A500h

; No interrupt request may happen while the flash is programmed

        MOV     #(Fwkey+WRT),&FCTL1:  : Enable Write to flash
        MOV     #123h,&0FF1Eh         : Write a word to flash

        MOV     #Fwkey,&FCTL1         : Reset Write bit
        XOR     #(Fxkey+Lock),&FCTL3  : Change Lock bit to 1

; Enable those interrupt sources that should be accepted
```

C.5.3.3 Example, Programming Byte Sequences Into a Flash Memory Module via Software

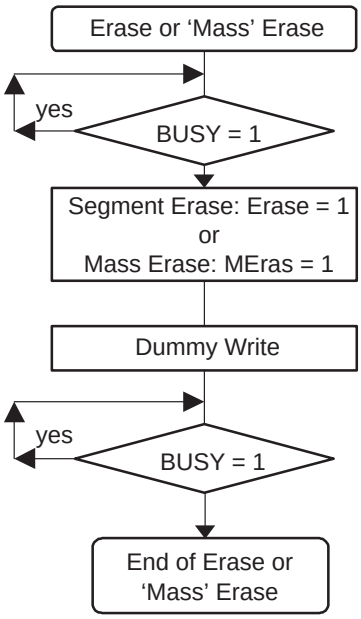
Sequences of data, bytes, or words can use the segment-write feature. This reduces the programming time by about one half.




```
Fxkey   .set   03300h
Fwkey   .set   0A500h
Frkey   .set   09600h

; Ensure that neither Watchdog Timer, nor
; interrupts nor Low–Power Modes may corrupt
; proper execution

RAM2FLASH
        MOV    #Start_Ptr,Rx
        MOV    #End_Ptr,Ry
        MOV    #Fkey,&FCTL3                ; Clear lock bit
Test_Busy1
        BIT    #BUSY,&FCTL3                ; Flash busy? SEGWRT ended?
        JNZ    Test_Busy1

        MOV    #(Fwkey+WRT+SEGWRT),&FCTL1  ; Segment write

Test_WAIT1
        BIT    #WAIT,&FCTL3
        JNE    Test_WAIT1

        CMP    Rx,Ry                       ; All data programmed?
        JZ     End_Seg_Write

        ; Program data: in this example one byte
        MOV.B  @Rx+,(Flash_Start_Ptr–Start_ptr–1) (Rx)

        BIT    #03Fh,Rx                    ; Segment border?
        JNZ    Test_Wait1

Test_Wait2
        BIT    #WAIT,&FCTL3                ; Test if data written
        JZ     Test_Wait2

        MOV    #Fwkey,&FCTL1               ; Stop segment write
        JMP    Test_busy1                  ; Segment write ends if busy=0

End_Seg_Write
                                           ; All data are programmed
        MOV    #Fwkey,&FCTL1               ; Stop segment write

Test_Busy2
        BIT    #BUSY,&FCTL3                ; Segment write ended?
        JNZ    Test_Busy2
        XOR    #(Fxkey+Lock),&FCTL3        ; Change Lock bit to '1
```

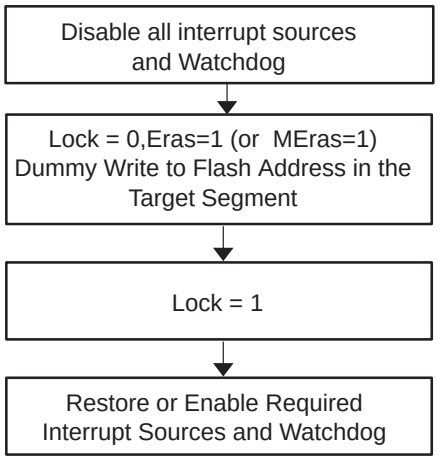
C.5.3.4 Example, Erase Flash Memory Segment or Module via Software Execution Outside This Flash Module

The following sequence can be used to erase a segment, or mass-erase a block of segments.




```
BIC     #( Fwkey+Lock),&FCTL3  ;Reset Lock bit

Test_Busy1
        BIT       #BUSY,&FCTL3
        JNZ       Test_Busy1

        MOV       #(Fwkey+Erase),&FCTL1  ; select segment erase


        CLR     &0F000h

Test_Busy2
        BIT       #BUSY,&FCTL3
        JNZ       Test_Busy2


        XOR     #(Fxkey+Lock),&FCTL3
```

C.5.3.5 Example, Erase Flash Memory Segment Module in the Same Flash Memory Module via Software

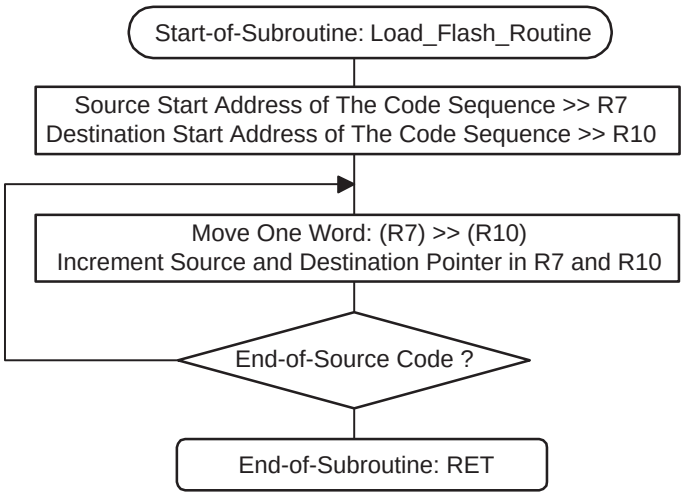



```
; Disable all possible interrupt sources and watchdog

MOV #(Fwkey+Eras),&FCTL1          ;Enable Erase of Flash
CLR  &0FA00h                      ;Dummy Write to Flash
                                  ;Erase Segment 2
; Program execution in information memory if MEras=1 (Eras=0)

XOR #(Fwkey+Lock),&FCTL3          : Change Lock bit to 1

; The erase bit Eras is automatically reset

; Enable those interrupt sources that should be accepted
```

C.5.3.6 Code for Write (Program), Erase, and Mass-Erase

Software that is active during write, erase, or mass-erase may not run in the flash memory module where it is written or erased. Software that controls write, erase, or mass-erase can be located in the flash memory module and copied during execution into RAM. In this case the code should be written position-independent, and should be loaded (for instance, to RAM) before it is used. The algorithm runs in RAM during the programming sequence to avoid conflict when the flash memory is written or erased.

The target flash memory module can not execute the programming code sequence while data is being written to it. In the following example, a subroutine moves the programming-code sequence to another memory such as RAM.




```
;-------------------------------------------------------------
; Definitions used in Subroutine:
; Move programming code sequence into RAM (load_flash_routine)
;-------------------------------------------------------------
Flash_ram          .set   0222h  ; Start address of flash
                                 ; program in the RAM
                                 ; program in the RAM
Prg_source_start   .set   0xxxxh ; Start address of code
                                 ; in the flash to be prg'ed
Prg_source_end     .set   0yyyyh ; End address of code
                                 ; in the flash to be prg'ed
Prg_dest_start     .set Flash_ram
load_flash_routine    ; The code of the program which moves
                      ; Flash access code (write, erase,..)
                      ; starts at label load_flash_routine
   push   r9
   push   r10
   mov    #Prg_source_start,R9  ; load pointer source
   mov    #Prg_dest_start,R10   ; load pointer destination
load_flash_prg
   mov    @R9,0(R10)            ; move a word
   incd   R10                   ; destination  pointer + 2
   incd   R9                    ; source pointer + 2
   cmp    # Prg_source_end,R9   ; compare to end_of_code
   jne    load_ flash_prg
   pop    r9
   pop    r10
   ret
```