TEXAS INSTRUMENTS

MSP430x1xx Family

User's Guide

2003

Mixed Signal Products

SLAU049C

IMPORTANT NOTICE

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

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

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

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

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

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

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Preface

Read This First

About This Manual

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.

Pin functions, internal signal connections and operational paramenters differ from device-to-device. The user should consult the device-specific datasheet for these details.

Related Documentation From Texas Instruments

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

Notational Conventions

Program examples, are shown in a `special typeface`.

Glossary

ACLK	Auxiliary Clock	See *Basic Clock Module*
ADC	Analog-to-Digital Converter	
BOR	Brown-Out Reset	See *System Resets, Interrupts, and Operating Modes*
BSL	Bootstrap Loader	See *www.ti.com/msp430* for application reports
CPU	Central Processing Unit	See *RISC 16-Bit CPU*
DAC	Digital-to-Analog Converter	
DCO	Digitally Controlled Oscillator	See *Basic Clock Module*
dst	Destination	See *RISC 16-Bit CPU*
FLL	Frequency Locked Loop	See *FLL+* in MSP430x4xx Family User's Guide
GIE	General Interrupt Enable	See *System Resets Interrupts and Operating Modes*
INT(N/2)	Integer portion of N/2	
I/O	Input/Output	See *Digital I/O*
ISR	Interrupt Service Routine	
LSB	Least-Significant Bit	
LSD	Least-Significant Digit	
LPM	Low-Power Mode	See *System Resets Interrupts and Operating Modes*
MAB	Memory Address Bus	
MCLK	Master Clock	See *Basic Clock Module*
MDB	Memory Data Bus	
MSB	Most-Significant Bit	
MSD	Most-Significant Digit	
NMI	(Non)-Maskable Interrupt	See *System Resets Interrupts and Operating Modes*
PC	Program Counter	See *RISC 16-Bit CPU*
POR	Power-On Reset	See *System Resets Interrupts and Operating Modes*
PUC	Power-Up Clear	See *System Resets Interrupts and Operating Modes*
RAM	Random Access Memory	
SCG	System Clock Generator	See *System Resets Interrupts and Operating Modes*
SFR	Special Function Register	
SMCLK	Sub-System Master Clock	See *Basic Clock Module*
SP	Stack Pointer	See *RISC 16-Bit CPU*
SR	Status Register	See *RISC 16-Bit CPU*
src	Source	See *RISC 16-Bit CPU*
TOS	Top-of-Stack	See *RISC 16-Bit CPU*
WDT	Watchdog Timer	See *Watchdog Timer*

Register Bit Conventions

Each register is shown with a key indicating the accessibility of the each individual bit, and the initial condition:

Register Bit Accessibility and Initial Condition

Key	Bit Accessibility
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
–(0),–(1)	Condition after POR

Contents

Chapter 1

Introduction

This chapter describes the architecture of the MSP430.

Topic **Page**

1.1 Architecture

The MSP430 incorporates a 16-bit RISC CPU, peripherals, and a flexible clock system that interconnect using a von-Neumann common memory address bus (MAB) and memory data bus (MDB). Partnering a modern CPU with modular memory-mapped analog and digital peripherals, the MSP430 offers solutions for demanding mixed-signal applications.

Key features of the MSP430 include:

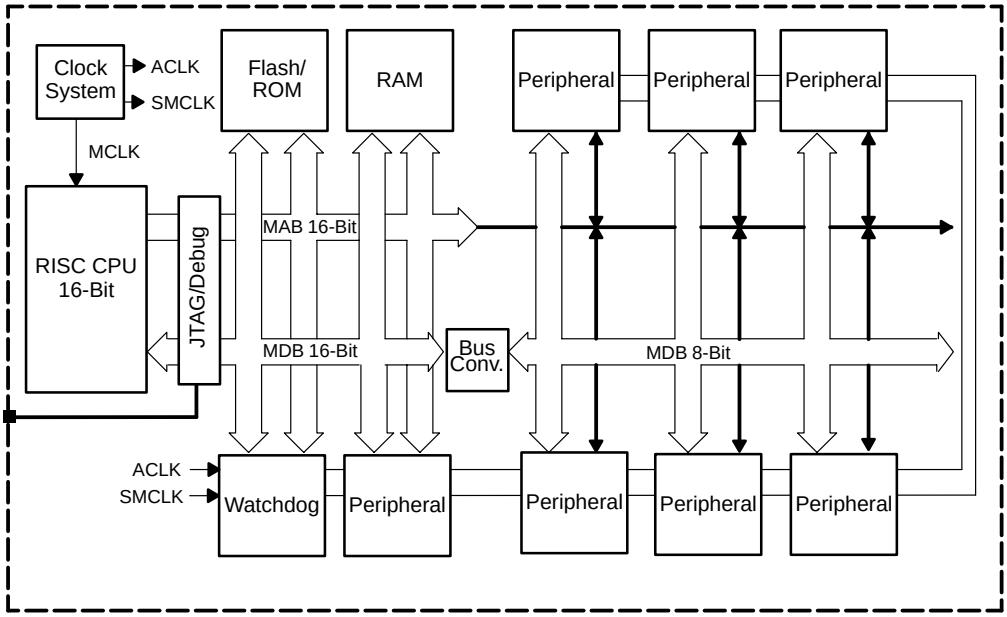
- Ultralow-power architecture extends battery life
  - 0.1-µA RAM retention
  - 0.8-µA real-time clock mode
  - 250-µA / MIPS active
- High-performance analog ideal for precision measurement
  - 12-bit or 10-bit ADC — 200 ksps, temperature sensor, V_{Ref}
  - 12-bit dual-DAC
  - Comparator-gated timers for measuring resistive elements
  - Supply voltage supervisor
- 16-bit RISC CPU enables new applications at a fraction of the code size.
  - Large register file eliminates working file bottleneck
  - Compact core design reduces power consumption and cost
  - Optimized for modern high-level programming
  - Only 27 core instructions and seven addressing modes
  - Extensive vectored-interrupt capability
- In-system programmable Flash permits flexible code changes, field upgrades and data logging

1.2 Flexible Clock System

The clock system is designed specifically for battery-powered applications. A low-frequency uxiliary clock (ACLK) is driven directly from a common 32-kHz watch crystal. The ACLK can be used for a background real-time clock self wake-up function. An integrated high-speed digitally controlled oscillator (DCO) can source the master clock (MCLK) used by the CPU and high-speed peripherals. By design, the DCO is active and stable in less than 6 µs. MSP430-based solutions effectively use the high-performance 16-bit RISC CPU in very short bursts.

- Low-frequency auxiliary clock = Ultralow-power stand-by mode
- High-speed master clock = High performance signal processing

Figure 1–1. MSP430 Architecture



1.3 Embedded Emulation

Dedicated embedded emulation logic resides on the device itself and is accessed via JTAG using no additional system resources.

The benefits of embedded emulation include:

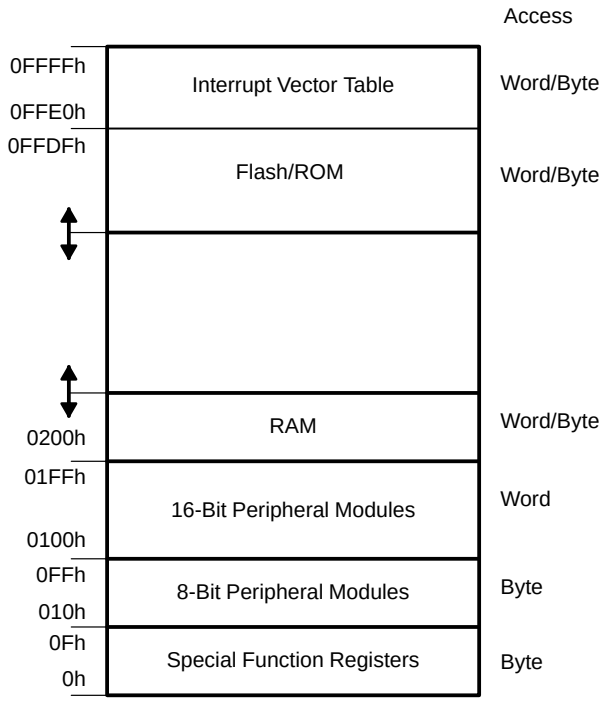
- ❏ Unobtrusive development and debug with full-speed execution, breakpoints, and single-steps in an application are supported.
- ❏ Development is in-system subject to the same characteristics as the final application.
- ❏ Mixed-signal integrity is preserved and not subject to cabling interference.

1.4 Address Space

The MSP430 von-Neumann architecture has one address space shared with special function registers (SFRs), peripherals, RAM, and Flash/ROM memory as shown in Figure 1–2. See the device-specific data sheets for specific memory maps. Code access are always performed on even addresses. Data can be accessed as bytes or words.

The addressable memory space is 64 KB with future expansion planned.

Figure 1–2. Memory Map



1.4.1 Flash/ROM

The start address of Flash/ROM depends on the amount of Flash/ROM present and varies by device. The end address for Flash/ROM is 0FFFFh. Flash can be used for both code and data. Word or byte tables can be stored and used in Flash/ROM without the need to copy the tables to RAM before using them.

The interrupt vector table is mapped into the the upper 16 words of Flash/ROM address space, with the highest priority interrupt vector at the highest Flash/ROM word address (0FFFEh).

1.4.2 RAM

RAM starts at 0200h. The end address of RAM depends on the amount of RAM present and varies by device. RAM can be used for both code and data.

1.4.3 Peripheral Modules

Peripheral modules are mapped into the address space. The address space from 0100 to 01FFh is reserved for 16-bit peripheral modules. These modules should be accessed with word instructions. If byte instructions are used, only even addresses are permissible, and the high byte of the result is always 0.

The address space from 010h to 0FFh is reserved for 8-bit peripheral modules. These modules should be accessed with byte instructions. Read access of byte modules using word instructions results in unpredictable data in the high byte. If word data is written to a byte module only the low byte is written into the peripheral register, ignoring the high byte.

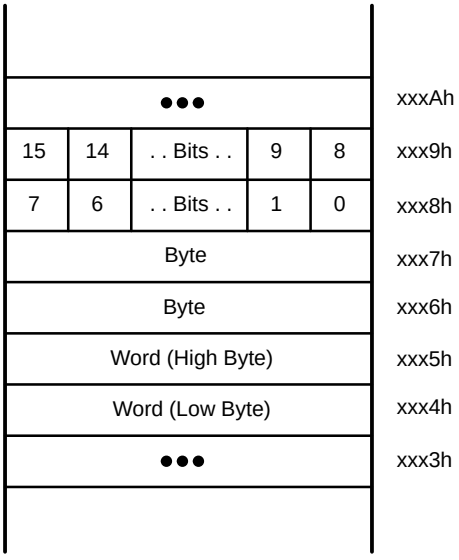
1.4.4 Special Function Registers (SFRs)

Some peripheral functions are configured in the SFRs. The SFRs are located in the lower 16 bytes of the address space, and are organized by byte. SFRs must be accessed using byte instructions only. See the device-specific data sheets for applicable SFR bits.

1.4.5 Memory Organization

Bytes are located at even or odd addresses. Words are only located at even addresses as shown in Figure 1–3. When using word instructions, only even addresses may be used. The low byte of a word is always an even address. The high byte is at the next odd address. For example, if a data word is located at address xxx4h, then the low byte of that data word is located at address xxx4h, and the high byte of that word is located at address xxx5h.

Figure 1–3. Bits, Bytes, and Words in a Byte-Organized Memory



Chapter 2

System Resets, Interrupts, and Operating Modes

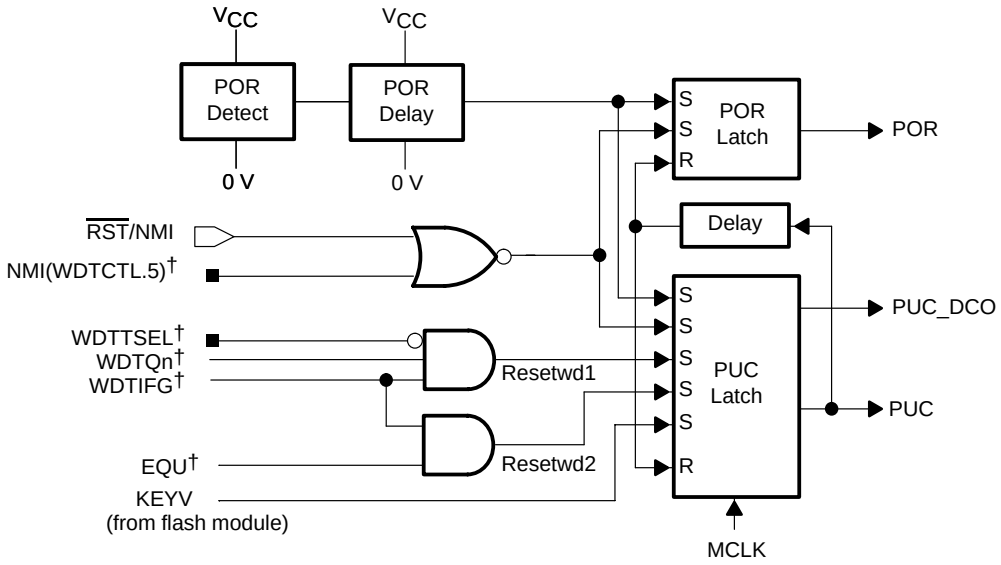
This chapter describes the MSP430x1xx system resets, interrupts, and operating modes.

Topic **Page**

2.1 System Reset and Initialization

The system reset circuitry shown in Figure 2–1 sources both a power-on reset (POR) and a power-up clear (PUC) signal. Different events trigger these reset signals and different initial conditions exist depending on which signal was generated.

Figure 2–1. Power-On Reset and Power-Up Clear Schematic



† From watchdog timer peripheral module

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

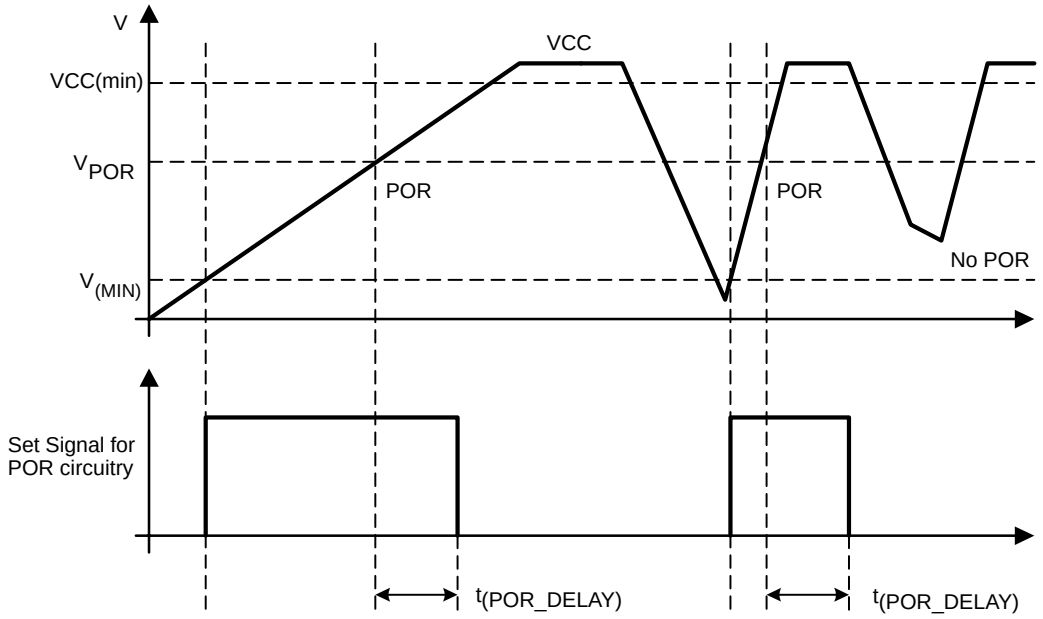
- A POR signal
- Watchdog timer expiration when in watchdog mode only
- Watchdog timer security key violation
- A Flash memory security key violation

2.1.1 Power-On Reset (POR)

When the V_{CC} rise time is slow, the POR detector holds the POR signal active until V_{CC} has risen above the $V_{(POR)}$ level, as shown in Figure 2–2. When the V_{CC} supply provides a fast rise time the POR delay, $t_{(POR\_DELAY)}$, provides active time on the POR signal to allow the MSP430 to initialize.

If power to the MSP430 is cycled, the supply voltage V_{CC} must fall below $V_{(min)}$ 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. See device-specific datasheet for parameters.

Figure 2–2. POR Timing

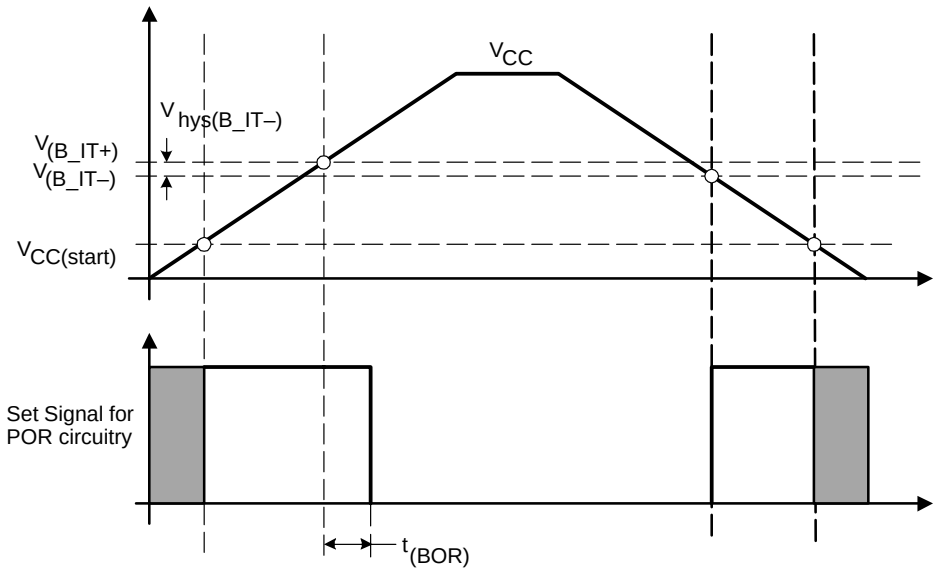


2.1.2 Brownout Reset (BOR)

Some devices have a brownout reset circuit (see device-specific datasheet) that replaces the POR detect and POR delay circuits. The brownout reset circuit detects low supply voltages such as when a supply voltage is applied to or removed from the V_{CC} terminal. The brownout reset circuit resets the device by triggering a POR signal when power is applied or removed. The operating levels are shown in Figure 2–3.

The POR signal becomes active when V_{CC} crosses the $V_{CC(start)}$ level. It remains active until V_{CC} crosses the $V_{(B\_IT+)}$ threshold and the delay $t_{(BOR)}$ elapses. The delay $t_{(BOR)}$ is adaptive being longer for a slow ramping V_{CC}. The hysteresis $V_{Hys(B\_IT-)}$ ensures that the supply voltage must drop below $V_{(B\_IT-)}$ to generate another POR signal from the brownout reset circuitry.

Figure 2–3. Brownout Timing



As the $V_{(B\_IT-)}$ level is significantly above the $V_{(MIN)}$ level of the POR circuit, the BOR provides a reset for power failures where V_{CC} does not fall below $V_{(MIN)}$. See device-specific datasheet for parameters.

2.1.3 Device Initial Conditions After System Reset

After a POR, the initial MSP430 conditions are:

- The $\overline{\text{RST}}$/NMI pin is configured in the reset mode.
- I/O pins are switched to input mode as described in the *Digital I/O* chapter.
- Other peripheral modules and registers are initialized as described in their respective chapters in this manual.
- Status register (SR) is reset.
- The watchdog timer powers up active in watchdog mode.
- Program counter (PC) is loaded with address contained at reset vector location (0FFFEh). CPU execution begins at that address.

Software Initialization

After a system reset, user software must initialize the MSP430 for the application requirements. The following must occur:

- Initialize the SP, typically to the top of RAM.
- Initialize the watchdog to the requirements of the application.
- Configure peripheral modules to the requirements of the application.

Additionally, the watchdog timer, oscillator fault, and flash memory flags can be evaluated to determine the source of the reset.

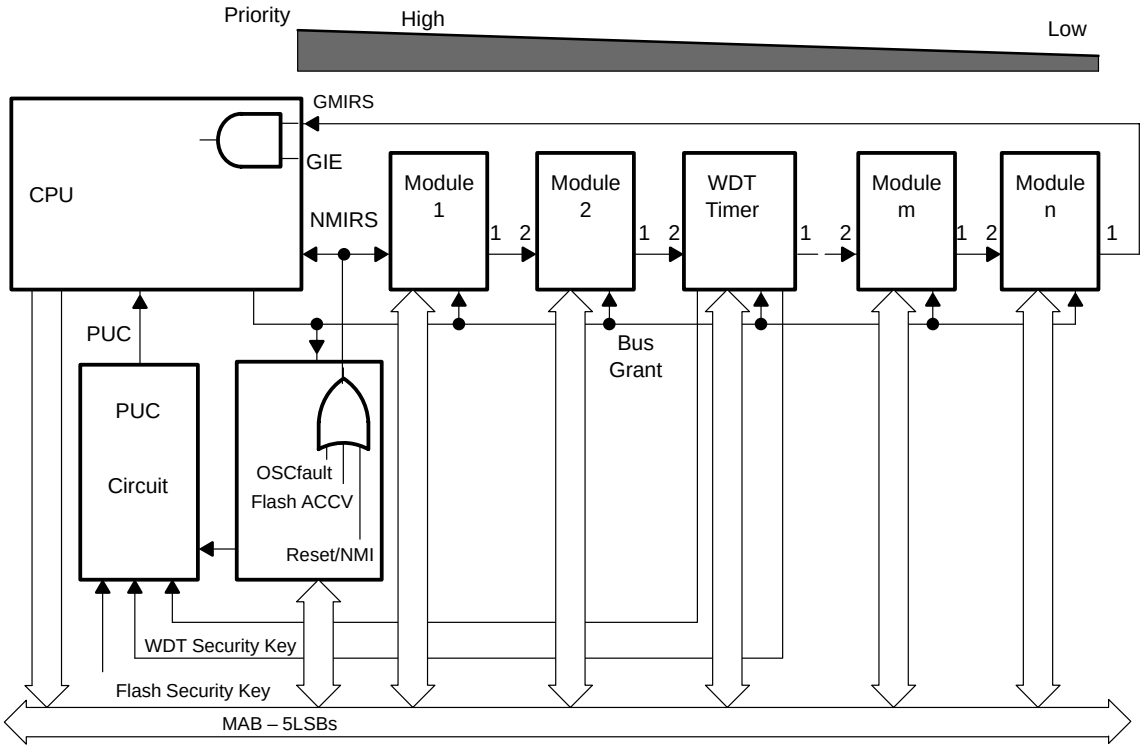
2.2 Interrupts

The interrupt priority is shown in Figure 2–4. The priorities are defined by the arrangement of the modules in the connection chain. The nearer a module is to the CPU/NMIRS, the higher the priority.

There are three types of interrupts:

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

Figure 2–4. Interrupt Priority



2.2.1 (Non)-Maskable Interrupts (NMI)

(Non)-maskable NMI interrupts are not masked by the general interrupt enable bit (GIE), but are enabled by individual interrupt enable bits (ACCVIE, NMIIE, OFIE). When a NMI interrupt is accepted, all NMI interrupt enable bits are automatically reset. Program execution begins at the address stored in the (non)-maskable interrupt vector, 0FFFCh. User software must set the required NMI interrupt enable bits for the interrupt to be re-enabled. The block diagram for NMI sources is shown in Figure 2–5.

A (non)-maskable NMI interrupt can be generated by three sources:

- An edge on the RST/NMI pin
- An oscillator fault occurs
- An access violation to the flash memory

Reset/NMI Pin

At power-up, the $\overline{\text{RST}}$/NMI pin is configured in the reset mode. The function of the $\overline{\text{RST}}$/NMI pins is selected in the watchdog control register WDTCTL. If the $\overline{\text{RST}}$/NMI pin is set to the reset function, the CPU is held in the reset state as long as the $\overline{\text{RST}}$/NMI pin is held low. After the input changes to a high state, the CPU starts program execution at the word address stored in the reset vector, 0FFFEh.

If the $\overline{\text{RST}}$/NMI pin is configured by user software to the NMI function, a signal edge selected by the NMIES bit generates an NMI interrupt if the NMIIE bit is set. The $\overline{\text{RST}}$/NMI flag NMIFG is also set.

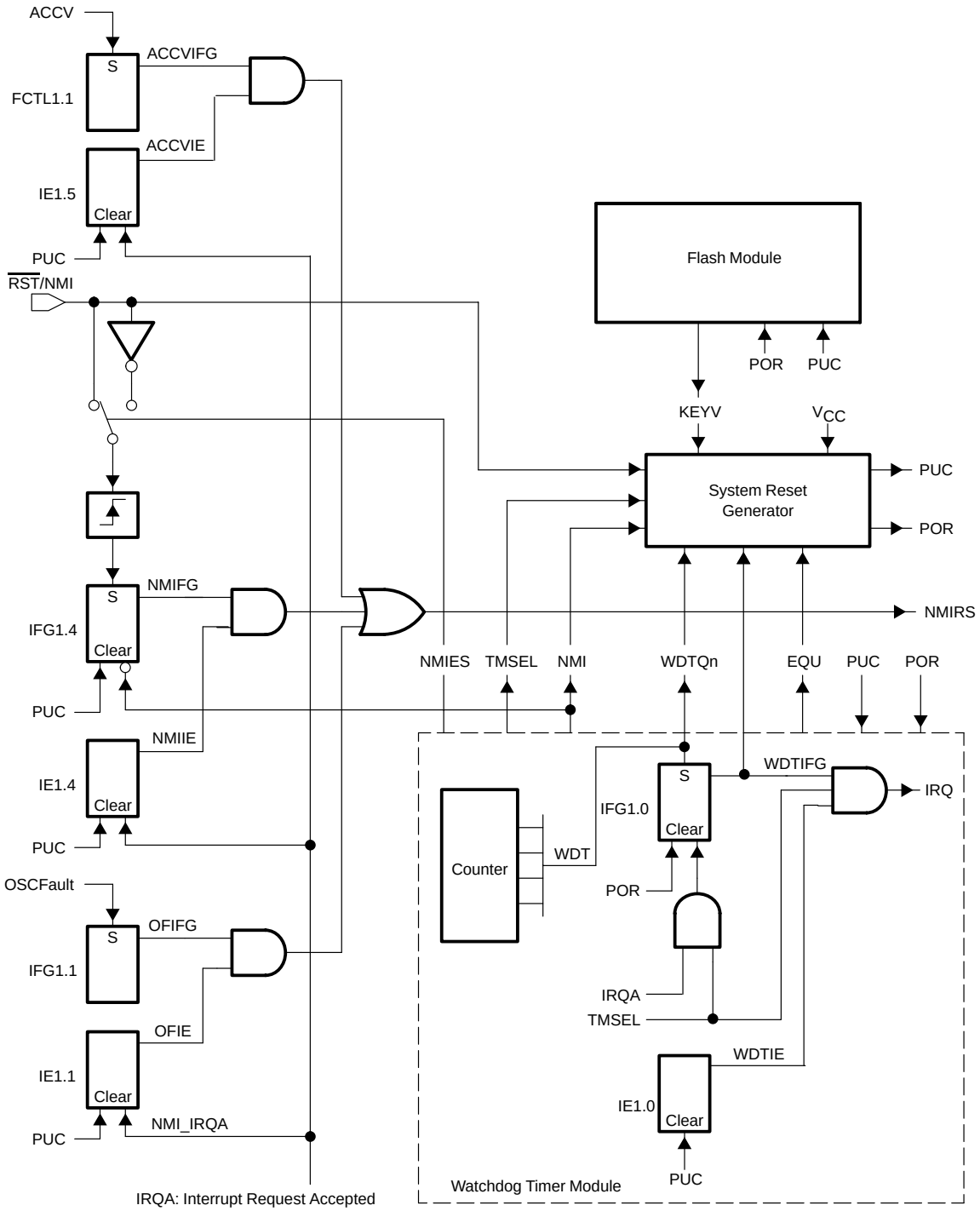
Note: Holding $\overline{\text{RST}}$/NMI Low

When configured in the NMI mode, a signal generating an NMI event should not hold the $\overline{\text{RST}}$/NMI pin low. If a PUC occurs from a different source while the NMI signal is low, the device will be held in the reset state because a PUC changes the $\overline{\text{RST}}$/NMI pin to the reset function.

Note: Modifying NMIES

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

Figure 2–5. Block Diagram of (Non)-Maskable Interrupt Sources



Oscillator Fault

The oscillator fault signal warns of a possible error condition with the crystal oscillator. The oscillator fault can be enabled to generate an NMI interrupt by setting the OFIE bit. The OFIFG flag can then be tested by NMI the interrupt service routine to determine if the NMI was caused by an oscillator fault.

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.

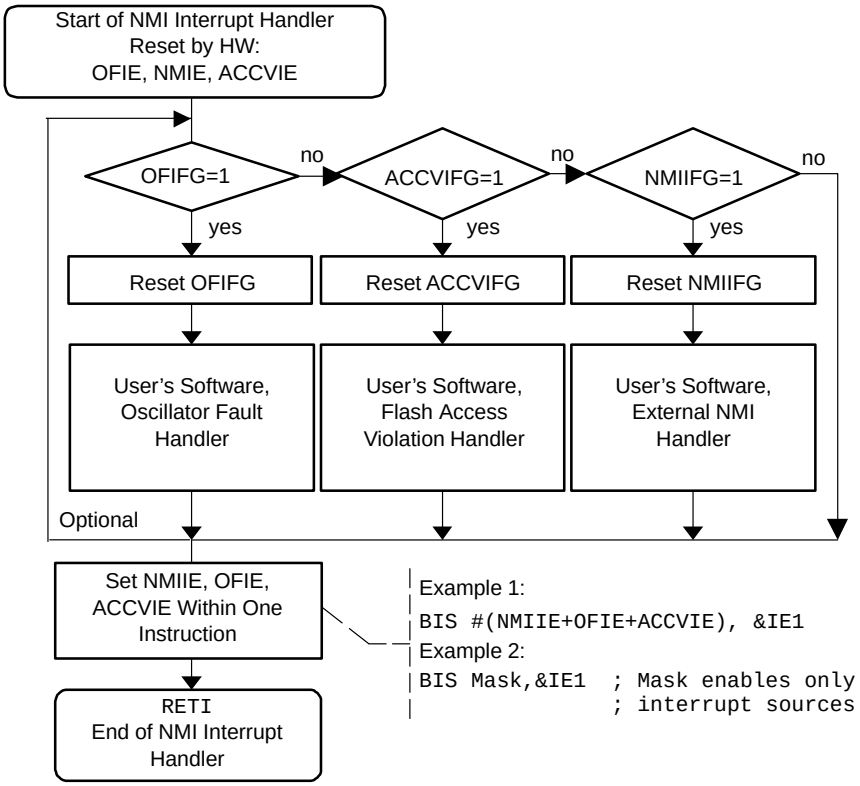
Flash Access Violation

The flash ACCVIFG flag is set when a flash access violation occurs. The flash access violation can be enabled to generate an NMI interrupt by setting the ACCVIE bit. The ACCVIFG flag can then be tested by NMI the interrupt service routine to determine if the NMI was caused by a flash access violation.

Example of an NMI Interrupt Handler

The NMI interrupt is a multiple-source interrupt. An NMI interrupt automatically resets the NMIIE, OFIE and ACCVIE interrupt-enable bits. The user NMI service routine resets the interrupt flags and re-enables the interrupt-enable bits according to the application needs as shown in Figure 2–6.

Figure 2–6. NMI Interrupt Handler



Note: Enabling NMI Interrupts with ACCVIE, NMIIE, and OFIE

The ACCVIE, NMIIE, and OFIE enable bits should not be set inside of an NMI interrupt service routine, unless they are set by the last instruction of the routine before the `RETI` instruction. Otherwise, nested NMI interrupts may occur, causing stack overflow and unpredictable operation.

2.2.2 Maskable Interrupts

Maskable interrupts are caused by peripherals with interrupt capability including the watchdog timer overflow in time mode. Each maskable 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 (SR).

Each individual peripheral interrupt is discussed in the associated peripheral module chapter in this manual.

2.2.3 Interrupt Processing

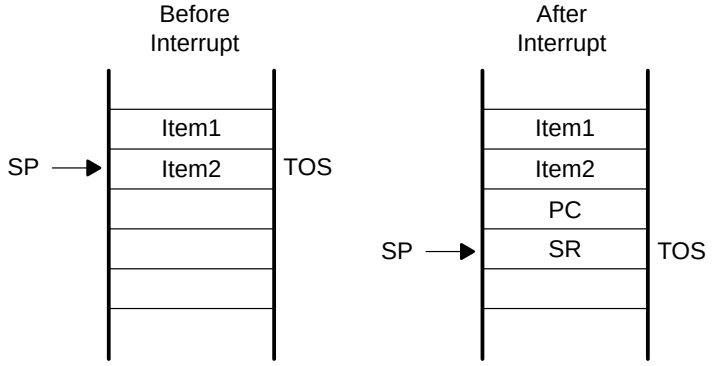
When an interrupt is requested from a peripheral and the peripheral interrupt enable bit and GIE bit are set, the interrupt service routine is requested. Only the individual enable bit must be set for (non)-maskable interrupts to be requested.

Interrupt Acceptance

The interrupt latency is 6 cycles, starting with the acceptance of an interrupt request, and lasting until the start of execution of the first instruction of the interrupt-service routine, as shown in Figure 2–7. The interrupt logic executes the following:

1) Any currently executing instruction is completed.
2) The PC, which points to the next instruction, is pushed onto the stack.
3) The SR is pushed onto the stack.
4) The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.
5) The interrupt request flag resets automatically on single-source flags. Multiple source flags remain set for servicing by software.
6) The SR is cleared with the exception of SCG0, which is left unchanged. This terminates any low-power mode.
7) The content of the interrupt vector is loaded into the PC: the program continues with the interrupt service routine at that address.

Figure 2–7. Interrupt Processing



Return From Interrupt

The interrupt handling routine terminates with the instruction:

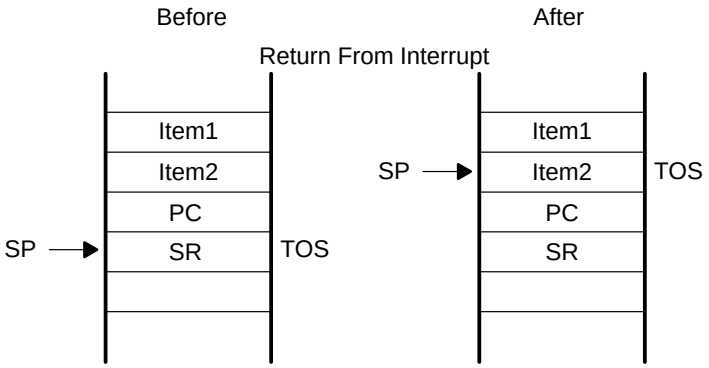
`RETI` (return from an interrupt service routine)

The return from the interrupt takes 5 cycles to execute the following actions and is illustrated in Figure 2–8.

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

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

Figure 2–8. Return From Interrupt



Interrupt nesting is enabled if the GIE bit is set inside the interrupt service routine.

2.2.4 Interrupt Vectors

The interrupt vectors and the power-up starting address are located in the address range 0FFFFh – 0FFE0h as described in Table 2–1. A vector is programmed by the user with the 16-bit address of the corresponding interrupt service routine. See the device-specific data sheet for the complete interrupt vector list.

Table 2–1. Interrupt Sources,Flags, and Vectors

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-up, external reset, watchdog, flash password	WDTIFG KEYV	Reset	0FFFEh	15, highest
NMI, oscillator fault, flash memory access violation	NMIIFG OFIFG ACCVIFG	(non)-maskable (non)-maskable (non)-maskable	0FFFCh	14
device-specific			0FFFAh	13
device-specific			0FFF8h	12
device-specific			0FFF6h	11
Watchdog timer	WDTIFG	maskable	0FFF4h	10
device-specific			0FFF2h	9
device-specific			0FFF0h	8
device-specific			0FFEEh	7
device-specific			0FFECh	6
device-specific			0FFEAh	5
device-specific			0FFE8h	4
I/O Port P2	P2IFG.0 to P2IFG.7	maskable	0FFE6h	3
I/O Port P1	P1IFG.0 to P1IFG.7	maskable	0FFE4h	2
device-specific			0FFE2h	1
device-specific			0FFE0h	0, lowest

2.2.5 Special Function Registers (SFRs)

Some module enable bits, interrupt enable bits, and interrupt flags are located in the SFRs. The SFRs are located in the lower address range and are implemented in byte format. SFRs must be accessed using byte instructions. See the device-specific datasheet for the SFR configuration.

2.3 Operating Modes

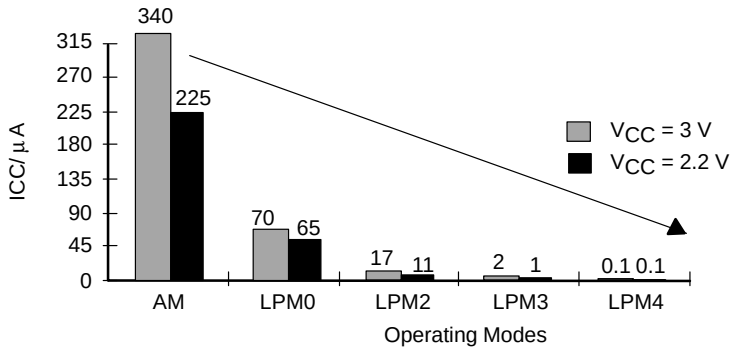
The MSP430 family is designed for ultralow-power applications and uses different operating modes shown in Figure 2–10.

The operating modes take into account three different needs:

- Ultralow-power
- Speed and data throughput
- Minimization of individual peripheral current consumption

The MSP430 typical current consumption is shown in Figure 2–9.

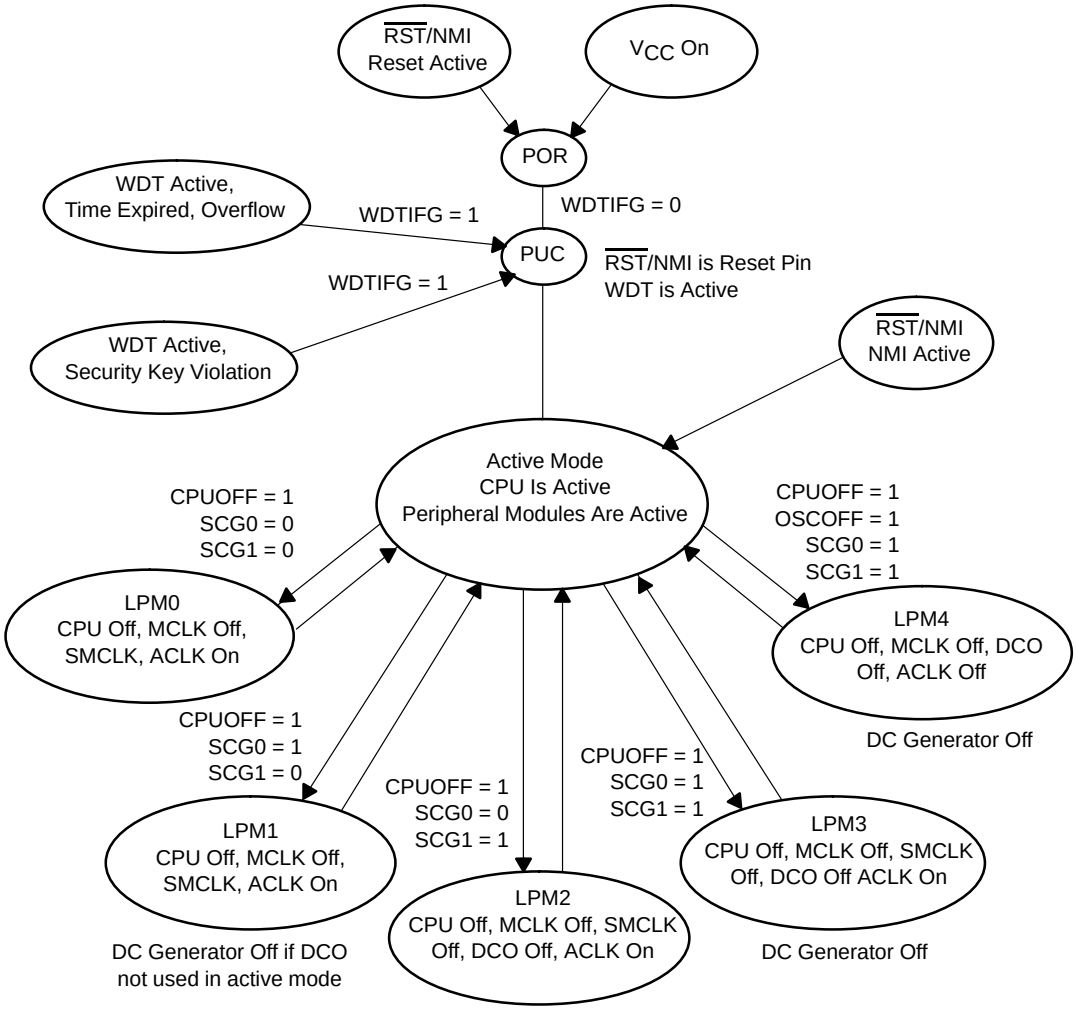
Figure 2–9. Typical Current Consumption of 13x and 14x Devices vs Operating Modes



The low-power modes 0–4 are configured with the CPUOFF, OSCOFF, SCG0, and SCG1 bits in the status register The advantage of including the CPUOFF, OSCOFF, SCG0, and SCG1 mode-control bits in the status register is that the present operating mode is saved onto the stack during an interrupt service routine. Program flow returns to the previous operating mode if the saved SR value is not altered during the interrupt service routine. Program flow can be returned to a different operating mode by manipulating the saved SR value on the stack inside of the interrupt service routine. The mode-control bits and the stack can be accessed with any instruction.

When setting any of the mode-control bits, the selected operating mode takes effect immediately. Peripherals operating with any disabled clock are disabled until the clock becomes active. The peripherals may also be disabled with their individual control register settings. All I/O port pins and RAM/registers are unchanged. Wake up is possible through all enabled interrupts.

Figure 2–10. MSP430x1xx Operating Modes For Basic Clock System



SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled SMCLK , ACLK are active
0	1	0	1	LPM1	CPU, MCLK, DCO osc. are disabled DC generator is disabled if the DCO is not used for MCLK or SMCLK in active mode SMCLK , ACLK are active
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO osc. are disabled DC generator remains enabled ACLK is active
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO osc. are disabled DC generator disabled ACLK is active
1	1	1	1	LPM4	CPU and all clocks disabled

2.3.1 Entering and Exiting Low-Power Modes

An enabled interrupt event wakes the MSP430 from any of the low-power operating modes. The program flow is:

- Enter interrupt service routine:
  - The PC and SR are stored on the stack
  - The CPUOFF, SCG1, and OSCOFF bits are automatically reset
- Options for returning from the interrupt service routine:
  - The original SR is popped from the stack, restoring the previous operating mode.
  - The SR bits stored on the stack can be modified within the interrupt service routine returning to a different operating mode when the RETI instruction is executed.

```
; Enter LPM0 Example
   BIS   #GIE+CPUOFF,SR           ; Enter LPM0
;  ...                            ; Program stops here
;
; Exit LPM0 Interrupt Service Routine
   BIC   #CPUOFF,0(SP)            ; Exit LPM0 on RETI
   RETI

; Enter LPM3 Example
   BIS   #GIE+CPUOFF+SCG1+SCG0,SR ; Enter LPM3
;  ...                            ; Program stops here
;
; Exit LPM3 Interrupt Service Routine
   BIC   #CPUOFF+SCG1+SCG0,0(SP)  ; Exit LPM3 on RETI
   RETI
```

Extended Time in Low-Power Modes

The negative temperature coefficient of the DCO should be considered when the DCO is disabled for extended low-power mode periods. If the temperature changes significantly, the DCO frequency at wake-up may be significantly different from when the low-power mode was entered and may be out of the specified operating range. To avoid this, the DCO can be set to it lowest value before entering the low-power mode for extended periods of time where temperature can change.

```
; Enter LPM4 Example with lowest DCO Setting
   BIC   #RSEL2+RSEL1+RSEL0,&BCSCTL1     ; Lowest RSEL
   BIS   #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR ; Enter LPM4
;  ...                                    ; Program stops
;
; Interrupt Service Routine
   BIC   #CPUOFF+OSCOFF+SCG1+SCG0,0(SR); Exit LPM4 on RETI
   RETI
```

2.4 Principles for Low-Power Applications

Often, the most important factor for reducing power consumption is using the MSP430's clock system to maximize the time in LPM3. LPM3 power consumption is less than 2 μA typical with both a real-time clock function and all interrupts active. A 32-kHz watch crystal is used for the ACLK and the CPU is clocked from the DCO (normally off) which has a 6-μs wake-up.

- ❏ Use interrupts to wake the processor and control program flow.
- ❏ Peripherals should be switched on only when needed.
- ❏ Use low-power integrated peripheral modules in place of software driven functions. For example Timer_A and Timer_B can automatically generate PWM and capture external timing, with no CPU resources.
- ❏ Calculated branching and fast table look-ups should be used in place of flag polling and long software calculations.
- ❏ Avoid frequent subroutine and function calls due to overhead.
- ❏ For longer software routines, single-cycle CPU registers should be used.

2.5 Connection of Unused Pins

The correct termination of all unused pins is listed in Table 2–2.

Table 2–2. Connection of Unused Pins

Pin	Potential	Comment
AV_{CC}	DV_{CC}	
AV_{SS}	DV_{SS}	
V_{REF+}	Open	
Ve_{REF+}	DV_{SS}	
V_{REF-}/Ve_{REF-}	DV_{SS}	
XIN	DV_{SS}	
XOUT	Open	
XT2IN	DV_{SS}	13x, 14x, 15x and 16x devices
XT2OUT	Open	13x, 14x, 15x and 16x devices
Px.0 to Px.7	Open	Switched to port function, output direction
RST/NMI	DV_{CC} or V_{CC}	Pullup resistor 100 kΩ
Test/V_{PP}	DV_{SS}	P11x devices
Test	DV_{SS}	11xx and 12xx devices
TDO	Open	
TDI	Open	
TMS	Open	
TCK	Open	

Chapter 3

RISC 16-Bit CPU

This chapter describes the MSP430 CPU, addressing modes, and instruction set.

Topic **Page**

3.1 CPU Introduction

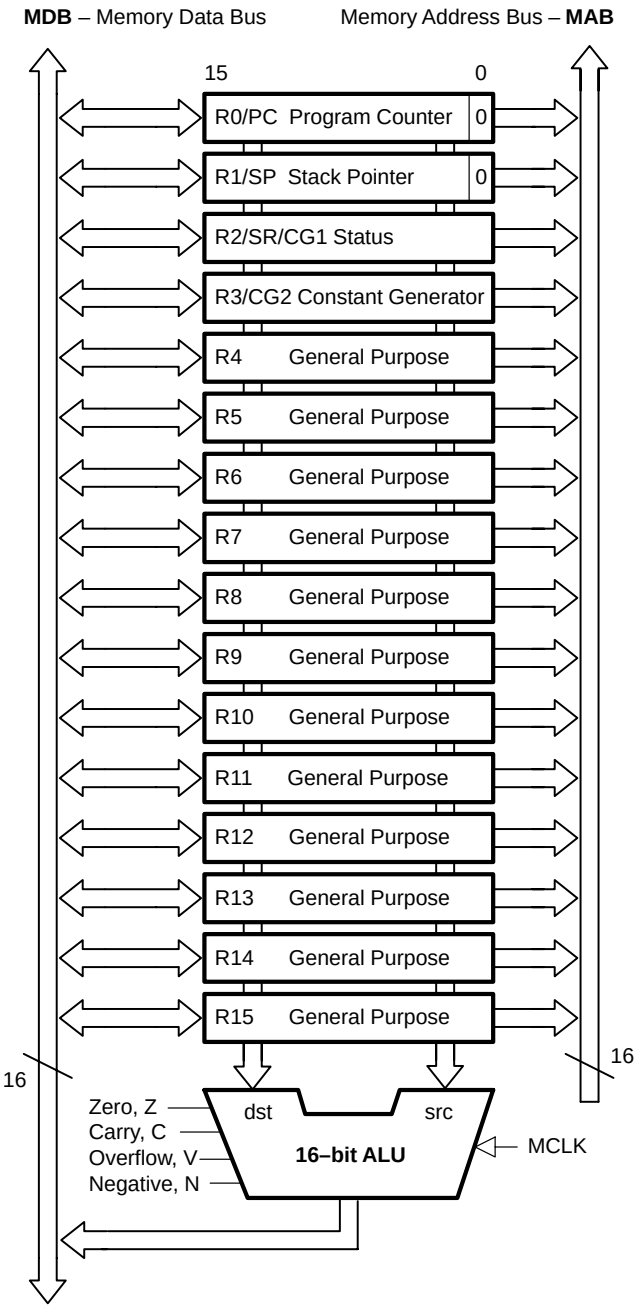
The CPU incorporates features specifically designed for modern programming techniques such as calculated branching, table processing and the use of high-level languages such as C. The CPU can address the complete address range without paging.

The CPU features include:

- ❑ RISC architecture with 27 instructions and 7 addressing modes.
- ❑ Orthogonal architecture with every instruction usable with every addressing mode.
- ❑ Full register access including program counter, status registers, and stack pointer.
- ❑ Single-cycle register operations.
- ❑ Large 16-bit register file reduces fetches to memory.
- ❑ 16-bit address bus allows direct access and branching throughout entire memory range.
- ❑ 16-bit data bus allows direct manipulation of word-wide arguments.
- ❑ Constant generator provides six most used immediate values and reduces code size.
- ❑ Direct memory-to-memory transfers without intermediate register holding.
- ❑ Word and byte addressing and instruction formats.

The block diagram of the CPU is shown in Figure 3–1.

Figure 3–1. CPU Block Diagram



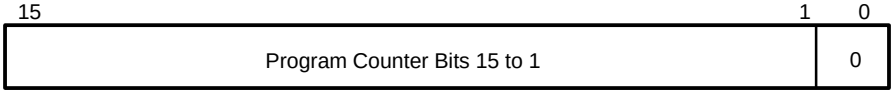
3.2 CPU Registers

The CPU incorporates sixteen 16-bit registers. R0, R1, R2 and R3 have dedicated functions. R4 to R15 are working registers for general use.

3.2.1 Program Counter (PC)

The 16-bit program counter (PC/R0) points to the next instruction to be executed. Each instruction uses an even number of bytes (two, four, or six), and the PC is incremented accordingly. Instruction accesses in the 64-KB address space are performed on word boundaries, and the PC is aligned to even addresses. Figure 3–2 shows the program counter.

Figure 3–2. Program Counter



The PC can be addressed with all instructions and addressing modes. A few examples:

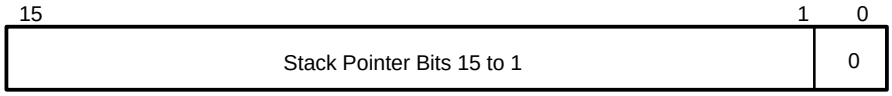
```
MOV   #LABEL,PC ; Branch to address LABEL
MOV   LABEL,PC  ; Branch to address contained in LABEL
MOV   @R14,PC   ; Branch indirect, indirect R14
```

3.2.2 Stack Pointer (SP)

The stack pointer (SP/R1) is used by the CPU to store the return addresses of subroutine calls and interrupts. It uses a predecrement, postincrement scheme. In addition, the SP can be used by software with all instructions and addressing modes. Figure 3–3 shows the SP. The SP is initialized into RAM by the user, and is aligned to even addresses.

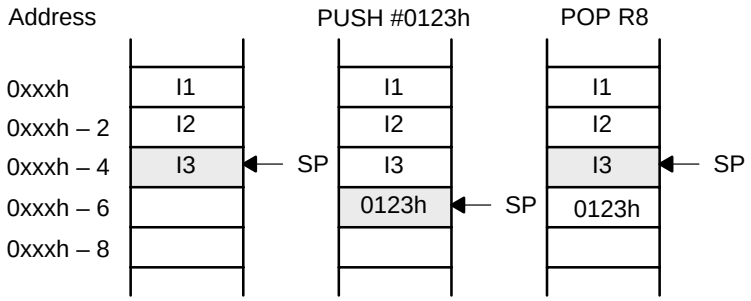
Figure 3–4 shows stack usage.

Figure 3–3. Stack Pointer




```
MOV   2(SP),R6  ; Item I2 –> R6
MOV   R7,0(SP)  ; Overwrite TOS with R7
PUSH  #0123h    ; Put 0123h onto TOS
POP   R8        ; R8 = 0123h
```

Figure 3–4. Stack Usage



The special cases of using the SP as an argument to the PUSH and POP instructions are described and shown in Figure 3–5.

Figure 3–5. PUSH SP - POP SP Sequence

The stack pointer is changed after a PUSH SP instruction.

The stack pointer is not changed after a POP SP instruction. The POP SP instruction places SP1 into the stack pointer SP (SP2=SP1)

3.2.3 Status Register (SR)

The status register (SR/R2), used as a source or destination register, can be used in the register mode only addressed with word instructions. The remaining combinations of addressing modes are used to support the constant generator. Figure 3–6 shows the SR bits.

Figure 3–6. Status Register Bits

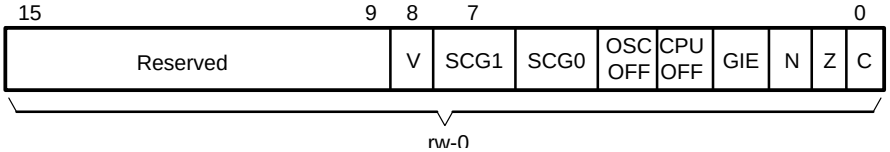


Table 3–1 describes the status register bits.

Table 3–1. Description of Status Register Bits

Bit	Description	
V	Overflow bit. This bit is set when the result of an arithmetic operation overflows the signed-variable range.	
	`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	System clock generator 1. This bit, when set, turns off the SMCLK.	
SCG0	System clock generator 0. This bit, when set, turns off the DCO dc generator, if DCOCLK is not used for MCLK or SMCLK.	
OSCOFF	Oscillator Off. This bit, when set, turns off the LFXT1 crystal oscillator, when LFXT1CLK is not use for MCLK or SMCLK	
CPUOFF	CPU off. This bit, when set, turns off the CPU.	
GIE	General interrupt enable. This bit, when set, enables maskable interrupts. When reset, all maskable interrupts are disabled.	
N	Negative bit. This bit is set when the result of a byte or word operation is negative and cleared when the result is not negative.	
	Word operation:	N is set to the value of bit 15 of the result
	Byte operation:	N is set to the value of bit 7 of the result
Z	Zero bit. This bit is set when the result of a byte or word operation is 0 and cleared when the result is not 0.	
C	Carry bit. This bit is set when the result of a byte or word operation produced a carry and cleared when no carry occurred.	

3.2.4 Constant Generator Registers CG1 and CG2

Six commonly-used constants are generated with the constant generator registers R2 and R3, without requiring an additional 16-bit word of program code. The constants are selected with the source-register addressing modes (As), as described in Table 3–2.

Table 3–2. 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 constant generator advantages are:

- ❏ No special instructions required
- ❏ No additional code word for the six constants
- ❏ No code memory access required to retrieve the constant

The assembler uses the constant generator automatically if one of the six constants is used as an immediate source operand. Registers R2 and R3, used in the constant mode, cannot be addressed explicitly; they act as source-only registers.

Constant Generator – Expanded Instruction Set

The RISC instruction set of the MSP430 has only 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
```

where the #0 is replaced by the assembler, and R3 is used with As=00.

```
INC             dst
```

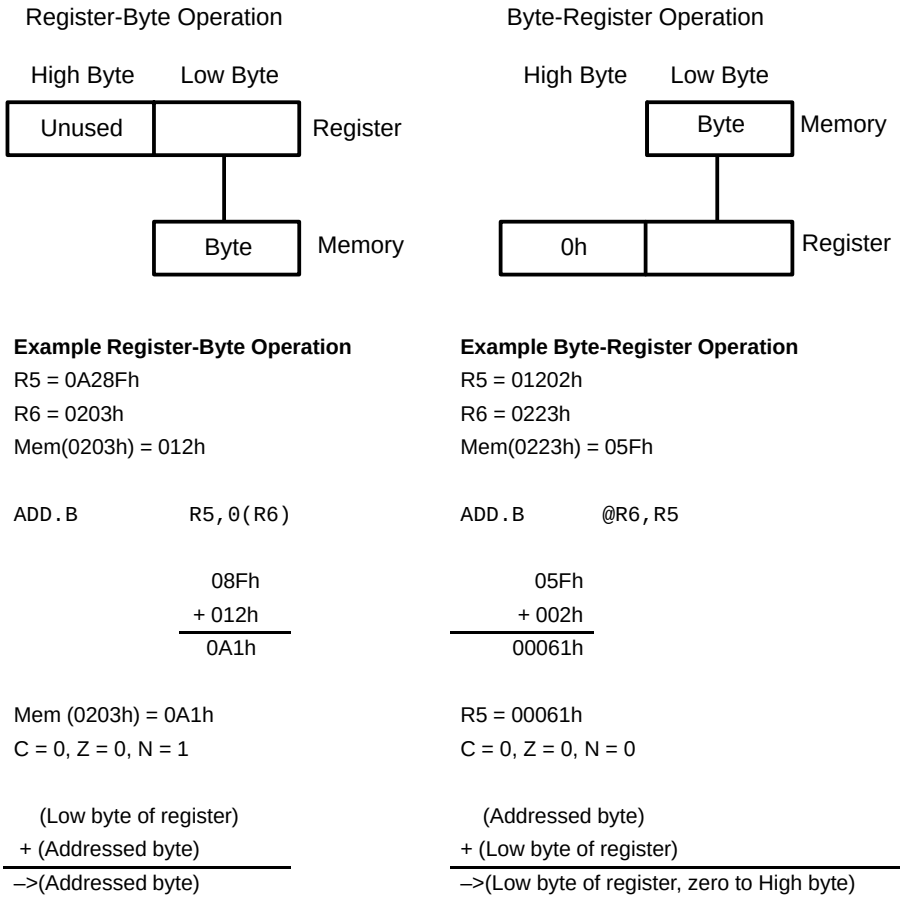
is replaced by:

```
ADD             0(R3),dst
```

3.2.5 General–Purpose Registers R4 - R15

The twelve registers, R4–R15, are general-purpose registers. All of these registers can be used as data registers, address pointers, or index values and can be accessed with byte or word instructions as shown in Figure 3–7.

Figure 3–7. Register-Byte/Byte-Register Operations



3.3 Addressing Modes

Seven addressing modes for the source operand and four addressing modes for the destination operand can address the complete address space with no exceptions. The bit numbers in Table 3–3 describe the contents of the As (source) and Ad (destination) mode bits.

Table 3–3. 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. X is stored in the next word. Indexed mode X(SR) is used.
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 by 1 for .B instructions and by 2 for .W instructions.
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.

Note: Use of Labels *EDE* and *TONI*

Throughout MSP430 documentation *EDE* and *TONI* are used as generic labels. They are only labels. They have no special meaning.

3.3.1 Register Mode

The register mode is described in Table 3–4.

Table 3–4. Register Mode Description

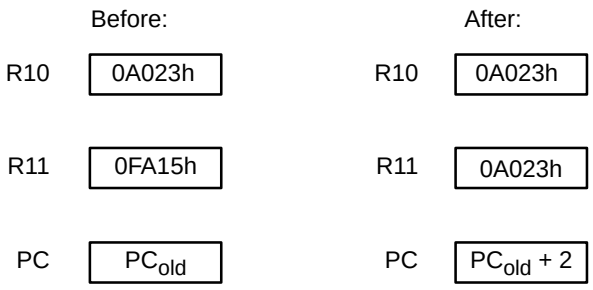
Assembler Code	Content of ROM
`MOV R10,R11`	`MOV R10,R11`

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.

3.3.2 Indexed Mode

The indexed mode is described in Table 3–5.

Table 3–5. Indexed Mode Description

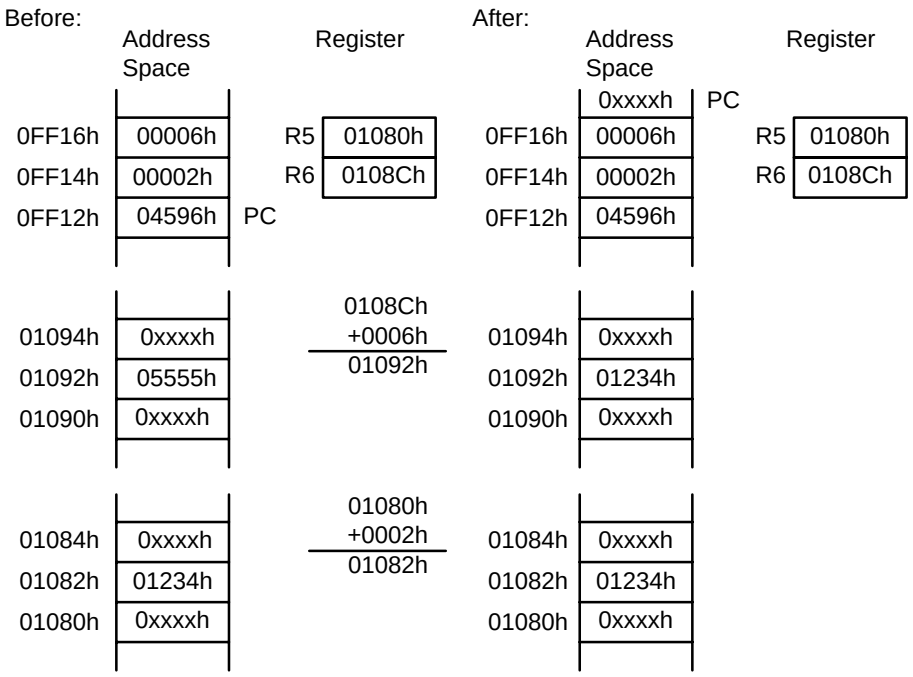
Assembler Code	Content of ROM
`MOV 2(R5),6(R6)`	`MOV X(R5),Y(R6)` X = 2 Y = 6

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):`



3.3.3 Symbolic Mode

The symbolic mode is described in Table 3–6.

Table 3–6. Symbolic Mode Description

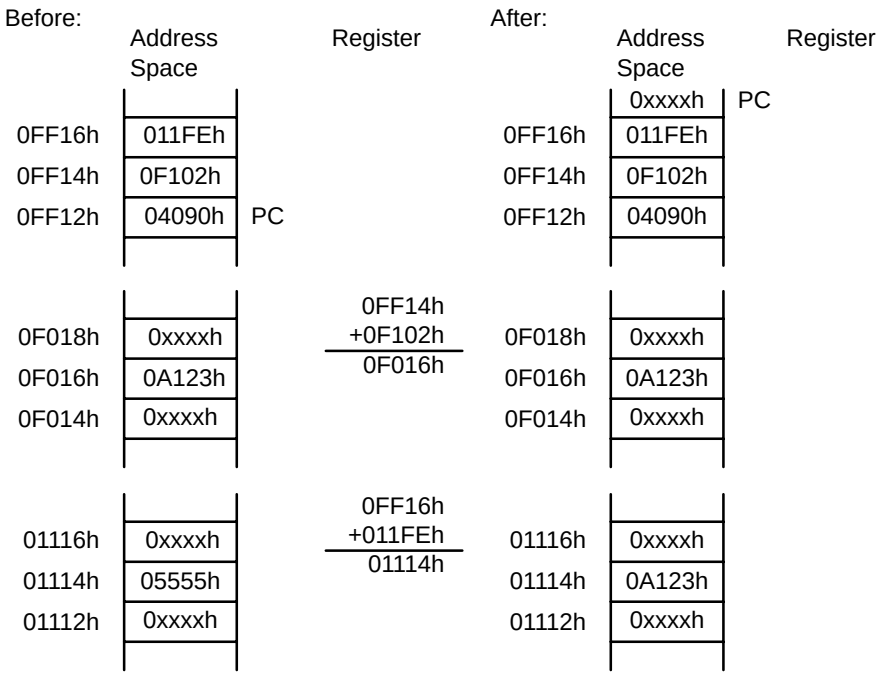
Assembler Code	Content of ROM
`MOV EDE,TONI`	`MOV X(PC),Y(PC)` X = EDE – PC Y = TONI – PC

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
```



3.3.4 Absolute Mode

The absolute mode is described in Table 3–7.

Table 3–7. Absolute Mode Description

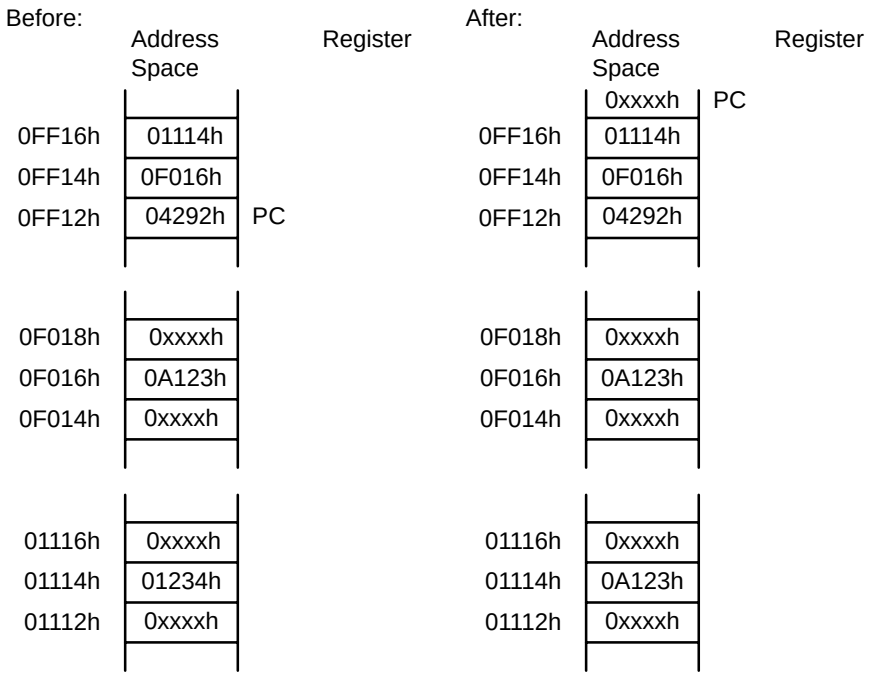
Assembler Code	Content of ROM
`MOV &EDE,&TONI`	`MOV X(0),Y(0)` X = EDE Y = TONI

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).

3.3.5 Indirect Register Mode

The indirect register mode is described in Table 3–8.

Table 3–8. Indirect Mode Description

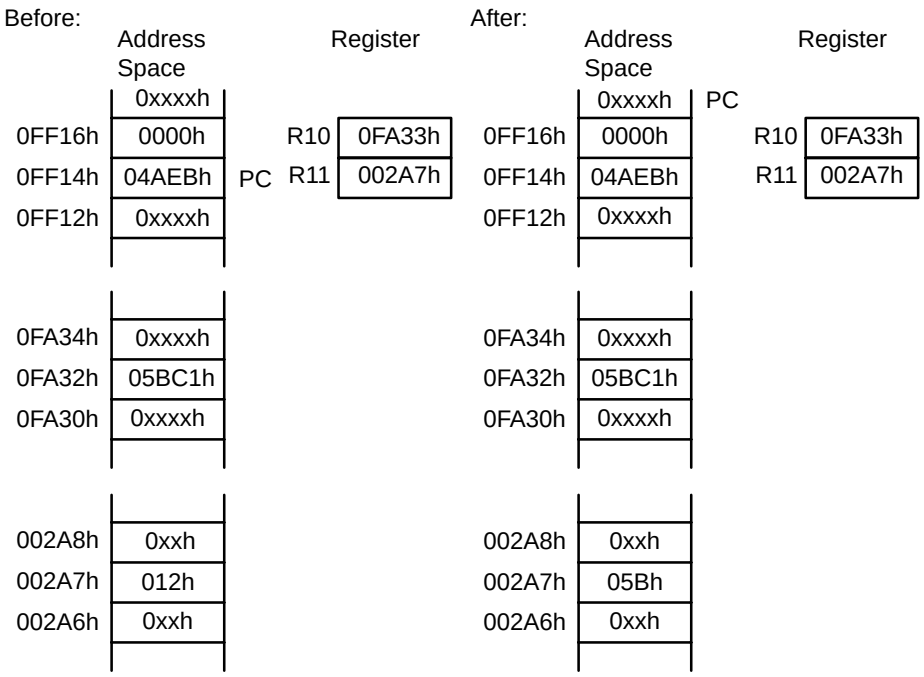
Assembler Code	Content of ROM
`MOV @R10,0(R11)`	`MOV @R10,0(R11)`

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)`



3.3.6 Indirect Autoincrement Mode

The indirect autoincrement mode is described in Table 3–9.

Table 3–9. Indirect Autoincrement Mode Description

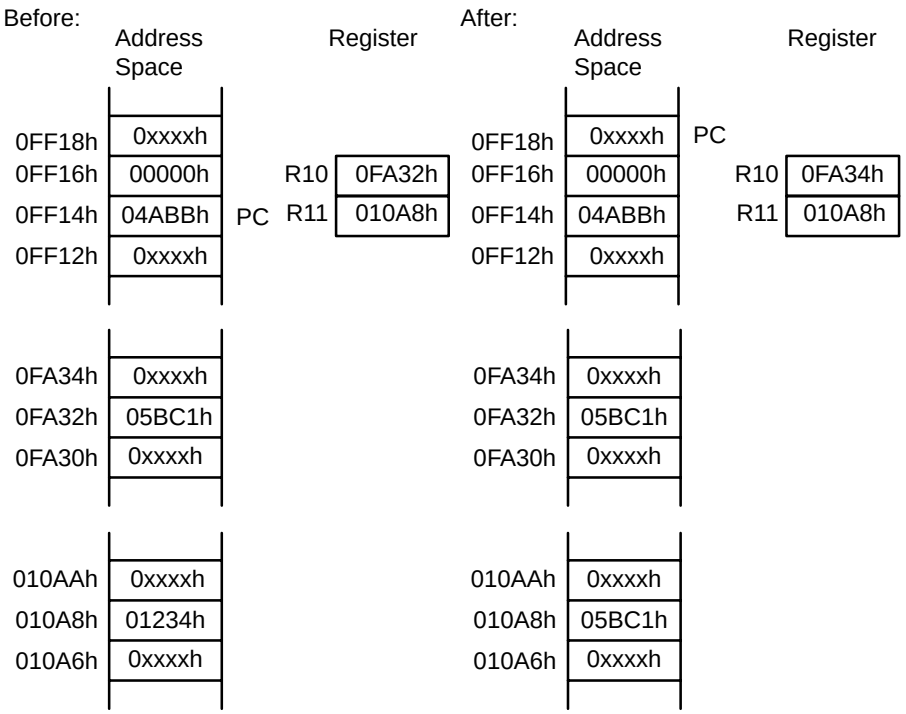
Assembler Code	Content of ROM
`MOV @R10+,0(R11)`	`MOV @R10+,0(R11)`

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.

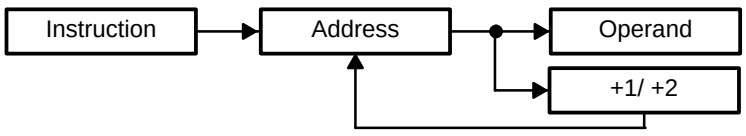
Comment: Valid only for source operand. The substitute for destination operand is 0(Rd) plus second instruction INCD Rd.

Example: `MOV @R10+,0(R11)`



The autoincrementing of the register contents occurs after the operand is fetched. This is shown in Figure 3–8.

Figure 3–8. Operand Fetch Operation



3.3.7 Immediate Mode

The immediate mode is described in Table 3–10.

Table 3–10. Immediate Mode Description

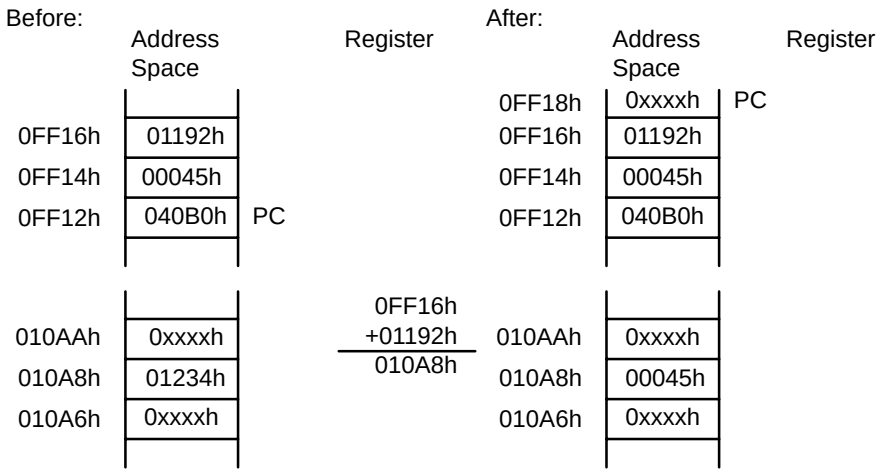
Assembler Code	Content of ROM
`MOV #45h,TONI`	`MOV @PC+,X(PC)` 45 X = TONI – PC

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 45h, 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 #45h,TONI`



3.4 Instruction Set

The complete MSP430 instruction set consists of 24 core instructions and 27 emulated instructions. The core instructions are instructions that have unique op-codes decoded by the CPU. The emulated instructions are instructions that make code easier to write and read, but do not have op-codes themselves, instead they are replaced automatically by the assembler with an equivalent core instruction. There is no code or performance penalty for using emulated instruction.

There are three core-instruction formats:

- ❑ Dual-operand
- ❑ Single-operand
- ❑ Jump

All single-operand and dual-operand instructions can be byte or word instructions by using .B or .W extensions. Byte instructions are used to access byte data or byte peripherals. Word instructions are used to access word data or word peripherals. If no extension is used, the instruction is a word instruction.

The source and destination 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 writable. 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.

Note: Use of Labels *EDE* and *TONI*

Throughout MSP430 documentation *EDE* and *TONI* are used as generic labels. They are only labels. They have no special meaning.

3.4.1 Double-Operand (Format I) Instructions

Figure 3–9 illustrates the double-operand instruction format.

Figure 3–9. Double Operand Instruction Format

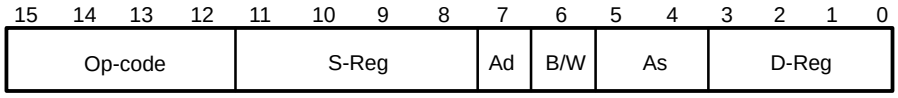


Table 3–11 lists and describes the double operand instructions.

Table 3–11. Double Operand Instructions

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

* 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.

3.4.2 Single-Operand (Format II) Instructions

Figure 3–10 illustrates the single-operand instruction format.

Figure 3–10. Single Operand Instruction Format

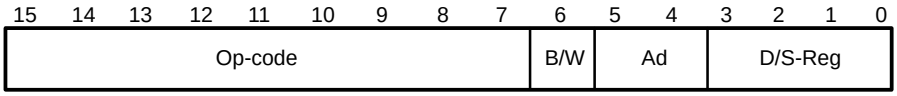


Table 3–12 lists and describes the single operand instructions.

Table 3–12. Single Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
RRC(.B)	dst	C → MSB →.....LSB → C	*	*	*	*
RRA(.B)	dst	MSB → MSB →....LSB → C	0	*	*	*
PUSH(.B)	src	SP – 2 → SP, src → @SP	–	–	–	–
SWPB	dst	Swap bytes	–	–	–	–
CALL	dst	SP – 2 → SP, PC+2 → @SP	–	–	–	–
		dst → PC				
RETI		TOS → SR, SP + 2 → SP	*	*	*	*
		TOS → PC,SP + 2 → SP				
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.

3.4.3 Jumps

Figure 3–11 shows the conditional-jump instruction format.

Figure 3–11. Jump Instruction Format

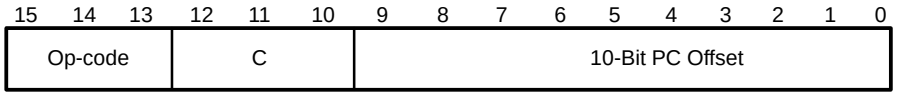


Table 3–13 lists and describes the jump instructions.

Table 3–13. 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

Conditional jumps support program branching relative to the PC and do not affect the status bits. The possible jump range is from –511 to +512 words relative to the PC value at the jump instruction. The 10-bit program-counter offset is treated as a signed 10-bit value that is doubled and added to the program counter:

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

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 Status bits are 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
```

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 Status bits are 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 The three MSBs of RAM byte TOM are set.

```
BIS.B   #0E0h,TOM       ; set the 3 MSBs in RAM location TOM
```

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 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(0) → 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
```

*** 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

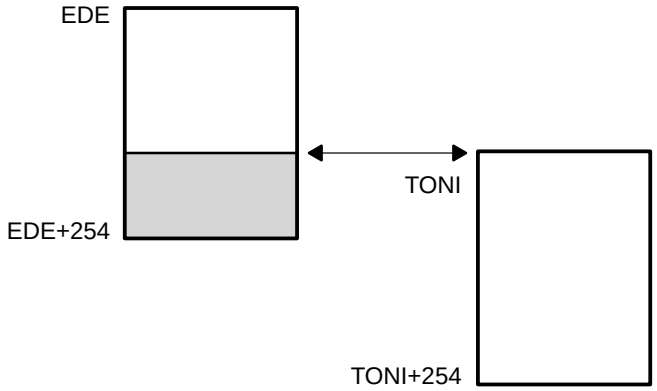
```
DEC     R10     ; Decrement R10
```

; 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 3–12.

Figure 3–12. Decrement Overlap



*** 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 Status bits are 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 a NOP instruction.

*** EINT** Enable (general) interrupts

Syntax EINT

Operation 1 → GIE
or
(0008h .OR. SR –> SR / .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 Status bits are 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 ports P1.2 to P1.7
; P1IN is the address of the register where all port bits are read. P1IFG is the address of
; the register where all interrupt events are latched.
;
                PUSH.B  &P1IN
                BIC.B   @SP,&P1IFG  ; 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 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 P1IN.1 signal is used to define or control the program flow.

```
BIT     #01h,&P1IN      ; 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 JMP 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
JLO     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 Jump if not equal
JNZ 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, R3

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 the NOP function while providing different numbers of instruction cycles and code words. Some examples are:

Examples:

```
MOV     #0,R3           ; 1 cycle, 1 word
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
```

However, care should be taken when using these examples to prevent unintended results. For example, if MOV 0(R4), 0(R4) is used and the value in R4 is 120h, then a security violation will occur with the watchdog timer (address 120h) because the security key was not used.

*** POP[.W]** Pop word from stack to destination
*** POP.B** Pop byte from stack to destination

Syntax
```
POP     dst
POP.B   dst
```

Operation
```
@SP   –> temp
SP + 2  –> SP
temp –> dst
```

Emulation
```
MOV     @SP+,dst    or    MOV.W    @SP+,dst
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 Status bits are 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.

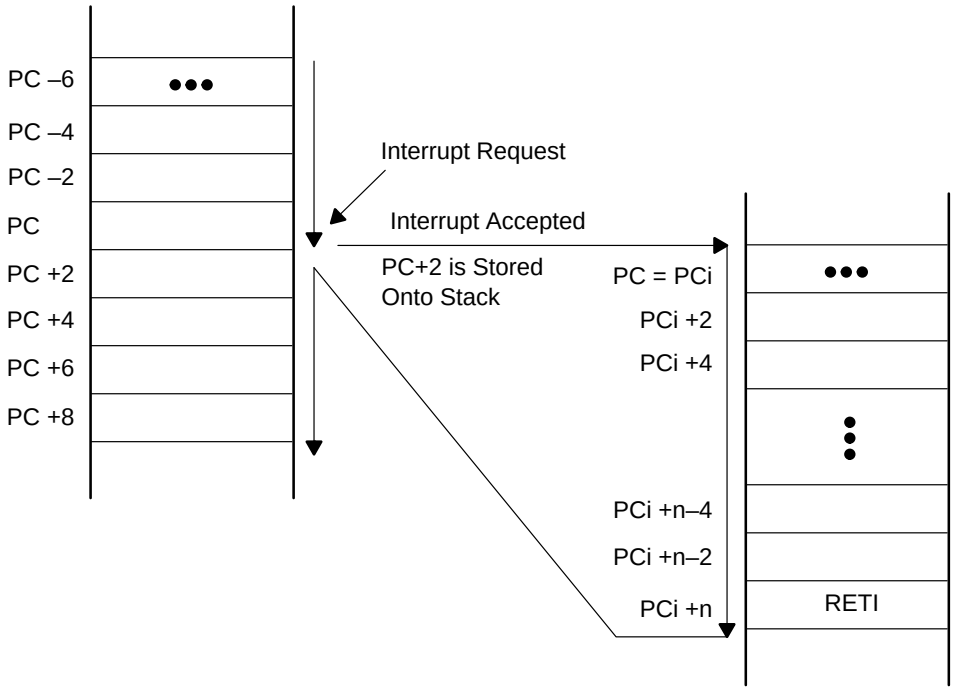
Status Bits

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 3–13 illustrates the main program interrupt.

Figure 3–13. Main Program Interrupt



*** RLA[.W]** Rotate left arithmetically
*** RLA.B** Rotate left arithmetically

Syntax
RLA dst or RLA.W dst
RLA.B dst

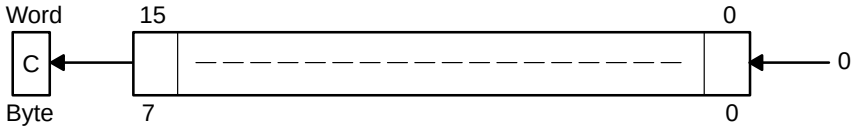
Operation
C <– MSB <– MSB–1 .... LSB+1 <– LSB <– 0

Emulation
ADD dst,dst
ADD.B dst,dst

Description
The destination operand is shifted left one position as shown in Figure 3–14. 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 3–14. 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; reset otherwise
Set if an arithmetic overflow occurs:
the initial value is 040h ≤ dst < 0C0h; reset otherwise

Mode Bits
OSCOFF, CPUOFF, and GIE are not affected.

Example
R7 is multiplied by 2.

```
RLA     R7      ; Shift left R7  (× 2)
```

Example
The low byte of R7 is multiplied by 4.

```
RLA.B   R7      ; Shift left low byte of R7  (× 2)
RLA.B   R7      ; Shift left low byte of R7  (× 4)
```

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

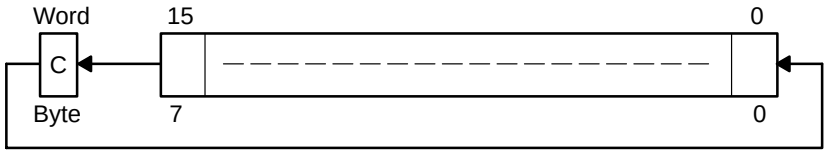
Emulation

ADDC dst,dst

Description

The destination operand is shifted left one position as shown in Figure 3–15. The carry bit (C) is shifted into the LSB and the MSB is shifted into the carry bit (C).

Figure 3–15. 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 an arithmetic overflow occurs
the initial value is 04000h ≤ dst < 0C000h; reset otherwise
Set if an arithmetic overflow occurs:
the initial value is 040h ≤ dst < 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 P1IN.1 information is shifted into the LSB of R5.

```
BIT.B   #2,&P1IN    ; 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)
```

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

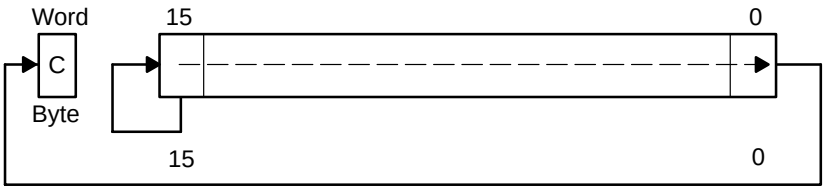
Operation

MSB –> MSB, MSB –> MSB–1, ... LSB+1 –> LSB, LSB –> C

Description

The destination operand is shifted right one position as shown in Figure 3–16. 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 3–16. 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
......
```

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
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
......
```

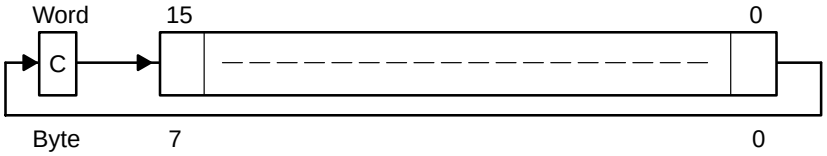
RRC[.W] Rotate right through carry
RRC.B Rotate right through carry

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 3–17. The carry bit (C) is shifted into the MSB, the LSB is shifted into the carry bit (C).

Figure 3–17. 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 source and borrow/.NOT. carry from destination
*** SBC.B** Subtract source and borrow/.NOT. carry 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: 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 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 Implementation.

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 Implementation

The borrow is treated as a .NOT. carry :

Borrow	Carry bit
Yes	0
No	1

SWPB Swap bytes

Syntax SWPB dst

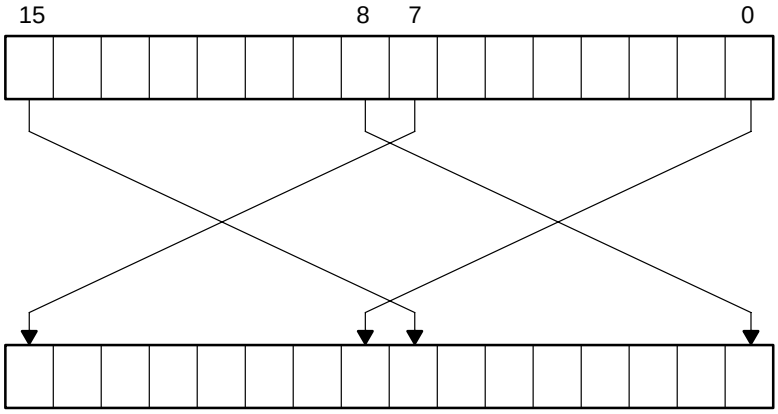
Operation Bits 15 to 8 <–> bits 7 to 0

Description The destination operand high and low bytes are exchanged as shown in Figure 3–18.

Status Bits Status bits are not affected.

Mode Bits OSCOFF, CPUOFF, and GIE are not affected.

Figure 3–18. 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

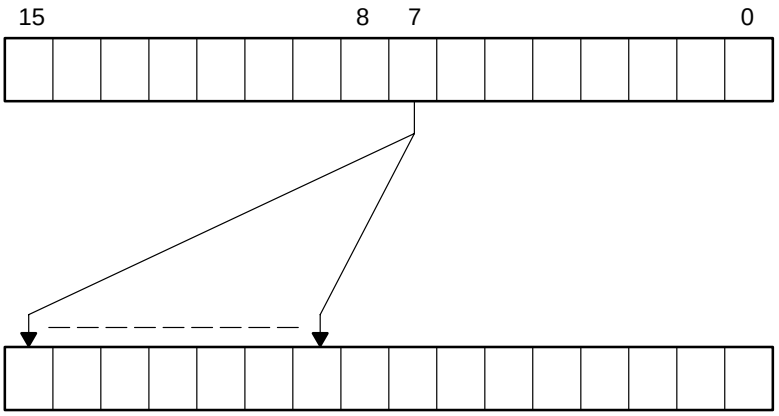
Operation Bit 7 –> Bit 8 ..... Bit 15

Description The sign of the low byte is extended into the high byte as shown in Figure 3–19.

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 3–19. Destination Operand Sign Extension



Example R7 is loaded with the P1IN 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   &P1IN,R7    ; P1IN = 080h:      . . . .  . . . . 1000 0000
SXT     R7          ; R7 = 0FF80h:      1111 1111 1000 0000
```

*** 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
```

3.4.4 Instruction Cycles and Lengths

The number of CPU clock cycles required for an instruction depends on the instruction format and the addressing modes used - not the instruction itself. The number of clock cycles refers to the MCLK.

Interrupt and Reset Cycles

Table 3–14 lists the CPU cycles for interrupt overhead and reset.

Table 3–14. Interrupt and Reset Cycles

Action	No. of Cycles	Length of Instruction
Return from interrupt (RETI)	5	1
Interrupt accepted	6	–
WDT reset	4	–
Reset ($\overline{\text{RST}}$/NMI)	4	–

Format-II (Single Operand) Instruction Cycles and Lengths

Table 3–15 lists the length and CPU cycles for all addressing modes of format-II instructions.

Table 3–15. Format-II Instruction Cycles and Lengths

Addressing Mode	No. of Cycles			Length of Instruction	Example
	RRA, RRC SWPB, SXT	PUSH	CALL		
Rn	1	3	4	1	SWPB R5
@Rn	3	4	4	1	RRC @R9
@Rn+	3	4	5	1	SWPB @R10+
#N	(See note)	4	5	2	CALL #81H
X(Rn)	4	5	5	2	CALL 2(R7)
EDE	4	5	5	2	PUSH EDE
&EDE	4	5	5	2	SXT &EDE

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 results in an unpredictable program operation.

Format-III (Jump) Instruction Cycles and Lengths

All jump instructions require one code word, and take two CPU cycles to execute, regardless of whether the jump is taken or not.

Format-I (Double Operand) Instruction Cycles and Lengths

Table 3–16 lists the length and CPU cycles for all addressing modes of format-I instructions.

Table 3–16. Format 1 Instruction Cycles and Lengths

Addressing Mode		No. of Cycles	Length of Instruction	Example	
Src	Dst				
Rn	Rm	1	1	MOV	R5,R8
	PC	2	1	BR	R9
	x(Rm)	4	2	ADD	R5,3(R6)
	EDE	4	2	XOR	R8,EDE
	&EDE	4	2	MOV	R5,&EDE
@Rn	Rm	2	1	AND	@R4,R5
	PC	3	1	BR	@R8
	x(Rm)	5	2	XOR	@R5,8(R6)
	EDE	5	2	MOV	@R5,EDE
	&EDE	5	2	XOR	@R5,&EDE
@Rn+	Rm	2	1	ADD	@R5+,R6
	PC	3	1	BR	@R9+
	x(Rm)	5	2	XOR	@R5,8(R6)
	EDE	5	2	MOV	@R9+,EDE
	&EDE	5	2	MOV	@R9+,&EDE
#N	Rm	2	2	MOV	#20,R9
	PC	3	2	BR	#2AEh
	x(Rm)	5	3	MOV	#0300h,0(SP)
	EDE	5	3	ADD	#33,EDE
	&EDE	5	3	ADD	#33,&EDE
x(Rn)	Rm	3	2	MOV	2(R5),R7
	PC	3	2	BR	2(R6)
	TONI	6	3	MOV	4(R7),TONI
	x(Rm)	6	3	ADD	3(R4),6(R9)
	&TONI	6	3	MOV	3(R4),&TONI
EDE	Rm	3	2	AND	EDE,R6
	PC	3	2	BR	EDE
	TONI	6	3	CMP	EDE,TONI
	x(Rm)	6	3	MOV	EDE,0(SP)
	&TONI	6	3	MOV	EDE,&TONI
&EDE	Rm	3	2	MOV	&EDE,R8
	PC	3	2	BRA	&EDE
	TONI	6	3	MOV	&EDE,TONI
	x(Rm)	6	3	MOV	&EDE,0(SP)
	&TONI	6	3	MOV	&EDE,&TONI

3.4.5 Instruction Set Description

The instruction map is shown in Figure 3–20 and the complete instruction set is summarized in Table 3–17.

Figure 3–20. Core Instruction Map

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx																
4xxx																
8xxx																
Cxxx																
1xxx	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI			
14xx																
18xx																
1Cxx																
20xx	JNE/JNZ															
24xx	JEQ/JZ															
28xx	JNC															
2Cxx	JC															
30xx	JN															
34xx	JGE															
38xx	JL															
3Cxx	JMP															
4xxx	MOV, MOV.B															
5xxx	ADD, ADD.B															
6xxx	ADDC, ADDC.B															
7xxx	SUBC, SUBC.B															
8xxx	SUB, SUB.B															
9xxx	CMP, CMP.B															
Axxx	DADD, DADD.B															
Bxxx	BIT, BIT.B															
Cxxx	BIC, BIC.B															
Dxxx	BIS, BIS.B															
Exxx	XOR, XOR.B															
Fxxx	AND, AND.B															

Table 3–17. MSP430 Instruction Set

Mnemonic		Description		V	N	Z	C
ADC(.B)*	dst	Add C to destination	dst + C → dst	*	*	*	*
ADD(.B)	src,dst	Add source to destination	src + dst → dst	*	*	*	*
ADDC(.B)	src,dst	Add source and C to destination	src + dst + C → dst	*	*	*	*
AND(.B)	src,dst	AND source and destination	src .and. dst → dst	0	*	*	*
BIC(.B)	src,dst	Clear bits in destination	.not.src .and. dst → dst	–	–	–	–
BIS(.B)	src,dst	Set bits in destination	src .or. dst → dst	–	–	–	–
BIT(.B)	src,dst	Test bits in destination	src .and. dst	0	*	*	*
BR*	dst	Branch to destination	dst → PC	–	–	–	–
CALL	dst	Call destination	PC+2 → stack, dst → PC	–	–	–	–
CLR(.B)*	dst	Clear destination	0 → dst	–	–	–	–
CLRC*		Clear C	0 → C	–	–	–	0
CLRN*		Clear N	0 → N	–	0	–	–
CLRZ*		Clear Z	0 → Z	–	–	0	–
CMP(.B)	src,dst	Compare source and destination	dst – src	*	*	*	*
DADC(.B)*	dst	Add C decimally to destination	dst + C → dst (decimally)	*	*	*	*
DADD(.B)	src,dst	Add source and C decimally to dst.	src + dst + C → dst (decimally)	*	*	*	*
DEC(.B)*	dst	Decrement destination	dst – 1 → dst	*	*	*	*
DECD(.B)*	dst	Double-decrement destination	dst – 2 → dst	*	*	*	*
DINT*		Disable interrupts	0 → GIE	–	–	–	–
EINT*		Enable interrupts	1 → GIE	–	–	–	–
INC(.B)*	dst	Increment destination	dst +1 → dst	*	*	*	*
INCD(.B)*	dst	Double-increment destination	dst+2 → dst	*	*	*	*
INV(.B)*	dst	Invert destination	.not.dst → dst	*	*	*	*
JC/JHS	label	Jump if C set/Jump if higher or same		–	–	–	–
JEQ/JZ	label	Jump if equal/Jump if Z set		–	–	–	–
JGE	label	Jump if greater or equal		–	–	–	–
JL	label	Jump if less		–	–	–	–
JMP	label	Jump	PC + 2 x offset → PC	–	–	–	–
JN	label	Jump if N set		–	–	–	–
JNC/JLO	label	Jump if C not set/Jump if lower		–	–	–	–
JNE/JNZ	label	Jump if not equal/Jump if Z not set		–	–	–	–
MOV(.B)	src,dst	Move source to destination	src → dst	–	–	–	–
NOP*		No operation		–	–	–	–
POP(.B)*	dst	Pop item from stack to destination	@SP → dst, SP+2 → SP	–	–	–	–
PUSH(.B)	src	Push source onto stack	SP – 2 → SP, src → @SP	–	–	–	–
RET*		Return from subroutine	@SP → PC, SP + 2 → SP	–	–	–	–
RETI		Return from interrupt		*	*	*	*
RLA(.B)*	dst	Rotate left arithmetically		*	*	*	*
RLC(.B)*	dst	Rotate left through C		*	*	*	*
RRA(.B)	dst	Rotate right arithmetically		0	*	*	*
RRC(.B)	dst	Rotate right through C		*	*	*	*
SBC(.B)*	dst	Subtract not(C) from destination	dst + 0FFFFh + C → dst	*	*	*	*
SETC*		Set C	1 → C	–	–	–	1
SETN*		Set N	1 → N	–	1	–	–
SETZ*		Set Z	1 → C	–	–	1	–
SUB(.B)	src,dst	Subtract source from destination	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src,dst	Subtract source and not(C) from dst.	dst + .not.src + C → dst	*	*	*	*
SWPB	dst	Swap bytes		–	–	–	–
SXT	dst	Extend sign		0	*	*	*
TST(.B)*	dst	Test destination	dst + 0FFFFh + 1	0	*	*	1
XOR(.B)	src,dst	Exclusive OR source and destination	src .xor. dst → dst	*	*	*	*

* Emulated Instruction

Chapter 4

Basic Clock Module

The basic clock module provides the clocks for MSP430x1xx devices. This chapter describes the operation of the basic clock module. The basic clock module is implemented in all MSP430x1xx devices.

Topic **Page**

4.1 Basic Clock Module Introduction

The basic clock module supports low system cost and ultralow-power consumption. Using three internal clock signals, the user 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, or with resonators, under full software control.

The basic clock module includes two or three clock sources:

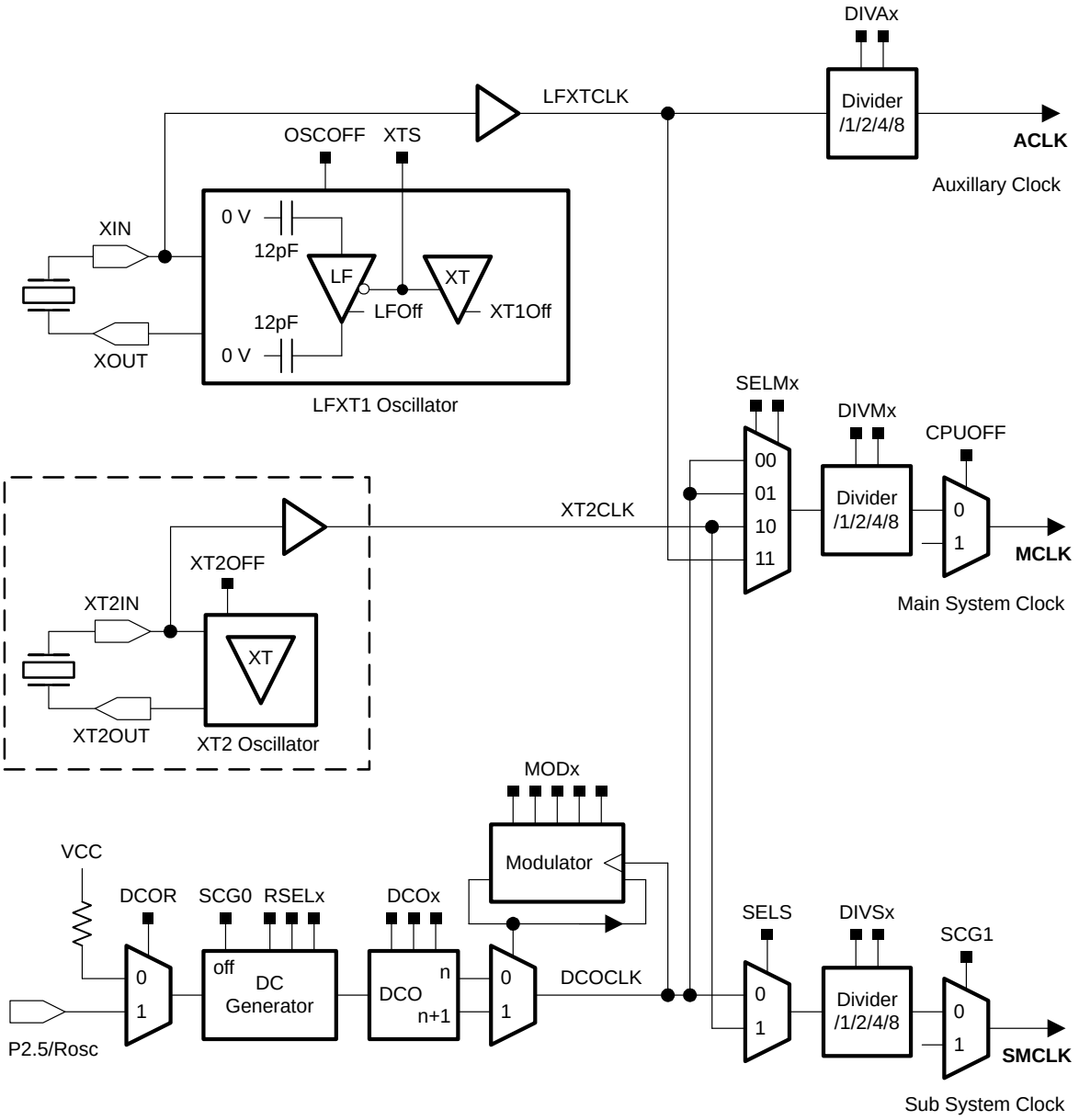
- LFXT1CLK: Low-frequency/high-frequency oscillator that can be used either with low-frequency 32,768-Hz watch crystals, or standard crystals, resonators, or external clock sources in the 450-kHz to 8-MHz range.
- XT2CLK: Optional high-frequency oscillator that can be used with standard crystals, resonators, or external clock sources in the 450-kHz to 8-MHz range.
- DCOCLK: Internal 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. 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. MCLK is used by the CPU and system.
- SMCLK: Sub-main clock. SMCLK is software selectable as LFXT1CLK, XT2CLK (if available), or DCOCLK. SMCLK is divided by 1, 2, 4, or 8. SMCLK is software selectable for individual peripheral modules.

The block diagram of the basic clock module is shown in Figure 4–1.

Figure 4–2. Basic Clock Block Diagram



Note: XT2 Oscillator

The XT2 Oscillator is not present on MSP430x11xx or MSP430x12xx devices. The LFXT1CLK is used in place of XT2CLK.

4.2 Basic Clock Module Operation

After a PUC, MCLK and SMCLK are sourced from DCOCLK at ~800 kHz (see device-specific datasheet for parameters) and ACLK is sourced from LFXT1 in LF mode.

Status register control bits SCG0, SCG1, OSCOFF, and CPUOFF configure the MSP430 operating modes and enable or disable portions of the basic clock module. See Chapter *System Resets, Interrupts and Operating Modes*. The DCOCTL, BCSCTL1, and BCSCTL2 registers configure the basic clock module

The basic clock can be configured or reconfigured by software at any time during program execution, for example:

```
BIS.B #RSEL2+RSEL1+RSEL0,&BCSCTL1 ;
BIS.B #DCO2+DCO1+DCO0,&DCOCTL     ; Set max DCO frequency
```

4.2.1 Basic Clock Module 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 user 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-power 32,786-Hz watch crystal, 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 that can be only activated when requested by interrupt-driven 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.

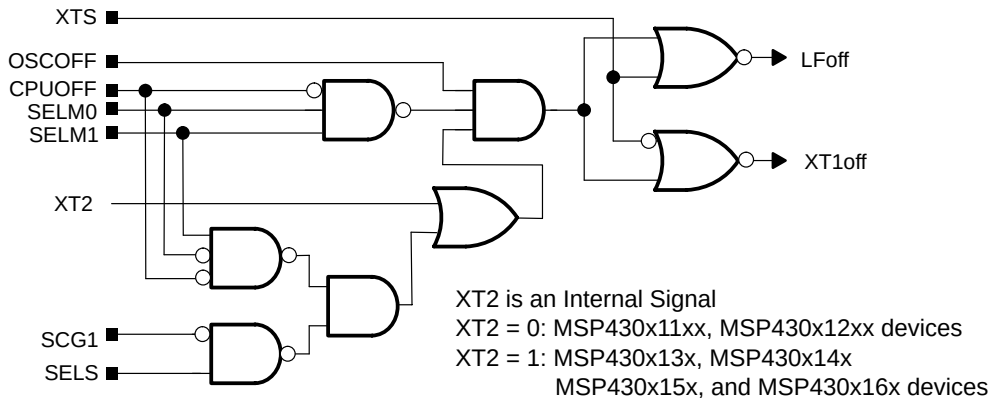
4.2.2 LFXT1 Oscillator

The LFXT1 oscillator supports ultralow-current consumption using a 32,768-Hz watch crystal in LF mode (XTS = 0). A watch crystal connects to XIN and XOUT without any other external components. Internal 12-pF load capacitors are provided for LFXT1 in LF mode. The capacitors add serially, providing a match for standard 32,768-Hz crystals requiring a 6-pF load. Additional capacitors can be added if necessary.

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

Software can disable LFXT1 by setting OSCOFF, if this signal does not source SMCLK or MCLK, as shown in Figure 4–3.

Figure 4–3. Off Signals for the LFXT1 Oscillator



Note: LFXT1 Oscillator Characteristics

Low-frequency crystals often require hundreds of milliseconds to start up, depending on the crystal. It is recommended to leave the LFXT1 oscillator on when in LF mode.

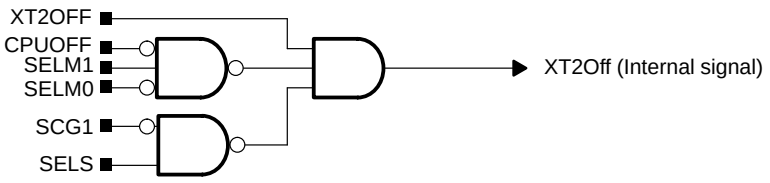
Ultralow-power oscillators such as the LFXT1 in LF mode should be guarded from noise coupling from other sources. The crystal should be placed as close as possible to the MSP430 with the crystal housing grounded and the crystal traces guarded with ground traces.

The LFXT1 oscillator in LF mode requires a 5.1-MΩ resistor from XOUT to V_{SS} when V_{CC} < 2.5 V.

4.2.3 XT2 Oscillator

Some devices have a second crystal oscillator, XT2. XT2 sources XT2CLK and its characteristics are identical to LFXT1 in HF mode. The XT2OFF bit disables the XT2 oscillator if XT2CLK is not used for MCLK or SMCLK as shown in Figure 4–4.

Figure 4–4. Off Signals for Oscillator XT2



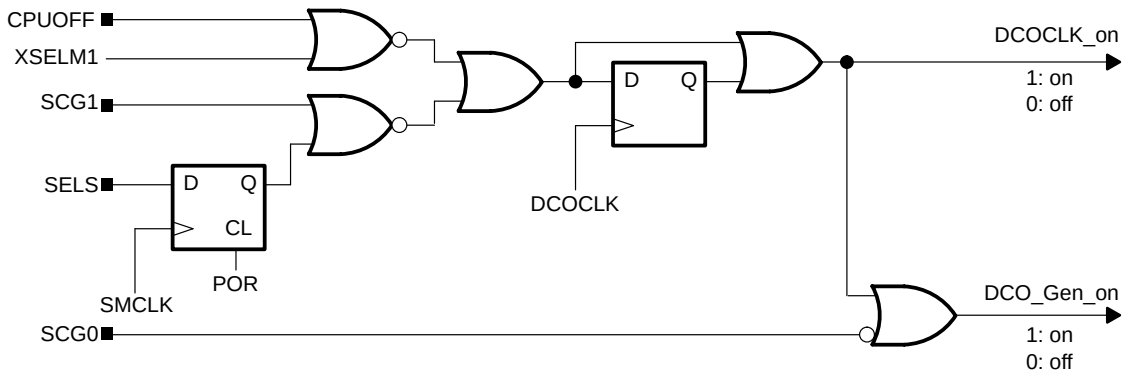
4.2.4 Digitally-Controlled Oscillator (DCO)

The DCO is an integrated ring oscillator with RC-type characteristics. As with any RC-type oscillator, frequency varies with temperature, voltage, and from device to device. The DCO frequency can be adjusted by software using the DCOx, MODx, and RSELx bits. The digital control of the oscillator allows frequency stabilization despite its RC-type characteristics.

Disabling the DCO

Software can disable DCOCLK when not used to source SMCLK or MCLK, as shown in Figure 4–5.

Figure 4–5. On/Off Control of DCO



Adjusting the DCO frequency

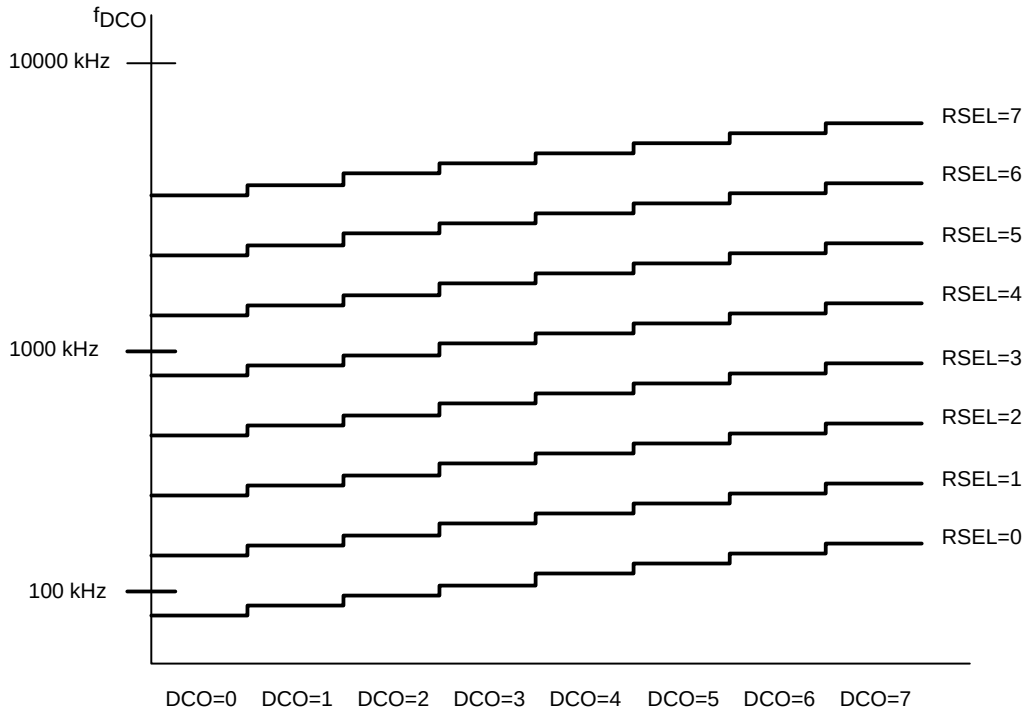
After a PUC, the internal resistor is selected for the DC generator, RSELx = 4, and DCOx = 3, allowing the DCO to start at a mid-range frequency. MCLK and SMCLK are sourced from DCOCLK. Because the CPU executes code from MCLK, which is sourced from the fast-starting DCO, code execution begins from PUC in less than 6 μs.

The frequency of DCOCLK is set by the following functions:

- The current injected into the DC generator by either the internal or external resistor defines the fundamental frequency. The DCOR bit selects the internal or external resistor.
- The three RSELx bits select one of eight nominal frequency ranges for the DCO. These ranges are defined for an individual device in the device-specific data sheet.
- The three DCOx bits divide the DCO range selected by the RSELx bits into 8 frequency steps, separated by approximately 10%.
- The five MODx bits, switch between the frequency selected by the DCOx bits and the next higher frequency set by DCO+1.

The DCOx and RSELx ranges and steps are shown in Figure 4–6.

Figure 4–6. DCOx Range and RSELx Steps

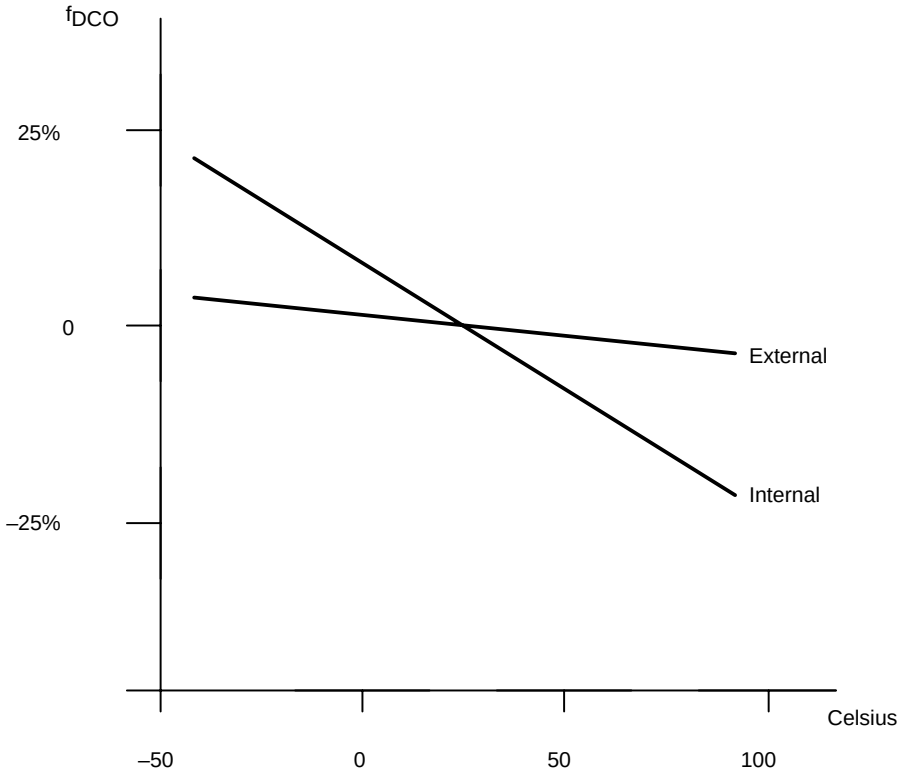


Using an External Resistor (R_{OSC}) for the DCO

The DCO temperature coefficient can be reduced by using an external resistor R_{OSC} to source the current for the DC generator. Figure 4–7 shows the typical relationship of f_{DCO} vs. temperature for both the internal and external resistor options. Using an external R_{OSC} reduces the DCO temperature coefficient to approximately –0.05%/C. See the device-specific data sheet for parameters.

R_{OSC} also allows the DCO to operate at higher frequencies. For example, the internal resistor nominal value is approximately 200 kΩ, allowing the DCO to operate up to approximately 5 MHz. When using an external R_{OSC} of approximately 100 kΩ the DCO can operate up to approximately 10 MHz. The user should take care to not exceed the maximum MCLK frequency specified in the datasheet, even though the DCO is capable of exceeding it.

Figure 4–7. DCO Frequency vs. Temperature



4.2.5 DCO Modulator

The modulator mixes two DCO frequencies, f_{DCO} and f_{DCO+1} to produce an intermediate effective frequency between f_{DCO} and f_{DCO+1} and spread the clock energy, reducing electromagnetic interference (EMI). The modulator mixes f_{DCO} and f_{DCO+1} for 32 DCOCLK clock cycles and is configured with the MODx bits. When MODx = 0 the modulator is off.

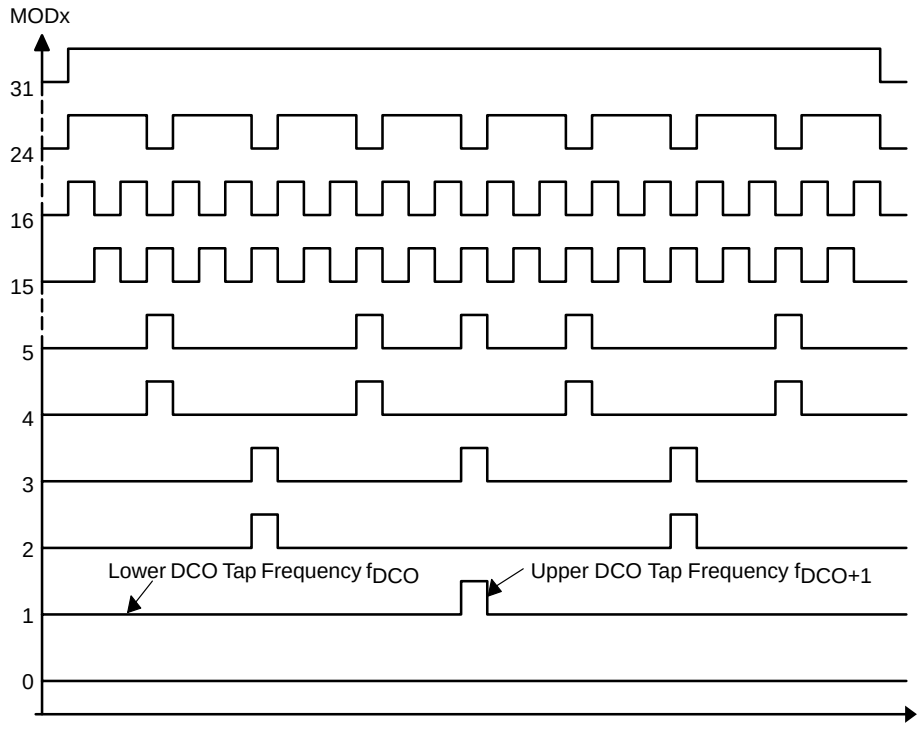
The modulator mixing formula is:

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

Because f_{DCO} is lower than the effective frequency and f_{DCO+1} is higher than the effective frequency, the error of the effective frequency integrates to zero. It does not accumulate. The error of the effective frequency is zero every 32 DCOCLK cycles. Figure 4–8 illustrates the modulator operation.

The modulator settings and DCO control are configured with software. The DCOCLK can be compared to a stable frequency of known value and adjusted with the DCOx, RSELx, and MODx bits. See http://www.ti.com/sc/msp430 for application notes and example code on configuring the DCO.

Figure 4–8. Modulator Patterns



4.2.6 Basic Clock Module Fail-Safe Operation

The basic clock module incorporates an oscillator-fault detection fail-safe feature. The oscillator fault detector is an analog circuit that monitors the LFXT1CLK (in HF mode) and the XT2CLK. An oscillator fault is detected when either clock signal is not present for approximately 50 µs. When an oscillator fault is detected, and when MCLK is sourced from either LFXT1 in HF mode or XT2, MCLK is automatically switched to the DCO for its clock source. This allows code execution to continue, even though the crystal oscillator has stopped.

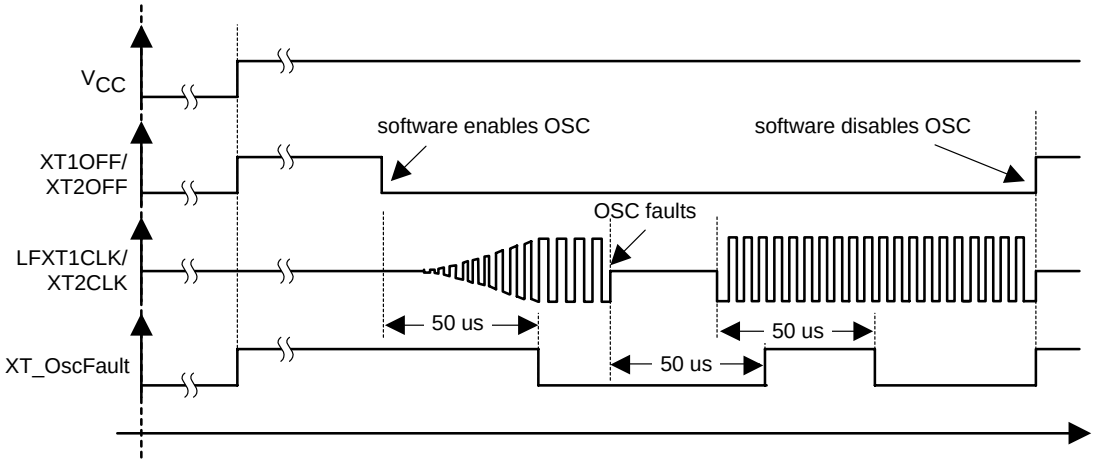
When OFIFG is set and OFIE is set, an NMI interrupt is requested. The NMI interrupt service routine can test the OFIFG flag to determine if an oscillator fault occurred. The OFIFG flag must be cleared by software.

Note: No Oscillator Fault Detection for LFXT1 in LF Mode

Oscillator fault detection is only applicable for LFXT1 in HF mode and XT2. There is no oscillator fault detection for LFXT1 in LF mode.

OFIFG is set by the oscillator fault signal, XT_OscFault. XT_OscFault is set at POR, when LFXT1 has an oscillator fault in HF mode, or when XT2 has an oscillator fault. When XT2 or LFXT1 in HF mode is stopped with software the XT_OscFault signal becomes active immediately, remains active until the oscillator is re-started, and becomes inactive approximately 50 µs after the oscillator re-starts as shown in Figure 4–9.

Figure 4–10. Oscillator-Fault Signal



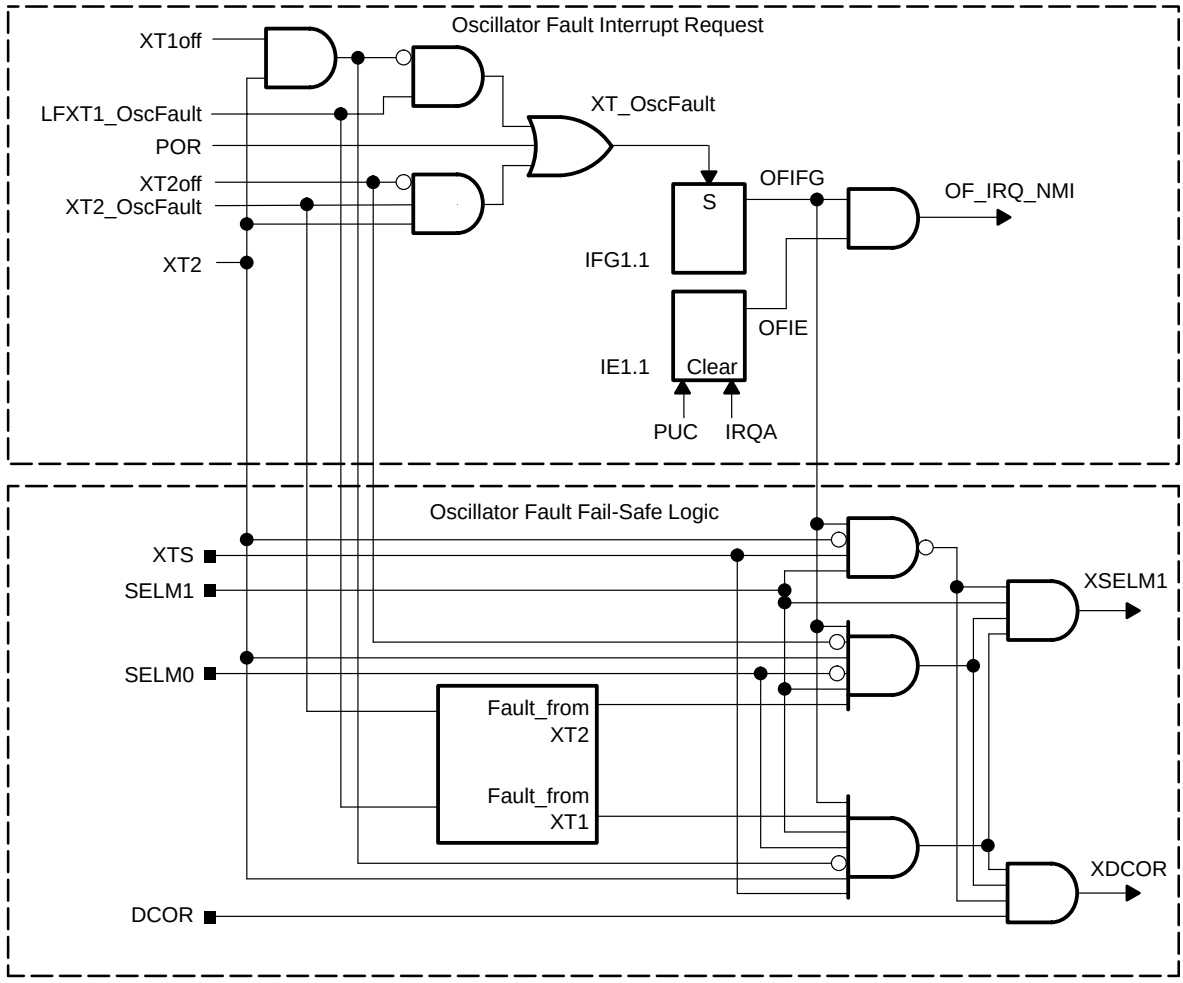
Oscillator Fault Detection

Signal XT_OscFault triggers the OFIFG flag as shown in Figure 4–11. The LFXT1_OscFault signal is low when LFXT1 is in LF mode.

On devices without XT2, the OFIFG flag cannot be cleared when LFXT1 is in LF mode. MCLK may be sourced by LFXT1CLK in LF mode by setting the SELMx bits, even though OFIFG remains set.

On devices with XT2, the OFIFG flag can be cleared by software when LFXT1 is in LF mode and it remains cleared. MCLK may be sourced by LFXT1CLK in LF mode regardless of the state of the OFIFG flag.

Figure 4–11. Oscillator-Fault-Interrupt



XT2 Is an internal signal. XT2 = 0 on devices without XT2 (MSP430x11xx and MSP430x12xx).
XT2 = 1 on devices with XT2 (MSP430F13x, MSP430F14x, MSP430F15x, and(MSP430F16x)
IRQA: Interrupt request accepted
LFXT1_OscFault: Only applicable to LFXT1 oscillator in HF mode.

Sourcing MCLK from a Crystal

After a PUC, the basic clock module uses DCOCLK for MCLK. If required, MCLK may be sourced from LFXT1 or XT2.

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

1) Switch on the crystal oscillator

2) Clear the OFIFG flag

3) Wait at least 50 µs

4) Test OFIFG, and repeat steps 1-4 until OFIFG remains cleared.

```
; Select LFXT1 (HF mode) for MCLK
   BIC   #OSCOFF,SR            ; Turn on osc.
   BIS.B #XTS,BCSCTL1          ; HF mode
L1 BIC.B #OFIFG,&IFG1          ; Clear OFIFG
   MOV   #0FFh,R15             ; Delay
L2 DEC   R15                   ;
   JNZ   L2                    ;
   BIT.B #OFIFG,&IFG1          ; Re-test OFIFG
   JNZ   L1                    ; Repeat test if needed
   BIS.B #SELM1+SELM0,&BCSCTL2 ; Select LFXT1CLK
```

4.2.7 Synchronization of Clock Signals

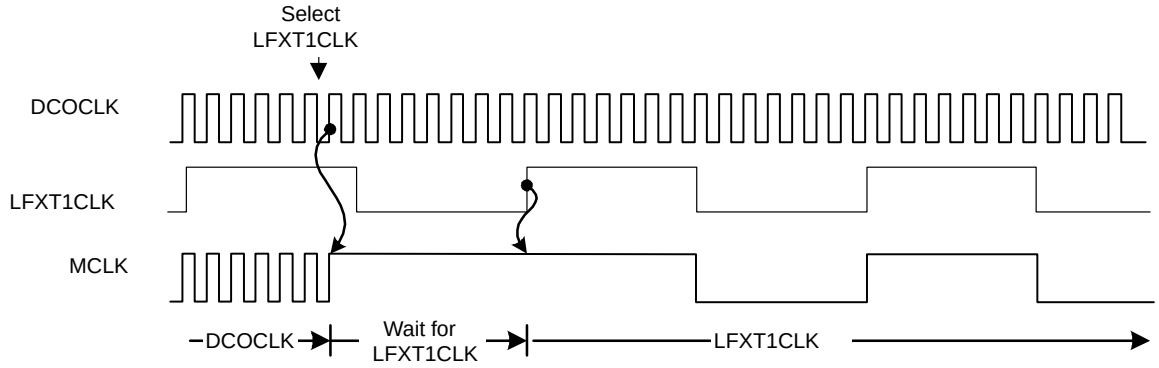
When switching MCLK or SMCLK from one clock source to the another, the switch is synchronized to avoid critical race conditions as shown in Figure 4–12:

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

2) The clock remains high until the next rising edge of the new clock.

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

Figure 4–12. Switch MCLK from DCOCLK to LFXT1CLK



4.3 Basic Clock Module Registers

The basic clock module registers are listed in Table 4–1:

Table 4–1. Basic Clock Module Registers

Register	Short Form	Register Type	Address	Initial State
DCO control register	DCOCTL	Read/write	056h	056h with PUC
Basic clock system control 1	BCSCTL1	Read/write	057h	084h with PUC
Basic clock system control 2	BCSCTL2	Read/write	058h	Reset with POR
SFR interrupt enable register 1	IE1	Read/write	0000h	Reset with PUC
SFR interrupt flag register 1	IFG1	Read/write	0002h	Reset with PUC

DCOCTL, DCO Control Register

7	6	5	4	3	2	1	0
DCOx			MODx				
rw–0	rw–1	rw–1	rw–0	rw–0	rw–0	rw–0	rw–0

DCOx	Bits 7-5	DCO frequency select. These bits select which of the eight discrete DCO frequencies of the RSELx setting is selected.
MODx	Bits 4-0	Modulator selection. These bits define how often the f_{DCO+1} frequency is used within a period of 32 DCOCLK cycles. During the remaining clock cycles (32–MOD) the f_{DCO} frequency is used. Not useable when DCOx=7.

BCSCTL1, Basic Clock System Control Register 1

7	6	5	4	3	2	1	0
XT2OFF	XTS	DIVAx		XT5V	RSELx		
rw–(1)	rw–(0)	rw–(0)	rw–(0)	rw–0	rw–1	rw–0	rw–0

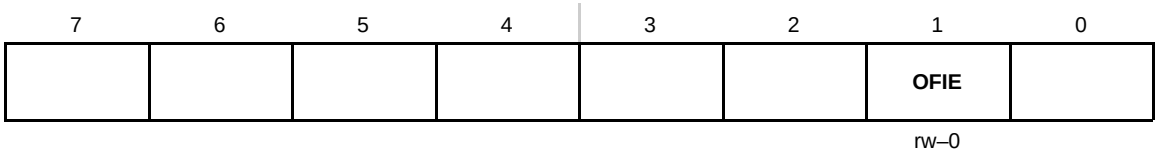
XT2OFF	Bit 7	XT2 off. This bit turns off the XT2 oscillator 0 XT2 is on 1 XT2 is off if it is not used for MCLK or SMCLK.
XTS	Bit 6	LFXT1 mode select. 0 Low frequency mode 1 High frequency mode
DIVAx	Bits 5-4	Divider for ACLK 00 /1 01 /2 10 /4 11 /8
XT5V	Bit 3	Unused. XT5V should always be reset.
RSELx	Bits 2-0	Resistor Select. The internal resistor is selected in eight different steps. The value of the resistor defines the nominal frequency. The lowest nominal frequency is selected by setting RSELx=0.

BCSCTL2, Basic Clock System Control Register 2

7	6	5	4	3	2	1	0
SELMx		**DIVMx**		**SELS**	**DIVSx**		**DCOR**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–0	rw–0	rw–0	rw–0

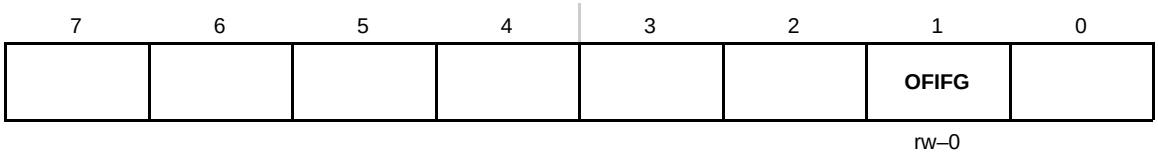
SELMx	Bits 7-6	Select MCLK. These bits select the MCLK source. 00 DCOCLK 01 DCOCLK 10 XT2CLK when XT2 present. LFXT1CLK when XT2 not present 11 LFXT1CLK
DIVMx	BitS 5-4	Divider for MCLK 00 /1 01 /2 10 /4 11 /8
SELS	Bit 3	Select SMCLK. This bit selects the SMCLK source. 0 DCOCLK 1 XT2CLK when XT2 present. LFXT1CLK when XT2 not present
DIVSx	BitS 2-1	Divider for SMCLK 00 /1 01 /2 10 /4 11 /8
DCOR	Bit 0	DCO resistor select. 0 Internal resistor 1 External resistor

IE1, Interrupt Enable Register 1



	Bits 7-2	These bits may be used by other modules. See device-specific datasheet.
OFIE	Bit 1	Oscillator fault interrupt enable. This bit enables the OFIFG interrupt. Because other bits in IE1 may be used for other modules, it is recommended to set or clear this bit using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions. 0 Interrupt not enabled 1 Interrupt enabled
	Bits 0	This bit may be used by other modules. See device-specific datasheet.

IFG1, Interrupt Flag Register 1



	Bits 7-2	These bits may be used by other modules. See device-specific datasheet.
OFIFG	Bit 1	Oscillator fault interrupt flag. Because other bits in IFG1 may be used for other modules, it is recommended to set or clear this bit using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions. 0 No interrupt pending 1 Interrupt pending
	Bits 0	This bit may be used by other modules. See device-specific datasheet.

Chapter 5

Flash Memory Controller

This chapter describes the operation of the MSP430 flash memory controller.

Topic **Page**

5.1 Flash Memory Introduction

The MSP430 flash memory is bit-, byte-, and word-addressable and programmable. The flash memory module has an integrated controller that controls programming and erase operations. The controller has three registers, a timing generator, and a voltage generator to supply program and erase voltages.

MSP430 flash memory features include:

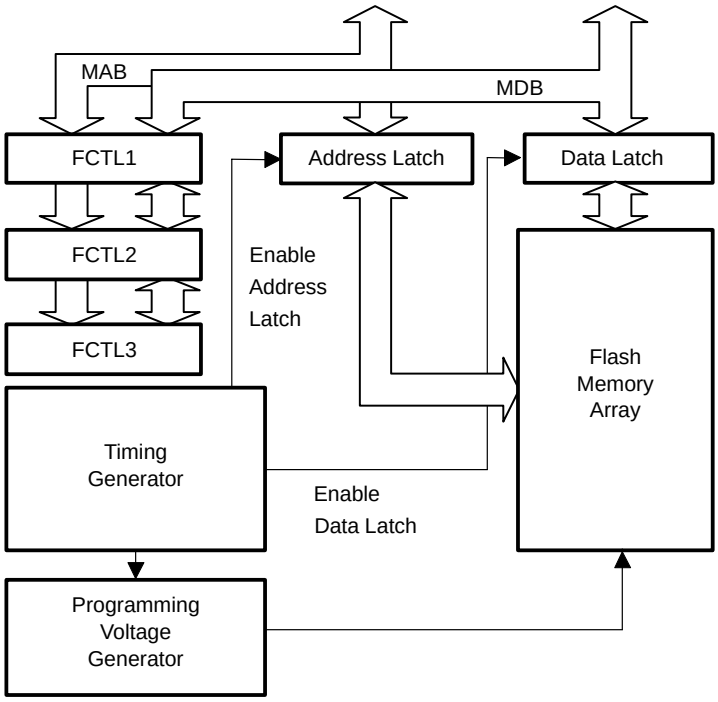
- Internal programming voltage generation
- Bit, byte or word programmable
- Ultralow-power operation
- Segment erase and mass erase

The block diagram of the flash memory and controller is shown in Figure 5–1.

Note: Minimum V_{CC} During Flash Write or Erase

The minimum V_{CC} voltage during a flash write or erase operation is 2.7 V. If V_{CC} falls below 2.7 V during a write or erase, the result of the write or erase will be unpredictable.

Figure 5–1. Flash Memory Module Block Diagram



5.2 Flash Memory Segmentation

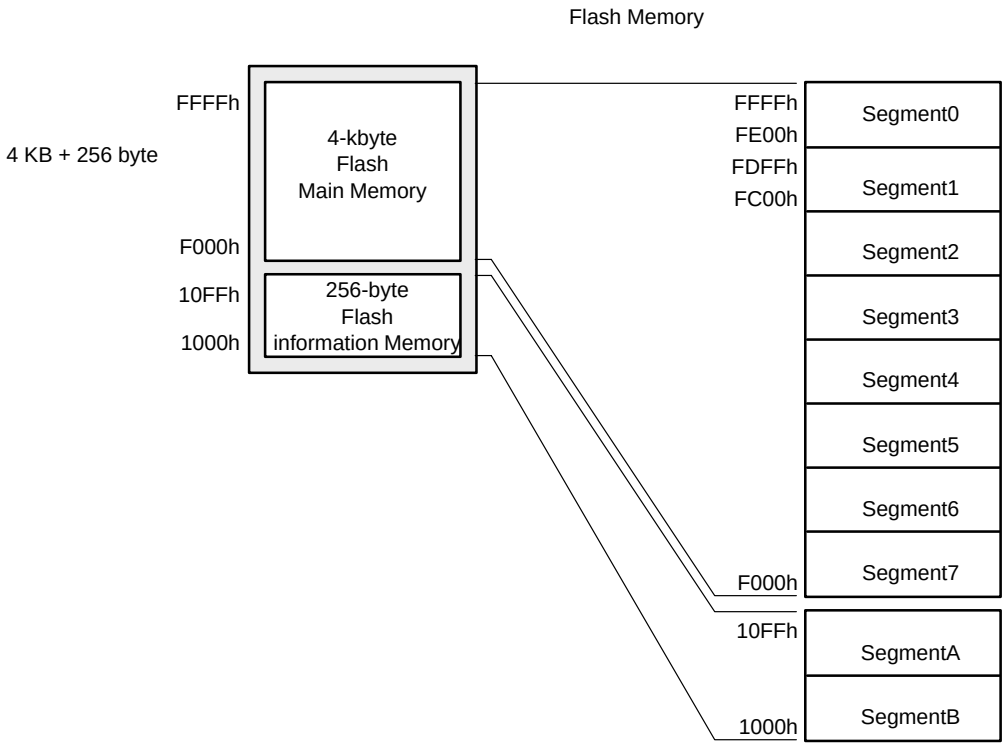
MSP430 flash memory is partitioned into segments. Single bits, bytes, or words can be written to flash memory, but the segment is the smallest size of flash memory that can be erased. Three erase modes provide the ability to erase a single segment, erase all main segments, or erase all segments (main and information segments).

The flash memory is partitioned into main and information memory sections. There is no difference in the operation of the main and information memory sections. Code or data can be located in either section. The differences between the two sections are the segment size and the physical addresses.

The information memory has two 128-byte segments (MSP430x1101 devices have only one). The main memory has two or more 512-byte segments. See the device-specific datasheet for the complete memory map of a device.

Figure 5–2 shows the flash segmentation using an example of 4-KB flash that has eight main segments and both information segments.

Figure 5–2. Flash Memory Segments, 4-KB Example



5.3 Flash Memory Operation

The default mode of the flash memory is read mode. In read mode, the flash memory is not being erased or written, the flash timing generator and voltage generator are off, and the memory operates identically to ROM.

MSP430 flash memory is in-system programmable (ISP) without the need for additional external voltage. The CPU can program its own flash memory. The flash memory write/erase modes are selected with the BLKWRT, WRT, MERAS, and ERASE bits and are:

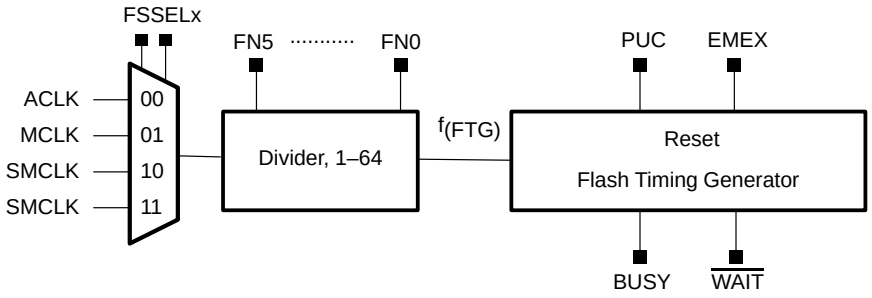
- Byte/word write
- Block write
- Segment Erase
- Mass Erase (all main memory segments)
- All Erase (all segments)

Reading or writing to flash memory while it is being programmed or erased is prohibited. If CPU execution is required during the write or erase, the code to be executed must be in RAM. Any flash update can be initiated from within flash memory or RAM.

5.3.1 Flash Memory Timing Generator

Write and erase operations are controlled by the flash timing generator shown in Figure 5–3. The flash timing generator operating frequency, $f_{(FTG)}$, must be in the range from ~ 257 kHz to ~ 476 kHz (see device-specific datasheet).

Figure 5–3. Flash Memory Timing Generator Block Diagram



The flash timing generator can be sourced from ACLK, SMCLK, or MCLK. The selected clock source should be divided using the FNx bits to meet the frequency requirements for $f_{(FTG)}$. If the $f_{(FTG)}$ frequency deviates from the specification during the write or erase operation, the result of the write or erase may be unpredictable, or the flash memory may be stressed above the limits of reliable operation.

5.3.2 Erasing Flash Memory

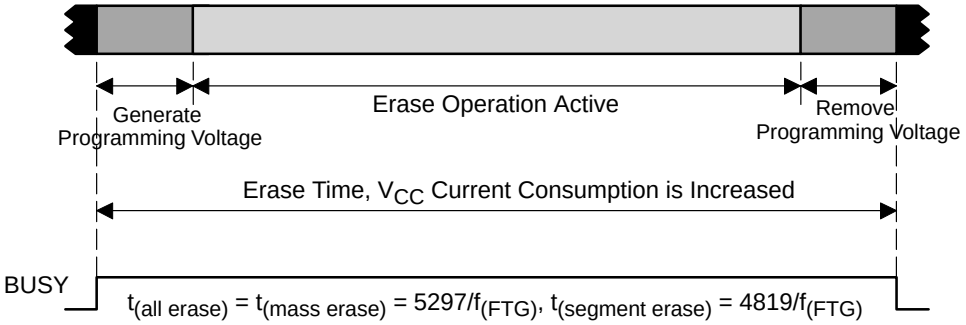
The erased level of a flash memory bit is 1. Each bit can be programmed from 1 to 0 individually but to reprogram from 0 to 1 requires an erase cycle. The smallest amount of flash that can be erased is a segment. There are three erase modes selected with the ERASE and MERAS bits listed in Table 5–1.

Table 5–1. Erase Modes

MERAS	ERASE	Erase Mode
0	1	Segment erase
1	0	Mass erase (all main memory segments)
1	1	Erase all flash memory (main and information segments)

Any erase is initiated by a dummy write into the address range to be erased. The dummy write starts the flash timing generator and the erase operation. Figure 5–4 shows the erase cycle timing. The BUSY bit is set immediately after the dummy write and remains set throughout the erase cycle. BUSY, MERAS, and ERASE are automatically cleared when the cycle completes. The erase cycle timing is not dependent on the amount of flash memory present on a device. Erase cycle times are equivalent for all MSP430 devices.

Figure 5–4. Erase Cycle Timing



A dummy write to an address not in the range to be erased does not start the erase cycle, does not affect the flash memory, and is not flagged in any way. This errant dummy write is ignored.

Interrupts should be disabled before a flash erase cycle. After the erase cycle has completed, interrupts may be re-enabled. Any interrupt that occurred during the erase cycle will have its associated flag set, and will generate an interrupt request when re-enabled.

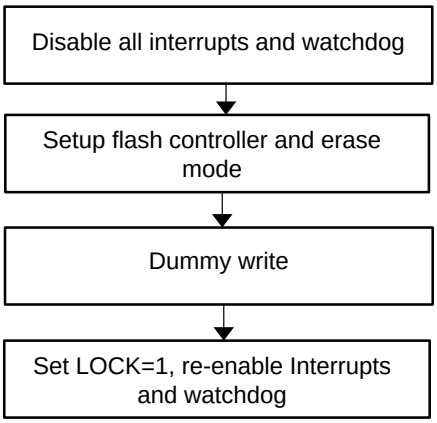
Initiating an Erase from Within Flash Memory

Any erase cycle can be initiated from within flash memory or from RAM. When a flash segment erase operation is initiated from within flash memory, all timing is controlled by the flash controller, and the CPU is held while the erase cycle completes. After the erase cycle completes, the CPU resumes code execution with the instruction following the dummy write.

When initiating an erase cycle from within flash memory, it is possible to erase the code needed for execution after the erase. If this occurs, CPU execution will be unpredictable after the erase cycle.

The flow to initiate an erase from flash is shown in Figure 5–5.

Figure 5–5. Erase Cycle from Within Flash Memory



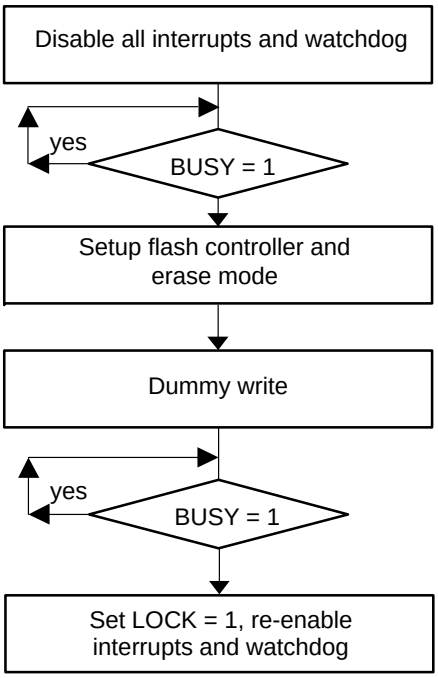

```
; Segment Erase from flash. 514 kHz < SMCLK < 952 kHz
; Assumes ACCVIE = NMIIE = OFIE = 0.
   MOV   #WDTPW+WDTHOLD,&WDTCTL   ; Disable WDT
   DINT                           ; Disable interrupts
   MOV   #FWKEY+FSSEL1+FN0,&FCTL2 ; SMCLK/2
   MOV   #FWKEY,&FCTL3            ; Clear LOCK
   MOV   #FWKEY+ERASE,&FCTL1      ; Enable segment erase
   CLR   &0FC10h                  ; Dummy write, erase S1
   MOV   #FWKEY+LOCK,&FCTL3       ; Done, set LOCK
   ...                            ; Re-enable WDT?
   EINT                           ; Enable interrupts
```

Initiating an Erase from RAM

Any erase cycle may be initiated from RAM. In this case, the CPU is not held and can continue to execute code from RAM. The BUSY bit must be polled to determine the end of the erase cycle before the CPU can access any flash address again. If a flash access occurs while BUSY=1, it is an access violation, ACCVIFG will be set, and the erase results will be unpredictable.

The flow to initiate an erase from flash from RAM is shown in Figure 5–6.

Figure 5–6. Erase Cycle from Within RAM




```
; Segment Erase from RAM. 514 kHz < SMCLK < 952 kHz
; Assumes ACCVIE = NMIIE = OFIE = 0.
   MOV   #WDTPW+WDTHOLD,&WDTCTL   ; Disable WDT
   DINT                           ; Disable interrupts
L1 BIT   #BUSY,&FCTL3             ; Test BUSY
   JNZ   L1                       ; Loop while busy
   MOV   #FWKEY+FSSEL1+FN0,&FCTL2 ; SMCLK/2
   MOV   #FWKEY,&FCTL3            ; Clear LOCK
   MOV   #FWKEY+ERASE,&FCTL1      ; Enable erase
   CLR   &0FC10h                  ; Dummy write, erase S1
L2 BIT   #BUSY,&FCTL3             ; Test BUSY
   JNZ   L2                       ; Loop while busy
   MOV   #FWKEY+LOCK,&FCTL3       ; Done, set LOCK
   ...                            ; Re-enable WDT?
   EINT                           ; Enable interrupts
```

5.3.3 Writing Flash Memory

The write modes, selected by the WRT and BLKWRT bits, are listed in Table 5–1.

Table 5–2. Write Modes

BLKWRT	WRT	Write Mode
0	1	Byte/word write
1	1	Block write

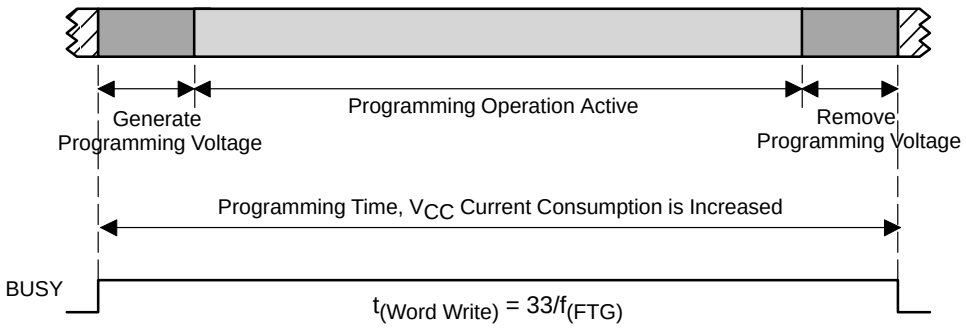
Both write modes use a sequence of individual write instructions, but using the block write mode is approximately twice as fast as byte/word mode, because the voltage generator remains on for the complete block write. Any instruction that modifies a destination can be used to modify a flash location in either byte/word write mode or block write mode.

The BUSY bit is set while a write operation is active and cleared when the operation completes. If the write operation is initiated from RAM, the CPU must not access flash while BUSY=1. Otherwise, an access violation occurs, ACCVIFG is set, and the flash write is unpredictable.

Byte/Word Write

A byte/word write operation can be initiated from within flash memory or from RAM. When initiating from within flash memory, all timing is controlled by the flash controller, and the CPU is held while the write completes. After the write completes, the CPU resumes code execution with the instruction following the write. The byte/word write timing is shown in Figure 5–7.

Figure 5–7. Byte/Word Write Timing

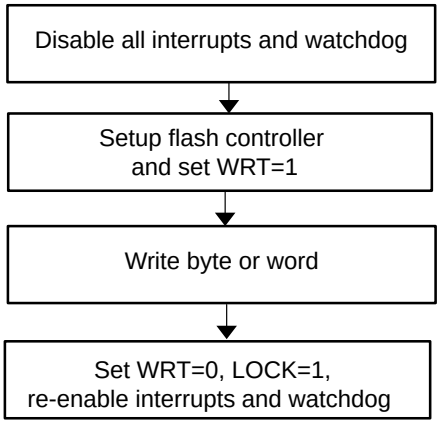


When a byte/word write is executed from RAM, the CPU continues to execute code from RAM. The BUSY bit must be zero before the CPU accesses flash again, otherwise an access violation occurs, ACCVIFG is set, and the write result is unpredictable.

Initiating a Byte/Word Write from Within Flash Memory

The flow to initiate a byte/word write from flash is shown in Figure 5–8.

Figure 5–8. Initiating a Byte/Word Write from Flash

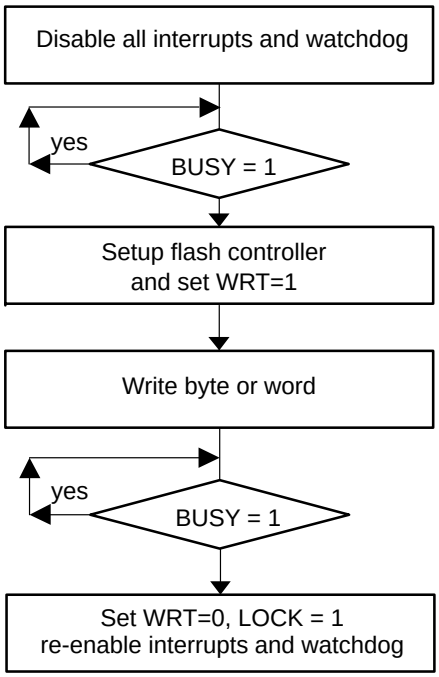



```
; Byte/word write from flash. 514 kHz < SMCLK < 952 kHz
; Assumes 0FF1Eh is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
   MOV   #WDTPW+WDTHOLD,&WDTCTL   ; Disable WDT
   DINT                           ; Disable interrupts
   MOV   #FWKEY+FSSEL1+FN0,&FCTL2 ; SMCLK/2
   MOV   #FWKEY,&FCTL3            ; Clear LOCK
   MOV   #FWKEY+WRT,&FCTL1        ; Enable write
   MOV   #0123h,&0FF1Eh           ; 0123h   -> 0FF1Eh
   MOV   #FWKEY,&FCTL1            ; Done. Clear WRT
   MOV   #FWKEY+LOCK,&FCTL3       ; Set LOCK
   ...                            ; Re-enable WDT?
   EINT                           ; Enable interrupts
```

Initiating a Byte/Word Write from RAM

The flow to initiate a byte/word write from RAM is shown in Figure 5–9.

Figure 5–9. Initiating a Byte/Word Write from RAM

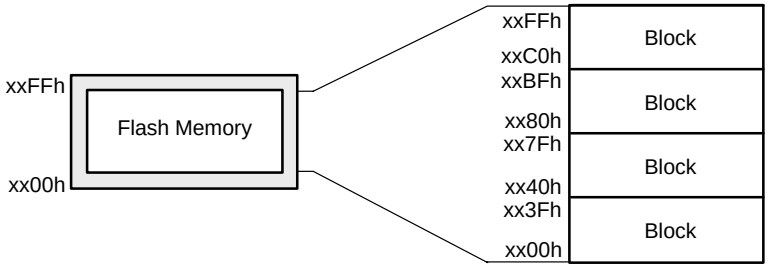



```
; Byte/word write from RAM. 514 kHz < SMCLK < 952 kHz
; Assumes 0FF1Eh is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
   MOV   #WDTPW+WDTHOLD,&WDTCTL   ; Disable WDT
   DINT                           ; Disable interrupts
L1 BIT   #BUSY,&FCTL3             ; Test BUSY
   JNZ   L1                       ; Loop while busy
   MOV   #FWKEY+FSSEL1+FN0,&FCTL2 ; SMCLK/2
   MOV   #FWKEY,&FCTL3            ; Clear LOCK
   MOV   #FWKEY+WRT,&FCTL1        ; Enable write
   MOV   #0123h,&0FF1Eh           ; 0123h -> 0FF1Eh
L2 BIT   #BUSY,&FCTL3             ; Test BUSY
   JNZ   L2                       ; Loop while busy
   MOV   #FWKEY,&FCTL1            ; Clear WRT
   MOV   #FWKEY+LOCK,&FCTL3       ; Set LOCK
   ...                            ; Re-enable WDT?
   EINT                           ; Enable interrupts
```

Block Write

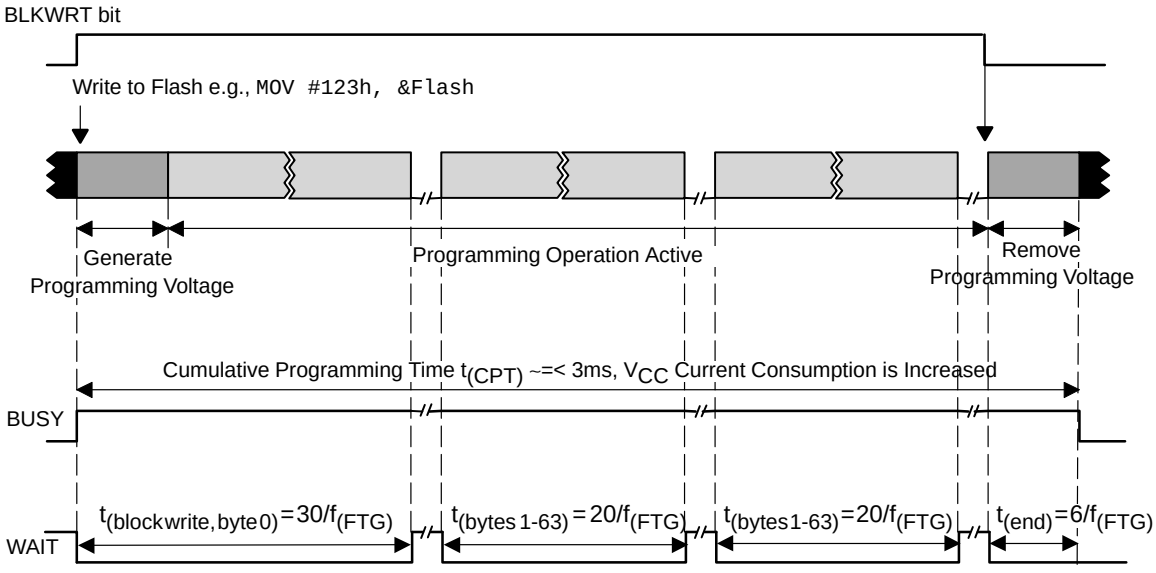
The block write can be used to accelerate the flash write process when many sequential bytes or words need to be programmed. A block is 64 bytes, starting at 0xx00h, 0xx40h, 0xx80h, or 0xxC0h, and ending at 0xx3Fh, 0xx7Fh, 0xxBFh, or 0xxFFh as shown in Figure 5–10. The flash programming voltage remains on for the duration of writing the 64-byte block.

Figure 5–10. Flash Memory Blocks



A block write cannot be initiated from within flash memory. The block write must be initiated from RAM or ROM only. The BUSY bit remains set throughout the duration of the block write. The WAIT bit must be checked between writing each byte or word in the block. When WAIT is set the next byte or word of the block can be written. When writing successive blocks, the BLKWRT bit must be cleared after the current block is complete. BLKWRT can be set initiating the next block write after the required flash recovery time given by t(end). BUSY is cleared following each block write completion indicating the next block can be written. Figure 5–11 shows the block write timing.

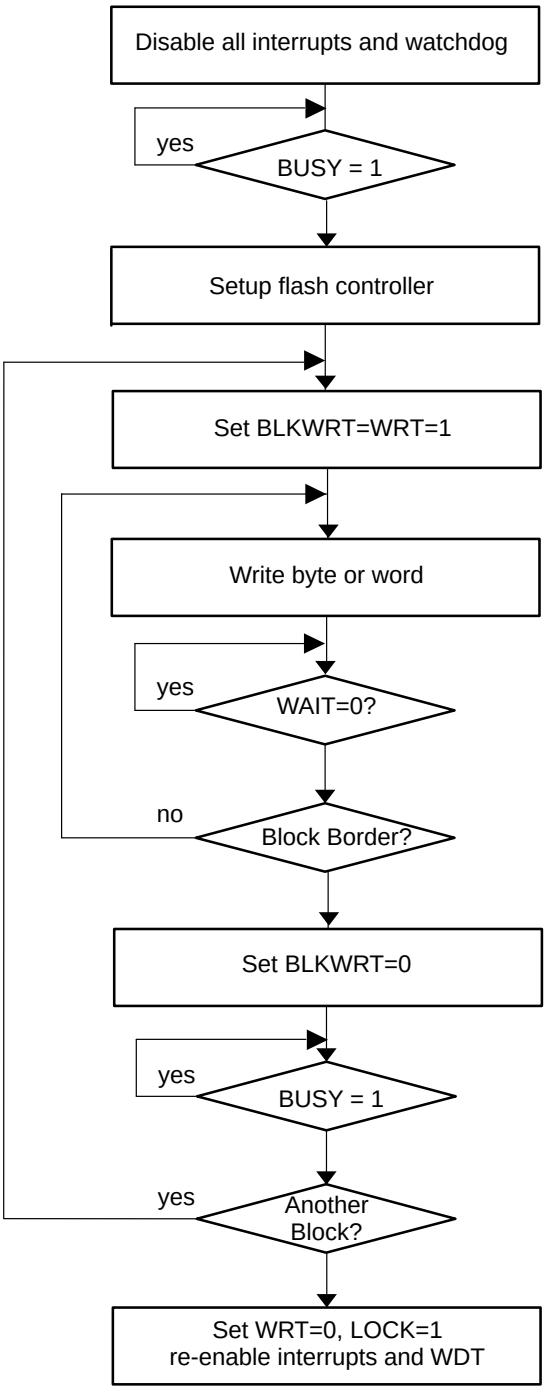
Figure 5–11. Block-Write Cycle Timing



Block Write Flow and Example

A block write flow is shown in Figure 5–8 and the following example.

Figure 5–12. Block Write Flow



```
; Write one block starting at 0F000h.
; Must be executed from RAM, Assumes Flash is already erased.
; 514 kHz < SMCLK < 952 kHz
; Assumes ACCVIE = NMIIE = OFIE = 0.
   MOV   #32,R5                    ; Use as write counter
   MOV   #0F000h,R6                ; Write pointer
   MOV   #WDTPW+WDTHOLD,&WDTCTL    ; Disable WDT
   DINT                            ; Disable interrupts
L1 BIT   #BUSY,&FCTL3              ; Test BUSY
   JNZ   L1                        ; Loop while busy
   MOV   #FWKEY+FSSEL1+FN0,&FCTL2  ; SMCLK/2
   MOV   #FWKEY,&FCTL3             ; Clear LOCK
   MOV   #FWKEY+BLKWRT+WRT,&FCTL1  ; Enable block write
L2 MOV   Write_Value,0(R6)         ; Write location
L3 BIT   #WAIT,&FCTL3              ; Test WAIT
   JZ    L3                        ; Loop while WAIT=0
   INCD  R6                        ; Point to next word
   DEC   R5                        ; Decrement write counter
   JNZ   L2                        ; End of block?
   MOV   #FWKEY,&FCTL1             ; Clear WRT,BLKWRT
L4 BIT   #BUSY,&FCTL3              ; Test BUSY
   JNZ   L4                        ; Loop while busy
   MOV   #FWKEY+LOCK,&FCTL3        ; Set LOCK
   ...                             ; Re-enable WDT if needed
   EINT                            ; Enable interrupts
```

5.3.4 Flash Memory Access During Write or Erase

When any write or any erase operation is initiated from RAM and while BUSY=1, the CPU may not read or write to or from any flash location. Otherwise, an access violation occurs, ACCVIFG is set, and the result is unpredictable. Also if a write to flash is attempted with WRT=0, the ACCVIFG interrupt flag is set, and the flash memory is unaffected.

When a byte/word write or any erase operation is initiated from within flash memory, the flash controller returns op-code 03FFFh to the CPU at the next instruction fetch. Op-code 03FFFh is the `JMP PC` instruction. This causes the CPU to loop until the flash operation is finished. When the operation is finished and BUSY=0, the flash controller allows the CPU to fetch the proper op-code and program execution resumes.

The flash access conditions while BUSY=1 are listed in Table 5–3.

Table 5–3. Flash Access While BUSY = 1

Flash Operation	Flash Access	WAIT	Result
Any erase, or Byte/word write	Read	0	ACCVIFG = 1. 03FFFh is the value read
	Write	0	ACCVIFG = 1. Write is ignored
	Instruction fetch	0	ACCVIFG = 0. CPU fetches 03FFFh. This is the `JMP PC` instruction.
Block write	Any	0	ACCVIFG = 1, LOCK = 1
	Read	1	ACCVIFG = 0, 03FFFh is the value read
	Write	1	ACCVIFG = 0, Write is ignored
	Instruction fetch	1	ACCVIFG = 1, LOCK = 1

All interrupt sources should be disabled before initiating any flash operation. If an enabled interrupt were to occur during a flash operation, the CPU would fetch 03FFFh as the address of the interrupt service routine. The CPU would then execute the `JMP PC` instruction while BUSY=1. When the flash operation finished, the CPU would begin executing code at address 03FFFh, not the correct address for interrupt service routine.

5.3.5 Stopping a Write or Erase Cycle

Any write or erase operation can be stopped before its normal completion by setting the emergency exit bit EMEX. Setting the EMEX bit stops the active operation immediately and stops the flash controller. All flash operations cease, the flash returns to read mode, and all bits in the FCTL1 register are reset. The result of the intended operation is unpredictable.

5.3.6 Configuring and Accessing the Flash Memory Controller

The FCTLx registers are 16-bit, password-protected, read/write registers. Any read or write access must use word instructions and write accesses must include the write password 0A5h in the upper byte. Any write to any FCTLx register with any value other than 0A5h in the upper byte is a security key violation, sets the KEYV flag and triggers a PUC system reset. Any read of any FCTLx registers reads 096h in the upper byte.

Any write to FCTL1 during an erase or byte/word write operation is an access violation and sets ACCVIFG. Writing to FCTL1 is allowed in block write mode when WAIT=1, but writing to FCTL1 in block write mode when WAIT=0 is an access violation and sets ACCVIFG.

Any write to FCTL2 when the BUSY=1 is an access violation.

Any FCTLx register may be read when BUSY=1. A read will not cause an access violation.

5.3.7 Flash Memory Controller Interrupts

The flash controller has two interrupt sources, KEYV, and ACCVIFG. ACCVIFG is set when an access violation occurs. When the ACCVIE bit is re-enabled after a flash write or erase, a set ACCVIFG flag will generate an interrupt request. ACCVIFG sources the NMI interrupt vector, so it is not necessary for GIE to be set for ACCVIFG to request an interrupt. ACCVIFG may also be checked by software to determine if an access violation occurred. ACCVIFG must be reset by software.

The key violation flag KEYV is set when any of the flash control registers are written with an incorrect password. When this occurs, a PUC is generated immediately resetting the device.

5.3.8 Programming Flash Memory Devices

There are three options for programming an MSP430 flash device. All options support in-system programming:

- ❏ Program via JTAG
- ❏ Program via the Bootstrap Loader
- ❏ Program via a custom solution

Programming Flash Memory via JTAG

MSP430 devices can be programmed via the JTAG port. The JTAG interface requires four signals (5 signals on 20- and 28-pin devices), ground and optionally V_{CC} and $\overline{RST}$/NMI.

The JTAG port is protected with a fuse. Blowing the fuse completely disables the JTAG port and is not reversible. Further access to the device via JTAG is not possible For more details see the Application report *Programming a Flash-Based MSP430 Using the JTAG Interface* at www.ti.com/sc/msp430

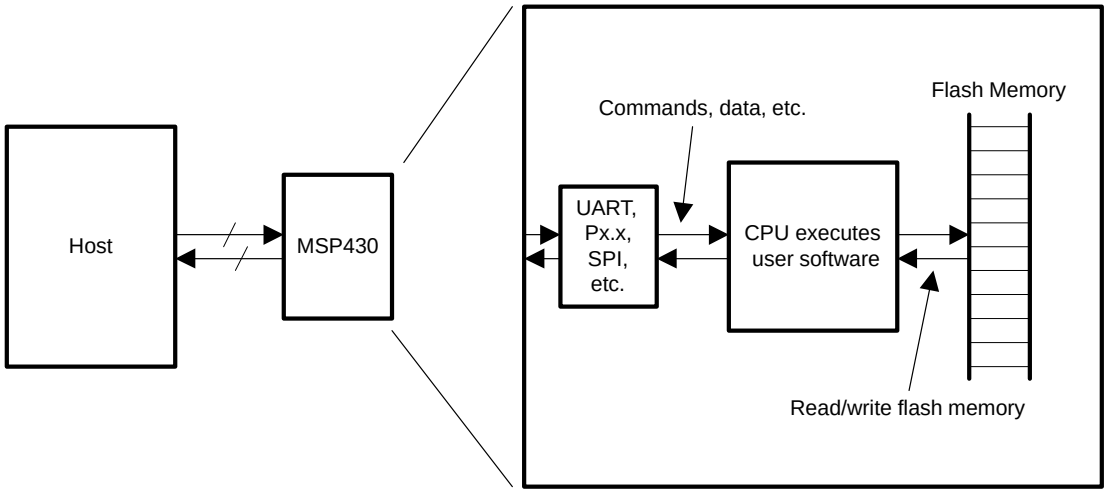
Programming Flash Memory via the Bootstrap loader (BSL)

Every MSP430 flash device contains a bootstrap loader. The BSL enables users to read or program the flash memory or RAM using a UART serial interface. Access to the MSP430 flash memory via the BSL is protected by a 256-bit, user-defined password. For more details see the Application report *Features of the MSP430 Bootstrap Loader* at www.ti.com/sc/msp430.

Programming Flash Memory via a Custom Solution

The ability of the MSP430 CPU to write to its own flash memory allows for in-system and external custom programming solutions as shown in Figure 5–13. The user can choose to provide data to the MSP430 through any means available (UART, SPI, etc.). User-developed software can receive the data and program the flash memory. Since this type of solution is developed by the user, it can be completely customized to fit the application needs for programming, erasing, or updating the flash memory.

Figure 5–13. User-Developed Programming Solution



5.4 Flash Memory Registers

The flash memory registers are listed in Table 5–4.

Table 5–4. Flash Memory Registers

Register	Short Form	Register Type	Address	Initial State
Flash memory control register 1	FCTL1	Read/write	0128h	09600h with PUC
Flash memory control register 2	FCTL2	Read/write	012Ah	09642h with PUC
Flash memory control register 3	FCTL3	Read/write	012Ch	09618h with PUC
Interrupt Enable 1	IE1	Read/write	000h	Reset with PUC

FCTL1, Flash Memory Control Register

15	14	13	12	11	10	9	8
FRKEY, Read as 096h FWKEY, Must be written as 0A5h							

7	6	5	4	3	2	1	0
BLKWRT	**WRT**	**Reserved**	**Reserved**	**Reserved**	**MERAS**	**ERASE**	**Reserved**
rw–0	rw–0	r0	r0	r0	rw–0	rw–0	r0

FRKEY/ FWKEY Bits 15-8 FCTLx password. Always read as 096h. Must be written as 0A5h or a PUC will be generated.

BLKWRT Bit 7 Block write mode. WRT must also be set for block write mode. BLKWRT is automatically reset when EMEX is set.
0 Block-write mode is off
1 Block-write mode is on

WRT Bit 6 Write. This bit is used to select any write mode. WRT is automatically reset when EMEX is set.
0 Write mode is off
1 Write mode is on

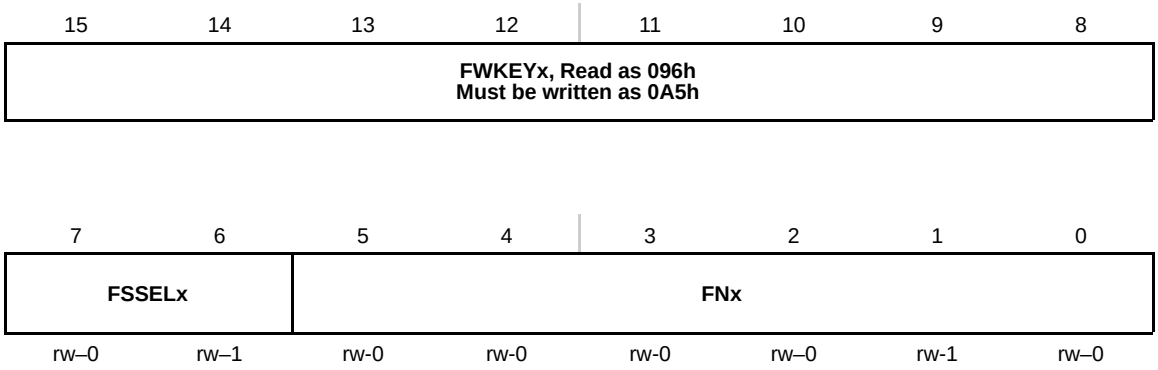
Reserved Bits 5-3 Reserved. Always read as 0.

MERAS ERASE Bit 2 Bit 1 Mass erase and erase. These bits are used together to select the erase mode. MERAS and ERASE are automatically reset when EMEX is set.

MERAS	ERASE	Erase Cycle
0	0	No erase
0	1	Erase individual segment only
1	0	Erase all main memory segments
1	1	Erase all main and information memory segments

Reserved Bit 0 Reserved. Always read as 0.

FCTL2, Flash Memory Control Register



FWKEYx	Bits 15-8	FCTLx password. Always read as 096h. Must be written as 0A5h or a PUC will be generated.
FSSELx	Bits 7–6	Flash controller clock source select 00 ACLK 01 MCLK 10 SMCLK 11 SMCLK
FNx	Bits 5-0	Flash controller clock divider. These six bits select the divider for the flash controller clock. The divisor value is FNx + 1. For example, when FNx=00h, the divisor is 1. When FNx=02Fh the divisor is 64.

FCTL3, Flash Memory Control Register FCTL3

15	14	13	12	11	10	9	8
FWKEYx, Read as 096h Must be written as 0A5h							

7	6	5	4	3	2	1	0
Reserved	**Reserved**	**EMEX**	**LOCK**	**WAIT**	**ACCVIFG**	**KEYV**	**BUSY**
r0	r0	rw-0	rw-1	r-1	rw–0	rw-(0)	r(w)–0

FWKEYx Bits 15-8 FCTLx password. Always read as 096h. Must be written as 0A5h or a PUC will be generated.

Reserved Bits 7-6 Reserved. Always read as 0.

EMEX Bit 5 Emergency exit
0 No emergency exit
1 Emergency exit

LOCK Bit 4 Lock. This bit unlocks the flash memory for writing or erasing. The LOCK bit can be set anytime during a byte/word write or erase operation and the operation will complete normally. In the block write mode if the LOCK bit is set while BLKWRT=WAIT=1, then BLKWRT and WAIT are reset and the mode ends normally.
0 Unlocked
1 Locked

WAIT Bit 3 Wait. Indicates the flash memory is being written to.
0 The flash memory is not ready for the next byte/word write
1 The flash memory is ready for the next byte/word write

ACCVIFG Bit 2 Access violation interrupt flag
0 No interrupt pending
1 Interrupt pending

KEYV Bit 1 Flash security key violation. This bit indicates an incorrect FCTLx password was written to any flash control register and generates a PUC when set. KEYV must be reset with software.
0 FCTLx password was written correctly
1 FCTLx password was written incorrectly

BUSY Bit 0 Busy. This bit indicates the status of the flash timing generator.
0 Not Busy
1 Busy

IE1, Interrupt Enable Register 1

7	6	5	4	3	2	1	0
		ACCVIE					
		rw–0					

	Bits 7-6, 4-0	These bits may be used by other modules. See device-specific datasheet.
ACCVIE	Bit 5	Flash memory access violation interrupt enable. This bit enables the ACCVIFG interrupt. Because other bits in IE1 may be used for other modules, it is recommended to set or clear this bit using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions. 0 Interrupt not enabled 1 Interrupt enabled

Chapter 6

Supply Voltage Supervisor

This chapter describes the operation of the SVS. The SVS is implemented in MSP430x15x and MSP430x16x devices.

Topic **Page**

6.1 SVS Introduction

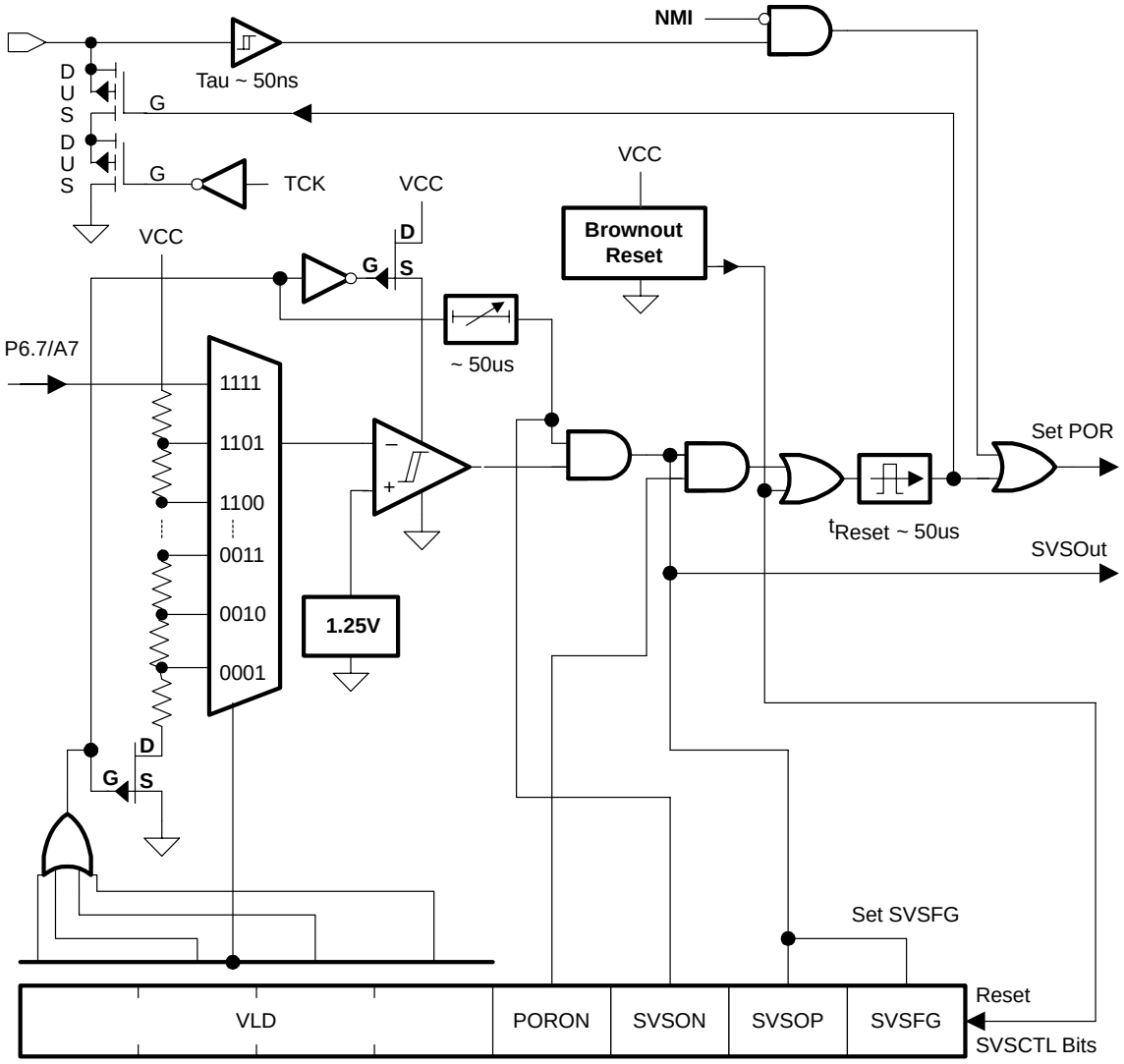
The supply voltage supervisor (SVS) is used to monitor the AV_{CC} supply voltage or an external voltage. The SVS can be configured to set a flag or generate a POR reset when the supply voltage or external voltage drops below a user-selected threshold.

The SVS features include:

- AV_{CC} monitoring
- Selectable generation of POR
- Output of SVS comparator accessible by software
- Low-voltage condition latched and accessible by software
- 14 selectable threshold levels
- External channel to monitor external voltage

The SVS block diagram is shown in Figure 6–1.

Figure 6–1. SVS Block Diagram



6.2 SVS Operation

The SVS detects if the AV_{CC} voltage drops below a selectable level. It can be configured to provide a POR or set a flag, when a low-voltage condition occurs. The SVS is disabled after a POR to conserve current consumption.

6.2.1 Configuring the SVS

The VLDx bits are used to enable/disable the SVS and select one of 14 threshold levels ($V_{(SVS\_IT-)}$) for comparison with AV_{CC}. The SVS is off when VLDx = 0 and on when VLDx > 0. The SVSON bit does not turn on the SVS. Instead, it reflects the on/off state of the SVS and can be used to determine when the SVS is on.

When VLDx = 1111, the external SVSin channel is selected. The voltage on SVSin is compared to an internal level of approximately 1.2 V.

6.2.2 SVS Comparator Operation

A low-voltage condition exists when AV_{CC} drops below the selected threshold or when the external voltage drops below its 1.2-V threshold. Any low-voltage condition sets the SVSFG bit.

The PORON bit enables or disables the device-reset function of the SVS. If PORON = 1, a POR is generated when SVSFG is set. If PORON = 0, a low-voltage condition sets SVSFG, but does not generate a POR.

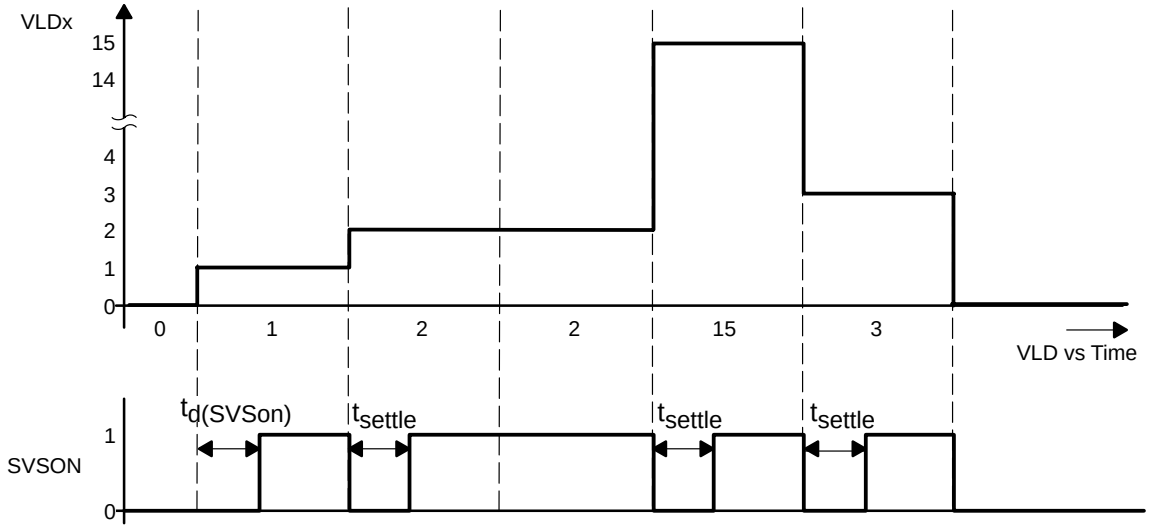
The SVSFG bit is latched. This allows user software to determine if a low-voltage condition occurred previously. The SVSFG bit must be reset by user software. If the low-voltage condition is still present when SVSFG is reset, it will be immediately set again by the SVS.

6.2.3 Changing the VLDx Bits

When the VLDx bits are changed, two settling delays are implemented to allows the SVS circuitry to settle. During each delay, the SVS will not set SVSFG. The delays, $t_{d(SVSon)}$ and $t_{settle,}$ are shown in Figure 6–2. The $t_{d(SVSon)}$ delay takes affect when VLDx is changed from zero to any non-zero value and is a approximately 50 µs. The t_{settle} delay takes affect when the VLDx bits change from any non-zero value to any other non-zero value and is a maximum of ~12 µs. See the device-specific datasheet for the delay parameters.

During the delays, the SVS will not flag a low-voltage condition or reset the device, and the SVSON bit is cleared. Software can test the SVSON bit to determine when the delay has elapsed and the SVS is monitoring the voltage properly.

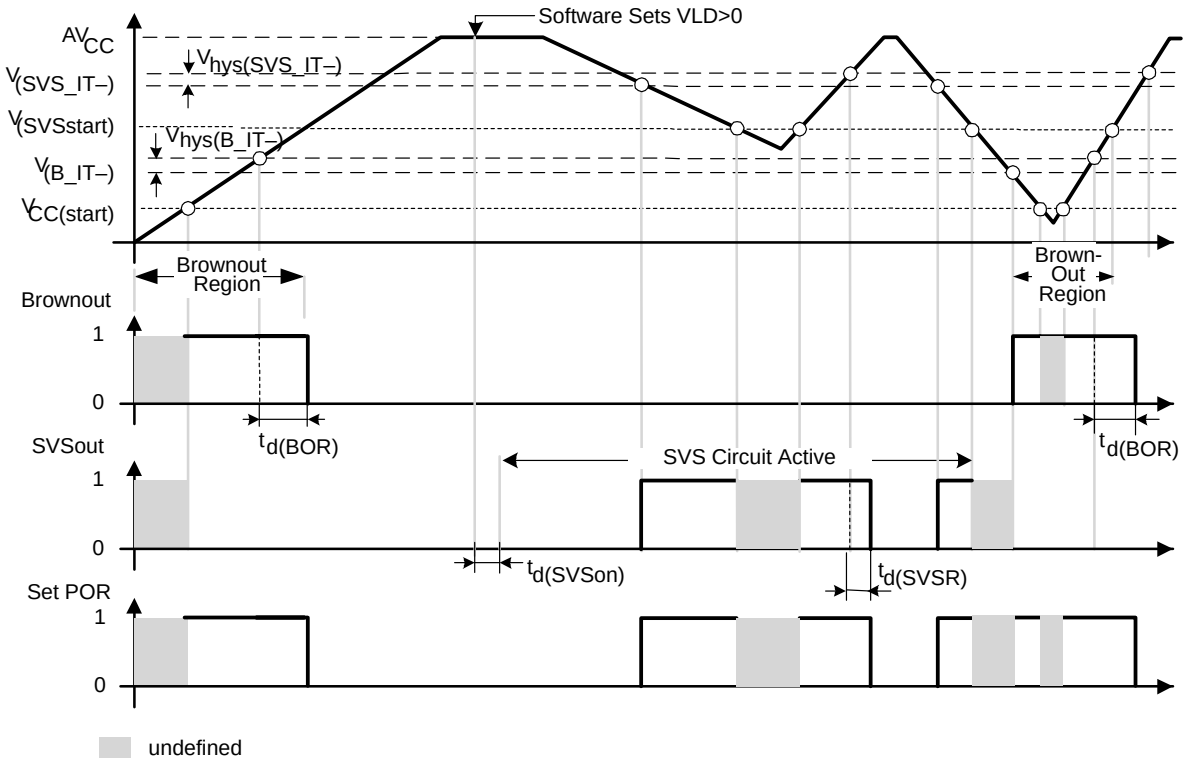
Figure 6–2. SVSON state When Changing VLDx



6.2.4 SVS Operating Range

Each SVS level has hysteresis to reduce sensitivity to small supply voltage changes when AV_{CC} is close to the threshold. The SVS operation and SVS/Brownout interoperation are shown in Figure 6–3.

Figure 6–3. Operating Levels for SVS and Brownout/Reset Circuit



6.3 SVS Registers

The SVS registers are listed in Table 6–1.

Table 6–1. SVS Registers

Register	Short Form	Register Type	Address	Initial State
SVS Control Register	SVSCTL	Read/write	050h	Reset with POR

SVSCTL, SVS Control Register

7	6	5	4	3	2	1	0
VLDx				**PORON**	**SVSON**	**SVSOP**	**SVSFG**
rw–0	rw–0	rw–0	rw–0	rw–0	r	r	rw–0

VLDx Bits 7-4 Voltage level detect. These bits turn on the SVS and select the nominal SVS threshold voltage level. See the device–specific datasheet for parameters.
0000 SVS is off
0001 1.9 V
0010 2.1 V
0011 2.2 V
0100 2.3 V
0101 2.4 V
0110 2.5 V
0111 2.65 V
1000 2.8 V
1001 2.9 V
1010 3.05
1011 3.2 V
1100 3.35 V
1101 3.5 V
1110 3.7 V
1111 Compares external input voltage SVSin to 1.2 V.

PORON Bit 3 POR on. This bit enables the SVSFG flag to cause a POR device reset.
0 SVSFG does not cause a POR
1 SVSFG causes a POR

SVSON Bit 2 SVS on. This bit reflects the status of SVS operation. This bit DOES NOT turn on the SVS. The SVS is turned on by setting VLDx > 0.
0 SVS is Off
1 SVS is On

SVSOP Bit 1 SVS output. This bit reflects the output value of the SVS comparator.
0 SVS comparator output is high
1 SVS comparator output is low

SVSFG Bit 0 SVS flag. This bit indicates a low voltage condition. SVSFG remains set after a low voltage condition until reset by software.
0 No low voltage condition occurred
1 A low condition is preset or has occurred

Chapter 7

Hardware Multiplier

This chapter describes the hardware multiplier. The hardware multiplier is implemented in MSP430x14x and MSP430x16x devices.

Topic **Page**

7.1 Hardware Multiplier Introduction

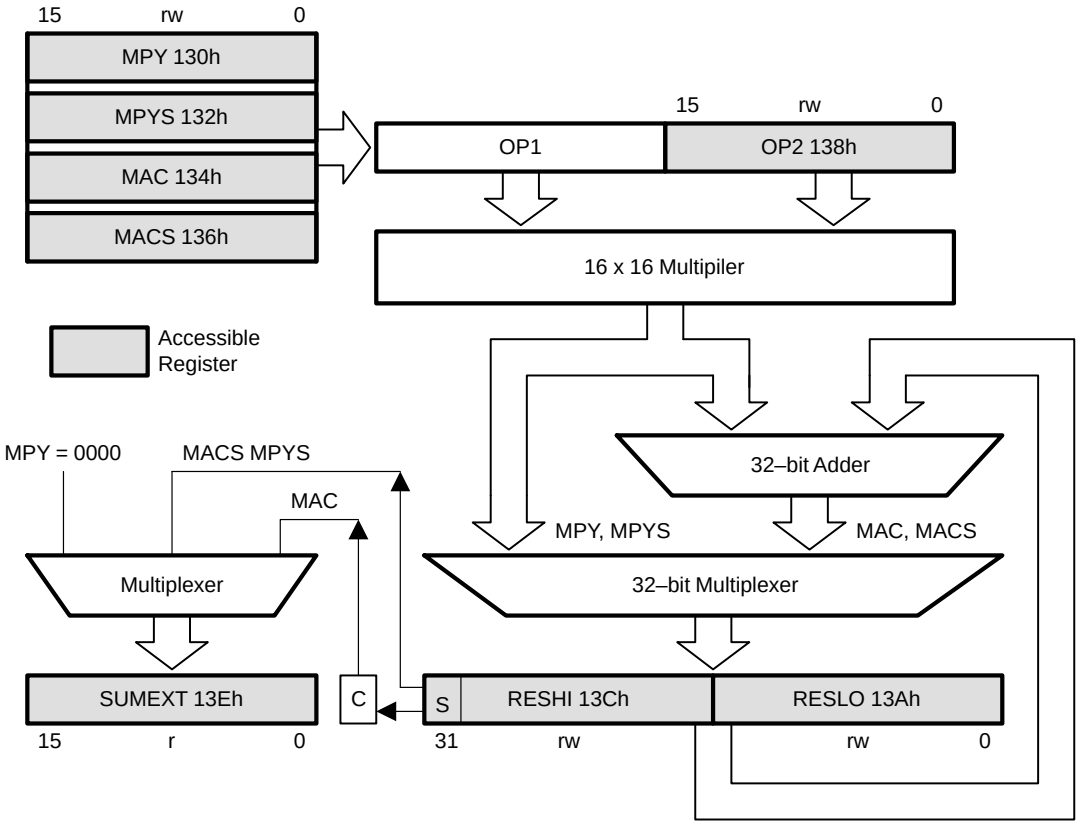
The hardware multiplier is a peripheral and is not part of the MSP430 CPU. This means, its activities do not interfere with the CPU activities. The multiplier registers are peripheral registers that are loaded and read with CPU instructions.

The hardware multiplier supports:

- ❏ Unsigned multiply
- ❏ Signed multiply
- ❏ Unsigned multiply accumulate
- ❏ Signed multiply accumulate
- ❏ 16×16 bits, 16×8 bits, 8×16 bits, 8×8 bits

The hardware multiplier block diagram is shown in Figure 7–1.

Figure 7–1. Hardware Multiplier Block Diagram



7.2 Hardware Multiplier Operation

The hardware multiplier supports unsigned multiply, signed multiply, unsigned multiply accumulate, and signed multiply accumulate operations. The type of operation is selected by the address the first operand is written to.

The hardware multiplier has two 16-bit operand registers, OP1 and OP2. and three result registers, RESLO, RESHI, and SUMEXT. RESLO stores the low word of the result, RESHI stores the high word of the result, and SUMEXT stores information about the result. The result can be read with the next instruction after writing to OP2, except when using an indirect addressing mode.

7.2.1 Operand Registers

The operand one register OP1 has four addresses, shown in Table 7–1, used to select the multiply mode. Writing the first operand to the desired address selects the type of multiply operation but does not start any operation. Writing the second operand to the operand two register OP2 initiates the multiply operation. Writing OP2 starts the selected operation with the values stored in OP1 and OP2. The result is written into the three result registers RESLO, RESHI, and SUMEXT.

Repeated multiply operations may be performed without reloading OP1 if the OP1 value is used for successive operations. It is not necessary to re-write the OP1 value to perform the operations.

Table 7–1. OP1 addresses

OP1 Address	Register Name	Operation
0130h	MPY	Unsigned multiply
0132h	MPYS	Signed multiply
0134h	MAC	Unsigned multiply accumulate
0136h	MACS	Signed multiply accumulate.

7.2.2 Result Registers

The result low register RESLO holds the lower 16-bits of the calculation result. The result high register RESHI contents depend on the multiply operation and are listed in Table 7–2.

Table 7–2. RESHI Contents

Mode	RESHI Contents
MPY	Upper 16-bits of the result
MPYS	The MSB is the sign of the result. The remaining bits are the upper 15-bits of the result. Two's complement notation is used for the result.
MAC	Upper 16-bits of the result
MACS	Upper 16-bits of the result. Two's complement notation is used for the result.

The sum extension registers SUMEXT contents depend on the multiply operation and are listed in Table 7–3.

Table 7–3. SUMEXT Contents

Mode	SUMEXT
MPY	SUMEXT is always 0000h
MPYS	SUMEXT contains the extended sign of the result 00000h Result was positive 0FFFFh Result was negative
MAC	SUMEXT contains the carry of the result 0000h No carry for result 0001h Result has a carry
MACS	SUMEXT contains the extended sign of the result 00000h Result was positive 0FFFFh Result was negative

MACS Underflow and Overflow

The multiplier does not automatically detect underflow or overflow in the MACS mode. The accumulator range for positive numbers is 0 to 7FFF FFFFh and for negative numbers is 0FFFF FFFFh to 8000 0000h. An overflow occurs when the sum of two negative numbers yields a result that is in the range 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. In both of these cases, the SUMEXT register contains the correct sign of the result, 0FFFFh for overflow and 0000h for underflow. User software must detect and handle these conditions appropriately.

7.2.3 Software Examples

Examples for all multiplier modes follow. All 8x8 modes use the absolute address for the registers because the assembler will not allow .B access to word registers when using the labels from the standard definitions file.

```
; 16x16 Unsigned Multiply
   MOV   #01234h,&MPY ; Load first operand
   MOV   #05678h,&OP2 ; Load second operand
;  ...               ; Process results

; 8x8 Unsigned Multiply. Absolute addressing.
   MOV.B #012h,&0130h ; Load first operand
   MOV.B #034h,&0138h ; Load 2nd operand
;  ...               ; Process results

; 16x16 Signed Multiply
   MOV   #01234h,&MPYS ; Load first operand
   MOV   #05678h,&OP2 ; Load 2nd operand
;  ...               ; Process results

; 8x8 Signed Multiply. Absolute addressing.
   MOV.B    #012h,&0132h ; Load first operand
   SXT   &MPYS        ; Sign extend first operand
   MOV.B    #034h,&0138h ; Load 2nd operand
   SXT   &OP2         ; Sign extend 2nd operand
                      ; (triggers 2nd multiplication)
;  ...               ; Process results

; 16x16 Unsigned Multiply Accumulate
   MOV   #01234h,&MAC ; Load first operand
   MOV   #05678h,&OP2 ; Load 2nd operand
;  ...               ; Process results

; 8x8 Unsigned Multiply Accumulate. Absolute addressing
   MOV.B    #012h,&0134h ; Load first operand
   MOV.B    #034h,&0138h ; Load 2nd operand
;  ...               ; Process results

; 16x16 Signed Multiply Accumulate
   MOV   #01234h,&MACS ; Load first operand
   MOV   #05678h,&OP2 ; Load 2nd operand
;  ...               ; Process results

; 8x8 Signed Multiply Accumulate. Absolute addressing
   MOV.B    #012h,&0136h ; Load first operand
   SXT   &MACS        ; Sign extend first operand
   MOV.B    #034h,R5     ; Temp. location for 2nd operand
   SXT   R5           ; Sign extend 2nd operand
   MOV   R5,&OP2      ; Load 2nd operand
;  ...               ; Process results
```

7.2.4 Indirect Addressing of RESLO

When using indirect or indirect autoincrement addressing mode to access the result registers, At least one instruction is needed between loading the second operand and accessing one of the result registers:

```
; Access multiplier results with indirect addressing
   MOV   #RESLO,R5    ; RESLO address in R5 for indirect
   MOV   &OPER1,&MPY  ; Load 1st operand
   MOV   &OPER2,&OP2  ; Load 2nd operand
   NOP                ; Need one cycle
   MOV   @R5+,&xxx    ; Move RESLO
   MOV   @R5,&xxx     ; Move RESHI
```

7.2.5 Using Interrupts

If an interrupt occurs after writing OP1, but before writing OP2, and the multiplier is used in servicing that interrupt, the original multiplier mode selection is lost and the results are unpredictable. To avoid this, disable interrupts before using the hardware multiplier or do not use the multiplier in interrupt service routines.

```
; Disable interrupts before using the hardware multiplier
   DINT            ; Disable interrupts
   NOP             ; Required for DINT
   MOV   #xxh,&MPY ; Load 1st operand
   MOV   #xxh,&OP2 ; Load 2nd operand
   EINT            ; Interrupts may be enable before
                   ; Process results
```

7.3 Hardware Multiplier Registers

The hardware multiplier registers are listed in Table 7–4.

Table 7–4. Hardware Multiplier Registers

Register	Short Form	Register Type	Address	Initial State
Operand one – multiply	MPY	Read/write	0130h	Unchanged
Operand one - signed multiply	MPYS	Read/write	0132h	Unchanged
Operand one - multiply accumulate	MAC	Read/write	0134h	Unchanged
Operand one - signed multiply accumulate	MACS	Read/write	0136h	Unchanged
Operand two	OP2	Read/write	0138h	Unchanged
Result low word	RESLO	Read/write	013Ah	Undefined
Result high word	RESHI	Read/write	013Ch	Undefined
Sum Extension register	SUMEXT	Read	013Eh	Undefined

Chapter 8

DMA Controller

The DMA controller module transfers data from one address to another without CPU intervention. This chapter describes the operation of the DMA controller. The DMA controller is implemented in MSP430x15x and MSP430x16x devices.

Topic **Page**

8.1 DMA Introduction

The direct memory access (DMA) controller transfers data from one address to another, without CPU intervention, across the entire address range. For example, the DMA controller can move data from the ADC12 conversion memory to RAM.

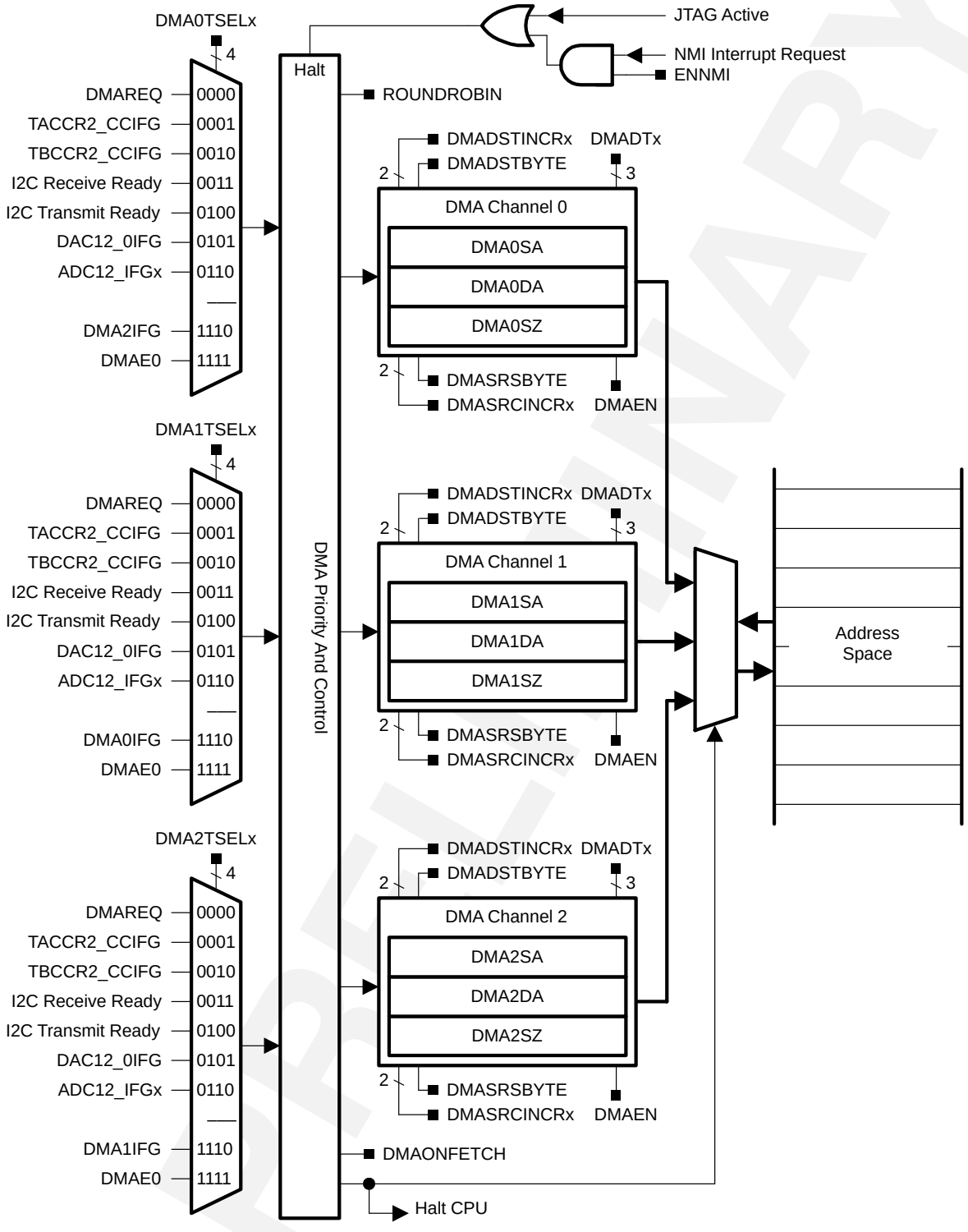
Using the DMA controller can increase the throughput of peripheral modules. It can also reduce system power consumption by allowing the CPU to remain in a low-power mode without having to awaken to move data to or from a peripheral.

The DMA controller features include:

- Three independent transfer channels
- Configurable DMA channel priorities
- Requires only two MCLK clock cycles
- Byte or word and mixed byte/word transfer capability
- Block sizes up to 65535 bytes or words
- Configurable transfer trigger selections
- Selectable edge or level-triggered transfer
- Four transfer-addressing modes
- Single, block, or burst-block transfer modes

The DMA controller block diagram is shown in Figure 8–1.

Figure 8–1. DMA Controller Block Diagram



8.2 DMA Operation

The DMA controller is configured with user software. The setup and operation of the DMA is discussed in the following sections.

8.2.1 DMA Addressing Modes

The DMA controller has four transfer addressing modes shown in Figure 8–2:

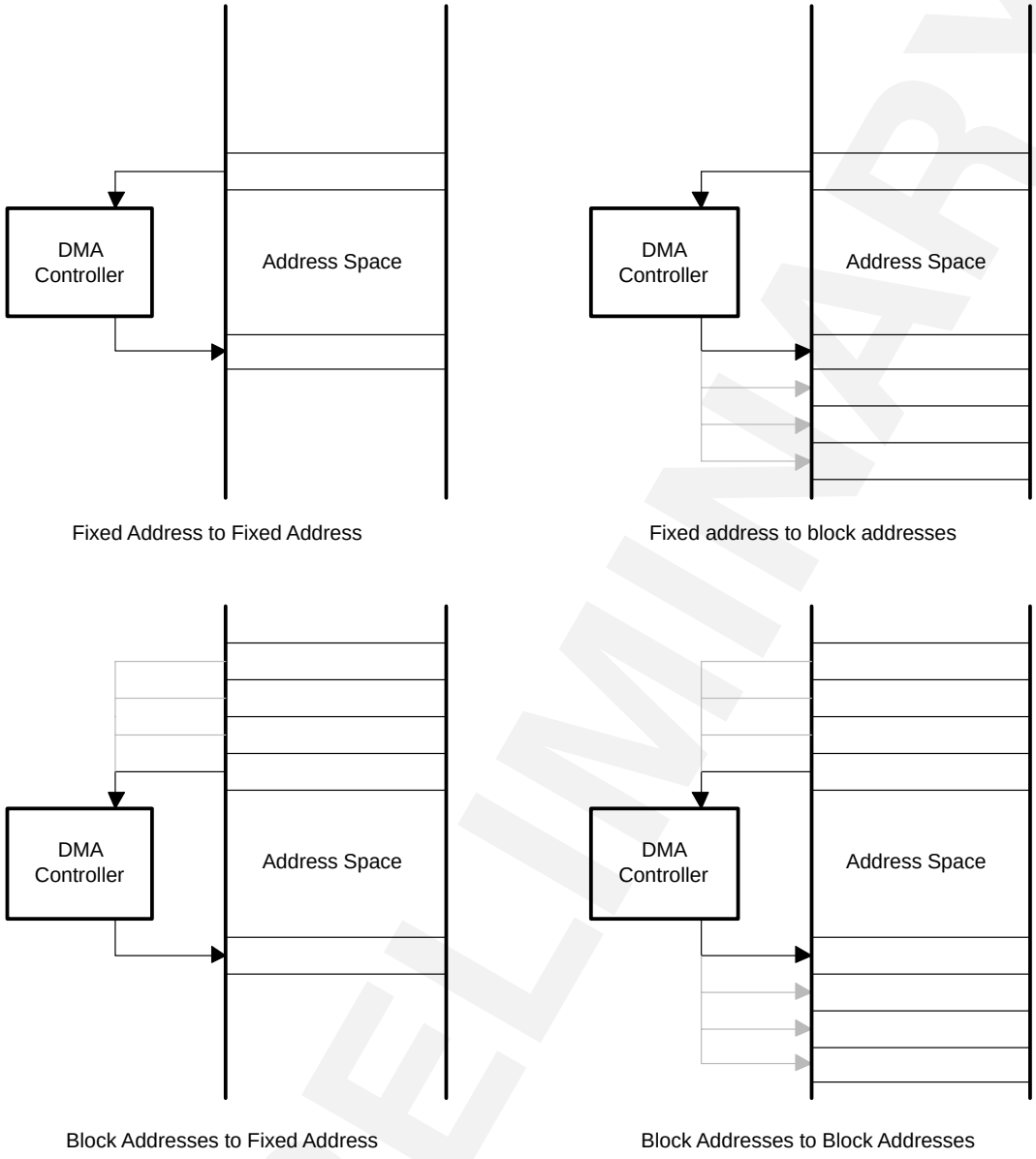
- Fixed address to fixed address
- Fixed address to block of addresses
- Block of addresses to fixed address
- Block of addresses to block of addresses

The transfer-addressing mode for each DMA channel is independently configurable. For example, channel 0 may transfer between two fixed addresses, while channel 1 transfers between two blocks of addresses.

The DMA addressing modes are configured with the DMASRCINCR and DMADSTINCR control bits. The DMASRCINCR bit selects if the DMA source address (DMAxSA) is incremented, decremented, or unchanged after each DMA transfer. The DMADSTINCR bit selects if the DMA destination address (DMAxDA) is incremented, decremented, or unchanged after each DMA transfer.

DMA transfers may be byte-to-byte, word-to-word, byte-to-word, or word-to-byte. When transferring word-to-byte, only the lower byte of the source-word transfers. When transferring byte-to-word, the upper byte of the destination-word is cleared when the transfer occurs.

Figure 8–2. DMA Addressing Modes



8.2.2 DMA Transfer Modes

The DMA controller has six transfer modes:

- ❑ Single transfer
- ❑ Block transfer
- ❑ Burst-block transfer
- ❑ Repeated single transfers
- ❑ Repeated block transfers
- ❑ Repeated burst-block transfers

Each DMA channel can be individually configurable for its transfer mode with the corresponding DMADTx bits. For example, channel 0 may be configured in single transfer mode, while channel 1 is configured for burst-block transfer mode, and channel 2 operates in repeated block mode. The DMA transfer mode is configured independently from the DMA addressing mode. Any DMA addressing mode can be used with any DMA transfer mode.

The DMA state diagram is shown in Figure 8–3.

Single Transfer

When a DMA channel is configured in single transfer mode, each byte/word transfer requires a separate trigger. Setting DMADTx = 0 configures single transfer mode. When DMADTx = 0, the DMAEN bit is cleared after each transfer and must be set again for another transfer to occur. Setting DMADTx = 4 configures repeated single transfer mode. When DMADTx = 4, the DMA controller remains enabled with DMAEN = 1, and a DMA transfer occurs every time a trigger occurs.

The DMAxSZ register contains the number of transfers to be made. If DMAxSZ = 0, not transfers occur. The DMAxSZ register is copied into a temporary register and decrements with every transfer. When the DMAxSZ register decrements to zero it is reloaded from its temporary register and the corresponding DMAIFG flag is set.

Block Transfers

In block mode, a transfer of a complete block of data occurs after just one trigger. Setting DMADTx = 1 configures block transfer mode. When DMADTx = 1, the DMAEN bit is cleared after the completion of the block transfer and must be set again before another block transfer can be triggered. After a DMA block transfer has been triggered, further trigger signals occurring during the block transfer are ignored.

Setting DMADTx = 5 configures repeated block transfer mode. When DMADTx = 5, the DMAEN bit remains set after completion of the block transfer. The next trigger after the completion of a repeated block transfer triggers another block transfer.

The DMAxSZ register is used to define the size of the block and the DMADSTINCR and DMASRCINCR bits select if the DMA destination address (DMAxDA) and the DMA source address (DMAxSA) are incremented or decremented after each transfer of the block.

During a block or repeated block transfer, the DMAxSA, DMAxDA, and DMAxSZ registers are copied into temporary registers. The temporary values of DMAxSA and DMAxDA are incremented or decremented after each transfer in the block. The DMAxSZ register is decremented after each transfer of the block and shows the number of transfers remaining in the block. When the DMAxSZ register decrements to zero it is reloaded from its temporary register and the corresponding DMAIFG flag is set.

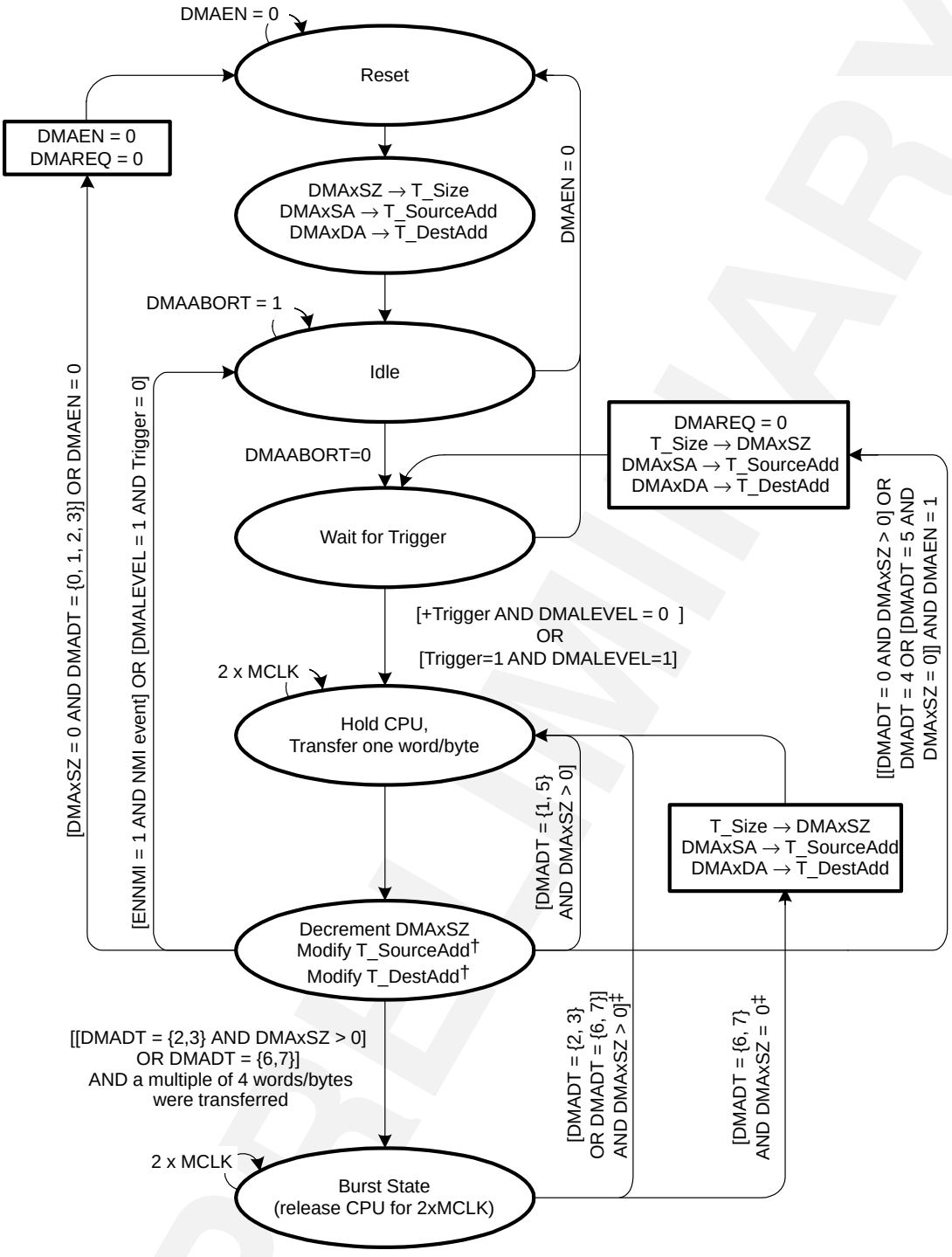
During a block transfer, the CPU is halted until the complete block has been transferred. The block transfer takes 2 x MCLK x DMAxSZ clock cycles to complete. CPU execution resumes with its previous state after the block transfer is complete.

Burst-Block Transfers

Burst-block transfers are block transfers with CPU activity interleaved. In burst-block mode, the CPU executes 2 MCLK cycles after every four byte/word transfers of the block. During a burst-block transfer, the CPU executes at 20% capacity. Setting DMADT = {2,3}, configures the burst-block mode. After the burst-block, CPU execution resumes at 100% capacity, the DMAEN bit is cleared. DMAEN must be set again before another burst-block transfer can be triggered. After a DMA burst-block transfer has been triggered, further trigger signals occurring during the burst-block transfer are ignored.

Setting DMADT = {6,7} configures repeated burst-block mode. When DMADTx = {6, 7}, the DMAEN bit remains set after completion of the burst-block transfer and no further trigger signals are required to initiate another burst-block transfer. Another burst-block transfer begins immediately after completion of a burst-block transfer. In this case, the DMA transfers must be stopped by by clearing the DMAEN bit, or by an NMI interrupt when ENNMI is set. In repeated burst-block mode the CPU executes at 20% capacity continuously until the repeated burst-block transfer is stopped.

Figure 8–3. DMA State Diagram



† The temporary registers are incremented or decremented, depending on the DMASRCINCR and DMADSTINCR bits. The DMADSTBYTE and DMASRCBYTE bits determine if they are incremented/decremented by one byte or word.

‡ Once started, the burst-block mode operates continuously until the DMAEN bit is reset. No additional trigger events are required.

8.2.3 Initiating DMA Transfers

DMA transfers are initiated from software or hardware and are described in Table 8–1. Each DMA channel is independently configured for its trigger source with the DMAxTSELx bits.

Table 8–1. DMA Trigger Sources

DMAx TSELx	DMA Trigger	Trigger Action
0000	DMAREQ	Setting the DMAREQ bit triggers a DMA transfer. DMAREQ is automatically reset when the DMA transfer starts.
0001	TACCR2 CCIFG	A DMA transfer is triggered when the TACCR2 CCIFG flag is set. TACCR2 CCIFG is automatically reset when the DMA transfer starts.
0010	TBCCR2 CCIFG	A DMA transfer is triggered when the TBCCR2 CCIFG flag is set. The TBCCR2 CCIFG is automatically reset when the DMA transfer starts.
0011	I^2C data received	A DMA transfer is triggered when the I^2C module receives new data. The DMA trigger is the condition, not the RXRDYIFG flag. RXRDYIFG is not cleared when the DMA transfer starts, and setting RXRDYIFG with software will not trigger a DMA transfer.
0100	I^2C transmit ready	A DMA transfer is triggered when the I^2C module is ready to transmit new data. The DMA trigger is the condition, not the TXRDYIFG flag. TXRDYIFG is not cleared when the DMA transfer starts, and setting TXRDYIFG with software will not trigger a DMA transfer.
0101	DAC12_0 DAC12IFG	A DMA transfer is triggered when the DAC12_0 DAC12IFG flag is set. The DAC12_0 DAC12IFG flag is automatically cleared when the DMA transfer starts.
0110	ADC12IFGx	A DMA transfer is triggered when ADC12IFGx is set. The ADC12IFGx flag is automatically selected by the ADC12 configuration. When the ADC12 performs a single or repeated conversion on a single channel, the ADC12IFGx flag for conversion is the DMA trigger. When the ADC12 performs a single or repeated sequence of conversions, the ADC12IFGx for the last conversion in the sequence is the DMA trigger. ADC12IFGx flags are not automatically reset when a DMA transfer starts. All ADC12IFGx flags are automatically reset when the corresponding ADC12MEMx register is accessed, either by software or by the DMA controller.
0111 -1101	No Trigger	No DMA transfers triggered
1110	DMAxIFG	DMA0IFG triggers DMA channel 1. DMA1IFG triggers DMA channel 2. DMA2IFG triggers DMA channel 0. None of the DMAxIFG flags are automatically reset when the DMA transfer starts.
1111	DMAE0	External trigger DMAE0

Edge-Sensitive Triggers

When DMALEVEL = 0, edge-sensitive triggers are used and the rising edge of the trigger signal initiates the transfer. In single-transfer mode, each DMA transfer requires its own trigger. When using block or burst-block modes, only one trigger is required to initiate the block or burst-block transfer.

Level-Sensitive Triggers

When DMALEVEL = 1, level-sensitive triggers are used. For proper operation, level-sensitive triggers can used only be used when external trigger DMAE0 is selected as the DMA trigger.

When DMALEVEL = 1, DMA transfers are triggered as long as the trigger signal is high and the DMAEN bit remains set. DMA transfer modes DMADTx = {0, 1, 2, 3} are recommended when DMALEVEL = 1 because the DMAEN bit is automatically reset after the configured DMA transfer.

When DMALEVEL = 1, the trigger signal must remain high for a block or burst-block transfer to complete. If the trigger signal goes low during a block or burst-block transfer, the DMA controller is held in its current state until the trigger goes back high or until the DMA registers are modified by software. If the DMA registers are not modified by software, when trigger signal goes high again, the transfer resumes from where it was when the trigger signal went low.

Halting Executing Instructions for DMA Transfers

The DMAONFETCH bit controls when the CPU is halted for a DMA transfer. When DMAONFETCH = 0, the CPU is halted immediately and the DMA transfer begins when a DMA trigger is received. When DMAONFETCH = 1, the CPU finishes the currently executing instruction before the DMA controller halts the CPU and the DMA transfer begins.

8.2.4 Stopping DMA Transfers

There are two ways to stop DMA transfers in progress:

- A single, block, or burst-block DMA transfer may be stopped with an NMI interrupt, if the ENNMI bit is set in register DMACTL1.
- A burst-block transfer may be stopped by clearing the DMAEN bit.

8.2.5 DMA Channel Priorities

The default DMA channel priorities are DMA0–DMA1–DMA2. If two or three DMA triggers happen simultaneously or are pending, the channel with the highest priority completes its DMA transfer (single, block or burst-block transfer) first, then the second priority channel, then the third priority channel. DMA transfers in progress are not halted if a higher priority DMA channel is triggered. The higher priority channel waits until the DMA transfer in progress completes before starting.

The DMA channel priorities are configurable with the ROUNDROBIN bit. When the ROUNDROBIN bit is set, the channel that completes a DMA transfer becomes the lowest priority. The *order* of the priority of the DMA channels always stays the same, DMA0–DMA1–DMA2, for example:

DMA Priority	Transfer Occurs	New DMA Priority
DMA0 – DMA1 – DMA2	DMA1	DMA2 – DMA0 – DMA1
DMA2 – DMA0 – DMA1	DMA2	DMA0 – DMA1 – DMA2
DMA0 – DMA1 – DMA2	DMA0	DMA1 – DMA2 – DMA0

When the ROUNDROBIN bit is cleared the DMA priority returns to the default priority.

8.2.6 DMA Transfer Cycle Time

The DMA requires one or two MCLK clock cycles to synchronize before each single transfer or complete block or burst-block transfer. Each byte/word transfer requires two MCLK cycles after synchronization, and one cycle of wait time after the transfer. Because the DMA uses MCLK, the DMA cycle time is dependent on the MSP430 operating mode and clock system setup.

If the MCLK source is active, but the CPU is off, the DMA will use the MCLK source for each transfer, without re-enabling the CPU. If the MCLK source is off, the DMA will temporarily restart MCLK, sourced with DCOCLK, for the single transfer or complete block or burst-block transfer. The CPU remains off, and after the transfer completes, MCLK is turned off. The maximum DMA cycle time for all operating modes is show in Table 8–2.

Table 8–2. Maximum DMA Cycle Time

CPU Operating Mode	Clock Source	Maximum DMA Cycle Time
Active mode	MCLK=DCOCLK	4 MCLK cycles
Active mode	MCLK=LFXT1CLK	4 MCLK cycles
Low-power mode LPM0/1	MCLK=DCOCLK	5 MCLK cycles
Low-power mode LPM3/4	MCLK=DCOCLK	5 MCLK cycles + 6 µs†
Low-power mode LPM0/1	MCLK=LFXT1CLK	5 MCLK cycles
Low-power mode LPM3	MCLK=LFXT1CLK	5 MCLK cycles
Low-power mode LPM4	MCLK=LFXT1CLK	5 MCLK cycles + 6 µs†

† The additional 6 µs are needed to start the DCOCLK. It is the $t_{(LPMx)}$ parameter in the data sheet.

8.2.7 Using DMA with System Interrupts

DMA transfers are not interruptible by system interrupts. System interrupts remain pending until the completion of the DMA transfer. NMI interrupts can interrupt the DMA controller if the ENNMI bit is set.

System interrupt service routines are interrupted by DMA events. If an interrupt service routine or other routine must execute with no interruptions, the DMA controller should be disabled prior to executing the routine.

8.2.8 DMA Controller Interrupts

Each DMA channel has its own DMAIFG flag. Each DMAIFG flag is set in any mode, when the corresponding DMAxSZ register counts to zero. If the corresponding DMAIE and GIE bits are set, an interrupt request is generated.

All DMAIFG flags source only one DMA controller interrupt vector and the interrupt vector is shared with the DAC12 module. Software must check the DMAIFG and DAC12IFG flags to determine the source of the interrupt. The DMAIFG flags are not reset automatically and must be reset by software.

8.3 DMA Registers

The DMA registers are listed in Table 8–3:

Table 8–3. DMA Registers

Register	Short Form	Register Type	Address	Initial State
DMA control 0	DMACTL0	Read/write	0122h	Reset with POR
DMA control 1	DMACTL1	Read/write	0124h	Reset with POR
DMA channel 0 control	DMA0CTL	Read/write	01E0h	Reset with POR
DMA channel 0 source address	DMA0SA	Read/write	01E2h	Unchanged
DMA channel 0 destination address	DMA0DA	Read/write	01E4h	Unchanged
DMA channel 0 transfer size	DMA0SZ	Read/write	01E6h	Unchanged
DMA channel 1 control	DMA1CTL	Read/write	01E8h	Reset with POR
DMA channel 1 source address	DMA1SA	Read/write	01EAh	Unchanged
DMA channel 1 destination address	DMA1DA	Read/write	01ECh	Unchanged
DMA channel 1 transfer size	DMA1SZ	Read/write	01EEh	Unchanged
DMA channel 2 control	DMA2CTL	Read/write	01F0h	Reset with POR
DMA channel 2 source address	DMA2SA	Read/write	01F2h	Unchanged
DMA channel 2 destination address	DMA2DA	Read/write	01F4h	Unchanged
DMA channel 2 transfer size	DMA2SZ	Read/write	01F6h	Unchanged

DMACTL0, DMA Control Register 0

15	14	13	12	11	10	9	8
Reserved				DMA2TSELx			
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
DMA1TSELx				DMA0TSELx			
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

Reserved Bits 15–12 Reserved

DMA2TSELx Bits 11–8 DMA trigger select. These bits select the DMA transfer trigger.

0000 DMAREQ bit (software trigger)
0001 Timer_A CCR2 output
0010 Timer_B CCR2 output
0011 I^2C receive ready
0100 I^2C transmit ready
0101 DAC12_0CTL DAC12IFG bit
0110 ADC12 ADC12IFGx
0111 No action
1000 No action
1001 No action
1010 No action
1011 No action
1100 No action
1101 No action
1110 DMA0IFG bit triggers DMA channel 1
DMA1IFG bit triggers DMA channel 2
DMA2IFG bit trigger DMA channel 0
1111 External trigger DMAE0

DMA1TSELx Bits 7–4 Same as DMA2TSELx

DMA0TSELx Bits 3–0 Same as DMA2TSELx

DMACTL1, DMA Control Register 1

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**DMA ONFETCH**	**ROUND ROBIN**	**ENNMI**
r0	r0	r0	r0	r0	rw–(0)	rw–(0)	rw–(0)

Reserved Bits 15–3 Reserved. Read only. Always read as 0.

DMA ONFETCH Bit 2 DMA on fetch
- 0 The DMA transfer occurs immediately
- 1 The DMA transfer occurs on next instruction fetch after the trigger

ROUND ROBIN Bit 1 Round robin. This bit enables the round-robin DMA channel priorities.
- 0 DMA channel priority is DMA0 – DMA1 – DMA2
- 1 DMA channel priority changes with each transfer

ENNMI Bit 0 Enable NMI. This bit enables the interruption of a DMA transfer by an NMI interrupt. When NMI interrupts a DMA transfer, the current transfer is completed normally further transfers are stopped and DMAABORT is set.
- 0 NMI interrupt does not interrupt DMA transfer.
- 1 NMI interrupt interrupts a DMA transfer.

DMAxCTL, DMA Channel x Control Register

15	14	13	12	11	10	9	8
Reserved	**DMADTx**			**DMADSTINCRx**		**DMASRCINCRx**	
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
DMA DSTBYTE	**DMA SRCBYTE**	**DMALEVEL**	**DMAEN**	**DMAIFG**	**DMAIE**	**DMA ABORT**	**DMAREQ**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

Reserved Bit 15 Reserved

DMADTx Bits 14–12 DMA Transfer mode.
000 Single transfer
001 Block transfer
010 Burst-block transfer
011 Burst-block transfer
100 Repeated single transfer
101 Repeated block transfer
110 Repeated burst-block transfer
111 Repeated burst-block transfer

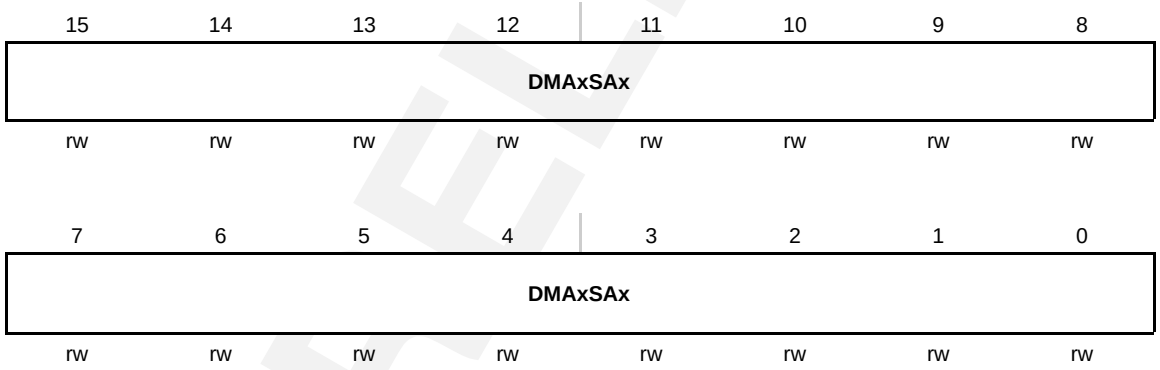
DMA DSTINCRx Bits 11–10 DMA destination increment. This bit selects automatic incrementing or decrementing of the destination address after each byte or word transfer. When DMADSTBYTE=1, the destination address increments by one. When DMADSTBYTE=0, the destination address increments by two. The DMAxDA is copied into a temporary register and the temporary register is incremented or decremented. DMAxDA is not incremented or decremented.
00 Destination address is unchanged
01 Destination address is unchanged
10 Destination address is decremented
11 Destination address is incremented

DMA SRCINCRx Bits 9–8 DMA source increment. This bit selects automatic incrementing or decrementing of the source address for each byte or word transfer. When DMASRCBYTE=1, the source address increments by one. When DMASRCBYTE=0, the source address increments by two. The DMAxSA is copied into a temporary register and the temporary register is incremented or decremented. DMAxSA is not incremented or decremented.
00 Source address is unchanged
01 Source address is unchanged
10 Source address is decremented
11 Source address is incremented

DMA DSTBYTE Bit 7 DMA destination byte. This bit selects the destination as a byte or word.
0 Word
1 Byte

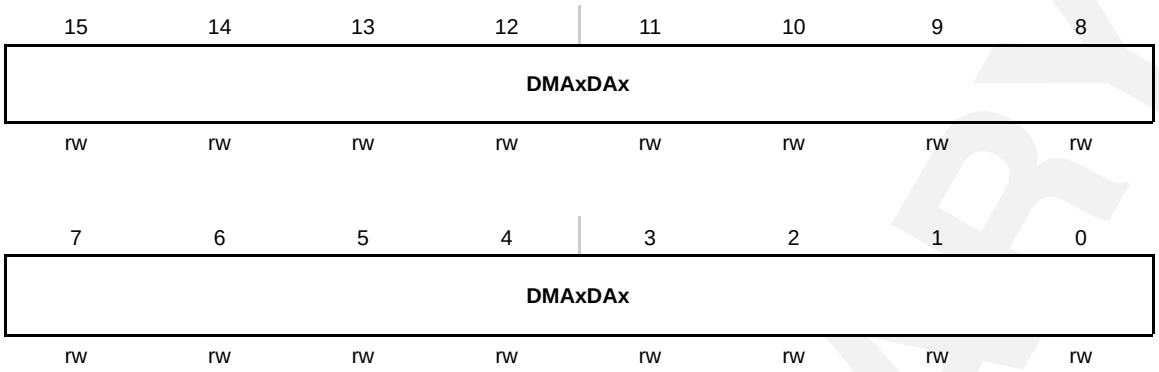
DMA SRCBYTE	Bit 6	DMA source byte. This bit selects the source as a byte or word. 0 Word 1 Byte
DMA LEVEL	Bit 5	DMA level. This bit selects between edge-sensitive and level-sensitive triggers. 0 Edge sensitive 1 Level sensitive
DMAEN	Bit 4	DMA enable 0 Disabled 1 Enabled
DMAIFG	Bit 3	DMA interrupt flag 0 No interrupt pending 1 Interrupt pending
DMAIE	Bit 2	DMA interrupt enable. 0 Disabled 1 Enabled.
DMA ABORT	Bit 1	DMA Abort. This bit indicates if a DMA transfer was interrupt by an NMI. 0 DMA transfer not interrupted 1 DMA transfer was interrupted by NMI
DMAREQ	Bit 0	DMA request. Software-controlled DMA start. DMAREQ is Reset automatically. 0 No DMA start 1 Start DMA

DMAxSA, DMA Source Address Register



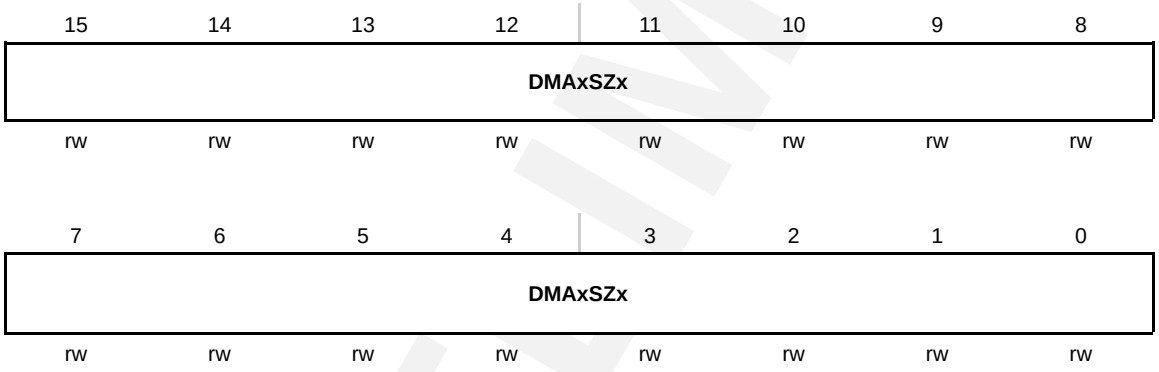
DMAxSAx	Bits 15–0	DMA source address. The source address register points to the DMA source address for single transfers or the first source address for block transfers. The source address register remains unchanged during block and burst-block transfers.

DMAxDA, DMA Destination Address Register



DMAxDAx Bits 15–0 DMA destination address. The destination address register points to the destination address for single transfers or the first address for block transfers. The DMAxDA register remains unchanged during block and burst-block transfers.

DMAxSZ, DMA Size Address Register



DMAxSZx Bits 15–0 DMA size. The DMA size register defines the number of data per block transfer. DMAxSZ register decrements with each word or byte transfer. When DMAxSZ decrements to 0, it is immediately and automatically reloaded with its previously initialized value.

00000h Transfer is disabled
00001h One byte or word is transferred
00002h Two bytes or words are transferred
:
0FFFFh 65535 bytes or words are transferred

Chapter 9

Digital I/O

This chapter describes the operation of the digital I/O ports. Ports P1-P2 are implemented in MSP430x11xx devices. Ports P1-P3 are implemented in MSP430x12xx devices. Ports P1-P6 are implemented in MSP430x13x, MSP430x14x, MSP430x15x, and MSP430x16x devices.

Topic **Page**

9.1 Digital I/O Introduction

MSP430 devices have up to 6 digital I/O ports implemented, P1 - P6. Each port has eight I/O pins. Every I/O pin is individually configurable for input or output direction, and each I/O line can be individually read or written to.

Ports P1 and P2 have interrupt capability. Each interrupt for the P1 and P2 I/O lines can be individually enabled and configured to provide an interrupt on a rising edge or falling edge of an input signal. All P1 I/O lines source a single interrupt vector, and all P2 I/O lines source a different, single interrupt vector.

The digital I/O features include:

- Independently programmable individual I/Os
- Any combination of input or output
- Individually configurable P1 and P2 interrupts
- Independent input and output data registers

9.2 Digital I/O Operation

The digital I/O is configured with user software. The setup and operation of the digital I/O is discussed in the following sections.

9.2.1 Input Register PnIN

Each bit in each PnIN register reflects the value of the input signal at the corresponding I/O pin when the pin is configured as I/O function.

Bit = 0: The input is low

Bit = 1: The input is high

Note: Writing to Read-Only Registers PxIN

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

9.2.2 Output Registers PnOUT

Each bit in each PnOUT register is the value to be output on the corresponding I/O pin when the pin is configured as I/O function and output direction.

Bit = 0: The output is low

Bit = 1: The output is high

9.2.3 Direction Registers PnDIR

Each bit in each PnDIR register selects the direction of the corresponding I/O pin, regardless of the selected function for the pin. PnDIR bits for I/O pins that are selected for other module functions must be set as required by the other function.

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

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

9.2.4 Function Select Registers PnSEL

Port pins are often multiplexed with other peripheral module functions. See the device-specific data sheet to determine pin functions. Each PnSEL bit is used to select the pin function – I/O port or peripheral module function.

Bit = 0: I/O Function is selected for the pin

Bit = 1: Peripheral module function is selected for the pin

Setting PnSELx = 1 does not automatically set the pin direction. Other peripheral module functions require the PnDIRx bits to be configured according to the direction needed for the module function.

```
;Output ACLK on P2.0 on MSP430F11x1
   BIS.B #01h,&P2SEL  ; Select ACLK function for pin
   BIS.B #01h,&P2DIR  ; Set direction to output *Required*
```

Note: P1 and P2 Interrupts Are Disabled When PnSEL = 1

When any P1SELx or P2SELx bit is set, the corresponding pin's interrupt function is disabled. Therefore, signals on these pins will not generate P1 or P2 interrupts, regardless of the state of the corresponding P1IE or P2IE bit.

When a port pin is selected as an input to a peripheral, the input signal to the peripheral is a latched representation of the signal at the device pin. While PnSELx=1, the internal input signal follows the signal at the pin. However, if the PnSELx=0, the input to the peripheral maintains the value of the input signal at the device pin before the PnSELx bit was reset.

9.2.5 P1 and P2 Interrupts

Each pin in ports P1 and P2 have interrupt capability, configured with the PnIFG, PnIE, and PnIES registers. All P1 pins source a single interrupt vector, and all P2 pins source a different single interrupt vector. The PnIFG register can be tested to determine the source of a P1 or P2 interrupt.

Interrupt Flag Registers P1IFG, P2IFG

Each PnIFGx bit is the interrupt flag for its corresponding I/O pin and is set when the selected input signal edge occurs at the pin. All PnIFGx interrupt flags request an interrupt when their corresponding PnIE bit and the GIE bit are set. Each PnIFG flag must be reset with software. Software can also set each PnIFG flag, providing a way to generate a software initiated interrupt.

Bit = 0: No interrupt is pending

Bit = 1: An interrupt is pending

Only transitions, not static levels, cause interrupts. If any PnIFGx flag becomes set during a Px interrupt service routine, or is set after the `RETI` instruction of a Px interrupt service routine is executed, the set PnIFGx flag generates another interrupt. This ensures that each transition is acknowledged.

Note: PnIFG Flags When Changing PnOUT or PnDIR

Writing to P1OUT, P1DIR, P2OUT, or P2DIR can result in setting the corresponding P1IFG or P2IFG flags.

Note: Length of I/O Pin Interrupt Event

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.

Interrupt Edge Select Registers P1IES, P2IES

Each PnIES bit selects the interrupt edge for the corresponding I/O pin.

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

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

Note: Writing to PnIESx

Writing to P1IES, or P2IES can result in setting the corresponding interrupt flags.

PnIESx	PnINx	PnIFGx
0 → 1	0	Unchanged
0 → 1	1	May be set
1 → 0	0	May be set
1 → 0	1	Unchanged

Interrupt Enable P1IE, P2IE

Each PnIE bit enables the associated PnIFG interrupt flag.

Bit = 0: The interrupt is disabled

Bit = 1: The interrupt is enabled

9.2.6 Configuring Unused Port Pins

Unused I/O pins should be configured as I/O function, output direction, and left unconnected on the PC board, to reduce power consumption. The value of the PxOUT bit is don't care, since the pin is unconnected. See chapter *System Resets, Interrupts, and Operating Modes* for termination unused pins.

9.3 Digital I/O Registers

Seven registers are used to configure P1 and P2. Four registers are used to configure ports P3 - P6. The digital I/O registers are listed in Table 9–1.

Table 9–1. Digital I/O Registers

Port	Register	Short Form	Address	Register Type	Initial State
P1	Input	P1IN	020h	Read only	–
	Output	P1OUT	021h	Read/write	Unchanged
	Direction	P1DIR	022h	Read/write	Reset with PUC
	Interrupt Flag	P1IFG	023h	Read/write	Reset with PUC
	Interrupt Edge Select	P1IES	024h	Read/write	Unchanged
	Interrupt Enable	P1IE	025h	Read/write	Reset with PUC
	Port Select	P1SEL	026h	Read/write	Reset with PUC
P2	Input	P2IN	028h	Read only	–
	Output	P2OUT	029h	Read/write	Unchanged
	Direction	P2DIR	02Ah	Read/write	Reset with PUC
	Interrupt Flag	P2IFG	02Bh	Read/write	Reset with PUC
	Interrupt Edge Select	P2IES	02Ch	Read/write	Unchanged
	Interrupt Enable	P2IE	02Dh	Read/write	Reset with PUC
	Port Select	P2SEL	02Eh	Read/write	Reset with PUC
P3	Input	P3IN	018h	Read only	–
	Output	P3OUT	019h	Read/write	Unchanged
	Direction	P3DIR	01Ah	Read/write	Reset with PUC
	Port Select	P3SEL	01Bh	Read/write	Reset with PUC
P4	Input	P4IN	01Ch	Read only	–
	Output	P4OUT	01Dh	Read/write	Unchanged
	Direction	P4DIR	01Eh	Read/write	Reset with PUC
	Port Select	P4SEL	01Fh	Read/write	Reset with PUC
P5	Input	P5IN	030h	Read only	–
	Output	P5OUT	031h	Read/write	Unchanged
	Direction	P5DIR	032h	Read/write	Reset with PUC
	Port Select	P5SEL	033h	Read/write	Reset with PUC
P6	Input	P6IN	034h	Read only	–
	Output	P6OUT	035h	Read/write	Unchanged
	Direction	P6DIR	036h	Read/write	Reset with PUC
	Port Select	P6SEL	037h	Read/write	Reset with PUC

Chapter 10

Watchdog Timer

The watchdog timer is a 16-bit timer that can be used as a watchdog or as an interval timer. This chapter describes the watchdog timer. The watchdog timer is implemented in all MSP430x1xx devices.

Topic **Page**

10.1 Watchdog Timer Introduction

The primary function of the watchdog timer (WDT) module 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 be configured as an interval timer and can generate interrupts at selected time intervals.

Features of the watchdog timer module include:

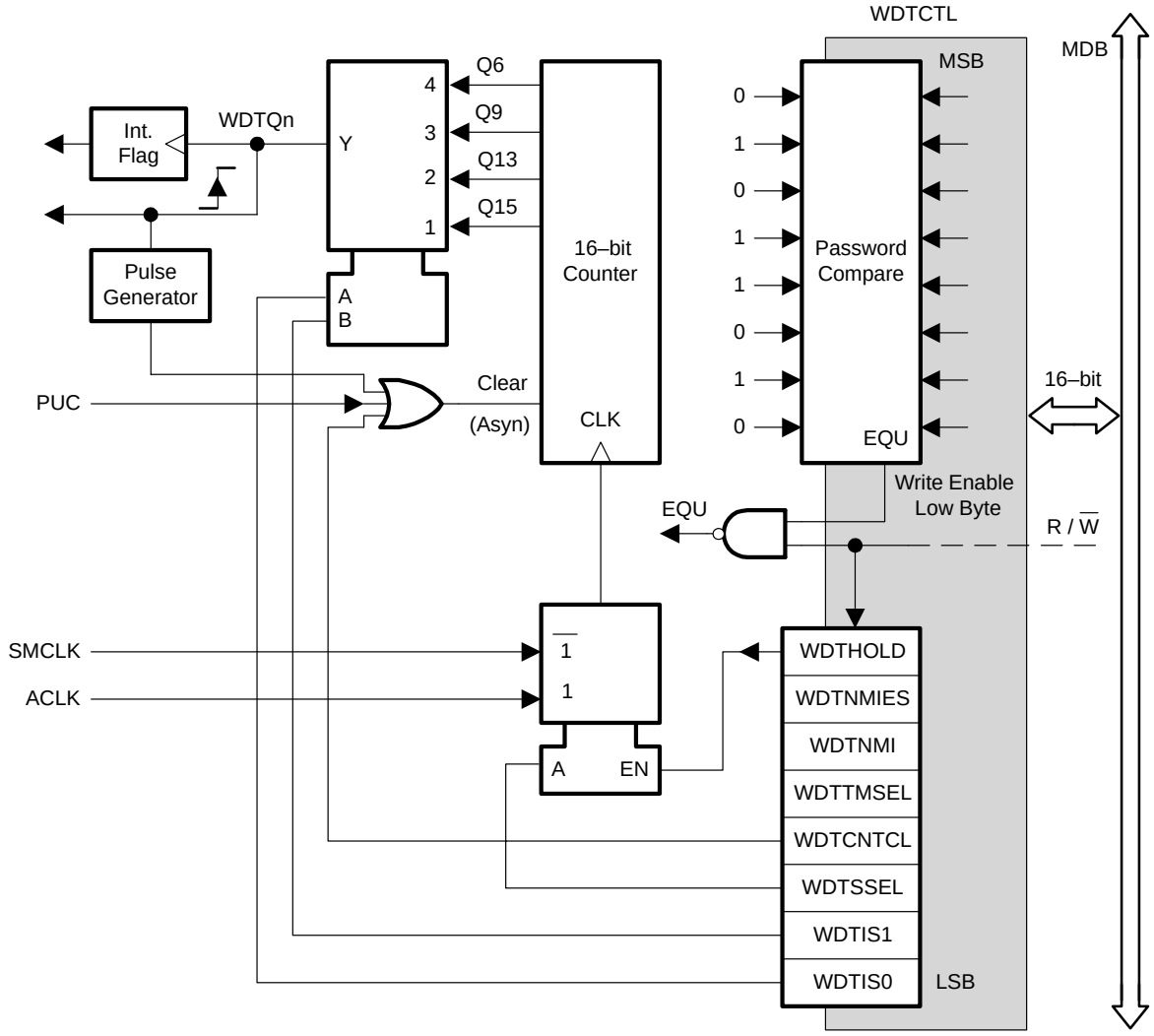
- Eight software-selectable time intervals
- Watchdog mode
- Interval mode
- Access to WDT control register is password protected
- Control of $\overline{\text{RST}}$/NMI pin function
- Selectable clock source
- Can be stopped to conserve power

The WDT block diagram is shown in Figure 9–1.

Note: Watchdog Timer Powers Up Active

After a PUC, the WDT module is automatically configured in the watchdog mode with an initial ~32-ms reset interval using the DCOCLK. The user must setup or halt the WDT prior to the expiration of the initial reset interval.

Figure 10–1. Watchdog Timer Block Diagram



10.2 Watchdog Timer Operation

The WDT module can be configured as either a watchdog or interval timer with the WDTCTL register. The WDTCTL register also contains control bits to configure the $\overline{\text{RST}}$/NMI pin. WDTCTL is a 16-bit, password-protected, read/write register. Any read or write access must use word instructions and write accesses must include the write password 05Ah in the upper byte. Any write to WDTCTL with any value other than 05Ah in the upper byte is a security key violation and triggers a PUC system reset regardless of timer mode. Any read of WDTCTL reads 069h in the upper byte.

10.2.1 Watchdog Timer Counter

The watchdog timer counter (WDTCNT) is a 16-bit up-counter that is not directly accessible by software. The WDTCNT is controlled and time intervals selected through the watchdog timer control register WDTCTL.

The WDTCNT can be sourced from ACLK or SMCLK. The clock source is selected with the WDTSSEL bit.

10.2.2 Watchdog Mode

After a PUC condition, the WDT module is configured in the watchdog mode with an initial ~32-ms reset interval using the DCOCLK. The user must setup, halt, or clear the WDT prior to the expiration of the initial reset interval or another PUC will be generated. When the WDT is configured to operate in watchdog mode, either writing to WDTCTL with an incorrect password, or expiration of the selected time interval triggers a PUC. A PUC resets the WDT to its default condition and configures the $\overline{\text{RST}}$/NMI pin to reset mode.

10.2.3 Interval Timer Mode

Setting the WDTTMSEL bit to 1 selects the interval timer mode. This mode can be used to provide periodic interrupts. In interval timer mode, the WDTIFG flag is set at the expiration of the selected time interval. A PUC is not generated in interval timer mode at expiration of the selected timer interval and the WDTIFG enable bit WDTIE remains unchanged.

When the WDTIE bit and the GIE bit are set, the WDTIFG flag requests an interrupt. The WDTIFG interrupt flag is automatically reset when its interrupt request is serviced, or may be reset by software. The interrupt vector address in interval timer mode is different from that in watchdog mode.

Note: Modifying the Watchdog Timer

The WDT interval should be changed together with WDTCNTCL = 1 in a single instruction to avoid an unexpected immediate PUC or interrupt.

The WDT should be halted before changing the clock source to avoid a possible incorrect interval.

10.2.4 Watchdog Timer Interrupts

The WDT uses two bits in the SFRs for interrupt control.

- The WDT interrupt flag, WDTIFG, located in IFG1.0
- The WDT interrupt enable, WDTIE, located in IE1.0

When using the WDT in the watchdog mode, the WDTIFG flag sources a reset vector interrupt. The WDTIFG can be 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 WDTIFG is cleared, the reset was caused by a different source.

When using the WDT in interval timer mode, the WDTIFG flag is set after the selected time interval and requests a WDT interval timer interrupt if the WDTIE and the GIE bits are set. The interval timer interrupt vector is different from the reset vector used in watchdog mode. In interval timer mode, the WDTIFG flag is reset automatically when the interrupt is serviced, or can be reset with software.

10.2.5 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 used determine how the WDT should be configured. 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 and the WDT would not function. When the watchdog timer is not required, the WDTHOLD bit can be used to hold the WDTCNT, reducing power consumption.

10.2.6 Software Examples

Any write operation to WDTCTL must be a word operation with 05Ah (WDTPW) in the upper byte:

```
; Periodically clear an active watchdog
      MOV #WDTPW+WDTCNTCL,&WDTCTL
;
; Change watchdog timer interval
      MOV #WDTPW+WDTCNTL+SSEL,&WDTCTL
;
; Stop the watchdog
      MOV #WDTPW+WDTHOLD,&WDTCTL
;
; Change WDT to interval timer mode, clock/8192 interval
      MOV #WDTPW+WDTCNTCL+WDTTMSEL+WDTIS0,&WDTCTL
```

10.3 Watchdog Timer Registers

The watchdog timer module registers are listed in Table 10–1.

Table 10–1. Watchdog Timer Registers

Register	Short Form	Register Type	Address	Initial State
Watchdog timer control register	WDTCTL	Read/write	0120h	06900h with PUC
SFR interrupt enable register 1	IE1	Read/write	0000h	Reset with PUC
SFR interrupt flag register 1	IFG1	Read/write	0002h	Reset with PUC[1]

1) WDTIFG is reset with POR

WDTCTL, Watchdog Timer Register

15	14	13	12	11	10	9	8
Read as 069h WDTPWx, must be written as 05Ah							

7	6	5	4	3	2	1	0
WDTHOLD	**WDTNMIES**	**WDTNMI**	**WDTTMSEL**	**WDTCNTCL**	**WDTSSEL**	**WDTISx**	
rw–0	rw–0	rw–0	rw–0	r0(w)	rw–0	rw–0	rw–0

WDTPWx Bits 15-8 Watchdog timer password. Always read as 069h. Must be written as 05Ah, or a PUC will be generated.

WDTHOLD Bit 7 Watchdog timer hold. This bit stops the watchdog timer. Setting WDTHOLD = 1 when the WDT is not in use conserves power.
0 Watchdog timer is not stopped
1 Watchdog timer is stopped

WDTNMIES Bit 6 Watchdog timer NMI edge select. This bit selects the interrupt edge for the NMI interrupt when WDTNMI = 1. Modifying this bit can trigger an NMI. Modify this bit when WDTNMI = 0 to avoid triggering an accidental NMI.
0 NMI on rising edge
1 NMI on falling edge

WDTNMI Bit 5 Watchdog timer NMI select. This bit selects the function for the $\overline{\text{RST}}$/NMI pin.
0 Reset function
1 NMI function

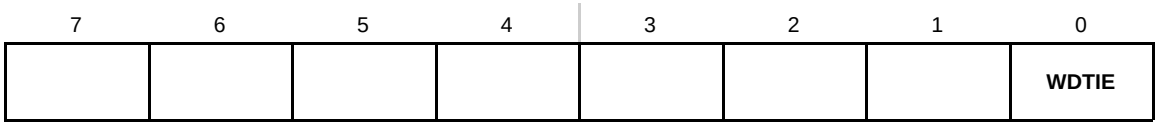
WDTTMSEL Bit 4 Watchdog timer mode select
0 Watchdog mode
1 Interval timer mode

WDTCNTCL Bit 3 Watchdog timer counter clear. Setting WDTCNTCL = 1 clears the count value to 0000h. WDTCNTCL is automatically reset.
0 No action
1 WDTCNT = 0000h

WDTSSEL Bit 2 Watchdog timer clock source select
0 SMCLK
1 ACLK

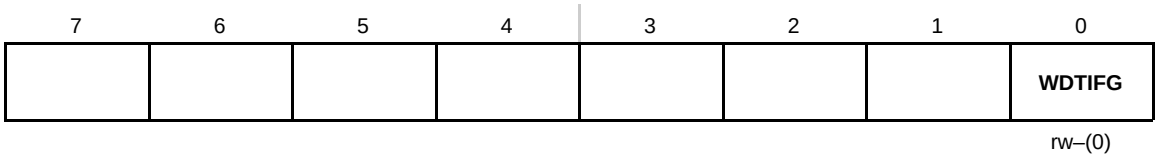
WDTISx Bits 1-0 Watchdog timer interval select. These bits select the watchdog timer interval to set the WDTIFG flag and/or generate a PUC.
00 Watchdog clock source /32768
01 Watchdog clock source /8192
10 Watchdog clock source /512
11 Watchdog clock source /64

IE1, Interrupt Enable Register 1



	Bits 7-1	These bits may be used by other modules. See device-specific datasheet.
WDTIE	Bit 0	Watchdog timer interrupt enable. This bit enables the WDTIFG interrupt for interval timer mode. It is not necessary to set this bit for watchdog mode. Because other bits in IE1 may be used for other modules, it is recommended to set or clear this bit using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions. 0 Interrupt not enabled 1 Interrupt enabled

IFG1, Interrupt Flag Register 1



	Bits 7-1	These bits may be used by other modules. See device-specific datasheet.
WDTIFG	Bit 0	Watchdog timer interrupt flag. In watchdog mode, WDTIFG remains set until reset by software. In interval mode, WDTIFG is reset automatically by servicing the interrupt, or can be reset by software. Because other bits in IFG1 may be used for other modules, it is recommended to clear WDTIFG by using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions. 0 No interrupt pending 1 Interrupt pending

Chapter 11

Timer_A

Timer_A is a 16-bit timer/counter with three capture/compare registers This chapter describes Timer_A. Timer_A is implemented in all MSP430x1xx devices.

Topic **Page**

11.1 Timer_A Introduction

Timer_A is a 16-bit timer/counter with three capture/compare registers. Timer_A can support multiple capture/compares, PWM outputs, and interval timing. Timer_A also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers.

Timer_A features include:

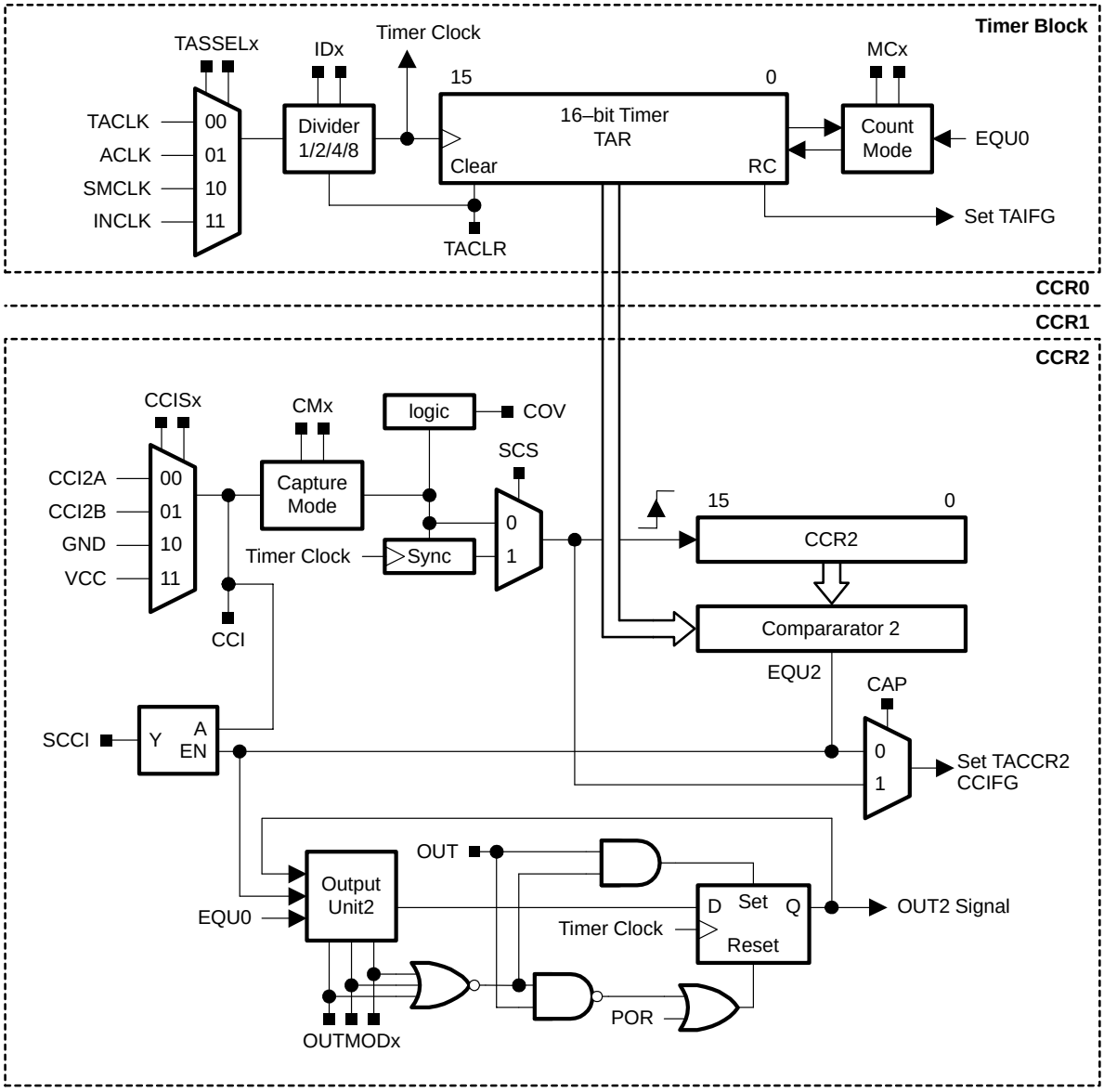
- ❑ Asynchronous 16-bit timer/counter with four operating modes
- ❑ Selectable and configurable clock source
- ❑ Three configurable capture/compare registers
- ❑ Configurable outputs with PWM capability
- ❑ Asynchronous input and output latching
- ❑ Interrupt vector register for fast decoding of all Timer_A interrupts

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

Note: Use of the Word *Count*

C*ount* is used throughout this chapter. It means 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 an associated action will not take place.

Figure 11–1. Timer_A Block Diagram



11.2 Timer_A Operation

The Timer_A module is configured with user software. The setup and operation of Timer_A is discussed in the following sections.

11.2.1 16-Bit Timer Counter

The 16-bit timer/counter register, TAR, increments or decrements (depending on mode of operation) with each rising edge of the clock signal. TAR can be read or written with software. Additionally, the timer can generate an interrupt when it overflows.

TAR may be cleared by setting the TACLR bit. Setting TACLR also clears the clock divider and count direction for up/down mode.

Note: Modifying Timer_A Registers

It is recommended to stop the timer before modifying its operation (with exception of the interrupt enable and interrupt flag) to avoid errant operating conditions.

When the TACLK is asynchronous to the CPU clock, any read from TAR should occur while the timer is not operating or the results may be unpredictable. Any write to TAR will take effect immediately.

Clock Source Select and Divider

The timer clock TACLK can be sourced from ACLK, SMCLK, or externally via TACLK or INCLK. The clock source is selected with the TASSELx bits. The selected clock source may be passed directly to the timer or divided by 2, 4, or 8, using the IDx bits.

11.2.2 Starting the Timer

The timer may be started, or restarted in the following ways:

- The timer counts when MCx > 0 and the clock source is active.
- When the timer mode is either up or up/down, the timer may be stopped by writing 0 to TACCR0. The timer may then be restarted by writing a nonzero value to TACCR0. In this scenario, the timer starts incrementing in the up direction from zero.

11.2.3 Timer Mode Control

The timer has four modes of operation as described in Table 11–1: stop, up, continuous, and up/down. The operating mode is selected with the MCx bits.

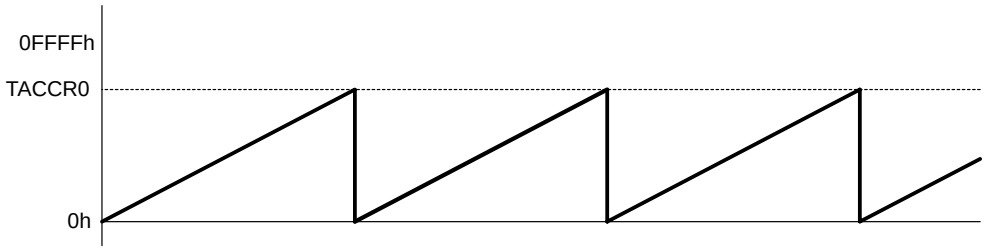
Table 11–1. Timer Modes

MCx	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TACCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh.
11	Up/down	The timer repeatedly counts from zero up to the value of TACCR0 and back down to zero.

Up Mode

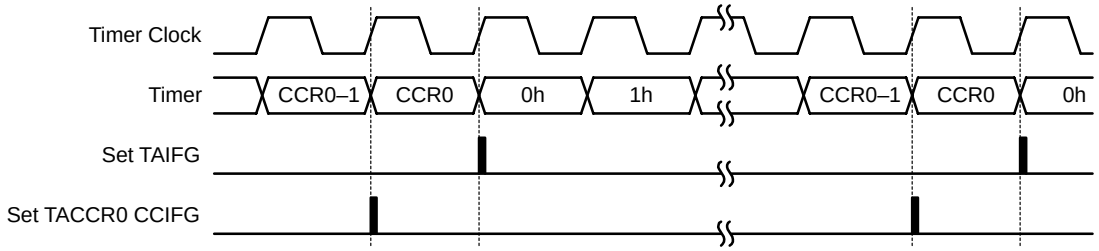
The up mode is used if the timer period must be different from 0FFFFh counts. The timer repeatedly counts up to the value of compare register TACCR0, which defines the period, as shown in Figure 11–2. The number of timer counts in the period is TACCR0+1. When the timer value equals TACCR0 the timer restarts counting from zero. If up mode is selected when the timer value is greater than TACCR0, the timer immediately restarts counting from zero.

Figure 11–2. Up Mode



The TACCR0 CCIFG interrupt flag is set when the timer equals the TACCR0 value. The TAIFG interrupt flag is set when the timer *counts* from TACCR0 to zero. Figure 11–3 shows the flag set cycle.

Figure 11–3. Up Mode Flag Setting



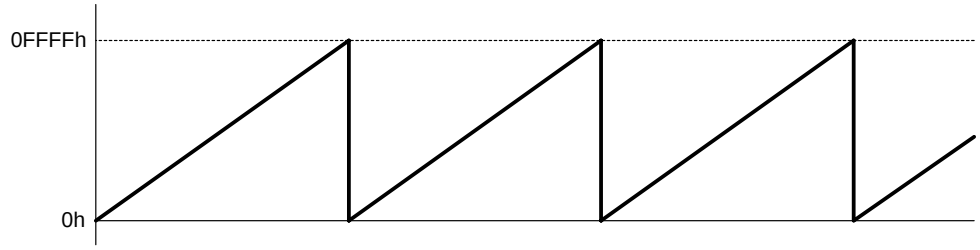
Changing the Period Register TACCR0

When changing TACCR0 while the timer is running, if the new period is greater than or equal to the old period, or greater than the current count value, the timer counts up to the new period. If the new period is less than the current count value, the timer rolls to zero. However, one additional count may occur before the counter rolls to zero.

Continuous Mode

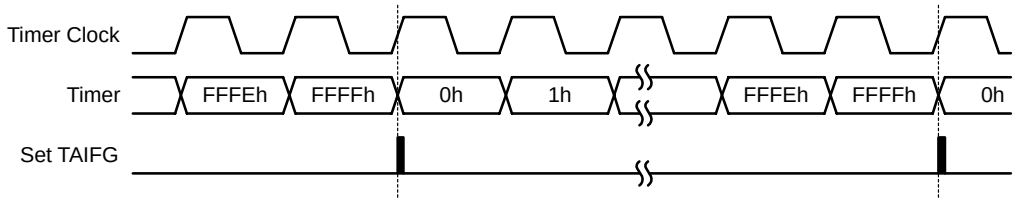
In the continuous mode, the timer repeatedly counts up to 0FFFFh and restarts from zero as shown in Figure 11–4. The capture/compare register TACCR0 works the same way as the other capture/compare registers.

Figure 11–4. Continuous Mode



The TAIFG interrupt flag is set when the timer *counts* from 0FFFFh to zero. Figure 11–5 shows the flag set cycle.

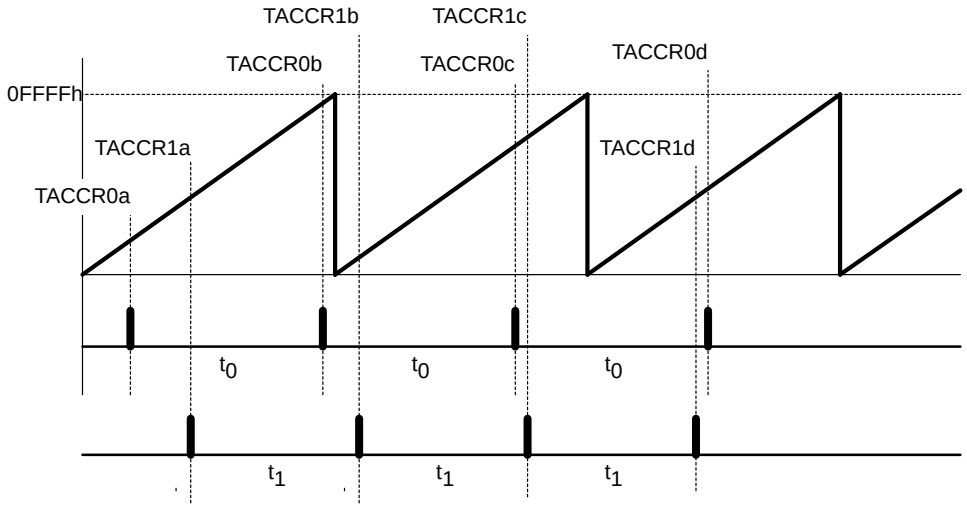
Figure 11–5. Continuous Mode Flag Setting



Use of the Continuous Mode

The continuous mode can be used to generate independent time intervals and output frequencies. Each time an interval is completed, an interrupt is generated. The next time interval is added to the TACCRx register in the interrupt service routine. Figure 11–6 shows two separate time intervals t_0 and t_1 being added to the capture/compare registers. In this usage, the time interval is controlled by hardware, not software, without impact from interrupt latency. Up to three independent time intervals or output frequencies can be generated using all three capture/compare registers.

Figure 11–6. Continuous Mode Time Intervals

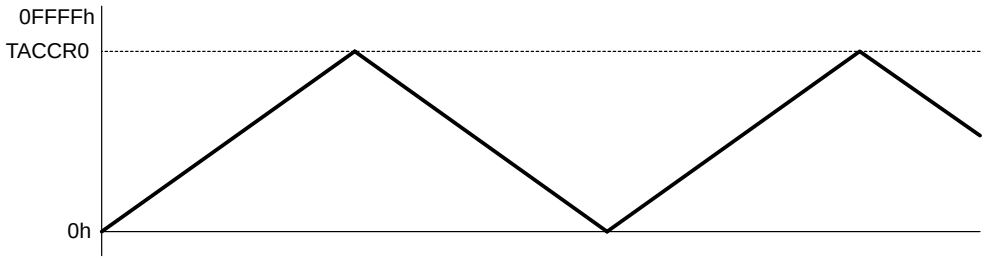


Time intervals can be produced with other modes as well, where TACCR0 is used as the period register. Their handling is more complex since the sum of the old TACCRx data and the new period can be higher than the TACCR0 value. When the previous TACCRx value plus t_x is greater than the TACCR0 data, the TACCR0 value must be subtracted to obtain the correct time interval.

Up/Down Mode

The up/down mode is used if the timer period must be different from 0FFFFh counts, and if symmetrical pulse generation is needed. The timer repeatedly counts up to the value of compare register TACCR0 and back down to zero, as shown in Figure 11–7. The period is twice the value in TACCR0.

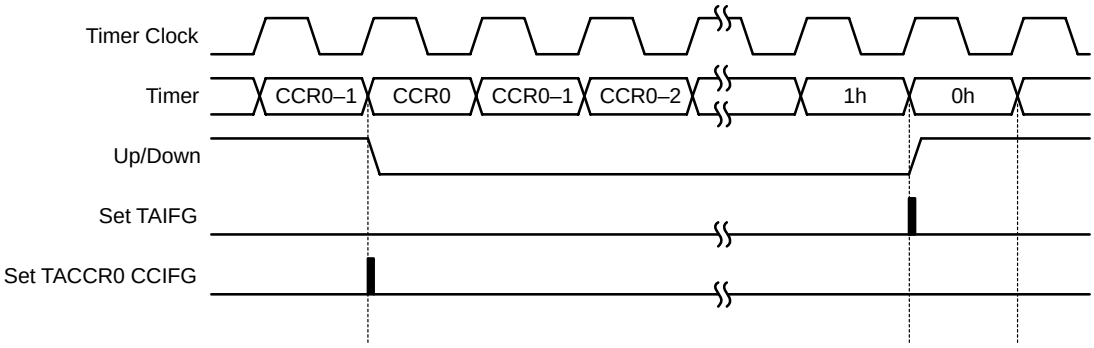
Figure 11–7. Up/Down Mode



The count direction is latched. This allows the timer to be stopped and then restarted in the same direction it was counting before it was stopped. If this is not desired, the TACLR bit must be set to clear the direction. The TACLR bit also clears the TAR value and the TACLK divider.

In up/down mode, the TACCR0 CCIFG interrupt flag and the TAIFG interrupt flag are set only once during a period, separated by 1/2 the timer period. The TACCR0 CCIFG interrupt flag is set when the timer *counts* from TACCR0–1 to TACCR0, and TAIFG is set when the timer completes *counting* down from 0001h to 0000h. Figure 11–8 shows the flag set cycle.

Figure 11–8. Up/Down Mode Flag Setting



Changing the Period Register TACCR0

When changing TACCR0 while the timer is running, and counting in the down direction, the timer continues its descent until it reaches zero. The new period takes affect after the counter counts down to zero.

When the timer is counting in the up direction, and the new period is greater than or equal to the old period, or greater than the current count value, the timer counts up to the new period before counting down. When the timer is counting in the up direction, and the new period is less than the current count value, the timer begins counting down. However, one additional count may occur before the counter begins counting down.

Use of the Up/Down Mode

The up/down mode supports applications that require dead times between output signals (See section *Timer_A Output Unit*). For example, to avoid overload conditions, two outputs driving an H-bridge must never be in a high state simultaneously. In the example shown in Figure 11–9 the t_{dead} is:

$$t_{dead} = t_{timer} \times (TACCR1 - TACCR2)$$

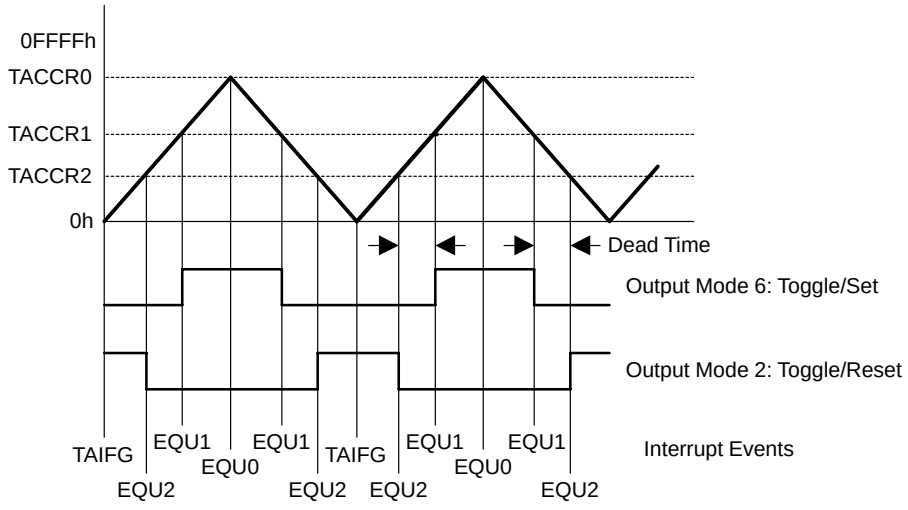
With: t_{dead} Time during which both outputs need to be inactive

t_{timer} Cycle time of the timer clock

TACCRx Content of capture/compare register x

The TACCRx registers are not buffered. They update immediately when written to. Therefore, any required dead time will not be maintained automatically.

Figure 11–9. Output Unit in Up/Down Mode



11.2.4 Capture/Compare Blocks

Three identical capture/compare blocks, TACCRx, are present in Timer_A. Any of the blocks may be used to capture the timer data, or to generate time intervals.

Capture Mode

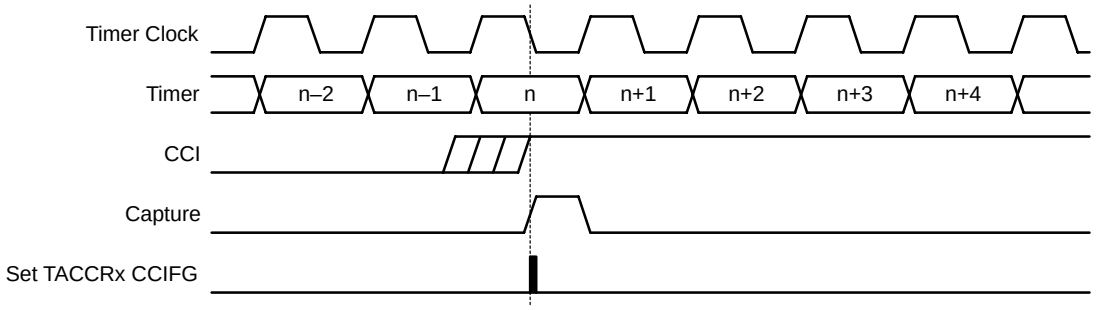
The capture mode is selected when CAP = 1. Capture mode is used to record time events. It can be used for speed computations or time measurements. The capture inputs CCIxA and CCIxB are connected to external pins or internal signals and are selected with the CCISx bits. The CMx bits select the capture edge of the input signal as rising, falling, or both. A capture occurs on the selected edge of the input signal. If a capture occurs:

- The timer value is copied into the TACCRx register
- The interrupt flag CCIFG is set

The input signal level can be read at any time by via the CCI bit. MSP430x1xx family devices may have different signals connected to CCIxA and CCIxB. Refer to the device-specific datasheet for the connections of these signals.

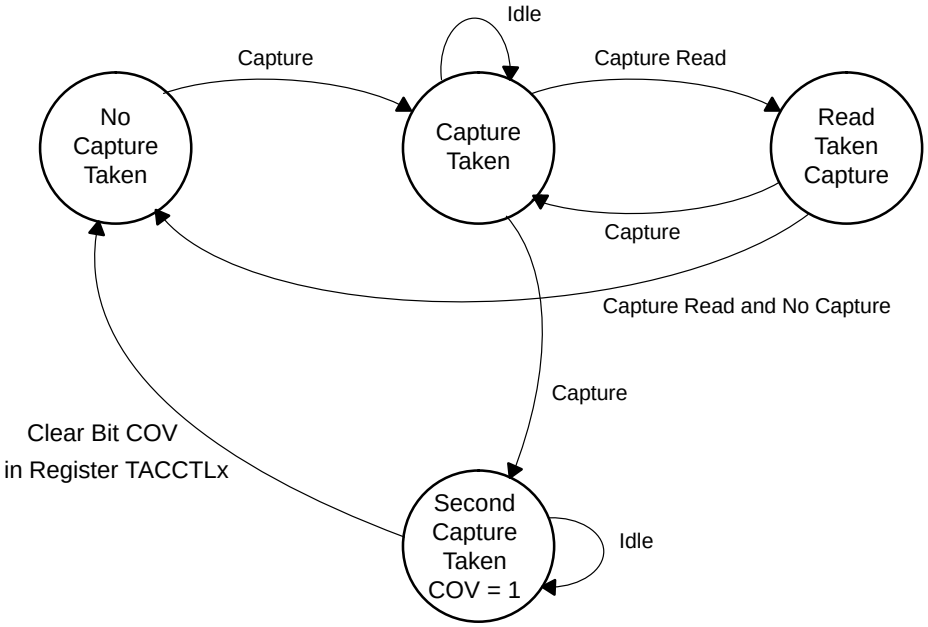
The capture signal can be asynchronous to the timer clock and cause a race condition. Setting the SCS bit will synchronize the capture with the next timer clock. Setting the SCS bit to synchronize the capture signal with the timer clock is recommended. This is illustrated in Figure 11–10.

Figure 11–10. Capture Signal (SCS=1)



Overflow logic is provided in each capture/compare register to indicate if a second capture was performed before the value from the first capture was read. Bit COV is set when this occurs as shown in Figure 11–11. COV must be reset with software.

Figure 11–11. Capture Cycle



Capture Initiated by Software

Captures can be initiated by software. The CMx bits can be set for capture on both edges. Software then sets CCI = 1 and toggles bit CCIS0 to switch the capture signal between V_{CC} and GND, initiating a capture each time CCIS0 changes state:

```
MOV   #CAP+SCS+CCIS1+CM_3,&TACCTLx ; Setup TACCTLx
XOR   #CCIS0,&TACCTLx              ; TACCTLx = TAR
```

Compare Mode

The compare mode is selected when CAP = 0. The compare mode is used to generate PWM output signals or interrupts at specific time intervals. When TAR *counts* to the value in a TACCRx:

- ❏ Interrupt flag CCIFG is set
- ❏ Internal signal EQUx = 1
- ❏ EQUx affects the output according to the output mode
- ❏ The input signal CCI is latched into SCCI

11.2.5 Output Unit

Each capture/compare block contains an output unit. The output unit is used to generate output signals such as PWM signals. Each output unit has eight operating modes that generate signals based on the EQU0 and EQUx signals.

Output Modes

The output modes are defined by the OUTMODx bits and are described in Table 11–2. 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 because EQUx = EQU0.

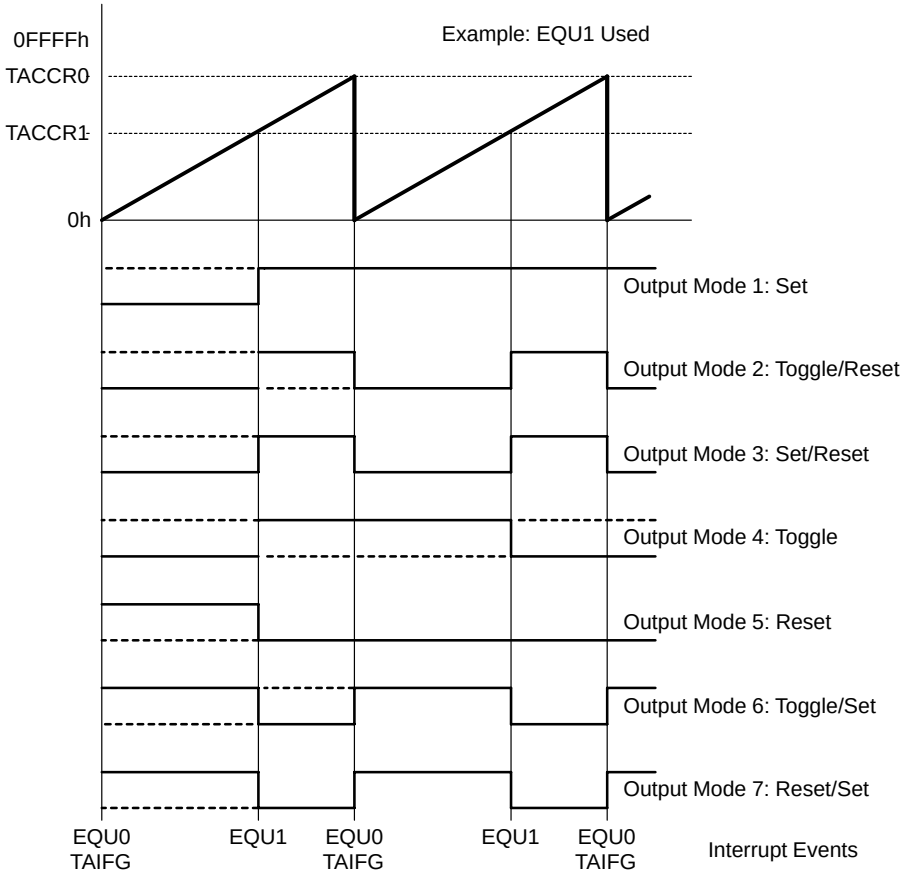
Table 11–2. Output Modes

OUTMODx	Mode	Description
000	Output	The output signal OUTx is defined by the OUTx bit. The OUTx signal updates immediately when OUTx is updated.
001	Set	The output is set when the timer *counts* to the TACCRx value. It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	The output is toggled when the timer *counts* to the TACCRx value. It is reset when the timer *counts* to the TACCR0 value.
011	Set/Reset	The output is set when the timer *counts* to the TACCRx value. It is reset when the timer *counts* to the TACCR0 value.
100	Toggle	The output is toggled when the timer *counts* to the TACCRx value. The output period is double the timer period.
101	Reset	The output is reset when the timer *counts* to the TACCRx value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer *counts* to the TACCRx value. It is set when the timer *counts* to the TACCR0 value.
111	Reset/Set	The output is reset when the timer *counts* to the TACCRx value. It is set when the timer *counts* to the TACCR0 value.

Output Example—Timer in Up Mode

The OUTx signal is changed when the timer *counts* up to the TACCRx value, and rolls from TACCR0 to zero, depending on the output mode. An example is shown in Figure 11–12 using TACCR0 and TACCR1.

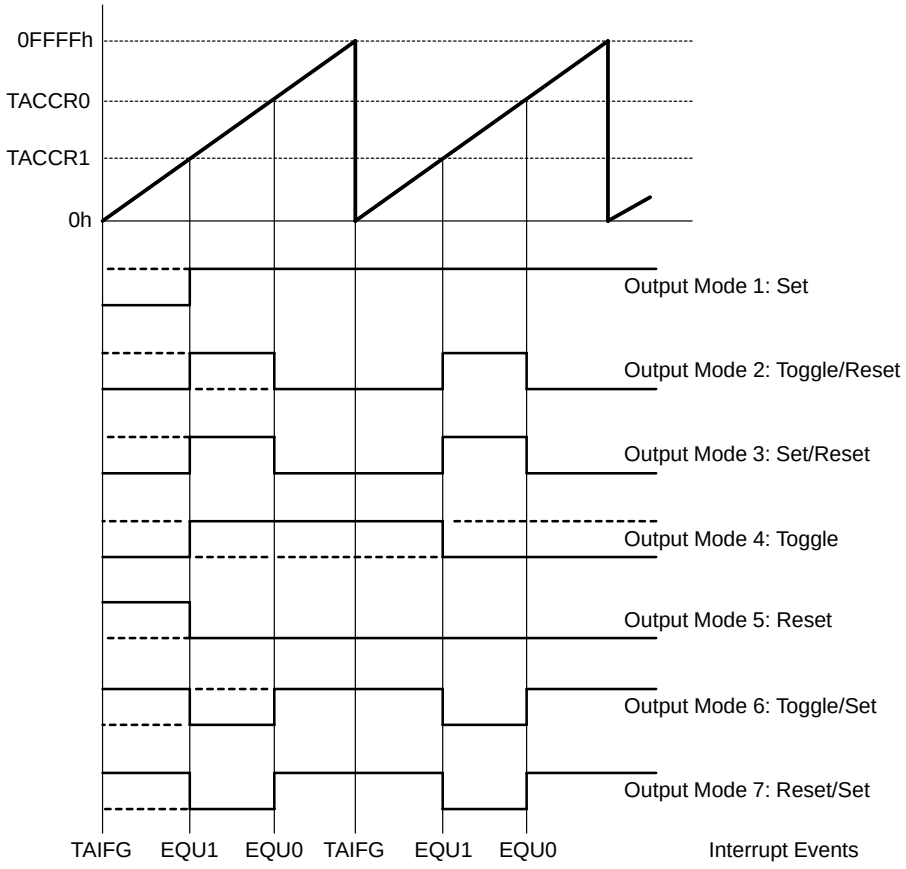
Figure 11–12. Output Example—Timer in Up Mode



Output Example—Timer in Continuous Mode

The OUTx signal is changed when the timer reaches the TACCRx and TACCR0 values, depending on the output mode. An example is shown in Figure 11–13 using TACCR0 and TACCR1.

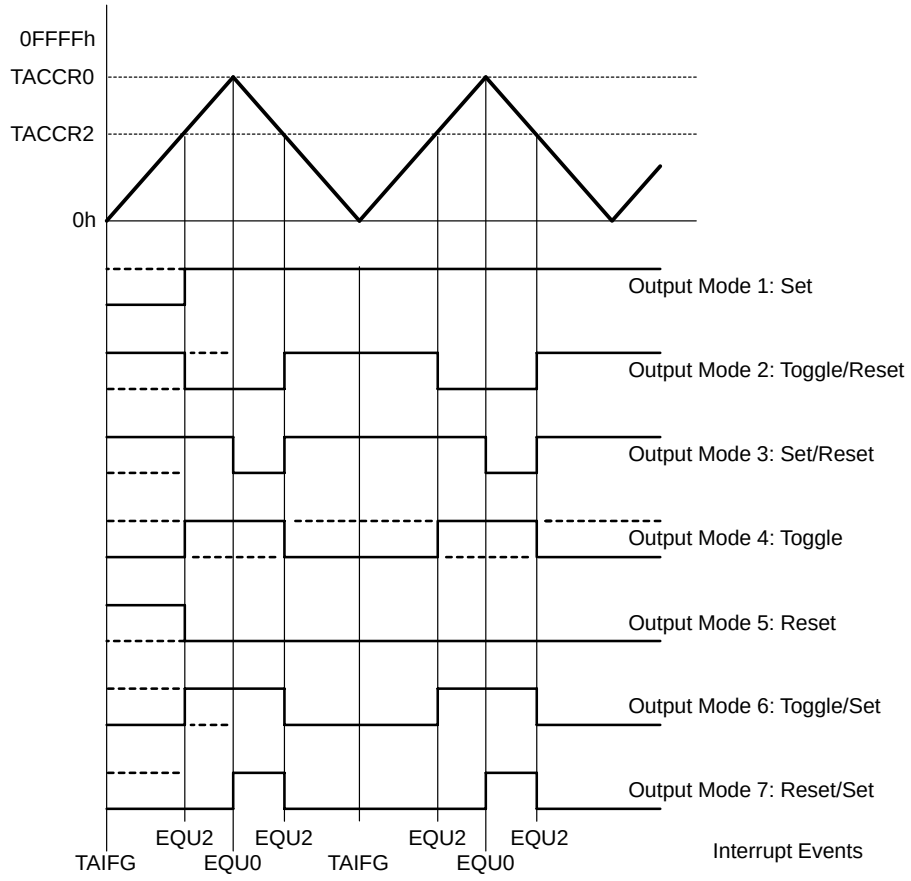
Figure 11–13. Output Example—Timer in Continuous Mode



Output Example—Timer in Up/Down Mode

The OUTx signal changes when the timer equals TACCRx in either count direction and when the timer equals TACCR0, depending on the output mode. An example is shown in Figure 11–14 using TACCR0 and TACCR2.

Figure 11–14. Output Example—Timer in Up/Down Mode



Note: *Switching Between Output Modes*

When switching between output modes, one of the OUTMODx bits should remain set during the transition, unless switching to mode 0. Otherwise, output glitching can occur because a NOR gate decodes output mode 0. A safe method for switching between output modes is to use output mode 7 as a transition state:

```
BIS   #OUTMOD_7,&TACCTLx ; Set output mode=7
BIC   #OUTMODx,&TACCTLx  ; Clear unwanted bits
```

11.2.6 Timer_A Interrupts

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

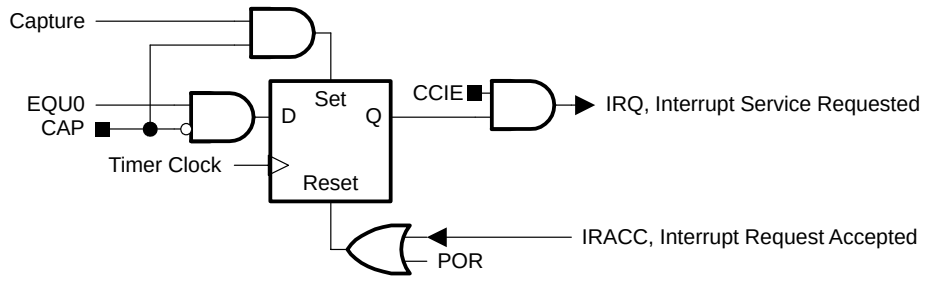
- TACCR0 interrupt vector for TACCR0 CCIFG
- TAIV interrupt vector for all other CCIFG flags and TAIFG

In capture mode any CCIFG flag is set when a timer value is captured in the associated TACCRx register. In compare mode, any CCIFG flag is set if TAR *counts* to the associated TACCRx value. Software may also set or clear any CCIFG flag. All CCIFG flags request an interrupt when their corresponding CCIE bit and the GIE bit are set.

TACCR0 Interrupt

The TACCR0 CCIFG flag has the highest Timer_A interrupt priority and has a dedicated interrupt vector as shown in Figure 11–15. The TACCR0 CCIFG flag is automatically reset when the TACCR0 interrupt request is serviced.

Figure 11–15. Capture/Compare TACCR0 Interrupt Flag



TAIV, Interrupt Vector Generator

The TACCR1 CCIFG, TACCR2 CCIFG, and TAIFG flags are prioritized and combined to source a single interrupt vector. The interrupt vector register TAIV is used to determine which flag requested an interrupt.

The highest priority enabled interrupt generates a number in the TAIV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled Timer_A interrupts do not affect the TAIV value.

Any access, read or write, of the TAIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. For example, if the TACCR1 and TACCR2 CCIFG flags are set when the interrupt service routine accesses the TAIV register, TACCR1 CCIFG is reset automatically. After the `RETI` instruction of the interrupt service routine is executed, the TACCR2 CCIFG flag will generate another interrupt.

TAIV Software Example

The following software example shows the recommended use of TAIV and the handling overhead. The TAIV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself. The latencies are:

- Capture/compare block TACCR0 — 11 cycles
- Capture/compare blocks TACCR1, TACCR2 — 16 cycles
- Timer overflow TAIFG — 14 cycles

```
; Interrupt handler for TACCR0 CCIFG.                Cycles
CCIFG_0_HND
;         ...       ; Start of handler Interrupt latency  6
          RETI                                             5

; Interrupt handler for TAIFG, TACCR1 and TACCR2 CCIFG.

TA_HND    ...                ; Interrupt latency          6
          ADD    &TAIV,PC    ; Add offset to Jump table   3
          RETI               ; Vector 0: No interrupt     5
          JMP    CCIFG_1_HND ; Vector 2: TACCR1           2
          JMP    CCIFG_2_HND ; Vector 4: TACCR2           2
          RETI               ; Vector 6: Reserved         5
          RETI               ; Vector 8: Reserved         5

TAIFG_HND                    ; Vector 10: TAIFG Flag
          ...                ; Task starts here
          RETI                                           5

CCIFG_2_HND                  ; Vector 4: TACCR2
          ...                ; Task starts here
          RETI               ; Back to main program       5

CCIFG_1_HND                  ; Vector 2: TACCR1
          ...                ; Task starts here
          RETI               ; Back to main program       5
```

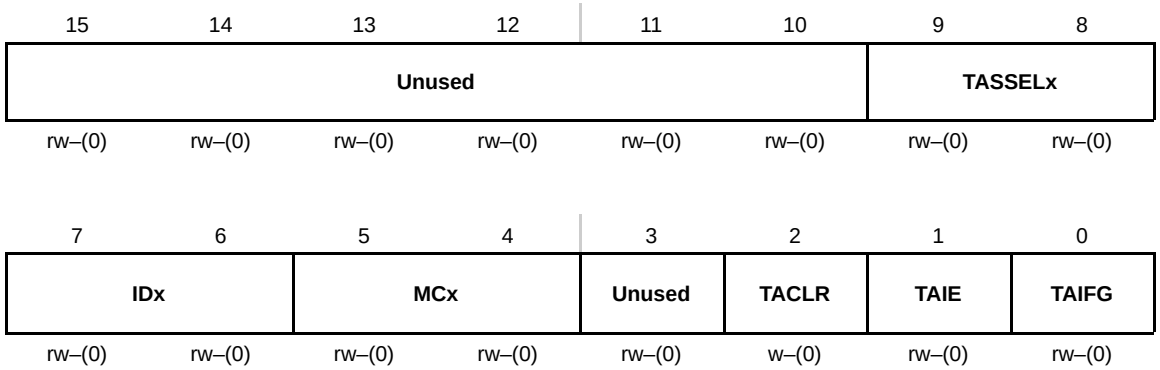
11.3 Timer_A Registers

The Timer_A registers are listed in Table 11–3:

Table 11–3. Timer_A Registers

Register	Short Form	Register Type	Address	Initial State
Timer_A control	TACTL	Read/write	0160h	Reset with POR
Timer_A counter	TAR	Read/write	0170h	Reset with POR
Timer_A capture/compare control 0	TACCTL0	Read/write	0162h	Reset with POR
Timer_A capture/compare 0	TACCR0	Read/write	0172h	Reset with POR
Timer_A capture/compare control 1	TACCTL1	Read/write	0164h	Reset with POR
Timer_A capture/compare 1	TACCR1	Read/write	0174h	Reset with POR
Timer_A capture/compare control 2	TACCTL2	Read/write	0166h	Reset with POR
Timer_A capture/compare 2	TACCR2	Read/write	0176h	Reset with POR
Timer_A Interrupt Vector	TAIV	Read only	012Eh	Reset with POR

TACTL, Timer_A Control Register



Unused Bits 15-10 Unused

TASSELx Bits 9-8 Timer_A clock source select
- 00 TACLK
- 01 ACLK
- 10 SMCLK
- 11 INCLK

IDx Bits 7-6 Input divider. These bits select the divider for the input clock.
- 00 /1
- 01 /2
- 10 /4
- 11 /8

MCx Bits 5-4 Mode control. Setting MCx = 00h when Timer_A is not in use conserves power.
- 00 Stop mode: the timer is halted
- 01 Up mode: the timer counts up to TACCR0
- 10 Continuous mode: the timer counts up to 0FFFFh
- 11 Up/down mode: the timer counts up to TACCR0 then down to 0000h

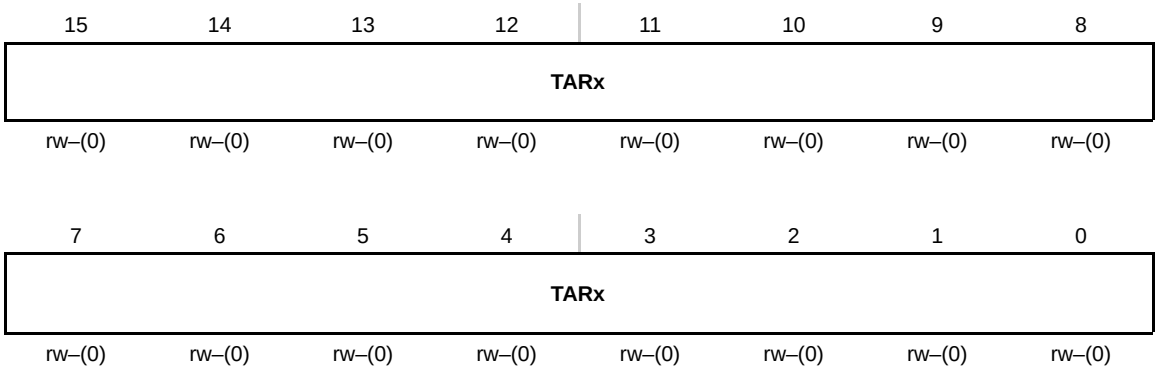
Unused Bit 3 Unused

TACLR Bit 2 Timer_A clear. Setting this bit resets TAR, IDx, and count direction. The TACLR bit is automatically reset and is always read as zero.

TAIE Bit 1 Timer_A interrupt enable. This bit enables the TAIFG interrupt request.
- 0 Interrupt disabled
- 1 Interrupt enabled

TAIFG Bit 0 Timer_A interrupt flag
- 0 No interrupt pending
- 1 Interrupt pending

TAR, Timer_A Register



TARx	Bits 15-0	Timer_A register. The TAR register is the count of Timer_A.

TACCTLx, Capture/Compare Control Register

15	14	13	12	11	10	9	8
CMx		CCISx		SCS	SCCI	Unused	CAP
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	r–(0)	r–(0)	rw–(0)

7	6	5	4	3	2	1	0
OUTMODx			CCIE	CCI	OUT	COV	CCIFG
rw–(0)	rw–(0)	rw–(0)	rw–(0)	r	rw–(0)	rw–(0)	rw–(0)

CMx Bit 15-14 Capture mode
- 00 No capture
- 01 Capture on rising edge
- 10 Capture on falling edge
- 11 Capture on both rising and falling edges

CCISx Bit 13-12 Capture/compare input select. These bits select the TACCRx input signal. See the device-specific datasheet for specific signal connections.
- 00 CCIxA
- 01 CCIxB
- 10 GND
- 11 V_{CC}

SCS Bit 11 Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.
- 0 Asynchronous capture
- 1 Synchronous capture

SCCI Bit 10 Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit

Unused Bit 9 Unused. Read only. Always read as 0.

CAP Bit 8 Capture mode
- 0 Compare mode
- 1 Capture mode

OUTMODx Bits 7-5 Output mode. Modes 2, 3, 6, and 7 are not useful for TACCR0 because EQUx = EQU0.
- 000 OUT bit value
- 001 Set
- 010 Toggle/reset
- 011 Set/reset
- 100 Toggle
- 101 Reset
- 110 Toggle/set
- 111 Reset/set

CCIE Bit 4 Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag.
0 Interrupt disabled
1 Interrupt enabled

CCI Bit 3 Capture/compare input. The selected input signal can be read by this bit.

OUT Bit 2 Output. This bit indicates the state of the output. For output mode 0, this bit directly controls the state of the output.
0 Output low
1 Output high

COV Bit 1 Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software.
0 No capture overflow occurred
1 Capture overflow occurred

CCIFG Bit 0 Capture/compare interrupt flag
0 No interrupt pending
1 Interrupt pending

TAIV, Timer_A Interrupt Vector Register

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**TAIVx**			**0**
r0	r0	r0	r0	r–(0)	r–(0)	r–(0)	r0

TAIVx Bits 15-0 Timer_A Interrupt Vector value

TAIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	–	
02h	Capture/compare 1	TACCR1 CCIFG	Highest
04h	Capture/compare 2	TACCR2 CCIFG	
06h	Reserved	–	
08h	Reserved	–	
0Ah	Timer overflow	TAIFG	
0Ch	Reserved	–	
0Eh	Reserved	–	Lowest

Chapter 12

Timer_B

Timer_B is a 16-bit timer/counter with multiple capture/compare registers. This chapter describes Timer_B. Timer_B3 (three capture/compare registers) is implemented in MSP430x13x, and MSP430x15x devices. Timer_B7 (seven capture/compare registers) is implemented in MSP430x14x and MSP430x16x devices.

Topic **Page**

12.1 Timer_B Introduction

Timer_B is a 16-bit timer/counter with three or seven capture/compare registers. Timer_B can support multiple capture/compares, PWM outputs, and interval timing. Timer_B also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers.

Timer_B features include :

- ❑ Asynchronous 16-bit timer/counter with four operating modes and four selectable lengths
- ❑ Selectable and configurable clock source
- ❑ Three or seven configurable capture/compare registers
- ❑ Configurable outputs with PWM capability
- ❑ Double-buffered compare latches with synchronized loading
- ❑ Interrupt vector register for fast decoding of all Timer_B interrupts

The block diagram of Timer_B is shown in Figure 12–1.

Note: Use of the Word *Count*

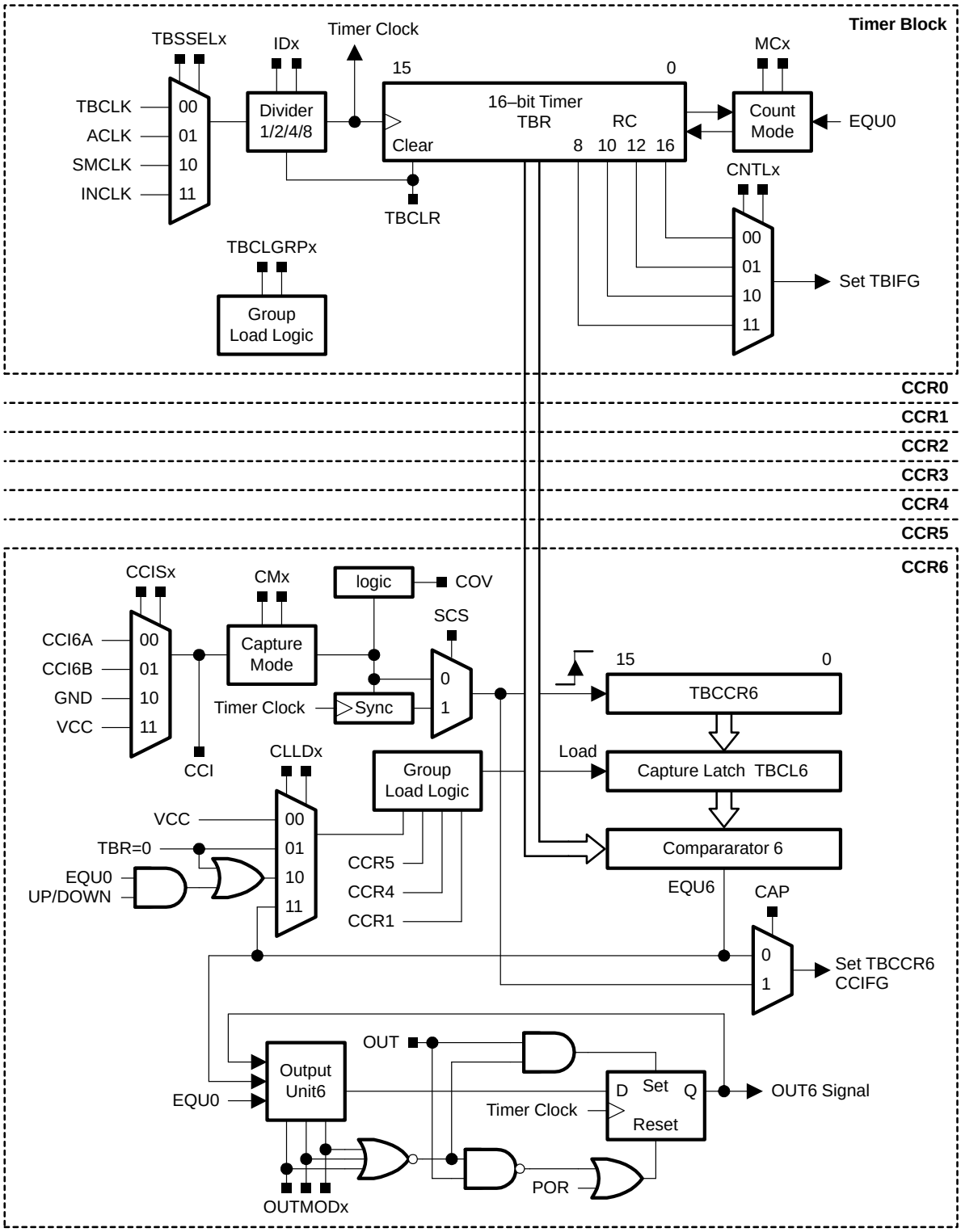
C*ount* is used throughout this chapter. It means the counter must be in the process of counting for the action to take place. If a particular value is directly written to the counter, then an associated action does not take place.

12.1.1 Similarities and Differences From Timer_A

Timer_B is identical to Timer_A with the following exceptions:

- ❑ The length of Timer_B is programmable to be 8, 10, 12, or 16 bits.
- ❑ Timer_B TBCCRx registers are double-buffered and can be grouped.
- ❑ All Timer_B outputs can be put into a high-impedance state.
- ❑ The SCCI bit function is not implemented in Timer_B.

Figure 12–1. Timer_B Block Diagram



12.2 Timer_B Operation

The Timer_B module is configured with user software. The setup and operation of Timer_B is discussed in the following sections.

12.2.1 16-Bit Timer Counter

The 16-bit timer/counter register, TBR, increments or decrements (depending on mode of operation) with each rising edge of the clock signal. TBR can be read or written with software. Additionally, the timer can generate an interrupt when it overflows.

TBR may be cleared by setting the TBCLR bit. Setting TBCLR also clears the clock divider and count direction for up/down mode.

Note: Modifying Timer_B Registers

It is recommended to stop the timer before modifying its operation (with exception of the interrupt enable and interrupt flag) to avoid errant operating conditions.

When TBCLK is asynchronous to the CPU clock, any read from TBR should occur while the timer is not operating or the results may be unpredictable. Any write to TBR will take effect immediately.

TBR Length

Timer_B is configurable to operate as an 8-, 10-, 12-, or 16-bit timer with the CNTLx bits. The maximum count value, $TBR_{(max)}$, for the selectable lengths is 0FFh, 03FFh, 0FFFh, and 0FFFFh, respectively. Data written to the TBR register in 8-, 10-, and 12-bit mode is right-justified with leading zeros.

Clock Source Select and Divider

The timer clock TBCLK can be sourced from ACLK, SMCLK, or externally via TBCLK or INCLK. The clock source is selected with the TBSSELx bits. The selected clock source may be passed directly to the timer or divided by 2,4, or 8, using the IDx bits.

12.2.2 Starting the Timer

The timer may be started or restarted in the following ways:

- The timer counts when MCx > 0 and the clock source is active.
- When the timer mode is either up or up/down, the timer may be stopped by loading 0 to TBCL0. 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.

12.2.3 Timer Mode Control

The timer has four modes of operation as described in Table 12–1: stop, up, continuous, and up/down. The operating mode is selected with the MCx bits.

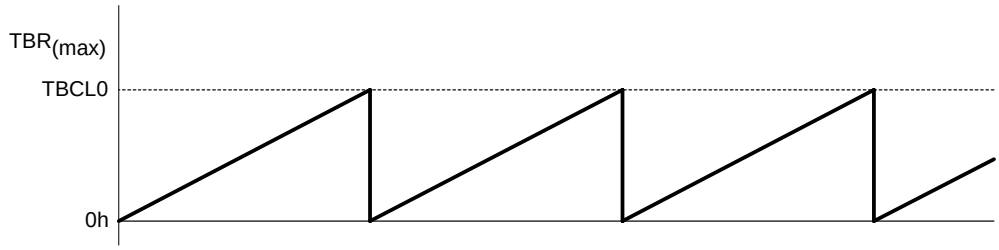
Table 12–1. Timer Modes

MCx	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of compare register TBCL0.
10	Continuous	The timer repeatedly counts from zero to the value selected by the TBCNTLx bits.
11	Up/down	The timer repeatedly counts from zero up to the value of TBCL0 and then back down to zero.

Up Mode

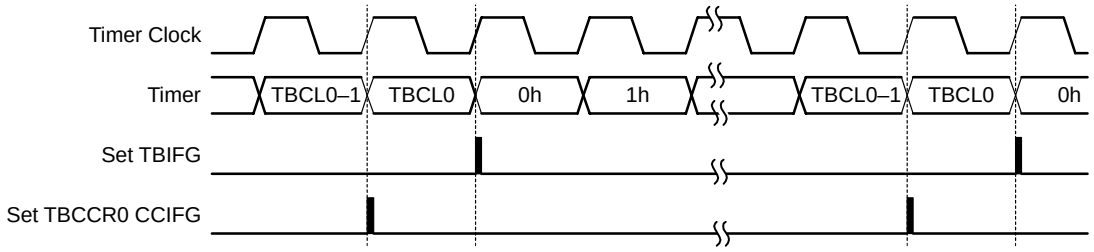
The up mode is used if the timer period must be different from $TBR_{(max)}$ counts. The timer repeatedly counts up to the value of compare latch TBCL0, which defines the period, as shown in Figure 12–2. The number of timer counts in the period is TBCL0+1. When the timer value equals TBCL0 the timer restarts counting from zero. If up mode is selected when the timer value is greater than TBCL0, the timer immediately restarts counting from zero.

Figure 12–2. Up Mode



The TBCCR0 CCIFG interrupt flag is set when the timer equals the TBCL0 value. The TBIFG interrupt flag is set when the timer *counts* from TBCL0 to zero. Figure 11–3 shows the flag set cycle.

Figure 12–3. Up Mode Flag Setting



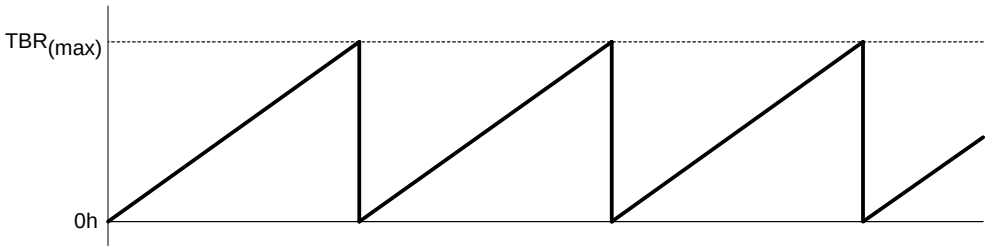
Changing the Period Register TBCL0

When changing TBCL0 while the timer is running and when the TBCL0 load mode is *immediate*, if the new period is greater than or equal to the old period, or greater than the current count value, the timer counts up to the new period. If the new period is less than the current count value, the timer rolls to zero. However, one additional count may occur before the counter rolls to zero.

Continuous Mode

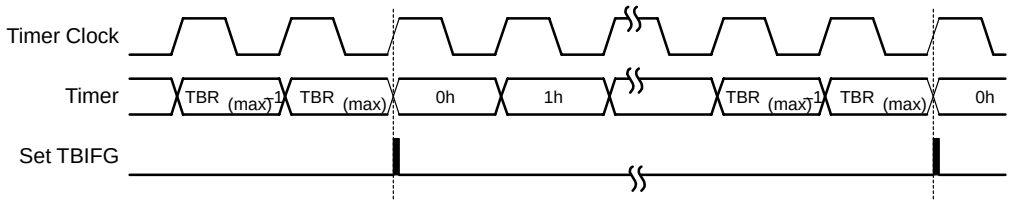
In continuous mode the timer repeatedly counts up to $TBR_{(max)}$ and restarts from zero as shown in Figure 12–4. The compare latch TBCL0 works the same way as the other capture/compare registers.

Figure 12–4. Continuous Mode



The TBIFG interrupt flag is set when the timer *counts* from $TBR_{(max)}$ to zero. Figure 12–5 shows the flag set cycle.

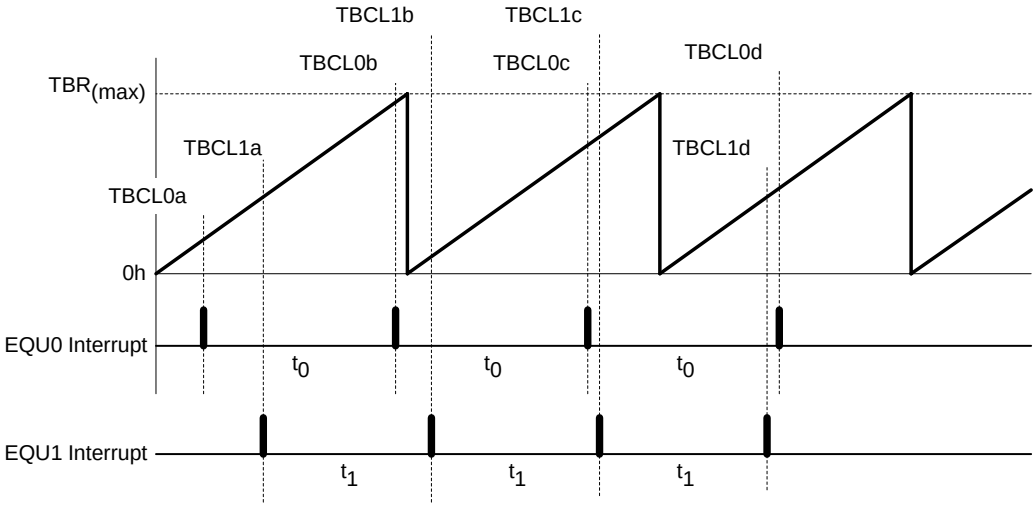
Figure 12–5. Continuous Mode Flag Setting



Use of the Continuous Mode

The continuous mode can be used to generate independent time intervals and output frequencies. Each time an interval is completed, an interrupt is generated. The next time interval is added to the TBCLx latch in the interrupt service routine. Figure 12–6 shows two separate time intervals t_0 and t_1 being added to the capture/compare registers. The time interval is controlled by hardware, not software, without impact from interrupt latency. Up to three (Timer_B3) or 7 (Timer_B7) independent time intervals or output frequencies can be generated using capture/compare registers.

Figure 12–6. Continuous Mode Time Intervals



Time intervals can be produced with other modes as well, where TBCL0 is used as the period register. Their handling is more complex since the sum of the old TBCLx data and the new period can be higher than the TBCL0 value. When the sum of the previous TBCLx value plus t_x is greater than the TBCL0 data, the old TBCL0 value must be subtracted to obtain the correct time interval.

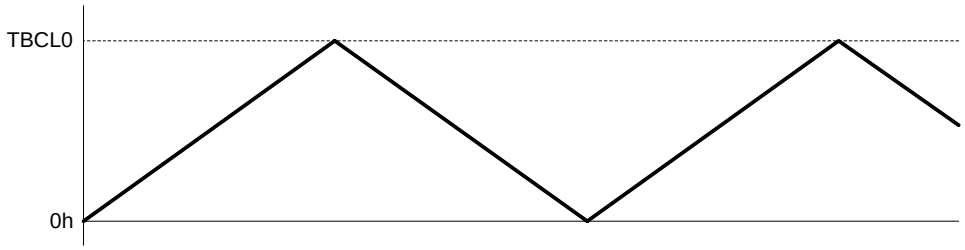
Up/Down Mode

The up/down mode is used if the timer period must be different from $TBR_{(max)}$ counts, and if symmetrical pulse generation is needed. The timer repeatedly counts up to the value of compare latch TBCL0, and back down to zero, as shown in Figure 12–7. The period is twice the value in TBCL0.

Note: TBCL0 > TBR(max)

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

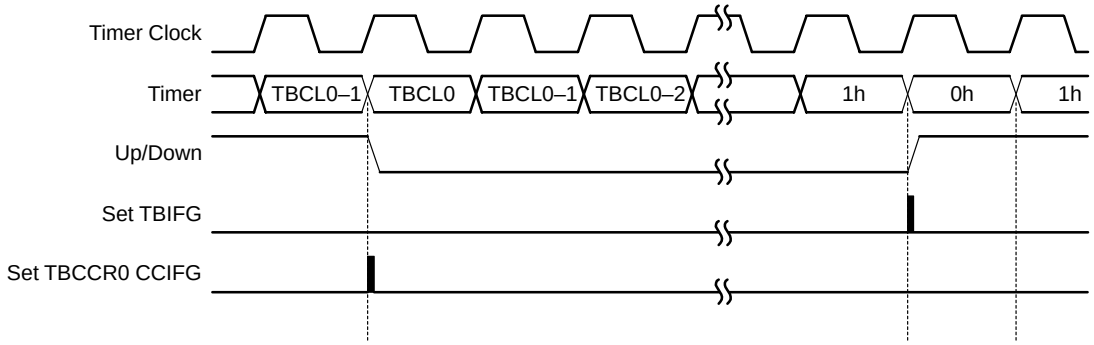
Figure 12–7. Up/Down Mode



The count direction is latched. This allows the timer to be stopped and then restarted in the same direction it was counting before it was stopped. If this is not desired, the TBCLR bit must be used to clear the direction. The TBCLR bit also clears the TBR value and the TBCLK divider.

In up/down mode, the TBCCR0 CCIFG interrupt flag and the TBIFG interrupt flag are set only once during the period, separated by 1/2 the timer period. The TBCCR0 CCIFG interrupt flag 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. Figure 12–8 shows the flag set cycle.

Figure 12–8. Up/Down Mode Flag Setting



Changing the Value of Period Register TBCL0

When changing TBCL0 while the timer is running, and counting in the down direction, and when the TBCL0 load mode is *immediate*, the timer continues its descent until it reaches zero. The new period takes effect after the counter counts down to zero.

If the timer is counting in the up direction when the new period is latched into TBCL0, and the new period is greater than or equal to the old period, or greater than the current count value, the timer counts up to the new period before counting down. When the timer is counting in the up direction, and the new period is less than the current count value when TBCL0 is loaded, the timer begins counting down. However, one additional count may occur before the counter begins counting down.

Use of the Up/Down Mode

The up/down mode supports applications that require dead times between output signals (see section *Timer_B Output Unit*). For example, to avoid overload conditions, two outputs driving an H-bridge must never be in a high state simultaneously. In the example shown in Figure 12–9 the t_{dead} is:

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

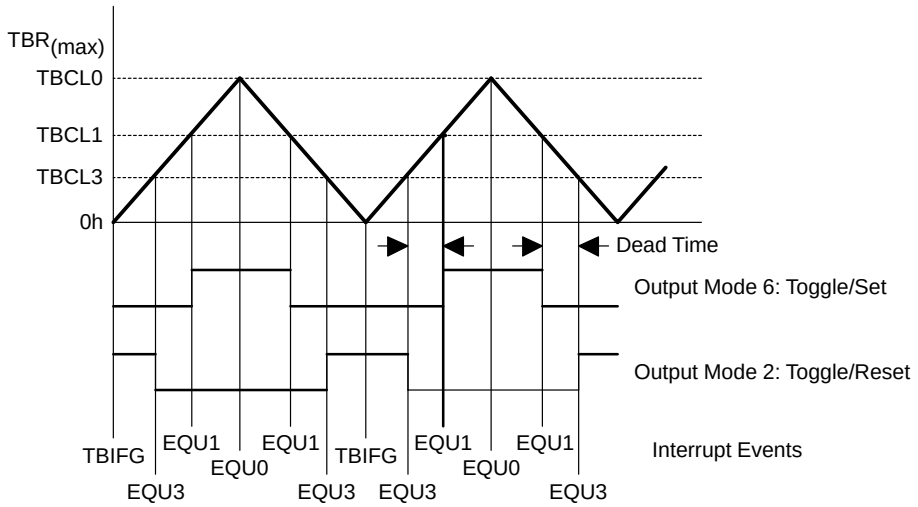
With: t_{dead} Time during which both outputs need to be inactive

t_{timer} Cycle time of the timer clock

TBCLx Content of compare latch x

The ability to simultaneously load grouped compare latches assures the dead times.

Figure 12–9. Output Unit in Up/Down Mode



12.2.4 Capture/Compare Blocks

Three or seven identical capture/compare blocks, TBCCRx, are present in Timer_B. Any of the blocks may be used to capture the timer data or to generate time intervals.

Capture Mode

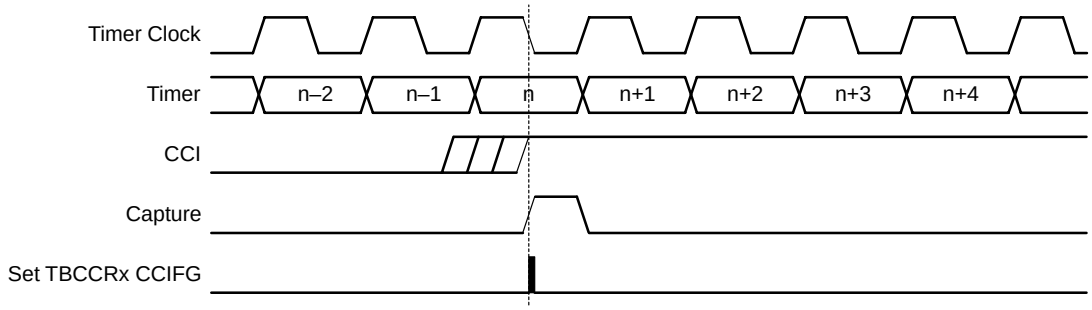
The capture mode is selected when CAP = 1. Capture mode is used to record time events. It can be used for speed computations or time measurements. The capture inputs CCIxA and CCIxB are connected to external pins or internal signals and are selected with the CCISx bits. The CMx bits select the capture edge of the input signal as rising, falling, or both. A capture occurs on the selected edge of the input signal. If a capture is performed:

- The timer value is copied into the TBCCRx register
- The interrupt flag CCIFG is set

The input signal level can be read at any time via the CCI bit. MSP430x1xx family devices may have different signals connected to CCIxA and CCIxB. Refer to the device-specific datasheet for the connections of these signals.

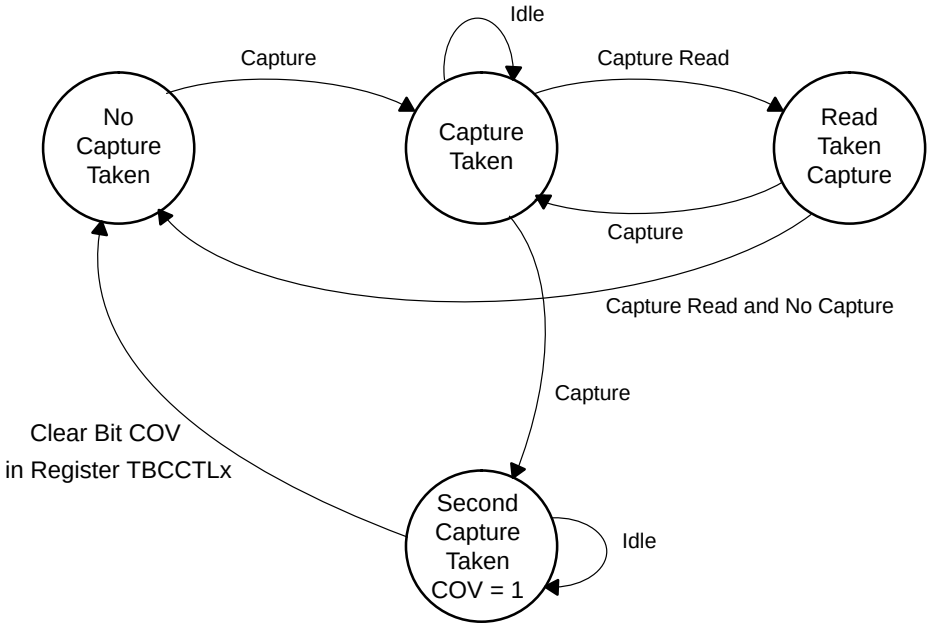
The capture signal can be asynchronous to the timer clock and cause a race condition. Setting the SCS bit will synchronize the capture with the next timer clock. Setting the SCS bit to synchronize the capture signal with the timer clock is recommended. This is illustrated in Figure 12–10.

Figure 12–10. Capture Signal (SCS=1)



Overflow logic is provided in each capture/compare register to indicate if a second capture was performed before the value from the first capture was read. Bit COV is set when this occurs as shown in Figure 12–11. COV must be reset with software.

Figure 12–11. Capture Cycle



Capture Initiated by Software

Captures can be initiated by software. The CMx bits can be set for capture on both edges. Software then sets bit CCIS1=1 and toggles bit CCIS0 to switch the capture signal between V_{CC} and GND, initiating a capture each time CCIS0 changes state:

```
MOV    #CAP+SCS+CCIS1+CM_3,&TBCCTLx ; Setup TBCCTLx
XOR    #CCIS0,&TBCCTLx              ; TBCCTLx = TBR
```

Compare Mode

The compare mode is selected when CAP = 0. Compare mode is used to generate PWM output signals or interrupts at specific time intervals. When TBR *counts* to the value in a TBCLx:

- ❏ Interrupt flag CCIFG is set
- ❏ Internal signal EQUx = 1
- ❏ EQUx affects the output according to the output mode

Compare Latch TBCLx

The TBCCRx compare latch, TBCLx, holds the data for the comparison to the timer value in compare mode. TBCLx is buffered by TBCCRx. The buffered compare latch gives the user control over when a compare period updates. The user cannot directly access TBCLx. Compare data is written to each TBCCRx and automatically transferred to TBCLx. The timing of the transfer from TBCCRx to TBCLx is user-selectable with the CLLDx bits as described in Table 12–2.

Table 12–2. TBCLx Load Events

CLLDx	Description
00	New data is transferred from TBCCRx to TBCLx immediately when TBCCRx is written to.
01	New data is transferred from TBCCRx to TBCLx when TBR *counts* to 0
10	New data is transferred from TBCCRx to TBCLx when TBR *counts* to 0 for up and continuous modes. New data is transferred to from TBCCRx to TBCLx when TBR *counts* to the old TBCL0 value or to 0 for up/down mode
11	New data is transferred from TBCCRx to TBCLx when TBR *counts* to the old TBCLx value.

Grouping Compare Latches

Multiple compare latches may be grouped together for simultaneous updates with the TBCLGRPx bits. When using groups, the CLLDx bits of the lowest numbered TBCCRx in the group determine the load event for each compare latch of the group, except when TBCLGRP = 3, as shown in Table 12–3. The CLLDx bits of the controlling TBCCRx must not be set to zero. When the CLLDx bits of the controlling TBCCRx are set to zero, all compare latches update immediately when their corresponding TBCCRx is written - no compare latches are grouped.

Two conditions must exist for the compare latches to be loaded when grouped. First, all TBCCRx registers of the group must be updated, even when new TBCCRx data = old TBCCRx data. Second, the load event must occur.

Table 12–3. Compare Latch Operating Modes

TBCLGRPx	Grouping	Update Control
00	None	Individual
01	TBCL1+TBCL2 TBCL3+TBCL4 TBCL5+TBCL6	TBCCR1 TBCCR3 TBCCR5
10	TBCL1+TBCL2+TBCL3 TBCL4+TBCL5+TBCL6	TBCCR1 TBCCR4
11	TBCL0+TBCL1+TBCL2+ TBCL3+TBCL4+TBCL5+TBCL6	TBCCR1

12.2.5 Output Unit

Each capture/compare block contains an output unit. The output unit is used to generate output signals such as PWM signals. Each output unit has eight operating modes that generate signals based on the EQU0 and EQUx signals. The TBoutH pin function can be used to put all Timer_B outputs into a high-impedance state. When the TBoutH pin function is selected for the pin, and when the pin is pulled high, all Timer_B outputs are in a high-impedance state.

Output Modes

The output modes are defined by the OUTMODx bits and are described in Table 12–4. 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 because EQUx = EQU0.

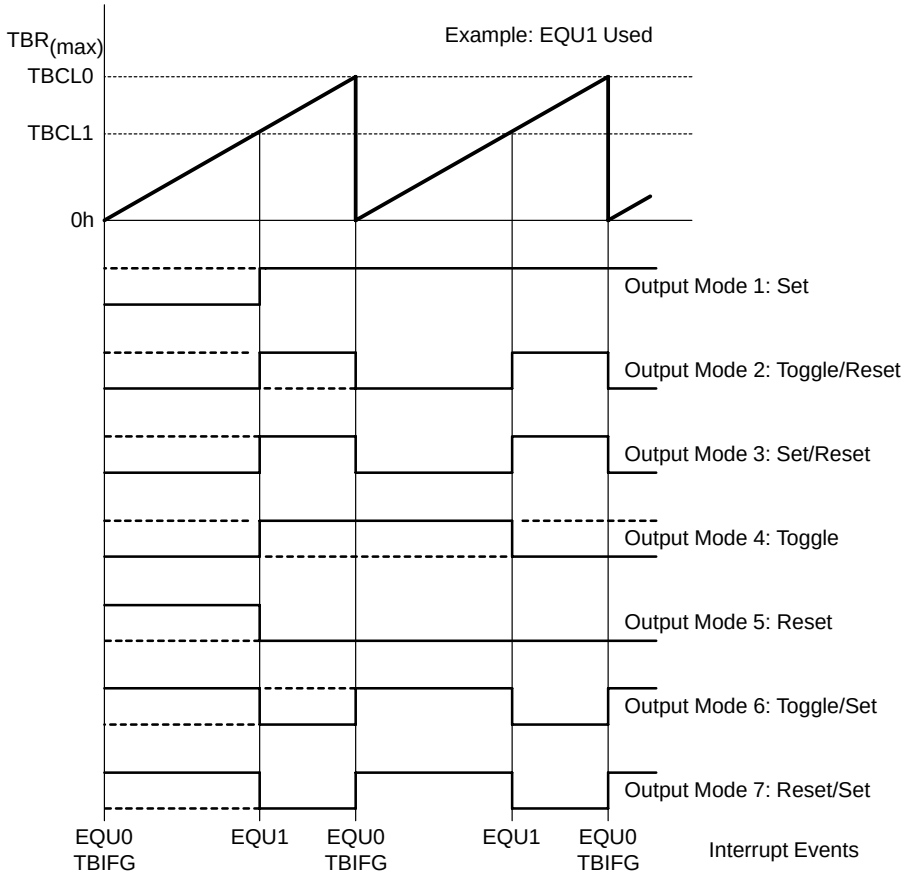
Table 12–4. Output Modes

OUTMODx	Mode	Description
000	Output	The output signal OUTx is defined by the OUTx bit. The OUTx signal updates immediately when OUTx is updated.
001	Set	The output is set when the timer *counts* to the TBCLx value. It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	The output is toggled when the timer *counts* to the TBCLx value. It is reset when the timer *counts* to the TBCL0 value.
011	Set/Reset	The output is set when the timer *counts* to the TBCLx value. It is reset when the timer *counts* to the TBCL0 value.
100	Toggle	The output is toggled when the timer *counts* to the TBCLx value. The output period is double the timer period.
101	Reset	The output is reset when the timer *counts* to the TBCLx value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer *counts* to the TBCLx value. It is set when the timer *counts* to the TBCL0 value.
111	Reset/Set	The output is reset when the timer *counts* to the TBCLx value. It is set when the timer *counts* to the TBCL0 value.

Output Example—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. An example is shown in Figure 12–12 using TBCL0 and TBCL1.

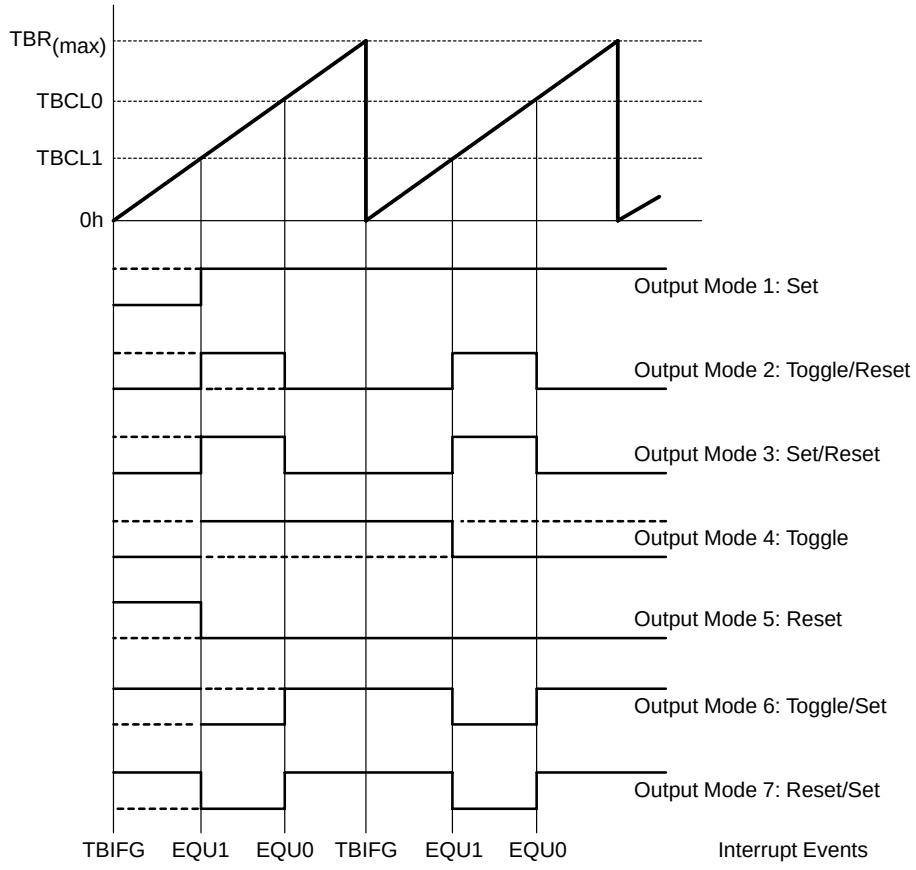
Figure 12–12. Output Example—Timer in Up Mode



Output Example—Timer in Continuous Mode

The OUTx signal is changed when the timer reaches the TBCLx and TBCL0 values, depending on the output mode, An example is shown in Figure 12–13 using TBCL0 and TBCL1.

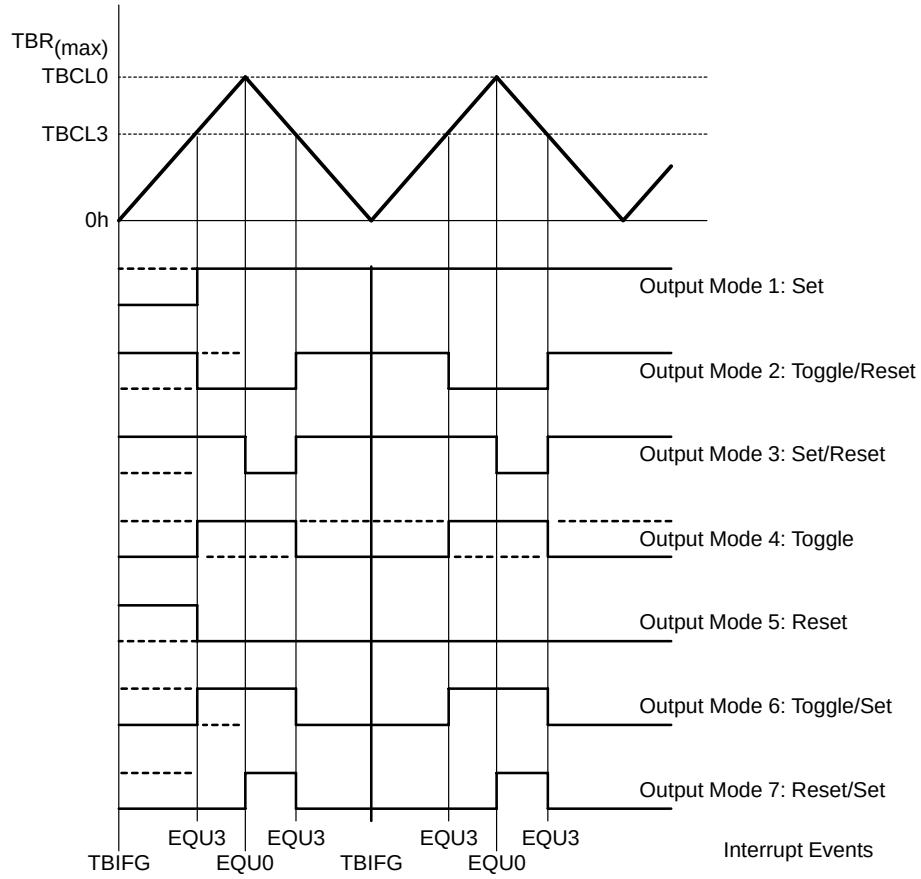
Figure 12–13. Output Example—Timer in Continuous Mode



Output Example – 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. An example is shown in Figure 12–14 using TBCL0 and TBCL3.

Figure 12–14. Output Example—Timer in Up/Down Mode



Note: *Switching Between Output Modes*

When switching between output modes, one of the OUTMODx bits should remain set during the transition, unless switching to mode 0. Otherwise, output glitching can occur because a NOR gate decodes output mode 0. A safe method for switching between output modes is to use output mode 7 as a transition state:

```
BIS   #OUTMOD_7,&TBCCTLx ; Set output mode=7
BIC   #OUTMODx,&TBCCTLx  ; Clear unwanted bits
```

12.2.6 Timer_B Interrupts

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

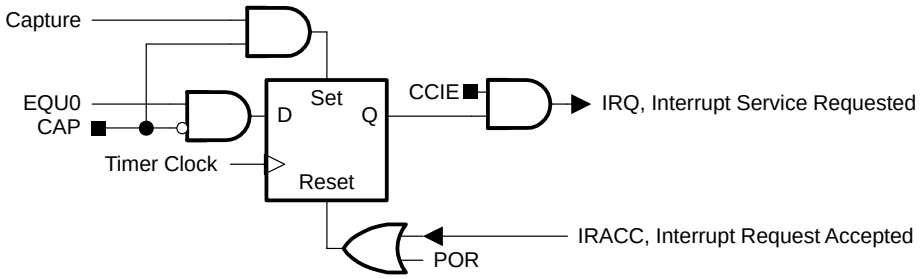
- TBCCR0 interrupt vector for TBCCR0 CCIFG
- TBIV interrupt vector for all other CCIFG flags and TBIFG

In capture mode, any CCIFG flag is set when a timer value is captured in the associated TBCCRx register. In compare mode, any CCIFG flag is set when TBR counts to the associated TBCLx value. Software may also set or clear any CCIFG flag. All CCIFG flags request an interrupt when their corresponding CCIE bit and the GIE bit are set.

TBCCR0 Interrupt Vector

The TBCCR0 CCIFG flag has the highest Timer_B interrupt priority and has a dedicated interrupt vector as shown in Figure 12–15. The TBCCR0 CCIFG flag is automatically reset when the TBCCR0 interrupt request is serviced.

Figure 12–15. Capture/Compare TBCCR0 Interrupt Flag



TBIV, Interrupt Vector Generator

The TBIFG flag and TBCCRx CCIFG flags (excluding TBCCR0 CCIFG) are prioritized and combined to source a single interrupt vector. The interrupt vector register TBIV is used to determine which flag requested an interrupt.

The highest priority enabled interrupt (excluding TBCCR0 CCIFG) generates a number in the TBIV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled Timer_B interrupts do not affect the TBIV value.

Any access, read or write, of the TBIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. For example, if the TBCCR1 and TBCCR2 CCIFG flags are set when the interrupt service routine accesses the TBIV register, TBCCR1 CCIFG is reset automatically. After the `RETI` instruction of the interrupt service routine is executed, the TBCCR2 CCIFG flag will generate another interrupt.

TBIV, Interrupt Handler Examples

The following software example shows the recommended use of TBIV and the handling overhead. The TBIV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU clock cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself. The latencies are:

- ❏ Capture/compare block CCR0 — 11 cycles
- ❏ Capture/compare blocks CCR1 to CCR6 — 16 cycles
- ❏ Timer overflow TBIFG — 14 cycles

The following software example shows the recommended use of TBIV for Timer_B3.

```
; Interrupt handler for TBCCR0 CCIFG.                Cycles
CCIFG_0_HND
          ...       ; Start of handler Interrupt latency  6
          RETI                                            5


; Interrupt handler for TBIFG, TBCCR1 and TBCCR2 CCIFG.
TB_HND    $                 ; Interrupt latency           6
          ADD   &TBIV,PC    ; Add offset to Jump table    3
          RETI              ; Vector  0: No interrupt     5
          JMP   CCIFG_1_HND ; Vector  2: Module 1         2
          JMP   CCIFG_2_HND ; Vector  4: Module 2         2
          RETI              ; Vector  6
          RETI              ; Vector  8
          RETI              ; Vector 10
          RETI              ; Vector 12

TBIFG_HND                   ; Vector 14: TIMOV Flag
          ...               ; Task starts here
          RETI                                            5

CCIFG_2_HND                 ; Vector 4: Module 2
          ...               ; 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
CCIFG_1_HND                 ; Vector 6: Module 3
          ...               ; Task starts here
          JMP   TB_HND      ; Look for pending ints       2
```

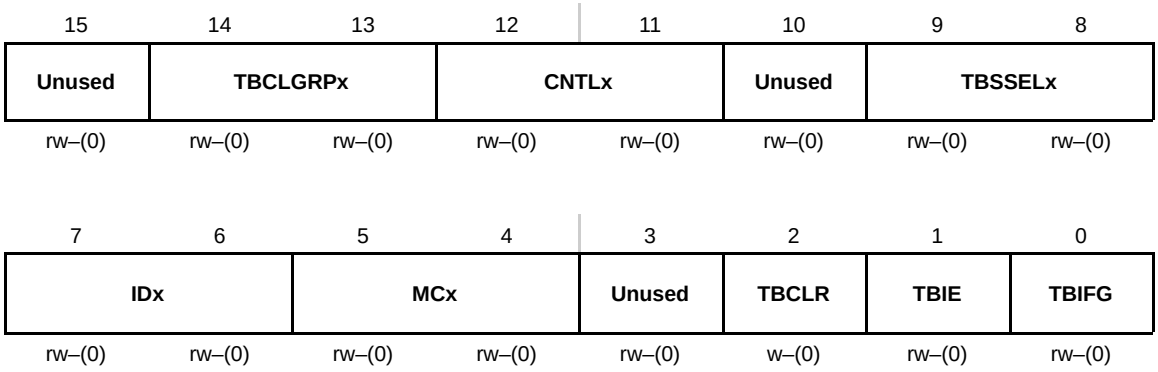
12.3 Timer_B Registers

The Timer_B registers are listed in Table 12–5:

Table 12–5. Timer_B Registers

Register	Short Form	Register Type	Address	Initial State
Timer_B control	TBCTL	Read/write	0180h	Reset with POR
Timer_B counter	TBR	Read/write	0190h	Reset with POR
Timer_B capture/compare control 0	TBCCTL0	Read/write	0182h	Reset with POR
Timer_B capture/compare 0	TBCCR0	Read/write	0192h	Reset with POR
Timer_B capture/compare control 1	TBCCTL1	Read/write	0184h	Reset with POR
Timer_B capture/compare 1	TBCCR1	Read/write	0194h	Reset with POR
Timer_B capture/compare control 2	TBCCTL 2	Read/write	0186h	Reset with POR
Timer_B capture/compare 2	TBCCR2	Read/write	0196h	Reset with POR
Timer_B capture/compare control 3	TBCCTL3	Read/write	0188h	Reset with POR
Timer_B capture/compare 3	TBCCR3	Read/write	0198h	Reset with POR
Timer_B capture/compare control 4	TBCCTL4	Read/write	018Ah	Reset with POR
Timer_B capture/compare 4	TBCCR4	Read/write	019Ah	Reset with POR
Timer_B capture/compare control 5	TBCCTL5	Read/write	018Ch	Reset with POR
Timer_B capture/compare 5	TBCCR5	Read/write	019Ch	Reset with POR
Timer_B capture/compare control 6	TBCCTL6	Read/write	018Eh	Reset with POR
Timer_B capture/compare 6	TBCCR6	Read/write	019Eh	Reset with POR
Timer_B Interrupt Vector	TBIV	Read only	011Eh	Reset with POR

Timer_B Control Register TBCTL



Unused Bit 15 Unused

TBCLGRP Bit 14-13 TBCLx group
- 00 Each TBCLx latch loads independently
- 01 TBCL1+TBCL2 (TBCCR1 CLLDx bits control the update)
  TBCL3+TBCL4 (TBCCR3 CLLDx bits control the update)
  TBCL5+TBCL6 (TBCCR5 CLLDx bits control the update)
  TBCL0 independent
- 10 TBCL1+TBCL2+TBCL3 (TBCCR1 CLLDx bits control the update)
  TBCL4+TBCL5+TBCL6 (TBCCR4 CLLDx bits control the update)
  TBCL0 independent
- 11 TBCL0+TBCL1+TBCL2+TBCL3+TBCL4+TBCL5+TBCL6
  (TBCCR1 CLLDx bits control the update)

CNTLx Bits 12-11 Counter Length
- 00 16-bit, $TBR_{(max)}$ = 0FFFFh
- 01 12-bit, $TBR_{(max)}$ = 0FFFh
- 10 10-bit, $TBR_{(max)}$ = 03FFh
- 11 8-bit, $TBR_{(max)}$ = 0FFh

Unused Bit 10 Unused

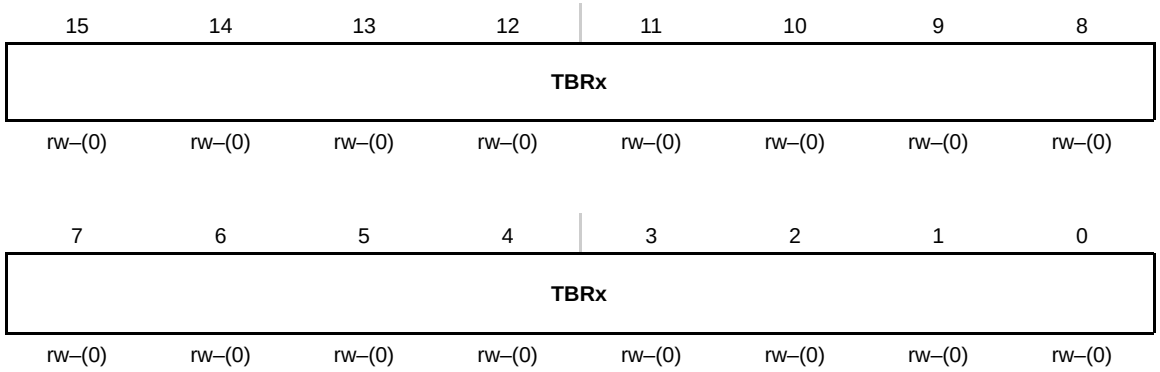
TBSSELx Bits 9-8 Timer_B clock source select.
- 00 TBCLK
- 01 ACLK
- 10 SMCLK
- 11 INCLK

IDx Bits 7-6 Input divider. These bits select the divider for the input clock.
- 00 /1
- 01 /2
- 10 /4
- 11 /8

MCx Bits 5-4 Mode control. Setting MCx = 00h when Timer_B is not in use conserves power.
- 00 Stop mode: the timer is halted
- 01 Up mode: the timer counts up to TBCL0
- 10 Continuous mode: the timer counts up to the value set by TBCNTLx
- 11 Up/down mode: the timer counts up to TBCL0 and down to 0000h

Unused	Bit 3	Unused
TBCLR	Bit 2	Timer_B clear. Setting this bit resets TBR, IDx, and count direction. The TBCLR bit is automatically reset and is always read as zero.
TBIE	Bit 1	Timer_B interrupt enable. This bit enables the TBIFG interrupt request. 0 Interrupt disabled 1 Interrupt enabled
TBIFG	Bit 0	Timer_B interrupt flag. 0 No interrupt pending 1 Interrupt pending

TBR, Timer_B Register



TBRx	Bits 15-0	Timer_B register. The TBR register is the count of Timer_B.

TBCCTLx, Capture/Compare Control Register

15	14	13	12	11	10	9	8
CMx		**CCISx**		**SCS**	**CLLDx**		**CAP**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	r–(0)	rw–(0)

7	6	5	4	3	2	1	0
OUTMODx			**CCIE**	**CCI**	**OUT**	**COV**	**CCIFG**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	r	rw–(0)	rw–(0)	rw–(0)

CMx Bit 15-14 Capture mode
- 00 No capture
- 01 Capture on rising edge
- 10 Capture on falling edge
- 11 Capture on both rising and falling edges

CCISx Bit 13-12 Capture/compare input select. These bits select the TBCCRx input signal. See the device-specific datasheet for specific signal connections.
- 00 CCIxA
- 01 CCIxB
- 10 GND
- 11 V_{CC}

SCS Bit 11 Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.
- 0 Asynchronous capture
- 1 Synchronous capture

CLLDx Bit 10-9 Compare latch load. These bits select the compare latch load event.
- 00 TBCLx loads on write to TBCCRx
- 01 TBCLx loads when TBR *counts* to 0
- 10 TBCLx loads when TBR *counts* to 0 (up or continuous mode)
  TBCLx loads when TBR *counts* to TBCL0 or to 0 (up/down mode)
- 11 TBCLx loads when TBR *counts* to TBCLx

CAP Bit 8 Capture mode
- 0 Compare mode
- 1 Capture mode

OUTMODx Bits 7-5 Output mode. Modes 2, 3, 6, and 7 are not useful for TBCL0 because EQUx = EQU0.
- 000 OUT bit value
- 001 Set
- 010 Toggle/reset
- 011 Set/reset
- 100 Toggle
- 101 Reset
- 110 Toggle/set
- 111 Reset/set

CCIE	Bit 4	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0 Interrupt disabled 1 Interrupt enabled
CCI	Bit 3	Capture/compare input. The selected input signal can be read by this bit.
OUT	Bit 2	Output. This bit indicates the state of the output. For output mode 0, this bit directly controls the state of the output. 0 Output low 1 Output high
COV	Bit 1	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. 0 No capture overflow occurred 1 Capture overflow occurred
CCIFG	Bit 0	Capture/compare interrupt flag 0 No interrupt pending 1 Interrupt pending

TBIV, Timer_B Interrupt Vector Register

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**TBIVx**			**0**
r0	r0	r0	r0	r–(0)	r–(0)	r–(0)	r0

TBIVx Bits 15-0 Timer_B interrupt vector value

TBIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	–	
02h	Capture/compare 1	TBCCR1 CCIFG	Highest
04h	Capture/compare 2	TBCCR2 CCIFG	
06h	Capture/compare 3†	TBCCR3 CCIFG	
08h	Capture/compare 4†	TBCCR4 CCIFG	
0Ah	Capture/compare 5†	TBCCR5 CCIFG	
0Ch	Capture/compare 6†	TBCCR6 CCIFG	
0Eh	Timer overflow	TBIFG	Lowest

† MSP430x14x, MSP430x16x devices only

Chapter 13

USART Peripheral Interface, UART Mode

The universal synchronous/asynchronous receive/transmit (USART) peripheral interface supports two serial modes with one hardware module. This chapter discusses the operation of the asynchronous UART mode. USART0 is implemented on the MSP430x12xx, MSP430x13xx, and MSP430x15x devices. In addition to USART0, the MSP430x14x and MSP430x16x devices implement a second identical USART module, USART1.

Topic **Page**

13.1 USART Introduction: UART Mode

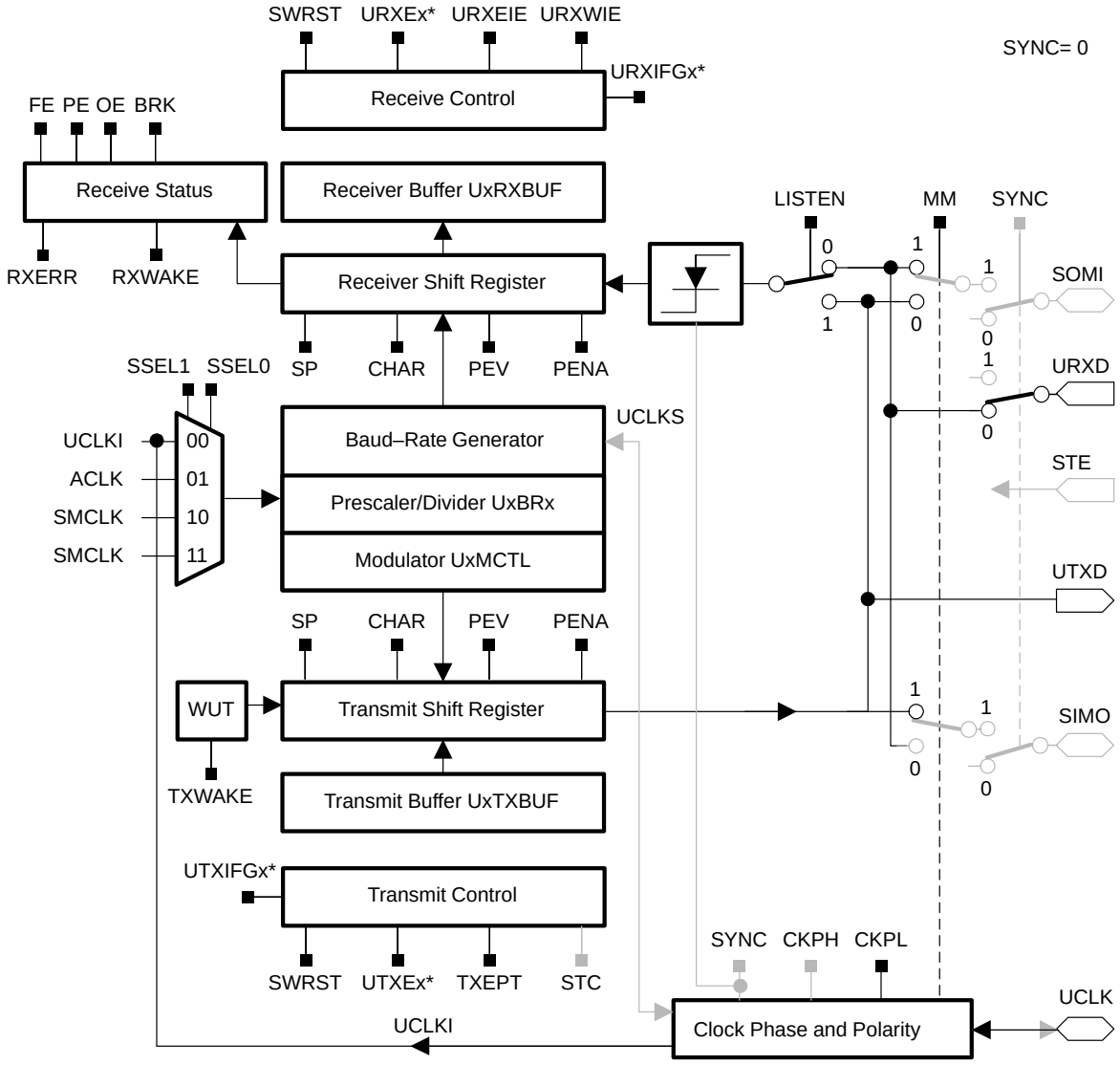
In asynchronous mode, the USART connects the MSP430 to an external system via two external pins, URXD and UTXD. UART mode is selected when the SYNC bit is cleared.

UART mode features include:

- 7- or 8-bit data with odd, even, or non-parity
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- LSB-first data transmit and receive
- Built-in idle-line and address-bit communication protocols for multiprocessor systems
- Receiver start-edge detection for auto-wake up from LPMx modes
- Programmable baud rate with modulation for fractional baud rate support
- Status flags for error detection and suppression and address detection
- Independent interrupt capability for receive and transmit

Figure 13–1 shows the USART when configured for UART mode.

Figure 13–1. USART Block Diagram: UART Mode



** Refer to the device-specific datasheet for SFR locations*

13.2 USART Operation: UART Mode

In UART mode, the USART transmits and receives characters at a bit rate asynchronous to another device. Timing for each character is based on the selected baud rate of the USART. The transmit and receive functions use the same baud rate frequency.

13.2.1 USART Initialization and Reset

The USART is reset by a PUC or by setting the SWRST bit. After a PUC, the SWRST bit is automatically set, keeping the USART in a reset condition. When set, the SWRST bit resets the URXIEx, UTXIEx, URXIFGx, RXWAKE, TXWAKE, RXERR, BRK, PE, OE, and FE bits and sets the UTXIFGx and TXEPT bits. The receive and transmit enable flags, URXEx and UTXEx, are not altered by SWRST. Clearing SWRST releases the USART for operation. See also chapter *USART Module, I2C mode* for USART0 when reconfiguring from I^2C mode to UART mode.

Note: Initializing or Re-Configuring the USART Module

The required USART initialization/re-configuration process is:

1) Set SWRST (`BIS.B #SWRST,&UxCTL`)

2) Initialize all USART registers with SWRST = 1 (including UxCTL)

3) Enable USART module via the MEx SFRs (URXEx and/or UTXEx)

4) Clear SWRST via software (`BIC.B #SWRST,&UxCTL`)

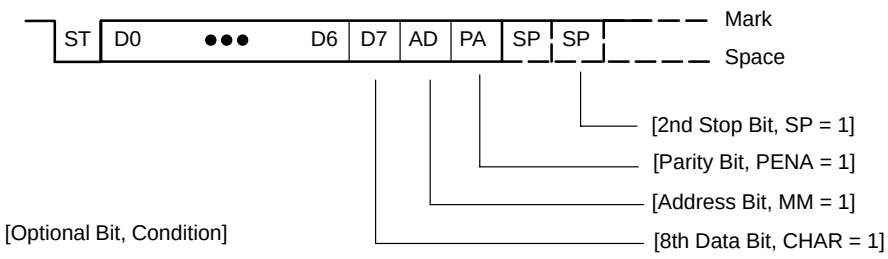
5) Enable interrupts (optional) via the IEx SFRs (URXIEx and/or UTXIEx)

Failure to follow this process may result in unpredictable USART behavior.

13.2.2 Character Format

The UART character format, shown in Figure 13–2, consists of a start bit, seven or eight data bits, an even/odd/no parity bit, an address bit (address-bit mode), and one or two stop bits. The bit period is defined by the selected clock source and setup of the baud rate registers.

Figure 13–2. Character Format



13.2.3 Asynchronous Communication Formats

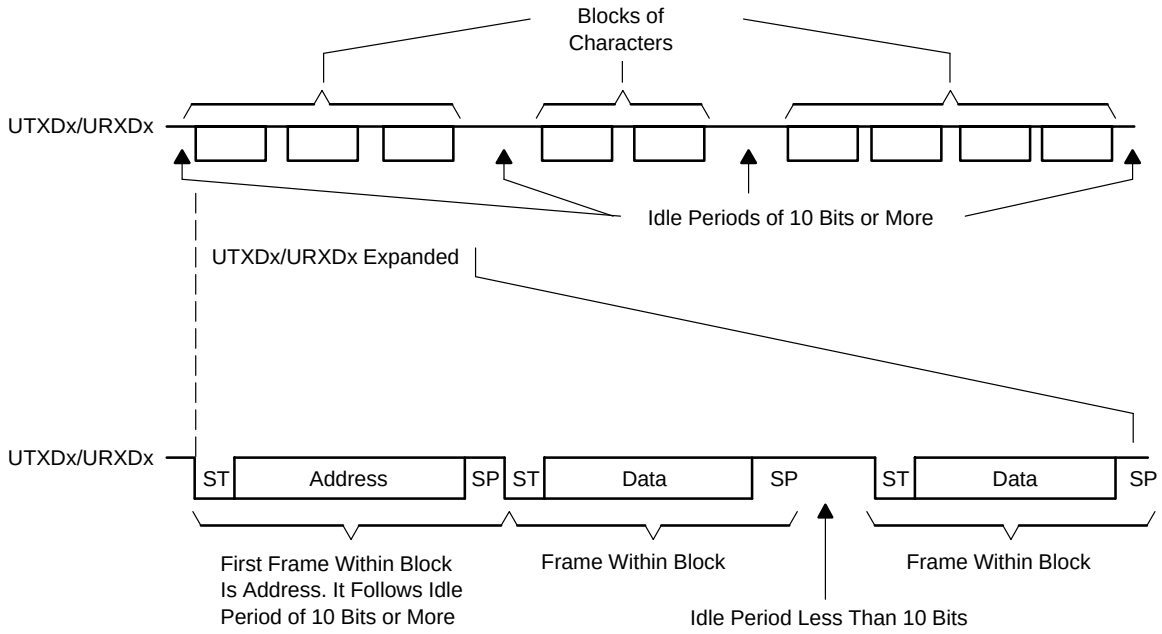
When two devices communicate asynchronously, the idle-line format is used for the protocol. When three or more devices communicate, the USART supports the idle-line and address-bit multiprocessor communication formats.

Idle-Line Multiprocessor Format

When MM = 0, the idle-line multiprocessor format is selected. Blocks of data are separated by an idle time on the transmit or receive lines as shown in Figure 13–3. An idle receive line is detected when 10 or more continuous ones (marks) are received after the first stop bit of a character. When two stop bits are used for the idle line the second stop bit 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 is used as an address tag for each frame. In the idle-line multiprocessor format, this bit is set when a received character is an address and is transferred to UxRXBUF.

Figure 13–3. Idle-Line Format



The URXWIE bit is used to control data reception in the idle-line multiprocessor format. When the URXWIE bit is set, all non-address characters are assembled but not transferred into the UxRXBUF, and interrupts are not generated. When an address character is received, the receiver is temporarily activated to transfer the character to UxRXBUF and sets the URXIFGx interrupt flag. Any applicable error flag is also set. The user can then validate the received address.

If an address is received, user software can validate the address and must reset URXWIE to continue receiving data. If URXWIE remains set, only address characters will be received. The URXWIE bit is not modified by the USART hardware automatically.

For address transmission in idle-line multiprocessor format, a precise idle period can be generated by the USART to generate address character identifiers on UTXDx. The wake-up temporary (WUT) flag is an internal flag double-buffered with the user-accessible TXWAKE bit. When the transmitter is loaded from UxTXBUF, WUT is also loaded from TXWAKE resetting the TXWAKE bit.

The following procedure sends out an idle frame to indicate an address character will follow:

1) Set TXWAKE, then write any character to UxTXBUF. UxTXBUF must be ready for new data (UTXIFGx = 1).

   The TXWAKE value is shifted to WUT and the contents of UxTXBUF are shifted to the transmit shift register when the shift register is ready for new data. This sets WUT, which suppresses the start, data, and parity bits of a normal transmission, then transmits an idle period of exactly 11 bits. When two stop bits are used for the idle line, the second stop bit is counted as the first mark bit of the idle period. TXWAKE is reset automatically.

2) Write desired address character to UxTXBUF. UxTXBUF must be ready for new data (UTXIFGx = 1).

   The new character representing the specified address is shifted out following the address-identifying idle period on UTXDx. Writing the first “don’t care” character to UxTXBUF is necessary in order to shift the TXWAKE bit to WUT and generate an idle-line condition. This data is discarded and does not appear on UTXDx.

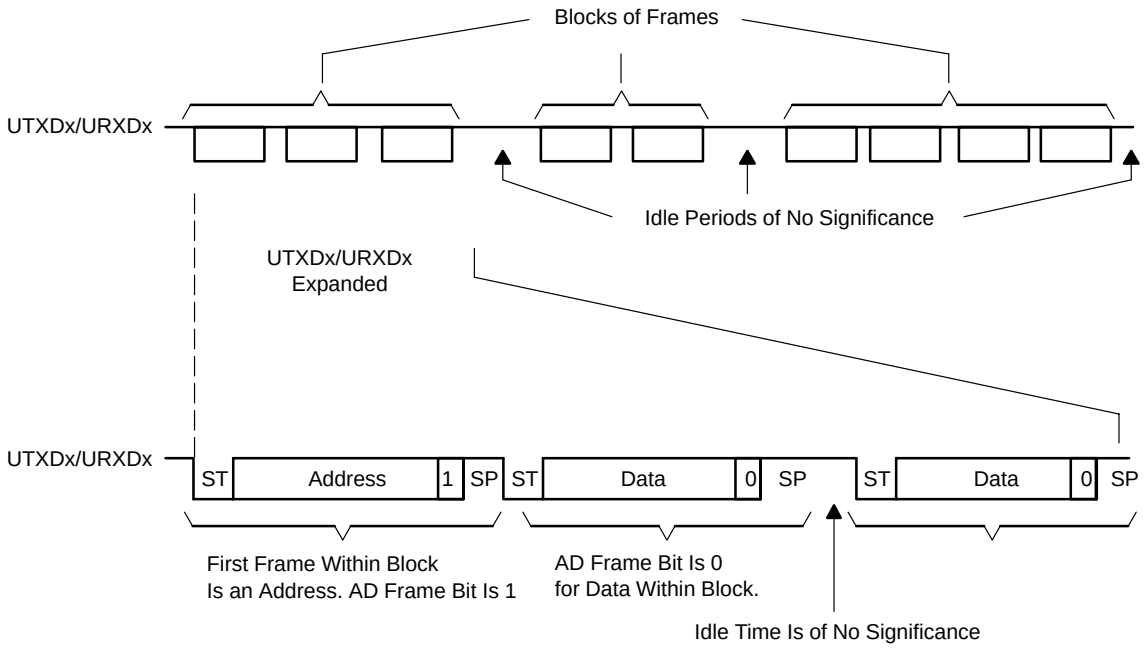
Address-Bit Multiprocessor Format

When MM = 1, the address-bit multiprocessor format is selected. Each processed character contains an extra bit used as an address indicator shown in Figure 13–4. The first character in a block of frames carries a set address bit which indicates that the character is an address. The USART RXWAKE bit is set when a received character is a valid address frame and is transferred to UxRXBUF.

The URXWIE bit is used to control data reception in the address-bit multiprocessor format. If URXWIE is set, data characters (address bit = 0) are assembled by the receiver but are not transferred to UxRXBUF and no interrupts are generated. When a character containing a set address bit is received, the receiver is temporarily activated to transfer the character to UxRXBUF and set URXIFGx. All applicable error status flags are also set.

If an address is received, user software must reset URXWIE to continue receiving data. If URXWIE remains set, only address characters (address bit = 1) will be received. The URXWIE bit is not modified by the USART hardware automatically.

Figure 13–4. Address-Bit Multiprocessor Format



For address transmission in 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 the character transferred from UxTXBUF to the transmit shift register, automatically clearing the TXWAKE bit. TXWAKE must not be cleared by software. It is cleared by USART hardware after it is transferred to WUT or by setting SWRST.

Automatic Error Detection

Glitch suppression prevents the USART from being accidentally started. Any low-level on URXDx shorter than the deglitch time t_τ (approximately 300 ns) will be ignored. See the device-specific datasheet for parameters.

When a low period on URXDx exceeds t_τ a majority vote is taken for the start bit. If the majority vote fails to detect a valid start bit the USART halts character reception and waits for the next low period on URXDx. The majority vote is also used for each bit in a character to prevent bit errors.

The USART module automatically detects framing errors, parity errors, overrun errors, and break conditions when receiving characters. The bits FE, PE, OE, and BRK are set when their respective condition is detected. When any of these error flags are set, RXERR is also set. The error conditions are described in Table 13–1.

Table 13–1. Receive Error Conditions

Error Condition	Description
Framing error	A framing error occurs when a low stop bit is detected. When two stop bits are used, only the first stop bit is checked for framing error. When a framing error is detected, the FE bit is set.
Parity error	A parity error is a mismatch between the number of 1s in a frame and the value of the parity bit. When an address bit is included in the frame, it is included in the parity calculation. When a parity error is detected, the PE bit is set.
Receive overrun error	An overrun error occurs when a character is loaded into UxRXBUF before the prior character has been read. When an overrun occurs, the OE bit is set.
Break condition	A break condition is a period of 10 or more low bits received on URXDx after a missing stop bit. When a break condition is detected, the BRK bit is set. A break condition can also set the interrupt flag URXIFGx.

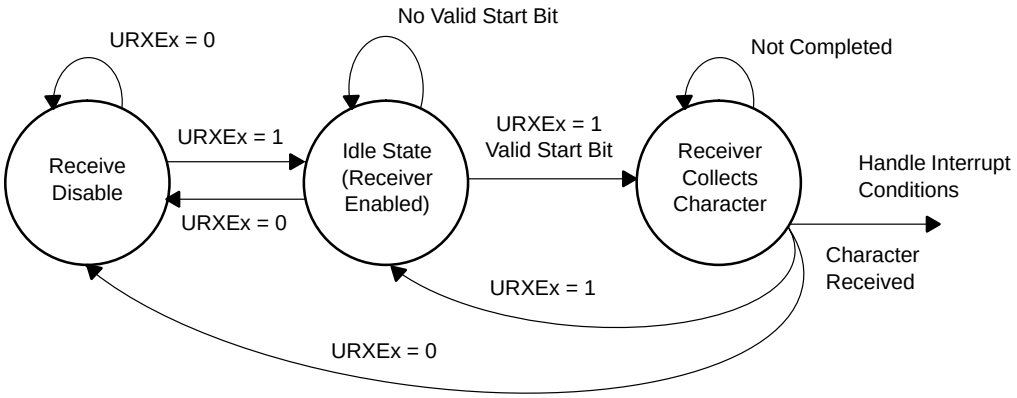
When URXEIE = 0 and a framing error, parity error, or break condition is detected, no character is received into UxRXBUF. When URXEIE = 1, characters are received into UxRXBUF and any applicable error bit is set.

When any of the FE, PE, OE, BRK, or RXERR bits is set, the bit remains set until user software resets it or UxRXBUF is read.

13.2.4 USART Receive Enable

The receive enable bit, URXEx, enables or disables data reception on URXDx as shown in Figure 13–5. Disabling the USART receiver stops the receive operation following completion of any character currently being received or immediately if no receive operation is active. The receive-data buffer, UxRXBUF, contains the character moved from the RX shift register after the character is received.

Figure 13–5. State Diagram of Receiver Enable



Note: Re-Enabling the Receiver (Setting URXEx): UART Mode

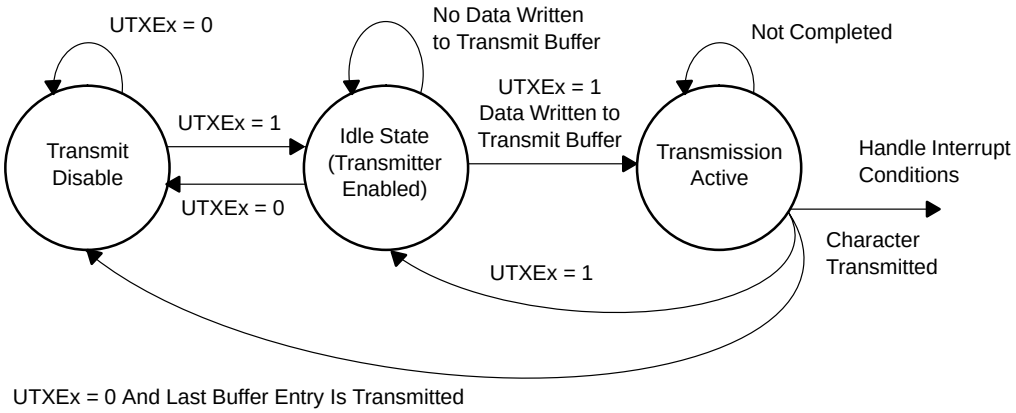
When the receiver is disabled (URXEx = 0), re-enabling the receiver (URXEx = 1) is asynchronous to any data stream that may be present on URXDx at the time. Synchronization can be performed by testing for an idle line condition before receiving a valid character (see URXWIE).

13.2.5 USART Transmit Enable

When UTXEx is set, the UART transmitter is enabled. Transmission is initiated by writing data to UxTXBUF. The data is then moved to the transmit shift register on the next BITCLK after the TX shift register is empty, and transmission begins. This process is shown in Figure 13–6.

When the UTXEx bit is reset the transmitter is stopped. Any data moved to UxTXBUF and any active transmission of data currently in the transmit shift register prior to clearing UTXEx will continue until all data transmission is completed.

Figure 13–6. State Diagram of Transmitter Enable



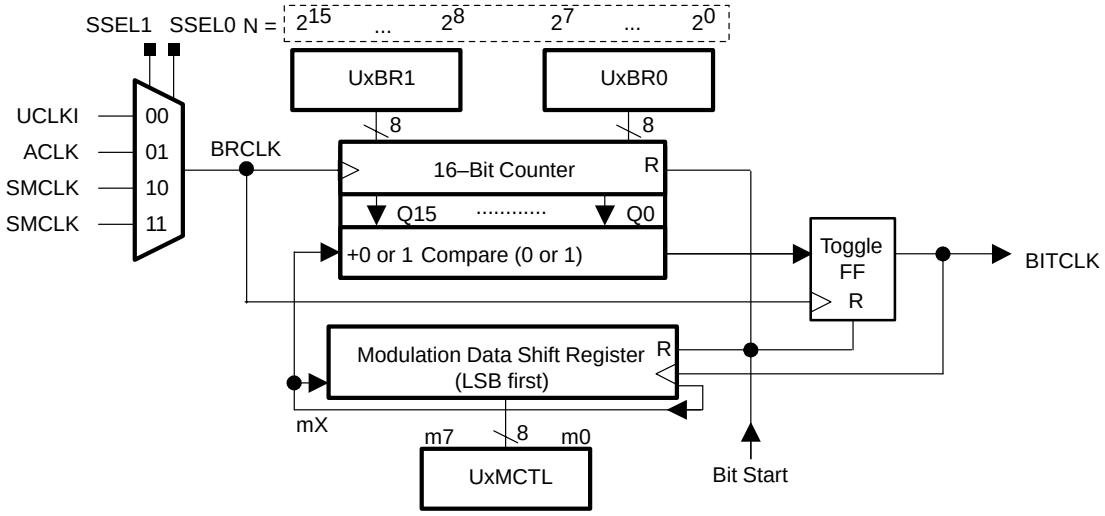
When the transmitter is enabled (UTXEx = 1), data should not be written to UxTXBUF unless it is ready for new data indicated by UTXIFGx = 1. Violation can result in an erroneous transmission if data in UxTXBUF is modified as it is being moved into the TX shift register.

It is recommended that the transmitter be disabled (UTXEx = 0) only after any active transmission is complete. This is indicated by a set transmitter empty bit (TXEPT = 1). Any data written to UxTXBUF while the transmitter is disabled will be held in the buffer but will not be moved to the transmit shift register or transmitted. Once UTXEx is set, the data in the transmit buffer is immediately loaded into the transmit shift register and character transmission resumes.

13.2.6 UART Baud Rate Generation

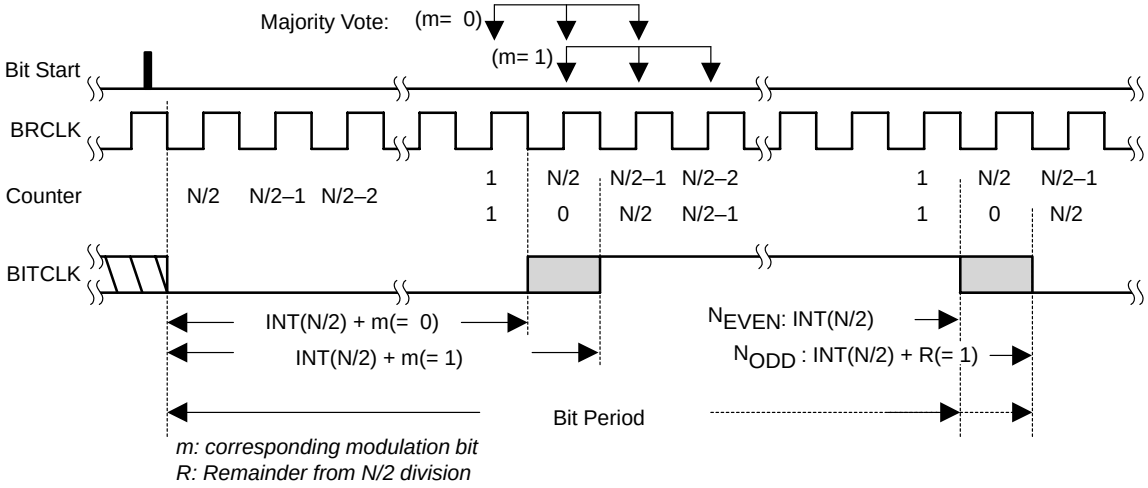
The USART baud rate generator is capable of producing standard baud rates from non-standard source frequencies. The baud rate generator uses one prescaler/divider and a modulator as shown in Figure 13–7. This combination supports fractional divisors for baud rate generation. The maximum USART baud rate is one-third the UART source clock frequency BRCLK.

Figure 13–7. MSP430 Baud Rate Generator



Timing for each bit is shown in Figure 13–8. For each bit received, a majority vote is taken to determine the bit value. These samples occur at the N/2–1, N/2, and N/2+1 BRCLK periods, where N is the number of BRCLKs per BITCLK.

Figure 13–8. BITCLK Baud Rate Timing



Baud Rate Bit Timing

The first stage of the baud rate generator is the 16-bit counter and comparator. At the beginning of each bit transmitted or received, the counter is loaded with INT(N/2) where N is the value stored in the combination of UxBR0 and UxBR1. The counter reloads INT(N/2) for each bit period half-cycle, giving a total bit period of N BRCLKs. For a given BRCLK clock source, the baud rate used determines the required division factor N:

$$N = \frac{\text{BRCLK}}{\text{baud rate}}$$

The division factor N is often a non-integer value of which the integer portion can be realized by the prescaler/divider. The second stage of the baud rate generator, the modulator, is used to meet the fractional part as closely as possible. The factor *N* is then defined as:

$$N = UxBR + \frac{1}{n}\sum_{i=0}^{n-1} m_i$$

Where:

N: Target division factor
UxBR: 16-bit representation of registers UxBR0 and UxBR1
i: Bit position in the frame
n: Total number of bits in the frame
m_i: Data of each corresponding modulation bit (1 or 0)

$$Baud\ rate = \frac{BRCLK}{N} = \frac{BRCLK}{UxBR + \frac{1}{n}\sum_{i=0}^{n-1} m_i}$$

The BITCLK can be adjusted from bit to bit with the modulator to meet timing requirements when a non-integer divisor is needed. Timing of each bit is expanded by one BRCLK clock cycle if the modulator bit m_i is set. Each time a bit is received or transmitted, the next bit in the modulation control register determines the timing for that bit. A set modulation bit increases the division factor by one while a cleared modulation bit maintains the division factor given by UxBR.

The timing for the start bit is determined by UxBR plus m0, the next bit is determined by UxBR plus m1, and so on. The modulation sequence begins with the LSB. When the character is greater than 8 bits, the modulation sequence restarts with m0 and continues until all bits are processed.

Determining the Modulation Value

Determining the modulation value is an interactive process. Using the timing error formula provided, beginning with the start bit , the individual bit errors are calculated with the corresponding modulator bit set and cleared. The modulation bit setting with the lower error is selected and the next bit error is calculated. This process is continued until all bit errors are minimized. When a frame contains more than 8 bits, the modulation bits repeat. For example, the 9th bit of a frame uses modulation bit 0.

Transmit Bit Timing

The timing for each character is the sum of the individual bit timings. By modulating each bit, the cumulative bit error is reduced. The individual bit error can be calculated by:

$$Error\ [\%] = \left\{\frac{baud\ rate}{BRCLK} \times \left[(j+1) \times UxBR + \sum_{i=0}^{n-1} m_i\right] - (j+1)\right\} \times 100\%$$

With:

baud rate: Desired baud rate
BRCLK: Input frequency – UCLKI, ACLK, or SMCLK
j: Bit position - 0 for the start bit, 1 for data bit D0, and so on
UxBR: Division factor in registers UxBR1 and UxBR0

For example, the transmit errors for the following conditions are are calculated:

Baud rate = 2400
BRCLK = 32,768 Hz (ACLK)
UxBR = 13, since the ideal division factor is 13.65
UxMCTL = 6Bh: m7=0, m6=1, m5=1, m4=0, m3=1, m2=0, m1=1, and m0=1. The LSB of UxMCTL is used first.

$$\text{Start bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((0+1) \times \text{UxBR} + 1) - 1\right) \times 100\% = 2.54\%$$

$$\text{Data bit D0 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((1+1) \times \text{UxBR} + 2) - 2\right) \times 100\% = 5.08\%$$

$$\text{Data bit D1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((2+1) \times \text{UxBR} + 2) - 3\right) \times 100\% = 0.29\%$$

$$\text{Data bit D2 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((3+1) \times \text{UxBR} + 3) - 4\right) \times 100\% = 2.83\%$$

$$\text{Data bit D3 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((4+1) \times \text{UxBR} + 3) - 5\right) \times 100\% = -1.95\%$$

$$\text{Data bit D4 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((5+1) \times \text{UxBR} + 4) - 6\right) \times 100\% = 0.59\%$$

$$\text{Data bit D5 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((6+1) \times \text{UxBR} + 5) - 7\right) \times 100\% = 3.13\%$$

$$\text{Data bit D6 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((7+1) \times \text{UxBR} + 5) - 8\right) \times 100\% = -1.66\%$$

$$\text{Data bit D7 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((8+1) \times \text{UxBR} + 6) - 9\right) \times 100\% = 0.88\%$$

$$\text{Parity bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((9+1) \times \text{UxBR} + 7) - 10\right) \times 100\% = 3.42\%$$

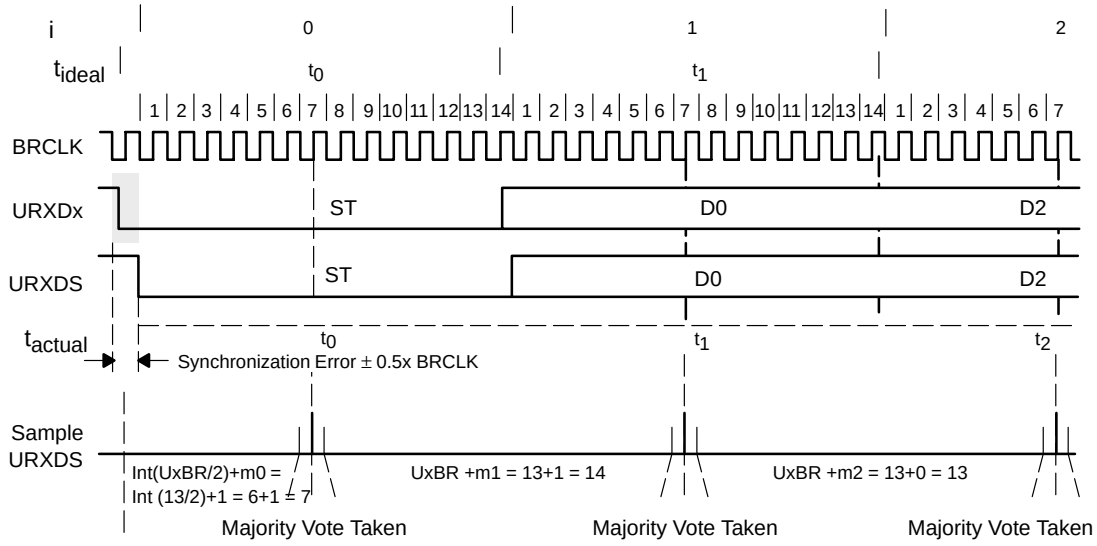
$$\text{Stop bit 1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times ((10+1) \times \text{UxBR} + 7) - 11\right) \times 100\% = -1.37\%$$

The results show the maximum per-bit error to be 5.08% of a BITCLK period.

Receive Bit Timing

Receive timing consists of two error sources. The first is the bit-to-bit timing error. The second is the error between a start edge occurring and the start edge being accepted by the USART. Figure 13–9 shows the asynchronous timing errors between data on the URXDx pin and the internal baud-rate clock.

Figure 13–9. Receive Error



The ideal start bit timing $t_{ideal(0)}$ is half the baud-rate timing $t_{baud\ rate}$ because the bit is tested in the middle of its period. The ideal baud rate timing $t_{ideal(i)}$ for the remaining character bits is the baud rate timing $t_{baud\ rate}$. The individual bit errors can be calculated by:

$$Error\ [\%] = \left(\frac{baud\ rate}{BRCLK} \times \left\{ 2 \times \left[m0 + int\left(\frac{UxBR}{2}\right) \right] + \left(i \times UxBR + \sum_{i=1}^{n-1} m_i \right) \right\} - 1 - j \right) \times 100\%$$

Where:

baud rate is the required baud rate
BRCLK is the input frequency—selected for UCLK, ACLK, or SMCLK
j = 0 for the start bit, 1 for data bit D0, and so on
UxBR is the division factor in registers UxBR1 and UxBR0

For example, the receive errors for the following conditions are are calculated:

Baud rate = 2400
BRCLK = 32,768 Hz (ACLK)
UxBR = 13, since the ideal division factor is 13.65
UxMCTL = 6B:m7=0, m6=1, m5=1, m4=0, m3=1, m2=0, m1=1 and m0=1 The LSB of UxMCTL is used first.

$$\text{Start bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (0 \times \text{UxBR} + 0)] - 1 - 0\right) \times 100\% = 2.54\%$$

$$\text{Data bit D0 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (1 \times \text{UxBR} + 1)]–1–1\right) \times 100\% = 5.08\%$$

$$\text{Data bit D1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (2 \times \text{UxBR} + 1)]–1–2\right) \times 100\% = 0.29\%$$

$$\text{Data bit D2 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (3 \times \text{UxBR} + 2)]–1–3\right) \times 100\% = 2.83\%$$

$$\text{Data bit D3 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (4 \times \text{UxBR} + 2)]–1–4\right) \times 100\% = –1.95\%$$

$$\text{Data bit D4 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (5 \times \text{UxBR} + 3)]–1–5\right) \times 100\% = 0.59\%$$

$$\text{Data bit D5 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (6 \times \text{UxBR} + 4)]–1–6\right) \times 100\% = 3.13\%$$

$$\text{Data bit D6 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (7 \times \text{UxBR} + 4)]–1–7\right) \times 100\% = –1.66\%$$

$$\text{Data bit D7 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (8 \times \text{UxBR} + 5)]–1–8\right) \times 100\% = 0.88\%$$

$$\text{Parity bit Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (9 \times \text{UxBR} + 6)]–1–9\right) \times 100\% = 3.42\%$$

$$\text{Stop bit 1 Error [\%]} = \left(\frac{\text{baud rate}}{\text{BRCLK}} \times [2x(1 + 6) + (10 \times \text{UxBR} + 6)]–1–10\right) \times 100\% = –1.37\%$$

The results show the maximum per-bit error to be 5.08% of a BITCLK period.

Typical Baud Rates and Errors

Standard baud rate frequency data for UxBRx and UxMCTL are listed in Table 13–2 for a 32,768-Hz watch crystal (ACLK) and a typical 1,048,576-Hz SMCLK.

The receive 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.

Table 13–2. Commonly Used Baud Rates, Baud Rate Data, and Errors

	Divide by		A: BRCLK = 32,768 Hz						B: BRCLK = 1,048,576 Hz				
Baud Rate	A:	B:	UxBR1	UxBR0	UxMCTL	Max. TX Error %	Max. RX Error %	Synchr. RX Error %	UxBR1	UxBR0	UxMCTL	Max. TX Error %	Max. RX Error %
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		9.1							0	09	08	–5/7	±7

13.2.7 USART Interrupts

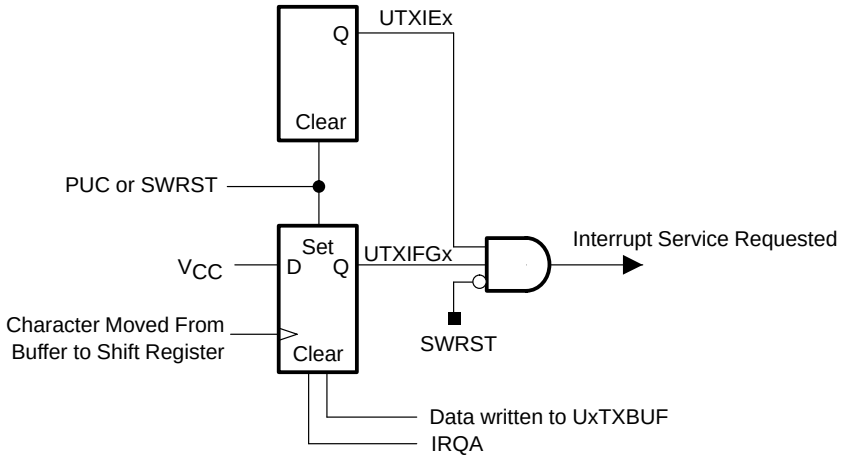
The USART has one interrupt vector for transmission and one interrupt vector for reception.

USART Transmit Interrupt Operation

The UTXIFGx interrupt flag is set by the transmitter to indicate that UxTXBUF is ready to accept another character. An interrupt request is generated if UTXIEx and GIE are also set. UTXIFGx is automatically reset if the interrupt request is serviced or if a character is written to UxTXBUF.

UTXIFGx is set after a PUC or when SWRST = 1. UTXIEx is reset after a PUC or when SWRST = 1. The operation is shown is Figure 13–10.

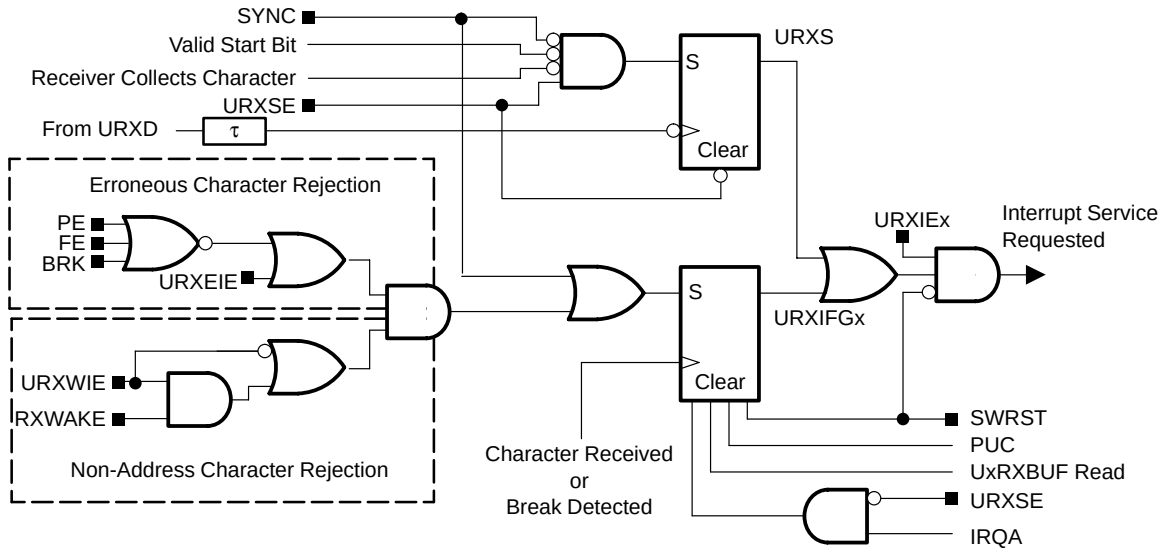
Figure 13–10. Transmit Interrupt Operation



USART Receive Interrupt Operation

The URXIFGx interrupt flag is set each time a character is received and loaded into UxRXBUF. An interrupt request is generated if URXIEx and GIE are also set. URXIFGx and URXIEx are reset by a system reset PUC signal or when SWRST = 1. URXIFGx is automatically reset if the pending interrupt is served (when URXSE = 0) or when UxRXBUF is read. The operation is shown in Figure 13–11.

Figure 13–11. Receive Interrupt Operation



URXEIE is used to enable or disable erroneous characters from setting URXIFGx. When using multiprocessor addressing modes, URXWIE is used to auto-detect valid address characters and reject unwanted data characters.

Two types of characters do not set URXIFGx:

- Erroneous characters when URXEIE = 0
- Non-address characters when URXWIE = 1

When URXEIE = 1 a break condition will set the BRK bit and the URXIFGx flag.

Receive-Start Edge Detect Operation

The URXSE bit enables the receive start-edge detection feature. The recommended usage of the receive-start edge feature is when BRCLK is sourced by the DCO and when the DCO is off because of low-power mode operation. The ultra-fast turn-on of the DCO allows character reception after the start edge detection.

When URXSE, URXIEx and GIE are set and a start edge occurs on URXDx, the internal signal URXS will be set. When URXS is set, a receive interrupt request is generated but URXIFGx is not set. User software in the receive interrupt service routine can test URXIFGx to determine the source of the interrupt. When URXIFGx = 0 a start edge was detected and when URXIFGx = 1 a valid character (or break) was received.

When the ISR determines the interrupt request was from a start edge, user software toggles URXSE, and must enable the BRCLK source by returning from the ISR to active mode or to a low-power mode where the source is active. If the ISR returns to a low-power mode where the BRCLK source is inactive, the character will not be received. Toggling URXSE clears the URXS signal and re-enables the start edge detect feature for future characters. See chapter *System Resets, Interrupts, and Operating Modes* for information on entering and exiting low-power modes.

The now active BRCLK allows the USART to receive the balance of the character. After the full character is received and moved to UxRXBUF, URXIFGx is set and an interrupt service is again requested. Upon ISR entry, URXIFGx = 1 indicating a character was received. The URXIFGx flag is cleared when user software reads UxRXBUF.

```
; Interrupt handler for frame start condition and
; Character receive. BRCLK = DCO.

U0RX_Int  BIT.B #URXIFG0,&IFG2  ; Test URXIFGx to determine
          JNE   ST_COND         ; If start or character
          MOV.B &UxRXBUF,dst    ; Read buffer
          ...                   ;
          RETI                  ;

ST_COND   BIC.B #URXSE,&U0TCTL  ; Clear URXS signal
          BIS.B #URXSE,&U0TCTL  ; Re-enable edge detect
          BIC   #SCG0+SCG1,0(SP) ; Enable BRCLK = DCO
          RETI                  ;
```

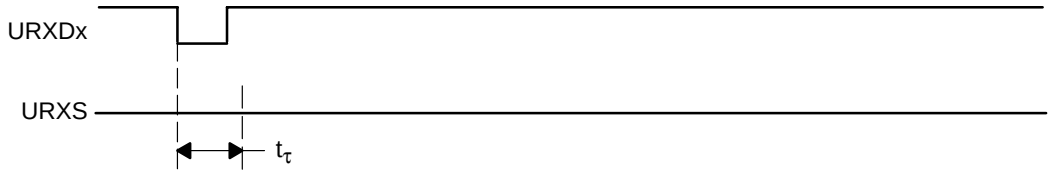
Note: Break Detect With Halted UART Clock

When using the receive start-edge detect feature a break condition cannot be detected when the BRCLK source is off.

Receive-Start Edge Detect Conditions

When URXSE = 1, glitch suppression prevents the USART from being accidentally started. Any low-level on URXDx shorter than the deglitch time t_τ (approximately 300 ns) will be ignored by the USART and no interrupt request will be generated as shown in Figure 13–12. See the device-specific datasheet for parameters.

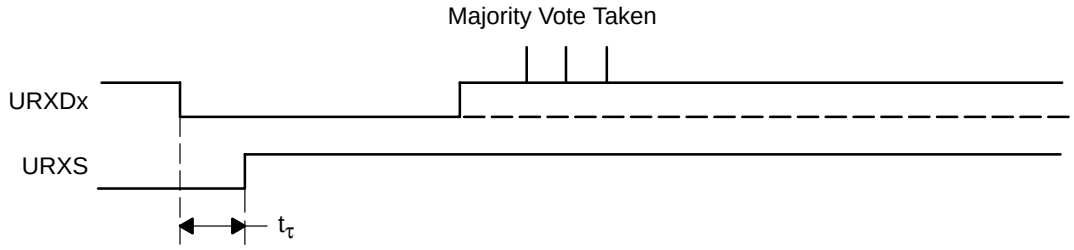
Figure 13–12. Glitch Suppression, USART Receive Not Started



When a glitch is longer than t_τ, or a valid start bit occurs on URXDx, the USART receive operation is started and a majority vote is taken as shown in Figure 13–13. If the majority vote fails to detect a start bit the USART halts character reception.

If character reception is halted, an active BRCLK is not necessary. A time-out period longer than the character receive duration can be used by software to indicate that a character was not received in the expected time and the software can disable BRCLK.

Figure 13–13. Glitch Suppression, USART Activated



13.3 USART Registers: UART Mode

Table 13–3 lists the registers for all devices implementing a USART module. Table 13–4 applies only to devices with a second USART module, USART1.

Table 13–3. USART0 Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USART control register	U0CTL	Read/write	070h	001h after PUC
Transmit control register	U0TCTL	Read/write	071h	001h after PUC
Receive control register	U0RCTL	Read/write	072h	000h after PUC
Modulation control register	U0MCTL	Read/write	073h	Unchanged
Baud rate control register 0	U0BR0	Read/write	074h	Unchanged
Baud rate control register 1	U0BR1	Read/write	075h	Unchanged
Receive buffer register	U0RXBUF	Read	076h	Unchanged
Transmit buffer register	U0TXBUF	Read/write	077h	Unchanged
SFR module enable register 1†	ME1	Read/write	004h	000h after PUC
SFR interrupt enable register 1†	IE1	Read/write	000h	000h after PUC
SFR interrupt flag register 1†	IFG1	Read/write	002h	082h after PUC

† Does not apply to ’12xx devices. Refer to the register definitions for registers and bit positions for these devices.

Table 13–4. USART1 Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USART control register	U1CTL	Read/write	078h	001h after PUC
Transmit control register	U1TCTL	Read/write	079h	001h after PUC
Receive control register	U1RCTL	Read/write	07Ah	000h after PUC
Modulation control register	U1MCTL	Read/write	07Bh	Unchanged
Baud rate control register 0	U1BR0	Read/write	07Ch	Unchanged
Baud rate control register 1	U1BR1	Read/write	07Dh	Unchanged
Receive buffer register	U1RXBUF	Read	07Eh	Unchanged
Transmit buffer register	U1TXBUF	Read/write	07Fh	Unchanged
SFR module enable register 2	ME2	Read/write	005h	000h after PUC
SFR interrupt enable register 2	IE2	Read/write	001h	000h after PUC
SFR interrupt flag register 2	IFG2	Read/write	003h	000h after PUC

Note: Modifying SFR bits

To avoid modifying control bits of other modules, it is recommended to set or clear the IEx and IFGx bits using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions.

UxCTL, USART Control Register

7	6	5	4	3	2	1	0
PENA	**PEV**	**SPB**	**CHAR**	**LISTEN**	**SYNC**	**MM**	**SWRST**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–1

PENA Bit 7 Parity enable
- 0 Parity disabled.
- 1 Parity enabled. Parity bit is generated (UTXDx) and expected (URXDx). In address-bit multiprocessor mode, the address bit is included in the parity calculation.

PEV Bit 6 Parity select. PEV is not used when parity is disabled.
- 0 Odd parity
- 1 Even parity

SPB Bit 5 Stop bit select. Number of stop bits transmitted. The receiver always checks for one stop bit.
- 0 One stop bit
- 1 Two stop bits

CHAR Bit 4 Character length. Selects 7-bit or 8-bit character length.
- 0 7-bit data
- 1 8-bit data

LISTEN Bit 3 Listen enable. The LISTEN bit selects loopback mode.
- 0 Disabled
- 1 Enabled. UTXDx is internally fed back to the receiver.

SYNC Bit 2 Synchronous mode enable
- 0 UART mode
- 1 SPI Mode

MM Bit 1 Multiprocessor mode select
- 0 Idle-line multiprocessor protocol
- 1 Address-bit multiprocessor protocol

SWRST Bit 0 Software reset enable
- 0 Disabled. USART reset released for operation
- 1 Enabled. USART logic held in reset state

UxTCTL, USART Transmit Control Register

7	6	5	4	3	2	1	0
Unused	**CKPL**	**SSELx**		**URXSE**	**TXWAKE**	**Unused**	**TXEPT**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–1

Unused	Bit 7	Unused
CKPL	Bit 6	Clock polarity select 0 UCLKI = UCLK 1 UCLKI = inverted UCLK
SSELx	Bits 5-4	Source select. These bits select the BRCLK source clock. 00 UCLKI 01 ACLK 10 SMCLK 11 SMCLK
URXSE	Bit 3	UART receive start-edge. The bit enables the UART receive start-edge feature. 0 Disabled 1 Enabled
TXWAKE	Bit 2	Transmitter wake 0 Next frame transmitted is data 1 Next frame transmitted is an address
Unused	Bit 1	Unused
TXEPT	Bit 0	Transmitter empty flag 0 UART is transmitting data and/or data is waiting in UxTXBUF 1 Transmitter shift register and UxTXBUF are empty or SWRST=1

UxRCTL, USART Receive Control Register

7	6	5	4	3	2	1	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

FE Bit 7 Framing error flag
- 0 No error
- 1 Character received with low stop bit

PE Bit 6 Parity error flag. When PENA = 0, PE is read as 0.
- 0 No error
- 1 Character received with parity error

OE Bit 5 Overrun error flag. This bit is set when a character is transferred into UxRXBUF before the previous character was read.
- 0 No error
- 1 Overrun error occurred

BRK Bit 4 Break detect flag
- 0 No break condition
- 1 Break condition occurred

URXEIE Bit 3 Receive erroneous-character interrupt-enable
- 0 Erroneous characters rejected and URXIFGx is not set
- 1 Erroneous characters received will set URXIFGx

URXWIE Bit 2 Receive wake-up interrupt-enable. This bit enables URXIFGx to be set when an address character is received. When URXEIE = 0, an address character will not set URXIFGx if it is received with errors.
- 0 All received characters set URXIFGx
- 1 Only received address characters set URXIFGx

RXWAKE Bit 1 Receive wake-up flag
- 0 Received character is data
- 1 Received character is an address

RXERR Bit 0 Receive error flag. This bit indicates a character was received with error(s). When RXERR = 1, on or more error flags (FE,PE,OE, BRK) is also set. RXERR is cleared when UxRXBUF is read.
- 0 No receive errors detected
- 1 Receive error detected

UxBR0, USART Baud Rate Control Register 0

7	6	5	4	3	2	1	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

UxBR1, USART Baud Rate Control Register 1

7	6	5	4	3	2	1	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

UxBRx The valid baud-rate control range is $3 \leq$ UxBR $<$ 0FFFFh, where UxBR = {UxBR1+UxBR0}. Unpredictable receive and transmit timing occurs if UxBR <3.

UxMCTL, USART Modulation Control Register

7	6	5	4	3	2	1	0
m7	m6	m5	m4	m3	m2	m1	m0
rw	rw	rw	rw	rw	rw	rw	rw

UxMCTLx Bits 7–0 Modulation bits. These bit select the modulation for BRCLK.

UxRXBUF, USART Receive Buffer Register

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
r	r	r	r	r	r	r	r

UxRXBUFx Bits 7–0 The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UxRXBUF resets the receive-error bits, the RXWAKE bit, and URXIFGx. In 7-bit data mode, UxRXBUF is LSB justified and the MSB is always reset.

UxTXBUF, USART Transmit Buffer Register

7	6	5	4	3	2	1	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

UxTXBUFx Bits 7–0 The transmit data buffer is user accessible and holds the data waiting to be moved into the transmit shift register and transmitted on UTXDx. Writing to the transmit data buffer clears UTXIFGx. The MSB of UxTXBUF is not used for 7-bit data and is reset.

ME1, Module Enable Register 1

7	6	5	4	3	2	1	0
UTXE0†	**URXE0**†						
rw–0	rw–0						

UTXE0† Bit 7 USART0 transmit enable. This bit enables the transmitter for USART0.
0 Module not enabled
1 Module enabled

URXE0† Bit 6 USART0 receive enable. This bit enables the receiver for USART0.
0 Module not enabled
1 Module enabled

Bits 5-0 These bits may be used by other modules. See device-specific datasheet.

† Does not apply to MSP430x12xx devices. See ME2 for the MSP430x12xx USART0 module enable bits

ME2, Module Enable Register 2

7	6	5	4	3	2	1	0
		UTXE1	**URXE1**			**UTXE0**‡	**URXE0**‡
		rw–0	rw–0			rw–0	rw–0

Bits 7-6 These bits may be used by other modules. See device-specific datasheet.

UTXE1 Bit 5 USART1 transmit enable. This bit enables the transmitter for USART1.
0 Module not enabled
1 Module enabled

URXE1 Bit 4 USART1 receive enable. This bit enables the receiver for USART1.
0 Module not enabled
1 Module enabled

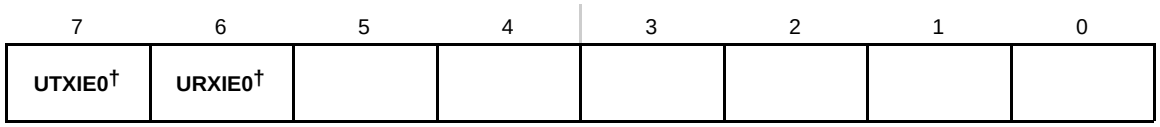
Bits 3-2 These bits may be used by other modules. See device-specific datasheet.

UTXE0‡ Bit 1 USART0 transmit enable. This bit enables the transmitter for USART0.
0 Module not enabled
1 Module enabled

URXE0‡ Bit 0 USART0 receive enable. This bit enables the receiver for USART0.
0 Module not enabled
1 Module enabled

‡ MSP430x12xx devices only

IE1, Interrupt Enable Register 1



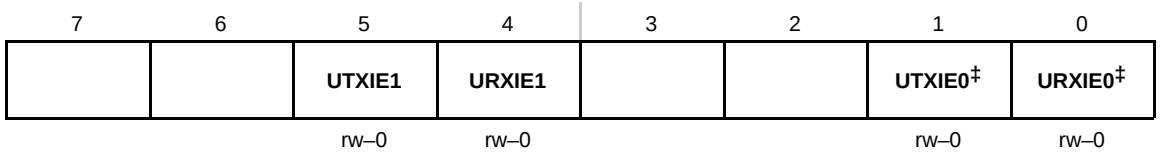
UTXIE0† Bit 7 USART0 transmit interrupt enable. This bit enables the UTXIFG0 interrupt.
0 Interrupt not enabled
1 Interrupt enabled

URXIE0† Bit 6 USART0 receive interrupt enable. This bit enables the URXIFG0 interrupt.
0 Interrupt not enabled
1 Interrupt enabled

Bits 5-0 These bits may be used by other modules. See device-specific datasheet.

† Does not apply to MSP430x12xx devices. See IE2 for the MSP430x12xx USART0 interrupt enable bits

IE2, Interrupt Enable Register 2



Bits 7-6 These bits may be used by other modules. See device-specific datasheet.

UTXIE1 Bit 5 USART1 transmit interrupt enable. This bit enables the UTXIFG1 interrupt.
0 Interrupt not enabled
1 Interrupt enabled

URXIE1 Bit 4 USART1 receive interrupt enable. This bit enables the URXIFG1 interrupt.
0 Interrupt not enabled
1 Interrupt enabled

Bits 3-2 These bits may be used by other modules. See device-specific datasheet.

UTXIE0‡ Bit 1 USART0 transmit interrupt enable. This bit enables the UTXIFG0 interrupt.
0 Interrupt not enabled
1 Interrupt enabled

URXIE0‡ Bit 0 USART0 receive interrupt enable. This bit enables the URXIFG0 interrupt.
0 Interrupt not enabled
1 Interrupt enabled

‡ MSP430x12xx devices only

IFG1, Interrupt Flag Register 1

7	6	5	4	3	2	1	0
UTXIFG0†	URXIFG0†						
rw–1	rw–0						

UTXIFG0† Bit 7 USART0 transmit interrupt flag. UTXIFG0 is set when U0TXBUF is empty.
0 No interrupt pending
1 Interrupt pending

URXIFG0† Bit 6 USART0 receive interrupt flag. URXIFG0 is set when U0RXBUF has received a complete character.
0 No interrupt pending
1 Interrupt pending

Bits 5-0 These bits may be used by other modules. See device-specific datasheet.

† Does not apply to MSP430x12xx devices. See IFG2 for the MSP430x12xx USART0 interrupt flag bits

IFG2, Interrupt Flag Register 2

7	6	5	4	3	2	1	0
		UTXIFG1	URXIFG1			UTXIFG0‡	URXIFG0‡
		rw–1	rw–0			rw–1	rw–0

Bits 7-6 These bits may be used by other modules. See device-specific datasheet.

UTXIFG1 Bit 5 USART1 transmit interrupt flag. UTXIFG1 is set when U1TXBUF empty.
0 No interrupt pending
1 Interrupt pending

URXIFG1 Bit 4 USART1 receive interrupt flag. URXIFG1 is set when U1RXBUF has received a complete character.
0 No interrupt pending
1 Interrupt pending

Bits 3-2 These bits may be used by other modules. See device-specific datasheet.

UTXIFG0‡ Bit 1 USART0 transmit interrupt flag. UTXIFG0 is set when U0TXBUF is empty.
0 No interrupt pending
1 Interrupt pending

URXIFG0‡ Bit 0 USART0 receive interrupt flag. URXIFG0 is set when U0RXBUF has received a complete character.
0 No interrupt pending
1 Interrupt pending

‡ MSP430x12xx devices only

Chapter 14

USART Peripheral Interface, SPI Mode

The universal synchronous/asynchronous receive/transmit (USART) peripheral interface supports two serial modes with one hardware module. This chapter discusses the operation of the synchronous peripheral interface or SPI mode. USART0 is implemented on the MSP430x12xx, MSP430x13xx, and MSP430x15x devices. In addition to USART0, the MSP430x14x and MSP430x16x devices implement a second identical USART module, USART1.

Topic **Page**

14.1 USART Introduction: SPI Mode

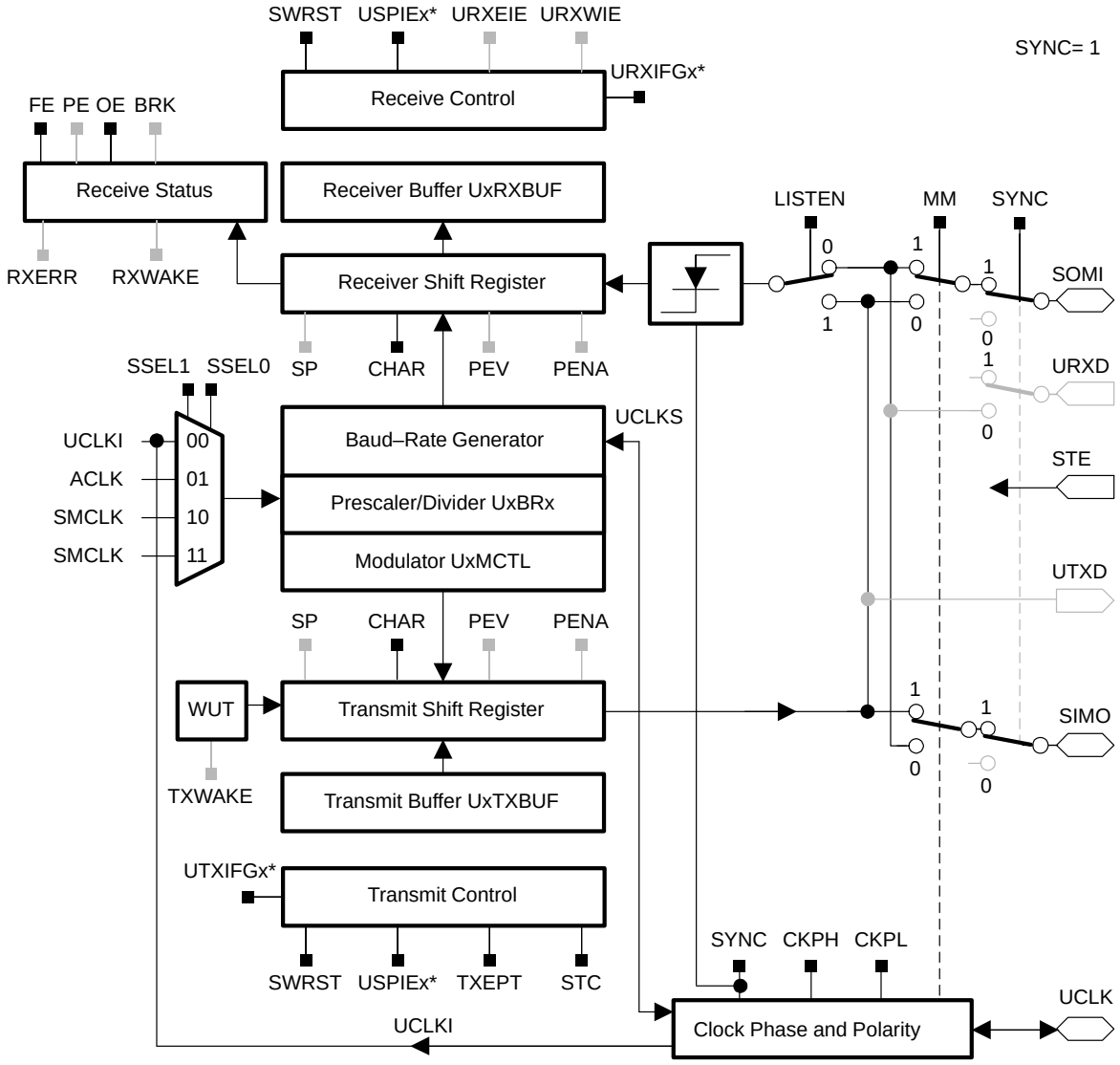
In synchronous mode, the USART connects the MSP430 to an external system via three or four pins: SIMO, SOMI, UCLK, and STE. SPI mode is selected when the SYNC bit is set and the I2C bit is cleared.

SPI mode features include:

- 7- or 8-bit data length
- 3-pin and 4-pin SPI operation
- Master or slave modes
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- Selectable UCLK polarity and phase control
- Programmable UCLK frequency in master mode
- Independent interrupt capability for receive and transmit

Figure 14–1 shows the USART when configured for SPI mode.

Figure 14–1. USART Block Diagram: SPI Mode



** Refer to the device-specific datasheet for SFR locations*

14.2 USART Operation: SPI Mode

In SPI mode, serial data is transmitted and received by multiple devices using a shared clock provided by the master. An additional pin, STE, is provided as to enable a device to receive and transmit data and is controlled by the master.

Three or four signals are used for SPI data exchange:

- SIMO — Slave in, master out
  Master mode: SIMO is the data output line.
  Slave mode: SIMO is the data input line.
- SOMI — Slave out, master in
  Master mode: SOMI is the data input line.
  Slave mode: SOMI is the data output line.
- UCLK — USART SPI clock
  Master mode: UCLK is an output.
  Slave mode: UCLK is an input.
- STE — Slave transmit enable. Used in 4-pin mode to allow multiple masters on a single bus. Not used in 3-pin mode.
  4-Pin master mode:
  When STE is high, SIMO and UCLK operate normally.
  When STE is low, SIMO and UCLK are set to the input direction.
  4-pin slave mode:
  When STE is high, RX/TX operation of the slave is disabled and SOMI is forced to the input direction.
  When STE is low, RX/TX operation of the slave is enabled and SOMI operates normally.

14.2.1 USART Initialization and Reset

The USART is reset by a PUC or by the SWRST bit. After a PUC, the SWRST bit is automatically set, keeping the USART in a reset condition. When set, the SWRST bit resets the URXIEx, UTXIEx, URXIFGx, OE, and FE bits and sets the UTXIFGx flag. The USPIEx bit is not altered by SWRST. Clearing SWRST releases the USART for operation. See also chapter *USART Module, I2C mode* for USART0 when reconfiguring from I^2C mode to SPI mode.

Note: Initializing or Re-Configuring the USART Module

The required USART initialization/re-configuration process is:

1) Set SWRST (`BIS.B #SWRST,&UxCTL`)

2) Initialize all USART registers with SWRST=1 (including UxCTL)

3) Enable USART module via the MEx SFRs (USPIEx)

4) Clear SWRST via software (`BIC.B #SWRST,&UxCTL`)

5) Enable interrupts (optional) via the IEx SFRs (URXIEx and/or UTXIEx)

Failure to follow this process may result in unpredictable USART behavior.

14.2.2 Master Mode

Figure 14–2. USART Master and External Slave

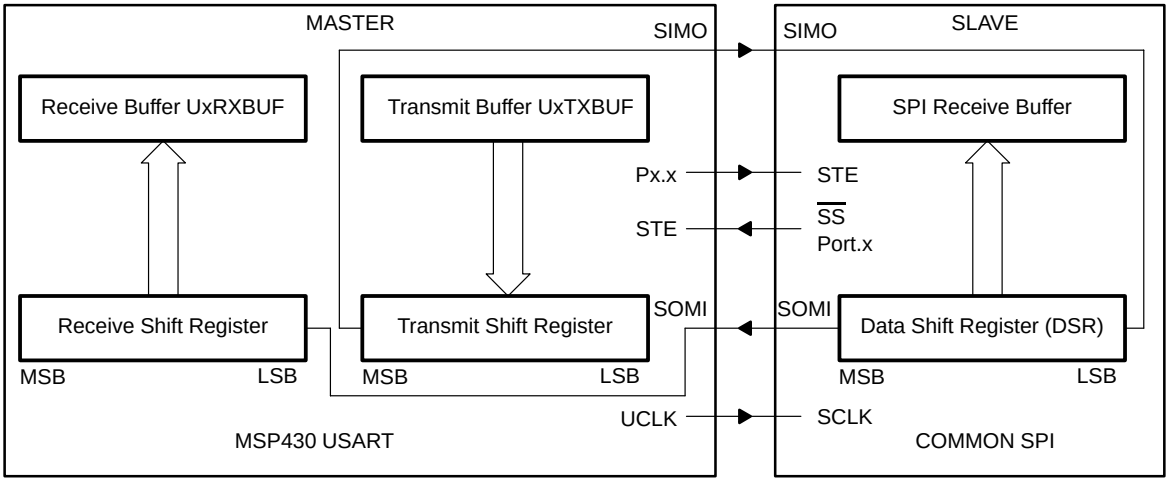


Figure 14–2 shows the USART as a master in both 3-pin and 4-pin configurations. The USART initiates data transfer when data is moved to the transmit data buffer UxTXBUF. The UxTXBUF data is moved to the TX shift register when the TX shift register is empty, initiating data transfer on SIMO starting with the most-significant bit. Data on SOMI is shifted into the receive shift register on the opposite clock edge, starting with the most-significant bit. When the character is received, the receive data is moved from the RX shift register to the received data buffer UxRXBUF and the receive interrupt flag, URXIFGx, is set, indicating the RX/TX operation is complete.

A set transmit interrupt flag, UTXIFGx, indicates that data has moved from UxTXBUF to the TX shift register and UxTXBUF is ready for new data. It does not indicate RX/TX completion.

To receive data into the USART in master mode, data must be written to UxTXBUF because receive and transmit operations operate concurrently.

Four-Pin SPI Master Mode

In 4-pin master mode, STE is used to prevent conflicts with another master. The master operates normally when STE is high. When STE is low:

- ❏ SIMO and UCLK are set to inputs and no longer drive the bus
- ❏ The error bit FE is set indicating a communication integrity violation to be handled by the user

A low STE signal does not reset the USART module. The STE input signal is not used in 3-pin master mode.

14.2.3 Slave Mode

Figure 14–3. USART Slave and External Master

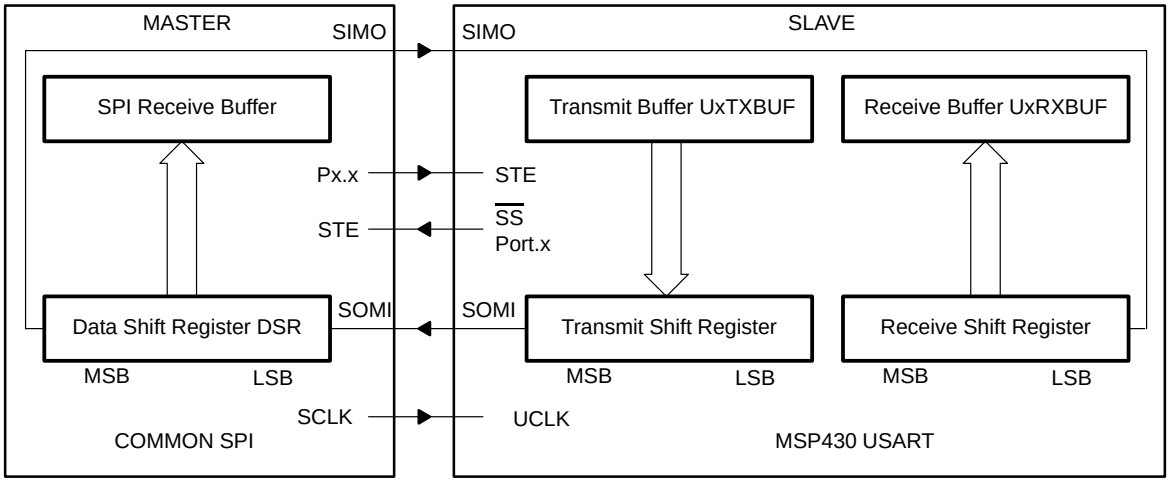


Figure 14–3 shows the USART as a slave in both 3-pin and 4-pin configurations. UCLK is used as the input for the SPI clock and must be supplied by the external master. The data-transfer rate is determined by this clock and not by the internal baud rate generator. Data written to UxTXBUF and moved to the TX shift register before the start of UCLK is transmitted on SOMI. Data on SIMO is shifted into the receive shift register on the opposite edge of UCLK and moved to UxRXBUF when the set number of bits are received. When data is moved from the RX shift register to UxRXBUF, the URXIFGx interrupt flag is set, indicating that data has been received. The overrun error bit, OE, is set when the previously received data is not read from UxRXBUF before new data is moved to UxRXBUF.

Four-Pin SPI Slave Mode

In 4-pin slave mode, STE is used by the slave to enable the transmit and receive operations and is provided by the SPI master. When STE is low, the slave operates normally. When STE is high:

- ❏ Any receive operation in progress on SIMO is halted
- ❏ SOMI is set to the input direction

A high STE signal does not reset the USART module. The STE input signal is not used in 3-pin slave mode.

14.2.4 SPI Enable

The SPI transmit/receive enable bit USPIEx enables or disables the USART in SPI mode. When USPIEx = 0, the USART stops operation after the current transfer completes, or immediately if no operation is active. A PUC or set SWRST bit disables the USART immediately and any active transfer is terminated.

Transmit Enable

When USPIEx = 0, any further write to UxTXBUF does not transmit. Data written to UxTXBUF will begin to transmit when USPIEx = 1 and the BRCLK source is active. Figure 14–4 and Figure 14–5 show the transmit enable state diagrams.

Figure 14–4. Master Mode Transmit Enable

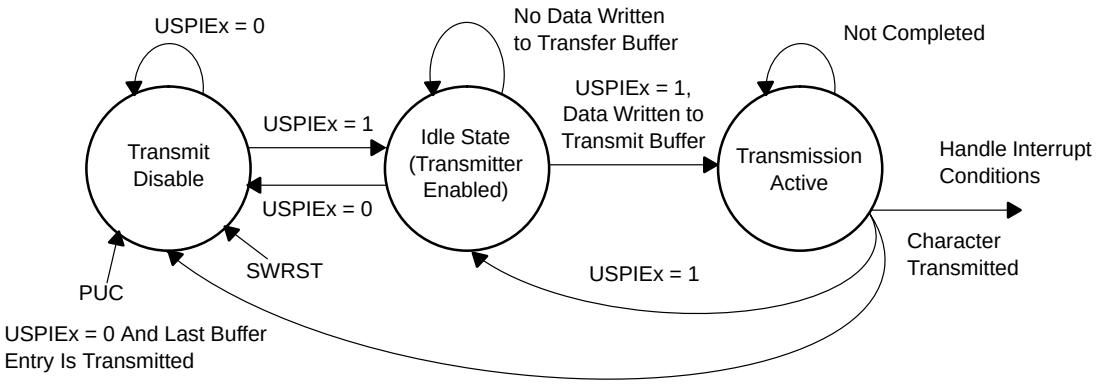
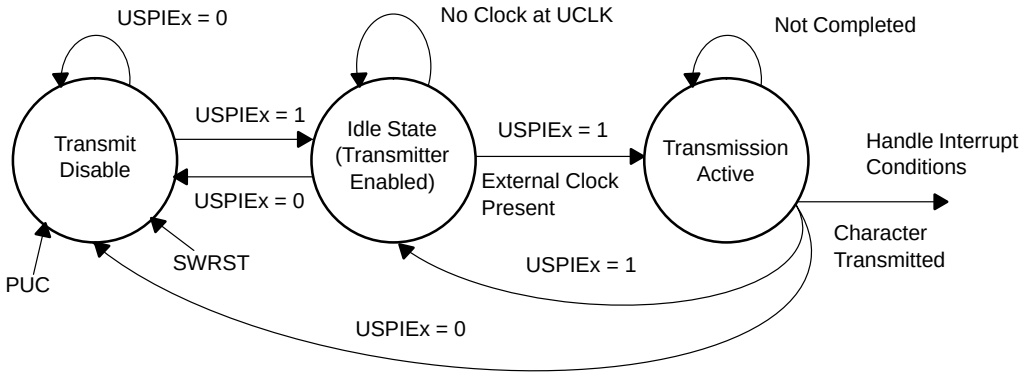


Figure 14–5. Slave Transmit Enable State Diagram



Receive Enable

The SPI receive enable state diagrams are shown in Figure 14–6 and Figure 14–7. When USPIEx = 0, UCLK is disabled from shifting data into the RX shift register.

Figure 14–6. SPI Master Receive-Enable State Diagram

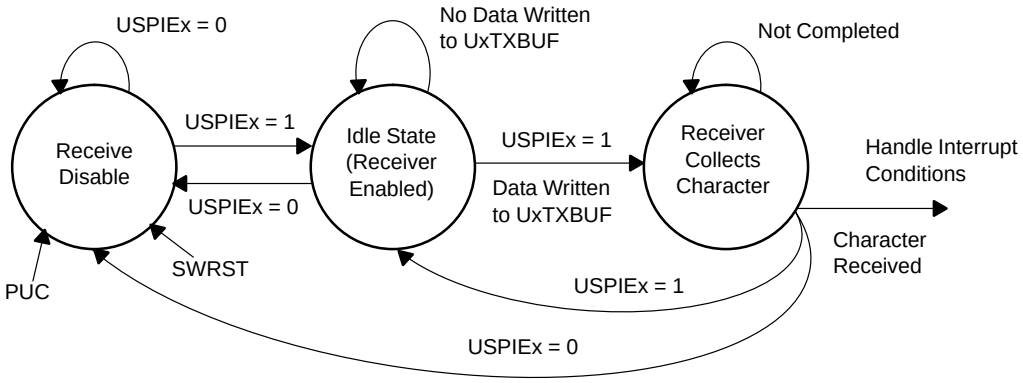
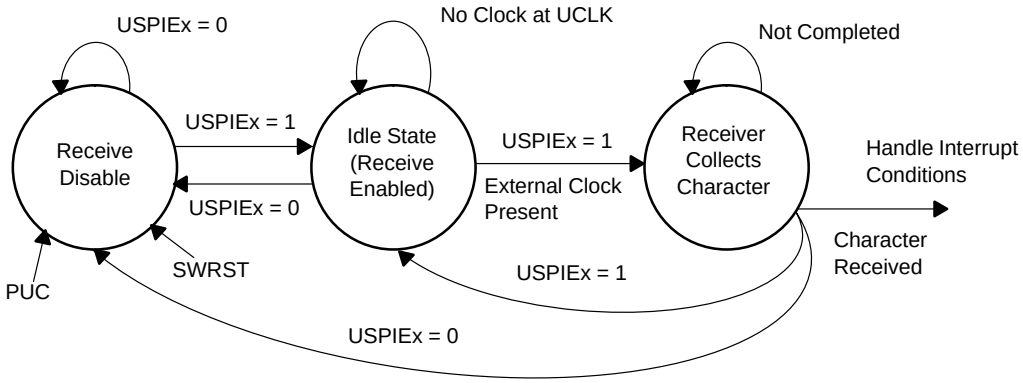


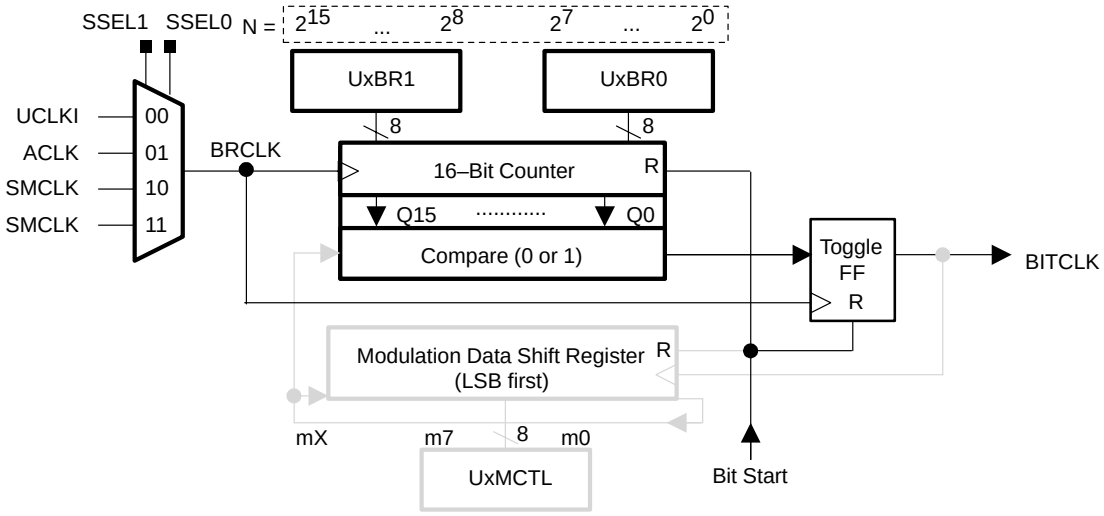
Figure 14–7. SPI Slave Receive-Enable State Diagram



14.2.5 Serial Clock Control

UCLK is provided by the master on the SPI bus. When MM = 1, BITCLK is provided by the USART baud rate generator on the UCLK pin as shown in Figure 14–8. When MM = 0, the USART clock is provided on the UCLK pin by the master and, the baud rate generator is not used and the SSELx bits are don't care. The SPI receiver and transmitter operate in parallel and use the same clock source for data transfer.

Figure 14–8. SPI Baud Rate Generator



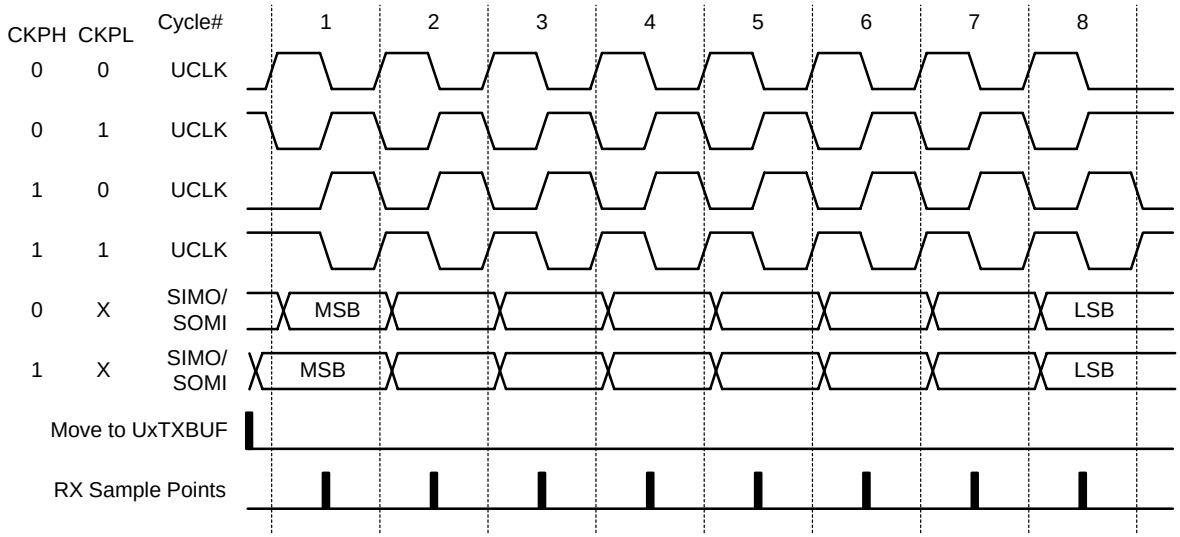
The 16-bit value of UxBR0+UxBR1 is the division factor of the USART clock source, BRCLK. The maximum baud rate that can be generated in master mode is BRCLK/2. The modulator in the USART baud rate generator is not used for SPI mode and is recommended to be set to 000h. The UCLK frequency is given by:

$$\text{Baud rate} = \frac{BRCLK}{UxBR} \text{ with UxBR= [UxBR1, UxBR0]}$$

Serial Clock Polarity and Phase

The polarity and phase of UCLK are independently configured via the CKPL and CKPH control bits of the USART. Timing for each case is shown in Figure 14–9.

Figure 14–9. USART SPI Timing



14.2.6 SPI Interrupts

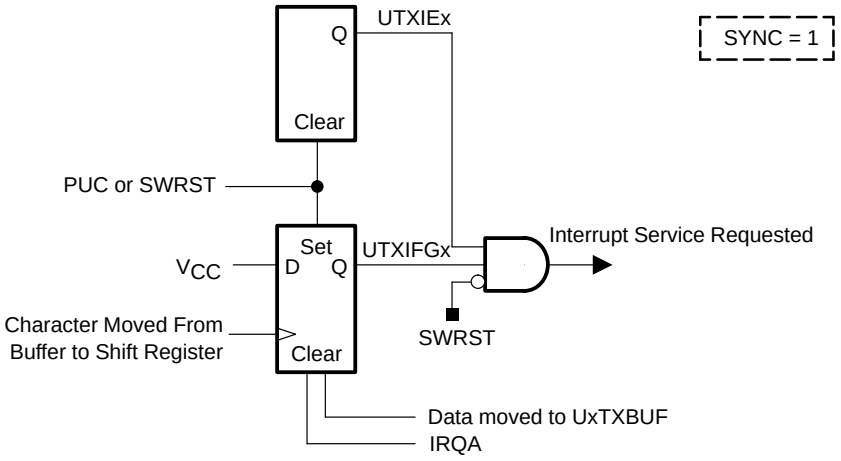
The USART has one interrupt vector for transmission and one interrupt vector for reception.

SPI Transmit Interrupt Operation

The UTXIFGx interrupt flag is set by the transmitter to indicate that UxTXBUF is ready to accept another character. An interrupt request is generated if UTXIEx and GIE are also set. UTXIFGx is automatically reset if the interrupt request is serviced or if a character is written to UxTXBUF.

UTXIFGx is set after a PUC or when SWRST = 1. UTXIEx is reset after a PUC or when SWRST = 1. The operation is shown is Figure 14–10.

Figure 14–10. Transmit Interrupt Operation



Note: Writing to UxTXBUF in SPI Mode

Data written to UxTXBUF when UTXIFGx = 0 and USPIEx = 1 may result in erroneous data transmission.

SPI Receive Interrupt Operation

The URXIFGx interrupt flag is set each time a character is received and loaded into UxRXBUF as shown in Figure 14–11 and Figure 14–12. An interrupt request is generated if URXIEx and GIE are also set. URXIFGx and URXIEx are reset by a system reset PUC signal or when SWRST = 1. URXIFGx is automatically reset if the pending interrupt is served or when UxRXBUF is read. The operation is shown in .

Figure 14–11. Receive Interrupt Operation

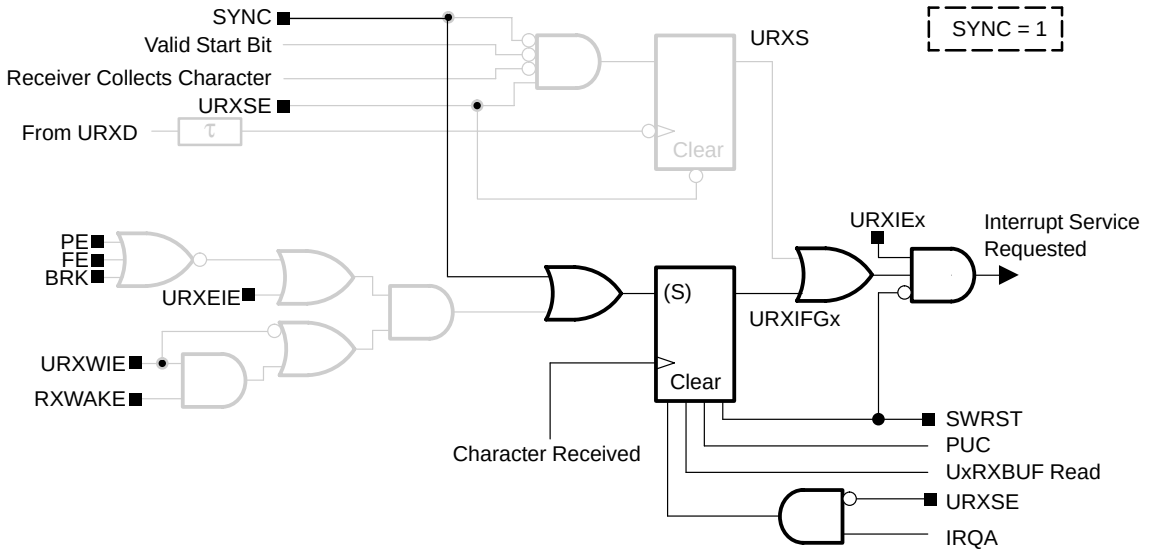
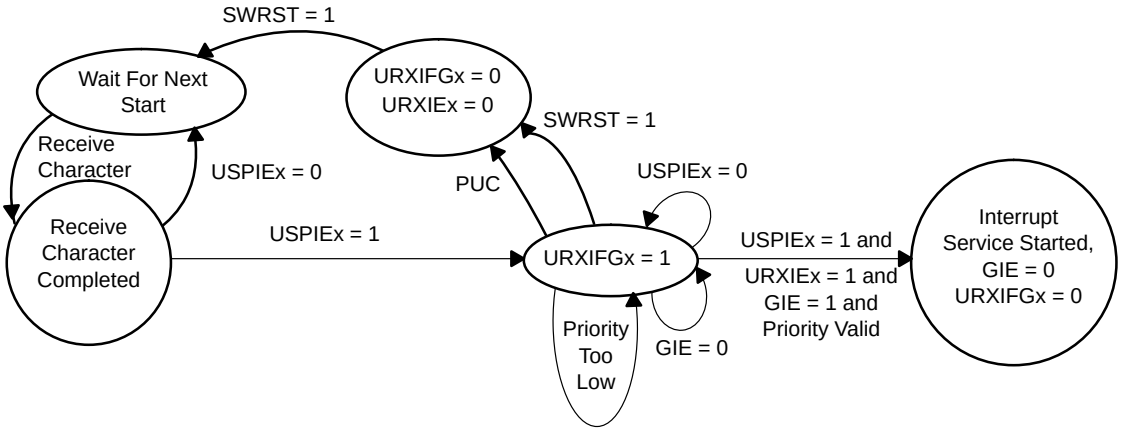


Figure 14–12. Receive Interrupt State Diagram



14.3 USART Registers: SPI Mode

The USART registers, shown in Table 14–1 and Table 14–2, are byte structured and should be accessed using byte instructions.

Table 14–1. USART0 Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USART control register	U0CTL	Read/write	070h	001h after PUC
Transmit control register	U0TCTL	Read/write	071h	001h after PUC
Receive control register	U0RCTL	Read/write	072h	000h after PUC
Modulation control register	U0MCTL	Read/write	073h	Unchanged
Baud rate control register 0	U0BR0	Read/write	074h	Unchanged
Baud rate control register 1	U0BR1	Read/write	075h	Unchanged
Receive buffer register	U0RXBUF	Read	076h	Unchanged
Transmit buffer register	U0TXBUF	Read/write	077h	Unchanged
SFR module enable register 1†	ME1	Read/write	004h	000h after PUC
SFR interrupt enable register 1†	IE1	Read/write	000h	000h after PUC
SFR interrupt flag register 1†	IFG1	Read/write	002h	082h after PUC

† Does not apply to MSP430x12xx devices. Refer to the register definitions for registers and bit positions for these devices.

Table 14–2. USART1 Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USART control register	U1CTL	Read/write	078h	001h after PUC
Transmit control register	U1TCTL	Read/write	079h	001h after PUC
Receive control register	U1RCTL	Read/write	07Ah	000h after PUC
Modulation control register	U1MCTL	Read/write	07Bh	Unchanged
Baud rate control register 0	U1BR0	Read/write	07Ch	Unchanged
Baud rate control register 1	U1BR1	Read/write	07Dh	Unchanged
Receive buffer register	U1RXBUF	Read	07Eh	Unchanged
Transmit buffer register	U1TXBUF	Read/write	07Fh	Unchanged
SFR module enable register 2	ME2	Read/write	005h	000h after PUC
SFR interrupt enable register 2	IE2	Read/write	001h	000h after PUC
SFR interrupt flag register 2	IFG2	Read/write	003h	020h after PUC

Note: Modifying the SFR bits

To avoid modifying control bits for other modules, it is recommended to set or clear the IEx and IFGx bits using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions.

UxCTL, USART Control Register

7	6	5	4	3	2	1	0
Unused	**Unused**	**I2C†**	**CHAR**	**LISTEN**	**SYNC**	**MM**	**SWRST**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–1

Unused	Bits 7–6	Unused
I2C†	Bit 5	I2C mode enable. This bit selects I2C or SPI operation when SYNC = 1. 0 SPI mode 1 I^2C mode
CHAR	Bit 4	Character length 0 7-bit data 1 8-bit data
LISTEN	Bit 3	Listen enable. The LISTEN bit selects the loopback mode 0 Disabled 1 Enabled. The transmit signal is internally fed back to the receiver
SYNC	Bit 2	Synchronous mode enable 0 UART mode 1 SPI mode
MM	Bit 1	Master mode 0 USART is slave 1 USART is master
SWRST	Bit 0	Software reset enable 0 Disabled. USART reset released for operation 1 Enabled. USART logic held in reset state

† Applies to USART0 on MSP430x15x and MSP430x16x devices only.

UxTCTL, USART Transmit Control Register

7	6	5	4	3	2	1	0
CKPH	**CKPL**	**SSELx**		**Unused**	**Unused**	**STC**	**TXEPT**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–1

CKPH Bit 7 Clock phase select. Controls the phase of UCLK.
- 0 Normal UCLK clocking scheme
- 1 UCLK is delayed by one half cycle

CKPL Bit 6 Clock polarity select
- 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.
- 1 The inactive level is high; data is output with the falling edge of UCLK; input data is latched with the rising edge of UCLK.

SSELx Bits 5-4 Source select. These bits select the BRCLK source clock.
- 00 External UCLK (valid for slave mode only)
- 01 ACLK (valid for master mode only)
- 10 SMCLK (valid for master mode only)
- 11 SMCLK (valid for master mode only)

Unused Bit 3 Unused

Unused Bit 2 Unused

STC Bit 1 Slave transmit control.
- 0 4-pin SPI mode: STE enabled.
- 1 3-pin SPI mode: STE disabled.

TXEPT Bit 0 Transmitter empty flag. The TXEPT flag is not used in slave mode.
- 0 Transmission active and/or data waiting in UxTXBUF
- 1 UxTXBUF and TX shift register are empty

UxRCTL, USART Receive Control Register

7	6	5	4	3	2	1	0
FE	**Unused**	**OE**	**Unused**	**Unused**	**Unused**	**Unused**	**Unused**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

FE	Bit 7	Framing error flag. This bit indicates a bus conflict when MM = 1 and STC = 0. FE is unused in slave mode. 0 No conflict detected 1 A negative edge occurred on STE, indicating bus conflict
Undefined	Bit 6	Unused
OE	Bit 5	Overrun error flag. This bit is set when a character is transferred into UxRXBUF before the previous character was read. OE is automatically reset when UxRXBUF is read, when SWRST = 1, or can be reset by software. 0 No error 1 Overrun error occurred
Unused	Bit 4	Unused
Unused	Bit 3	Unused
Unused	Bit 2	Unused
Unused	Bit 1	Unused
Unused	Bit 0	Unused

UxBR0, USART Baud Rate Control Register 0

7	6	5	4	3	2	1	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

UxBR1, USART Baud Rate Control Register 1

7	6	5	4	3	2	1	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

UxBRx The baud-rate generator uses the content of {UxBR1+UxBR0} to set the baud rate. The smallest division factor is two.

UxMCTL, USART Modulation Control Register

7	6	5	4	3	2	1	0
m7	m6	m5	m4	m3	m2	m1	m0
rw	rw	rw	rw	rw	rw	rw	rw

UxMCTLx Bits 7–0 The modulation control register is not used for SPI mode and should be set to 000h.

UxRXBUF, USART Receive Buffer Register

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
r	r	r	r	r	r	r	r

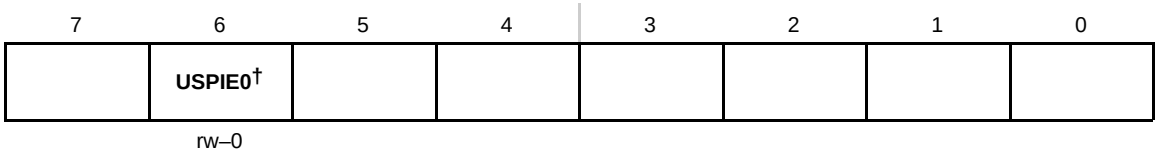
UxRXBUFx Bits 7–0 The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UxRXBUF resets the OE bit and URXIFGx flag. In 7-bit data mode, UxRXBUF is LSB justified and the MSB is always reset.

UxTXBUF, USART Transmit Buffer Register

7	6	5	4	3	2	1	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

UxTXBUFx Bits 7–0 The transmit data buffer is user accessible and contains current data to be transmitted. When seven-bit character-length is used, the data should be MSB justified before being moved into UxTXBUF. Data is transmitted MSB first. Writing to UxTXBUF clears UTXIFGx.

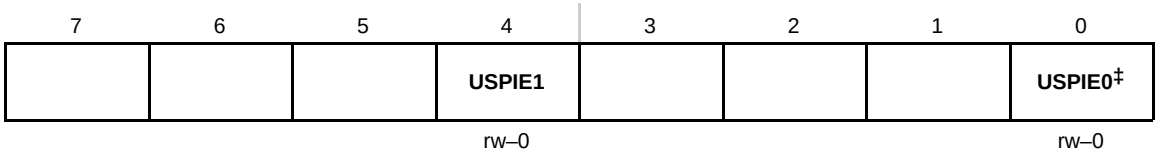
ME1, Module Enable Register 1



	Bit 7	This bit may be used by other modules. See device-specific datasheet.
USPIE0†	Bit 6	USART0 SPI enable. This bit enables the SPI mode for USART0. 0 Module not enabled 1 Module enabled
	Bits 5-0	These bits may be used by other modules. See device-specific datasheet.

† Does not apply to MSP430x12xx devices. See ME2 for the MSP430x12xx USART0 module enable bit

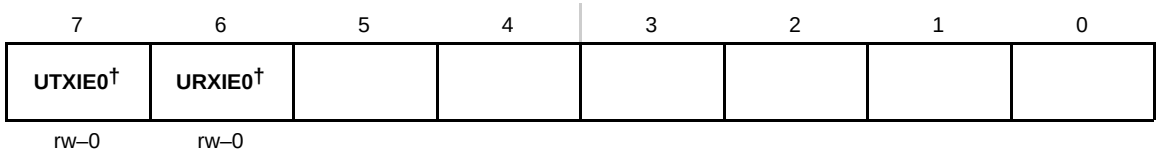
ME2, Module Enable Register 2



	Bits 7-5	These bits may be used by other modules. See device-specific datasheet.
USPIE1	Bit 4	USART1 SPI enable. This bit enables the SPI mode for USART1. 0 Module not enabled 1 Module enabled
	Bits 3-1	These bits may be used by other modules. See device-specific datasheet.
USPIE0‡	Bit 0	USART0 SPI enable. This bit enables the SPI mode for USART0. 0 Module not enabled 1 Module enabled

‡ MSP430x12xx devices only

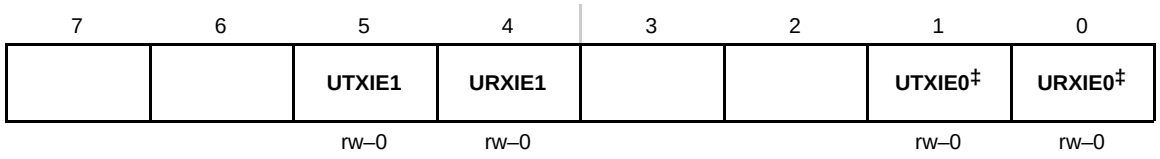
IE1, Interrupt Enable Register 1



UTXIE0†	Bit 7	USART0 transmit interrupt enable. This bit enables the UTXIFG0 interrupt. 0 Interrupt not enabled 1 Interrupt enabled
URXIE0†	Bit 6	USART0 receive interrupt enable. This bit enables the URXIFG0 interrupt. 0 Interrupt not enabled 1 Interrupt enabled
	Bits 5-0	These bits may be used by other modules. See device-specific datasheet.

† Does not apply to MSP430x12xx devices. See IE2 for the MSP430x12xx USART0 interrupt enable bits

IE2, Interrupt Enable Register 2



	Bits 7-6	These bits may be used by other modules. See device-specific datasheet.
UTXIE1	Bit 5	USART1 transmit interrupt enable. This bit enables the UTXIFG1 interrupt. 0 Interrupt not enabled 1 Interrupt enabled
URXIE1	Bit 4	USART1 receive interrupt enable. This bit enables the URXIFG1 interrupt. 0 Interrupt not enabled 1 Interrupt enabled
	Bits 3-2	These bits may be used by other modules. See device-specific datasheet.
UTXIE0‡	Bit 1	USART0 transmit interrupt enable. This bit enables the UTXIFG0 interrupt. 0 Interrupt not enabled 1 Interrupt enabled
URXIE0‡	Bit 0	USART0 receive interrupt enable. This bit enables the URXIFG0 interrupt for USART0. 0 Interrupt not enabled 1 Interrupt enabled

‡ MSP430x12xx devices only

IFG1, Interrupt Flag Register 1

7	6	5	4	3	2	1	0
UTXIFG0†	URXIFG0†						
rw–1	rw–0						

UTXIFG0† Bit 7 USART0 transmit interrupt flag. UTXIFG0 is set when U0TXBUF is empty.
0 No interrupt pending
1 Interrupt pending

URXIFG0† Bit 6 USART0 receive interrupt flag. URXIFG0 is set when U0RXBUF has received a complete character.
0 No interrupt pending
1 Interrupt pending

Bits 5-0 These bits may be used by other modules. See device-specific datasheet.

† Does not apply to MSP430x12xx devices. See IFG2 for the MSP430x12xx USART0 interrupt flag bits

IFG2, Interrupt Flag Register 2

7	6	5	4	3	2	1	0
		UTXIFG1	URXIFG1			UTXIFG0‡	URXIFG0‡
		rw–1	rw–0			rw–1	rw–0

Bits 7-6 These bits may be used by other modules. See device-specific datasheet.

UTXIFG1 Bit 5 USART1 transmit interrupt flag. UTXIFG1 is set when U1TXBUF is empty.
0 No interrupt pending
1 Interrupt pending

URXIFG1 Bit 4 USART1 receive interrupt flag. URXIFG1 is set when U1RXBUF has received a complete character.
0 No interrupt pending
1 Interrupt pending

Bits 3-2 These bits may be used by other modules. See device-specific datasheet.

UTXIFG0‡ Bit 1 USART0 transmit interrupt flag. UTXIFG0 is set when U0TXBUF is empty.
0 No interrupt pending
1 Interrupt pending

URXIFG0‡ Bit 0 USART0 receive interrupt flag. URXIFG0 is set when U0RXBUF has received a complete character.
0 No interrupt pending
1 Interrupt pending

‡ MSP430x12xx devices only

Chapter 15

USART Peripheral Interface, I2C Mode

The universal synchronous/asynchronous receive/transmit (USART) peripheral interface supports I2C communication in USART0 modules on the MSP430x15x and MSP430x16x devices. This chapter describes the I2C mode.

Topic **Page**

15.1 I2C Module Introduction

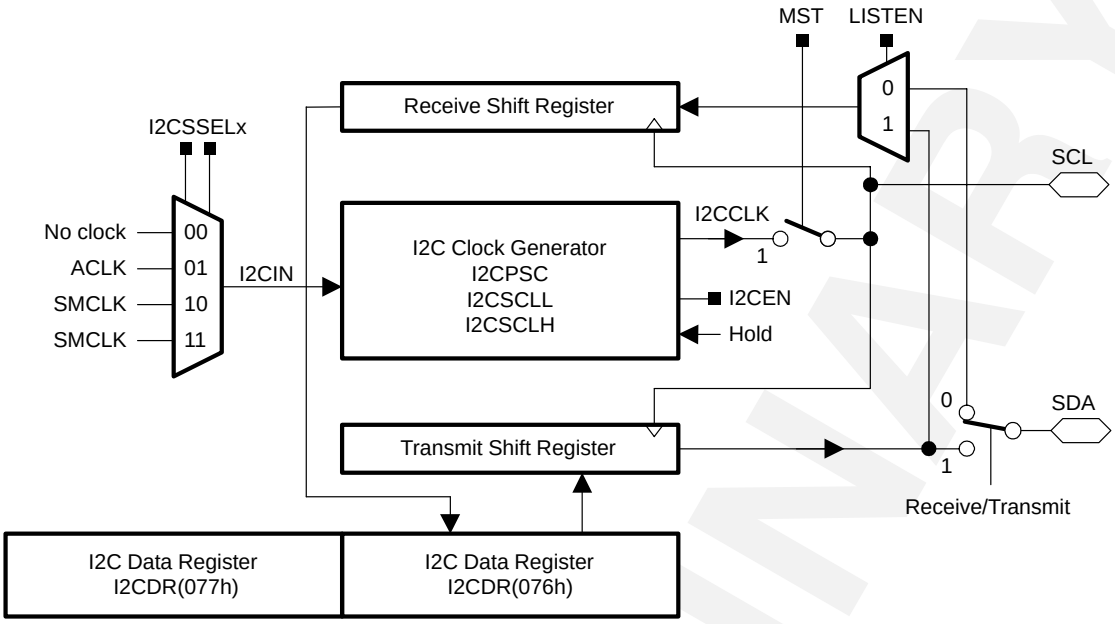
The inter-IC control (I2C) module provides an interface between the MSP430 and I2C-compatible devices connected by way of the two-wire I2C serial bus. External components attached to the I2C bus serially transmit and/or receive serial date to/from the USART through the 2-wire I2C interface.

The I2C module has the following features:

- Compliance to the Philips Semiconductor I2C specification v2.1
  - Bit/Byte format transfer
  - 7-bit and 10-bit device addressing modes
  - General call
  - Start/restart/stop
  - Multi-master transmitter/slave receiver mode
  - Multi-master receiver/slave transmitter mode
  - Combined master transmit/receive and receive/transmit mode
  - Standard mode up to100 kbps and fast mode up to 400 kbps support
- Built-in FIFO for buffered read and write
- Programmable clock generation
- 16-bit wide data access to maximize bus throughput
- Designed for low power
- Two DMA triggers
- Extensive interrupt capability
- Implemented on USART0 only

The I2C block diagram is shown in Figure 15–1.

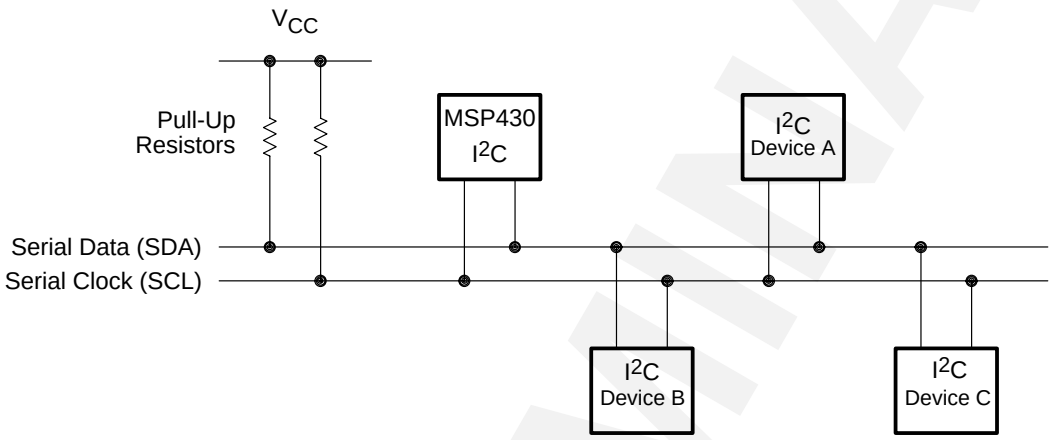
Figure 15–1. USART Block Diagram: I2C Mode



15.2 I2C Module Operation

The I2C module supports any slave or master I2C-compatible device. Figure 15–2 shows an example of an I2C bus. Each I2C device is recognized by a unique address and can operate as either a transmitter or a receiver. A device connected to the I2C bus can be considered as the master or the slave when performing data transfers. A master initiates a data transfer and generates the clock signal SCL. Any device addressed by a master is considered a slave.

Figure 15–2. I2C Bus Connection Diagram



I2C data is communicated using the serial data pin (SDA) and the serial clock pin (SCL). Both SDA and SCL are bidirectional, and must be connected to a positive supply voltage using a pull-up resistor.

Note: SDA and SCL Levels

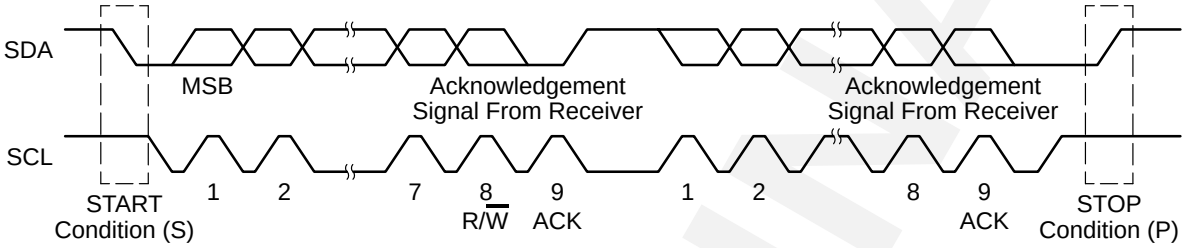
The MSP430 SDA and SCL pins must not be pulled up above the MSP430 V_{CC} level.

15.2.1 I2C Serial Data

One clock pulse is generated by the master device for each data bit transferred. The I2C module operates with byte data. Data is transferred most significant bit first as shown in Figure 15–3.

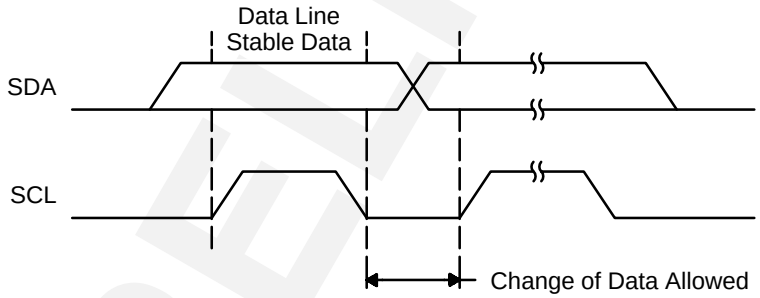
The first byte after a START condition consists of a 7-bit slave address and the R/$\overline{W}$ bit. When R/$\overline{W}$ = 0, the master transmits data to a slave. When R/$\overline{W}$ = 1, the master receives data from a slave. The ACK bit is sent from the receiver after each byte on the 9th SCL clock.

Figure 15–3. I2C Module Data Transfer



Data on SDA must be stable during the high period of SCL as shown in Figure 15–4. The high and low state of SDA can only change when SCL is low, otherwise start or stop conditions will be generated.

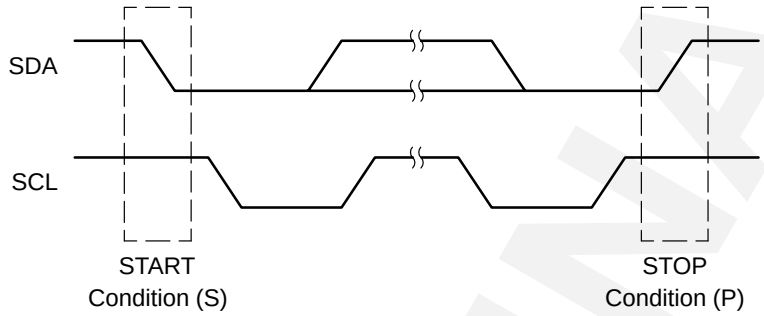
Figure 15–4. Bit Transfer on the I2C Bus



15.2.2 I2C START and STOP Conditions

START and STOP conditions are generated by the master and are shown in Figure 15–5. A START condition is a high-to-low transition on the SDA line while SCL is high. A STOP condition is a low-to-high transition on the SDA line while SCL is high. The busy bit, I2CBB, is set after a START and cleared after a STOP.

Figure 15–5. I2C Module START and STOP Conditions



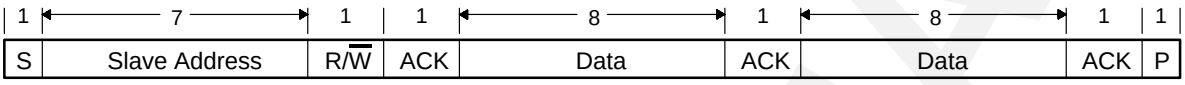
15.2.3 I2C Addressing Modes

The I2C module supports7-bit and 10-bit addressing modes.

7-Bit Addressing

In the 7-bit addressing format, shown in Figure 15–6, the first byte is the 7-bit slave address and the R/$\overline{W}$ bit. The ACK bit is sent from the receiver after each byte.

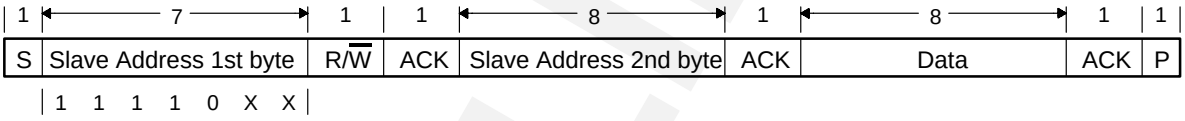
Figure 15–6. I2C Module 7-Bit Addressing Format



10-Bit Addressing

In the 10-bit addressing format, shown in Figure 15–7, the first byte is made up of 11110b plus the two MSBs of the 10-bit slave address and the R/$\overline{W}$ bit. The ACK bit is sent from the receiver after each byte. The next byte is the remaining 8 bits of the 10-bit slave address, followed by the ACK bit and the 8-bit data.

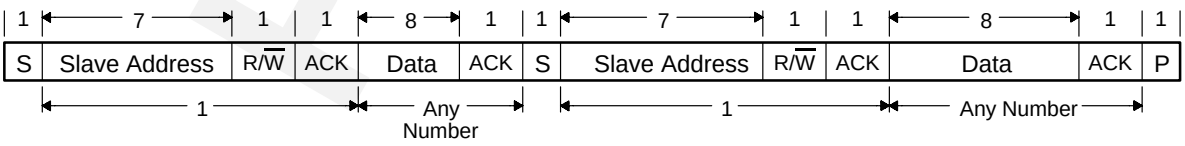
Figure 15–7. I2C Module 10-Bit Addressing Format



Repeated Start Conditions

The direction of data flow on SDA can be changed by the master, without first stopping a transfer, by issuing a repeated START condition. This is called a RESTART. After a RESTART is issued, the slave address is again sent out with the new data direction specified by the R/$\overline{W}$ bit. The RESTART condition is shown in Figure 15–8

Figure 15–8. I2C Module Addressing Format with Repeated START Condition



15.2.4 I2C Module Operating Modes

The I2C module operates in master transmitter, master receiver, slave transmitter, or slave receiver mode.

Master Mode

In master mode, transmit and receive operation is controlled with the I2CRM, I2CSTT, and I2CSTP bits as described in Table 15–1. The master receiver mode is entered by setting I2CTRX = 0 after a slave address byte and a set R/$\overline{W}$ bit has been transmitted. The master transmitter and master receiver modes are shown in Figure 15–9 and Figure 15–10.

SCL is held low when the intervention of the CPU is required after a byte has been transmitted.

Table 15–1. Master Operation

I2CRM	I2CSTP	I2CSTT	Condition Or Bus Activity
X	0	0	The I2C module is in master mode, but is idle. No start or stop condition is generated.
0	0	1	Setting I2CSTT initiates activity. I2CNDAT is used to determine length of transmission. A stop condition is not automatically generated after the I2CNDAT number of bytes have been transferred. Software must set I2CSTP to generate a stop condition at the end of transmission.
0	1	1	I2CNDAT is used to determine length of transmission. Setting I2CSTT initiates activity. A stop condition is automatically generated after I2CNDAT number of bytes have been transferred.
1	0	1	I2CNDAT is not used to determine length of transmission. Software must control the length of the transmission. Setting the I2CSTT bit initiates activity. Software must set the I2CSTP bit to initiate a stop condition and stop activity. This mode is useful if > 256 bytes are to be transferred.
0	1	0	Setting the I2CSTP bit generates a stop condition on the bus after I2CNDAT number of bytes have been sent, or immediately if I2CNDAT number of bytes have already been sent.
1	1	0	Setting the I2CSTP bit generates a stop condition on the bus after the current transmission completes, or immediately if no transmission is currently active.
1	1	1	Reserved, no bus activity.

I2C State Diagrams

Figure 15–9. Master Transmitter Mode, I2CWORD = 1

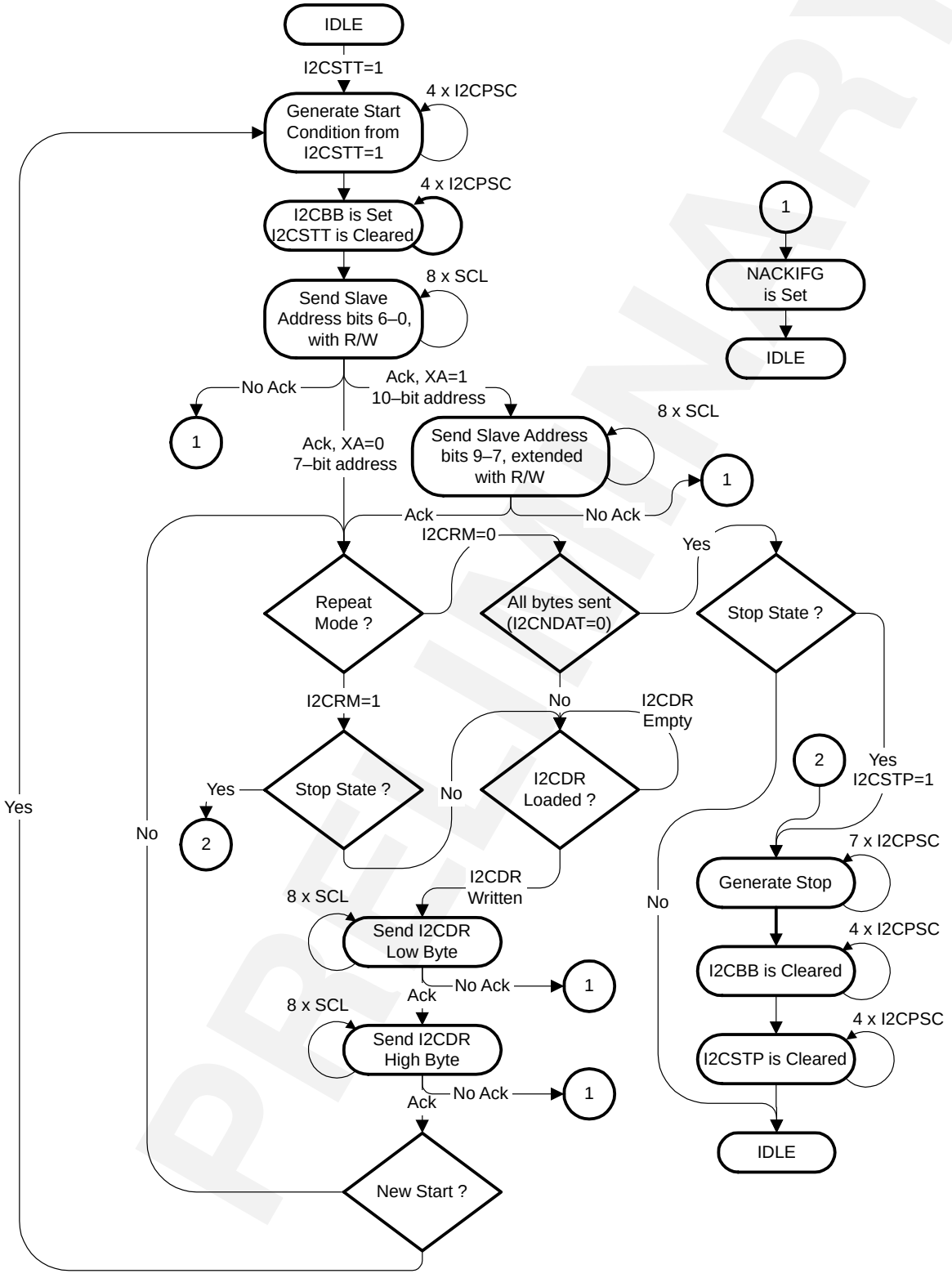
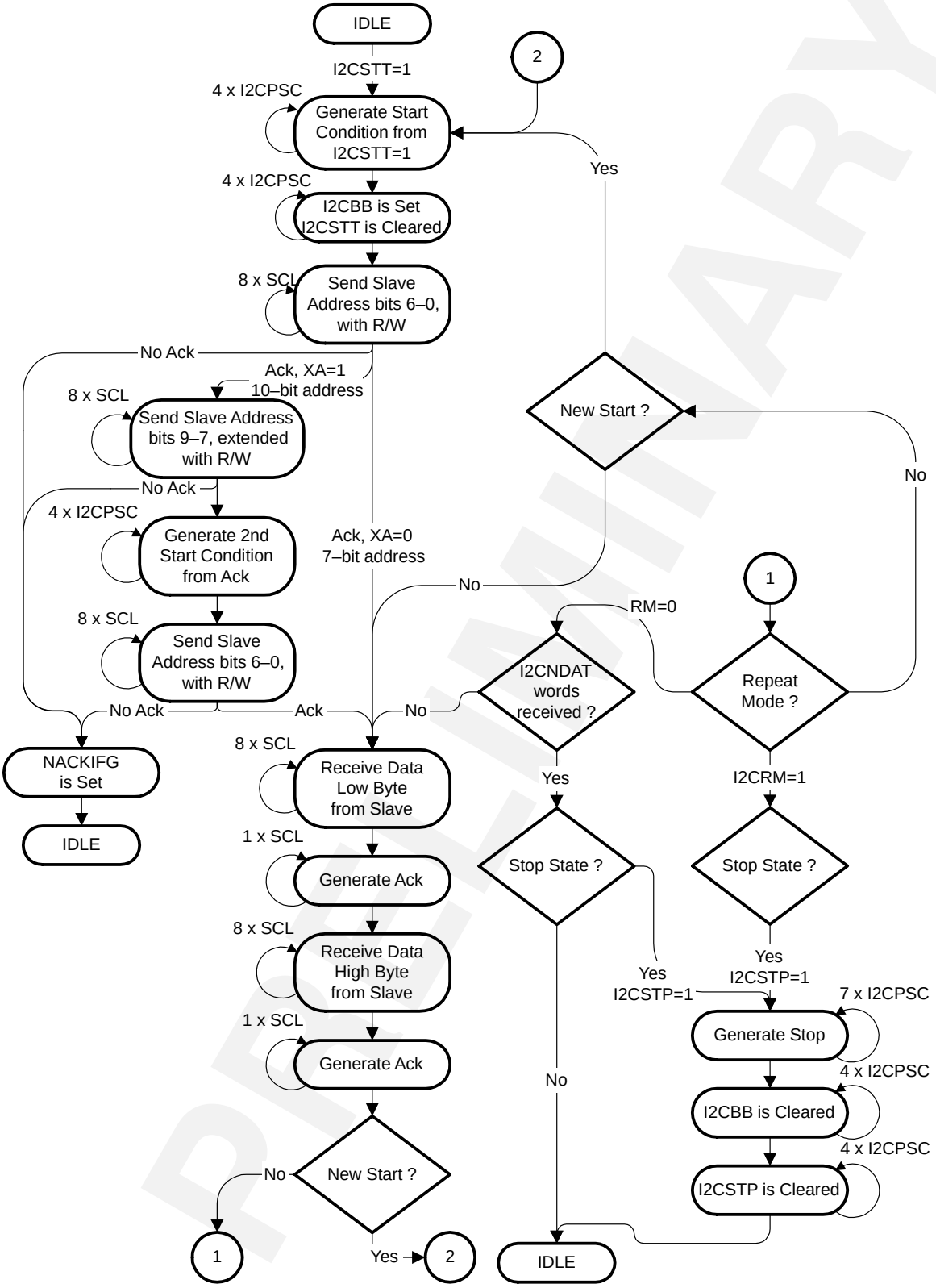


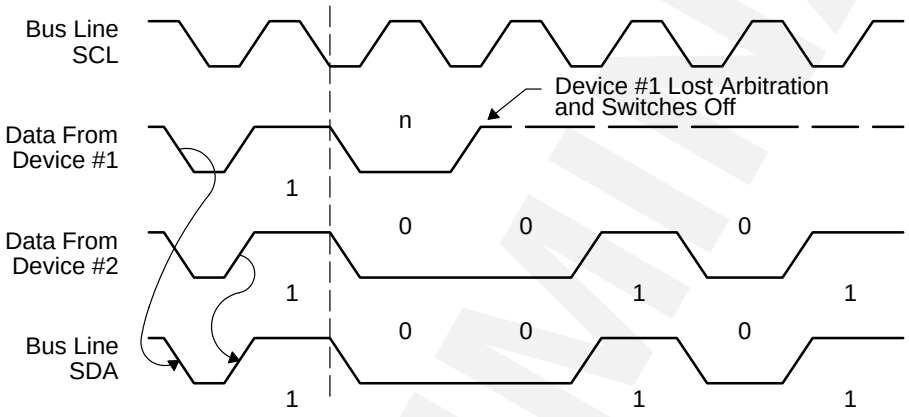
Figure 15–10. Master Receiver Mode, I2CWORD = 1



Arbitration

If two or more master transmitters simultaneously start a transmission on the bus, an arbitration procedure is invoked. Figure 15–11 illustrates the arbitration procedure between two devices. The arbitration procedure uses the data presented on SDA by the competing transmitters. The first master transmitter that generates a logic high is overruled by the opposing master generating a logic low. The arbitration procedure gives priority to the device that transmits the serial data stream with the lowest binary value. The master transmitter that lost arbitration switches to the slave receiver mode, and sets the arbitration lost flag ALIFG. If two or more devices send identical first bytes, arbitration continues on the subsequent bytes.

Figure 15–11. Arbitration Procedure Between Two Master Transmitters



If the arbitration procedure is in progress when a repeated START condition or STOP condition is transmitted on SDA, the master transmitters involved in arbitration must send the repeated START condition or STOP condition at the same position in the format frame. Arbitration is not allowed between:

- A repeated START condition and a data bit
- A STOP condition and a data bit
- A repeated START condition and a STOP condition

Slave Mode

In slave mode, transmit and receive operations are controlled automatically by the I2C module. The slave transmitter and slave receiver modes are shown in Figure 15–12 and Figure 15–13.

In slave receiver mode, serial data bits received on SDA are shifted in with the clock pulses that are generated by the master device. The slave device does not generate the clock, but it can hold SCL low if intervention of the CPU is required after a byte has been received.

Slave transmitter mode can only be entered from the slave receiver mode. The slave transmitter mode is entered if the slave address byte transmitted by the master is the same as its own address and a set R/$\overline{W}$ bit has been transmitted indicating a request to send data to the master. The slave transmitter shifts the serial data out on SDA with the clock pulses that are generated by the master device. The slave device does not generate the clock, but it can hold SCL low while intervention of the CPU is required after a byte has been transmitted.

Figure 15–12. Slave Transmitter, I2CWORD = 1

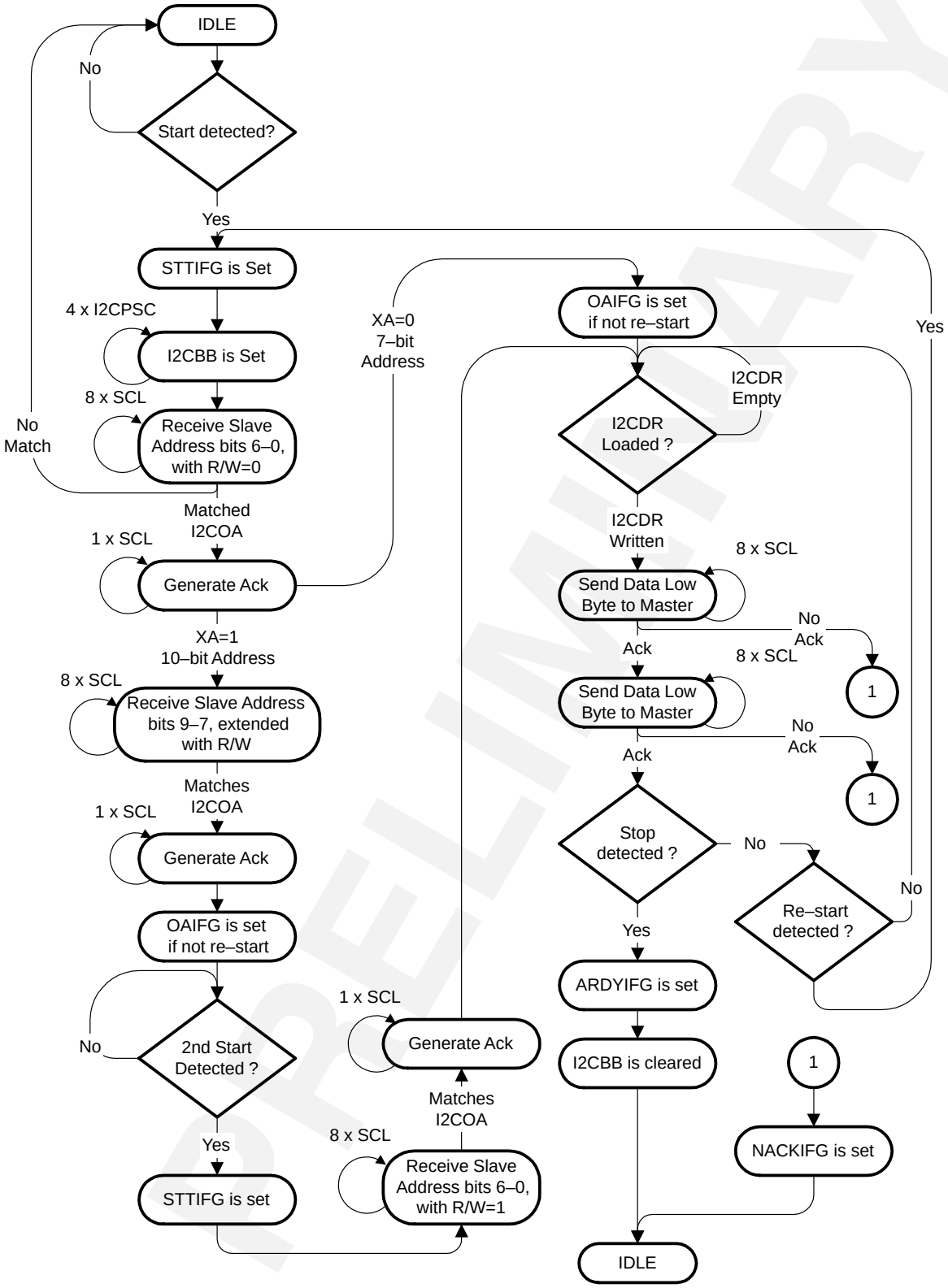
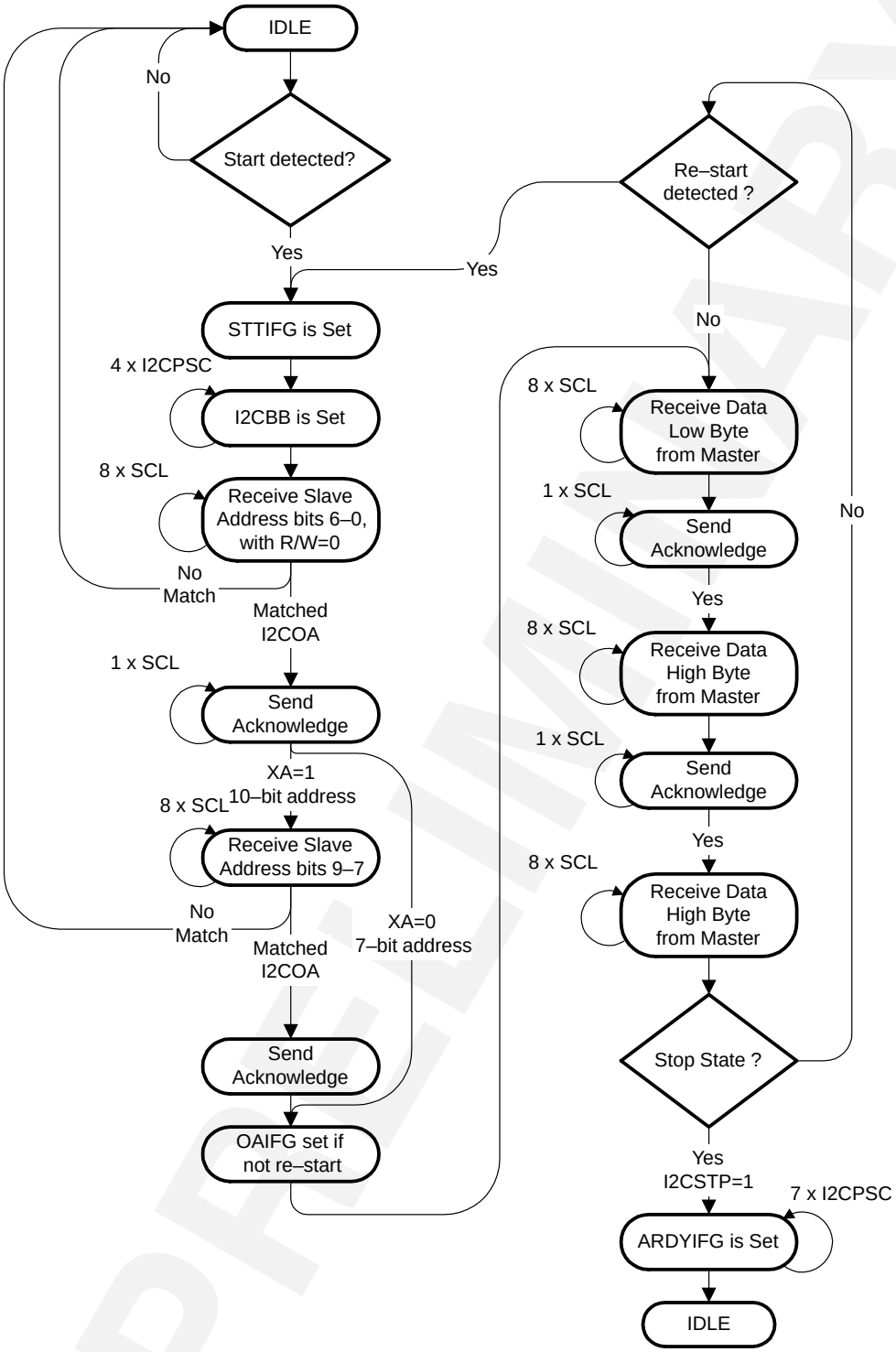


Figure 15–13. Slave Receiver, I2CWORD = 1



15.2.5 The I2C Data Register I2CDR

The I2CDR register can be accessed as an 8-bit or 16-bit register selected by the I2CWORD bit. The I2CDR register functions as described in Table 15–2.

Table 15–2. I2CDR Register Function

I2CWORD	I2CTRX	I2CDR Function
0	1	*Byte mode Transmit:* Only the low byte is used. The byte is double buffered. If a new byte is written before the previous byte has been transmitted, the new byte is held in a temporary buffer before being latched into the I2CDR low byte. TXRDYIFG is set when I2CDR is ready to be accessed.
0	0	*Byte mode receive:* Only the low byte is used. The byte is double buffered. If a new byte is received before the previous byte has been read, the new byte is held in a temporary buffer before being latched into the I2CDR low byte. RXRDYIFG is set when I2CDR is ready to be read.
1	1	*Word mode, transmit:* The low byte of the word is sent first, then the high byte. The register is double buffered. If a new word is written before the previous word has been transmitted, the new word is held in a temporary buffer before being latched into the I2CDR register. If the last data of a transmission is only one byte, then the high byte must be zero. TXRDYIFG is set when I2CDR is ready to be accessed.
1	0	*Word mode receive:* The low byte of the word was received first, then the high byte. The register is double buffered. If a new word is received before the previous word has been read, the new word is held in a temporary buffer before being latched into the I2CDR register. If the last reception was only one byte, then the high byte is 00h and the I2CSDB bit is set. RXRDYIFG is set when I2CDR is ready to be accessed.

Transmit Underflow

In master mode, underflow occurs when the transmit shift register and the transmit buffer are empty and I2CNDAT > 0. In slave mode, underflow occurs when the transmit shift register and the transmit buffer are empty and the external I2C master still requests data. When transmit underflow occurs, the I2CTXUDF bit is set. Writing data to I2CDR register or resetting I2CEN bit resets I2CTXUDF. I2CTXUDF is used in transmit mode only.

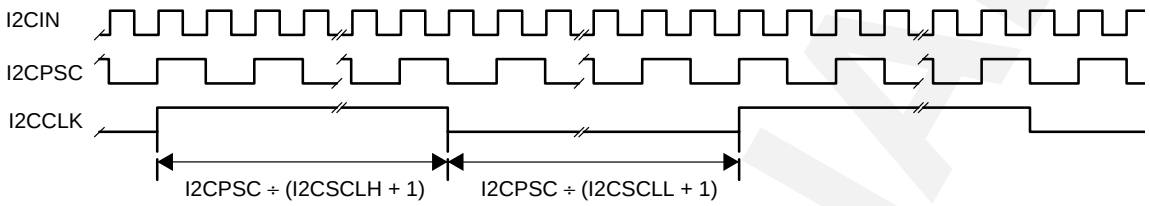
Receive Overrun

Receive overrun occurs when the receive shift register is full and the receive buffer is full. The I2CRXOVR bit is set when receive overrun occurs No data is lost because SCL is held low in this condition, which stops further bus activity. Reading the I2CDR register or resetting I2CEN bit resets I2CRXOVR bit. The I2CRXOVR bit is used in receive mode only.

15.2.6 I2C Clock Generation and Synchronization

The I2C module is operated with the clock source selected by the I2CSSELx bits. The prescaler, I2CPSC, and the I2CSCLH and I2CSCLL registers determine the frequency and duty cycle of the SCL clock signal for master mode as shown in Figure 15–14. The I2C module clock source must be at least 10x the SCL frequency in both master and slave modes.

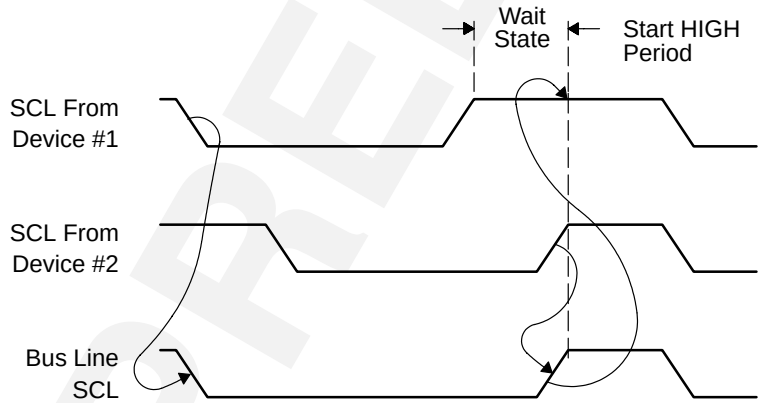
Figure 15–14. I2C Module SCL Generation



During the arbitration procedure the clocks from the different masters must be synchronized. A device that first generates a low period on SCL overrules the other devices forcing them to start their own low periods. SCL is then held low by the device with the longest low period. The other devices must wait for SCL to be released before starting their high periods. Figure 15–15 illustrates the clock synchronization.

A slow slave may pull SCL low to slow down a fast master. When this occurs, all other devices must enter the wait state. This allows a slow slave to slow down a fast master.

Figure 15–15. Synchronization of Two I2C Clock Generators During Arbitration



15.2.7 Using the I2C Module with Low Power Modes

The I2C module can be used with MSP430 low-power modes. When the internal clock source for the I2C module is present, the module operates normally regardless of the MSP430 operating mode.

When in slave mode, and when the internal clock source is not present, the I2C module can provide automatic start bit detection to wake the CPU. To enable this feature, the STTIE and GIE bits must be set to enable the STTIFG flag to interrupt the CPU. When the I2C module detects a start condition, the STTIFG flag is set, and the module holds the SCL line low, halting further bus activity. The interrupt service routine must then re-enable the I2C internal clock source for the I2C module to release the SCL line and allow bus activity to continue normally.

15.2.8 Using the I2C Module with the DMA Controller

The I2C module provides two trigger sources for the DMA controller. The RXRDYIFG flag can trigger a DMA transfer when new I2C data is received and the TXRDYIFG flag can trigger a DMA transfer when the I2C module needs new transmit data.

The TXDMAEN and RXDMAEN bits enable or disable the use of the DMA controller with the I2C module. When RXDMAEN = 1, the DMA controller can be used to transfer data from the I2C module after the I2C modules receives data. When RXDMAEN = 1, RXDYIE is automatically cleared.

When TXDMAEN = 1, the DMA controller can be used to transfer data to the I2C module for transmission. When TXDMAEN = 1, the TXRDYIE is automatically cleared.

See the *DMA Controller* chapter for more details on the DMA controller.

15.2.9 Configuring the USART for I2C Operation

The I2C controller is part of the USART peripheral. Individual bit definitions when using USART0 in I2C mode are different from that in SPI or UART mode. The default value for the U0CTL register is the UART mode and the register contains the following bits:

To select SPI or I2C operation the SYNC bit must be set. Setting the SYNC bit with SWRST = 1 selects SPI mode. Setting the I2C bit when SYNC = 1 selects the I2C mode. The SYNC and I2C bits can be set together in a single instruction to select the I2C mode for USART0.

After module initialization, the I2C module is ready for transmit or receive operation. Clearing I2CEN releases the I2C module for operation.

Note: Re-Configuring the I2C Module for UART or SPI operation

The required USART re-configuration process is:

1) Clear the I2C bit (`BIC.B #I2C,&U0CTL`)

2) Clear U0CTL and set SWRST bit (`MOV.B #SWRST,&U0CTL`)

3) Initialize all USART registers with SWRST = 1 (including UxCTL)

4) Enable USART module via the MEx SFRs (URXEx and/or UTXEx)

5) Clear SWRST via software (`BIC.B #SWRST,&UxCTL`)

6) Enable interrupts (optional) via the IEx SFRs (URXIEx and/or UTXIEx)

Failure to follow this process may result in unpredictable USART behavior.

Note: Re-Configuring the I2C Module for Different I2C Conditions

The required I2C re-configuration process is:

1) Clear the I2CEN bit (`BIC.B #I2CEN,&U0CTL`)

2) Re-configure the I2C module with I2CEN = 0

3) Set I2CEN via software (`BIS.B #I2CEN,&U0CTL`)

Failure to follow this process may result in unpredictable USART behavior.

I2C Module Reset

After a PUC, the USART module is configured in the UART mode, with SWRST = 1. In I2C mode, setting I2CEN = 0 has the following effects:

- ❑ I2C communication stops
- ❑ SDA and SCL are high impedance
- ❑ I2CTCTL, bits 3-0 are cleared and bits 7-4 are unchanged
- ❑ I2CDCTL and I2CDR register is cleared
- ❑ Transmit and receive shift registers are cleared
- ❑ U0CTL, I2CNDAT, I2CPSC, I2CSCLL, I2CSCLH registers are unchanged
- ❑ I2COA, I2CSA, I2CIE, I2CIFG, and I2CIV registers are unchanged

15.2.10 I2C Interrupts

The I2C module has one interrupt vector for eight interrupt flags. Each interrupt flag has its own interrupt enable bit. When an interrupt is enabled, and the GIE bit is set, the interrupt flag will generate an interrupt request. The I2C interrupt events are:

Interrupt Flag	Interrupt Condition
ALIFG	Arbitration-lost. Arbitration can be lost when two or more transmitters start a transmission simultaneously, or when the software attempts to initiate an I2C transfer while I2CBB = 1. The ALIFG flag is set when arbitration has been lost. When ALIFG is set the MST and I2CSTP bits are cleared and the I2C controller becomes a slave receiver.
NACKIFG	No-acknowledge interrupt. This flag is set when the master does not receive an acknowledge from the slave. NACKIFG is used in master mode only.
OAIFG	Own-address interrupt. The OAIFG interrupt flag is set when another master has addressed the I2C module. OAIFG is used in slave mode only.
ARDYIFG	Register-access-ready interrupt. This flag is set when the previously programmed transfer has completed and the status bits have been updated. This interrupt is used to notify the CPU that the I2C registers are ready to be accessed.
RXRDYIFG	Receive ready interrupt/status. this flag is set when the I2C module has received new data. RXRDYIFG is automatically cleared when I2CDR is read and the receive buffer is empty. A receiver overrun is indicated if bit I2CRXOVR = 1. RXRDYIFG is used in receive mode only.
TXRDYIFG	Transmit ready interrupt/status. the I2CDR register is ready for new transmit data when I2CNDAT > 0 OR I2CRM = 1 (master transmit mode) or when another master is requesting data (slave transmit mode). TXRDYIFG is automatically cleared when I2CDR and the transmit buffer are full. A transmit underflow is indicated if I2CTXUDF = 1. Unused in receive mode.
GCIFG	General call interrupt. This flag is set when the I2C module received the general call address (00h). GCIFG is used in receive mode only.
STTIFG	Start condition detected interrupt. This flag is set when the I2C module detects a start condition while in slave mode. This allows the MSP430 to be in a low power mode with the I2C clock source inactive until a master initiates I2C communication. STTIFG is used in slave mode only.

I2CIV, Interrupt Vector Generator

The I2C interrupt flags are prioritized and combined to source a single interrupt vector. The interrupt vector register I2CIV is used to determine which flag requested an interrupt. The highest priority enabled interrupt generates a number in the I2CIV register. This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled I2C interrupts do not affect the I2CIV value.

Any access, read or write, of the I2CIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.

I2CIV Software Example

The following software example shows the recommended use of I2CIV. The I2CIV value is added to the PC to automatically jump to the appropriate routine.

```
I2C_ISR
          ADD    &I2CIV, PC   ; Add offset to jump table
          RETI                ; Vector 0: No interrupt
          JMP    ALIFG_ISR    ; Vector 2: ALIFG
          JMP    NACKIFG_ISR  ; Vector 4: NACKIFG
          JMP    OAIFG_ISR    ; Vector 6: OAIFG
          JMP    ARDYIFG_ISR  ; Vector 8: ARDYIFG
          JMP    RXRDYIFG_ISR ; Vector 10: RXRDYIFG
          JMP    TXRDYIFG_ISR ; Vector 12: TXRDYIFG
          JMP    GCIFG_ISR    ; Vector 14: GCIFG

STTIFG_ISR                    ; Vector 16
          ...                 ; Task starts here
          RETI                ; Return
ALIFG_ISR                     ; Vector 2
          ...                 ; Task starts here
          RETI                ; Return
NACKIFG_ISR                   ; Vector 4
          ...                 ; Task starts here
          RETI                ; Return
OAIFG_ISR                     ; Vector 6
          ...                 ; Task starts here
          RETI                ; Return
ARDYIFG_ISR                   ; Vector 8
          ...                 ; Task starts here
          RETI                ; Return
RXRDYIFG_ISR                  ; Vector 10
          ...                 ; Task starts here
          RETI                ; Return
TXRDYIFG_ISR                  ; Vector 12
          ...                 ; Task starts here
          RETI                ; Return
GCIFG_ISR                     ; Vector 14
          ...                 ; Task starts here
          RETI                ; Return
```

15.3 I2C Module Registers

The I2C module registers and respective addresses are listed in Table 15–3.

Table 15–3. I2C Registers

Register	Short Form	Register Type	Address	Initial State
I2C interrupt enable	I2CIE	Read/write	050h	Reset with PUC
I2C interrupt flag	I2CIFG	Read/write	051h	Reset with PUC
I2C data count	I2CNDAT	Read/write	052h	Reset with PUC
USART control	U0CTL	Read/write	070h	Reset with PUC
I2C transfer control	I2CTCTL	Read/write	071h	Reset with PUC
I2C data control	I2CDCTL	Read only	072h	Reset with PUC
I2C prescaler	I2CPSC	Read/write	073h	Reset with PUC
I2C SCL high	I2CSCLH	Read/write	074h	Reset with PUC
I2C SCL low	I2CSCLL	Read/write	075h	Reset with PUC
I2C data	I2CDR	Read/write	076h	Reset with PUC
I2C own address	I2COA	Read/write	0118h	Reset with PUC
I2C slave address	I2CSA	Read/write	011Ah	Reset with PUC
I2C interrupt vector	I2CIV	Read only	011Ch	Reset with PUC

U0CTL, USART0 Control Register-I2C Mode

7	6	5	4	3	2	1	0
RXDMAEN	**TXDMAEN**	**I2C**	**XA**	**LISTEN**	**SYNC**	**MST**	**I2CEN**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–1

RXDMAEN Bit 7 Receive DMA enable. This bit enables the DMA controller to be used to transfer data from the I2C module after the I2C modules receives data. When RXDMAEN = 1, RXRDYIE is automatically cleared.
0 Disabled
1 Enabled

TXDMAEN Bit 6 Transmit DMA enable. This bit enables the DMA controller to be used to provide data to the I2C module for transmission. When TXDMAEN = 1, TXRDYIE, is automatically cleared.
0 Disabled.
1 Enabled.

I2C Bit 5 I2C mode enable. This bit select I2C or SPI operation when SYNC = 1.
0 SPI mode
1 I2C mode

XA Bit 4 Extended Addressing
0 7-bit addressing
1 10-bit addressing

LISTEN Bit 3 Listen. This bit selects loopback mode. LISTEN is only valid when MST = 1 and I2CTRX = 1 (master transmitter).
0 Normal mode
1 SDA is internally fed back to the receiver (loopback).

SYNC Bit 2 Synchronous mode enable
0 UART mode
1 SPI or I2C mode

MST Bit 1 Master. This bit selects master or slave mode. The MST bit is automatically cleared when arbitration is lost.
0 Slave mode
1 Master mode

I2CEN Bit 0 I2C enable. The bit enables or disables the I2C module. The initial condition for this bit is set, and SWRST function for UART or SPI. When the I2C and SYNC bits are first set after a PUC, this bit becomes I2CEN function and is automatically cleared.
0 I2C operation is disabled
1 I2C operation is enabled

I2CTCTL, I2C Transmit Control Register

7	6	5	4	3	2	1	0
I2CWORD	I2CRM	I2CSSELx		I2CTRX	I2CSTB	I2CSTP	I2CSTT
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

Modifiable only when I2CEN = 0

I2CWORD Bit 7 I2C word mode. Selects byte or word mode for the I2C data register.
- 0 Byte mode
- 1 Word mode

I2CRM Bit 6 I2C repeat mode
- 0 I2CNDAT defines the number of bytes transmitted.
- 1 Number of byte transmitted is controlled by software. I2CNDAT is unused.

I2CSSELx Bits 5–4 I2C clock source select. When MST = 1 and arbitration is lost, the external SCL signal is automatically used.
- 00 No clock – I2C module is inactive
- 01 ACLK
- 10 SMCLK
- 11 SMCLK

I2CTRX Bit 3 I2C transmit. This bit selects the transmit or receive function for the I2C controller when MST = 1. When MST = 0, the R/W bit of the address byte defines the data direction and I2CTRX reflects the direction of the SDA pin.
- 0 Receive mode. Data is received on the SDA pin.
- 1 Transmit mode. Data transmitted on the SDA pin.

I2CSTB Bit 2 Start byte. Setting the I2CSTB bit when MST = 1 initiates a start byte.
- 0: No action
- 1: Send START condition, start byte (03h), but no stop condition.

I2CSTP Bit 1 Stop bit. This bit is used to generate STOP condition. After the stop condition, the I2CSTP is automatically cleared.
- 0: No action
- 1: Send STOP condition

I2CSTT Bit 0 Start bit. This bit is used to generate a START condition. After the start condition the I2CSTT is automatically cleared.
- 0: No action
- 1: Send START condition

I2CDCTL, I2C Data Control Register

7	6	5	4	3	2	1	0
Unused	**Unused**	**Unused**	**I2C SCLLOW**	**I2XSBD**	**I2XTXUDF**	**I2CRXOVR**	**I2CBB**
r0	r0	r0	r–0	r–0	r–0	r–0	r–0

Unused Bits 7–5 Unused. Always read as 0.

I2C SCLLOW Bit 4 I2C SCL low. This bit indicates if a slave is holding the SCL line low while the MSP430 is the master and is unused in slave mode.
0 SCL is not being held low
1 SCL is being held low

I2CSBD Bit 3 I2C single byte data. This bit indicates if the receive register holds a word or a byte. I2CSBD is valid only when I2CWORD = 1.
0 A complete word was received
1 Only the lower byte in I2CDR is valid

I2CTXUDF Bit 2 I2C transmit underflow
0 No underflow occurred
1 Transmit underflow occurred

I2CRXOVR Bit 1 I2C receive overrun
0 No receive overrun occurred
1 Receiver overrun occurred

I2CBB Bit 0 I2C busy bit. A start condition sets I2CBB to 1. I2CBB is reset by a stop condition or when I2CEN=0.
0 Not busy
1 Busy

I2CDR, I2C Data Register

15	14	13	12	11	10	9	8
I2C Data High Byte (second to be tranceived)							
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

7	6	5	4	3	2	1	0
I2C Data Low Byte, (first to be tranceived)							
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

I2CDRx Bits 15–0 I2C data.

I2CNDAT, I2C Transfer Byte Count Register

7	6	5	4	3	2	1	0
I2CNDATx							
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

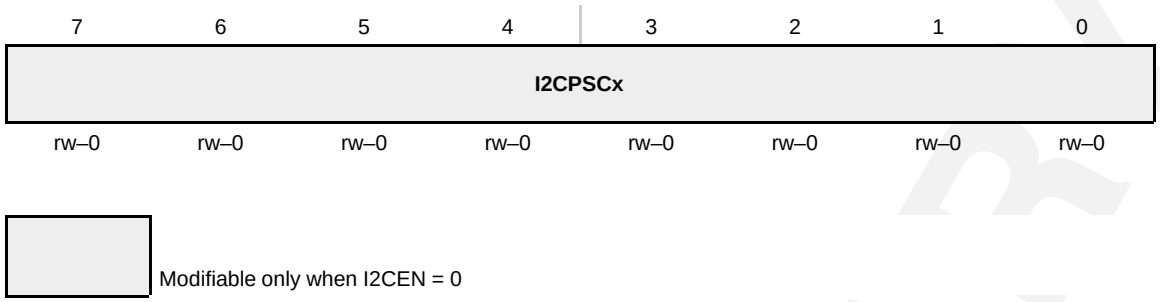
I2CNDATx Bits 7–0 I2C number of bytes. These bits support automatic data byte counting. In word mode, I2CNDATx must be an even value.

Write to register Number of bytes
Read from register, I2CBB = 1 Number of bytes remaining in transfer
Read from register, I2CBB = 0 Number of bytes to be transferred

Note: I2CNDAT Register

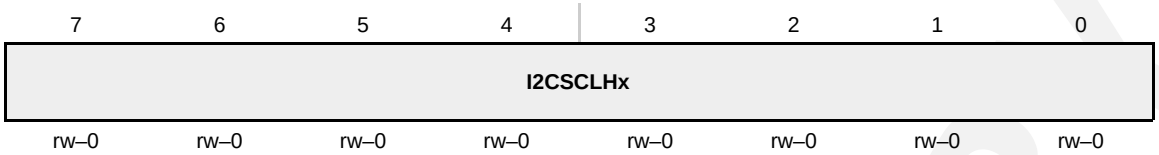
Do not change the I2CNDAT register while I2CBB = 1 and I2CRM = 0.

I2CPSC, I2C Clock Prescaler Register



I2CPSCx Bits 7–0 I2C clock prescaler. The I2C clock input is divided by the I2CPSCx value to produce the internal I2C clock frequency. The division rate is I2CPSC+1.
000h Divide by 1
001h Divide by 2
:
0FFh Divide by 256

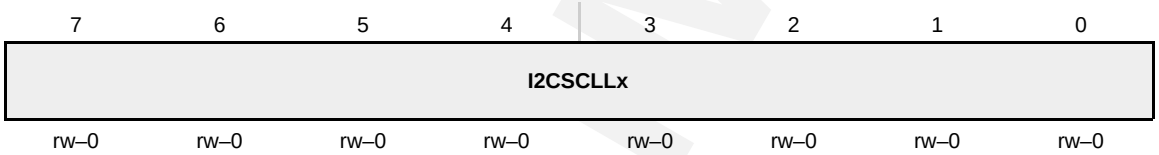
I2CSCLH, I2C Shift Clock High Register



Modifiable only when I2CEN = 0

I2CSCLHx Bits 7–0 — I2C shift clock high. These bits define the high period of SCL when the I2C controller is in master mode. The SCL high period is (I2CSCLH+2) x I2CPSC.
000h N/A
001h N/A
002h N/A
003h SCL high period = 5 x I2CPSC
:
0FFh SCL high period = 257 x I2CPSC

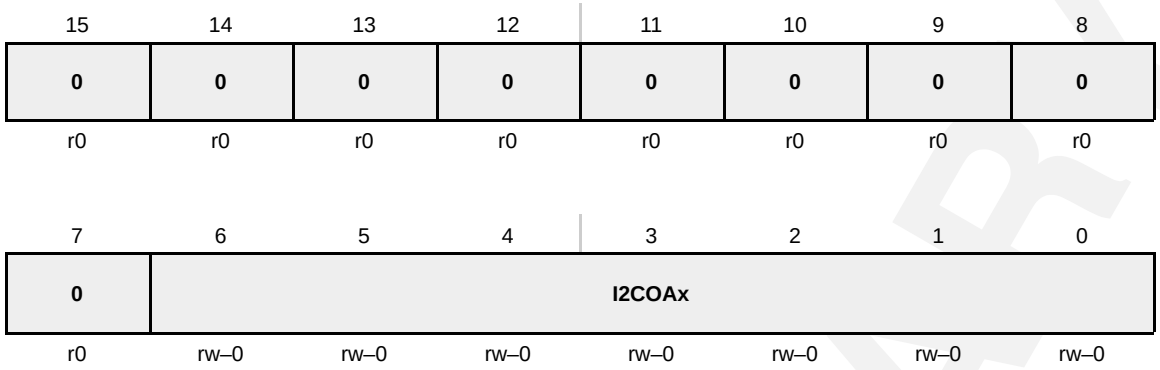
I2CSCLL, Shift Clock Low Register



Modifiable only when I2CEN = 0

I2CSCLLx Bits 7–0 — I2C shift clock low. These bits define the low period of SCL when the I2C controller is in master mode. The SCL low period is (I2CSCLL+2) x I2CPSC
000h N/A
001h N/A
002h N/A
003h SCL low period = 5 x I2CPSC
:
0FFh SCL low period = 257 x I2CPSC

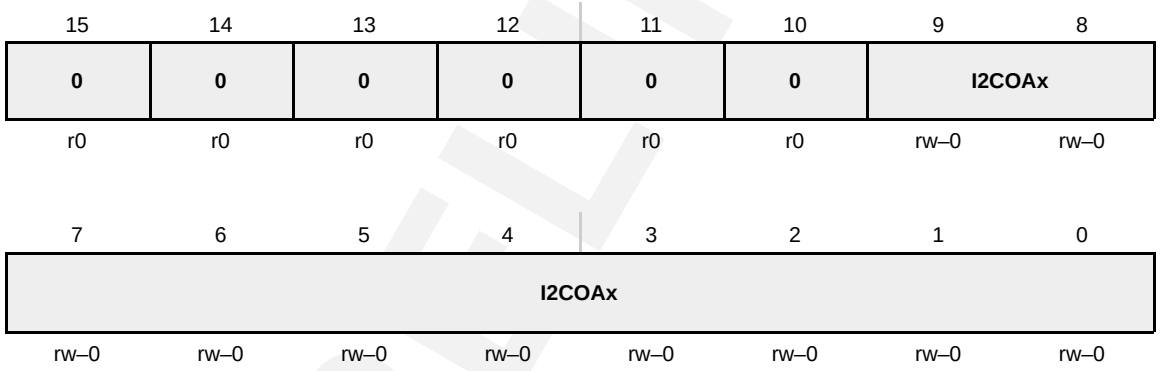
I2COA, I2C Own Address Register, 7-Bit Addressing Mode



Modifiable only when I2CEN = 0

I2COAx Bits 15-0 I2C own address. The I2COA register contains the local address of the MSP430 I2C controller. The I2COA register is right-justified. Bit 6 is the MSB. Bits 15-7 are always 0.

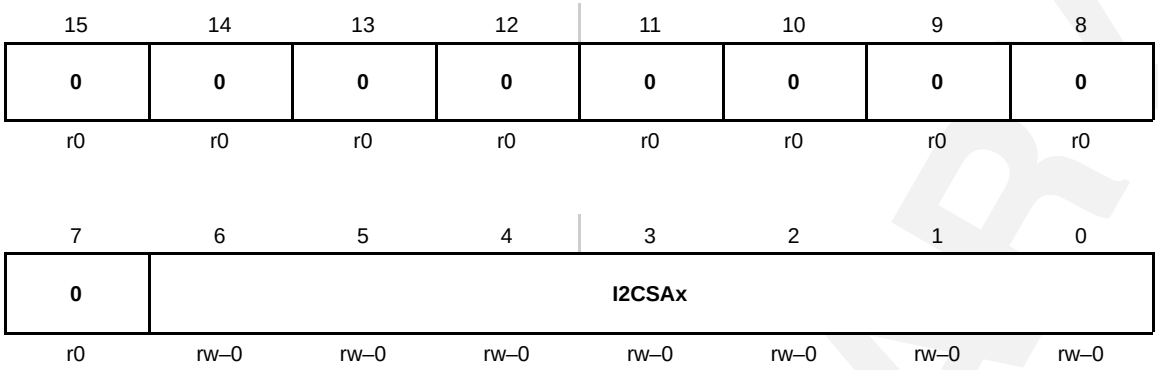
I2COA, I2C Own Address Register, 10-Bit Addressing Mode



Modifiable only when I2CEN = 0

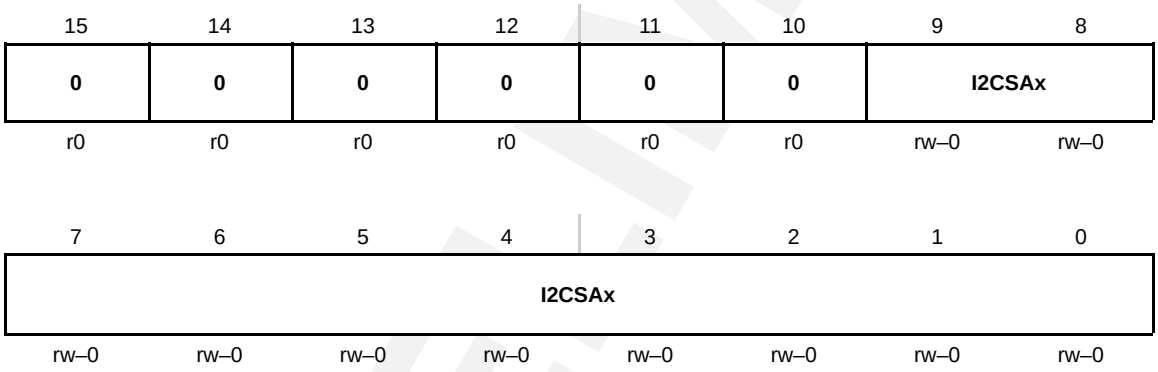
I2COAx Bits 15-0 I2C own address. The I2COA register contains the local address of the MSP430 I2C controller. The I2COA register is right-justified. Bit 9 is the MSB. Bits 15-10 are always 0.

I2CSA, I2C Slave Address Register, 7-Bit Addressing Mode



I2CSAx Bits 15-0 I2C slave address. The I2CSA register contains the slave address of the external device to be addressed by the MSP430. It is only used in master mode. The I2CSA register is right-justified. Bit 6 is the MSB. Bits 15-7 are always 0.

I2CSA, I2C Slave Address Register, 10-Bit Addressing Mode



I2CSAx Bits 15-0 I2C slave address. The 2CSA register contains the slave address of the external device to be addressed by the MSP430. It is only used in master mode. The I2CSA register is right-justified. Bit 9 is the MSB. Bits 15-10 are always 0.

I2CIE, I2C Interrupt Enable Register

7	6	5	4	3	2	1	0
STTIE	**GCIE**	**TXRDYIE**	**RXRDYIE**	**ARDYIE**	**OAIE**	**NACKIE**	**ALIE**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

STTIE Bit 7 Start detect interrupt enable
0 Interrupt enabled
1 Interrupt disabled

GCIE Bit 6 General call interrupt enable
0 Interrupt enabled
1 Interrupt disabled

TXRDYIE Bit 5 Transmit ready interrupt enable. TXRDYIE is automatically cleared when TXDMAEN = 1.
0 Interrupt enabled
1 Interrupt disabled

RXRDYIE Bit 4 Receive ready interrupt enable RXRDYIE is automatically cleared when RXDMAEN = 1.
0 Interrupt enabled
1 Interrupt disabled

ARDYIE Bit 3 Access ready interrupt enable
0 Interrupt enabled
1 Interrupt disabled

OAIE Bit 2 Own address interrupt enable
0 Interrupt enabled
1 Interrupt disabled

NACKIE Bit 1 No acknowledge interrupt enable
0 Interrupt enabled
1 Interrupt disabled

ALIE Bit 0 Arbitration lost interrupt enable
0 Interrupt enabled
1 Interrupt disabled

I2CIFG, I2C Interrupt Flag Register

7	6	5	4	3	2	1	0
STTIFG	GCIFG	TXRDYIFG	RXRDYIFG	ARDYIFG	OAIFG	NACKIFG	ALIFG
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

STTIFG Bit 7 Start detect interrupt flag
0 No interrupt pending
1 Interrupt pending

GCIFG Bit 6 General call interrupt flag
0 No interrupt pending
1 Interrupt pending

TXRDYIFG Bit 5 Transmit ready interrupt flag
0 No interrupt pending
1 Interrupt pending

RXRDYIFG Bit 4 Receive ready interrupt flag
0 No interrupt pending
1 Interrupt pending

ARDYIFG Bit 3 Access ready interrupt flag. The ARDYIFG set conditions are:

Mode	Applicable Bits	ARDYIFG Set Conditions
Master transmit	I2CSTP = 1 I2CRM = 0	I2CNDAT = 0 and all data transmitted.
	I2CRM = 1	Last byte of data sent after I2CSTP has been set.
Master receive	I2CSTP = 1 I2CRM = 0	I2CNDAT = 0 and receive buffer empty.
	I2CRM = 1	Last byte of data received and receive buffer empty after I2CSTP has been set.
Slave transmit	–	Stop condition received.
Slave receive	–	Stop condition received and receive buffer empty.

OAIFG Bit 2 Own address interrupt flag
0 No interrupt pending
1 Interrupt pending

NACKIFG Bit 1 No acknowledge interrupt flag
0 No interrupt pending
1 Interrupt pending

ALIFG Bit 0 Arbitration lost interrupt flag
0 No interrupt pending
1 Interrupt pending

I2CIV, I2C Interrupt Vector Register

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**I2CIVx**				**0**
r0	r0	r0	r–0	r–0	r–0	r–0	r0

I2CIVx Bits 15-0 I2C interrupt vector value

I2CIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No interrupt pending	–	
002h	Arbitration lost	ALIFG	Highest
004h	No acknowledgement	NACKIFG	
006h	Own address	OAIFG	
008h	Register access ready	ARDYIFG	
00Ah	Receive data ready	RXRDYIFG	
00Ch	Transmit data ready	TXRDYIFG	
00Eh	General call	GCIFG	
010h	Start condition received	STTIFG	Lowest

Chapter 16

Comparator_A

Comparator_A is an analog voltage comparator. This chapter describes Comparator_A. Comparator_A is implemented in MSP430x11x1, MSP430x12x, MSP430x13x, MSP430x14x, MSP430x15x and MSP430x16x devices.

Topic **Page**

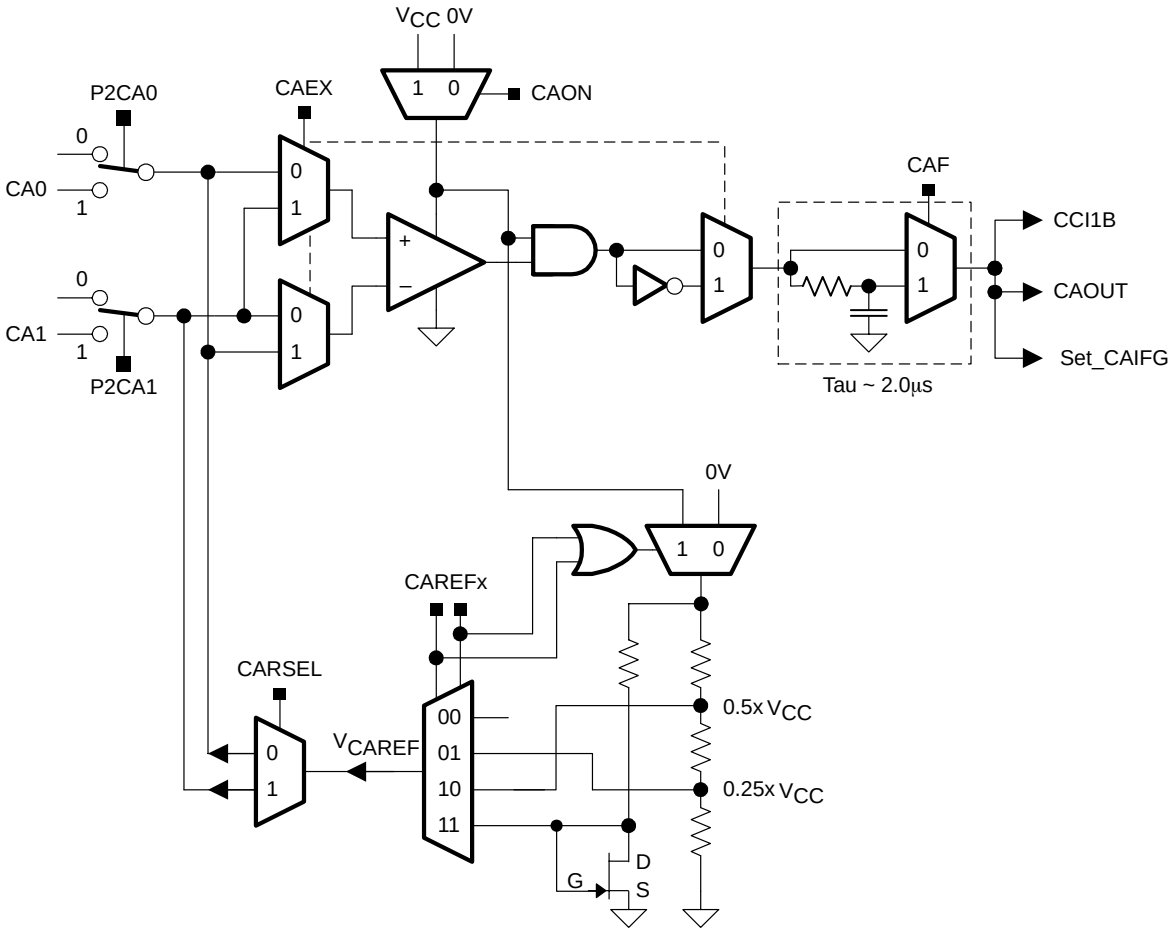
16.1 Comparator_A Introduction

The comparator_A module supports precision slope analog-to-digital conversions, supply voltage supervision, and monitoring of external analog signals. The Comparator_A block diagram is shown in Figure 16–1.

Features of Comparator_A include:

- ❏ Inverting and non-inverting terminal input multiplexer
- ❏ Software selectable RC-filter for the comparator output
- ❏ Output provided to Timer_A capture input
- ❏ Software control of the port input buffer
- ❏ Interrupt capability
- ❏ Selectable reference voltage generator
- ❏ Comparator and reference generator can be powered down

Figure 16–1. Comparator_A Block Diagram



16.2 Comparator_A Operation

The comparator_A module is configured with user software. The setup and operation of comparator_A is discussed in the following sections.

16.2.1 Comparator

The comparator compares the analog voltages at the + and – input terminals. If the + terminal is more positive than the – terminal, the comparator output CAOUT is high. The comparator can be switched on or off using control bit CAON. The comparator should be switched off when not in use to reduce current consumption. When the comparator is switched off, the CAOUT is always low.

16.2.2 Input Analog Switches

The analog input switches connect or disconnect the two comparator input terminals to associated port pins using the P2CAx bits. Both comparator terminal inputs can be controlled individually. The P2CAx bits allow:

- ❏ Application of an external signal to the + and – terminals of the comparator
- ❏ Routing of an internal reference voltage to an associated output port pin

Internally, the input switch is constructed as a T-switch to suppress distortion in the signal path.

Note: Comparator Input Connection

When the comparator is on, the input terminals should be connected to a signal, power, or ground. Otherwise, floating levels may cause unexpected interrupts and increased current consumption.

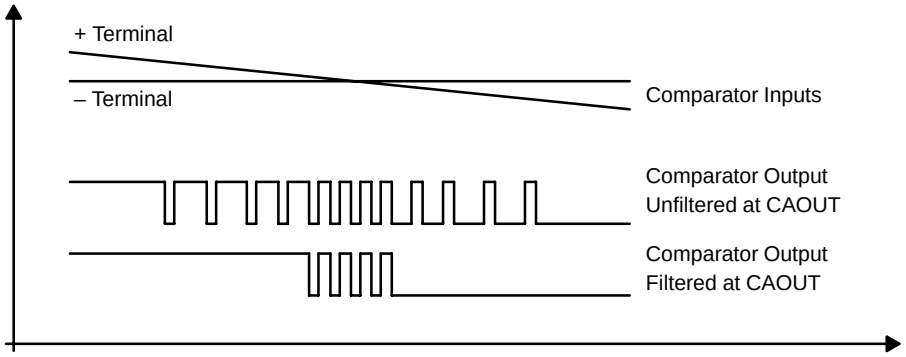
The CAEX bit controls the input multiplexer, exchanging which input signals are connected to the comparator's + and – terminals. Additionally, when the comparator terminals are exchanged, the output signal from the comparator is inverted. This allows the user to determine or compensate for the comparator input offset voltage.

16.2.3 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 on-chip RC-filter.

Any comparator output oscillates 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 as shown in Figure 16–2. The comparator output oscillation reduces accuracy and resolution of the comparison result. Selecting the output filter can reduce errors associated with comparator oscillation.

Figure 16–2. RC-Filter Response at the Output of the Comparator



16.2.4 Voltage Reference Generator

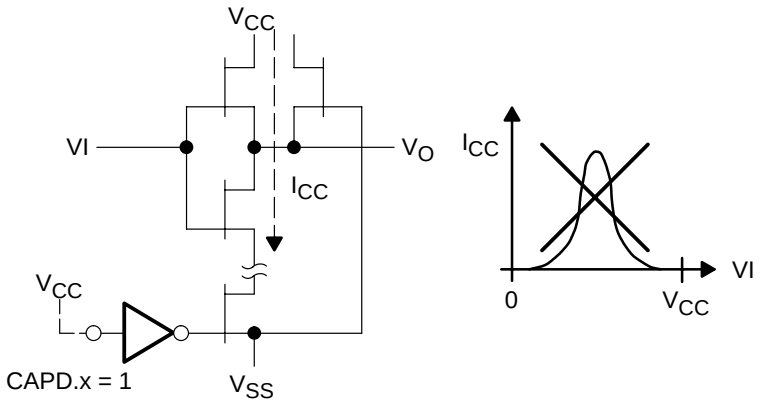
The voltage reference generator is used to generate V_{CAREF}, which can be applied to either comparator input terminal. The CAREFx bits control the output of the voltage generator. The CARSEL bit selects the comparator terminal to which V_{CAREF} is applied. If external signals are applied to both comparator input terminals, the internal reference generator should be turned off to reduce current consumption. The voltage reference generator can generate a fraction of the device's V_{CC} or a fixed transistor threshold voltage of ~ 0.55 V. The transistor threshold voltage has a tolerance and temperature coefficient specified in the device-specific datasheet.

16.2.5 Comparator_A, Port Disable Register CAPD

The comparator input and output functions are multiplexed with the associated I/O port pins, which are digital CMOS gates. When analog signals are applied to digital CMOS gates, parasitic current can flow from V_{CC} to GND. This parasitic current occurs if the input voltage is near the transition level of the gate. Disabling the port pin buffer eliminates the parasitic current flow and therefore reduces overall current consumption.

The CAPDx bits, when set, disable the corresponding P2 input buffer as shown in Figure 16–3. When current consumption is critical, any P2 pin connected to analog signals should be disabled with their associated CAPDx bit.

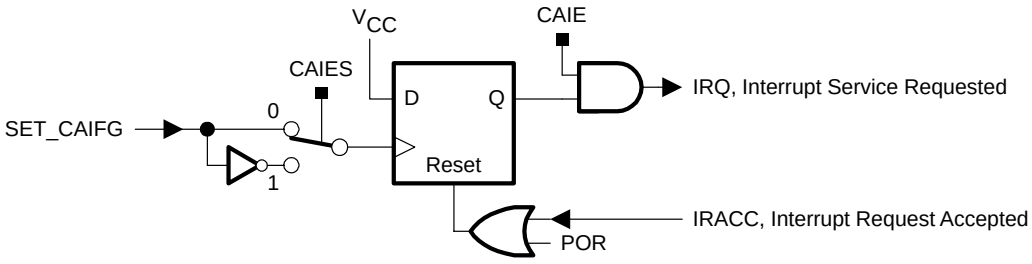
Figure 16–3. Transfer Characteristic and Power Dissipation in a CMOS Inverter/Buffer



16.2.6 Comparator_A Interrupts

One interrupt flag and one interrupt vector are associated with the Comparator_A as shown in Figure 16–4. The interrupt flag CAIFG is set on either the rising or falling edge of the comparator output, selected by the CAIES bit. If both the CAIE and the GIE bits are set, then the CAIFG flag generates an interrupt request. The CAIFG flag is automatically reset when the interrupt request is serviced or may be reset with software.

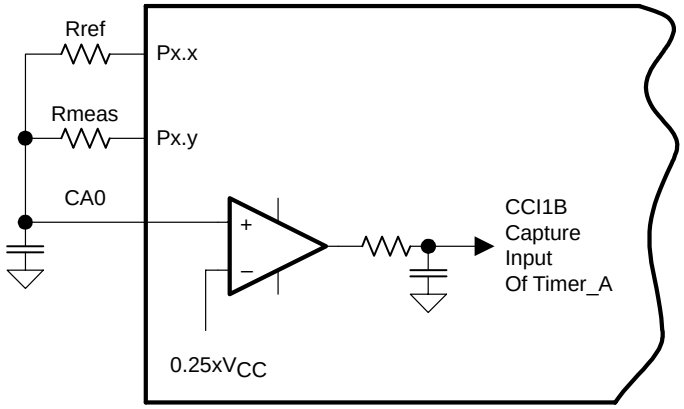
Figure 16–4. Comparator_A Interrupt System



16.2.7 Comparator_A Used to Measure Resistive Elements

The Comparator_A can be optimized to precisely measure resistive elements using single slope analog-to-digital conversion. For example, temperature can be converted into digital data using a thermistor, by comparing the thermistor's capacitor discharge time to that of a reference resistor as shown in Figure 16–5. A reference resister Rref is compared to Rmeas.

Figure 16–5. Temperature Measurement System



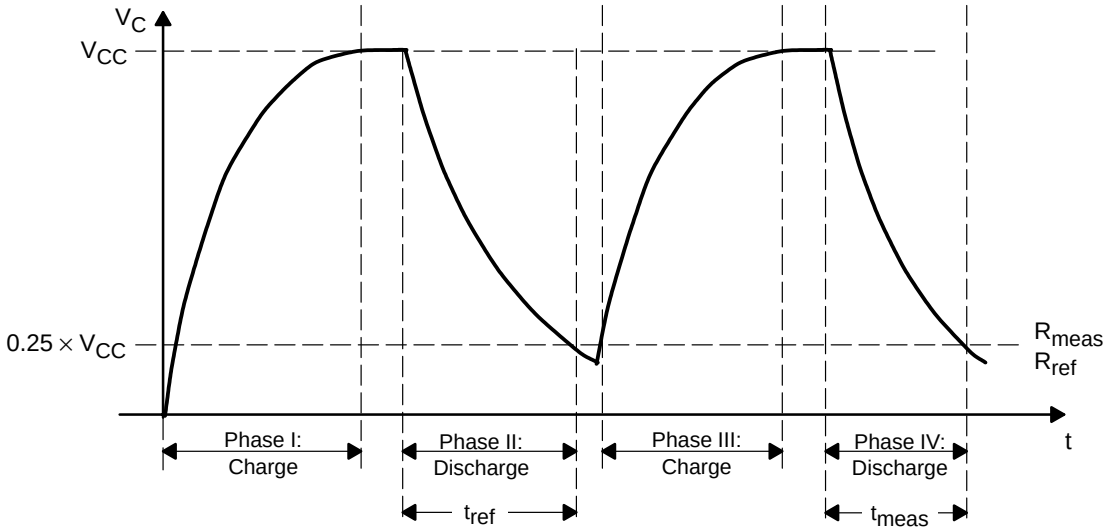
The MSP430 resources used to calculate the temperature sensed by Rmeas are:

- Two digital I/O pins to charge and discharge the capacitor.
- I/O set to output high (V_{CC}) to charge capacitor, reset to discharge.
- I/O switched to high-impedance input with CAPDx set when not in use.
- One output charges and discharges the capacitor via Rref.
- One output discharges capacitor via Rmeas.
- The + terminal is connected to the positive terminal of the capacitor.
- The – terminal is connected to a reference level, for example 0.25 x V_{CC}.
- The output filter should be used to minimize switching noise.
- CAOUT used to gate Timer_A CCI1B, capturing capacitor discharge time.

More than one resistive element can be measured. Additional elements are connected to CA0 with available I/O pins and switched to high impedance when not being measured.

The thermistor measurement is based on a ratiometric conversion principle. The ratio of two capacitor discharge times is calculated as shown in Figure 16–6.

Figure 16–6. Timing for Temperature Measurement Systems



The V_{CC} voltage and the capacitor value should remain constant during the conversion, but are not critical since they cancel in the ratio:

$$\frac{N_{meas}}{N_{ref}} = \frac{-R_{meas} \times C \times ln \frac{V_{ref}}{V_{CC}}}{-R_{ref} \times C \times ln \frac{V_{ref}}{V_{CC}}}$$

$$\frac{N_{meas}}{N_{ref}} = \frac{R_{meas}}{R_{ref}}$$

$$R_{meas} = R_{ref} \times \frac{N_{meas}}{N_{ref}}$$

16.3 Comparator_A Registers

The Comparator_A registers are listed in Table 16–1:

Table 16–1. Comparator_A Registers

Register	Short Form	Register Type	Address	Initial State
Comparator_A control register 1	CACTL1	Read/write	059h	Reset with POR
Comparator_A control register 2	CACTL2	Read/write	05Ah	Reset with POR
Comparator_A port disable	CAPD	Read/write	05Bh	Reset with POR

CACTL1, Comparator_A Control Register 1

7	6	5	4	3	2	1	0
CAEX	**CARSEL**	**CAREFx**		**CAON**	**CAIES**	**CAIE**	**CAIFG**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

CAEX Bit 7 Comparator_A exchange. This bit exchanges the comparator inputs and inverts the comparator output.

CARSEL Bit 6 Comparator_A reference select. This bit selects which terminal the V_{CAREF} is applied to.
When CAEX = 0:
0 V_{CAREF} is applied to the + terminal
1 V_{CAREF} is applied to the – terminal
When CAEX = 1:
0 V_{CAREF} is applied to the – terminal
1 V_{CAREF} is applied to the + terminal

CAREF Bits 5-4 Comparator_A reference. These bits select the reference voltage V_{CAREF}.
00 Internal reference off. An external reference can be applied.
01 $0.25*V_{CC}$
10 $0.50*V_{CC}$
11 Diode reference is selected

CAON Bit 3 Comparator_A on. This bit turns on the comparator. When the comparator is off it consumes no current. The reference circuitry is enabled or disabled independently.
0 Off
1 On

CAIES Bit 2 Comparator_A interrupt edge select
0 Rising edge
1 Falling edge

CAIE Bit 1 Comparator_A interrupt enable
0 Disabled
1 Enabled

CAIFG Bit 0 The Comparator_A interrupt flag
0 No interrupt pending
1 Interrupt pending

Comparator_A, Control Register CACTL2

7	6	5	4	3	2	1	0
Unused				P2CA1	P2CA0	CAF	CAOUT
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	r–(0)

Unused Bits 7-4 Unused.

P2CA1 Bit 3 Pin to CA1. This bit selects the CA1 pin function.
0 The pin is not connected to CA1
1 The pin is connected to CA1

P2CA0 Bit 2 Pin to CA0. This bit selects the CA0 pin function.
0 The pin is not connected to CA0
1 The pin is connected to CA0

CAF Bit 1 Comparator_A output filter
0 Comparator_A output is not filtered
1 Comparator_A output is filtered

CAOUT Bit 0 Comparator_A output. This bit reflects the value of the comparator output. Writing this bit has no effect.

Comparator_A, Port Disable Register CAPD

7	6	5	4	3	2	1	0
CAPD7	CAPD6	CAPD5	CAPD4	CAPD3	CAPD2	CAPD1	CAPD0
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

CAPDx Bits 7-0 Comparator_A port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_A. For example, if CAOUT is on pin P2.2, the CAPDx bits can be used to individually enable or disable each P2.x pin buffer. CAPD0 disables P2.0, CAPD1 disables P2.1, etc.
0 The input buffer is enabled.
1 The input buffer is disabled.

Chapter 17

ADC12

The ADC12 module is a high-performance 12-bit analog-to-digital converter. This chapter describes the ADC12. The ADC12 is implemented in the MSP430x13x, MSP430x14x, MSP430x15x, and MSP430x16x devices.

Topic **Page**

17.1 ADC12 Introduction

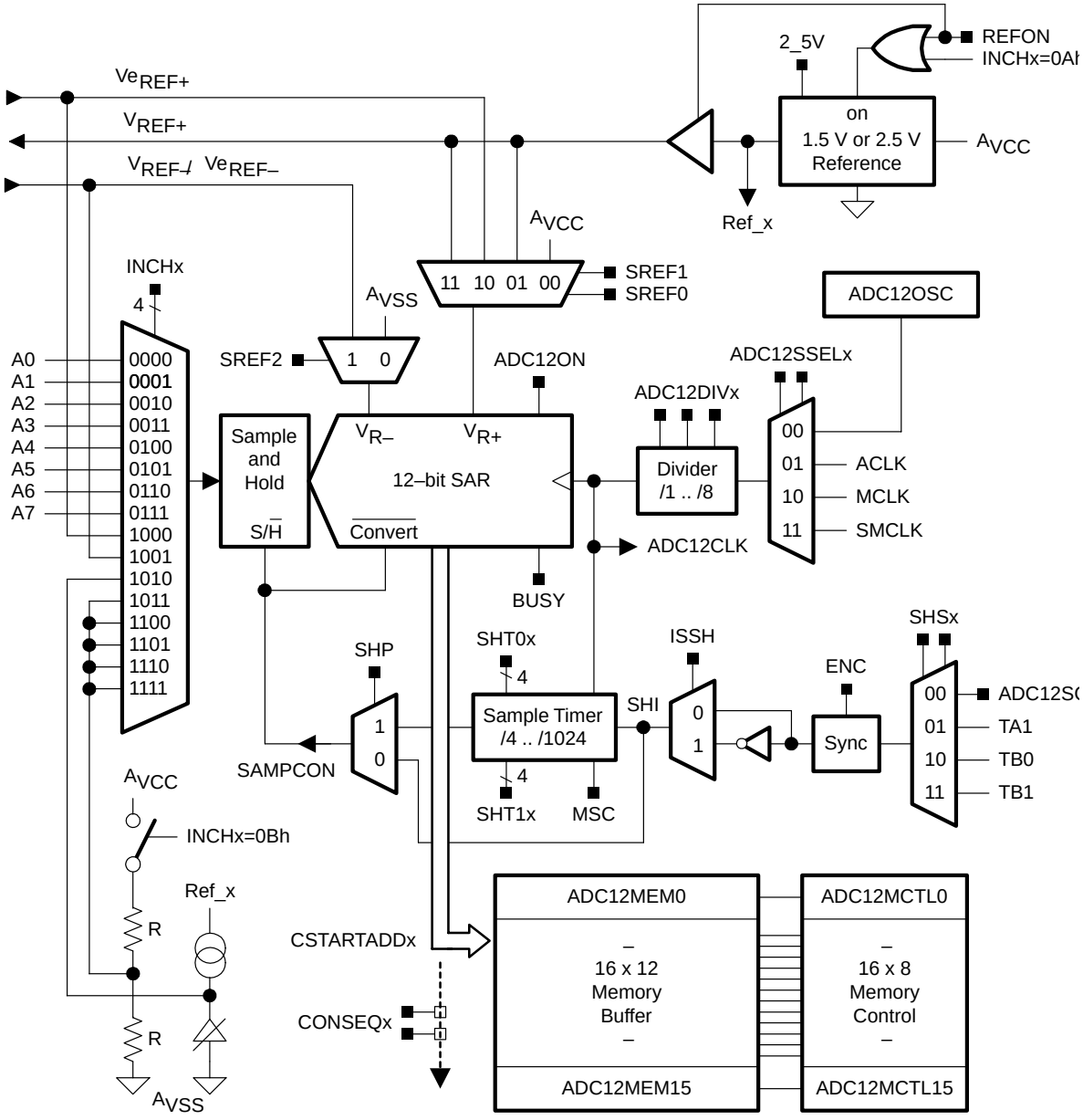
The ADC12 module supports fast, 12-bit analog-to-digital conversions. The module implements a 12-bit SAR core, sample select control, reference generator and a 16 word conversion-and-control buffer. The conversion-and-control buffer allows up to 16 independent ADC samples to be converted and stored without any CPU intervention.

ADC12 features include:

- Greater than 200 ksps maximum conversion rate
- Monotonic 12-bit converter with no missing codes
- Sample-and-hold with programmable sampling periods controlled by software or timers.
- Conversion initiation by software, Timer_A, or Timer_B
- Software selectable on-chip reference voltage generation (1.5 V or 2.5 V)
- Software selectable internal or external reference
- Eight individually configurable external input channels
- Conversion channels for internal temperature sensor, AV_{CC}, and external references
- Independent channel-selectable reference sources for both positive and negative references
- Selectable conversion clock source
- Single-channel, repeat-single-channel, sequence, and repeat-sequence conversion modes
- ADC core and reference voltage can be powered down separately
- Interrupt vector register for fast decoding of 18 ADC interrupts
- 16 conversion-result storage registers

The block diagram of ADC12 is shown in Figure 17–1.

Figure 17–1. ADC12 Block Diagram



17.2 ADC12 Operation

The ADC12 module is configured with user software. The setup and operation of the ADC12 is discussed in the following sections.

17.2.1 12-Bit ADC Core

The ADC core converts an analog input to its 12-bit digital representation and stores the result in 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. The digital output (N_{ADC}) is full scale (0FFFh) 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 for the ADC result N_{ADC} is:

$$N_{ADC} = 4095 \times \frac{Vin - V_{R-}}{V_{R+} - V_{R-}}$$

The ADC12 core is configured by two control registers, ADC12CTL0 and ADC12CTL1. The core is enabled with the ADC12ON bit. The ADC12 can be turned off when not in use to save power. With few exceptions the ADC12 control bits can only be modified when ENC = 0. ENC must be set to 1 before any conversion can take place.

Conversion Clock Selection

The ADC12CLK is used both as the conversion clock and to generate the sampling period when the pulse sampling mode is selected. The ADC12 source clock is selected using the ADC12SSELx bits and can be divided from 1-8 using the ADC12DIVx bits. Possible ADC12CLK sources are SMCLK, MCLK, ACLK, and an internal oscillator ADC12OSC.

The ADC12OSC, generated internally, is in the 5-MHz range, but varies with individual devices, supply voltage, and temperature. See the device-specific data sheet for the ADC12OSC specification.

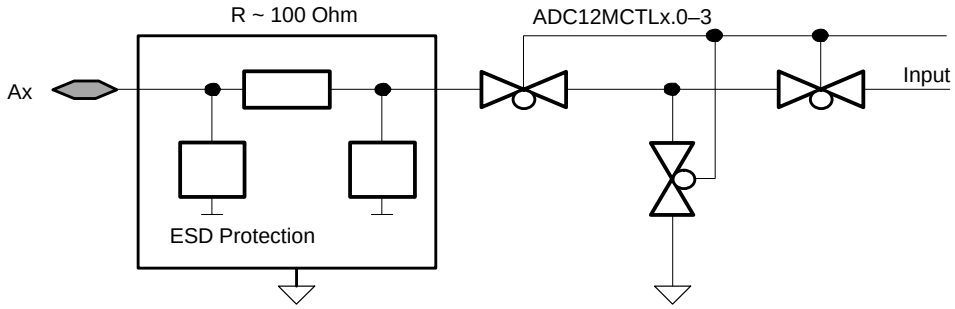
The user must ensure that the clock chosen for ADC12CLK remains active until the end of a conversion. If the clock is removed during a conversion, the operation will not complete and any result will be invalid.

17.2.2 ADC12 Inputs and Multiplexer

The eight external and four internal analog signals are selected as the channel for conversion by the analog input multiplexer. The input multiplexer is a break-before-make type to reduce input-to-input noise injection resulting from channel switching as shown in Figure 17–2. 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.

The ADC12 uses the charge redistribution method. When the inputs are internally switched, the switching action may cause transients on the input signal. These transients decay and settle before causing errant conversion.

Figure 17–2. Analog Multiplexer



Analog Port Selection

The ADC12 inputs are multiplexed with the port P6 pins, which are digital CMOS gates. When analog signals are applied to digital CMOS gates, parasitic current can flow from V_{CC} to GND. This parasitic current occurs if the input voltage is near the transition level of the gate. Disabling the port pin buffer eliminates the parasitic current flow and therefore reduces overall current consumption. The P6SELx bits provide the ability to disable the port pin input buffer.

```
; P6.0 and P6.1 configured for analog input
      BIS.B #3h,&P6SEL   ; P6.1 and P6.0 ADC12 function
      BIC.B #3h,&P6DIR   ; P6.1 and P6.0 input direction
```

17.2.3 Voltage Reference Generator

The ADC12 module contains a built-in voltage reference with two selectable voltage levels, 1.5 V and 2.5 V. Either of these reference voltages may be used internally and externally on pin V_{REF+}.

Setting REFON=1 enables the internal reference. When REF2_5V = 1, the internal reference is 2.5 V, the reference is 1.5 V when REF2_5V = 0. The reference can be turned off to save power when not in use.

For proper operation the internal voltage reference generator must be supplied with storage capacitance across V_{REF+} and A_{VSS}. The recommended storage capacitance is a parallel combination of 10-μF and 0.1-μF capacitors. From turn-on, a maximum of 17 ms must be allowed for the voltage reference generator to bias the recommended storage capacitors. If the internal reference generator is not used for the conversion, the storage capacitors are not required.

Note: Reference Decoupling

Approximately 200 μA is required from *any* reference used by the ADC12 while the two LSBs are being resolved during a conversion. A parallel combination of 10-μF and 0.1-μF capacitors is recommended for *any* reference used.

External references may be supplied for V_{R+} and V_{R-} through pins Ve_{REF+} and V_{REF-}/Ve_{REF-} respectively.

17.2.4 Sample and Conversion Timing

An analog-to-digital conversion is initiated with a rising edge of the sample input signal SHI. The source for SHI is selected with the SHSx bits and includes the following:

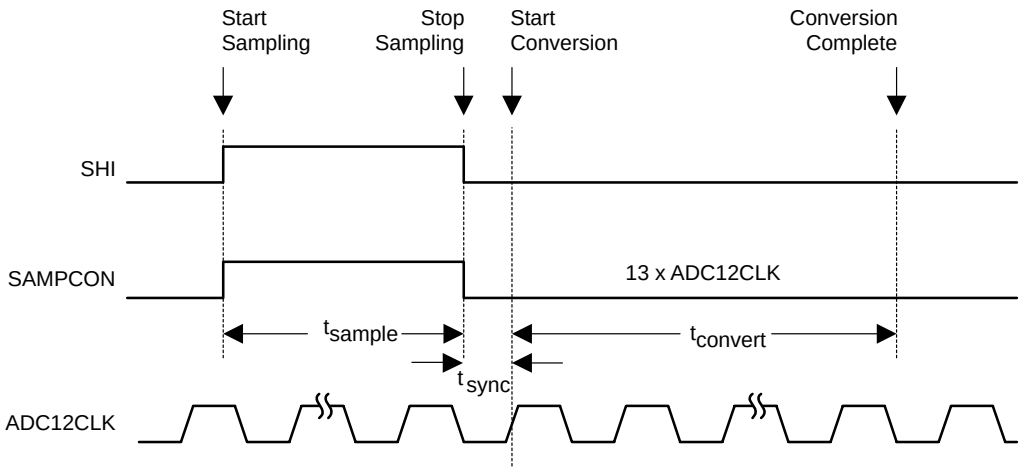
- The ADC12SC bit
- The Timer_A Output Unit 1
- The Timer_B Output Unit 0
- The Timer_B Output Unit 1

The polarity of the SHI signal source can be inverted with the ISSH bit. The SAMPCON signal controls the sample period and start of conversion. When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the analog-to-digital conversion, which requires 13 ADC12CLK cycles. Two different sample-timing methods are defined by control bit SHP, extended sample mode and pulse mode.

Extended Sample Mode

The extended sample mode is selected when SHP = 0. The SHI signal directly controls SAMPCON and defines the length of the sample period t_{sample}. When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the conversion after synchronization with ADC12CLK. See Figure 17–3.

Figure 17–3. Extended Sample Mode

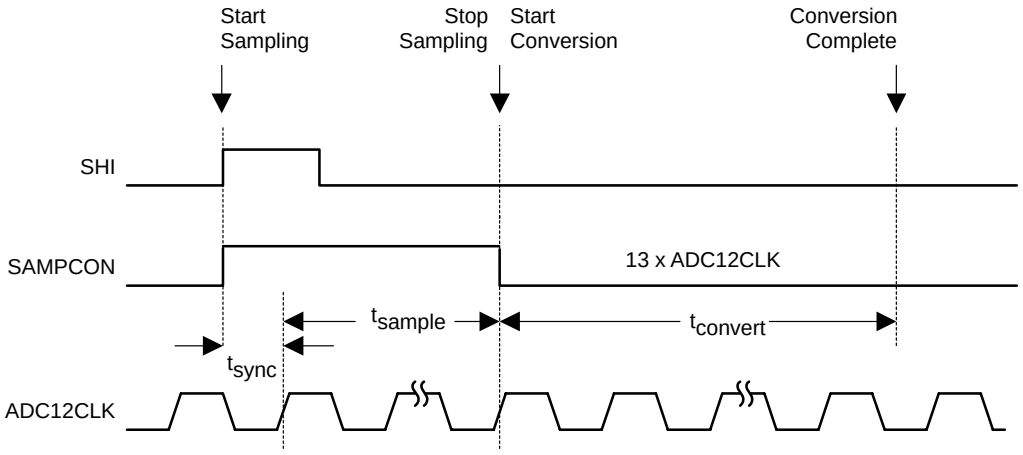


Pulse Sample Mode

The pulse sample mode is selected when SHP = 1. The SHI signal is used to trigger the sampling timer. The SHT0x and SHT1x bits in ADC12CTL0 control the interval of the sampling timer that defines the SAMPCON sample period t_{sample}. The sampling timer keeps SAMPCON high after synchronization with AD12CLK for a programmed interval t_{sample}. The total sampling time is t_{sample} plus t_{sync}. See Figure 17–4.

The SHTx bits select the sampling time in 4x multiples of ADC12CLK. SHT0x selects the sampling time for ADC12MCTL0 to 7 and SHT1x selects the sampling time for ADC12MCTL8 to 15.

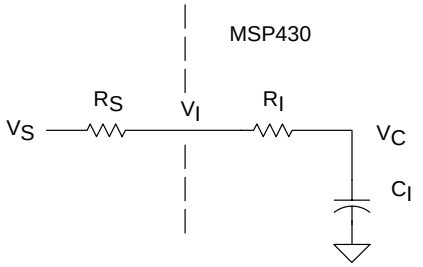
Figure 17–4. Pulse Sample Mode



Sample Timing Considerations

When SAMPCON = 0 all Ax inputs are high impedance. When SAMPCON = 1, the selected Ax input can be modeled as an RC low-pass filter during the sampling time t_{sample}, as shown below in Figure 17–5. An internal MUX-on input resistance R_I (max. 2 kΩ) in series with capacitor C_I (max. 40 pF) is seen by the source. The capacitor C_I voltage V_C must be charged to within $1/2$ LSB of the source voltage V_S for an accurate 12-bit conversion.

Figure 17–5. Analog Input Equivalent Circuit



V_I = Input voltage at pin Ax
V_S = External source voltage
R_S = External source resistance
R_I = Internal MUX-on input resistance
C_I = Input capacitance
V_C = Capacitance-charging voltage

The resistance of the source R_S and R_I affect t_{sample}. The following equation can be used to calculate the minimum sampling time t_{sample} for a 12-bit conversion:

$$t_{sample} > (R_S + R_I) \times \ln(2^{13}) \times C_I \quad (1)$$

Substituting the values for R_I and C_I given above, the equation becomes:

$$t_{sample} > (R_S + 2k\Omega) \times 9.011 \times 40pF \quad (2)$$

For example, if R_S is 10 kΩ, t_{sample} must be greater than 4.33 μs.

17.2.5 Conversion Memory

There are 16 ADC12MEMx conversion memory registers to store conversion results. Each ADC12MEMx is configured with an associated ADC12MCTLx control register. The SREFx bits define the voltage reference and the INCHx bits select the input channel. The EOS bit defines the end of sequence when a sequential conversion mode is used. A sequence rolls over from ADC12MEM15 to ADC12MEM0 when the EOS bit in ADC12MCTL15 is not set.

The CSTARTADDx bits define the first ADC12MCTLx used for any conversion. If the conversion mode is single-channel or repeat-single-channel the CSTARTADDx points to the single ADC12MCTLx to be used.

If the conversion mode selected is either sequence-of-channels or repeat-sequence-of-channels, CSTARTADDx points to the first ADC12MCTLx location to be used in a sequence. A pointer, not visible to software, is incremented automatically to the next ADC12MCTLx in a sequence when each conversion completes. The sequence continues until an EOS bit in ADC12MCTLx is processed - this is the last control byte processed.

When conversion results are written to a selected ADC12MEMx, the corresponding flag in the ADC12IFGx register is set.

17.2.6 ADC12 Conversion Modes

The ADC12 has four operating modes selected by the CONSEQx bits as discussed in Table 17–1.

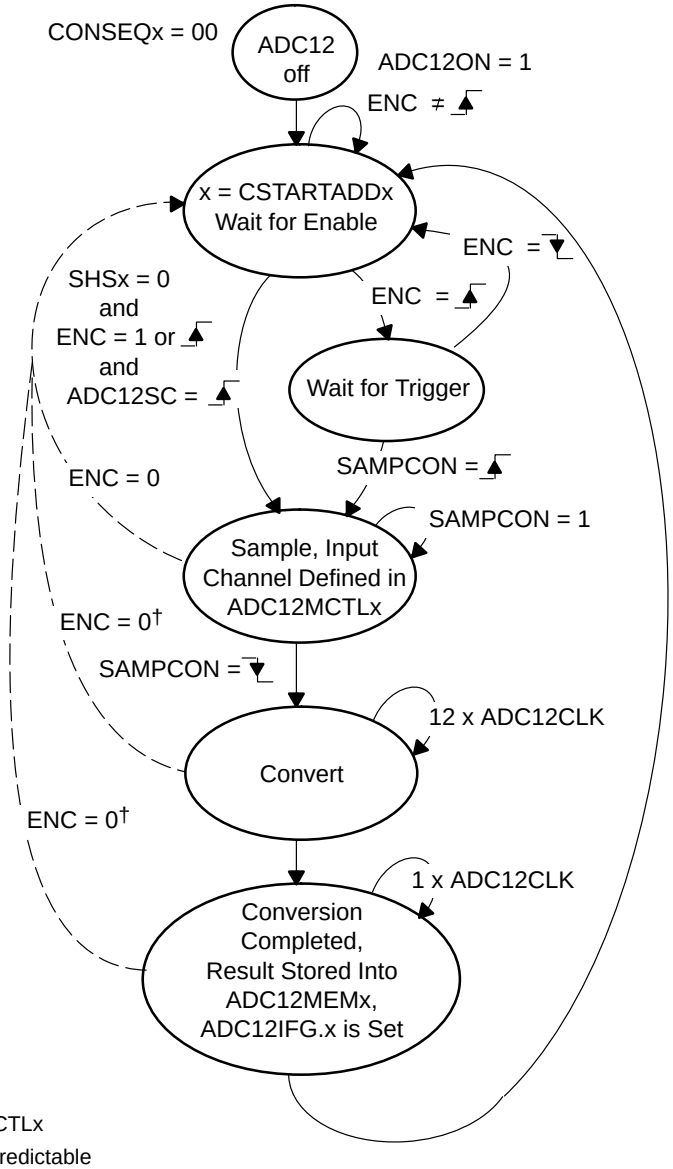
Table 17–1. Conversion Mode Summary

CONSEQx	MODE	OPERATION
00	Single channel single-conversion	A single channel is converted once.
01	Sequence-of-channels	A sequence of channels is converted once.
10	Repeat-single-channel	A single channel is converted repeatedly.
11	Repeat-sequence-of-channels	A sequence of channels is converted repeatedly.

Single-Channel Single-Conversion Mode

A single channel is sampled and converted once. The ADC result is written to the ADC12MEMx defined by the CSTARTADDx bits. Figure 17–6 shows the flow of the Single-Channel, Single-Conversion mode. When ADC12SC triggers a conversion, successive conversions can be triggered by the ADC12SC bit. When any other trigger source is used, ENC must be toggled between each conversion.

Figure 17–6. Single-Channel, Single-Conversion Mode

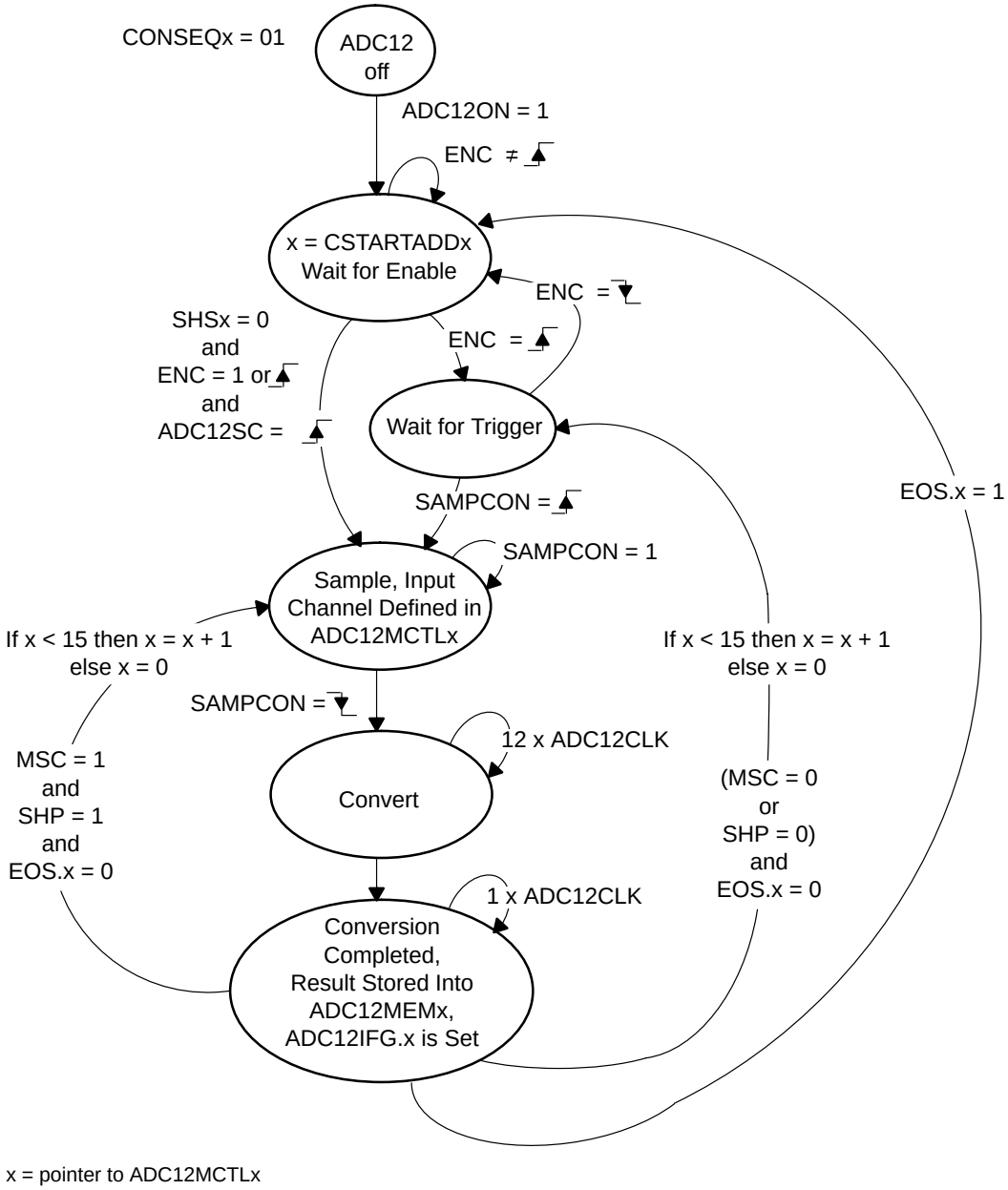


x = pointer to ADC12MCTLx
†Conversion result is unpredictable

Sequence-of-Channels Mode

A sequence of channels is sampled and converted once. The ADC results are written to the conversion memories starting with the ADCMEMx defined by the CSTARTADDx bits. The sequence stops after the measurement of the channel with a set EOS bit. Figure 17–7 shows the sequence-of-channels mode. When ADC12SC triggers a sequence, successive sequences can be triggered by the ADC12SC bit. When any other trigger source is used, ENC must be toggled between each sequence.

Figure 17–7. Sequence-of-Channels Mode

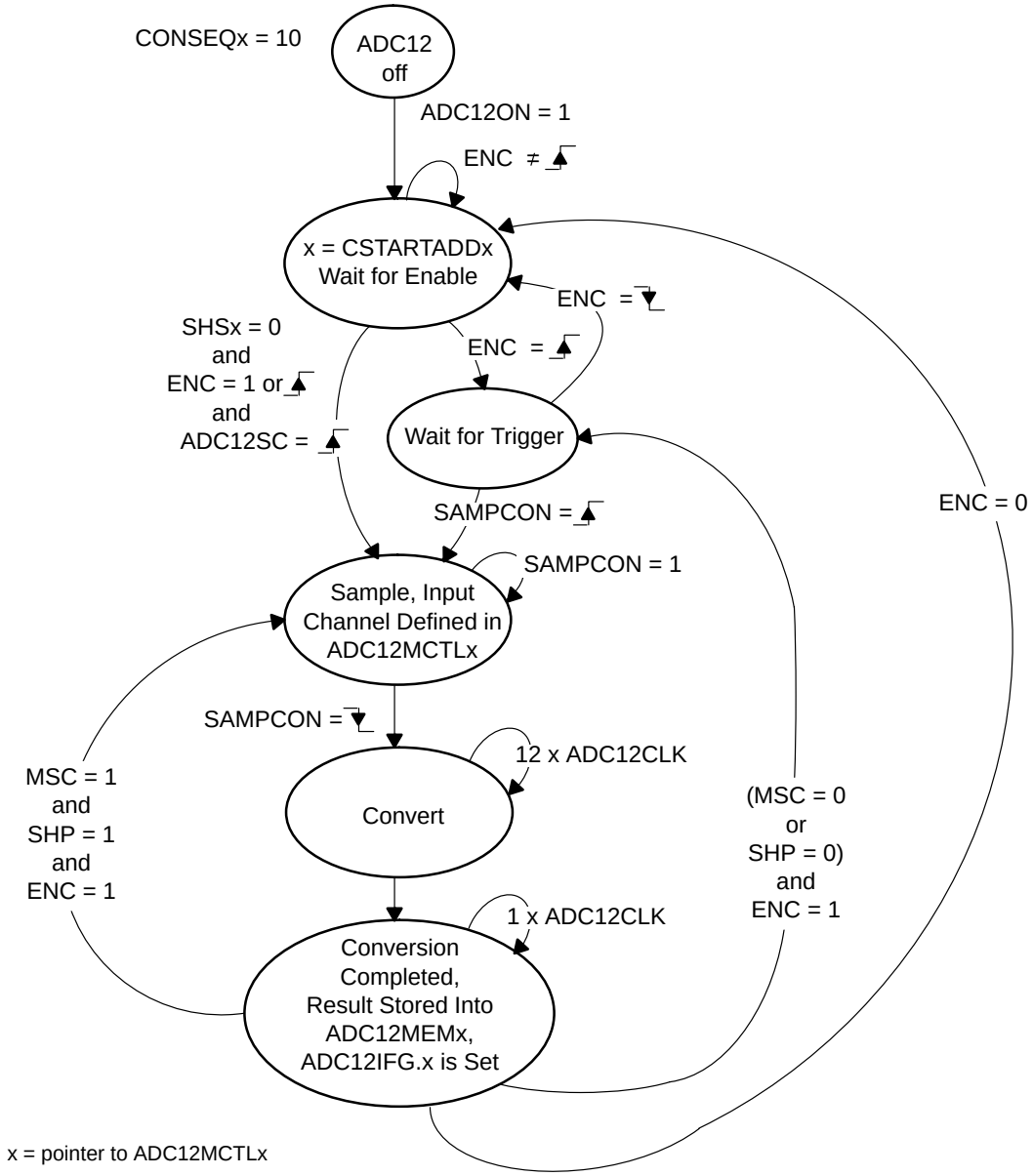


x = pointer to ADC12MCTLx

Repeat-Single-Channel Mode

A single channel is sampled and converted continuously. The ADC results are written to the ADC12MEMx defined by the CSTARTADDx bits. It is necessary to read the result after the completed conversion because only one ADC12MEMx memory is used and is overwritten by the next conversion. Figure 17–8 shows repeat-single-channel mode

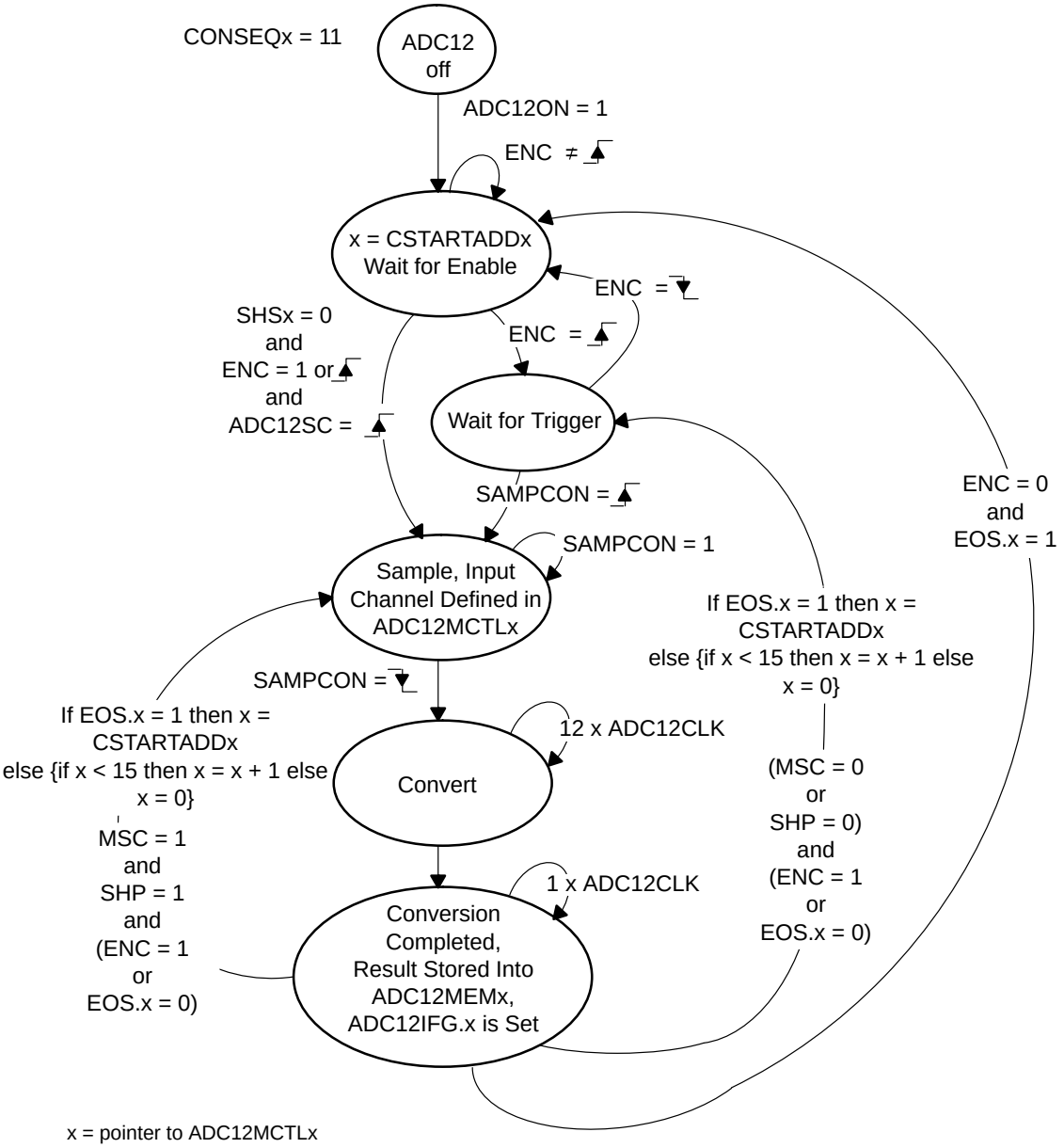
Figure 17–8. Repeat-Single-Channel Mode



Repeat-Sequence-of-Channels Mode

A sequence of channels is sampled and converted repeatedly. The ADC results are written to the conversion memories starting with the ADC12MEMx defined by the CSTARTADDx bits. The sequence ends after the measurement of the channel with a set EOS bit and the next trigger signal re-starts the sequence. Figure 17–9 shows the repeat-sequence-of-channels mode.

Figure 17–9. Repeat-Sequence-of-Channels Mode



Using the Multiple Sample and Convert (MSC) Bit

To configure the converter to perform successive conversions automatically and as quickly as possible, a multiple sample and convert function is available. When MSC = 1, CONSEQx > 0, and the sample timer is used, the first rising edge of the SHI signal triggers the first conversion. 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 in the single-sequence mode or until the ENC bit is toggled in repeat-single-channel, or repeated-sequence modes. The function of the ENC bit is unchanged when using the MSC bit.

Stopping Conversions

Stopping ADC12 activity depends on the mode of operation. The recommended ways to stop an active conversion or conversion sequence are:

- Resetting ENC in single-channel single-conversion mode stops a conversion immediately and the results are unpredictable. For correct results, poll the busy bit until reset before clearing ENC.
- Resetting ENC during repeat-single-channel operation stops the converter at the end of the current conversion.
- Resetting ENC during a sequence or repeat-sequence mode stops the converter at the end of the sequence.
- Any conversion mode may be stopped immediately by setting the CONSEQx = 0 and resetting ENC bit. Conversion data are unreliable.

Note: No EOS Bit Set For Sequence

If no EOS bit is set and a sequence mode is selected, resetting the ENC bit does not stop the sequence. To stop the sequence, first select a single-channel mode and then reset ENC.

17.2.7 Using ADC12 with the DMA Controller

MSP430 devices with an integrated DMA controller can automatically move data from any ADC12MEMx register to another location. DMA transfers are done without CPU intervention and independently of any low-power modes. The DMA controller increases throughput of the ADC12 module, and enhances low-power applications allowing the CPU to remain off while data transfers occur.

DMA transfers can be triggered from any ADC12IFGx flag. When CONSEQx = {0,2} the ADC12IFGx flag for the ADC12MEMx used for the conversion can trigger a DMA transfer. When CONSEQx = {1,3}, the ADC12IFGx flag for the last ADC12MEMx in the sequence can trigger a DMA transfer. Any ADC12IFGx flag is automatically cleared when the DMA controller accesses the corresponding ADC12MEMx. See the *DMA Controller* chapter.

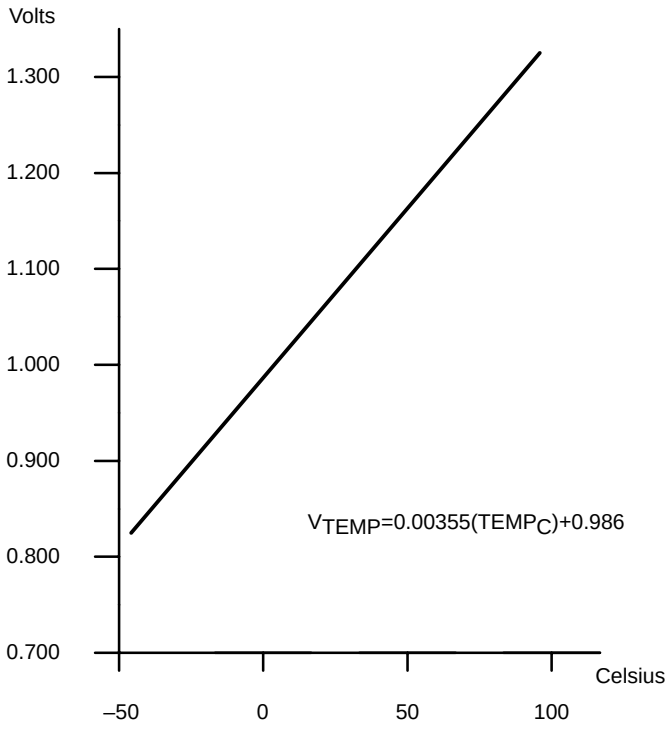
17.2.8 Using the Integrated Temperature Sensor

To use the on-chip temperature sensor, the user selects the analog input channel INCHx = 1010. Any other configuration is done as if an external channel was selected, including reference selection, conversion-memory selection, etc.

The typical temperature sensor transfer function is shown in Figure 17–10. When using the temperature sensor, the sample period must be greater than 30 μs. The temperature sensor offset error can be large, and may need to be calibrated for most applications. See device-specific data sheet for parameters.

Selecting the temperature sensor automatically turns on the on-chip reference generator as a voltage source for the temperature sensor. However, it does not enable the V_{REF+} output or affect the reference selections for the conversion. The reference choices for converting the temperature sensor are the same as with any other channel.

Figure 17–10. Typical Temperature Sensor Transfer Function



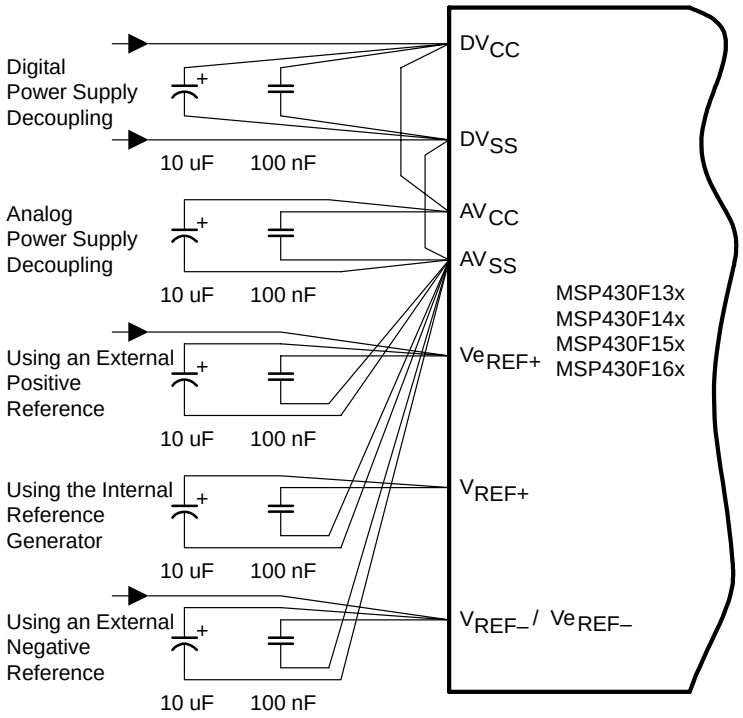
17.2.9 ADC12 Grounding and Noise Considerations

As with any high-resolution ADC, appropriate printed-circuit-board layout and grounding techniques should be followed to eliminate ground loops, unwanted parasitic effects, and noise.

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. The connections shown in Figure 17–11 help avoid this.

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. A noise-free design using separate analog and digital ground planes with a single-point connection is recommend to achieve high accuracy.

Figure 17–11. ADC12 Grounding and Noise Considerations



17.2.10 ADC12 Interrupts

The ADC12 has 18 interrupt sources:

- ADC12IFG0-ADC12IFG15
- ADC12OV, ADC12MEMx overflow
- ADC12TOV, ADC12 conversion time overflow

The ADC12IFGx bits are set when their corresponding ADC12MEMx memory register is loaded with a conversion result. An interrupt request is generated if the corresponding ADC12IEx bit and the GIE bit are set. The ADC12OV condition occurs when a conversion result is written to any ADC12MEMx before its previous conversion result was read. The ADC12TOV condition is generated when another sample-and-conversion is requested before the current conversion is completed.

ADC12IV, Interrupt Vector Generator

All ADC12 interrupt sources are prioritized and combined to source a single interrupt vector. The interrupt vector register ADC12IV is used to determine which enabled ADC12 interrupt source requested an interrupt.

The highest priority enabled ADC12 interrupt generates a number in the ADC12IV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled ADC12 interrupts do not affect the ADC12IV value.

Any access, read or write, of the ADC12IV register automatically resets the ADC12OV condition or the ADC12TOV condition if either was the highest pending interrupt. Neither interrupt condition has an accessible interrupt flag. The ADC12IFGx flags are not reset by an ADC12IV access. ADC12IFGx bits are reset automatically by accessing their associated ADC12MEMx register or may be reset with software.

If another interrupt is pending after servicing of an interrupt, another interrupt is generated. For example, if the ADC12OV and ADC12IFG3 interrupts are pending when the interrupt service routine accesses the ADC12IV register, the ADC12OV interrupt condition is reset automatically. After the RETI instruction of the interrupt service routine is executed, the ADC12IFG3 generates another interrupt.

ADC12 Interrupt Handling Software Example

The following software example shows the recommended use of ADC12IV and the handling overhead. The ADC12IV value is added to the PC to automatically jump to the appropriate routine.

The numbers at the right margin show the necessary CPU cycles for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself. The latencies are:

- ADC12IFG0 - ADC12IFG14, ADC12TOV and ADC12OV 16 cycles
- ADC12IFG15 14 cycles

The interrupt handler for ADC12IFG15 shows a way to check immediately if a higher prioritized interrupt occurred during the processing of ADC12IFG15. This saves nine cycles if another ADC12 interrupt is pending.

```
; Interrupt handler for ADC12.
INT_ADC12          ; Enter Interrupt Service Routine     6
   ADD&ADC12IV,PC  ; Add offset to PC                    3
   RETI            ; Vector 0: No interrupt              5
   JMPADOV         ; Vector 2: ADC overflow              2
   JMPADTOV        ; Vector 4: ADC timing overflow       2
   JMPADM0         ; Vector 6: ADC12IFG0                 2
      ...          ; Vectors 8-32                        2
   JMPADM14        ; Vector 34: ADC12IFG14               2
;
; Handler for ADC12IFG15 starts here. No JMP required.
;
ADM15    MOV  &ADC12MEM15,xxx; Move result, flag is reset
         ...                 ; Other instruction needed?
         JMP  INT_ADC12      ; Check other int pending
;
;  ADC12IFG14-ADC12IFG1 handlers go here
;
ADM0     MOV  &ADC12MEM0,xxx ; Move result, flag is reset
         ...                 ; Other instruction needed?
         RETI                ; Return                    5
;
ADTOV     ...                ; Handle Conv. time overflow
          RETI               ; Return                    5
;
ADOV      ...                ; Handle ADCMEMx overflow
          RETI               ; Return                    5
```

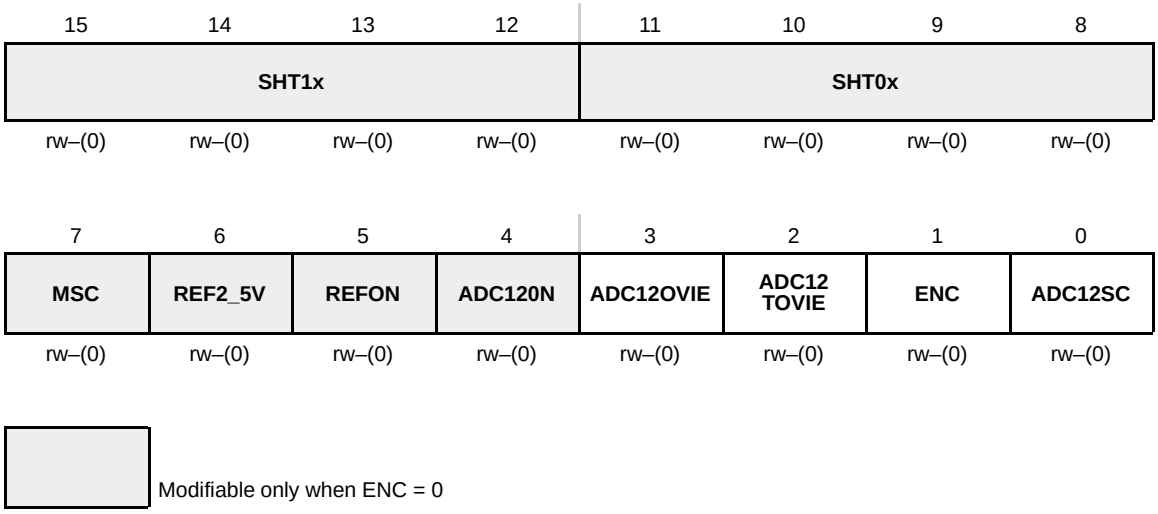
17.3 ADC12 Registers

The ADC12 registers are listed in Table 17–2:

Table 17–2. ADC12 Registers

Register	Short Form	Register Type	Address	Initial State
ADC12 control register 0	ADC12CTL0	Read/write	01A0h	Reset with POR
ADC12 control register 1	ADC12CTL1	Read/write	01A2h	Reset with POR
ADC12 interrupt flag register	ADC12IFG	Read/write	01A4h	Reset with POR
ADC12 interrupt enable register	ADC12IE	Read/write	01A6h	Reset with POR
ADC12 interrupt vector word	ADC12IV	Read	01A8h	Reset with POR
ADC12 memory 0	ADC12MEM0	Read/write	0140h	Unchanged
ADC12 memory 1	ADC12MEM1	Read/write	0142h	Unchanged
ADC12 memory 2	ADC12MEM2	Read/write	0144h	Unchanged
ADC12 memory 3	ADC12MEM3	Read/write	0146h	Unchanged
ADC12 memory 4	ADC12MEM4	Read/write	0148h	Unchanged
ADC12 memory 5	ADC12MEM5	Read/write	014Ah	Unchanged
ADC12 memory 6	ADC12MEM6	Read/write	014Ch	Unchanged
ADC12 memory 7	ADC12MEM7	Read/write	014Eh	Unchanged
ADC12 memory 8	ADC12MEM8	Read/write	0150h	Unchanged
ADC12 memory 9	ADC12MEM9	Read/write	0152h	Unchanged
ADC12 memory 10	ADC12MEM10	Read/write	0154h	Unchanged
ADC12 memory 11	ADC12MEM11	Read/write	0156h	Unchanged
ADC12 memory 12	ADC12MEM12	Read/write	0158h	Unchanged
ADC12 memory 13	ADC12MEM13	Read/write	015Ah	Unchanged
ADC12 memory 14	ADC12MEM14	Read/write	015Ch	Unchanged
ADC12 memory 15	ADC12MEM15	Read/write	015Eh	Unchanged
ADC12 memory control 0	ADC12MCTL0	Read/write	080h	Reset with POR
ADC12 memory control 1	ADC12MCTL1	Read/write	081h	Reset with POR
ADC12 memory control 2	ADC12MCTL2	Read/write	082h	Reset with POR
ADC12 memory control 3	ADC12MCTL3	Read/write	083h	Reset with POR
ADC12 memory control 4	ADC12MCTL4	Read/write	084h	Reset with POR
ADC12 memory control 5	ADC12MCTL5	Read/write	085h	Reset with POR
ADC12 memory control 6	ADC12MCTL6	Read/write	086h	Reset with POR
ADC12 memory control 7	ADC12MCTL7	Read/write	087h	Reset with POR
ADC12 memory control 8	ADC12MCTL8	Read/write	088h	Reset with POR
ADC12 memory control 9	ADC12MCTL9	Read/write	089h	Reset with POR
ADC12 memory control 10	ADC12MCTL10	Read/write	08Ah	Reset with POR
ADC12 memory control 11	ADC12MCTL11	Read/write	08Bh	Reset with POR
ADC12 memory control 12	ADC12MCTL12	Read/write	08Ch	Reset with POR
ADC12 memory control 13	ADC12MCTL13	Read/write	08Dh	Reset with POR
ADC12 memory control 14	ADC12MCTL14	Read/write	08Eh	Reset with POR
ADC12 memory control 15	ADC12MCTL15	Read/write	08Fh	Reset with POR

ADC12CTL0, ADC12 Control Register 0



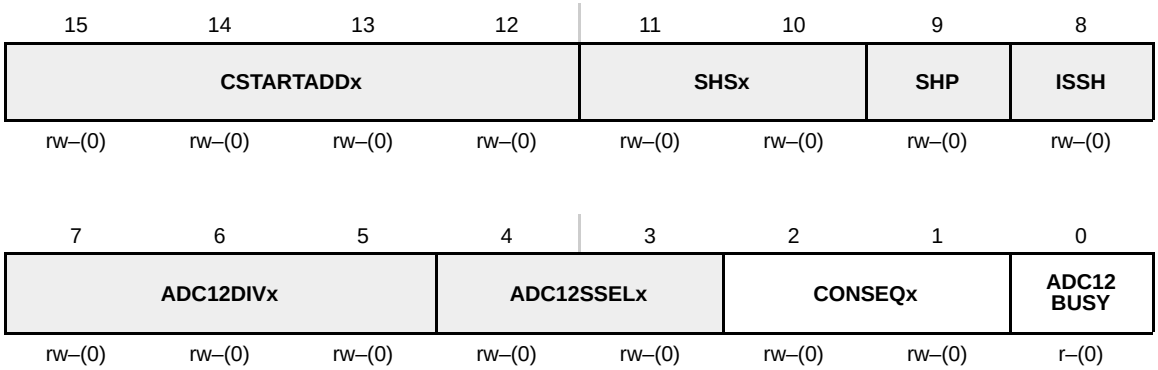
SHT1x Bits 15-12 Sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM8 to ADC12MEM15.

SHT0x Bits 11-8 Sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM0 to ADC12MEM7.

SHTx Bits	ADC12CLK cycles
0000	4
0001	8
0010	16
0011	32
0100	64
0101	96
0110	128
0111	192
1000	256
1001	384
1010	512
1011	768
1100	1024
1101	1024
1110	1024
1111	1024

MSC	Bit 7	Multiple sample and conversion. Valid only for sequence or repeated modes. 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-conversions are performed automatically as soon as the prior conversion is completed.
REF2_5V	Bit 6	Reference generator voltage. REFON must also be set. 0 1.5 V 1 2.5 V
REFON	Bit 5	Reference generator on 0 Reference off 1 Reference on
ADC12ON	Bit 4	ADC12 on 0 ADC12 off 1 ADC12 on
ADC12OVIE	Bit 3	ADC12MEMx overflow-interrupt enable. The GIE bit must also be set to enable the interrupt. 0 Overflow interrupt disabled 1 Overflow interrupt enabled
ADC12 TOVIE	Bit 2	ADC12 conversion-time-overflow interrupt enable. The GIE bit must also be set to enable the interrupt. 0 Conversion time overflow interrupt disabled 1 Conversion time overflow interrupt enabled
ENC	Bit 1	Enable conversion 0 ADC12 disabled 1 ADC12 enabled
ADC12SC	Bit 0	Start conversion. Software-controlled sample-and-conversion start. ADC12SC and ENC may be set together with one instruction. ADC12SC is reset automatically. 0 No sample-and-conversion-start 1 Start sample-and-conversion

ADC12CTL1, ADC12 Control Register 1



CSTART ADDx Bits 15-12 Conversion start address. These bits select which ADC12 conversion-memory register is used for a single conversion or for the first conversion in a sequence. The value of CSTARTADDx is 0 to 0Fh, corresponding to ADC12MEM0 to ADC12MEM15.

SHSx Bits 11-10 Sample-and-hold source select
00 ADC12SC bit
01 Timer_A.OUT1
10 Timer_B.OUT0
11 Timer_B.OUT1

SHP Bit 9 Sample-and-hold pulse-mode select. This 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 SAMPCON signal is sourced from the sample-input signal.
1 SAMPCON signal is sourced from the sampling timer.

ISSH Bit 8 Invert signal sample-and-hold
0 The sample-input signal is not inverted.
1 The sample-input signal is inverted.

ADC12DIVx Bits 7-5 ADC12 clock divider
000 /1
001 /2
010 /3
011 /4
100 /5
101 /6
110 /7
111 /8

ADC12 SSELx	Bits 4-3	ADC12 clock source select 00 ADC12OSC 01 ACLK 10 MCLK 11 SMCLK
CONSEQx	Bits 2-1	Conversion sequence mode select 00 Single-channel, single-conversion 01 Sequence-of-channels 10 Repeat-single-channel 11 Repeat-sequence-of-channels
ADC12 BUSY	Bit 0	ADC12 busy. This bit indicates an active sample or conversion operation. 0 No operation is active. 1 A sequence, sample, or conversion is active**.**

ADC12MEMx, ADC12 Conversion Memory Registers

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**Conversion Results**			
r0	r0	r0	r0	rw	rw	rw	rw

7	6	5	4	3	2	1	0
Conversion Results							
rw	rw	rw	rw	rw	rw	rw	rw

Conversion Results	Bits 15-0	The 12-bit conversion results are right-justified. Bit 11 is the MSB. Bits 15-12 are always 0. Writing to the conversion memory registers will corrupt the results.

ADC12MCTLx, ADC12 Conversion Memory Control Registers

7	6	5	4	3	2	1	0
EOS	**SREFx**			**INCHx**			
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

Modifiable only when ENC = 0

EOS Bit 7 End of sequence. Indicates the last conversion in a sequence.
- 0 Not end of sequence
- 1 End of sequence

SREFx Bits 6-4 Select reference
- 000 $V_{R+} = AV_{CC}$ and $V_{R-} = AV_{SS}$
- 001 $V_{R+} = V_{REF+}$ and $V_{R-} = AV_{SS}$
- 010 $V_{R+} = Ve_{REF+}$ and $V_{R-} = AV_{SS}$
- 011 $V_{R+} = Ve_{REF+}$ and $V_{R-} = AV_{SS}$
- 100 $V_{R+} = AV_{CC}$ and $V_{R-} = V_{REF-}/ Ve_{REF-}$
- 101 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{REF-}/ Ve_{REF-}$
- 110 $V_{R+} = Ve_{REF+}$ and $V_{R-} = V_{REF-}/ Ve_{REF-}$
- 111 $V_{R+} = Ve_{REF+}$ and $V_{R-} = V_{REF-}/ Ve_{REF-}$

INCHx Bits 3-0 Input channel select
- 0000 A0
- 0001 A1
- 0010 A2
- 0011 A3
- 0100 A4
- 0101 A5
- 0110 A6
- 0111 A7
- 1000 Ve_{REF+}
- 1001 V_{REF-}/Ve_{REF-}
- 1010 Temperature diode
- 1011 $(AV_{CC} – AV_{SS}) / 2$
- 1100 $(AV_{CC} – AV_{SS}) / 2$
- 1101 $(AV_{CC} – AV_{SS}) / 2$
- 1110 $(AV_{CC} – AV_{SS}) / 2$
- 1111 $(AV_{CC} – AV_{SS}) / 2$

ADC12IE, ADC12 Interrupt Enable Register

15	14	13	12	11	10	9	8
ADC12IE15	**ADC12IE14**	**ADC12IE13**	**ADC12IE12**	**ADC12IE11**	**ADC12IE10**	**ADC12IFG9**	**ADC12IE8**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
ADC12IE7	**ADC12IE6**	**ADC12IE5**	**ADC12IE4**	**ADC12IE3**	**ADC12IE2**	**ADC12IE1**	**ADC12IE0**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

ADC12IEx Bits 15-0 Interrupt enable. These bits enable or disable the interrupt request for the ADC12IFGx bits.
- 0 Interrupt disabled
- 1 Interrupt enabled

ADC12IFG, ADC12 Interrupt Flag Register

15	14	13	12	11	10	9	8
ADC12 IFG15	**ADC12 IFG14**	**ADC12 IFG13**	**ADC12 IFG12**	**ADC12 IFG11**	**ADC12 IFG10**	**ADC12 IFG9**	**ADC12 IFG8**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
ADC12 IFG7	**ADC12 IFG6**	**ADC12 IFG5**	**ADC12 IFG4**	**ADC12 IFG3**	**ADC12 IFG2**	**ADC12 IFG1**	**ADC12 IFG0**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

ADC12IFGx Bits 15-0 ADC12MEMx Interrupt flag. These bits are set when corresponding ADC12MEMx is loaded with a conversion result. The ADC12IFGx bits are reset if the corresponding ADC12MEMx is accessed, or may be reset with software.
- 0 No interrupt pending
- 1 Interrupt pending

ADC12IV, ADC12 Interrupt Vector Register

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**ADC12IVx**					**0**
r0	r0	r–(0)	r–(0)	r–(0)	r–(0)	r–(0)	r0

ADC12IVx Bits 15-0 ADC12 interrupt vector value

ADC12IV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No interrupt pending	–	
002h	ADC12MEMx overflow	–	Highest
004h	Conversion time overflow	–	
006h	ADC12MEM0 interrupt flag	ADC12IFG0	
008h	ADC12MEM1 interrupt flag	ADC12IFG1	
00Ah	ADC12MEM2 interrupt flag	ADC12IFG2	
00Ch	ADC12MEM3 interrupt flag	ADC12IFG3	
00Eh	ADC12MEM4 interrupt flag	ADC12IFG4	
010h	ADC12MEM5 interrupt flag	ADC12IFG5	
012h	ADC12MEM6 interrupt flag	ADC12IFG6	
014h	ADC12MEM7 interrupt flag	ADC12IFG7	
016h	ADC12MEM8 interrupt flag	ADC12IFG8	
018h	ADC12MEM9 interrupt flag	ADC12IFG9	
01Ah	ADC12MEM10 interrupt flag	ADC12IFG10	
01Ch	ADC12MEM11 interrupt flag	ADC12IFG11	
01Eh	ADC12MEM12 interrupt flag	ADC12IFG12	
020h	ADC12MEM13 interrupt flag	ADC12IFG13	
022h	ADC12MEM14 interrupt flag	ADC12IFG14	
024h	ADC12MEM15 interrupt flag	ADC12IFG15	Lowest

Chapter 18

ADC10

The ADC10 module is a high-performance 10-bit analog-to-digital converter. This chapter describes the ADC10. The ADC10 is implemented in the MSP430x11x2, MSP430x12x2 devices.

Topic **Page**

18.1 ADC10 Introduction

The ADC10 module supports fast, 10-bit analog-to-digital conversions. The module implements a 10-bit SAR core, sample select control, reference generator, and data transfer controller (DTC).

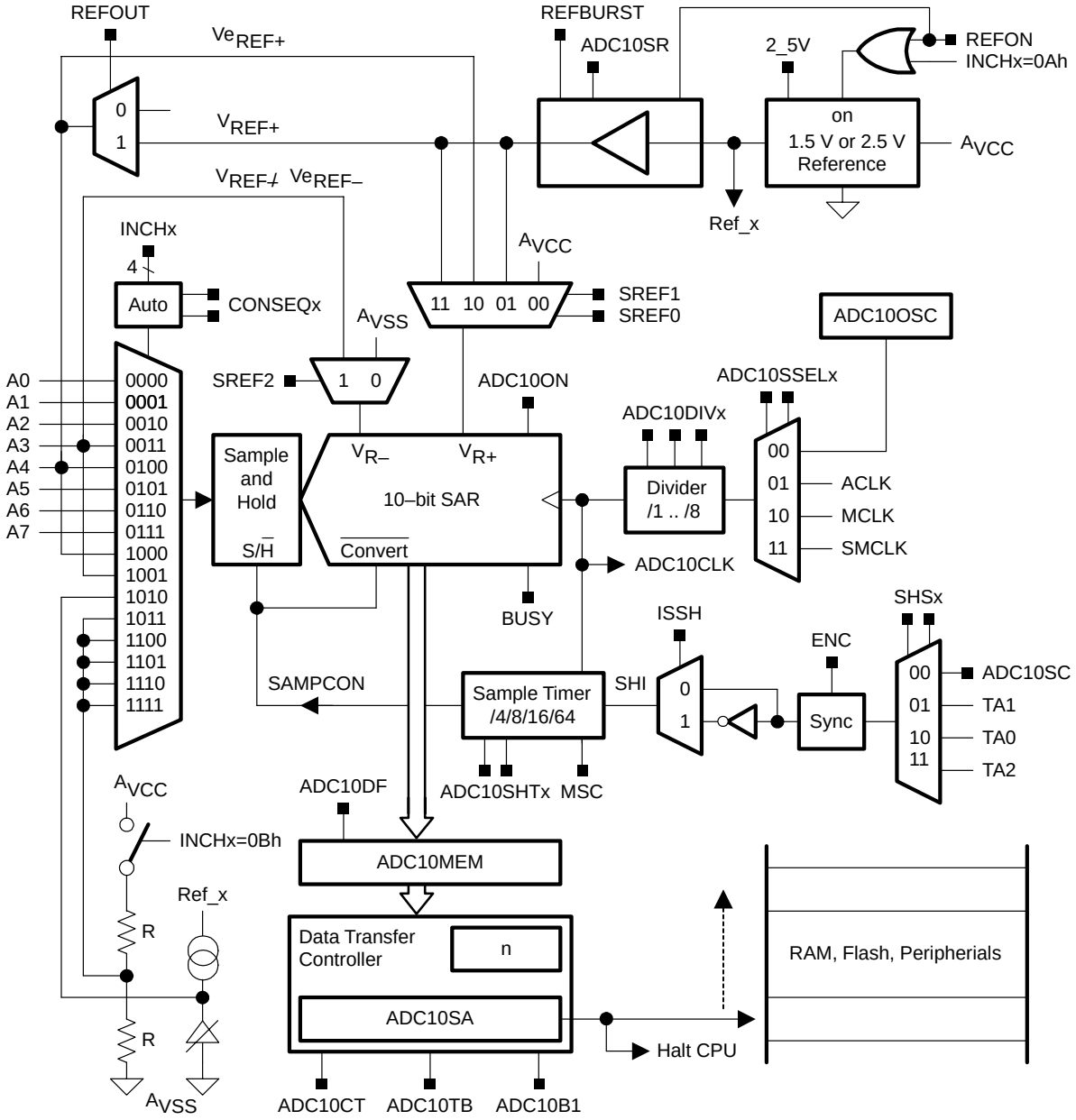
The DTC allows ADC10 samples to be converted and stored anywhere in memory without CPU intervention. The module can be configured with user software to support a variety of applications.

ADC10 features include:

- Greater than 200-ksps maximum conversion rate
- Monotonic10-bit converter with no missing codes
- Sample-and-hold with programmable sample periods
- Conversion initiation by software or Timer_A
- Software selectable on-chip reference voltage generation (1.5 V or 2.5 V)
- Software selectable internal or external reference
- Eight external input channels
- Conversion channels for internal temperature sensor, AV_{CC}, and external references
- Selectable conversion clock source
- Single-channel, repeated single-channel, sequence, and repeated sequence conversion modes
- ADC core and reference voltage can be powered down separately
- Data transfer controller for automatic storage of conversion results

The block diagram of ADC10 is shown in Figure 18–1.

Figure 18–1. ADC10 Block Diagram



18.2 ADC10 Operation

The ADC10 module is configured with user software. The setup and operation of the ADC10 is discussed in the following sections.

18.2.1 10-Bit ADC Core

The ADC core converts an analog input to its 10-bit digital representation and stores the result in the ADC10MEM register. The core uses two programmable/selectable voltage levels (V_{R+} and V_{R-}) to define the upper and lower limits of the conversion. The digital output (N_{ADC}) is full scale (03FFh) 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. Conversion results may be in straight binary format or 2s-complement format. The conversion formula for the ADC result when using straight binary format is:

$$N_{ADC} = 1023 \times \frac{Vin - V_{R-}}{V_{R+} - V_{R-}}$$

The ADC10 core is configured by two control registers, ADC10CTL0 and ADC10CTL1. The core is enabled with the ADC10ON bit. With few exceptions the ADC10 control bits can only be modified when ENC = 0. ENC must be set to 1 before any conversion can take place.

Conversion Clock Selection

The ADC10CLK is used both as the conversion clock and to generate the sampling period. The ADC10 source clock is selected using the ADC10SSELx bits and can be divided from 1-8 using the ADC10DIVx bits. Possible ADC10CLK sources are SMCLK, MCLK, ACLK and an internal oscillator ADC10OSC .

The ADC10OSC, generated internally, is in the 5-MHz range, but varies with individual devices, supply voltage, and temperature. See the device-specific data sheet for the ADC10OSC specification.

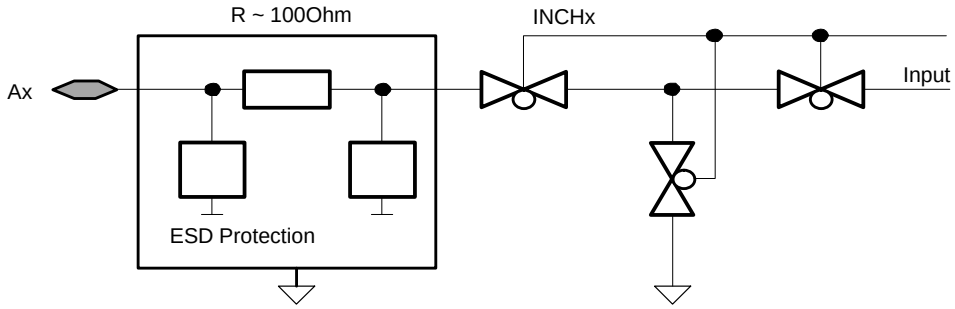
The user must ensure that the clock chosen for ADC10CLK remains active until the end of a conversion. If the clock is removed during a conversion, the operation will not complete, and any result will be invalid.

18.2.2 ADC10 Inputs and Multiplexer

The eight external and four internal analog signals are selected as the channel for conversion by the analog input multiplexer. The input multiplexer is a break-before-make type to reduce input-to-input noise injection resulting from channel switching as shown in Figure 18–2. 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.

The ADC10 uses the charge redistribution method. When the inputs are internally switched, the switching action may cause transients on the input signal. These transients decay and settle before causing errant conversion.

Figure 18–2. Analog Multiplexer



Analog Port Selection

The ADC10 external inputs A0 to A4 and Ve_{REF+} and $V_{REF–}$ share terminals with I/O port P2, which are digital CMOS gates. Optional inputs A5 to A7 are shared on port P3 on selected devices (see device-specific data sheet). When analog signals are applied to digital CMOS gates, parasitic current can flow from V_{CC} to GND. This parasitic current occurs if the input voltage is near the transition level of the gate. Disabling the port pin buffer eliminates the parasitic current flow and therefore reduces overall current consumption. The ADC10AEx bits provide the ability to disable the port pin input buffer.

```
; P2.3 configured for analog input
      BIS.B #4h,&ADC10AE ; P2.3 ADC10 function and enable
      BIC.B #4h,&P2DIR   ; P2.3 input direction
```

18.2.3 Voltage Reference Generator

The ADC10 module contains a built-in voltage reference with two selectable voltage levels, 1.5 V and 2.5 V. Either of these reference voltages may be used internally and externally on pin V_{REF+}.

Setting REFON = 1 enables the internal reference. When REF2_5V = 1, the internal reference is 2.5 V, the reference is 1.5 V when REF2_5V = 0.

External references may be supplied for V_{R+} and V_{R-} through pins A4 and A3 respectively.

Low Power Applications

The ADC10 internal voltage reference generator is designed for low power applications, with specific features for a fast startup. For proper operation, an external storage capacitance is not required and has no associated bias time. The total reference turn on time is less than 30 μs. Normal power supply decoupling across V_{CC} and V_{SS} using a parallel combination of 10-μF and 100-nF capacitors are all that is required.

- When using V_{CC} and V_{SS} as reference voltages, the internal reference should be powered off completely with REFON = 0.
- When using an external reference, the internal reference should be powered off completely. External references may be supplied for V_{R+} and V_{R-} through pins A4 and A3 respectively.
- When the internal reference is used, and the maximum conversion rate is below 50 ksps, setting ADC10SR = 1 reduces the current consumption of the internal reference buffer approximately 50%.
- When both REFOUT = 1 and REFBURST = 1, the reference is present externally only during the sample and conversion period. When REFOUT = 1, and REFBURST = 0 is cleared, the reference voltage is continuously present externally.

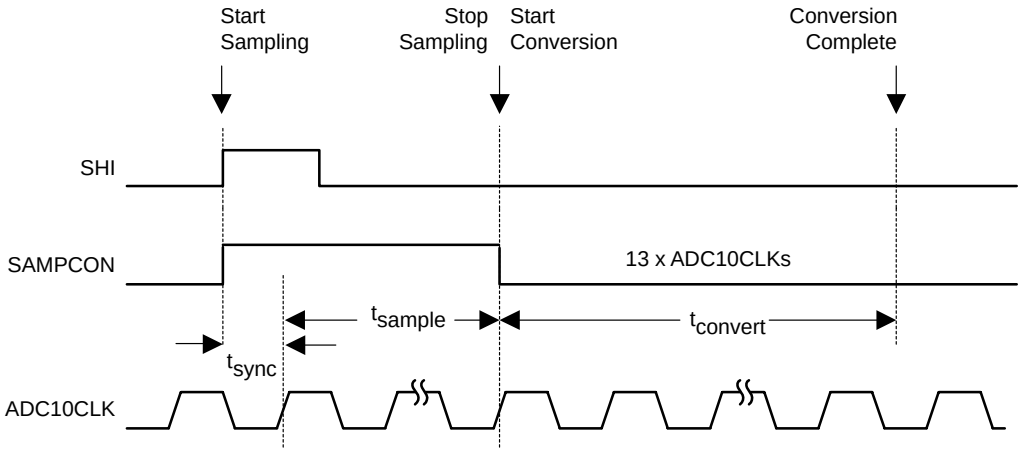
18.2.4 Sample and Conversion Timing

An analog-to-digital conversion is initiated with a rising edge of sample input signal SHI. The source for SHI is selected with the SHSx bits and includes the following:

- The ADC10SC bit
- The Timer_A Output Unit 1
- The Timer_A Output Unit 0
- The Timer_A Output Unit 2

The polarity of the SHI signal source can be inverted with the ISSH bit. The SHTx bits select the sample period t_{sample} to be 4, 8, 16, or 64 ADC10CLK cycles. The sampling timer sets SAMPCON high for the selected sample period after synchronization with ADC10CLK. Total sampling time is t_{sample} plus t_{sync}.The high-to-low SAMPCON transition starts the analog-to-digital conversion, which requires 13 ADC10CLK cycles as shown in Figure 18–3.

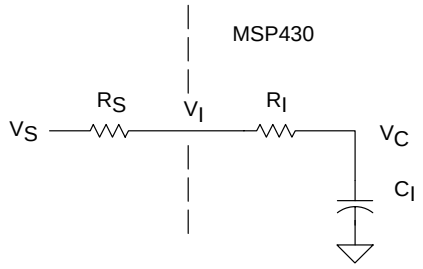
Figure 18–3. Sample Timing



Sample Timing Considerations

When SAMPCON = 0 all Ax inputs are high impedance. When SAMPCON = 1, the selected Ax input can be modeled as an RC low-pass filter during the sampling time t_{sample}, as shown below in Figure 18–4. An internal MUX-on input resistance R_I (max. 2 kΩ) in series with capacitor C_I (max. 20 pF) is seen by the source. The capacitor C_I voltage V_C must be charged to within ½ LSB of the source voltage V_S for an accurate 10-bit conversion.

Figure 18–4. Analog Input Equivalent Circuit



V_I = Input voltage at pin Ax
V_S = External source voltage
R_S = External source resistance
R_I = Internal MUX-on input resistance
C_I = Input capacitance
V_C = Capacitance-charging voltage

The resistance of the source R_S and R_I affect t_{sample}.The following equation can be used to calculate the minimum sampling time t_{sample} for a 10-bit conversion:

$$t_{sample} > (R_S + R_I) \times \ln(2^{11}) \times C_I \quad (1)$$

Substituting the values for R_I and C_I given above, the equation becomes:

$$t_{sample} > (R_S + 2k) \times 7.625 \times 20pF \quad (2)$$

For example, if R_S is 10 kΩ, t_{sample} must be greater than 1.83 μs.

18.2.5 Conversion Modes

The ADC10 has four operating modes selected by the CONSEQx bits as discussed in Table 18–1.

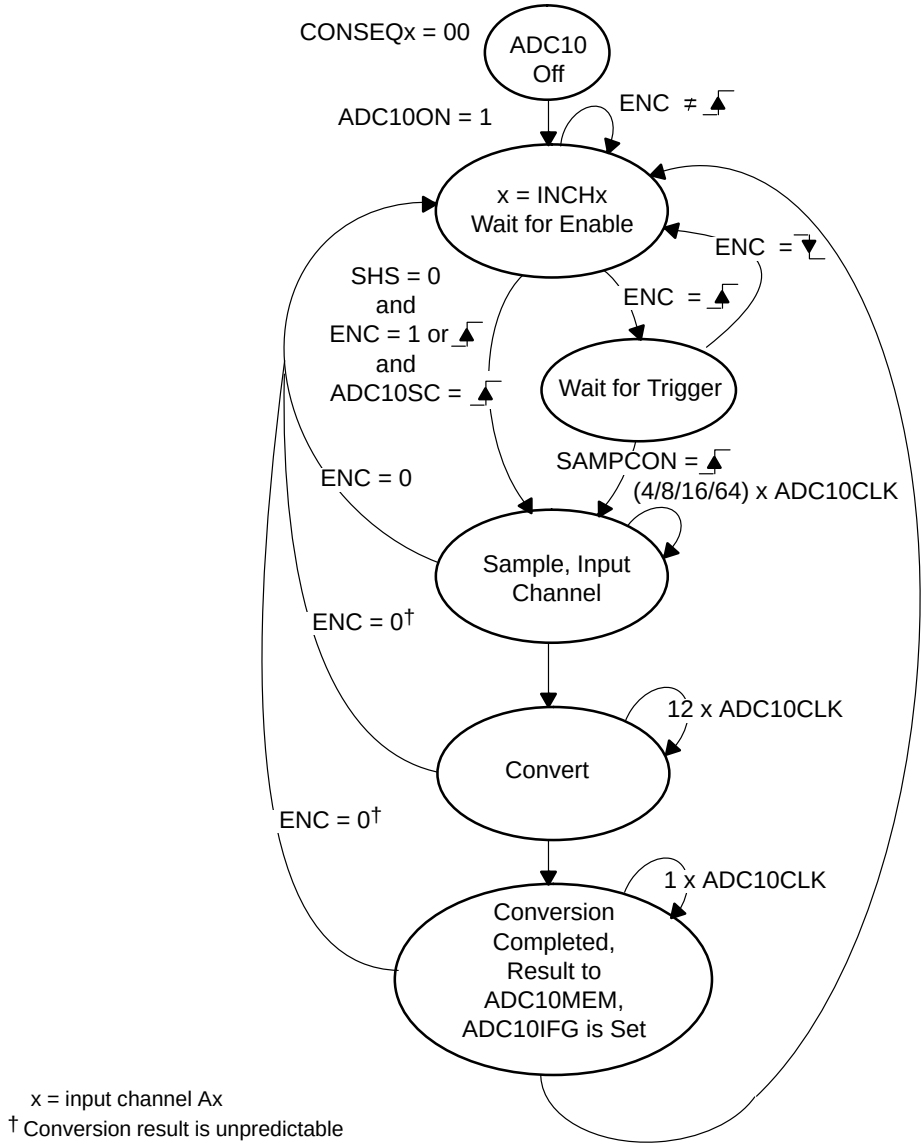
Table 18–1. Conversion Mode Summary

CONSEQx	MODE	OPERATION
00	Single channel single-conversion	A single channel is converted once.
01	Sequence-of-channels	A sequence of channels is converted once.
10	Repeat single channel	A single channel is converted repeatedly.
11	Repeat sequence-of-channels	A sequence of channels is converted repeatedly.

Single-Channel Single-Conversion Mode

A single channel selected by INCHx is sampled and converted once. The ADC result is written to ADC10MEM. Figure 18–5 shows the flow of the single-channel, single-conversion mode. When ADC10SC triggers a conversion, successive conversions can be triggered by the ADC10SC bit. When any other trigger source is used, ENC must be toggled between each conversion.

Figure 18–5. Single-Channel Single-Conversion Mode

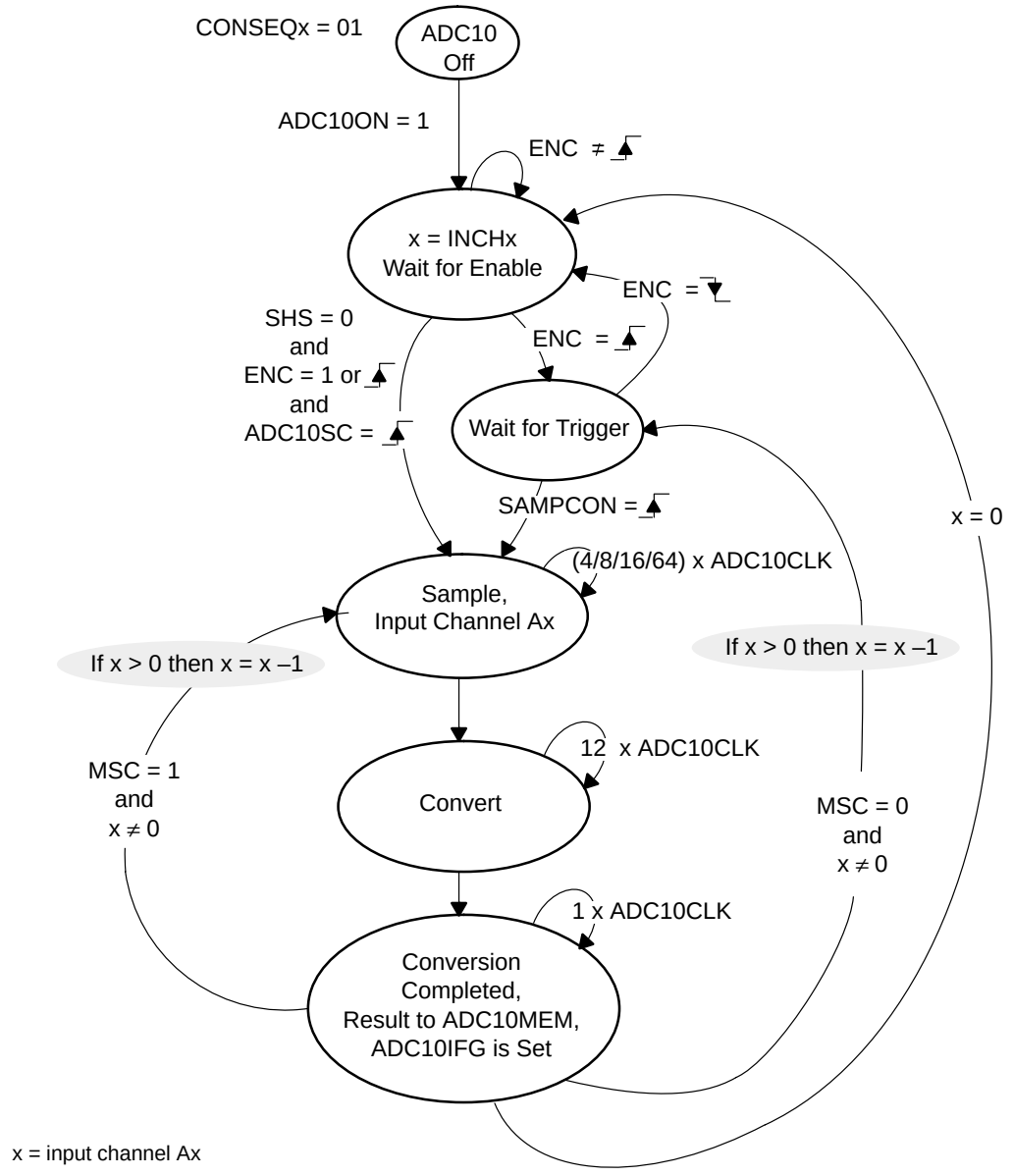


x = input channel Ax
† Conversion result is unpredictable

Sequence-of-Channels Mode

A sequence of channels is sampled and converted once. The sequence begins with the channel selected by INCHx and decrements to channel A0. Each ADC result is written to ADC10MEM. The sequence stops after conversion of channel A0. Figure 18–6 shows the sequence-of-channels mode. When ADC10SC triggers a sequence, successive sequences can be triggered by the ADC10SC bit . When any other trigger source is used, ENC must be toggled between each sequence.

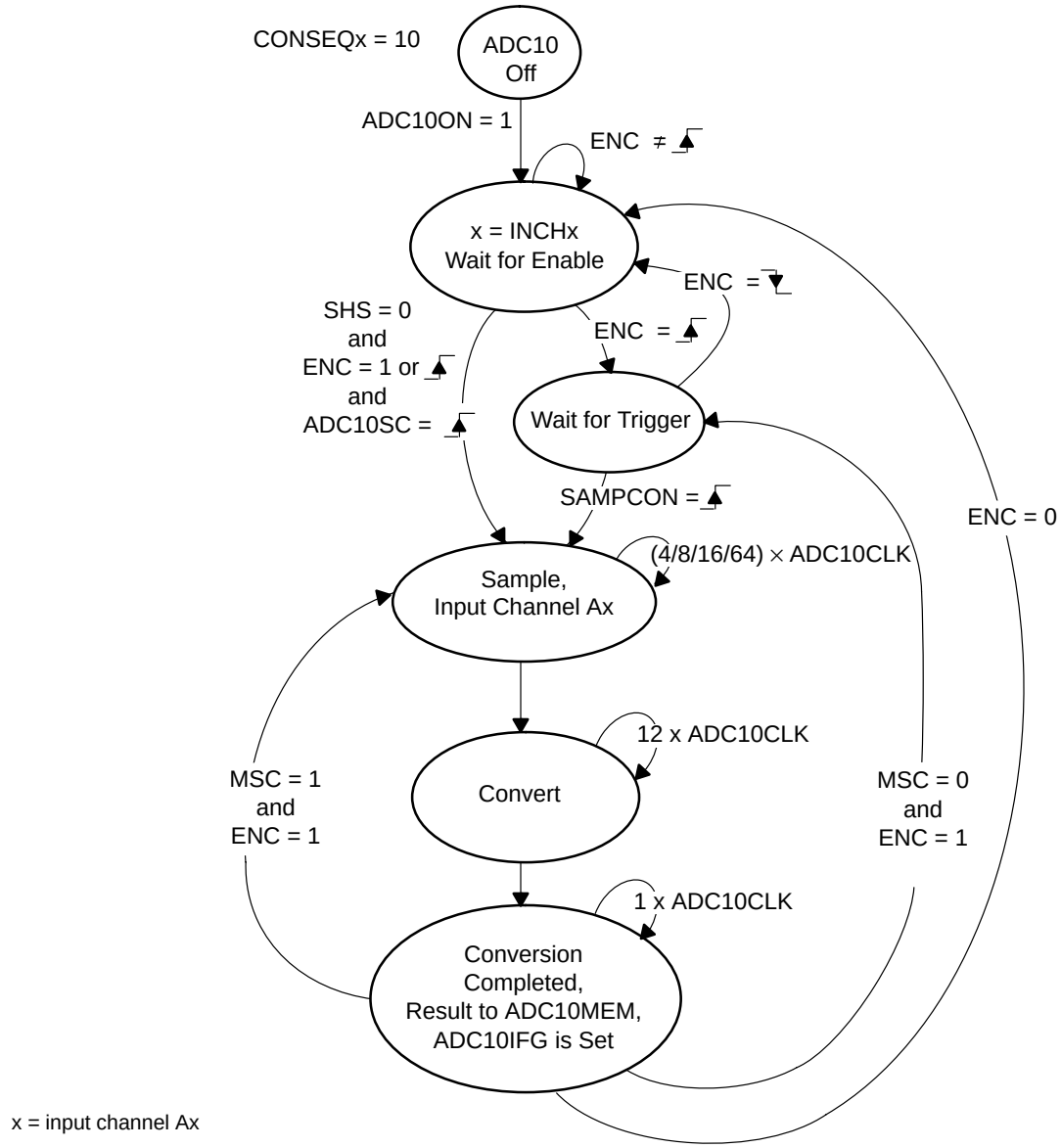
Figure 18–6. Sequence-of-Channels Mode



Repeat-Single-Channel Mode

A single channel selected by INCHx is sampled and converted continuously. Each ADC result is written to ADC10MEM. Figure 18–7 shows the repeat-single-channel mode.

Figure 18–7. Repeat-Single-Channel Mode

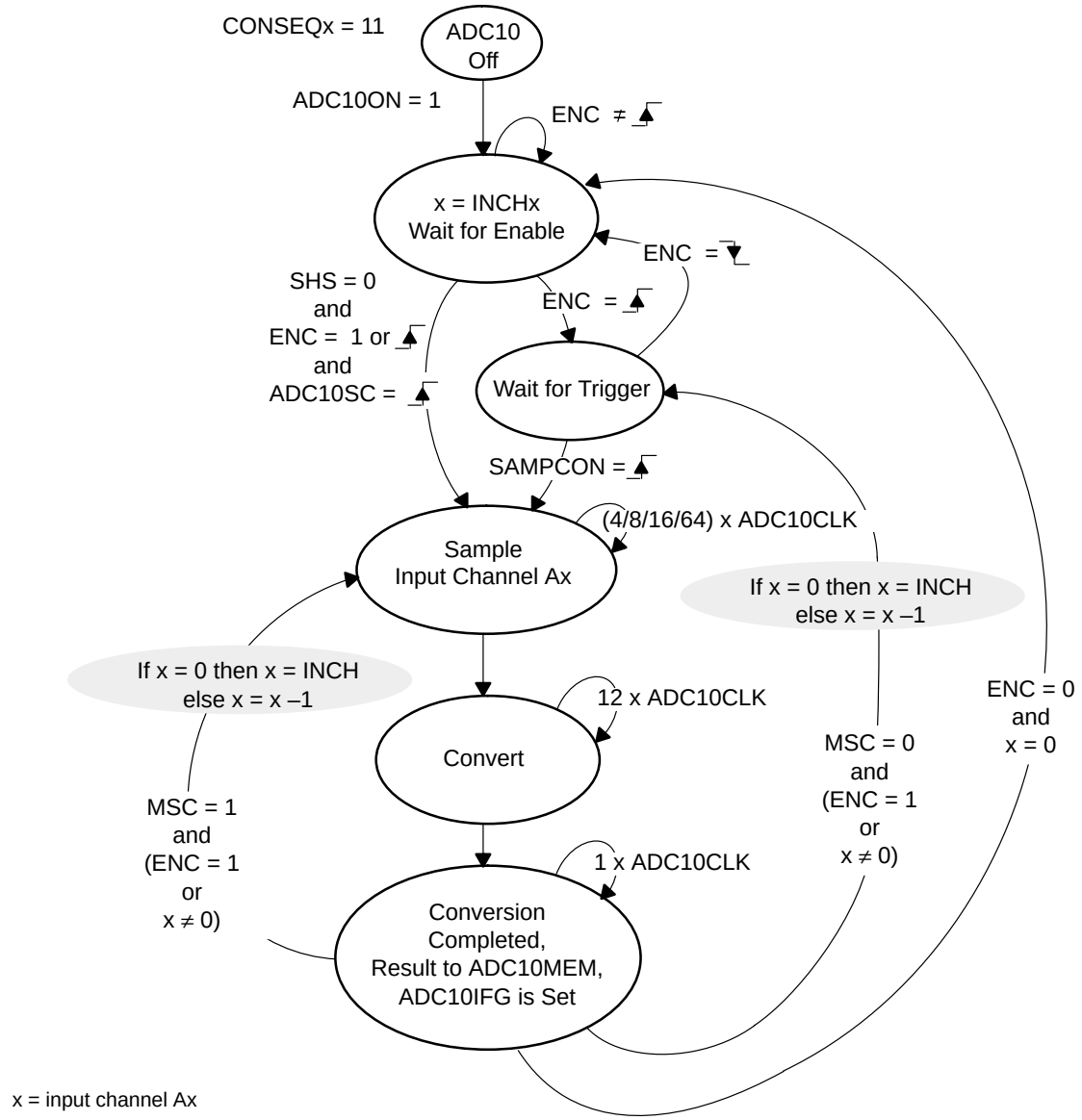


x = input channel Ax

Repeat-Sequence-of-Channels Mode

A sequence of channels is sampled and converted repeatedly. The sequence begins with the channel selected by INCHx and decrements to channel A0. Each ADC result is written to ADC10MEM. The sequence ends after conversion of channel A0, and the next trigger signal re-starts the sequence. Figure 18–8 shows the repeat-sequence-of-channels mode.

Figure 18–8. Repeat-Sequence-of-Channels Mode



Using the MSC Bit

To configure the converter to perform successive conversions automatically and as quickly as possible, a multiple sample and convert function is available. When MSC = 1 and CONSEQx > 0 the first rising edge of the SHI signal triggers the first conversion. 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 in the single-sequence mode or until the ENC bit is toggled in repeat-single-channel, or repeated-sequence modes. The function of the ENC bit is unchanged when using the MSC bit.

Stopping Conversions

Stopping ADC10 activity depends on the mode of operation. The recommended ways to stop an active conversion or conversion sequence are:

- Resetting ENC in single-channel single-conversion mode stops a conversion immediately and the results are unpredictable. For correct results, poll the ADC10BUSY bit until reset before clearing ENC.
- Resetting ENC during repeat-single-channel operation stops the converter at the end of the current conversion.
- Resetting ENC during a sequence or repeat sequence mode stops the converter at the end of the sequence.
- Any conversion mode may be stopped immediately by setting the CONSEQx=0 and resetting the ENC bit. Conversion data is unreliable.

18.2.6 ADC10 Data Transfer Controller

The ADC10 includes a data transfer controller (DTC) to automatically transfer conversion results from ADC10MEM to other on-chip memory locations. The DTC is enabled by setting the ADC10DTC1 register to a nonzero value.

When the DTC is enabled, each time the ADC10 completes a conversion and loads the result to ADC10MEM, a data transfer is triggered. No software intervention is required to manage the ADC10 until the predefined amount of conversion data has been transferred. Each DTC transfer requires one CPU MCLK. To avoid any bus contention during the DTC transfer, the CPU is halted, if active, for the one MCLK required for the transfer.

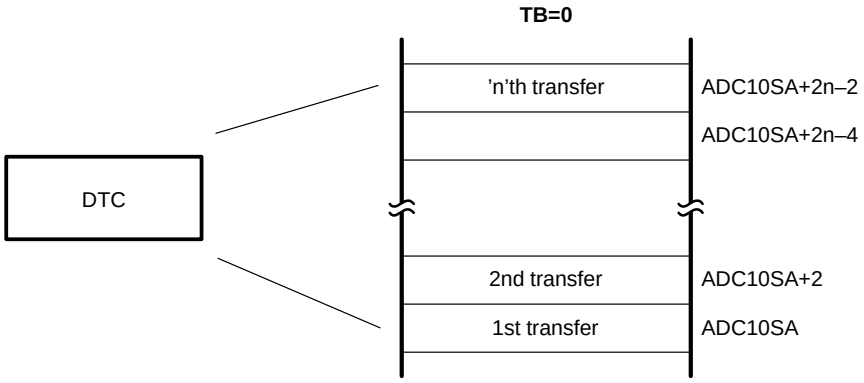
A DTC transfer must not be initiated while the ADC10 is busy. Software must ensure that no active conversion or sequence is in progress when the DTC is configured:

```
; ADC10 activity test
          BIC.W #ENC,&ADC10CTL0 ;
busy_test BIT.W #BUSY,&ADC10CTL1;
          JNZ   busy_test       ;
          MOV.W #xxx,&ADC10SA   ; Safe
          MOV.B #xx,&ADC10DTC1  ;
; continue setup
```

One-Block Transfer Mode

The one-block mode is selected if the ADC10TB is reset. The value n in ADC10DTC1 defines the total number of transfers for a block. The block start address is defined anywhere in the MSP430 address range using the 16-bit register ADC10SA. The block ends at ADC10SA+2n–2. The one-block transfer mode is shown in Figure 18–9.

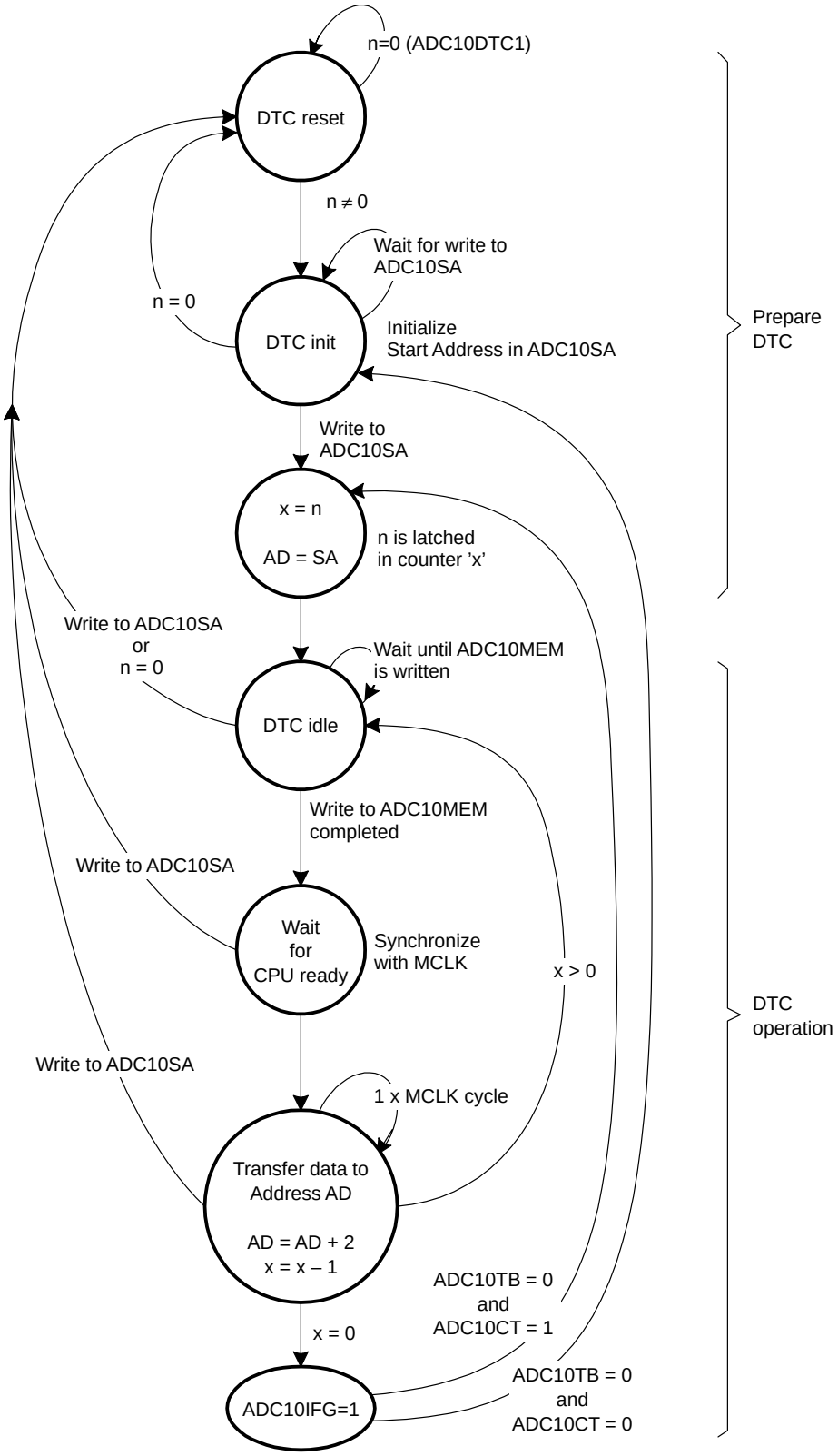
Figure 18–9. One-Block Transfer



The internal address pointer is initially equal to ADC10SA and the internal transfer counter is initially equal to 'n'. The internal pointer and counter are not visible to software. The DTC transfers the word-value of ADC10MEM to the address pointer ADC10SA. After each DTC transfer, the internal address pointer is incremented by two and the internal transfer counter is decremented by one.

The DTC transfers continue with each loading of ADC10MEM, until the internal transfer counter becomes equal to zero. No additional DTC transfers will occur until a write to ADC10SA. When using the DTC in the one-block mode, the ADC10IFG flag is set only after a complete block has been transferred. Figure 18–10 shows a state diagram of the one-block mode.

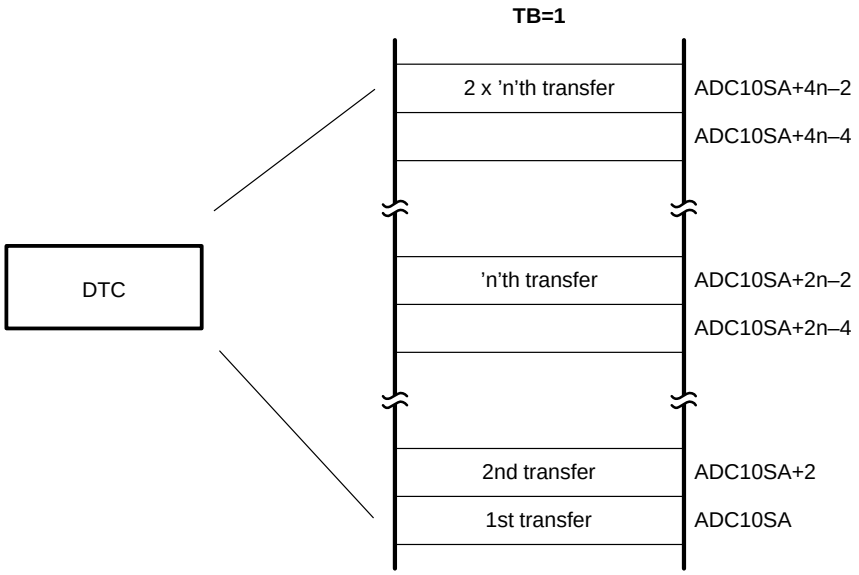
Figure 18–10. State Diagram for Data Transfer Control in One-Block Transfer Mode



Two-Block Transfer Mode

The two-block mode is selected if the ADC10TB bit is set. The value n in ADC10DTC1 defines the number of transfers for one block. The address range of the first block is defined anywhere in the MSP430 address range with the 16-bit register ADC10SA. The first block ends at ADC10SA+2n–2. The address range for the second block is defined as SA+2n to SA+4n–2. The two-block transfer mode is shown in Figure 18–11**.**

Figure 18–11. Two-Block Transfer

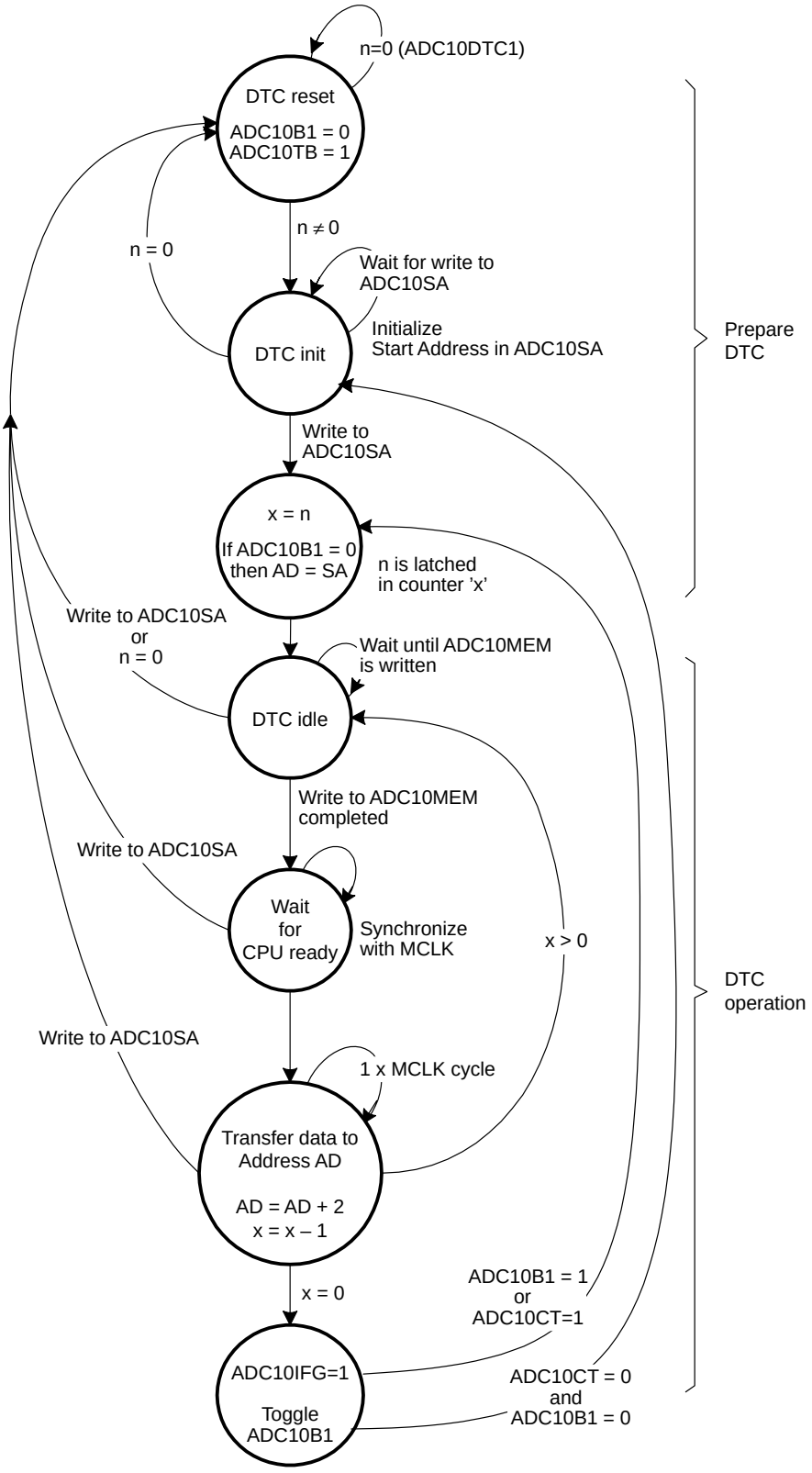


The internal address pointer is initially equal to ADC10SA and the internal transfer counter is initially equal to ‘n’. The internal pointer and counter are not visible to software. The DTC transfers the word-value of ADC10MEM to the address pointer ADC10SA. After each DTC transfer the internal address pointer is incremented by two and the internal transfer counter is decremented by one.

The DTC transfers continue, with each loading of ADC10MEM, until the internal transfer counter becomes equal to zero. At this point, block one is full and both the ADC10IFG flag the ADC10B1 bit are set. The user can test the ADC10B1 bit to determine that block one is full.

The DTC continues with block two. The internal transfer counter is automatically reloaded with ’n’. At the next load of the ADC10MEM, the DTC begins transferring conversion results to block two. After n transfers have completed, block two is full. The ADC10IFG flag is set and the ADC10B1 bit is cleared. User software can test the cleared ADC10B1 bit to determine that block two is full. Figure 18–12 shows a state diagram of the two-block mode.

Figure 18–12. State Diagram for Data Transfer Control in Two-Block Transfer Mode



Continuous Transfer

A continuous transfer is selected if ADC10CT bit is set. The DTC will not stop after block one in (one-block mode) or block two (two-block mode) has been transferred. The internal address pointer and transfer counter are set equal to ADC10SA and n respectively. Transfers continue starting in block one. If the ADC10CT bit is reset, DTC transfers cease after the current completion of transfers into block one (in the one-block mode) or block two (in the two-block mode) have been transfer.

DTC Transfer Cycle Time

For each ADC10MEM transfer, the DTC requires one or two MCLK clock cycles to synchronize, one for the actual transfer (while the CPU is halted), and one cycle of wait time. Because the DTC uses MCLK, the DTC cycle time is dependent on the MSP430 operating mode and clock system setup.

If the MCLK source is active, but the CPU is off, the DTC uses the MCLK source for each transfer, without re-enabling the CPU. If the MCLK source is off, the DTC temporarily restarts MCLK, sourced with DCOCLK, only during a transfer. The CPU remains off and after the DTC transfer, MCLK is again turned off. The maximum DTC cycle time for all operating modes is show in Table 18–2.

Table 18–2. Maximum DTC Cycle Time

CPU Operating Mode	Clock Source	Maximum DTC Cycle Time
Active mode	MCLK=DCOCLK	3 MCLK cycles
Active mode	MCLK=LFXT1CLK	3 MCLK cycles
Low-power mode LPM0/1	MCLK=DCOCLK	4 MCLK cycles
Low-power mode LPM3/4	MCLK=DCOCLK	4 MCLK cycles + 6 μs†
Low-power mode LPM0/1	MCLK=LFXT1CLK	4 MCLK cycles
Low-power mode LPM3	MCLK=LFXT1CLK	4 MCLK cycles
Low-power mode LPM4	MCLK=LFXT1CLK	4 MCLK cycles + 6 μs†

† The additional 6 μs are needed to start the DCOCLK. It is the $t_{(LPMx)}$ parameter in the data sheet.

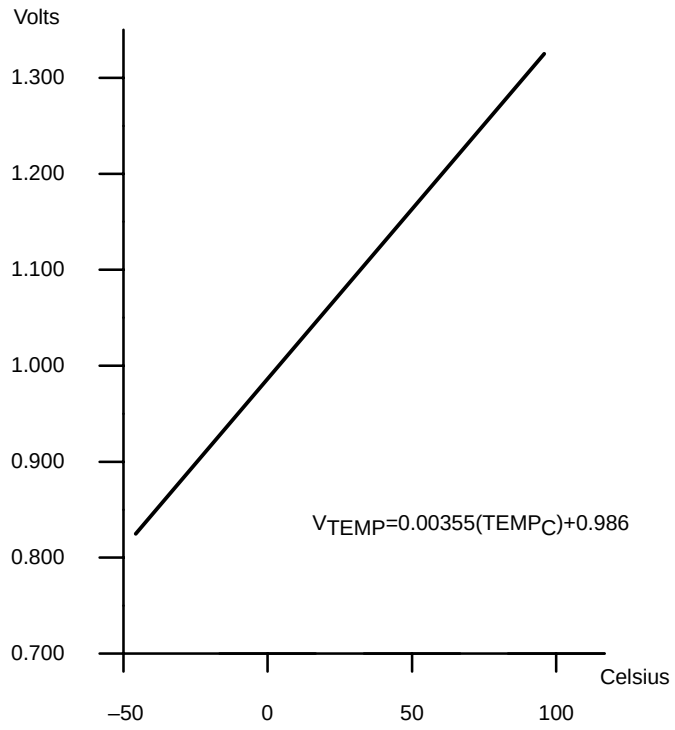
18.2.7 Using the Integrated Temperature Sensor

To use the on-chip temperature sensor, the user selects the analog input channel INCHx = 1010. Any other configuration is done as if an external channel was selected, including reference selection, conversion-memory selection, etc.

The typical temperature sensor transfer function is shown in Figure 18–13. When using the temperature sensor, the sample period must be greater than 30 μs. The temperature sensor offset error can be large, and may need to be calibrated for most applications. See the device-specific data sheet for the parameters.

Selecting the temperature sensor automatically turns on the on-chip reference generator as a voltage source for the temperature sensor. However, it does not enable the V_{REF+} output or affect the reference selections for the conversion. The reference choices for converting the temperature sensor are the same as with any other channel.

Figure 18–14. Typical Temperature Sensor Transfer Function



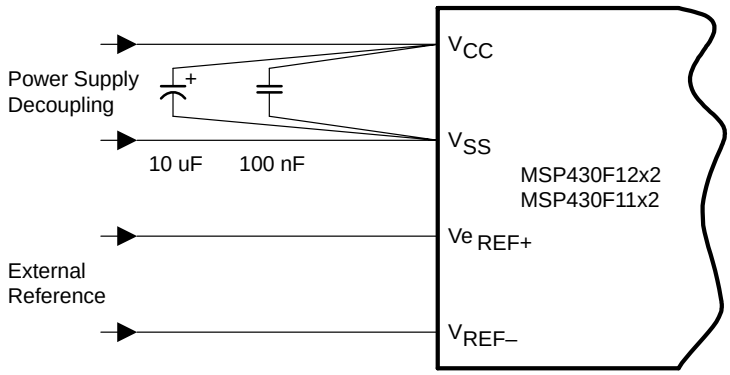
18.2.8 A/D Grounding and Noise Considerations

As with any high-resolution ADC, appropriate printed-circuit-board layout and grounding techniques should be followed to eliminate ground loops, unwanted parasitic effects, and noise.

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. The connections shown in Figure 18–15 help avoid this.

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. A noise-free design is important to achieve high accuracy.

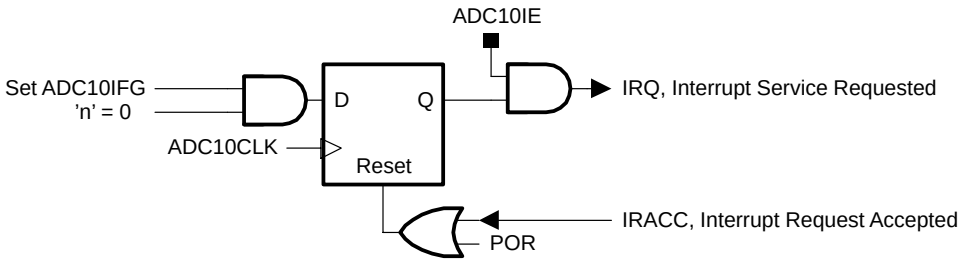
Figure 18–16. ADC10 Grounding and Noise Considerations



18.2.9 ADC10 Interrupts

One interrupt and one interrupt vector are associated with the ADC10 as shown in Figure 18–17. When the DTC is not used (ADC10DTC1 = 0) ADC10IFG is set when conversion results are loaded into ADC10MEM. When DTC is used (ADC10DTC1 > 0) ADC10IFG is set when a block transfer completes and the internal transfer counter 'n' = 0. If both the ADC10IE and the GIE bits are set, then the ADC10IFG flag generates an interrupt request. The ADC10IFG flag is automatically reset when the interrupt request is serviced or may be reset by software.

Figure 18–17. ADC10 Interrupt System



18.3 ADC10 Registers

The ADC10 registers are listed in Table 18–3.

Table 18–3.ADC10 Registers

Register	Short Form	Register Type	Address	Initial State
ADC10 Input enable register	ADC10AE	Read/write	04Ah	Reset with POR
ADC10 control register 0	ADC10CTL0	Read/write	01B0h	Reset with POR
ADC10 control register 1	ADC10CTL1	Read/write	01B2h	Reset with POR
ADC10 memory	ADC10MEM	Read	01B4h	Unchanged
ADC10 data transfer control register 0	ADC10DTC0	Read/write	048h	Reset with POR
ADC10 data transfer control register 1	ADC10DTC1	Read/write	049h	Reset with POR
ADC10 data transfer start address	ADC10SA	Read/write	01BCh	0200h with POR

ADC10CTL0, ADC10 Control Register 0

15	14	13	12	11	10	9	8
SREFx			**ADC10SHTx**		**ADC10SR**	**REFOUT**	**REFBURST**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
MSC	**REF2_5V**	**REFON**	**ADC10ON**	**ADC10IE**	**ADC10IFG**	**ENC**	**ADC10SC**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

Modifiable only when ENC = 0

SREFx Bits 15-13 Select reference
- 000 $V_{R+} = AV_{CC}$ and $V_{R-} = AV_{SS}$
- 001 $V_{R+} = V_{REF+}$ and $V_{R-} = AV_{SS}$
- 010 $V_{R+} = Ve_{REF+}$ and $V_{R-} = AV_{SS}$
- 011 $V_{R+} = Ve_{REF+}$ and $V_{R-} = AV_{SS}$
- 100 $V_{R+} = AV_{CC}$ and $V_{R-} = V_{REF-}/ Ve_{REF-}$
- 101 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{REF-}/ Ve_{REF-}$
- 110 $V_{R+} = Ve_{REF+}$ and $V_{R-} = V_{REF-}/ Ve_{REF-}$
- 111 $V_{R+} = Ve_{REF+}$ and $V_{R-} = V_{REF-}/ Ve_{REF-}$

ADC10 SHTx Bits 12-11 ADC10 sample-and-hold time
- 00 4 x ADC10CLKs
- 01 8 x ADC10CLKs
- 10 16 x ADC10CLKs
- 11 64 x ADC10CLKs

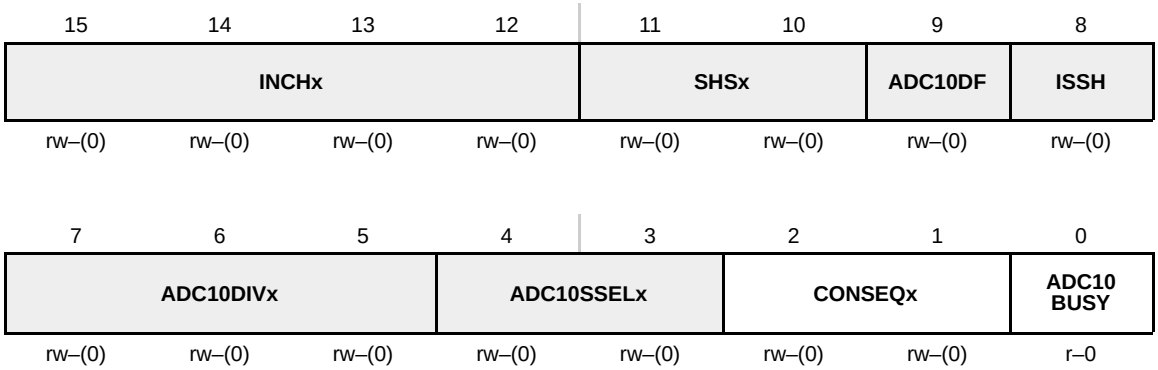
ADC10SR Bit 10 ADC10 sampling rate. This bit selects the approximate maximum sample rate of the ADC10.
- 0 ~200 ksps
- 1 ~50 ksps

REFOUT Bit 9 Reference output
- 0 Reference output off
- 1 Reference output on

REFBURST Bit 8 Reference burst. REFOUT must also be set.
- 0 Reference voltage output continuously
- 1 Reference voltage output only during sample-and-conversion

MSC	Bit 7	Multiple sample and conversion. Valid only for sequence or repeated modes. 0 The sampling 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-conversions are performed automatically as soon as the prior conversion is completed
REF2_5V	Bit 6	Reference-generator voltage. REFON must also be set. 0 1.5 V 1 2.5 V
REFON	Bit 5	Reference generator on 0 Reference off 1 Reference on
ADC10ON	Bit 4	ADC10 on 0 ADC10 off 1 ADC10 on
ADC10IE	Bit 3	ADC10 interrupt enable 0 Interrupt disabled 1 interrupt enabled
ADC10IFG	Bit 2	ADC10 interrupt flag. This bit is set if ADC10MEM is loaded with a conversion result. It is automatically reset when the interrupt request is accepted, or it may be reset by software. When using the DTC this flag is set when a block of transfers is completed. 0 No interrupt pending 1 Interrupt pending
ENC	Bit 1	Enable conversion 0 ADC10 disabled 1 ADC10 enabled
ADC10SC	Bit 0	Start conversion. Software-controlled sample-and-conversion start. ADC10SC and ENC may be set together with one instruction. ADC10SC is reset automatically. 0 No sample-and-conversion start 1 Start sample-and-conversion

ADC10CTL1, ADC10 Control Register 1



Modifiable only when ENC = 0

INCHx Bits 15-12 — Input channel select

0000	A0
0001	A1
0010	A2
0011	A3
0100	A4
0101	A5
0110	A6
0111	A7
1000	Ve_{REF+}
1001	V_{REF-}/Ve_{REF-}
1010	Temperature diode
1011	$(AV_{CC} – AV_{SS}) / 2$
1100	$(AV_{CC} – AV_{SS}) / 2$
1101	$(AV_{CC} – AV_{SS}) / 2$
1110	$(AV_{CC} – AV_{SS}) / 2$
1111	$(AV_{CC} – AV_{SS}) / 2$

SHSx Bits 11-10 — Sample-and-hold source select

00	ADC10SC bit
10	Timer_A.OUT1
10	Timer_A.OUT0
11	Timer_A.OUT2

ADC10DF Bit 9 — ADC10 data format

0	Straight binary
1	2’s complement

ISSH Bit 8 — Invert signal sample-and-hold

0	The sample-input signal is not inverted.
1	The sample-input signal is inverted.

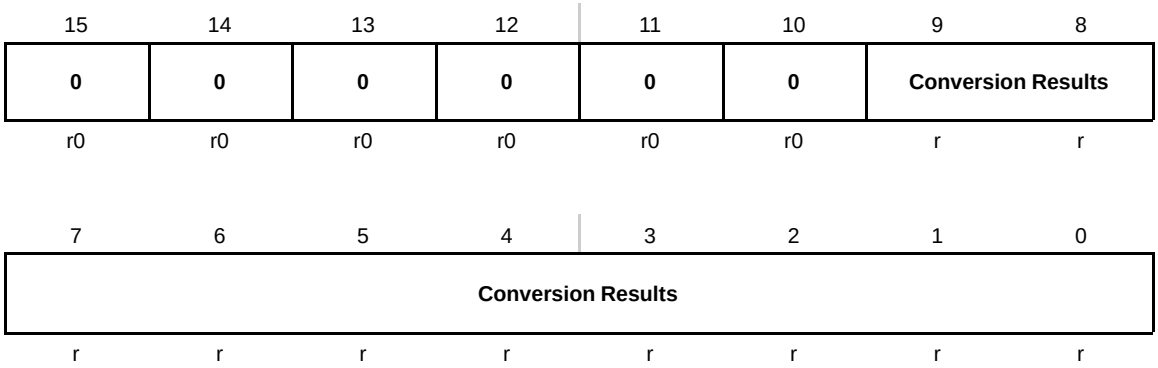
ADC10DIVx	Bits 7-5	ADC10 clock divider 000 /1 001 /2 010 /3 011 /4 100 /5 101 /6 110 /7 111 /8
ADC10 SSELx	Bits 4-3	ADC10 clock source select 00 ADC10OSC 01 ACLK 10 MCLK 11 SMCLK
CONSEQx	Bits 2-1	Conversion sequence mode select 00 Single-channel-single-conversion 01 Sequence-of-channels 10 Repeat-single-channel 11 Repeat-sequence-of-channels
ADC10 BUSY	Bit 0	ADC10 busy. This bit indicates an active sample or conversion operation 0 No operation is active. 1 A sequence, sample, or conversion is active.

ADC10AE, Analog (Input) Enable Control Register

7	6	5	4	3	2	1	0
ADC10AE7	**ADC10AE6**	**ADC10AE5**	**ADC10AE4**	**ADC10AE3**	**ADC10AE2**	**ADC10AE1**	**ADC10AE0**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

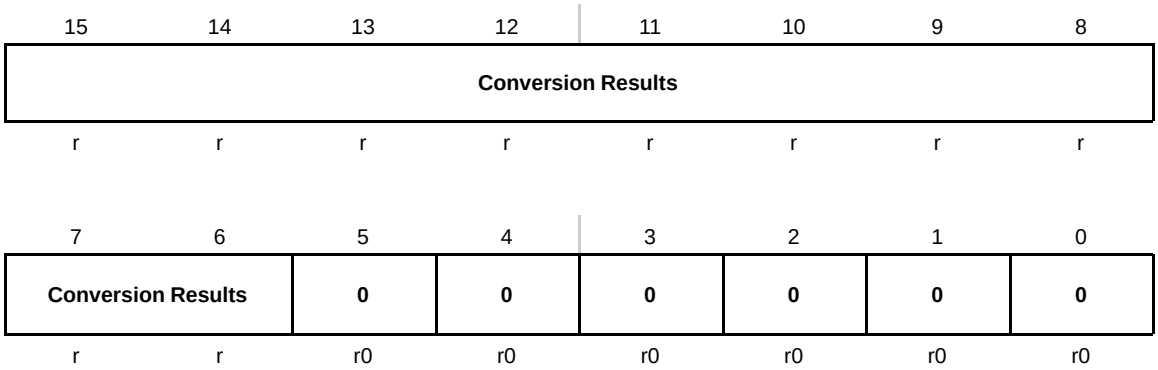
ADC10AEx	Bits 7-0	ADC10 analog enable 0 Analog input disabled 1 Analog input enabled

ADC10MEM, Conversion-Memory Register, Binary Format



Conversion Results	Bits 15-0	The 10-bit conversion results are right justified, straight-binary format. Bit 9 is the MSB. Bits 15-10 are always 0.

ADC10MEM, Conversion-Memory Register, 2's Complement Format



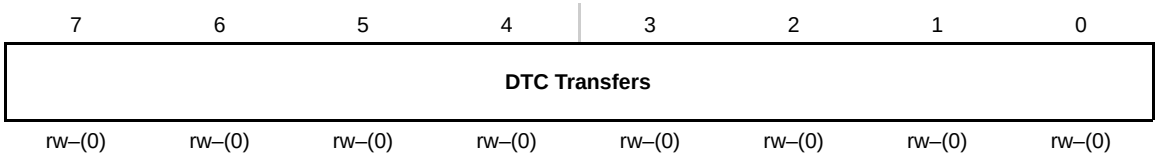
Conversion Results	Bits 15-0	The 10-bit conversion results are left-justified, 2's complement format. Bit 15 is the MSB. Bits 5-0 are always 0.

ADC10DTC0, Data Transfer Control Register 0

7	6	5	4	3	2	1	0
Reserved				ADC10TB	ADC10CT	ADC10B1	ADC10 FETCH
r0	r0	r0	r0	rw–(0)	rw–(0)	rw–(0)	rw–(0)

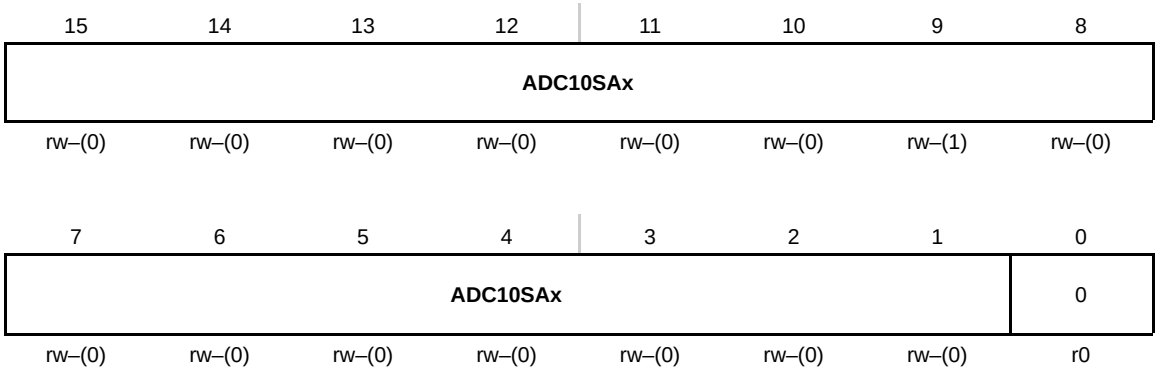
Reserved	Bits 7-4	Reserved. Always read as 0.
ADC10TB	Bit 3	ADC10 two-block mode. 0 One-block transfer mode 1 Two-block transfer mode
ADC10CT	Bit 2	ADC10 continuous transfer. 0 Data transfer stops when one block (one-block mode) or two blocks (two-block mode) have completed. 1 Data is transferred continuously. DTC operation is stopped only if ADC10CT cleared, or ADC10SA is written to.
ADC10B1	Bit 1	ADC10 block one. This bit indicates for two-block mode which block is filled with ADC10 conversion results. ADC10B1 is valid only after ADC10IFG has been set the first time during DTC operation. ADC10TB must also be set 0 Block 1 is filled 1 Block 2 is filled
ADC10 FETCH	Bit 0	This bit should normally be reset.

ADC10DTC1, Data Transfer Control Register 1



DTC Transfers	Bits 7-0	DTC transfers. These bits define the number of transfers in each block. 0 DTC is disabled 01h-0FFh Number of transfers per block

ADC10SA, Start Address Register for Data Transfer



ADC10SAx	Bits 15-1	ADC10 start address. These bits are the start address for the DTC. A write to register ADC10SA is required to initiate DTC transfers.
Unused	Bit 0	Unused, Read only. Always read as 0.

Chapter 19

DAC12

The DAC12 module is a 12-bit, voltage output digital-to-analog converter. This chapter describes the DAC12. Two DAC12 modules are implemented in the MSP430x15x and MSP430x16x devices.

Topic **Page**

19.1 DAC12 Introduction

The DAC12 module is a 12-bit, R-ladder, voltage output DAC. The DAC12 can be configured in 8- or 12-bit mode and may be used in conjunction with the DMA controller. When multiple DAC12 modules are present, they may be grouped together for synchronous update operation.

Features of the DAC12 include:

- 12-bit monotonic output
- 8- or 12-bit voltage output resolution
- Programmable settling time vs power consumption
- Internal or external reference selection
- Straight binary or 2's compliment data format
- Self-calibration option for offset correction
- Synchronized update capability for multiple DAC12s
- Multiple DAC12 modules can be grouped for simultaneous update

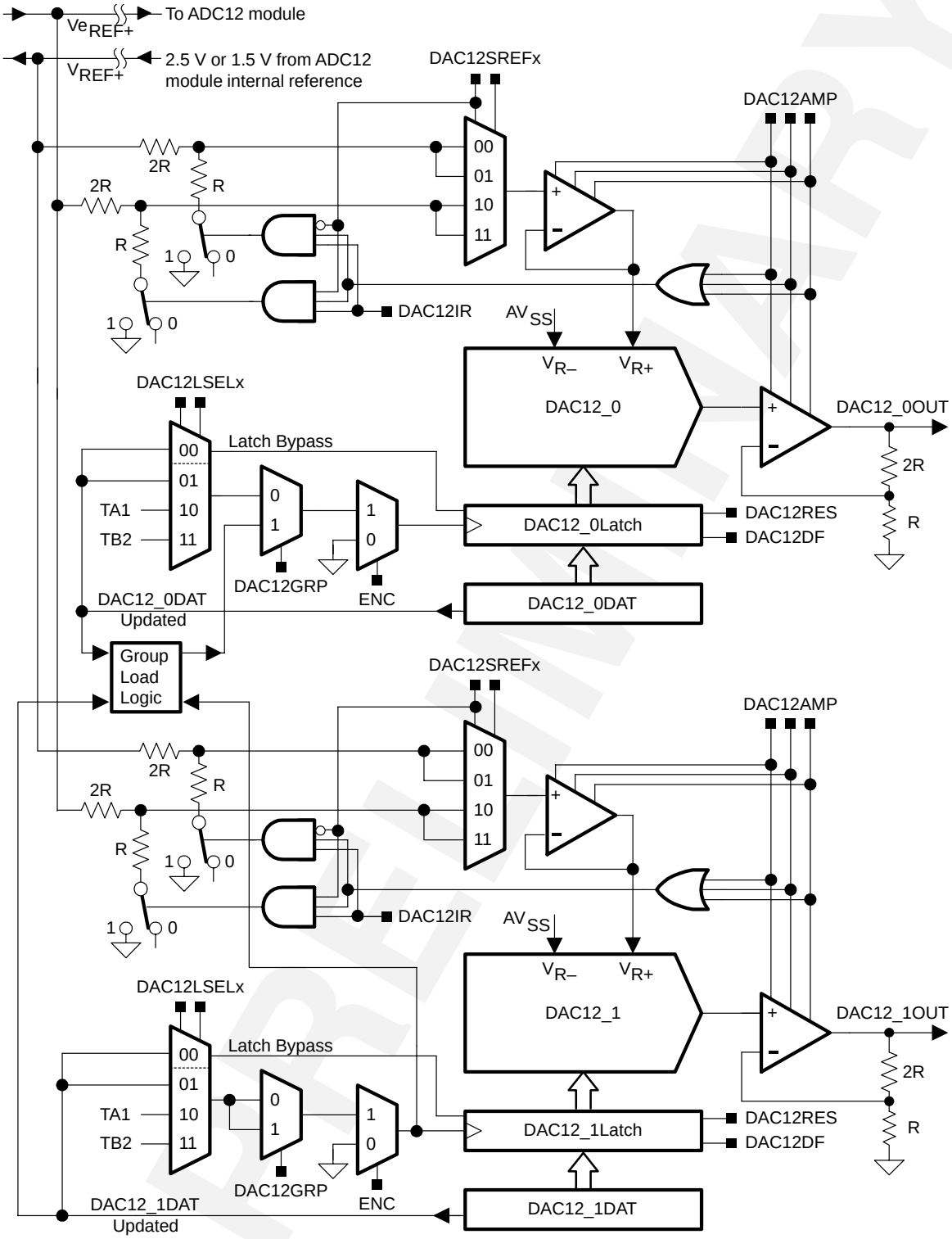
Note: Multiple DAC12 Modules

Some devices may integrate more than one DAC12 module. In the case where more than one DAC12 is present on a device, the multiple DAC12 modules operate identically.

Throughout this chapter, nomenclature appears such as DAC12_xDAT or DAC12_xCTL to describe register names. When this occurs, the x is used to indicate which DAC12 module is being discussed. In cases where operation is identical, the register is simply referred to as DAC12_xCTL.

The block diagram of the two DAC12 modules in the MSP430F15x/16x devices is shown in Figure 19–1.

Figure 19–1. DAC12 Block Diagram



19.2 DAC12 Operation

The DAC12 module is configured with user software. The setup and operation of the DAC12 is discussed in the following sections.

19.2.1 DAC12 Core

The DAC12 can be configured to operate in 8- or 12-bit mode using the DAC12RES bit. In addition, the full-scale output is programmable to be 1x or 3x the selected reference voltage via the DAC12IR bit. This feature allows the user to control the dynamic range of the DAC12. When using the internal reference, the full-scale output is always 1x the reference voltage. The DAC12DF bit allows the user to select between straight binary data and 2's compliment data for the DAC. When using straight binary data format, the formula for the output voltage is given in Table 19–1.

Table 19–1. DAC12 Full-Scale Range (Vref = V_{eREF+} or V_{REF+})

Resolution	DAC12RES	DAC12IR	Output Voltage Formula
12 bit	0	0	$Vout = Vref \times 3 \times \frac{DAC12\_xDAT}{4096}$
12 bit	0	1	$Vout = Vref \times \frac{DAC12\_xDAT}{4096}$
8 bit	1	0	$Vout = Vref \times 3 \times \frac{DAC12\_xDAT}{256}$
8 bit	1	1	$Vout = Vref \times \frac{DAC12\_xDAT}{256}$

In 8-bit mode, the maximum useable value for DAC12_xDAT is 0FFh and in 12-bit mode the maximum useable value for DAC12_xDAT is 0FFFh. Values greater than these may be written to the register, but all leading bits are ignored.

DAC12 Port Selection

The DAC12 outputs are multiplexed with the port P6 pins and ADC12 analog inputs. When DAC12AMPx > 0, the DAC12 function is automatically selected for the pin, regardless of the state of the associated P6SELx and P6DIRx bits.

19.2.2 DAC12 Reference

The reference for the DAC12 is configured to use either of two external reference voltages or the internal 1.5-V/2.5-V reference from the ADC12 module with the DAC12SREFx bits. When DAC12SREFx = {0,1} the V_{REF+} signal is used as the reference and when DAC12SREFx = {2,3} the Ve_{REF+} signal is used as the reference.

To use the ADC12 internal reference, it must be enabled and configured via the applicable ADC12 control bits (see the *ADC12* chapter). Once the ADC12 reference is configured, the reference voltage appears on the V_{REF+} signal.

DAC12 Reference Input and Voltage Output Buffers

The reference input and voltage output buffers of the DAC12 can be configured for optimized settling time vs power consumption. Eight combinations are selected using the DAC12AMPx bits. In the low/low setting, the settling time is the slowest, and the current consumption of both buffers is the lowest. The medium and high settings have faster settling times, but the current consumption increases. See the device-specific data sheet for parameters.

19.2.3 Updating the DAC12 Voltage Output

The DAC12_xDAT register can be connected directly to the DAC12 core or double buffered. The trigger for updating the DAC12 voltage output is selected with the DAC12LSELx bits.

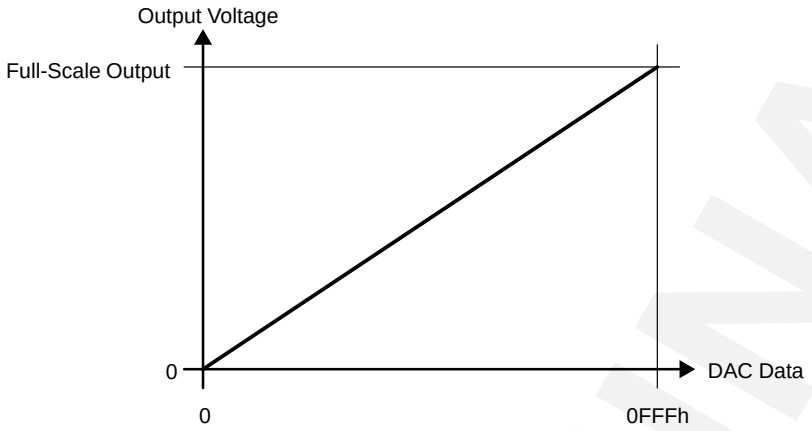
When DAC12LSELx = 0 the data latch is transparent and the DAC12_xDAT register is applied directly to the DAC12 core. the DAC12 output updates immediately when new DAC12 data is written to the DAC12_xDAT register, regardless of the state of the DAC12ENC bit.

When DAC12LSELx = 1, DAC12 data is latched and applied to the DAC12 core after new data is written to DAC12_xDAT. When DAC12LSELx = 2 or 3, data is latched on the rising edge from the Timer_A CCR1 output or Timer_B CCR2 output respectively. DAC12ENC must be set to latch the new data when DAC12LSELx > 0.

19.2.4 DAC12_xDAT Data Format

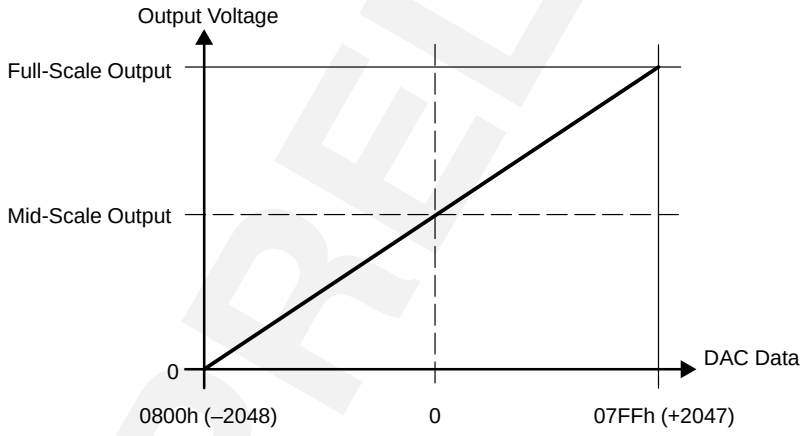
The DAC12 supports both straight binary and 2's compliment data formats. When using straight binary data format, the full-scale output value is 0FFFh in 12-bit mode (0FFh in 8-bit mode) as shown in Figure 19–2.

Figure 19–2. Output Voltage vs DAC12 Data, 12-Bit, Straight Binary Mode



When using 2's compliment data format, the range is shifted such that a DAC12_xDAT value of 0800h (0080h in 8-bit mode) results in a zero output voltage, 0000h is the mid-scale output voltage, and 07FFh (007Fh for 8-bit mode) is the full-scale voltage output as shown in Figure 19–3.

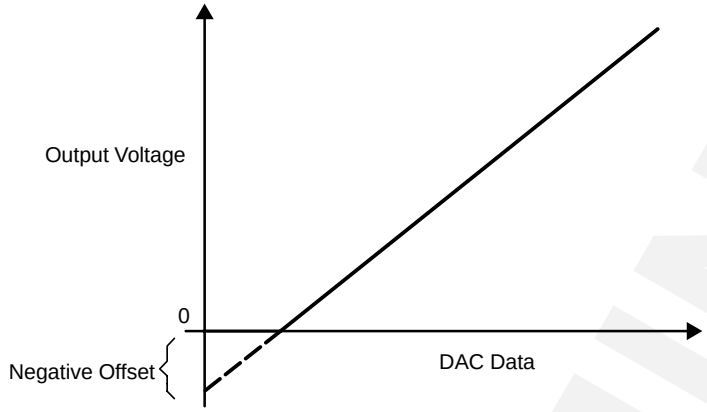
Figure 19–3. Output Voltage vs DAC12 Data, 12-Bit, 2s Compliment Mode



19.2.5 DAC12 Output Amplifier Offset Calibration

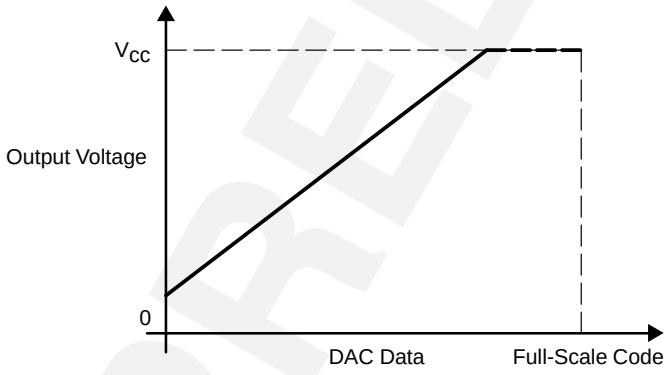
The offset voltage of the DAC12 output amplifier can be positive or negative. When the offset is negative, the output amplifier attempts to drive the voltage negative, but cannot do so. The output voltage remains at zero until the DAC12 digital input produces a sufficient positive output voltage to overcome the negative offset voltage, resulting in the transfer function shown in Figure 19–4.

Figure 19–4. Negative Offset



When the output amplifier has a positive offset, a digital input of zero does not result in a zero output voltage. The DAC12 output voltage reaches the maximum output level before the DAC12 data reaches the maximum code. This is shown in Figure 19–5.

Figure 19–5. Positive Offset



The DAC12 has the capability to calibrate the offset voltage of the output amplifier. Setting the DAC12CALON bit initiates the offset calibration. The calibration should complete before using the DAC12. When the calibration is complete, the DAC12CALON bit is automatically reset. The DAC12AMPx bits should be configured before calibration.

19.2.6 Grouping Multiple DAC12 Modules

Multiple DAC12s can be grouped together with the DAC12GRP bit to synchronize the update of each DAC12 output. Hardware ensures that all DAC12 modules in a group update simultaneously independent of any interrupt or NMI event.

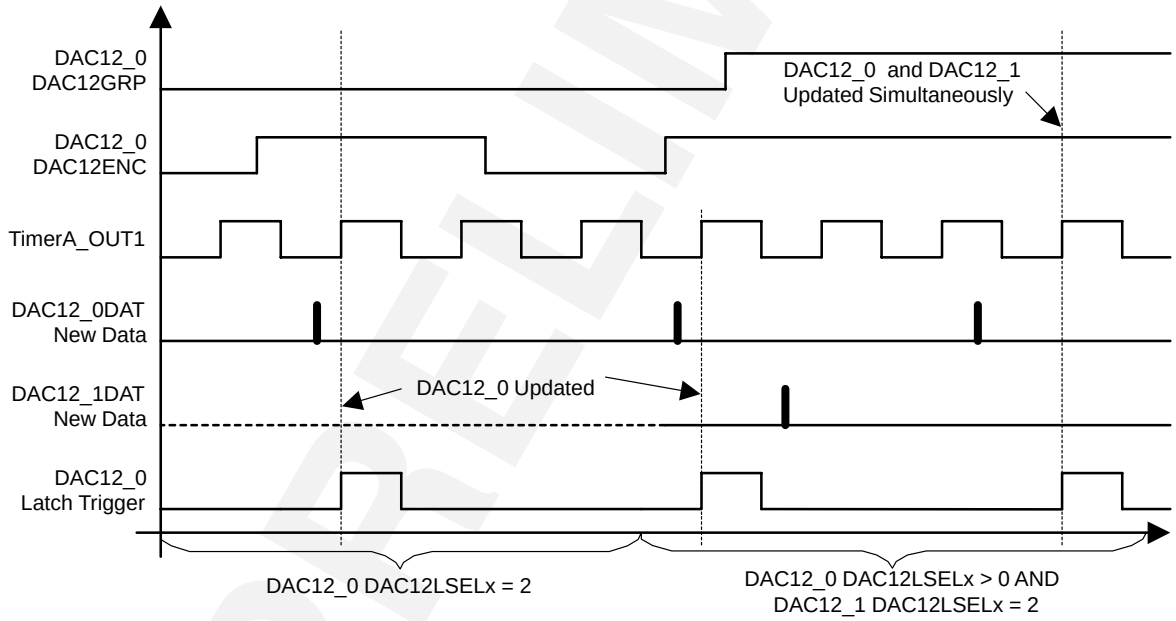
On the MSP430x15x and MSP430x16x devices, DAC12_0 and DAC12_1 are grouped by setting the DAC12GRP bit of DAC12_0. The DAC12GRP bit of DAC12_1 is don't care. When DAC12_0 and DAC12_1 are grouped:

- The DAC12_1 DAC12LSELx bits select the update trigger for both DACs
- The DAC12LSELx bits for both DACs must be > 0
- The DAC12ENC bits of both DACs must be set to 1

When DAC12_0 and DAC12_1 are grouped, both DAC12_xDAT registers must be written to before the outputs update - even if data for one or both of the DACs is not changed. Figure 19–6 shows a latch-update timing example for grouped DAC12_0 and DAC12_1.

When DAC12_0 DAC12GRP = 1 and both DAC12_x DAC12LSELx > 0 and either DAC12ENC = 0, neither DAC12 will update.

Figure 19–6. DAC12 Group Update Example, Timer_A3 Trigger



19.2.7 Using DAC12 With the DMA Controller

MSP430 devices with an integrated DMA controller can automatically move data to the DAC12_xDAT register. DMA transfers are done without CPU intervention and independently of any low-power modes. The DMA controller increases throughput to the DAC12 module, and enhances low-power applications allowing the CPU to remain off while data transfers occur.

Applications requiring periodic waveform generation can benefit from using the DMA controller with the DAC12. For example, an application that produces a sinusoidal waveform may store the sinusoid values in a table. The DMA controller can continuously and automatically transfer the values to the DAC12 at specific intervals creating the sinusoid zero CPU execution. See the *DMA Controller* chapter for more information on the DMA controller.

Note: DAC12 Settling Time

The DMA controller is capable of transferring data to the DAC12 faster than the DAC12 output can settle. The user must assure the DAC12 settling time is not violated when using the DMA controller. See the device-specific data sheet for parameters.

19.2.8 DAC12 Interrupts

The DAC12 interrupt vector is shared with the DMA controller. Software must check the DAC12IFG and DMAIFG flags to determine the source of the interrupt.

The DAC12IFG bit is set when DAC12xLSELx > 0 and DAC12 data is latched from the DAC12_xDAT register into the data latch. When DAC12xLSELx = 0, the DAC12IFG flag is not set.

A set DAC12IFG bit indicates that the DAC12 is ready for new data. If both the DAC12IE and GIE bits are set, the DAC12IFG generates an interrupt request. The DAC12IFG flag is not reset automatically. It must be reset by software.

19.3 DAC12 Registers

The DAC12 registers are listed in Table 19–2:

Table 19–2. DAC12 Registers

Register	Short Form	Register Type	Address	Initial State
DAC12_0 control	DAC12_0CTL	Read/write	01C0h	Reset with POR
DAC12_0 data	DAC12_0DAT	Read/write	01C8h	Reset with POR
DAC12_1 control	DAC12_1CTL	Read/write	01C2h	Reset with POR
DAC12_1 data	DAC12_1DAT	Read/write	01CAh	Reset with POR

DAC12_xCTL, DAC12 Control Register

15	14	13	12	11	10	9	8
Reserved	DAC12SREFx		DAC12RES	DAC12LSELx		DAC12 CALON	DAC12IR
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
DAC12AMPx			DAC12DF	DAC12IE	DAC12IFG	DAC12ENC	DAC12 GRP†
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

Modifiable only when DAC12ENC = 0

† Not used for DAC12_1 on MSP430x15x and MSP430x16x devices.

Reserved Bit 15 Reserved

DAC12 SREFx Bits 14-13 DAC12 select reference voltage
- 00 V_{REF+}
- 01 V_{REF+}
- 10 Ve_{REF+}
- 11 Ve_{REF+}

DAC12 RES Bit 12 DAC12 resolution select
- 0 12-bit resolution
- 1 8-bit resolution

DAC12 LSELx Bits 11-10 DAC12 load select. Selects the load trigger for the DAC12 latch. DAC12ENC must be set for the DAC to update, except when DAC12LSELx = 0.
- 00 DAC12 latch loads when DAC12_xDAT written (DAC12ENC is ignored)
- 01 DAC12 latch loads when DAC12_xDAT written, or, when grouped, when all DAC12_xDAT registers in the group have been written.
- 10 Rising edge of Timer_A3.OUT1 (TA1)
- 11 Rising edge of Timer_B7.OUT2 (TB2)

DAC12 CALON Bit 9 DAC12 calibration on. This bit initiates the DAC12 offset calibration sequence and is automatically reset when the calibration completes.
- 0 Calibration is not active
- 1 Initiate calibration/calibration in progress

DAC12IR Bit 8 DAC12 input range. This bit sets the reference input and voltage output range.
- 0 DAC12 full-scale output = 3x reference voltage
- 1 DAC12 full-scale output = 1x reference voltage

DAC12 AMPx Bits 7-5 DAC12 amplifier setting. These bits select settling time vs. current consumption for the DAC12 input and output amplifiers.

DAC12AMPx	Input Buffer	Output Buffer
000	Off	DAC12 off, output high Z
001	Off	DAC12 off, output 0 V
010	Low speed/current	Low speed/current
011	Low speed/current	Medium speed/current
100	Low speed/current	High speed/current
101	Medium speed/current	Medium speed/current
110	Medium speed/current	High speed/current
111	High speed/current	High speed/current

DAC12DF Bit 4 DAC12 data format
0 Straight Binary
1 2's compliment

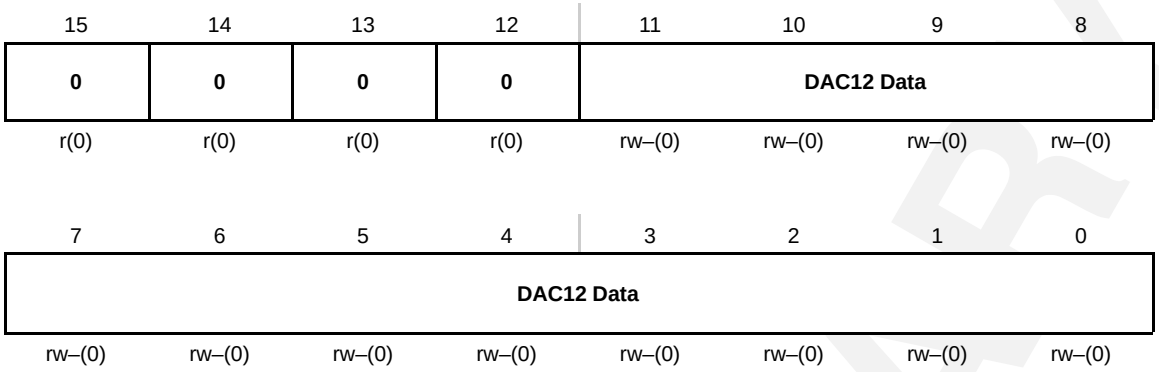
DAC12IE Bit 3 DAC12 interrupt enable
0 Disabled
1 Enabled

DAC12IFG Bit 2 DAC12 Interrupt flag
0 No interrupt pending
1 Interrupt pending

DAC12 ENC Bit 1 DAC12 enable conversion. This bit enables the DAC12 module when DAC12LSELx > 0. when DAC12LSELx = 0, DAC12ENC is ignored.
0 DAC12 disabled
1 DAC12 enabled

DAC12 GRP Bit 0 DAC12 group. Groups DAC12_x with the next higher DAC12_x.
0 Not grouped
1 Grouped

DAC12_xDAT, DAC12 Data Register



Unused Bits 15-12 Unused. These bits are always 0 and do not affect the DAC12 core.

DAC12 Data Bits 11-0 DAC12 data

DAC12 Data Format	DAC12 Data
12-bit binary	The DAC12 data are right-justified. Bit 11 is the MSB.
12-bit 2's complement	The DAC12 data are right-justified. Bit 11 is the MSB (sign).
8-bit binary	The DAC12 data are right-justified. Bit 7 is the MSB. Bits 11-8 are don't care and do not effect the DAC12 core.
8-bit 2's complement	The DAC12 data are right-justified. Bit 7 is the MSB (sign). Bits 11-8 are don't care and do not effect the DAC12 core.