# MSP430x2xx Family

# User's Guide

2006

Mixed Signal Products

SLAU144B

## IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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

## Preface

# Read This First

### About This Manual

This manual discusses modules and peripherals of the MSP430x2xx 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 MSP430x2xx family include:

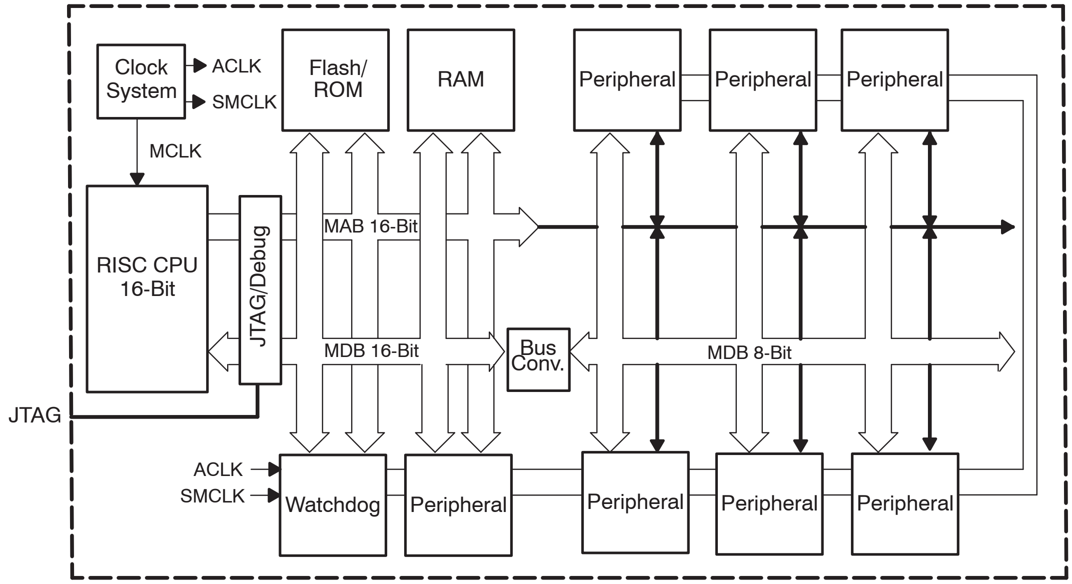
- Ultralow-power architecture extends battery life
  - 0.1-µA RAM retention
  - 0.8-µA real-time clock mode
  - 250-µA / MIPS active
- High-performance analog ideal for precision measurement
  - Comparator-gated timers for measuring resistive elements
- 16-bit RISC CPU enables new applications at a fraction 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 auxiliary 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 2 µs @ 1 Mhz. 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:

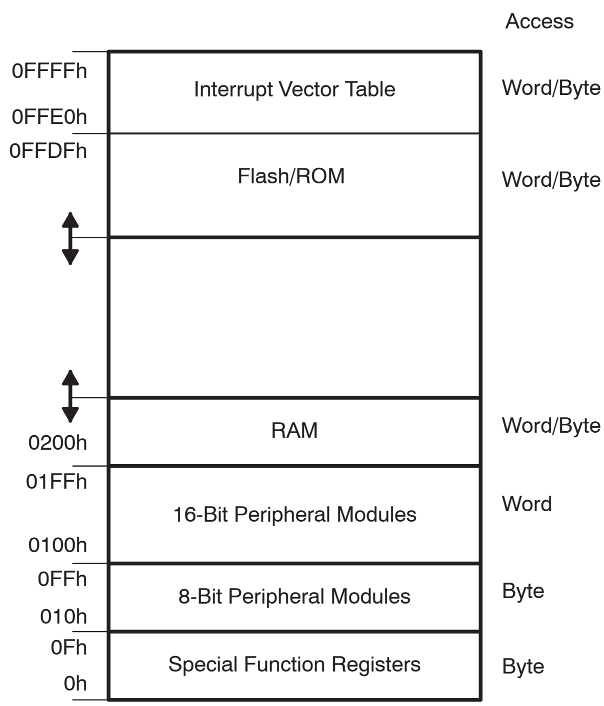
- Unobtrusive development and debug with full-speed execution, breakpoints, and single-steps in an application are supported.
- Development is in-system subject to the same characteristics as the final application.
- Mixed-signal integrity is preserved and not subject to cabling interference.

## 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 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.

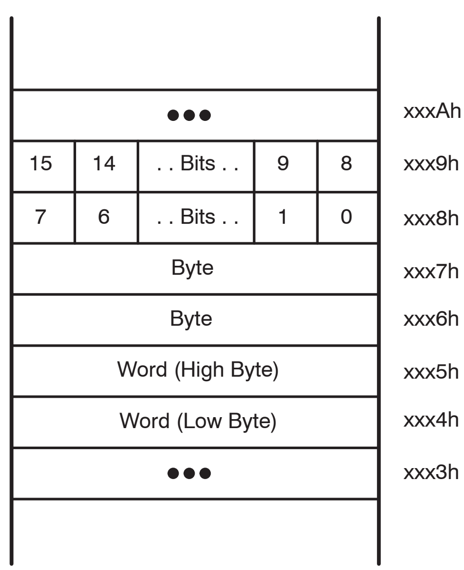
### 1.4.4 Special Function Registers (SFRs)

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*



## 1.5 MSP430x2xx Family Enhancements

Table 1−1 highlights enhancements made to the MSP430x2xx family. The enhancements are discussed fully in the following chapters, or in the case of improved device parameters, shown in the device-specific datasheet.

*Table 1−1. MSP430x2xx Family Enhancements*

| Subject | Enhancement |
|---|---|
| Reset | − Brownout reset is included on all MSP430x2xx devices.<br>− PORIFG and RSTIFG flags have been added to IFG1 to indicate the cause of a reset.<br>− An instruction fetch from the address range 0x0000 − 0x01FF will reset the device. |
| Watchdog Timer | − All MSP430x2xx devices integrate the Watchdog Timer+ module (WDT+). The WDT+ ensures the clock source for the timer is never disabled. |
| Basic Clock System | − The LFXT1 oscillator has selectable load capacitors in LF mode.<br>− The LFXT1 supports up to 16-MHz crystals in HF mode.<br>− The LFXT1 includes oscillator fault detection in LF mode.<br>− The XIN and XOUT pins are shared function pins on 20- and 28-pin devices.<br>− The external $R_{OSC}$ feature of the DCO not supported on some devices. Software should not set the LSB of the BCSCTL2 register in this case. See the device-specific datasheet for details.<br>− The DCO operating frequency has been significantly increased.<br>− The DCO temperature stability has been significantly improved. |
| Flash Memory | − The information memory has 4 segments of 64-Bytes each.<br>− SegmentA is individually locked with the LOCKA bit.<br>− All information if protected from mass erase with the LOCKA bit.<br>− Segment erases can be interrupted by an interrupt.<br>− Flash updates can be aborted by an interrupt.<br>− Flash programming voltage has been lowered to 2.2 V<br>− Program/erase time has been reduced.<br>− Clock failure aborts a flash update. |
| Digital I/O | − Ports 1 and 2 have integrated pull-up/down resistors.<br>− P2.6 and P2.7 functions have been added to 20- and 28- pin devices. These are shared functions with XIN and XOUT. Software must not clear the P2SELx bits for these pins if crystal operation is required. |
| Comparator_A | − Comparator_A has expanded input capability with a new input multiplexer. |
| Low Power | − Typical LPM3 current consumption has been reduced almost 50% @3V.<br>− DCO startup time has been significantly reduced. |
| Operating frequency | − The target maximum operating frequency is 16Mhz @ 3.3V. |
| BSL | − An incorrect password causes a mass erase.<br>− BSL entry sequence is more robust to prevent accidental entry and erasure. |

Chapter 2

# System Resets, Interrupts, and Operating Modes

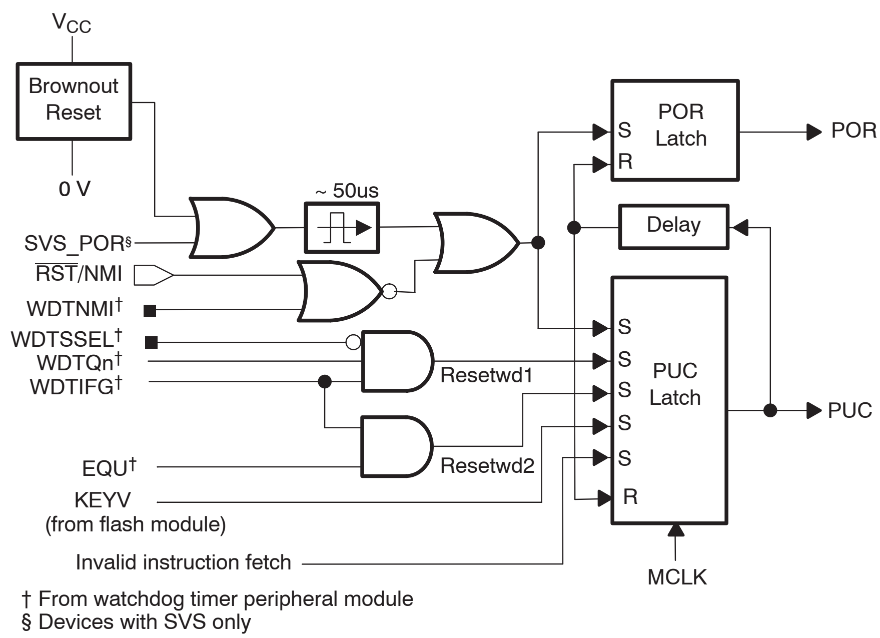
This chapter describes the MSP430x2xx system resets, interrupts, and operating modes.

| 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*




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

- Powering up the device
- A low signal on the $\overline{\text{RST}}$/NMI pin when configured in the reset mode
- An SVS low condition when PORON = 1.

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:

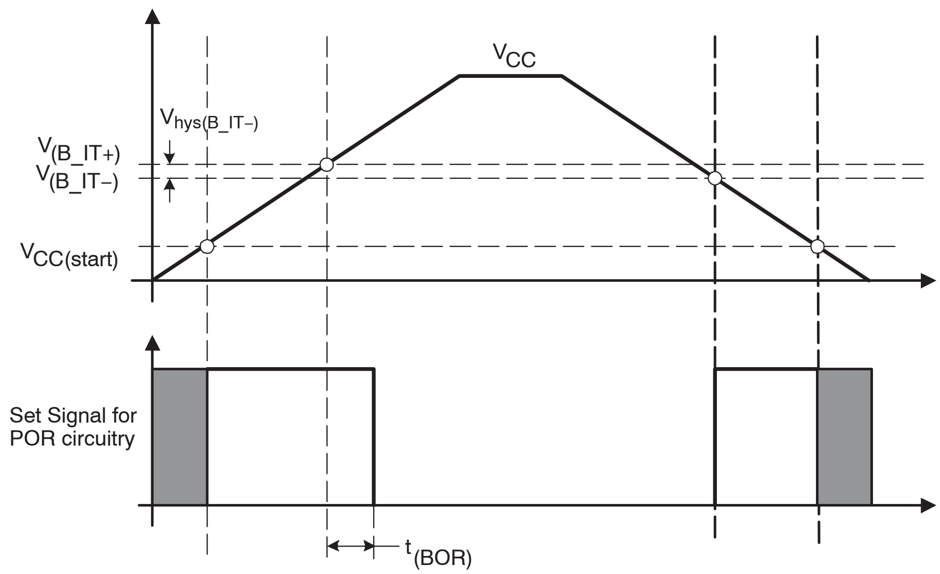
- A POR signal
- Watchdog timer expiration when in watchdog mode only
- Watchdog timer security key violation
- A Flash memory security key violation
- A CPU instruction fetch from the peripheral address range 0h − 01FFh

### 2.1.1 Brownout Reset (BOR)

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 2−2.

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 2−2. 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.

### 2.1.2 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). If the reset vectors content is 0FFFFh the device will be disabled for minimum power consumption.

#### 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.

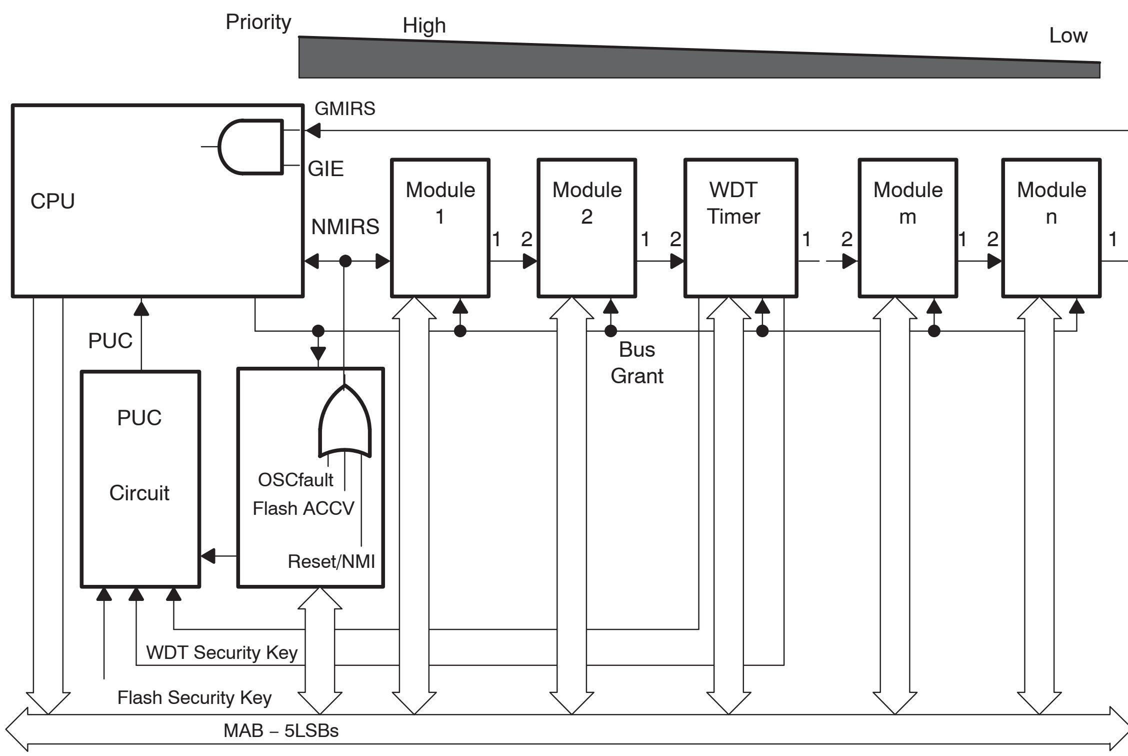
## 2.2 Interrupts

The interrupt priorities are fixed and defined by the arrangement of the modules in the connection chain as shown in Figure 2−3. The nearer a module is to the CPU/NMIRS, the higher the priority. Interrupt priorities determine what interrupt is taken when more than one interrupt is pending simultaneously.

There are three types of interrupts:

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

*Figure 2−3. 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 (NMIIE, ACCVIE, 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 2−4.

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

- An edge on the $\overline{\text{RST}}$/NMI pin when configured in NMI mode
- 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, and the RSTIFG flag is set.

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

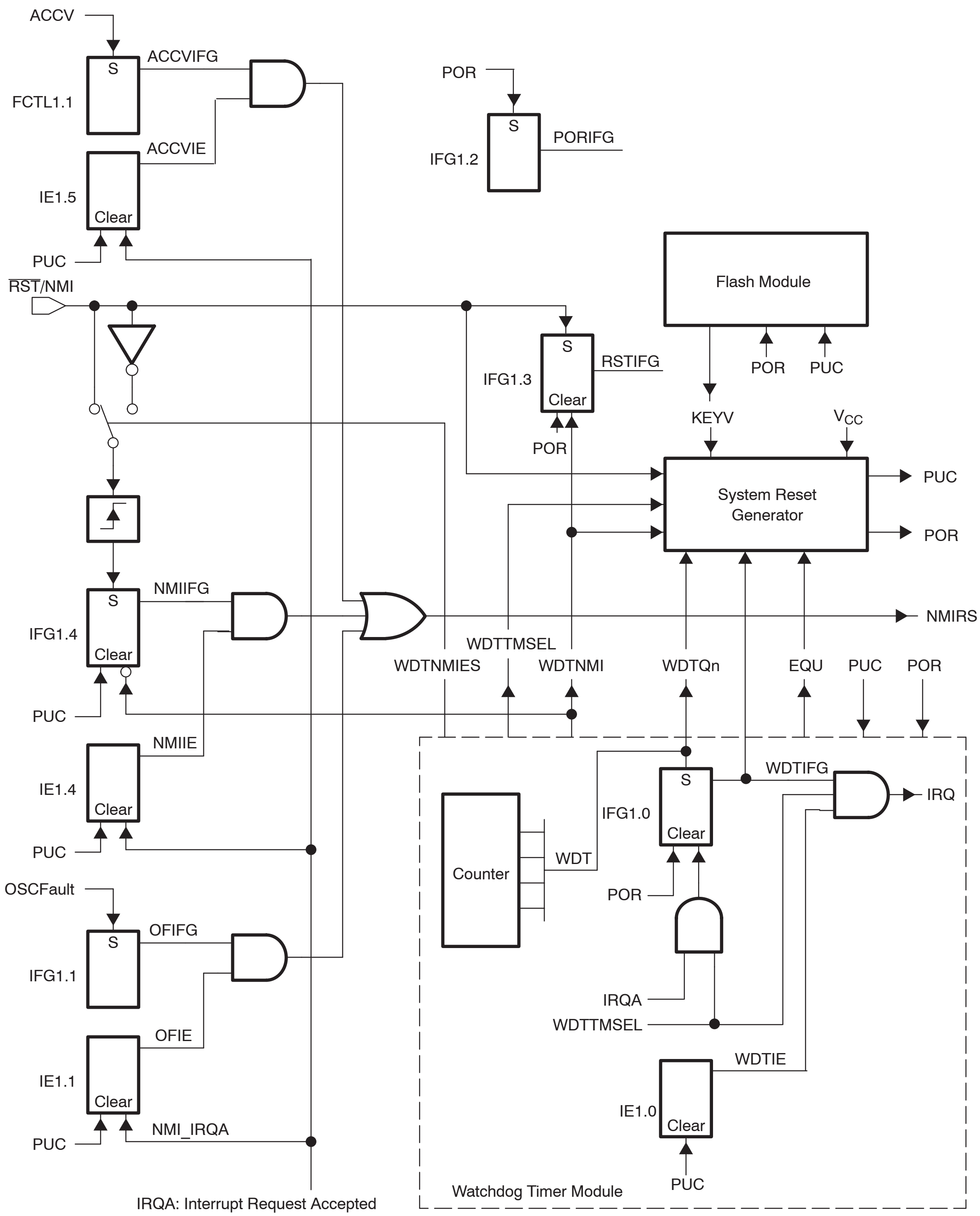
**Note: Holding $\overline{\text{RST}}$/NMI Low**

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 WDTNMIES**

When NMI mode is selected and the WDTNMIES 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 2−4. Block Diagram of (Non)-Maskable Interrupt Sources*



## 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.

## Oscillator Fault

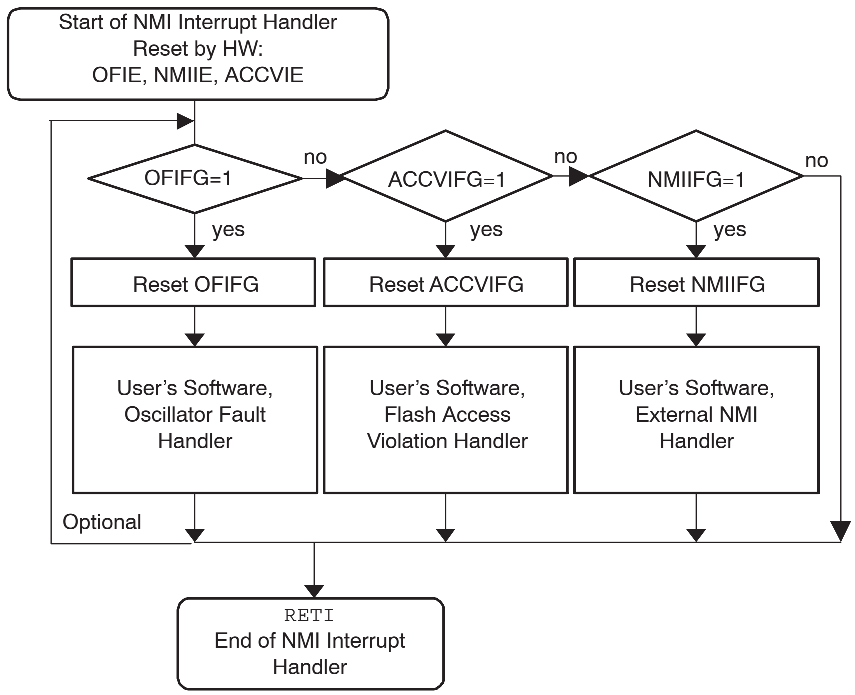
The oscillator fault signal warns of a possible error condition with the crystal oscillator. The oscillator fault can be enabled to generate an NMI interrupt by setting the OFIE bit. The OFIFG flag can then be tested by NMI the interrupt service routine to determine if the NMI was caused by an oscillator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

### 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 2−5.

*Figure 2−5. NMI Interrupt Handler*




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

To prevent nested NMI interrupts, the ACCVIE, NMIIE, and OFIE enable bits should not be set inside of an NMI interrupt service routine.

### 2.2.2 Maskable Interrupts

Maskable interrupts are caused by peripherals with interrupt capability including the watchdog timer overflow in interval-timer 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 2−6. 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.

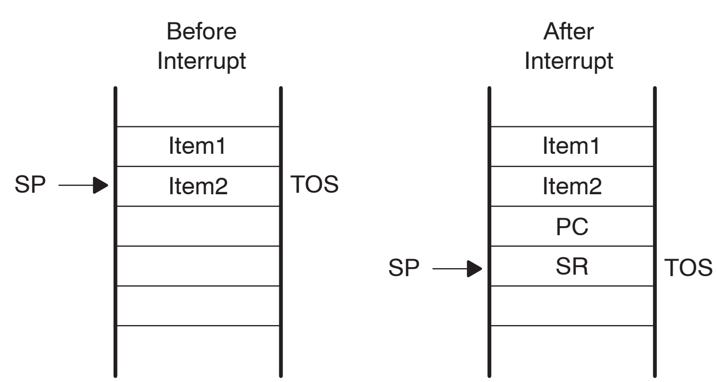
4) The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.

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. This terminates any low-power mode. Because the GIE bit is cleared, further interrupts are disabled.

7) The content of the interrupt vector is loaded into the PC: the program continues with the interrupt service routine at that address.

*Figure 2−6. Interrupt Processing*



## Return From Interrupt

The interrupt handling routine terminates with the instruction:

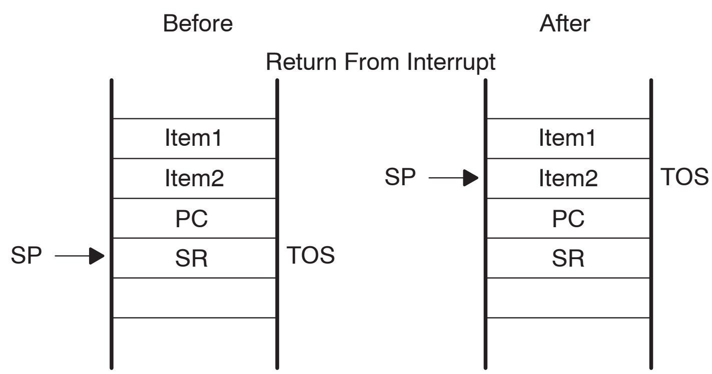
`RETI` (return from an interrupt service routine)

The return from the interrupt takes 5 cycles to execute the following actions and is illustrated in Figure 2−7.

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 2−7. Return From Interrupt*




## Interrupt Nesting

Interrupt nesting is enabled if the GIE bit is set inside an interrupt service routine. When interrupt nesting is enabled, any interrupt occurring during an interrupt service routine will interrupt the routine, regardless of the interrupt priorities.

### 2.2.4 Interrupt Vectors

The interrupt vectors and the power-up starting address are located in the address range 0FFFFh − 0FFC0h 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.

It is recommended to provide an interrupt service routine for each interrupt vector that is assigned to a module. A dummy interrupt service routine can consist of just the RETI instruction and several interrupt vectors can point to it.

Unassigned interrupt vectors can be used for regular program code if necessary.

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.

*Table 2−1. Interrupt Sources,Flags, and Vectors*

| INTERRUPT SOURCE | INTERRUPT FLAG | SYSTEM INTERRUPT | WORD ADDRESS | PRIORITY |
|---|---|---|---|---|
| Power-up, external reset, watchdog, flash password, illegal instruction fetch | PORIFG<br>RSTIFG<br>WDTIFG<br>KEYV | Reset | 0FFFEh | 31, highest |
| NMI, oscillator fault, flash memory access violation | NMIIFG<br>OFIFG<br>ACCVIFG | (non)-maskable<br>(non)-maskable<br>(non)-maskable | 0FFFCh | 30 |
| device-specific | | | 0FFFAh | 29 |
| device-specific | | | 0FFF8h | 28 |
| device-specific | | | 0FFF6h | 27 |
| Watchdog timer | WDTIFG | maskable | 0FFF4h | 26 |
| device-specific | | | 0FFF2h | 25 |
| device-specific | | | 0FFF0h | 24 |
| device-specific | | | 0FFEEh | 23 |
| device-specific | | | 0FFECh | 22 |
| device-specific | | | 0FFEAh | 21 |
| device-specific | | | 0FFE8h | 20 |
| device-specific | | | 0FFE6h | 19 |
| device-specific | | | 0FFE4h | 18 |
| device-specific | | | 0FFE2h | 17 |
| device-specific | | | 0FFE0h | 16 |
| device-specific | | | 0FFDEh | 15 |
| device-specific | | | 0FFDCh | 14 |
| device-specific | | | 0FFDAh | 13 |
| device-specific | | | 0FFD8h | 12 |
| device-specific | | | 0FFD6h | 11 |
| device-specific | | | 0FFD4h | 10 |
| device-specific | | | 0FFD2h | 9 |
| device-specific | | | 0FFD0h | 8 |
| device-specific | | | 0FFCEh | 7 |
| device-specific | | | 0FFCCh | 6 |
| device-specific | | | 0FFCAh | 5 |
| device-specific | | | 0FFC8h | 4 |
| device-specific | | | 0FFC6h | 3 |
| device-specific | | | 0FFC4h | 2 |
| device-specific | | | 0FFC2h | 1 |
| device-specific | | | 0FFC0h | 0, lowest |

## 2.3 Operating Modes

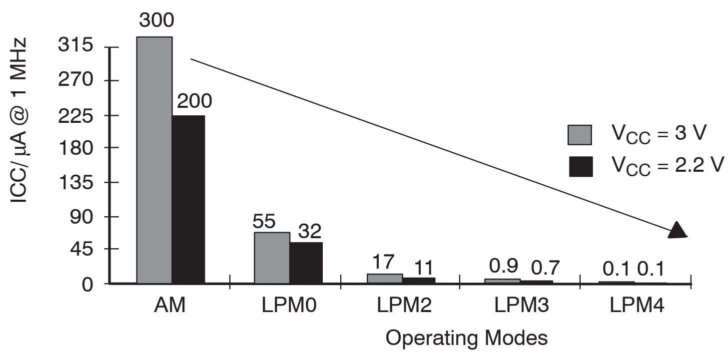
The MSP430 family is designed for ultralow-power applications and uses different operating modes shown in Figure 2−9.

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 2−8.

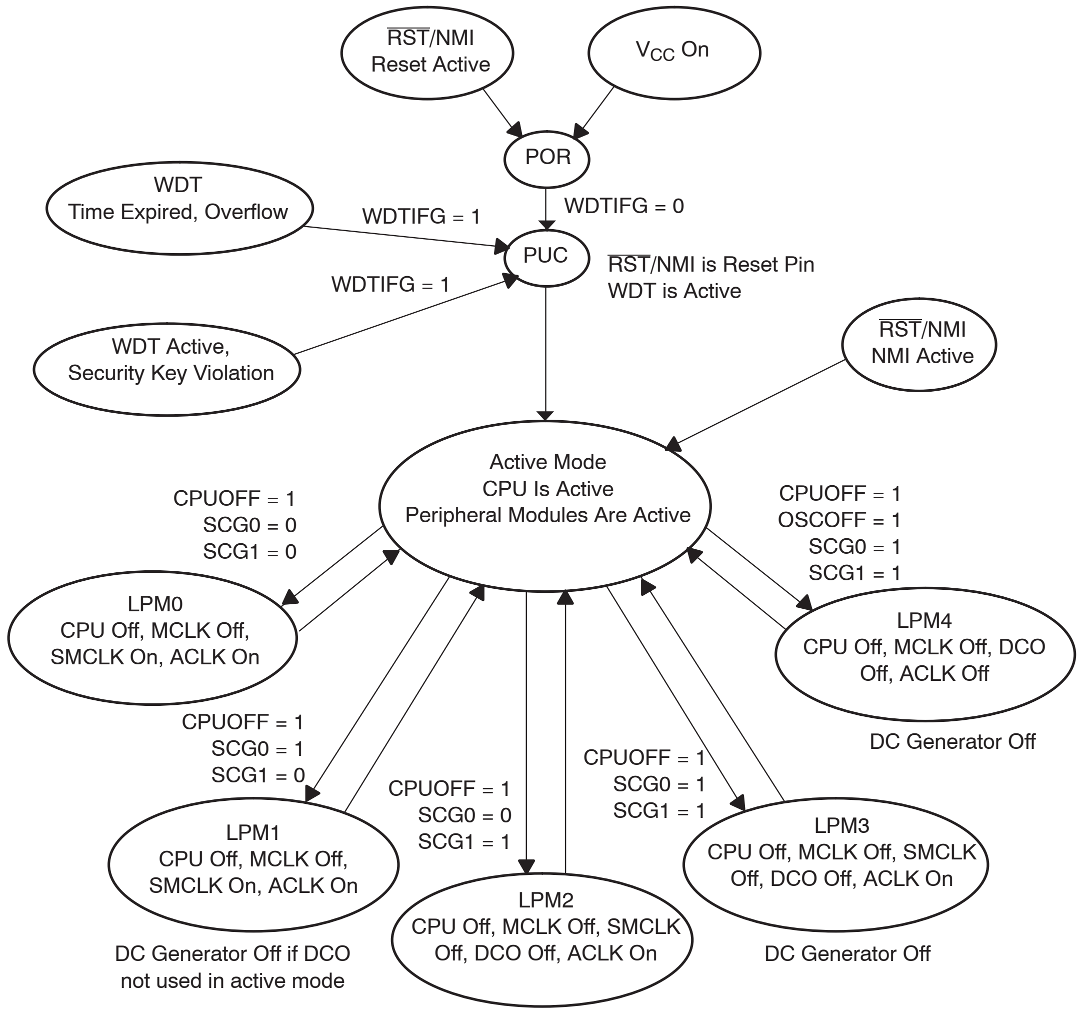
*Figure 2−8. Typical Current Consumption of 21x1 Devices vs Operating Modes*




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 2−9. MSP430x2xx 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<br>SMCLK , ACLK are active |
| 0 | 1 | 0 | 1 | LPM1 | CPU, MCLK, DCO osc. are disabled<br>DC generator is disabled if the DCO is not used for MCLK or SMCLK in active mode<br>SMCLK , ACLK are active |
| 1 | 0 | 0 | 1 | LPM2 | CPU, MCLK, SMCLK, DCO osc. are disabled<br>DC generator remains enabled<br>ACLK is active |
| 1 | 1 | 0 | 1 | LPM3 | CPU, MCLK, SMCLK, DCO osc. are disabled<br>DC generator disabled<br>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
```

## 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** | $DV_{CC}$ | |
| **XOUT** | Open | |
| **XT2IN** | $DV_{SS}$ | |
| **XT2OUT** | Open | |
| **Px.0 to Px.7** | Open | Switched to port function, output direction or input with pull−up/down enabled |
| **$\overline{RST}$/NMI** | $DV_{CC}$ or $V_{CC}$ | 47 kΩ pullup with 10nF pull down |
| **Test** | Open | 21x1, 22xx 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

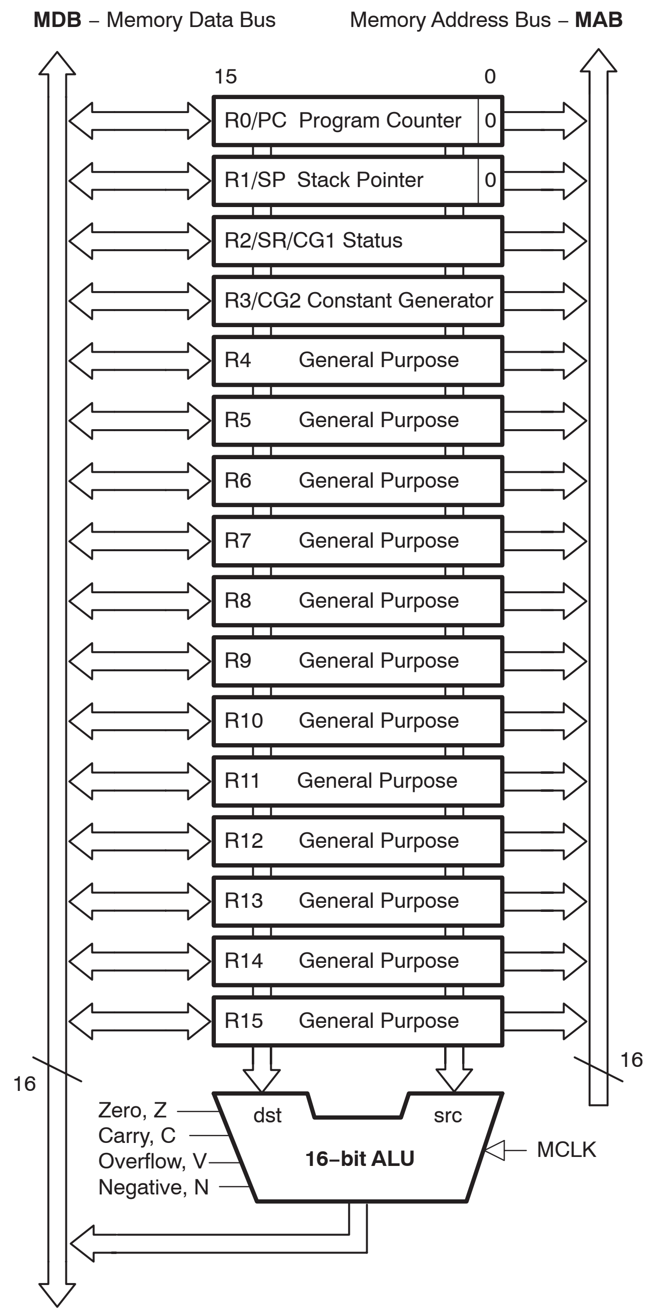
The CPU incorporates features specifically designed for modern programming techniques such as calculated branching, table processing and the use of high-level languages such as C. The CPU can address the complete address range without paging.

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*

| 15 1 | 0 |
|---|---|
| Program Counter Bits 15 to 1 | 0 |

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

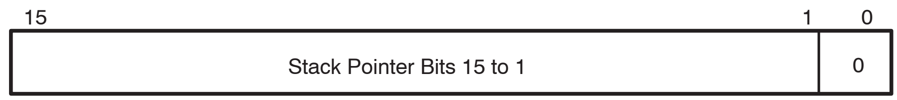
```
MOV   #LABEL,PC ; Branch to address LABEL
MOV   LABEL,PC  ; Branch to address contained in LABEL
MOV   @R14,PC   ; Branch indirect to address in R14
```

### 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.

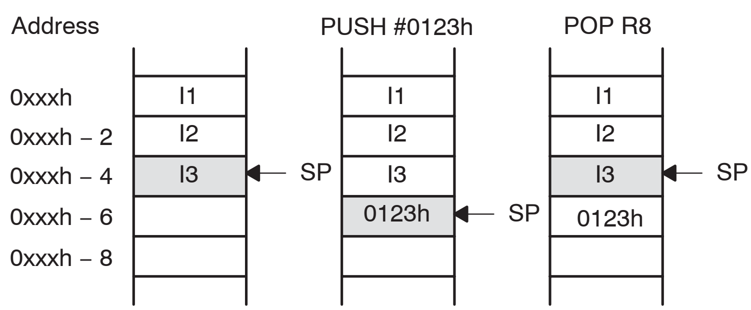
Figure 3−4 shows stack usage.

*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*

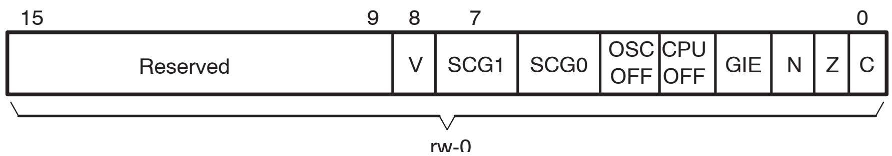



Table 3−1 describes the status register bits.

*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.<br>`ADD(.B),ADDC(.B)` — Set when:<br>Positive + Positive = Negative<br>Negative + Negative = Positive,<br>otherwise reset<br>`SUB(.B),SUBC(.B),CMP(.B)` — Set when:<br>Positive − Negative = Negative<br>Negative − Positive = Positive,<br>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.<br>Word operation: N is set to the value of bit 15 of the result<br>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
```

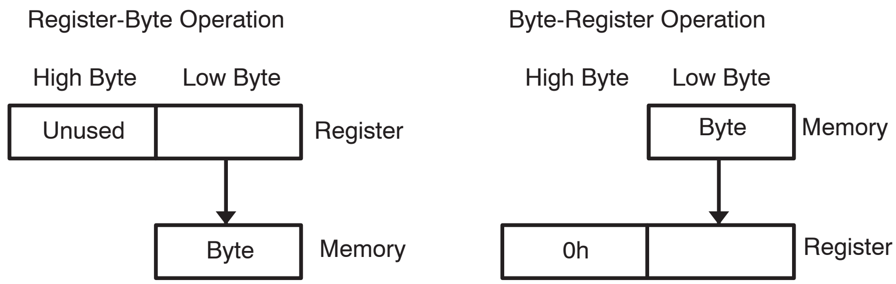
is replaced by:

```
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*




**Example Register-Byte Operation**

R5 = 0A28Fh
R6 = 0203h
Mem(0203h) = 012h

```
ADD.B          R5,0(R6)
```

```
  08Fh
+ 012h
------
  0A1h
```

Mem (0203h) = 0A1h
C = 0, Z = 0, N = 1

```
   (Low byte of register)
 + (Addressed byte)
 ----------------------
 −>(Addressed byte)
```

**Example Byte-Register Operation**

R5 = 01202h
R6 = 0223h
Mem(0223h) = 05Fh

```
ADD.B      @R6,R5
```

```
    05Fh
  + 002h
--------
  00061h
```

R5 = 00061h
C = 0, Z = 0, N = 0

```
   (Addressed byte)
 + (Low byte of register)
 ----------------------
 −>(Low byte of register, zero to High byte)
```

## 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*, *TONI, TOM, and LEO***

Throughout MSP430 documentation *EDE, TONI, TOM, and LEO* 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`

| | Before: | | After: |
|---|---|---|---|
| R10 | 0A023h | R10 | 0A023h |
| R11 | 0FA15h | R11 | 0A023h |
| PC | $PC_{old}$ | PC | $PC_{old} + 2$ |

**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*

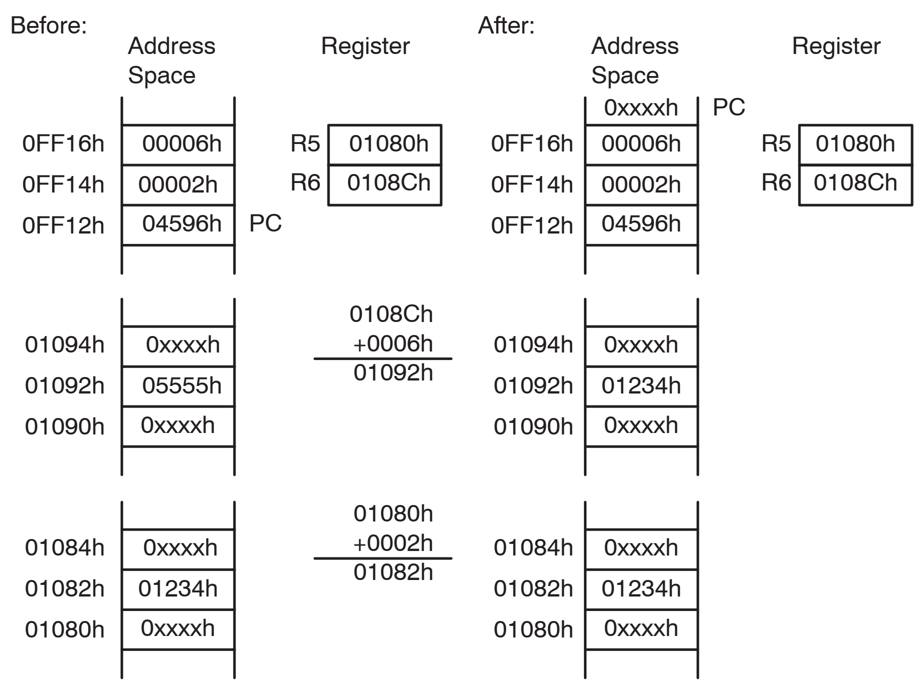
| Assembler Code | Content of ROM |
|---|---|
| `MOV 2(R5),6(R6)` | `MOV X(R5),Y(R6)`<br>X = 2<br>Y = 6 |

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 |

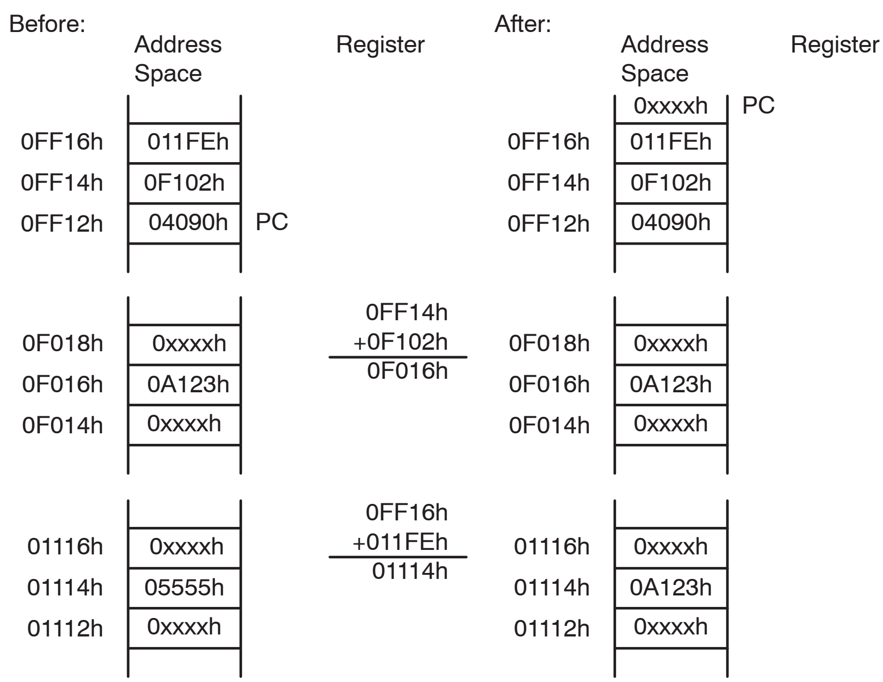
Length: Two or three words

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*

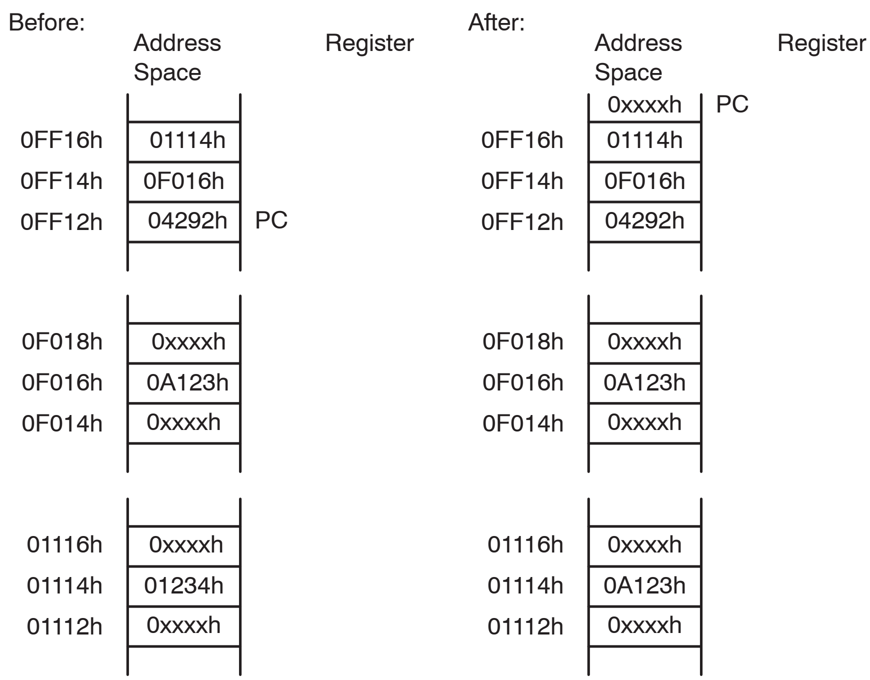
| Assembler Code | Content of ROM |
|---|---|
| `MOV &EDE,&TONI` | `MOV X(0),Y(0)`<br>X = EDE<br>Y = TONI |

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*

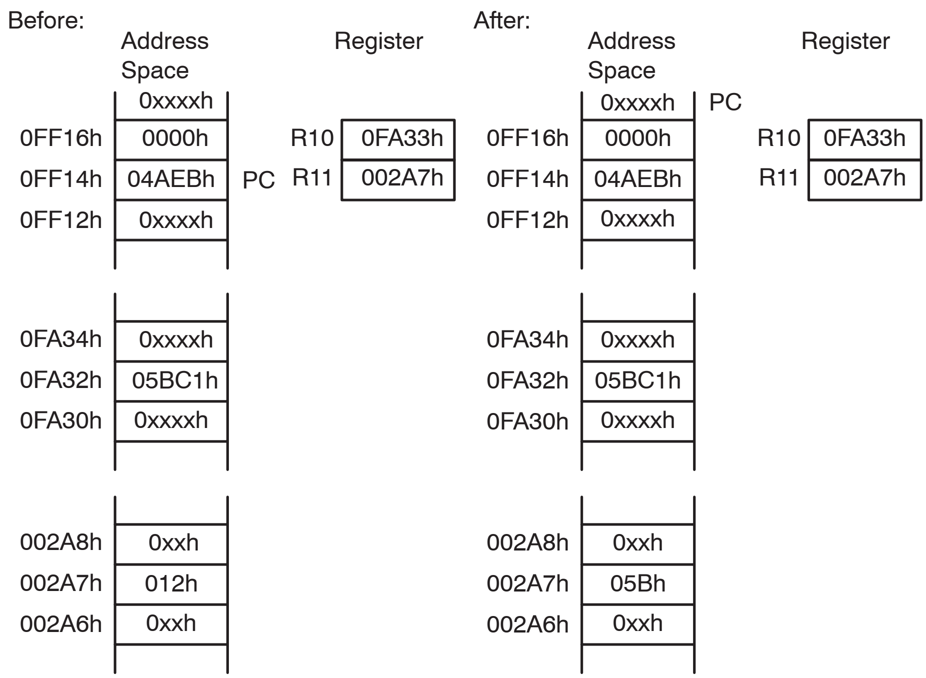
| Assembler Code | Content of ROM |
|---|---|
| `MOV @R10,0(R11)` | `MOV @R10,0(R11)` |

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*

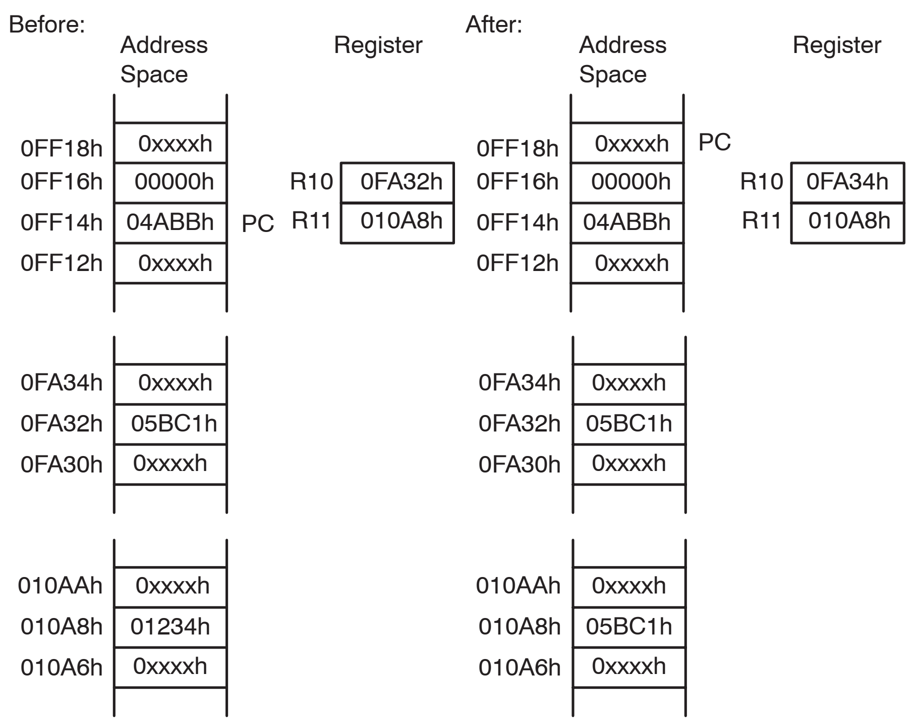
| Assembler Code | Content of ROM |
|---|---|
| `MOV @R10+,0(R11)` | `MOV @R10+,0(R11)` |

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.

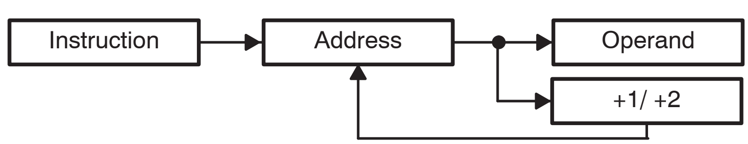
Comment: Valid only for source operand. The substitute for destination operand is 0(Rd) plus second instruction INCD Rd.

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*

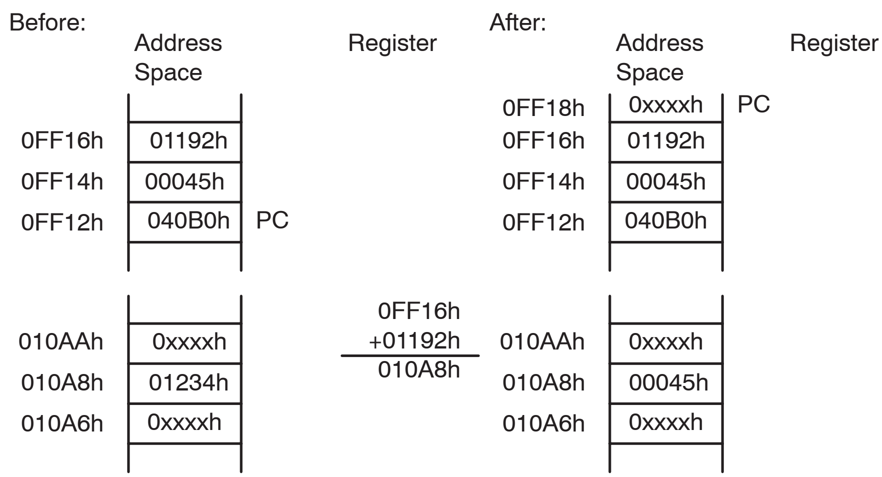
| Assembler Code | Content of ROM |
|---|---|
| `MOV #45h,TONI` | `MOV @PC+,X(PC)` |
| | 45 |
| | X = TONI − PC |

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 27 core instructions and 24 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:<br>0: word operation<br>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.

### 3.4.1 Double-Operand (Format I) Instructions

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

*Figure 3−9. Double Operand Instruction Format*

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Op-code | | | | S-Reg | | | | Ad | B/W | As | | D-Reg | | | |

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*

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Op-code | | | | | | | | | B/W | Ad | | D/S-Reg | | | |

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*

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Op-code | | | C | | | 10-Bit PC Offset | | | | | | | | | |

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 0A5h
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 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<br>CLR.B dst |
| **Operation** | 0 −> dst |
| **Emulation** | MOV #0,dst<br>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** $0 \rightarrow N$
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** $0 \rightarrow Z$
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
        MOV    #BLOCK1,R6       ; BLOCK1 start address in R6
        MOV    #BLOCK2,R7       ; BLOCK2 start address in R7
L$1     CMP    @R6+,0(R7)       ; Are Words equal? R6 increments
        JNZ    ERROR            ; No, branch to ERROR
        INCD   R7               ; Increment R7 pointer
        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.

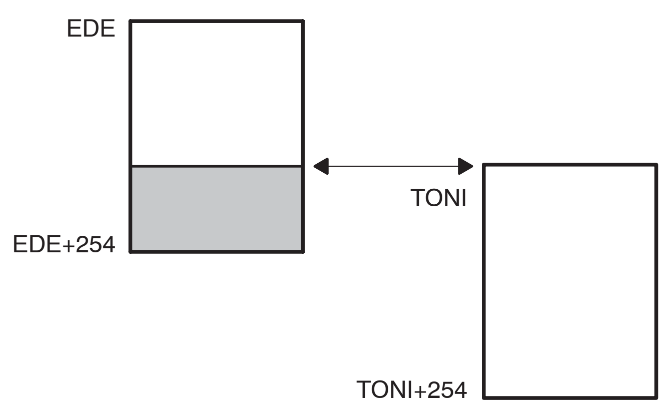
**Example** R10 is decremented by 1

```
        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 1 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<br>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<br>Z: Set if dst contained 0FFFFh, reset otherwise<br>Set if dst contained 0FFh, reset otherwise<br>C: Set if dst contained 0FFFFh, reset otherwise<br>Set if dst contained 0FFh, reset otherwise<br>V: Set if dst contained 07FFFh, reset otherwise<br>Set if dst contained 07Fh, reset otherwise |
| **Mode Bits** | OSCOFF, CPUOFF, and GIE are not affected. |
| **Example** | The status byte, STATUS, of a process 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 fill 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
**Emulation** MOV.B @SP+,dst

**Description** The stack location pointed to by the stack pointer (TOS) is moved to the destination. The stack pointer is incremented by two afterwards.

**Status Bits** Status bits are not affected.

**Example** The contents of R7 and the status register are restored from the stack.

```
POP      R7      ; Restore R7
POP      SR      ; Restore status register
```

**Example** The contents of RAM byte LEO is restored from the stack.

```
POP.B    LEO     ; The low byte of the stack is moved to LEO.
```

**Example** The contents of R7 is restored from the stack.

```
POP.B    R7      ; The low byte of the stack is moved to R7,
                 ; the high byte of R7 is 00h
```

**Example** The contents of the memory pointed to by R7 and the status register are restored from the stack.

```
POP.B    0(R7)   ; The low byte of the stack is moved to the
                 ; the byte which is pointed to by R7
                 : Example:   R7 = 203h
                 ;            Mem(R7) = low byte of system stack
                 : Example:   R7 = 20Ah
                 ;            Mem(R7) = low byte of system stack
POP      SR      ; Last word on stack moved to the 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.

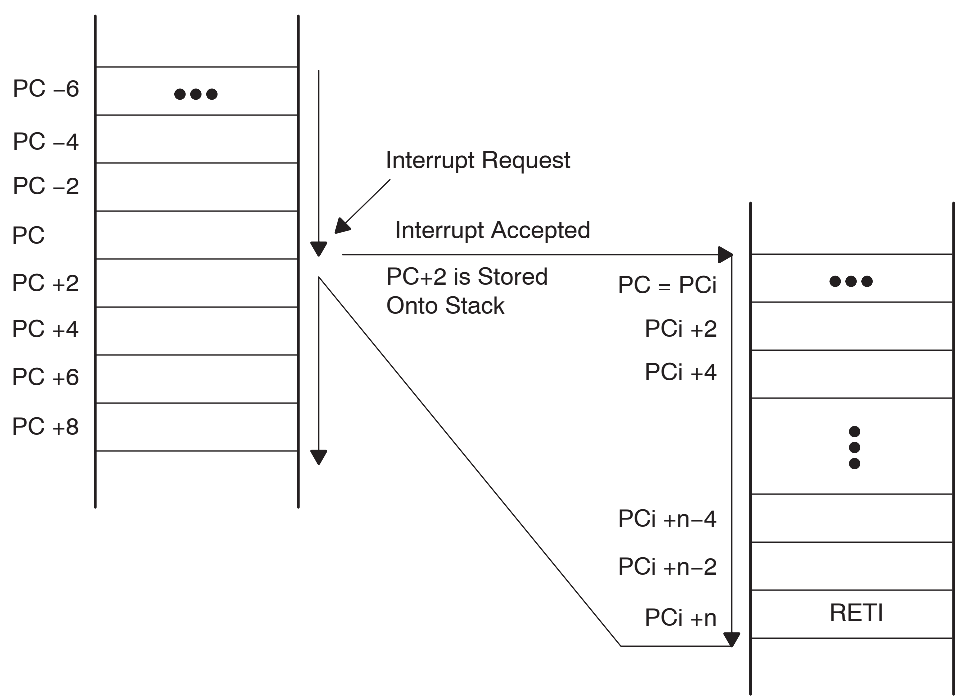
**Status Bits**

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**

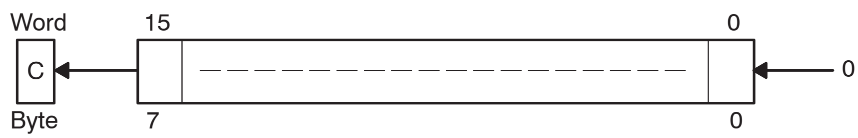
```
ADD     dst,dst
ADD.B   dst,dst
```

**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+,           RLA.B  @R5+,          or     RLA(.B) @R5
```

It must be substituted by:

```
ADD   @R5+,–2(R5)     ADD.B  @R5+,–1(R5)    or     ADD(.B) @R5
```

---

*** RLC[.W]** Rotate left through carry

*** RLC.B** Rotate left through carry

**Syntax**

```
RLC     dst     or     RLC.W     dst
RLC.B   dst
```

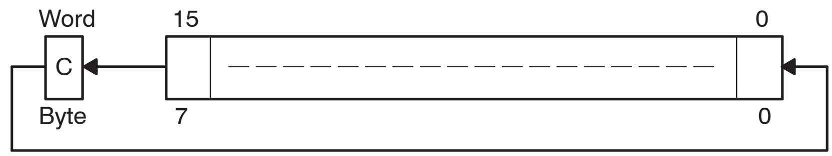
**Operation** C <– MSB <– MSB–1 .... LSB+1 <– LSB <– C

**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 Substitution**

The assembler does not recognize the instruction:

```
RLC @R5+,            RLC.B @R5+,           or RLC(.B) @R5
```

It must be substituted by:

```
ADDC @R5+,–2(R5)     ADDC.B @R5+,–1(R5)    or ADDC(.B) @R5
```

**RRA[.W]** Rotate right arithmetically
**RRA.B** Rotate right arithmetically

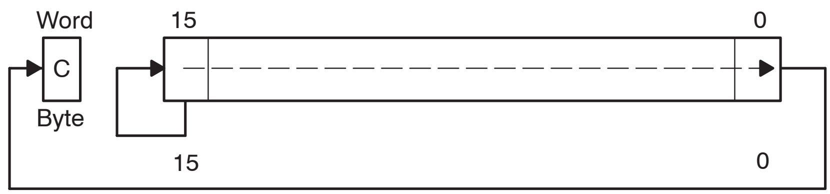
**Syntax**

```
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
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

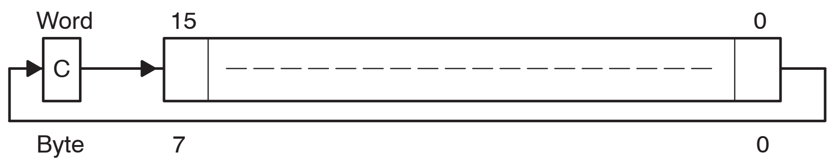
**Syntax**

```
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: 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<br>Z: Not affected<br>C: Set<br>V: Not affected |
| **Mode Bits** | OSCOFF, CPUOFF, and GIE are not affected. |
| **Example** | Emulation of the decimal subtraction:<br>Subtract R5 from R6 decimally<br>Assume that R5 = 03987h and R6 = 04137h |

```
DSUB    ADD     #06666h,R5      ; Move content R5 from 0−9 to 6−0Fh
                                ; R5 = 03987h + 06666h = 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:
                                ; (010000h − R5 − 1)
                                ; R6 = R6 + R5 + 1
                                ; R6 = 0150h
```

| | |
|---|---|
| *** SETN** | Set negative bit |
| **Syntax** | SETN |
| **Operation** | 1 –> N |
| **Emulation** | BIS #4,SR |
| **Description** | The negative bit (N) is set. |
| **Status Bits** | N: Set<br>Z: Not affected<br>C: Not affected<br>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

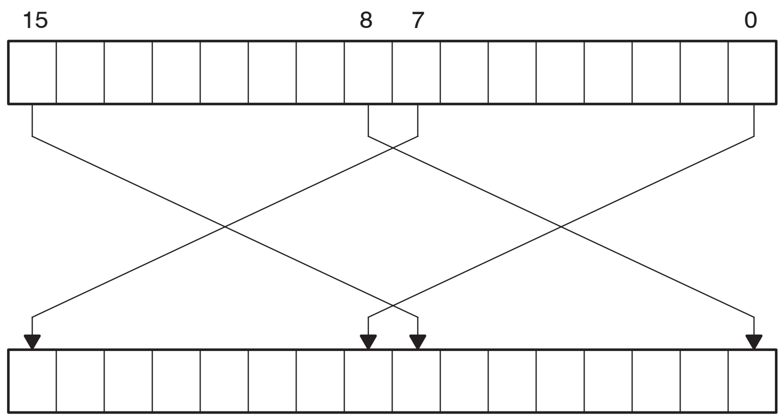
**Operation** Bits 15 to 8 <–> bits 7 to 0

**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

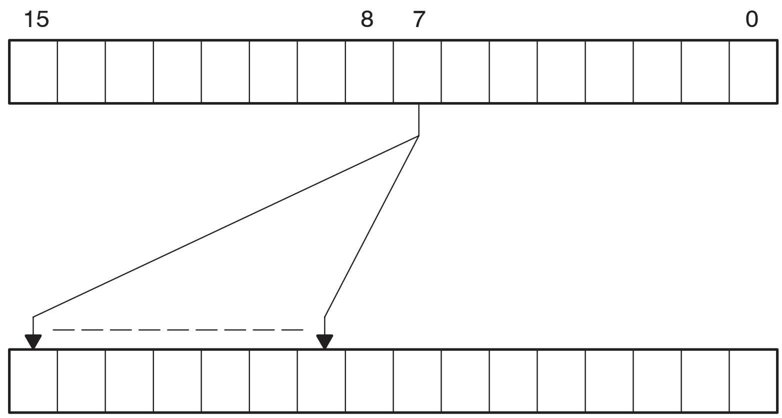
**Description** The sign of the low byte is extended into the high byte as shown in Figure 3−19.

**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 of byte TONI on the 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: RRA, RRC SWPB, SXT | No. of Cycles: PUSH | No. of Cycles: CALL | Length of Instruction | Example |
|---|---|---|---|---|---|
| Rn | 1 | 3 | 4 | 1 | `SWPB R5` |
| @Rn | 3 | 4 | 4 | 1 | `RRC @R9` |
| @Rn+ | 3 | 5 | 5 | 1 | `SWPB @R10+` |
| #N | (See note) | 4 | 5 | 2 | `CALL #0F000h` |
| 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 Src | Dst | No. of Cycles | Length of Instruction | Example | |
|---|---|---|---|---|---|
| Rn | Rm | 1 | 1 | MOV | R5,R8 |
| | PC | 2 | 1 | BR | R9 |
| | x(Rm) | 4 | 2 | ADD | R5,4(R6) |
| | EDE | 4 | 2 | XOR | R8,EDE |
| | &EDE | 4 | 2 | MOV | R5,&EDE |
| @Rn | Rm | 2 | 1 | AND | @R4,R5 |
| | PC | 2 | 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 | 4(R4),6(R9) |
| | &TONI | 6 | 3 | MOV | 2(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 MSP430x2xx devices. This chapter describes the operation of the basic clock module+. The basic clock module+ is implemented in all MSP430x2xx 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 three or four clock sources:

- ❑ LFXT1CLK: Low-frequency/high-frequency oscillator that can be used with low-frequency watch crystals or external clock sources of 32,768-Hz. or with standard crystals, resonators, or external clock sources in the 400-kHz to 16-MHz range.
- ❑ XT2CLK: Optional high-frequency oscillator that can be used with standard crystals, resonators, or external clock sources in the 400-kHz to 16-MHz range.
- ❑ DCOCLK: Internal digitally controlled oscillator (DCO).
- ❑ VLOCLK: Internal very low power, low frequency oscillator with 12kHz typical frequency.

Three clock signals are available from the basic clock module+:

- ❑ ACLK: Auxiliary clock. ACLK is software selectable as LFXT1CLK or VLOCLK. ACLK is divided by 1, 2, 4, or 8. ACLK is software selectable for individual peripheral modules.
- ❑ MCLK: Master clock. MCLK is software selectable as LFXT1CLK, VLOCLK, XT2CLK (if available on-chip), 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, VLOCLK, XT2CLK (if available on-chip), or DCOCLK. SMCLK is divided by 1, 2, 4, or 8. SMCLK is software selectable for individual peripheral modules.

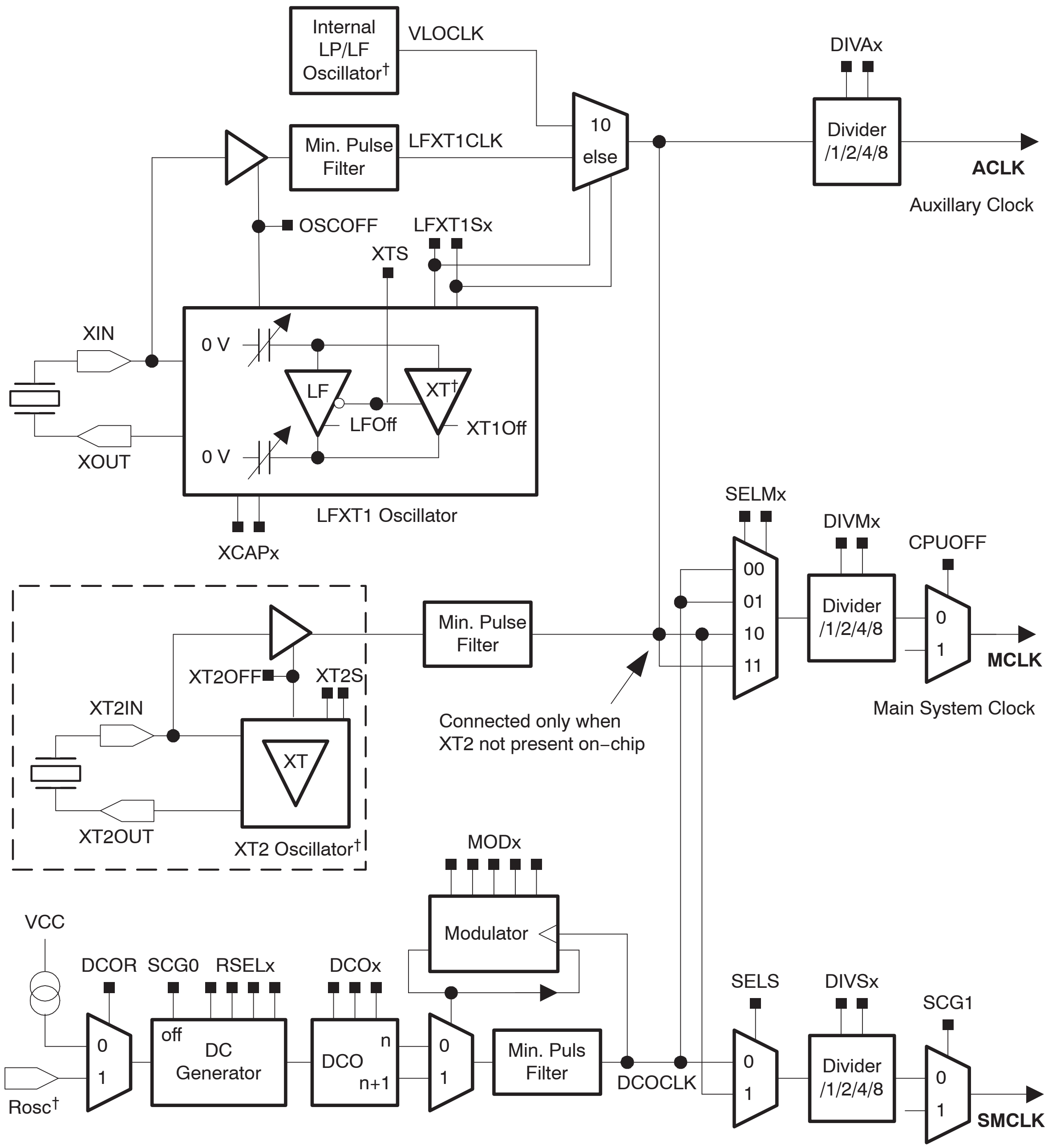
The block diagram of the basic clock module+ is shown in Figure 4−1.

**Note: Device-Specific Clock Variations**

All clock features are not available on all MSP430x2xx devices.

MSP430x20xx: LFXT1 does not support HF mode, XT2 is not present, $R_{OSC}$ is not supported.
MSP430x21xx: Internal LP/LF oscillator is not present, XT2 is not present, $R_{OSC}$ is not supported.
MSP430x22xx: XT2 is not present.

*Figure 4−1. Basic Clock Module+ Block Diagram*




**†Note: Device-Specific Clock Variations**

All clock features are not available on all MSP430x2xx devices.

MSP430x20xx: LFXT1 does not support HF mode, XT2 is not present, $R_{OSC}$ is not supported.
MSP430x21xx: Internal LP/LF oscillator is not present, XT2 is not present, $R_{OSC}$ is not supported.
MSP430x22xx: XT2 is not present.

## 4.2 Basic Clock Module+ Operation

After a PUC, MCLK and SMCLK are sourced from DCOCLK at ~1.1 MHz (see device-specific datasheet for parameters) and ACLK is sourced from LFXT1CLK in LF mode with an internal load capacitance of 6pF.

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, BCSCTL2, and BCSCTL3 registers configure the basic clock module+.

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

```
BIS.B  #RSEL2+RSEL1+RSEL0,&BCSCTL1  ; Select range 7
BIS.B  #DCO2+DCO1+DCO0,&DCOCTL      ; Select max DCO tap
```

### 4.2.1 Basic Clock Module+ Features for Low-Power Applications

Conflicting requirements typically exist in battery-powered applications:

- Low clock frequency for energy conservation and time keeping
- High clock frequency for fast reaction to events and fast burst processing capability
- Clock stability over operating temperature and supply voltage

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, ACLK can be sourced from a low-power 32,768-Hz watch crystal, providing a stable time base for the system and low power stand-by operation, or from the internal low-frequency oscillator when crystal-accurate time keeping is not required.. The MCLK can be configured to operate from the on-chip DCO that can be activated when requested by interrupt-driven events. The SMCLK can be configured to operate from a 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 Internal Very Low Power, Low Frequency Oscillator

The internal very-low-power, low-frequency oscillator (VLO) provides a typical frequency of 12kHz (see device-specific datasheet for parameters) without requiring a crystal. VLOCLK source is selected by setting LFXT1Sx = 10 when XTS = 0. The OSCOFF bit disables the VLO for LPM4. The LFXT1 crystal oscillators are disabled when the VLO is selected reducing current consumption. The VLO consumes no power when not being used.

### 4.2.3 LFXT1 Oscillator

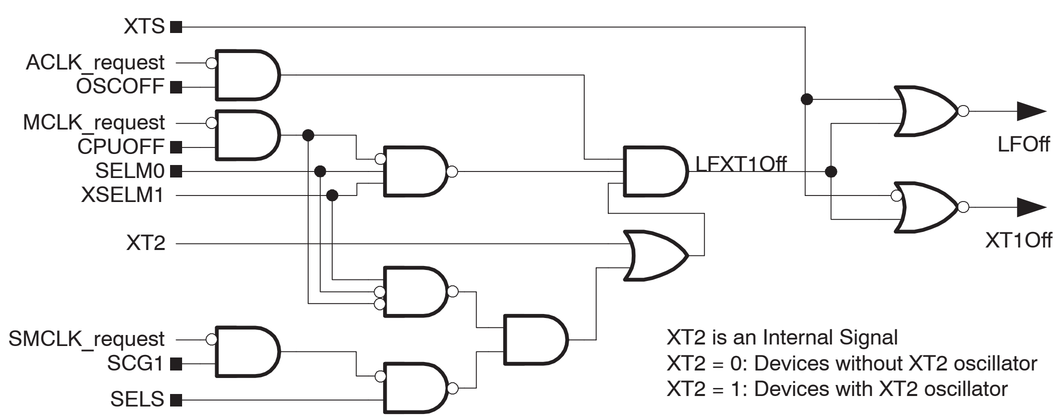
The LFXT1 oscillator supports ultralow-current consumption using a 32,768-Hz watch crystal in LF mode (XTS = 0). A watch crystal connects to XIN and XOUT without any other external components. The software-selectable XCAPx bits configure the internally provided load capacitance for the LFXT1 crystal in LF mode. This capacitance can be selected as 1pF, 6pF, 10pF or 12.5pF typical. Additional external 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. When LFXT1 is in HF mode, the LFXT1Sx bits select the range of operation.

LFXT1 may be used with an external clock signal on the XIN pin in either LF or HF mode when LFXT1Sx = 11 and OSCOFF = 0. When used with an external signal, the external frequency must meet the datasheet parameters for the chosen mode. When the input frequency is below the specified lower limit, the LFXT1OF bit may be set preventing the CPU from being clocked with LFXT1CLK.

Software can disable LFXT1 by setting OSCOFF, if LFXT1CLK does not source SMCLK or MCLK, as shown in Figure 4−2.

*Figure 4−2. 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.

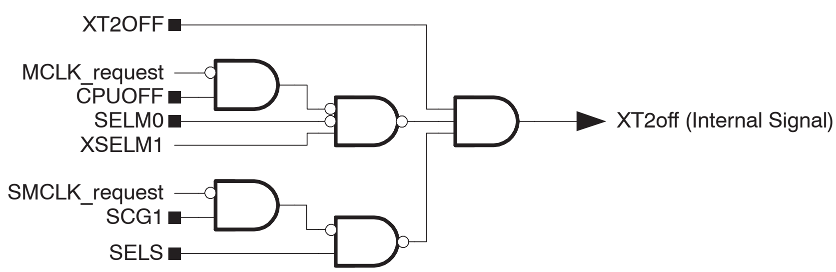
Ultralow-power oscillators such as the LFXT1 in LF mode should be guarded from noise coupling from other sources. The crystal should be placed as close as possible to the MSP430 with the crystal housing grounded and the crystal traces guarded with ground traces.

### 4.2.4 XT2 Oscillator

Some devices have a second crystal oscillator, XT2. XT2 sources XT2CLK and its characteristics are identical to LFXT1 in HF mode. The XT2Sx bits select the range of operation of XT2. The XT2OFF bit disables the XT2 oscillator if XT2CLK is not used for MCLK or SMCLK as shown in Figure 4−3.

XT2 may be used with external clock signals on the XT2IN pin when XT2Sx = 11 and XT2OFF = 0. When used with an external signal, the external frequency must meet the datasheet parameters for XT2. When the input frequency is below the specified lower limit, the XT2OF bit may be set preventing the CPU from being clocked with XT2CLK.

*Figure 4−3. Off Signals for Oscillator XT2*



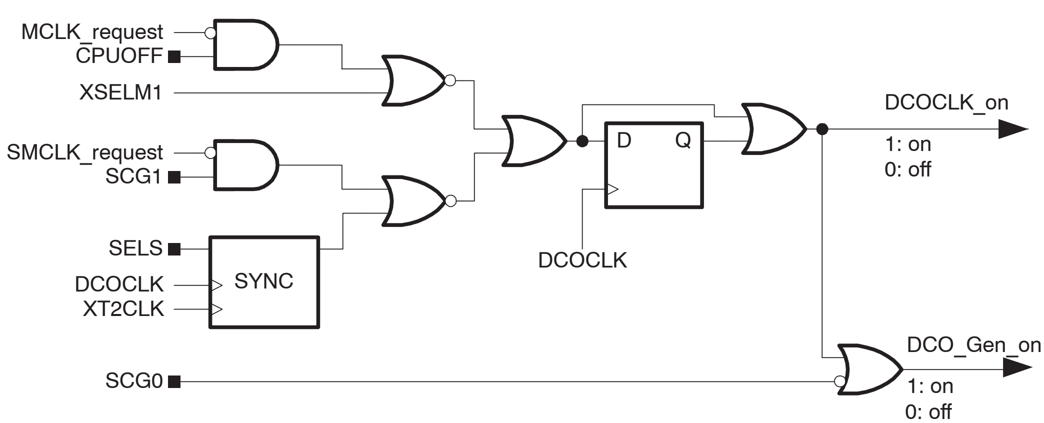

### 4.2.5 Digitally-Controlled Oscillator (DCO)

The DCO is an integrated digitally controlled oscillator. The DCO frequency can be adjusted by software using the DCOx, MODx, and RSELx bits.

#### Disabling the DCO

Software can disable DCOCLK by setting SCG0 when it is not used to source SMCLK or MCLK in active mode, as shown in Figure 4−4.

*Figure 4−4. On/Off Control of DCO*



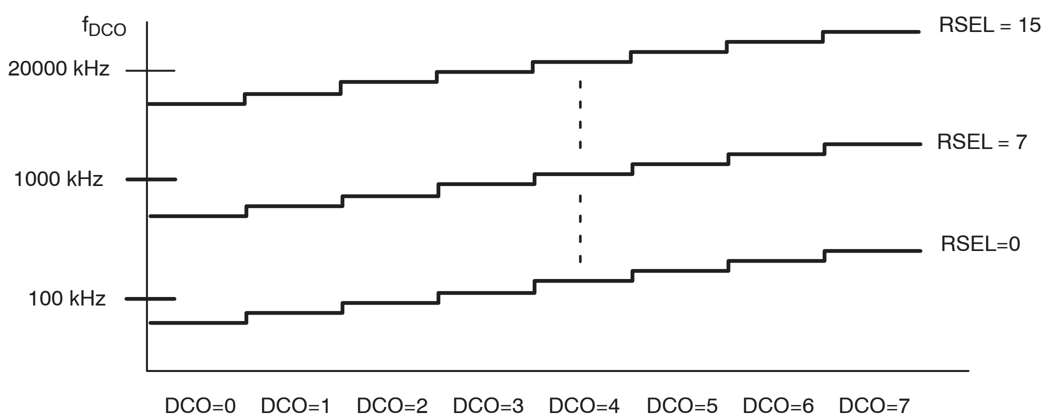
### Adjusting the DCO frequency

After a PUC, RSELx = 7 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 typically begins from PUC in less than 2 μs. The typical DCOx and RSELx ranges and steps are shown in Figure 4−5.

The frequency of DCOCLK is set by the following functions:

- The four RSELx bits select one of sixteen 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 DCOx+1. When DCOx = 07h, the MODx bits have no effect because the DCO is already at the highest setting for the selected RSELx range.

*Figure 4−5. Typical DCOx Range and RSELx Steps*



Each MSP430F2xx device has calibrated DCOCTL and BCSCTL1 register settings for specific frequencies stored in information memory segment A. To use the calibrated settings, the information is copied into the DCOCTL and BCSCTL1 registers. The calibrated settings affect the DCOx, MODx, and RSELx bits, and clear all other bits, except XT2OFF which remains set. The remaining bits of BCSCTL1 can be set or cleared as needed with `BIS.B` or `BIC.B` instructions.

```
; Set DCO to 1 MHz:
MOV.B  &CALBC1_1MHZ,&BCSCTL1 ; Set range
MOV.B  &CALDCO_1MHZ,&DCOCTL  ; Set DCO step + modulation
```

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

Some MSP430F2xx devices provide the option to source the DCO current through an external resistor, $R_{OSC}$, tied to $DV_{CC}$, when DCOR = 1. In this case, the DCO has the same characteristics as MSP430x1xx devices, and the RSELx setting is limited to 0 to 7 with the RSEL3 ignored. This option provides an additional method to tune the DCO frequency by varying the resistor value. See the device-specific datasheet for parameters.

### 4.2.6 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.

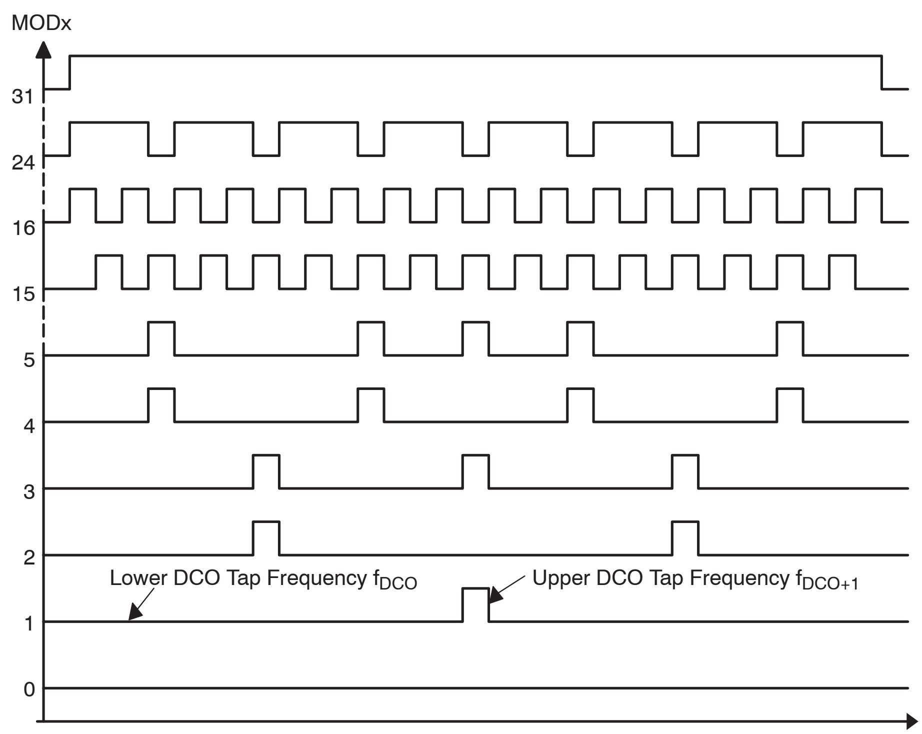
The modulator mixing formula is:

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

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 4−6 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.msp430.com for application notes and example code on configuring the DCO.

*Figure 4−6. Modulator Patterns*



### 4.2.7 Basic Clock Module+ Fail-Safe Operation

The basic clock module+ incorporates an oscillator-fault fail-safe feature. This feature detects an oscillator fault for LFXT1 and XT2 as shown in Figure 4−7 The available fault conditions are:

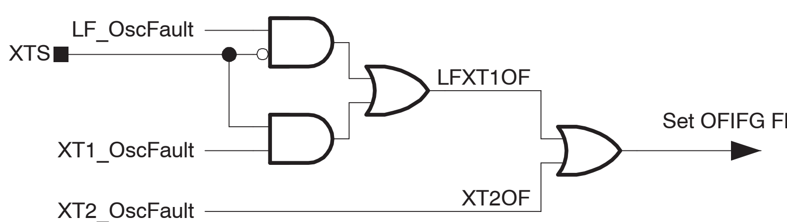
- Low-frequency oscillator fault (LFXT1OF) for LFXT1 in LF mode
- High-frequency oscillator fault (LFXT1OF) for LFXT1 in HF mode
- High-frequency oscillator fault (XT2OF) for XT2

The crystal oscillator fault bits LFXT1OF, and XT2OF are set if the corresponding crystal oscillator is turned on and not operating properly. The fault bits remain set as long as the fault condition exists and are automatically cleared if the enabled oscillators function normally.

The OFIFG oscillator-fault flag is set and latched at POR or when an oscillator fault (LFXT1OF, or XT2OF) is detected. When OFIFG is set, MCLK is sourced from the DCO, and if OFIE is set, the OFIFG requests and NMI interrupt. When the interrupt is granted, the OFIE is reset automatically. The OFIFG flag must be cleared by software. The source of the fault can be identified by checking the individual fault bits.

If a fault is detected for the crystal oscillator sourcing the MCLK, the MCLK is automatically switched to the DCO for its clock source. This does not change the SELMx bit settings. This condition must be handled by user software.

*Figure 4−7. Oscillator-Fault Logic*



## 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 and select appropriate mode
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.W  #OSCOFF,SR              ; Turn on osc.
   BIS.B  #XTS,&BCSCTL1           ; HF mode
   MOV.B  #LFXT1S0,&BCSCTL3       ; 1-3MHz Crystal
L1 BIC.B  #OFIFG,&IFG1            ; Clear OFIFG
   MOV.W  #0FFh,R15               ; Delay
L2 DEC.W  R15                     ;
   JNZ    L2                      ;
   BIT.B  #OFIFG,&IFG1            ; Re-test OFIFG
   JNZ    L1                      ; Repeat test if needed
   BIS.B  #SELM1+SELM0,&BCSCTL2   ; Select LFXT1CLK
```

### 4.2.8 Synchronization of Clock Signals

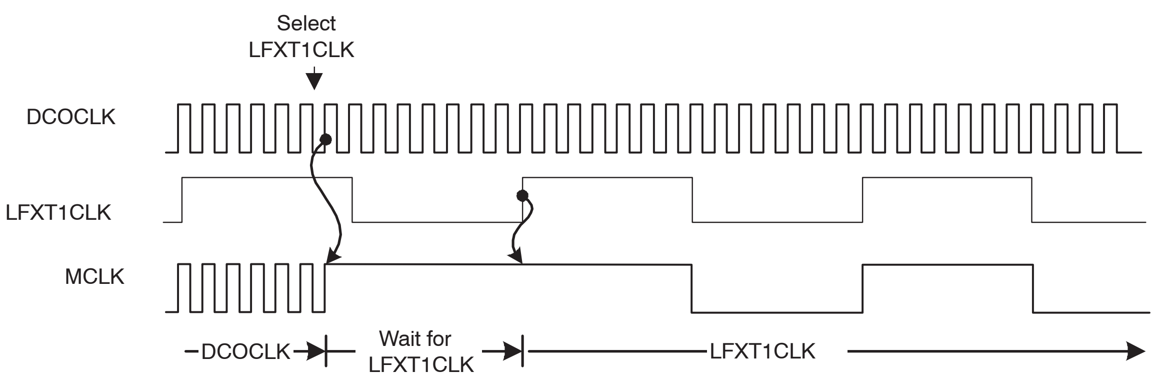
When switching MCLK or SMCLK from one clock source to the another, the switch is synchronized to avoid critical race conditions as shown in Figure 4−8:

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 4−8. 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 | 060h with PUC |
| Basic clock system control 1 | BCSCTL1 | Read/write | 057h | 087h with POR |
| Basic clock system control 2 | BCSCTL2 | Read/write | 058h | Reset with PUC |
| Basic clock system control 3 | BCSCTL3 | Read/write | 053h | 005h with PUC |
| SFR interrupt enable register 1 | IE1 | Read/write | 000h | Reset with PUC |
| SFR interrupt flag register 1 | IFG1 | Read/write | 002h | 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 within the range defined by 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 | | RSELx | | | |
| rw–(1) | rw–(0) | rw–(0) | rw–(0) | rw–0 | rw–1 | rw–1 | rw–1 |

| | | |
|---|---|---|
| **XT2OFF** | Bit 7 | XT2 off. This bit turns off the XT2 oscillator<br>0 XT2 is on<br>1 XT2 is off if it is not used for MCLK or SMCLK. |
| **XTS** | Bit 6 | LFXT1 mode select.<br>0 Low frequency mode<br>1 High frequency mode |
| **DIVAx** | Bits 5-4 | Divider for ACLK<br>00 /1<br>01 /2<br>10 /4<br>11 /8 |
| **RSELx** | Bits 3-0 | Range Select. Sixteen different frequency ranges are available. The lowest frequency range is selected by setting RSELx=0. RSEL3 is ignored when DCOR = 1. |

## 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 |

† Does not apply to MSP430x20xx or MSP430x21xx.

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

## BCSCTL3, Basic Clock System Control Register 3

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| XT2Sx | | LFXT1Sx | | XCAPx | | XT2OF† | LFXT1OF |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–1 | r0 | r–(1) |

† Does not apply to MSP430x2xx, MSP430x21xx, or MSP430x22xx devices

| | | |
|---|---|---|
| **XT2Sx** | Bits 7-6 | XT2 range select. These bits select the frequency range for XT2.<br>00 0.4 – 1MHz crystal or resonator<br>01 1 – 3MHz crystal or resonator<br>10 3 – 16MHz crystal or resonator<br>11 Digital external 0.4 – 16MHz clock source |
| **LFXT1Sx** | Bits 5-4 | Low-frequency clock select and LFXT1 range select. These bits select between LFXT1 and VLO when XTS = 0, and select the frequency range for LFXT1 when XTS = 1.<br>When XTS = 0:<br>00 32768 Hz Crystal on LFXT1<br>01 Reserved<br>10 VLOCLK (Reserved in MSP430x21x1 devices)<br>11 Digital external clock source<br>When XTS = 1 (Not applicable for MSP430x20xx devices)<br>00 0.4 – 1MHz crystal or resonator<br>01 1 – 3MHz crystal or resonator<br>10 3 – 16MHz crystal or resonator<br>11 Digital external 0.4 – 16MHz clock source |
| **XCAPx** | Bits 3-2 | Oscillator capacitor selection. These bits select the effective capacitance seen by the LFXT1 crystal when XTS = 0.<br>00 ~1pF<br>01 ~6pF<br>10 ~10pF<br>11 ~12.5pF |
| **XT2OF** | Bit 1 | XT2 oscillator fault.<br>0 No fault condition present<br>1 Fault condition present |
| **LFXT1OF** | Bit 0 | LFXT1 oscillator fault<br>0 No fault condition present<br>1 Fault condition present |

### IE1, Interrupt Enable Register 1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | | | **OFIE** | |
| | | | | | | rw–0 | |

| | | |
|---|---|---|
| | 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.<br>0 Interrupt not enabled<br>1 Interrupt enabled |
| | Bits 0 | This bit may be used by other modules. See device-specific datasheet. |

### IFG1, Interrupt Flag Register 1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | | | **OFIFG** | |
| | | | | | | rw–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.<br>0 No interrupt pending<br>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 MSP430x2xx 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:

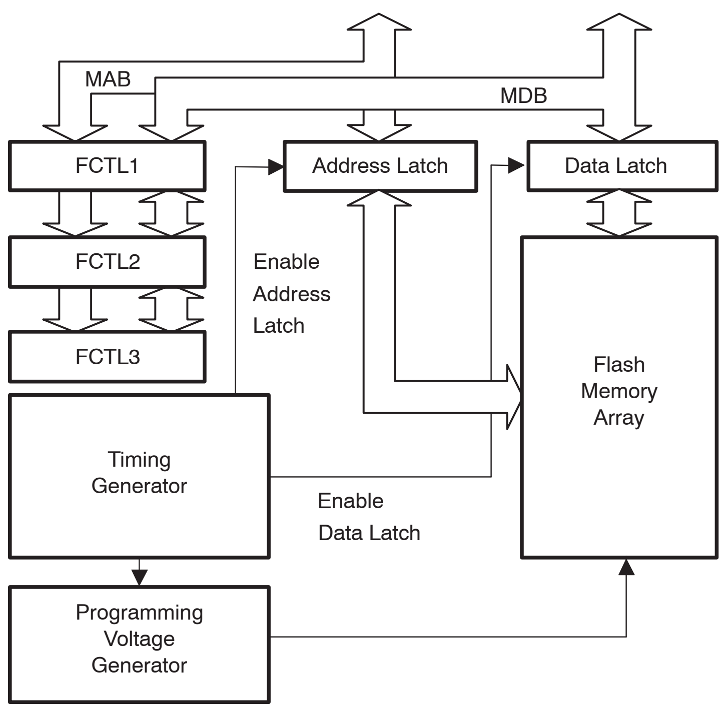
- ❑ Internal programming voltage generation
- ❑ Bit, byte or word programmable
- ❑ Ultralow-power operation
- ❑ Segment erase and mass erase

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 2.2 V. If $V_{CC}$ falls below 2.2 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.

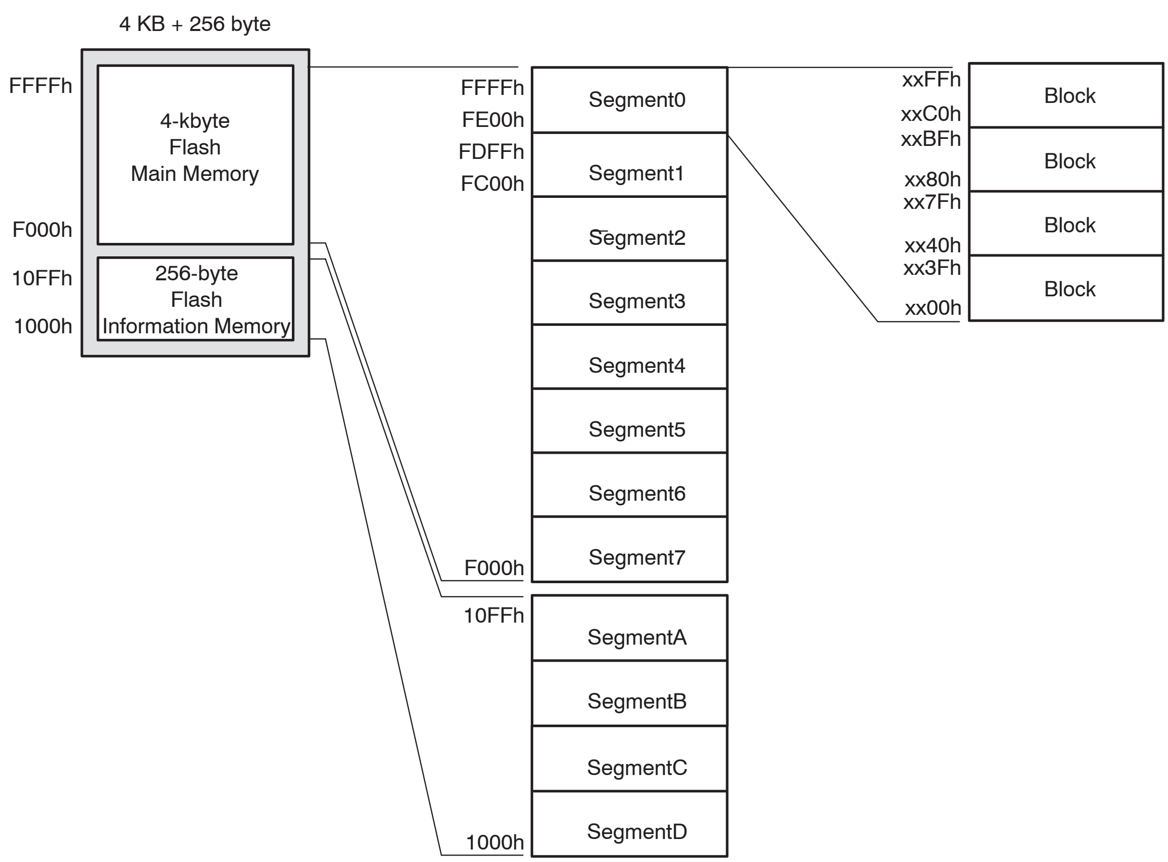
The flash memory is partitioned into main and information memory sections. There is no difference in the operation of the main and information memory sections. Code or data can be located in either section. The differences between the two sections are the segment size and the physical addresses.

The information memory has four 64-byte segments. The main memory has two or more 512-byte segments. See the device-specific datasheet for the complete memory map of a device.

The segments are further divided into blocks. A block is 64 bytes, starting at 0xx00h, 0xx40h, 0xx80h, or 0xxC0h, and ending at 0xx3Fh, 0xx7Fh, 0xxBFh, or 0xxFFh.

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

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



### 5.2.1 SegmentA

SegmentA of the information memory is locked separately from all other segments with the LOCKA bit. When LOCKA = 1, SegmentA cannot be written or erased and all information memory is protected from erasure during a mass erase. When LOCKA = 0, SegmentA can be erased and written as any other flash memory segment, and all information memory is erased during a mass erase. Segments B, C, and D can always be erased with a segment erase, regardless of the state of the LOCKA bit.

The state of the LOCKA bit is toggled when a 1 is written to it. Writing a 0 to LOCKA has no affect. This allows existing flash programming routines to be used unchanged.

```
; Unlock SegmentA
   BIT    #LOCKA,&FCTL3             ; Test LOCKA
   JZ     SEGA_UNLOCKED             ; Already unlocked?
   MOV    #FWKEY+LOCKA,&FCTL3       ; No, unlock SegmentA
SEGA_UNLOCKED                       ; Yes, continue
; SegmentA is unlocked


; Lock SegmentA
   BIT    #LOCKA,&FCTL3             ; Test LOCKA
   JNZ    SEGALOCKED                ; Already locked?
   MOV    #FWKEY+LOCKA,&FCTL3       ; No, lock SegmentA
SEGA_LOCKED                         ; Yes, continue
; SegmentA is locked
```

## 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:

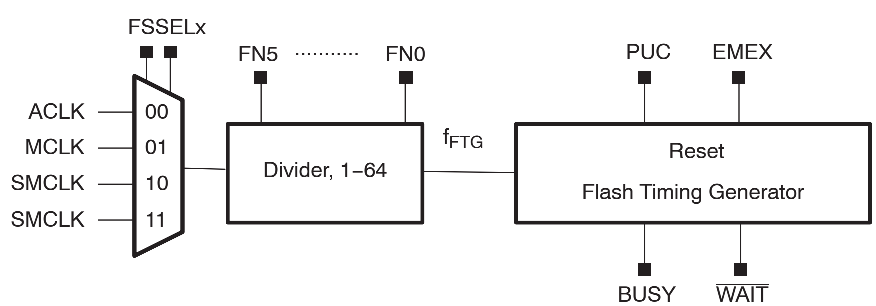
- ❑ Byte/word write
- ❑ Block write
- ❑ Segment Erase
- ❑ Mass Erase (all main memory segments)
- ❑ All Erase (all segments)

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*



### Flash Timing Generator Clock Selection

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.

If a clock failure is detected during a write or erase operation, the operation is aborted, the FAIL flag is set, and the result of the operation is unpredictable.

While a write or erase operation is active the selected clock source can not be disabled by putting the MSP430 into a low-power mode. The selected clock source will remain active until the operation is completed before being disabled.

### 5.3.2 Erasing Flash Memory

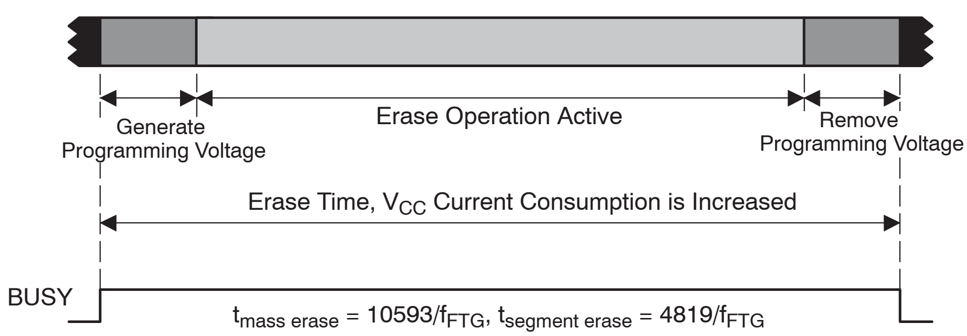
The erased level of a flash memory bit is 1. Each bit can be programmed from 1 to 0 individually but to reprogram from 0 to 1 requires an erase cycle. The smallest amount of flash that can be erased is a segment. There are three erase modes selected with the ERASE and MERAS bits listed in Table 5−1.

*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) if LOCKA = 0. Main segments only if LOCKA = 1. |

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 MSP430F2xx 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.

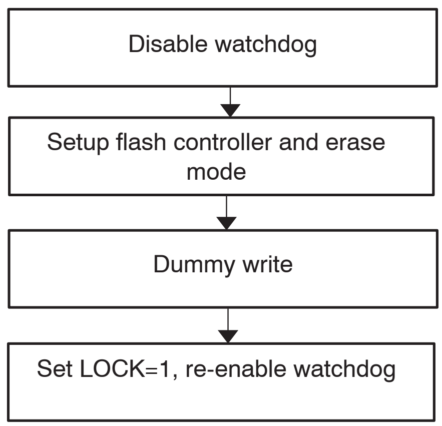
### Initiating an Erase from Within Flash Memory

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*



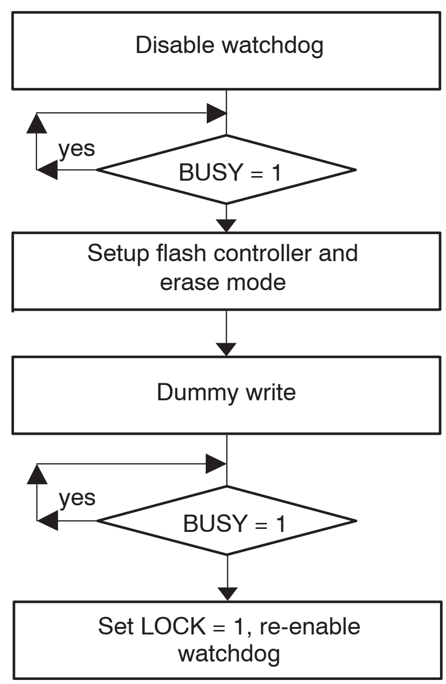

```
; Segment Erase from flash. 514 kHz < SMCLK < 952 kHz
; Assumes ACCVIE = NMIIE = OFIE = 0.
   MOV    #WDTPW+WDTHOLD,&WDTCTL    ; Disable WDT
   MOV    #FWKEY+FSSEL1+FN0,&FCTL2  ; SMCLK/2
   MOV    #FWKEY,&FCTL3             ; Clear LOCK
   MOV    #FWKEY+ERASE,&FCTL1       ; Enable segment erase
 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

### 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 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
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?
```

### 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 |

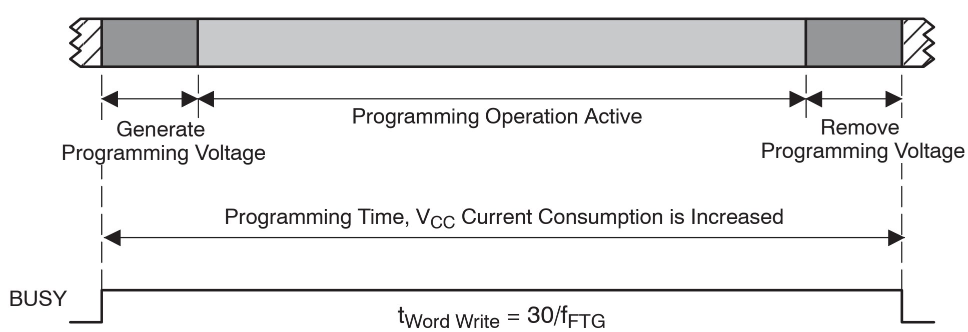
Both write modes use a sequence of individual write instructions, but using the block write mode is approximately twice as fast as byte/word mode, because the voltage generator remains on for the complete block write. Any instruction that modifies a destination can be used to modify a flash location in either byte/word mode or block-write mode. A flash word (low + high byte) must not be written more than twice between erasures. Otherwise, damage can occur.

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*



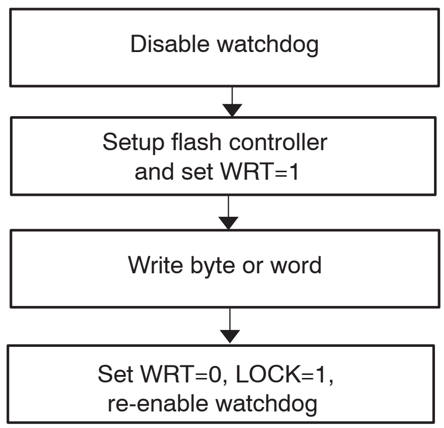

When a byte/word write is executed from RAM, the CPU continues to execute code from RAM. The BUSY bit must be zero before the CPU accesses flash again, otherwise an access violation occurs, ACCVIFG is set, and the write result is unpredictable.

In byte/word mode, the internally-generated programming voltage is applied to the complete 64-byte block, each time a byte or word is written, for 27 of the 30 $f_{FTG}$ cycles. With each byte or word write, the amount of time the block is subjected to the programming voltage accumulates. The cumulative programming time, $t_{CPT}$, must not be exceeded for any block. If the cumulative programming time is met, the block must be erased before performing any further writes to any address within the block. See the device-specific datasheet for specifications.

### 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*

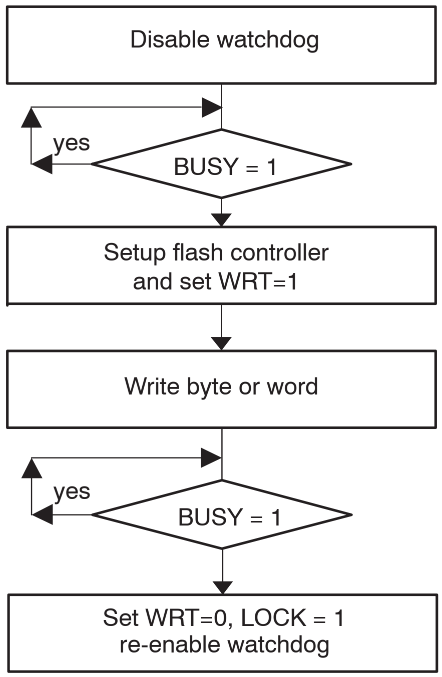



```
; Byte/word write from flash. 514 kHz < SMCLK < 952 kHz
; Assumes 0FF1Eh is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
   MOV   #WDTPW+WDTHOLD,&WDTCTL    ; Disable WDT
   MOV   #FWKEY+FSSEL1+FN0,&FCTL2  ; SMCLK/2
   MOV   #FWKEY,&FCTL3             ; Clear LOCK
   MOV   #FWKEY+WRT,&FCTL1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

## 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*



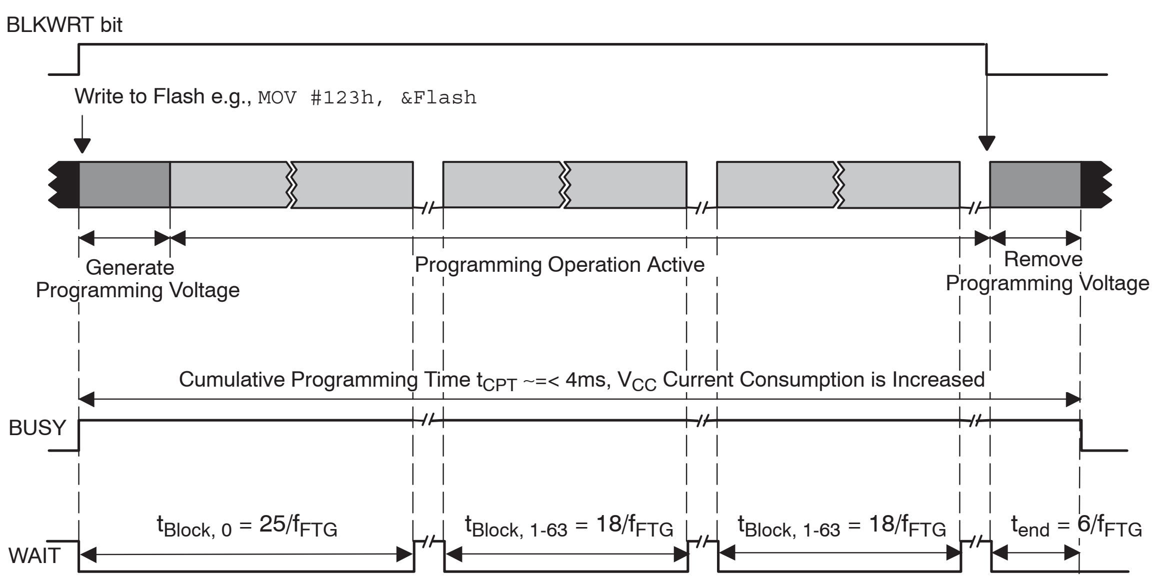

```
; Byte/word write from RAM. 514 kHz < SMCLK < 952 kHz
; Assumes 0FF1Eh is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
   MOV   #WDTPW+WDTHOLD,&WDTCTL    ; Disable WDT
L1 BIT   #BUSY,&FCTL3              ; Test BUSY
   JNZ   L1                        ; Loop while busy
   MOV   #FWKEY+FSSEL1+FN0,&FCTL2  ; 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?
```

### Block Write

The block write can be used to accelerate the flash write process when many sequential bytes or words need to be programmed. The flash programming voltage remains on for the duration of writing the 64-byte block. The cumulative programming time $t_{CPT}$ must not be exceeded for any block during a block write.

A block write cannot be initiated from within flash memory. The block write must be initiated from RAM 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 5−10 shows the block write timing.

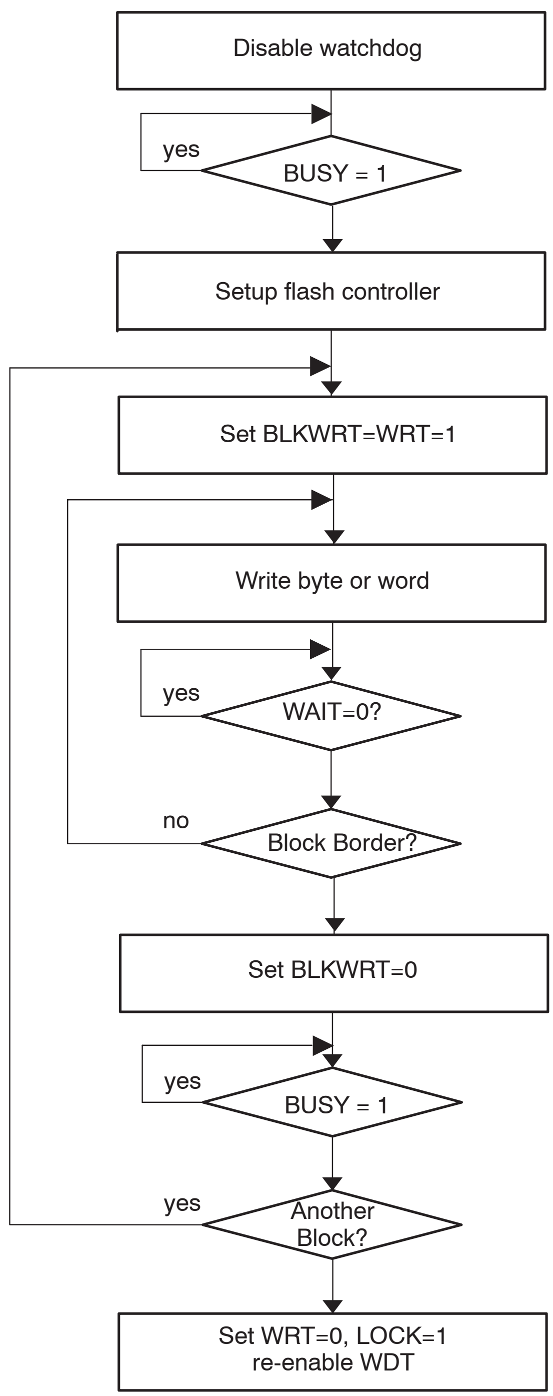
*Figure 5−10. Block-Write Cycle Timing*



## Block Write Flow and Example

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

*Figure 5−11. 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
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
```

### 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 = 0. 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, Flash is written |
| | Instruction fetch | 1 | ACCVIFG = 1, LOCK = 1 |

Interrupts are automatically disabled during any flash operation when EEI = 0 and EEIEX = 0 and on MSP430x20xx devices where EEI and EEIEX are not present. After the flash operation has completed, interrupts are automatically re-enabled. Any interrupt that occurred during the operation will have its associated flag set, and will generate an interrupt request when re-enabled.

When EEIEX = 1 and GIE = 1, an interrupt will immediately abort any flash operation and the FAIL flag will be set. When EEI = 1, GIE = 1, and EEIEX = 0, a segment erase will be interrupted by a pending interrupt every 32 $f_{FTG}$ cycles. After servicing the interrupt, the segment erase is continued for at least 32 $f_{FTG}$ cycles or until it is complete. During the servicing of the interrupt, the BUSY bit remains set but the flash memory can be accessed by the CPU without causing an access violation occurs. Nested interrupts are not supported.

The watchdog timer (in watchdog mode) should be disabled before a flash erase cycle. A reset will abort the erase and the result will be unpredictable. After the erase cycle has completed, the watchdog may be re-enabled.

### 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.msp430.com.

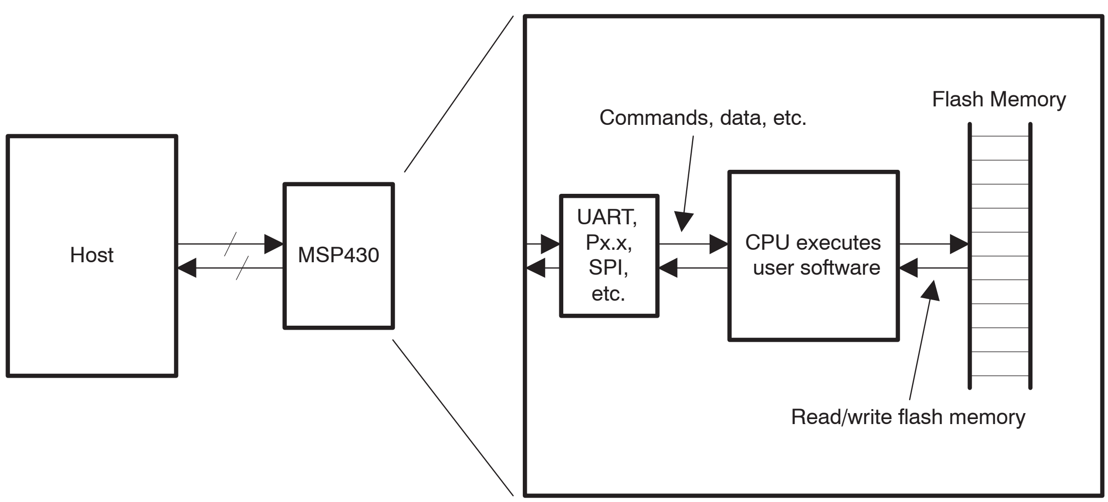
### Programming Flash Memory via the Bootstrap loader (BSL)

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/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 5−12. 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 5−12. 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 | 09658h 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<br>FWKEY, Must be written as 0A5h | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| BLKWRT | WRT | Reserved | EEIEX† | EEI† | MERAS | ERASE | Reserved |
| rw–0 | rw–0 | r0 | rw–0 | rw–0 | rw–0 | rw–0 | r0 |

† Not present on MSP430x20xx Devices

**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** Bit 5 Reserved. Always read as 0.

**EEIEX** Bit 4 Enable Emergency Interrupt Exit. Setting this bit enables an interrupt to cause an emergency exit from a flash operation when GIE = 1. EEIEX is automatically reset when EMEX is set.
- 0 Exit interrupt disabled.
- 1 Exit on interrupt enabled.

**EEI** Bits 3 Enable Erase Interrupts. Setting this bit allows a segment erase to be interrupted by an interrupt request. After the interrupt is serviced the erase cycle is resumed. EEI is automatically reset when EMEX is set.
- 0 Interrupts during segment erase disabled.
- 1 Interrupts during segment erase enabled.

**MERAS** Bit 2
**ERASE** 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 if LOCKA = 0. Main segments only if LOCKA = 1. |

| | | |
|---|---|---|
| **Reserved** | Bit 0 | Reserved. Always read as 0. |

## FCTL2, Flash Memory Control Register

| 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 |
|---|---|---|---|---|---|---|---|
| **FSSELx** | | **FNx** | | | | | |
| rw–0 | rw–1 | rw-0 | rw-0 | rw-0 | rw–0 | rw-1 | rw–0 |

| | | |
|---|---|---|
| **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<br>00 ACLK<br>01 MCLK<br>10 SMCLK<br>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=03Fh the divisor is 64. |

## FCTL3, Flash Memory Control Register FCTL3

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| FWKEYx, Read as 096h<br>Must be written as 0A5h | | | | | | | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| FAIL | LOCKA | EMEX | LOCK | WAIT | ACCVIFG | KEYV | BUSY |
| r(w)–0 | r(w)–1 | 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.

**FAIL** Bit 7 Operation failure. This bit is set if the $f_{FTG}$ clock source fails, or a flash operation is aborted from an interrupt when EEIEX = 1. FAIL must be reset with software.
- 0 No failure
- 1 Failure

**LOCKA** Bit 6 SegmentA and Info lock. Write a 1 to this bit to change its state. Writing 0 has no effect.
- 0 Segment A unlocked and all information memory is erased during a mass erase.
- 1 Segment A locked and all information memory is protected from erasure during a mass erase.

**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. ACCVIFG must be reset with software.
- 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.<br>0 FCTLx password was written correctly<br>1 FCTLx password was written incorrectly |
| **BUSY** | Bit 0 | Busy. This bit indicates the status of the flash timing generator.<br>0 Not Busy<br>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.<br>0 Interrupt not enabled<br>1 Interrupt enabled |

Chapter 6

# Digital I/O

This chapter describes the operation of the digital I/O ports.

**Topic** **Page**

## 6.1 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
- Individually configurable pull-up or pull-down resistors

## 6.2 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.

### 6.2.1 Input Register PxIN

Each bit in each PxIN 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.

### 6.2.2 Output Registers PxOUT

Each bit in each PxOUT register is the value to be output on the corresponding I/O pin when the pin is configured as I/O function, output direction, and the pull-up/down resistor is disabled.

Bit = 0: The output is low

Bit = 1: The output is high

If the pin's pull−up/down resistor is enabled, the corresponding bit in the PxOUT register selects pull-up or pull-down.

Bit = 0: The pin is pulled down

Bit = 1: The pin is pulled up

### 6.2.3 Direction Registers PxDIR

Each bit in each PxDIR register selects the direction of the corresponding I/O pin, regardless of the selected function for the pin. PxDIR bits for I/O pins that are selected for other 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

### 6.2.4 Pull−Up/Down Resistor Enable Registers PxREN

Each bit in each PxREN register enables or disables the pull-up/down resistor of the corresponding I/O pin. The corresponding bit in the PxOUT register selects if the pin is pulled up or pulled down.

Bit = 0: Pull-up/down resistor disabled

Bit = 1: Pull-up/down resistor enabled

### 6.2.5 Function Select Registers PxSEL

Port pins are often multiplexed with other peripheral module functions. See the device-specific data sheet to determine pin functions. Each PxSEL 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 PxSELx = 1 does not automatically set the pin direction. Other peripheral module functions may require the PxDIRx bits to be configured according to the direction needed for the module function. See the pin schematics in the device-specific datasheet.

```
;Output ACLK on P2.0 on MSP430F21x1
   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 PxSEL = 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 PxSELx=1, the internal input signal follows the signal at the pin. However, if the PxSELx=0, the input to the peripheral maintains the value of the input signal at the device pin before the PxSELx bit was reset.

### 6.2.6 P1 and P2 Interrupts

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

#### Interrupt Flag Registers P1IFG, P2IFG

Each PxIFGx 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 PxIFGx interrupt flags request an interrupt when their corresponding PxIE bit and the GIE bit are set. Each PxIFG flag must be reset with software. Software can also set each PxIFG 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 PxIFGx 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 PxIFGx flag generates another interrupt. This ensures that each transition is acknowledged.

**Note: PxIFG Flags When Changing PxOUT or PxDIR**

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

### Interrupt Edge Select Registers P1IES, P2IES

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

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

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

**Note: Writing to PxIESx**

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

| PxIESx | PxINx | PxIFGx |
|---|---|---|
| $0 \rightarrow 1$ | 0 | May be set |
| $0 \rightarrow 1$ | 1 | Unchanged |
| $1 \rightarrow 0$ | 0 | Unchanged |
| $1 \rightarrow 0$ | 1 | May be set |

### Interrupt Enable P1IE, P2IE

Each PxIE bit enables the associated PxIFG interrupt flag.

Bit = 0: The interrupt is disabled

Bit = 1: The interrupt is enabled

## 6.2.7 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 prevent a floating input and reduce power consumption. The value of the PxOUT bit is don’t care, since the pin is unconnected. Alternatively, the integrated pull-up/down resistor can be enabled by setting the PxREN bit of the unused pin to prevent the floating input. See chapter *System Resets, Interrupts, and Operating Modes* for termination unused pins.

## 6.3 Digital I/O Registers

The digital I/O registers are listed in Table 6−1.

*Table 6−1. 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 |
| | Resistor Enable | P1REN | 027h | 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 | 0C0h with PUC |
| | Resistor Enable | P2REN | 02Fh | 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 |
| | Resistor Enable | P3REN | 010h | 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 |
| | Resistor Enable | P4REN | 011h | 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 |
| | Resistor Enable | P5REN | 012h | 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 |
| | Resistor Enable | P6REN | 013h | Read/write | Reset with PUC |

Chapter 7

# Watchdog Timer+

The watchdog timer+ (WDT+) is a 16-bit timer that can be used as a watchdog or as an interval timer. This chapter describes the WDT+ The WDT+ is implemented in all MSP430x2xx devices.

Topic Page

## 7.1 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:

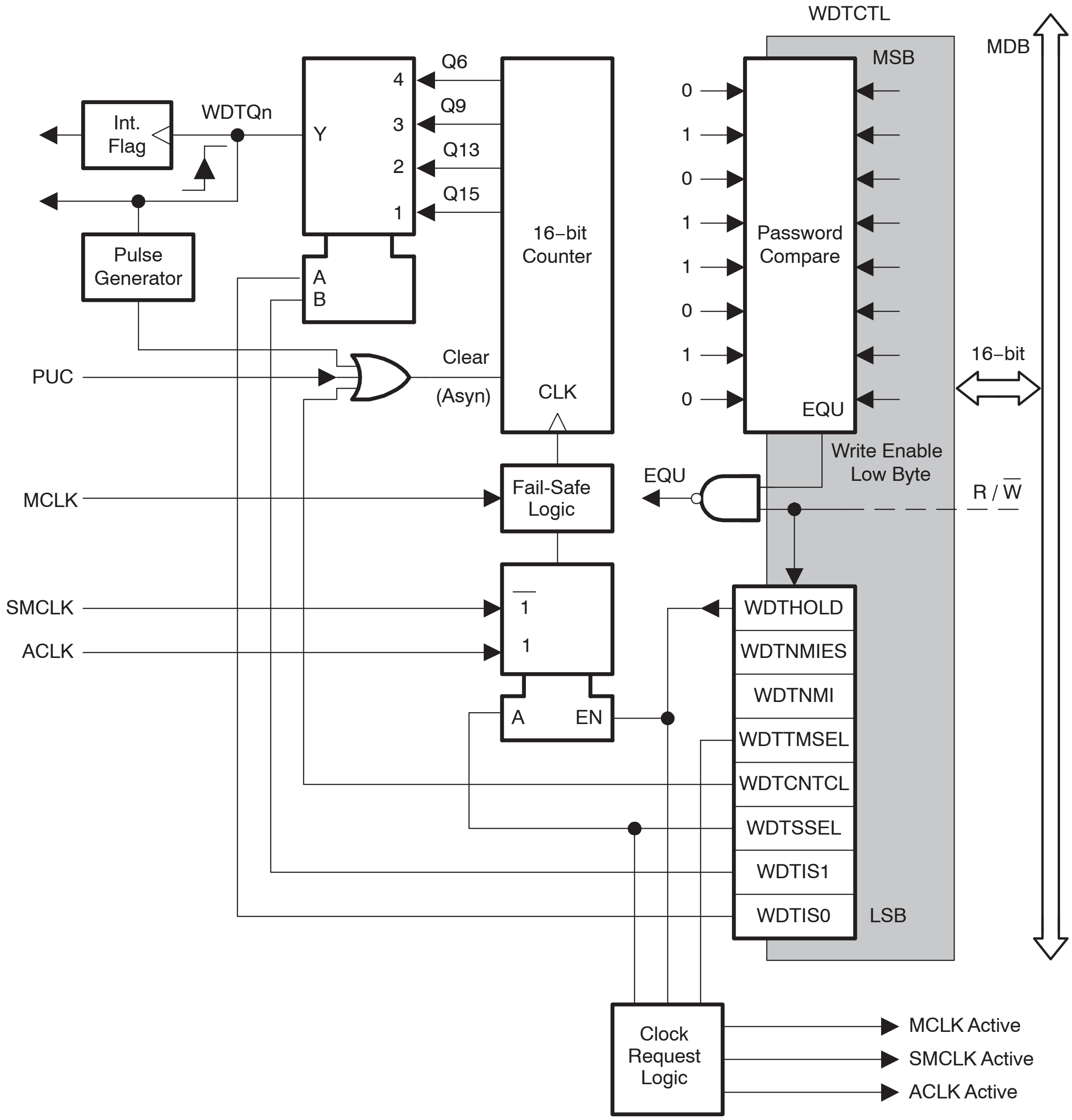
- Four software-selectable time intervals
- Watchdog mode
- Interval mode
- Access to WDT+ control register is password protected
- Control of $\overline{\text{RST}}$/NMI pin function
- Selectable clock source
- Can be stopped to conserve power
- Clock fail-safe feature

The WDT+ block diagram is shown in Figure 7−1.

**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 7−1. Watchdog Timer+ Block Diagram*



## 7.2 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.

### 7.2.1 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.

### 7.2.2 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.

### 7.2.3 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.

### 7.2.4 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.

### 7.2.5 Watchdog timer+ Clock Fail−safe Operation

The WDT+ module provides a fail-safe clocking feature assuring the clock to the WDT+ cannot be disabled while in watchdog mode. This means the low-power modes may be affected by the choice for the WDT+ clock. For example, if ACLK is the WDT+ clock source, LPM4 will not be available, because the WDT+ will prevent ACLK from being disabled. Also, if ACLK or SMCLK fail while sourcing the WDT+, the WDT+ clock source is automatically switched to MCLK. In this case, if MCLK is sourced from a crystal, and the crystal has failed, the fail-safe feature will activate the DCO and use it as the source for MCLK.

When the WDT+ module is used in interval timer mode, there is no fail-safe feature for the clock source.

### 7.2.6 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 the WDT+ will keep SMCLK enabled for its clock source, increasing the current consumption of LPM3. When the watchdog timer+ is not required, the WDTHOLD bit can be used to hold the WDTCNT, reducing power consumption.

### 7.2.7 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
```

## 7.3 Watchdog Timer+ Registers

The WDT+ registers are listed in Table 7−1.

*Table 7−1. 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† |

† WDTIFG is reset with POR

## WDTCTL, Watchdog timer+ Register

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| **Read as 069h WDTPW, 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 |

**WDTPW** 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

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | NMIIE | | | | WDTIE |
| | | | | | | | rw–0 |

| | | |
|---|---|---|
| | Bits 7-5 | These bits may be used by other modules. See device-specific datasheet. |
| **NMIIE** | Bit 4 | NMI interrupt enable. This bit enables the NMI 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.<br>0 Interrupt not enabled<br>1 Interrupt enabled |
| | Bits 3-1 | 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.<br>0 Interrupt not enabled<br>1 Interrupt enabled |

## IFG1, Interrupt Flag Register 1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | NMIIFG | | | | WDTIFG |
| | | | | | | | rw–(0) |

| | | |
|---|---|---|
| | Bits 7-5 | These bits may be used by other modules. See device-specific datasheet. |
| **NMIIFG** | Bit 4 | NMI interrupt flag. NMIIFG must be reset by software. Because other bits in IFG1 may be used for other modules, it is recommended to clear NMIIFG by using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions.<br>0 No interrupt pending<br>1 Interrupt pending |
| | Bits 3-1 | 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.<br>0 No interrupt pending<br>1 Interrupt pending |

## Chapter 8

# Timer_A

Timer_A is a 16-bit timer/counter with multiple capture/compare registers. This chapter describes Timer_A. Timer_A3 (three capture/compare registers) is implemented in all MSP430x2xx devices, except for MSP430x20xx devices. Those devices implement Timer_A2 (two capture/compare registers).

**Topic** **Page**

## 8.1 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:

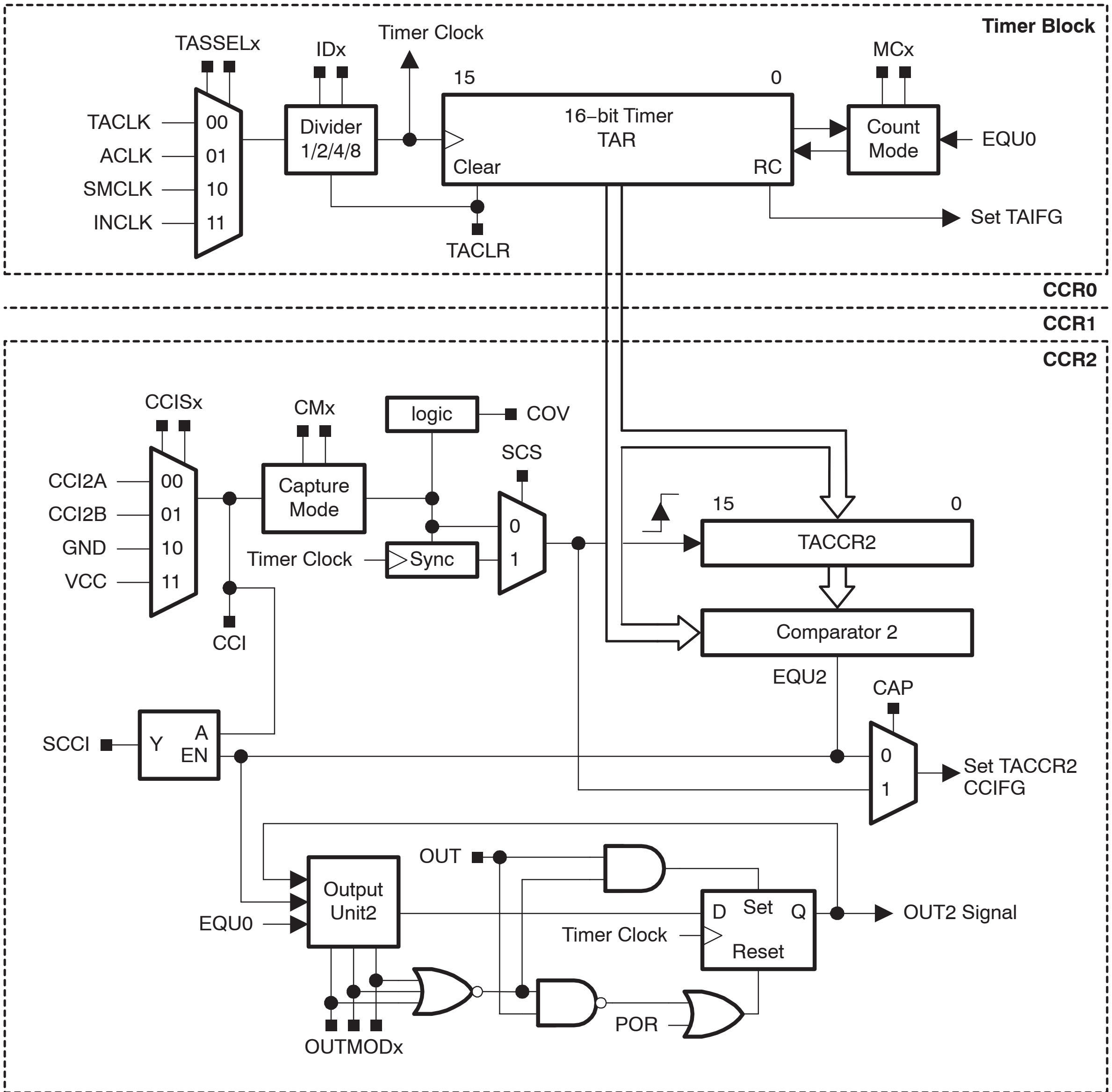
- ❏ Asynchronous 16-bit timer/counter with four operating modes
- ❏ Selectable and configurable clock source
- ❏ Two or Three configurable capture/compare registers
- ❏ Configurable outputs with PWM capability
- ❏ Asynchronous input and output latching
- ❏ Interrupt vector register for fast decoding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

The block diagram of Timer_A is shown in Figure 8−1.

**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 8−1. Timer_A Block Diagram*



## 8.2 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.

### 8.2.1 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, interrupt flag, and TACLR) to avoid errant operating conditions.

When the timer clock is asynchronous to the CPU clock, any read from TAR should occur while the timer is not operating or the results may be unpredictable. Alternatively, the timer may be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Any write to TAR will take effect immediately.

#### Clock Source Select and Divider

The timer clock 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. The timer clock divider is reset when TACLR is set.

### 8.2.2 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.

### 8.2.3 Timer Mode Control

The timer has four modes of operation as described in Table 8−1: stop, up, continuous, and up/down. The operating mode is selected with the MCx bits.

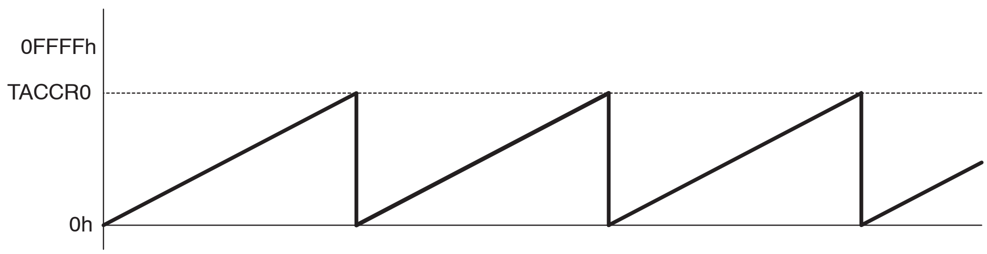
*Table 8−1. Timer Modes*

| 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

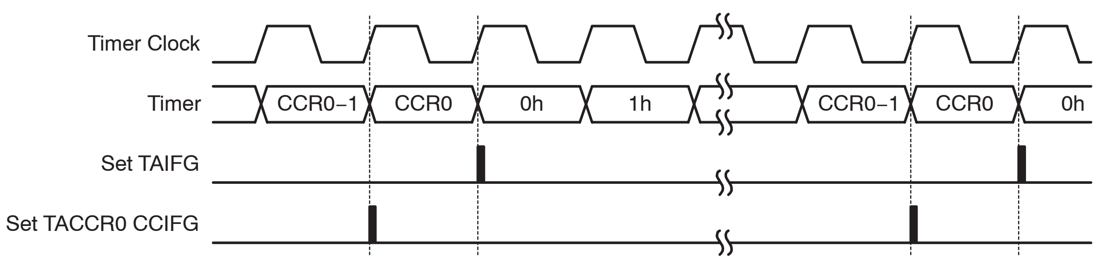
The up mode is used if the timer period must be different from 0FFFFh counts. The timer repeatedly counts up to the value of compare register TACCR0, which defines the period, as shown in Figure 8−2. The number of timer counts in the period is TACCR0+1. When the timer value equals TACCR0 the timer restarts counting from zero. If up mode is selected when the timer value is greater than TACCR0, the timer immediately restarts counting from zero.

*Figure 8−2. Up Mode*




The TACCR0 CCIFG interrupt flag is set when the timer *counts* to the TACCR0 value. The TAIFG interrupt flag is set when the timer *counts* from TACCR0 to zero. Figure 8−3 shows the flag set cycle.

*Figure 8−3. Up Mode Flag Setting*



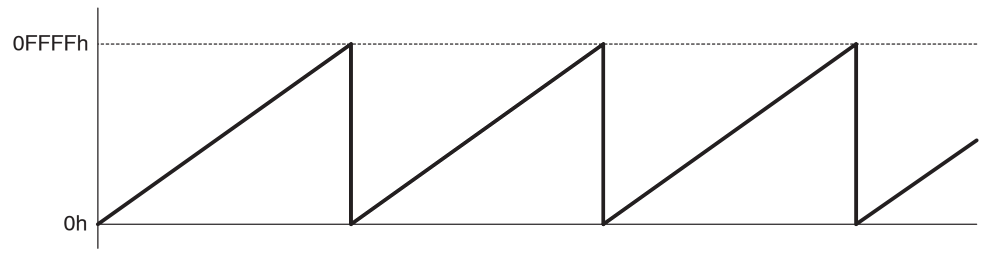

### *Changing the Period Register TACCR0*

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

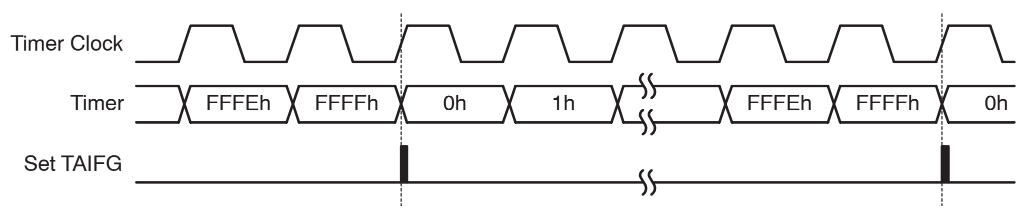
In the continuous mode, the timer repeatedly counts up to 0FFFFh and restarts from zero as shown in Figure 8−4. The capture/compare register TACCR0 works the same way as the other capture/compare registers.

*Figure 8−4. Continuous Mode*




The TAIFG interrupt flag is set when the timer *counts* from 0FFFFh to zero. Figure 8−5 shows the flag set cycle.

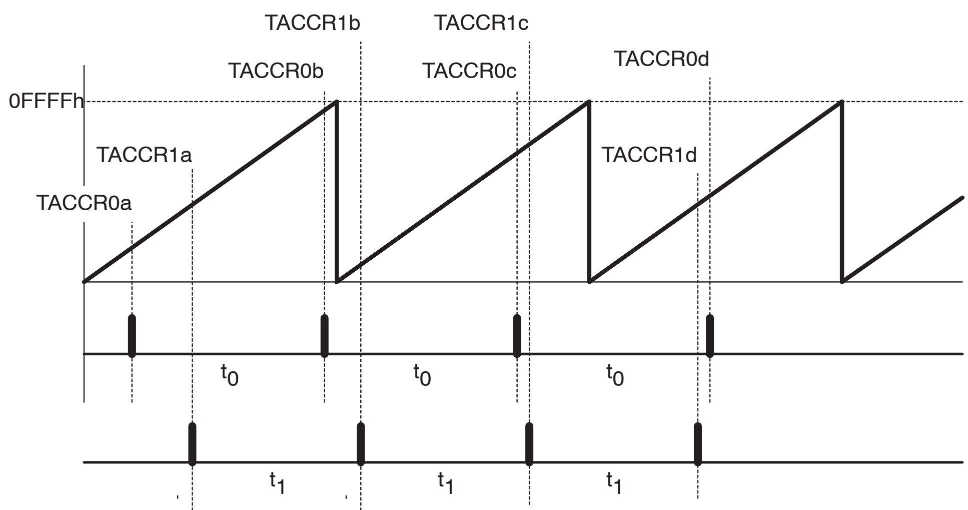
*Figure 8−5. Continuous Mode Flag Setting*



### 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 8−6 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 8−6. Continuous Mode Time Intervals*

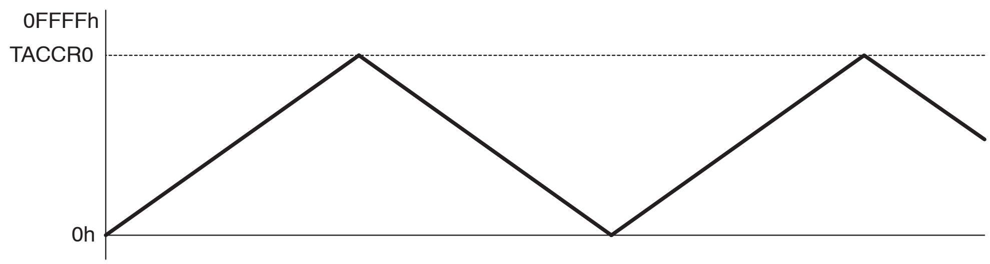



Time intervals can be produced with other modes as well, where TACCR0 is used as the period register. Their handling is more complex since the sum of the old TACCRx data and the new period can be higher than the TACCR0 value. When the previous TACCRx value plus $t_x$ is greater than the TACCR0 data, 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 8−7. The period is twice the value in TACCR0.

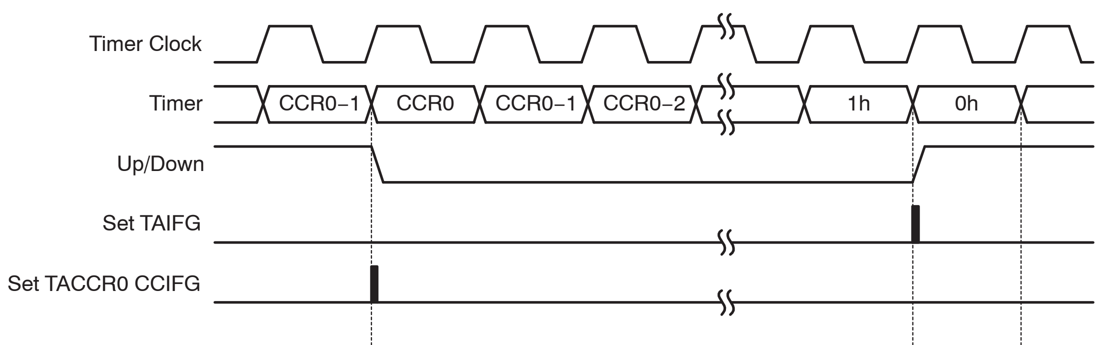
*Figure 8−7. Up/Down Mode*




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 timer clock 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 8−8 shows the flag set cycle.

*Figure 8−8. 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 8−9 the $t_{dead}$ is:

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

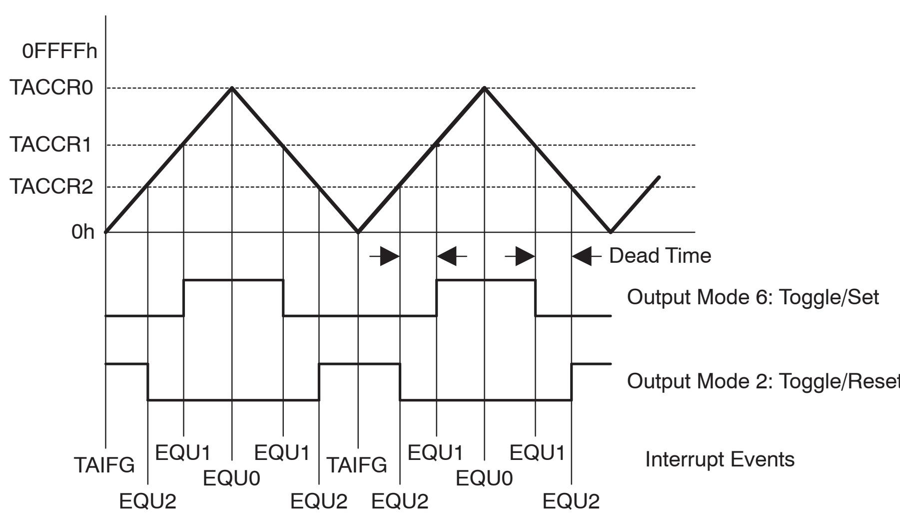
With: $t_{dead}$ Time during which both outputs need to be inactive

$t_{timer}$ 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 8−9. Output Unit in Up/Down Mode*



### 8.2.4 Capture/Compare Blocks

Two or 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

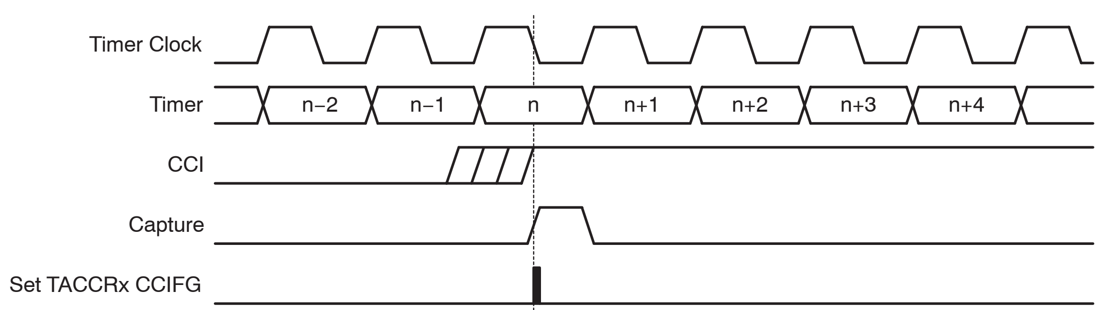
The capture mode is selected when CAP = 1. Capture mode is used to record time events. It can be used for speed computations or time measurements. The capture inputs CCIxA and CCIxB are connected to external pins or internal signals and are selected with the CCISx bits. The CMx bits select the capture 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 via the CCI bit. MSP430x2xx family devices may have different signals connected to CCIxA and CCIxB. Refer to the device-specific datasheet for the connections of these signals.

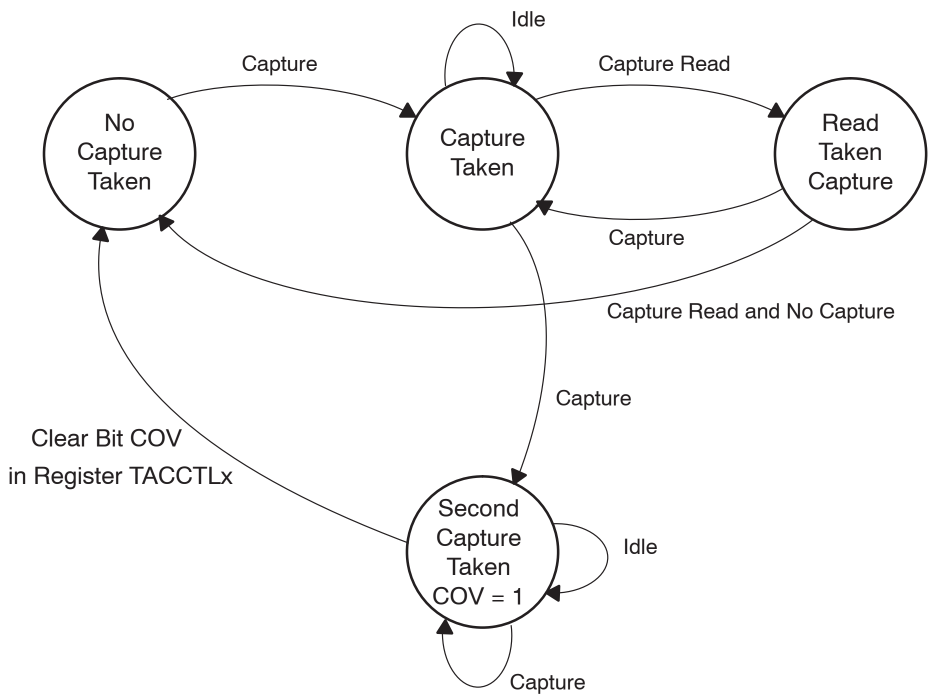
The capture signal can be asynchronous to the timer clock and cause a race condition. Setting the SCS bit will synchronize the capture with the next timer clock. Setting the SCS bit to synchronize the capture signal with the timer clock is recommended. This is illustrated in Figure 8−10.

*Figure 8−10. 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 8−11. COV must be reset with software.

*Figure 8−11. 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 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,&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

### 8.2.5 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 8−2. 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.

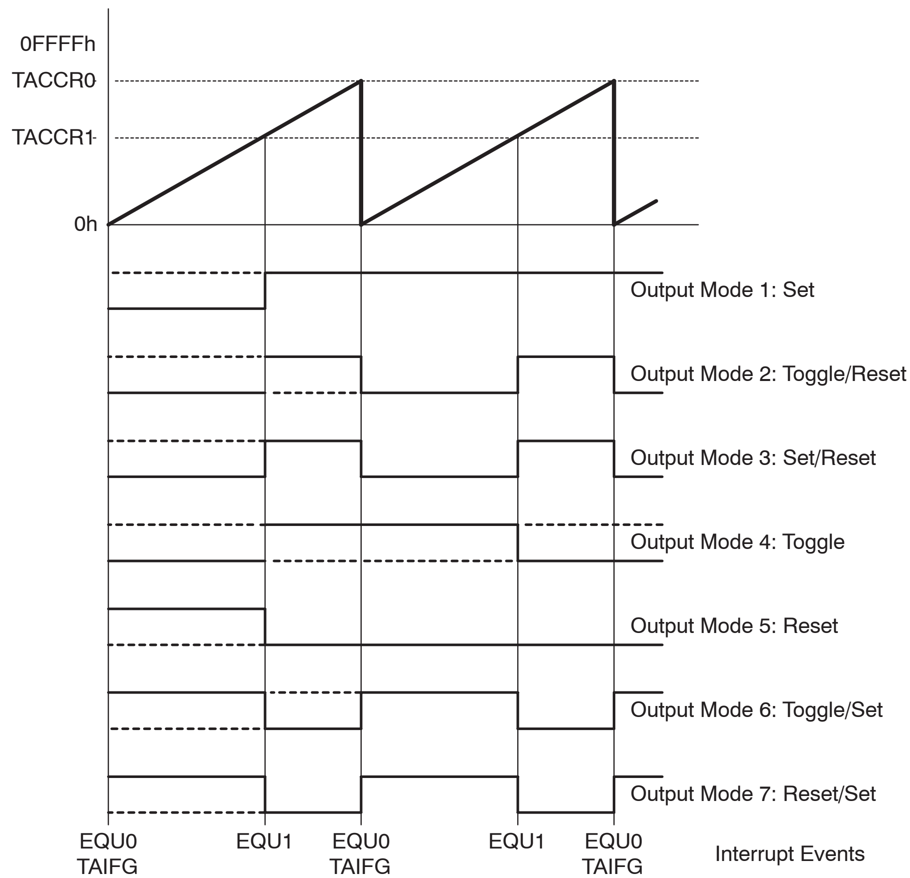
*Table 8−2. Output Modes*

| 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 8−12 using TACCR0 and TACCR1.

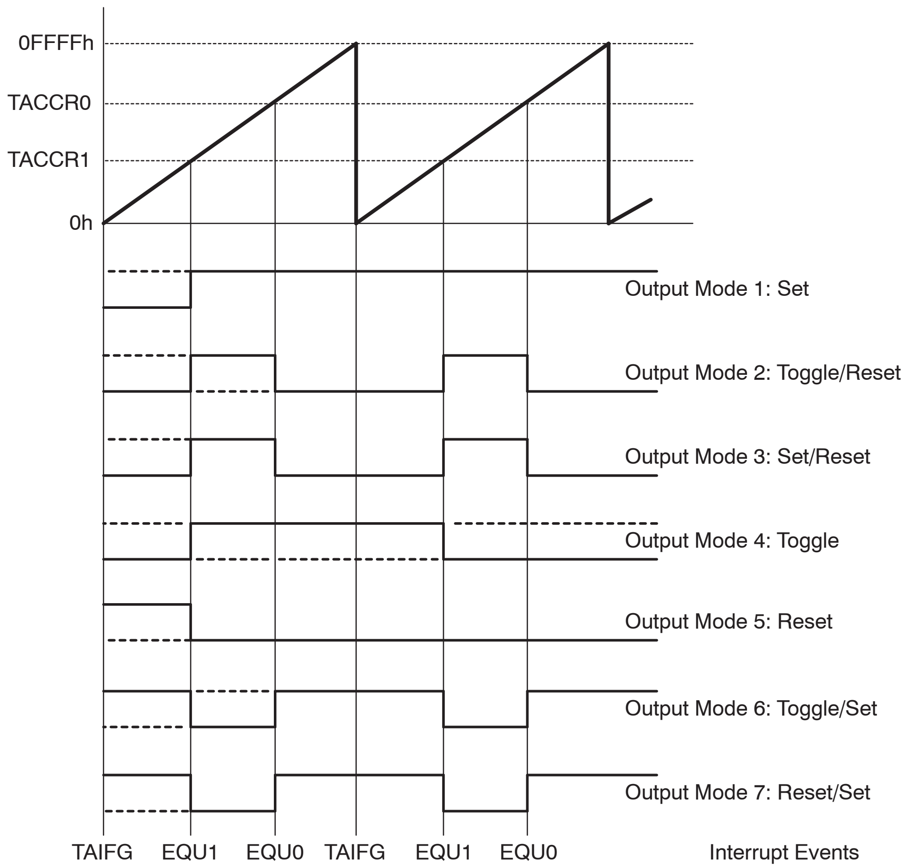
*Figure 8−12. Output Example—Timer in Up Mode*



### *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 8−13 using TACCR0 and TACCR1.

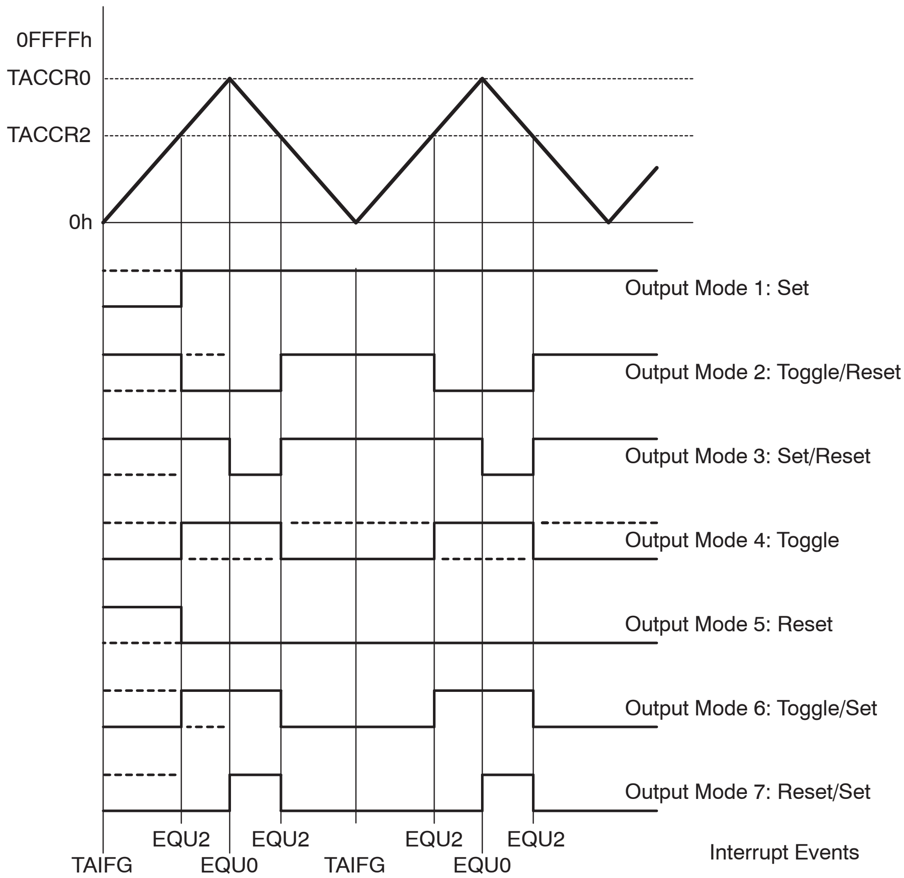
*Figure 8−13. Output Example—Timer in Continuous Mode*



### *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 8−14 using TACCR0 and TACCR2.

*Figure 8−14. 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
```

### 8.2.6 Timer_A Interrupts

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

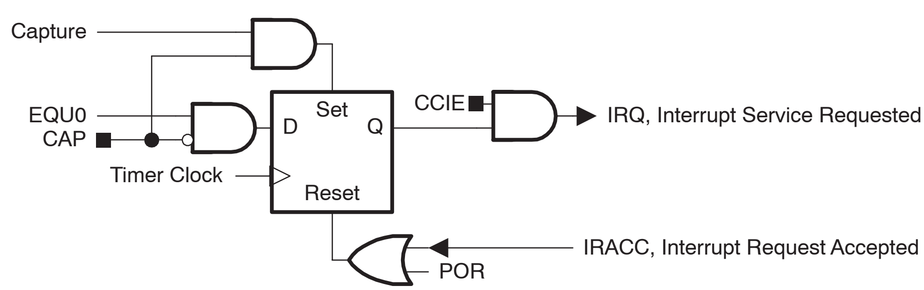
- ❑ TACCR0 interrupt vector for TACCR0 CCIFG
- ❑ TAIV interrupt vector for all other CCIFG flags and TAIFG

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 8−15. The TACCR0 CCIFG flag is automatically reset when the TACCR0 interrupt request is serviced.

*Figure 8−15. 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
```

## 8.3 Timer_A Registers

The Timer_A registers are listed in Table 8−3:

† Not present on MSP430x20xx Devices

*Table 8−3. 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 |

† Not present on MSP430x20xx Devices

## TACTL, Timer_A Control Register

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Unused | | | | | | TASSELx | |
| rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IDx | | MCx | | Unused | TACLR | TAIE | TAIFG |
| rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | w–(0) | rw–(0) | rw–(0) |

| | | |
|---|---|---|
| **Unused** | Bits 15-10 | Unused |
| **TASSELx** | Bits 9-8 | Timer_A clock source select<br>00 TACLK<br>01 ACLK<br>10 SMCLK<br>11 INCLK |
| **IDx** | Bits 7-6 | Input divider. These bits select the divider for the input clock.<br>00 /1<br>01 /2<br>10 /4<br>11 /8 |
| **MCx** | Bits 5-4 | Mode control. Setting MCx = 00h when Timer_A is not in use conserves power.<br>00 Stop mode: the timer is halted<br>01 Up mode: the timer counts up to TACCR0<br>10 Continuous mode: the timer counts up to 0FFFFh<br>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, the clock divider, and the 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.<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **TAIFG** | Bit 0 | Timer_A interrupt flag<br>0 No interrupt pending<br>1 Interrupt pending |

## TAR, Timer_A Register

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| TARx | | | | | | | |
| rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| TARx | | | | | | | |
| rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) |

| | | |
|---|---|---|
| **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 | r0 | 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

Synchronized ：同步的

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

† Not Implemented in MSP430x20xx, devices

Chapter 9

# 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 MSP430x22x4 devices.

| Topic | Page |
|---|---|

## 9.1 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 9−1.

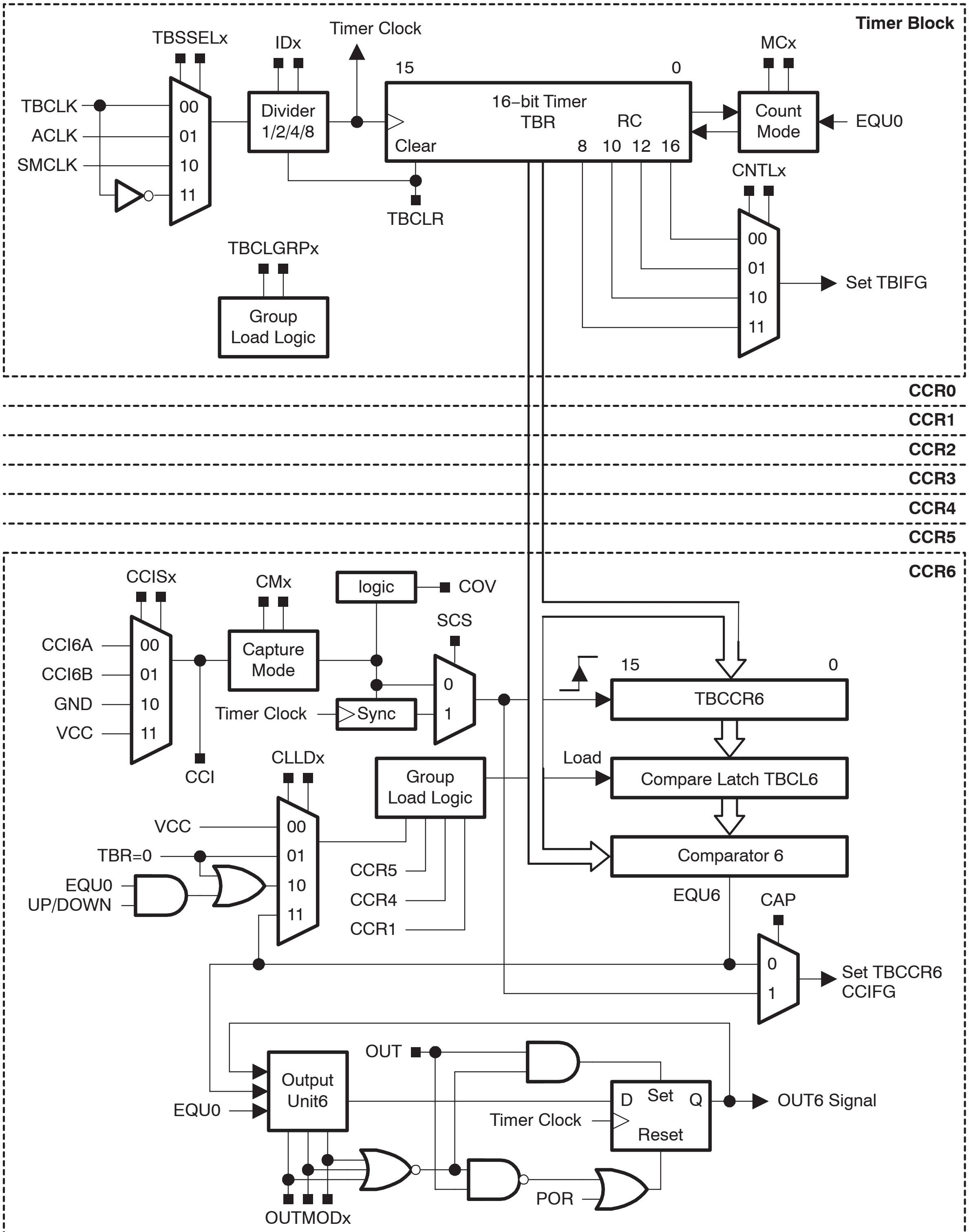
> **Note: Use of the Word *Count***
>
> *Count* is used throughout this chapter. It means the counter must be in the process of counting for the action to take place. If a particular value is directly written to the counter, then an associated action does not take place.

### 9.1.1 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 9−1. Timer_B Block Diagram*



## 9.2 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.

### 9.2.1 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, interrupt flag, and TBCLR) to avoid errant operating conditions.

When the timer clock is asynchronous to the CPU clock, any read from TBR should occur while the timer is not operating or the results may be unpredictable. Alternatively, the timer may be read multiple times while operating, and a majority vote taken in software to determine the correct reading. 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 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. The clock divider is reset when TBCLR is set.

### 9.2.2 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.

### 9.2.3 Timer Mode Control

The timer has four modes of operation as described in Table 9−1: stop, up, continuous, and up/down. The operating mode is selected with the MCx bits.

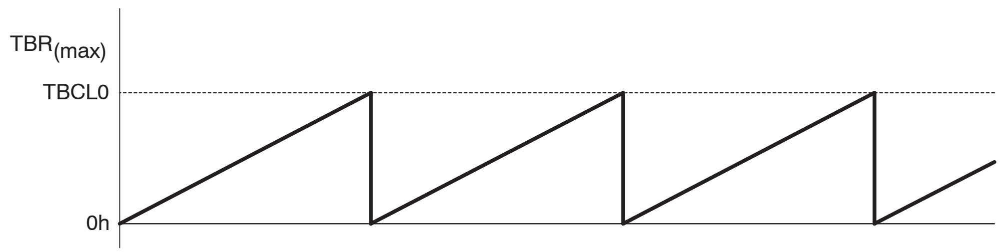
*Table 9−1. Timer Modes*

| 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

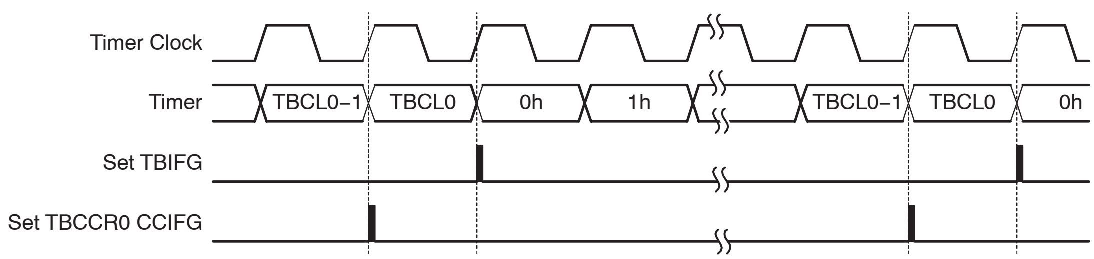
The up mode is used if the timer period must be different from $TBR_{(max)}$ counts. The timer repeatedly counts up to the value of compare latch TBCL0, which defines the period, as shown in Figure 9−2. The number of timer counts in the period is TBCL0+1. When the timer value equals TBCL0 the timer restarts counting from zero. If up mode is selected when the timer value is greater than TBCL0, the timer immediately restarts counting from zero.

*Figure 9−2. Up Mode*




The TBCCR0 CCIFG interrupt flag is set when the timer *counts* to the TBCL0 value. The TBIFG interrupt flag is set when the timer *counts* from TBCL0 to zero. Figure 8−3 shows the flag set cycle.

*Figure 9−3. Up Mode Flag Setting*



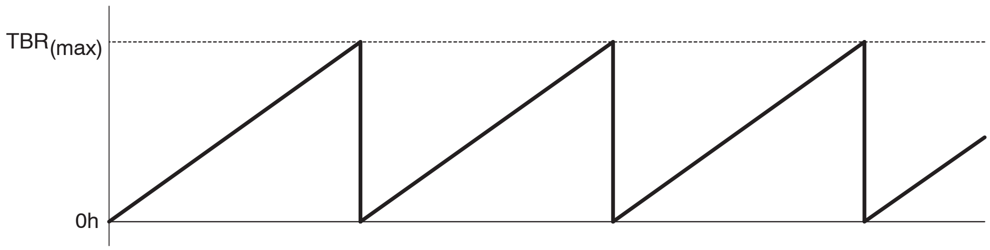

### *Changing the Period Register TBCL0*

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

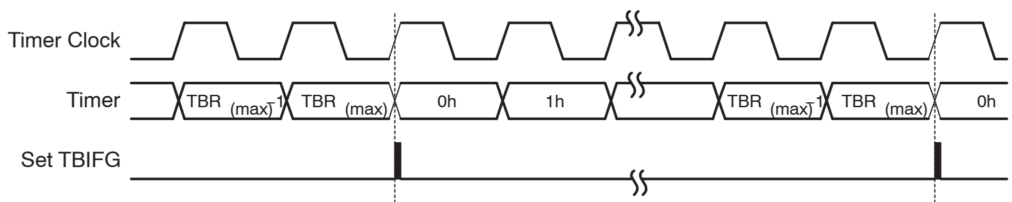
In continuous mode the timer repeatedly counts up to $TBR_{(max)}$ and restarts from zero as shown in Figure 9−4. The compare latch TBCL0 works the same way as the other capture/compare registers.

*Figure 9−4. Continuous Mode*




The TBIFG interrupt flag is set when the timer *counts* from $TBR_{(max)}$ to zero. Figure 9−5 shows the flag set cycle.

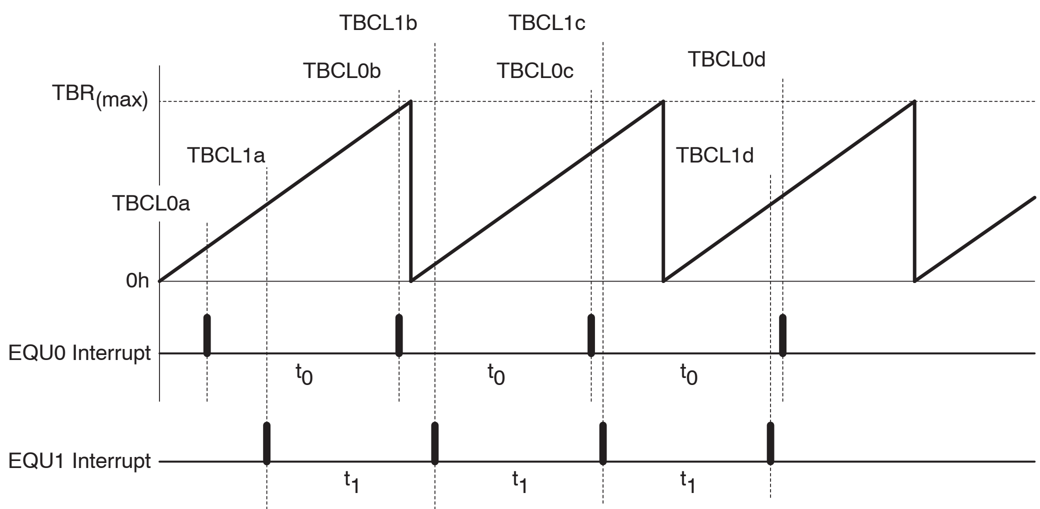
*Figure 9−5. Continuous Mode Flag Setting*



### 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 9−6 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 9−6. 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.

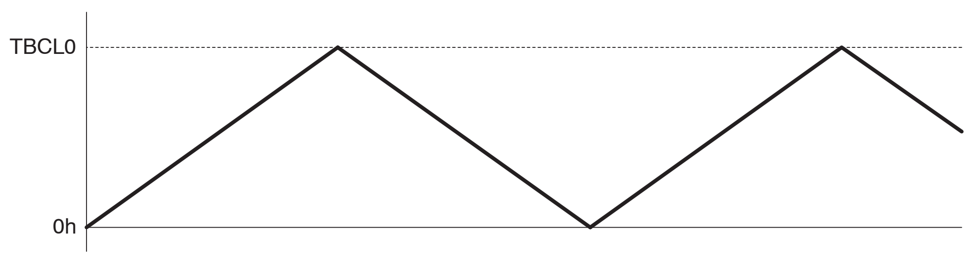
## Up/Down Mode

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 9−7. 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.

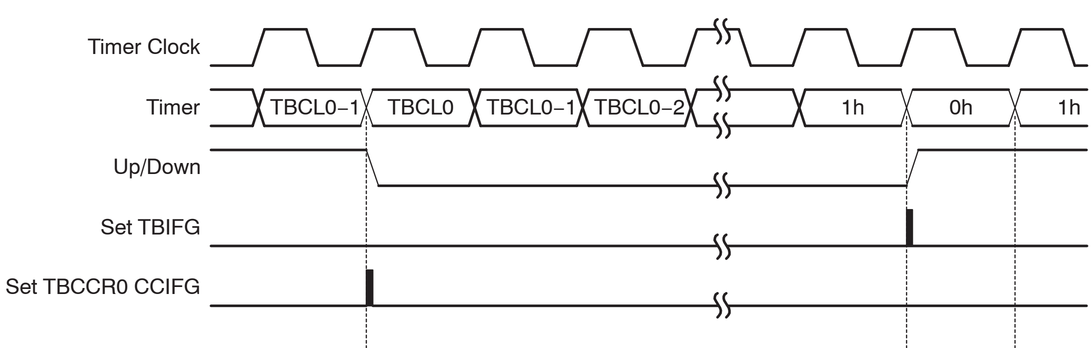
*Figure 9−7. Up/Down Mode*




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 clock 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 9−8 shows the flag set cycle.

*Figure 9−8. 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 9−9 the $t_{dead}$ is:

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

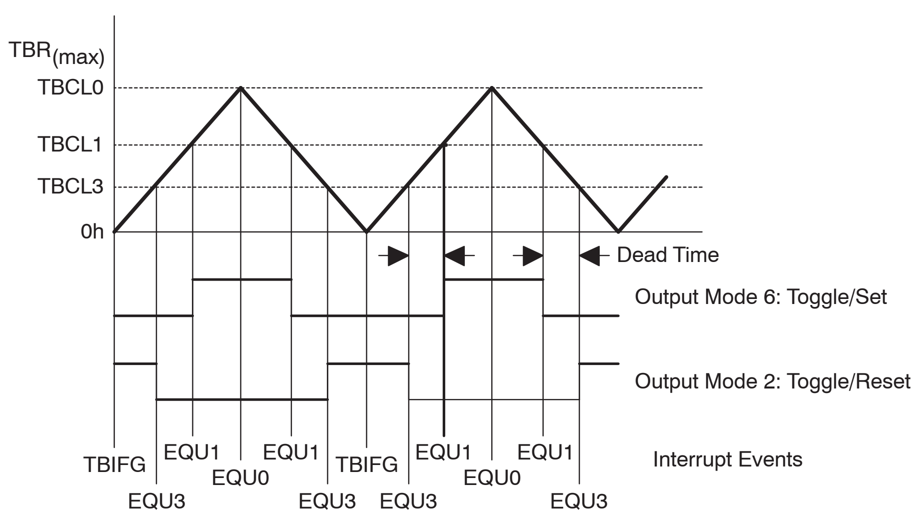
With: $t_{dead}$ Time during which both outputs need to be inactive

$t_{timer}$ 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 9−9. Output Unit in Up/Down Mode*



### 9.2.4 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

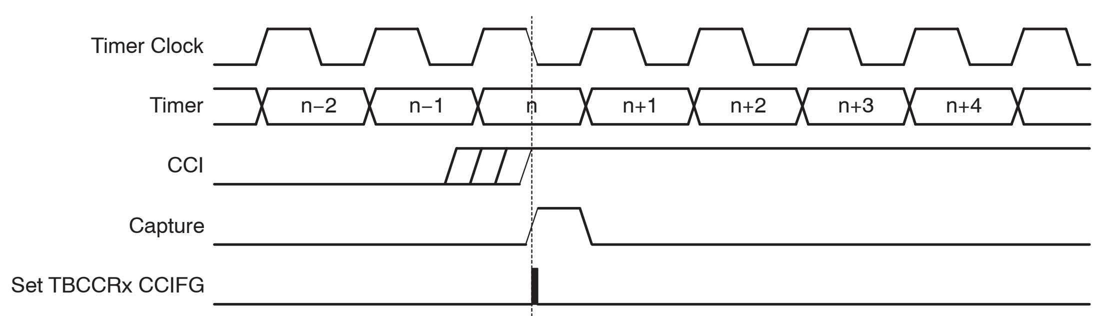
The capture mode is selected when CAP = 1. Capture mode is used to record time events. It can be used for speed computations or time measurements. The capture inputs CCIxA and CCIxB are connected to external pins or internal signals and are selected with the CCISx bits. The CMx bits select the capture 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. MSP430x2xx family devices may have different signals connected to CCIxA and CCIxB. Refer to the device-specific datasheet for the connections of these signals.

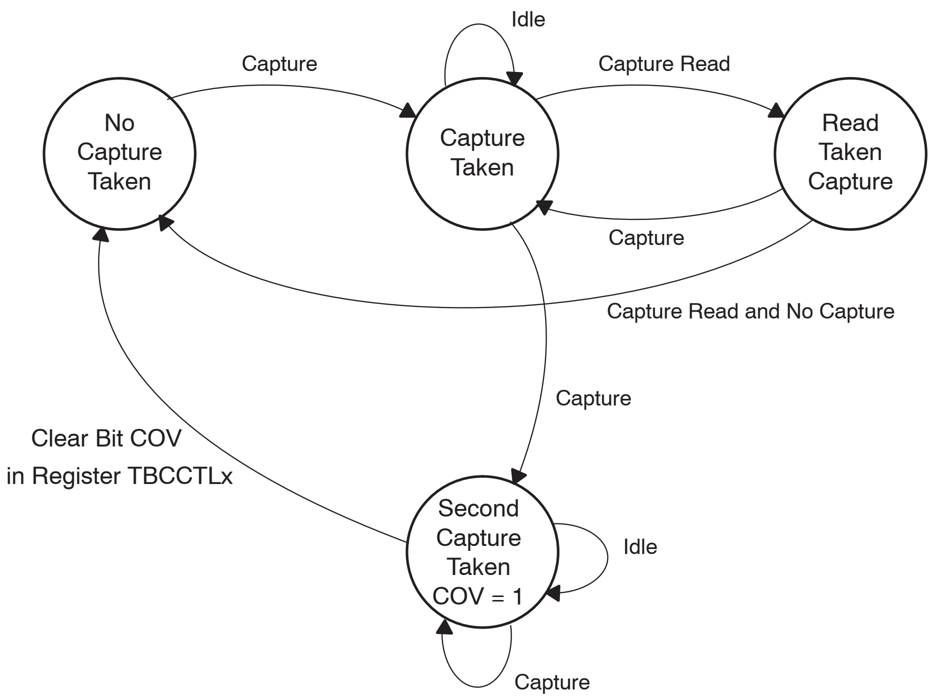
The capture signal can be asynchronous to the timer clock and cause a race condition. Setting the SCS bit will synchronize the capture with the next timer clock. Setting the SCS bit to synchronize the capture signal with the timer clock is recommended. This is illustrated in Figure 9−10.

*Figure 9−10. 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 9−11. COV must be reset with software.

*Figure 9−11. 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 9−2.

*Table 9−2. 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 9−3. 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 9−3. Compare Latch Operating Modes*

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

### 9.2.5 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 9−4. 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.

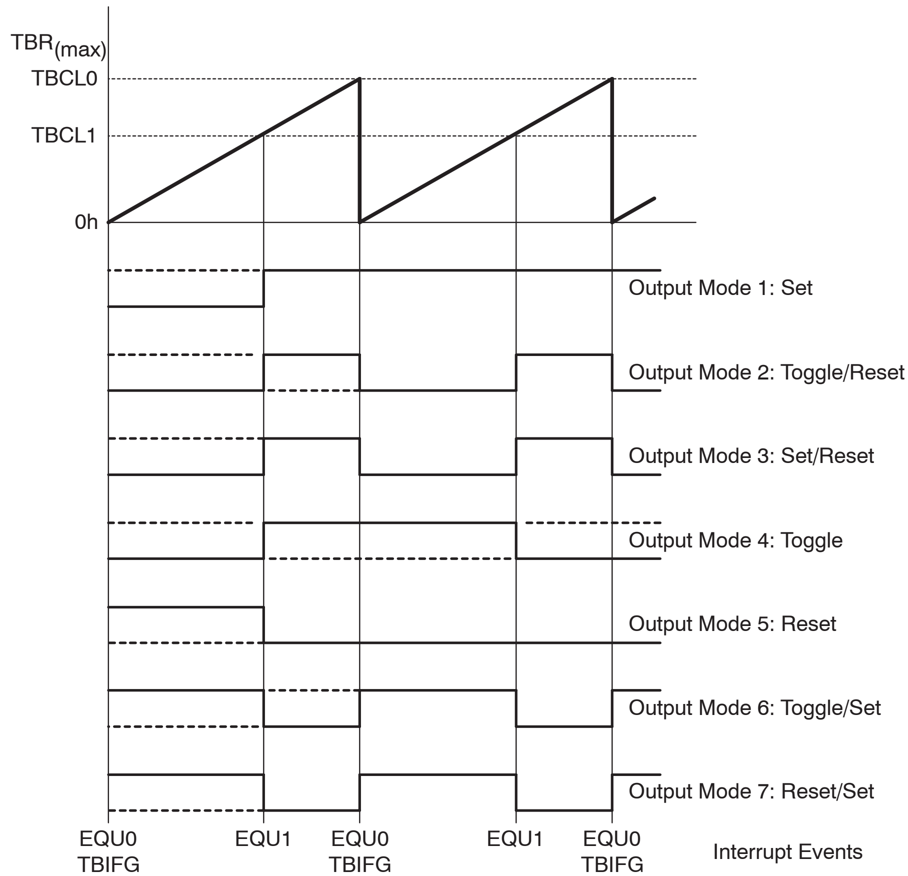
*Table 9−4. Output Modes*

| 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 9−12 using TBCL0 and TBCL1.

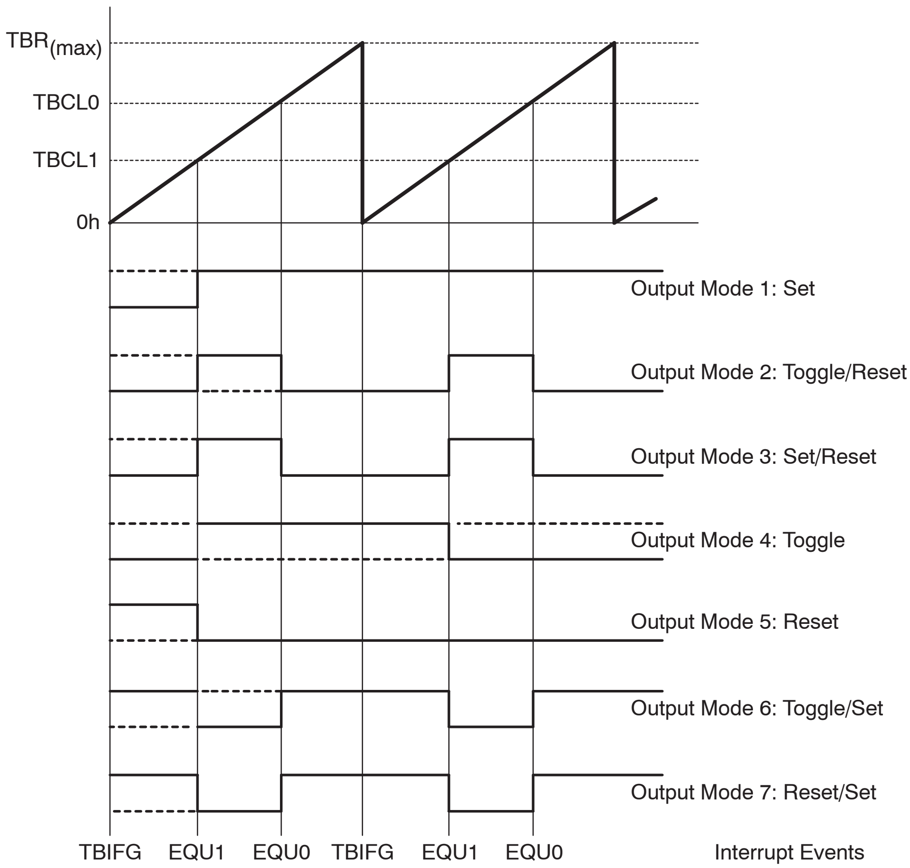
*Figure 9−12. Output Example—Timer in Up Mode*



### *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 9−13 using TBCL0 and TBCL1.

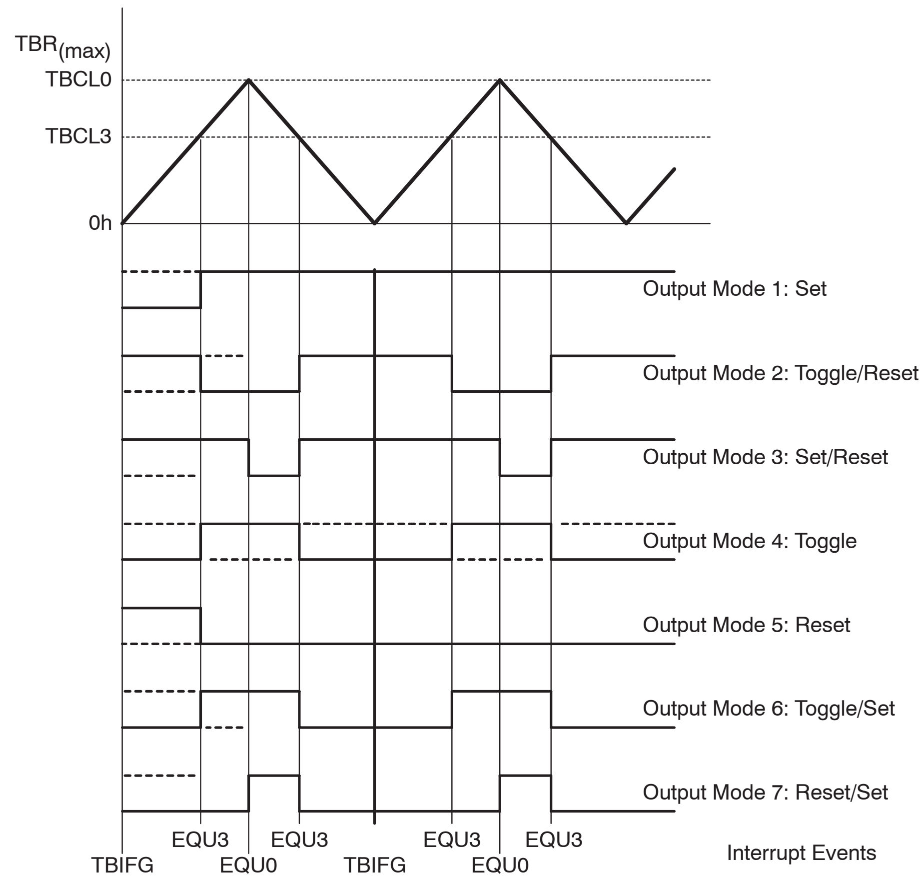
*Figure 9−13. Output Example—Timer in Continuous Mode*



### 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 9−14 using TBCL0 and TBCL3.

*Figure 9−14. 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
```

### 9.2.6 Timer_B Interrupts

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

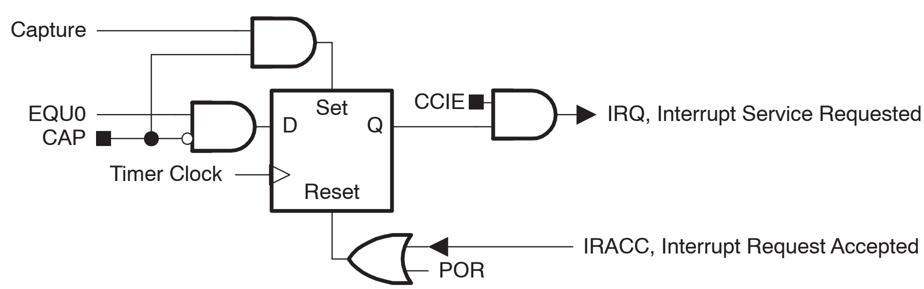
- TBCCR0 interrupt vector for TBCCR0 CCIFG
- TBIV interrupt vector for all other CCIFG flags and TBIFG

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 9−15. The TBCCR0 CCIFG flag is automatically reset when the TBCCR0 interrupt request is serviced.

*Figure 9−15. 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
```

## 9.3 Timer_B Registers

The Timer_B registers are listed in Table 9−5:

*Table 9−5. 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 | TBCCTL2 | 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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Unused | TBCLGRPx | | CNTLx | | Unused | TBSSELx | |
| rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| IDx | | MCx | | Unused | TBCLR | TBIE | TBIFG |
| rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | w–(0) | rw–(0) | rw–(0) |

**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 Inverted TBCLK

**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, the clock divider, and the 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.<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **TBIFG** | Bit 0 | Timer_B interrupt flag.<br>0 No interrupt pending<br>1 Interrupt pending |

### TBR, Timer_B Register

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| **TBRx** | | | | | | | |
| rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **TBRx** | | | | | | | |
| rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) |

| | | |
|---|---|---|
| **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<br>00 No capture<br>01 Capture on rising edge<br>10 Capture on falling edge<br>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.<br>00 CCIxA<br>01 CCIxB<br>10 GND<br>11 $V_{CC}$ |
| **SCS** | Bit 11 | Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.<br>0 Asynchronous capture<br>1 Synchronous capture |
| **CLLDx** | Bit 10-9 | Compare latch load. These bits select the compare latch load event.<br>00 TBCLx loads on write to TBCCRx<br>01 TBCLx loads when TBR *counts* to 0<br>10 TBCLx loads when TBR *counts* to 0 (up or continuous mode)<br>TBCLx loads when TBR *counts* to TBCL0 or to 0 (up/down mode)<br>11 TBCLx loads when TBR *counts* to TBCLx |
| **CAP** | Bit 8 | Capture mode<br>0 Compare mode<br>1 Capture mode |
| **OUTMODx** | Bits 7-5 | Output mode. Modes 2, 3, 6, and 7 are not useful for TBCL0 because EQUx = EQU0.<br>000 OUT bit value<br>001 Set<br>010 Toggle/reset<br>011 Set/reset<br>100 Toggle<br>101 Reset<br>110 Toggle/set<br>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. 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 |

† Not available on all devices

## Chapter 10

# Universal Serial Interface

The Universal Serial Interface (USI) module provides SPI and $I^2C$ serial communication with one hardware module. This chapter discusses both modes. The USI module is implemented in the MSP430x20xx devices.

**Topic** **Page**

## 10.1 USI Introduction

The USI module provides the basic functionality to support synchronous serial communication. In its simplest form, it is an 8- or 16-bit shift register that can be used to output data streams, or when combined with minimal software, can implement serial communication. In addition, the USI includes built-in hardware functionality to ease the implementation of SPI and $I^2C$ communication. The USI module also includes interrupts to further reduce the necessary software overhead for serial communication and to maintain the ultralow-power capabilities of the MSP430.

The USI module features include:

- Three-wire SPI mode support
- $I^2C$ mode support
- Variable data length
- Slave operation in LPM4 − no internal clock required
- Selectable MSB or LSB data order
- START and STOP detection for $I^2C$ mode with automatic SCL control
- Arbitration lost detection in master mode
- Programmable clock generation
- Selectable clock polarity and phase control

Figure 10−1 shows the USI module in SPI mode. Figure 10−2 shows the USI module in I2C mode.

*Figure 10−1. USI Block Diagram: SPI Mode*

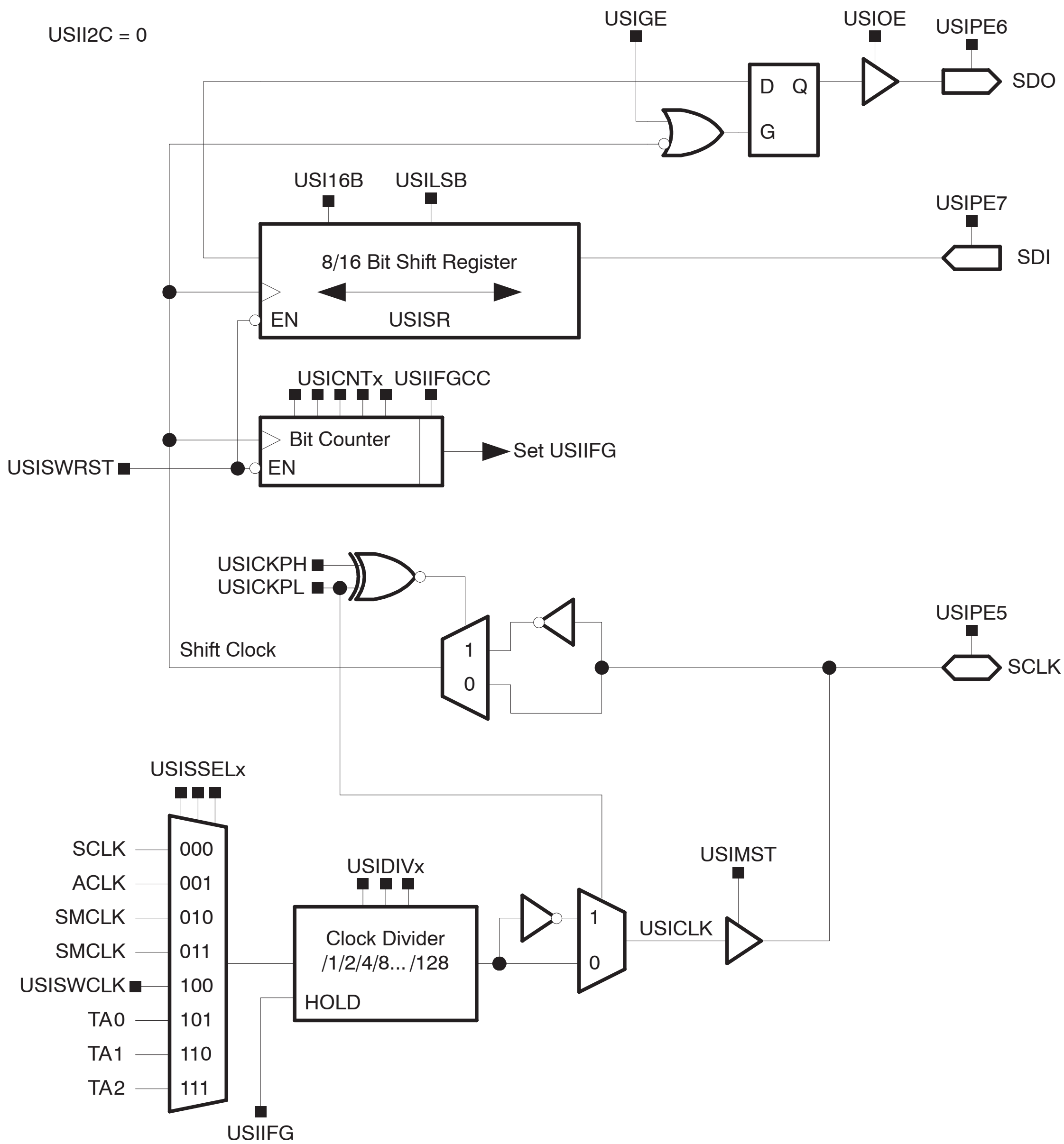
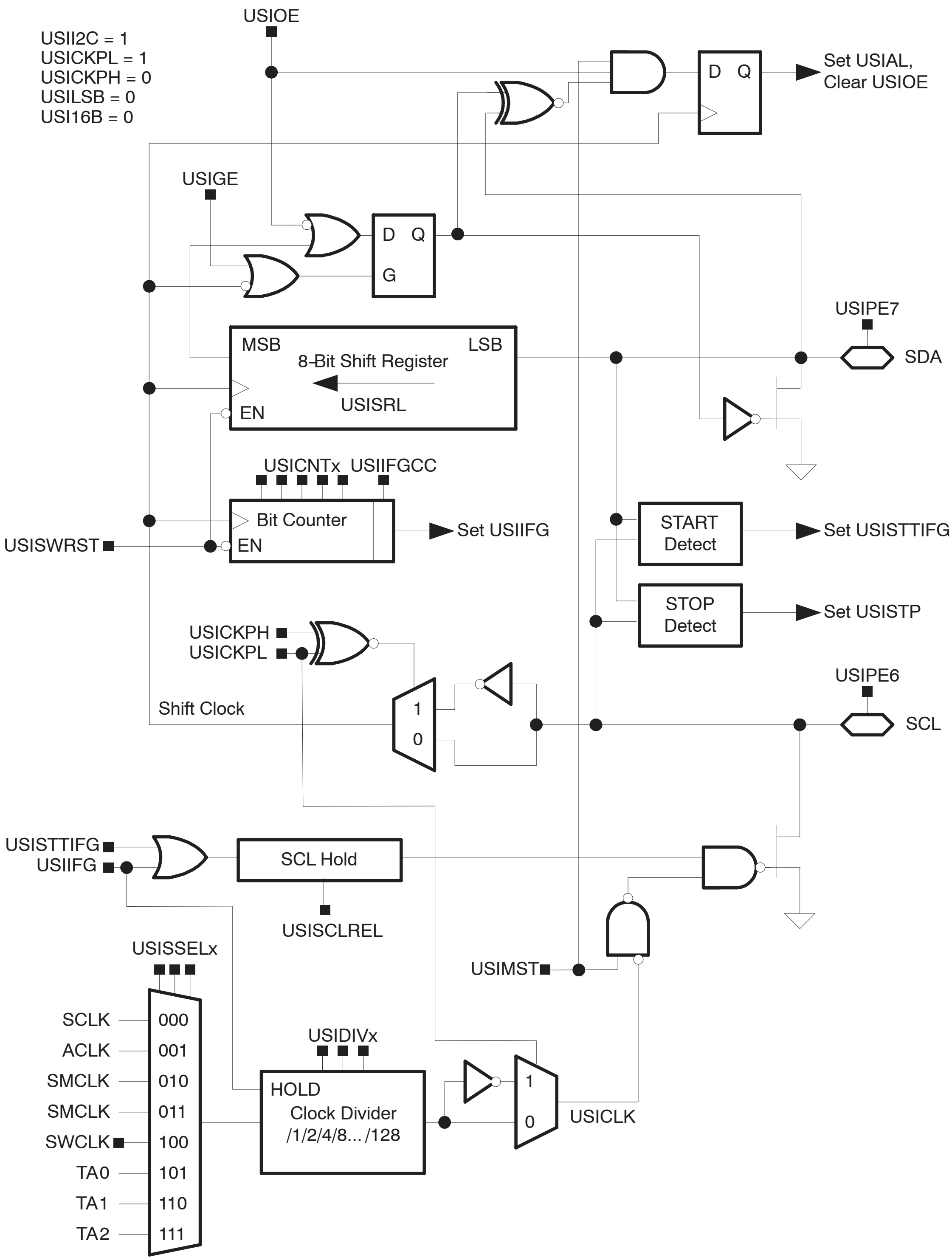


*Figure 10−2. USI Block Diagram: I2C Mode*



## 10.2 USI Operation

The USI module is a shift register and bit counter that includes logic to support SPI and $I^2C$ communication. The USI shift register, USISR, is directly accessible by software and contains the data to be transmitted or the data that has been received.

The bit counter counts the number of sampled bits and sets the USI interrupt flag USIIFG when the USICNTx value becomes zero - either by decrementing or by directly writing zero to the USICNTx bits. Writing USICNTx with a value > 0 automatically clears USIIFG when USIIFGCC = 0, otherwise USIIFG is not affected. The USICNTx bits stop decrementing when they become 0. They will not underflow to 0FFh.

Both the counter and the shift register are driven by the same shift clock. On a rising shift clock edge, USICNTx decrements and USISR samples the next bit input. The latch connected to the shift register's output delays the change of the output to the falling edge of shift clock. It can be made transparent by setting the USIGE bit. This setting will immediately output the MSB or LSB of USISR to the SDO pin, depending on the USILSB bit.

### 10.2.1 USI Initialization

While the USI software reset bit, USISWRST, is set, the flags USIIFG, USISTTIFG, USISTP, and USIAL will be held in their reset state. USISR and USICNTx are not clocked and their contents are not affected. In $I^2C$ mode, the SCL line is also released to the idle state by the USI hardware.

To activate USI port functionality the corresponding USIPEx bits in the USI control register must be set. This will select the USI function for the pin and maintains the PxIN and PxIFG functions for the pin as well. With this feature, the port input levels can be read via the PxIN register by software and the incoming data stream can generate port interrupts on data transitions. This is useful, for example, to generate a port interrupt on a START edge.

## 10.2.2 USI Clock Generation

The USI clock generator contains a clock selection multiplexer, a divider, and the ability to select the clock polarity as shown in the block diagrams Figure 11−1 and Figure 10−2.

The clock source can be selected from the internal clocks ACLK or SMCLK, from an external clock SCLK, as well as from the capture/compare outputs of Timer_A. In addition, it is possible to clock the module by software using the USISWCLK bit when USISSELx = 100.

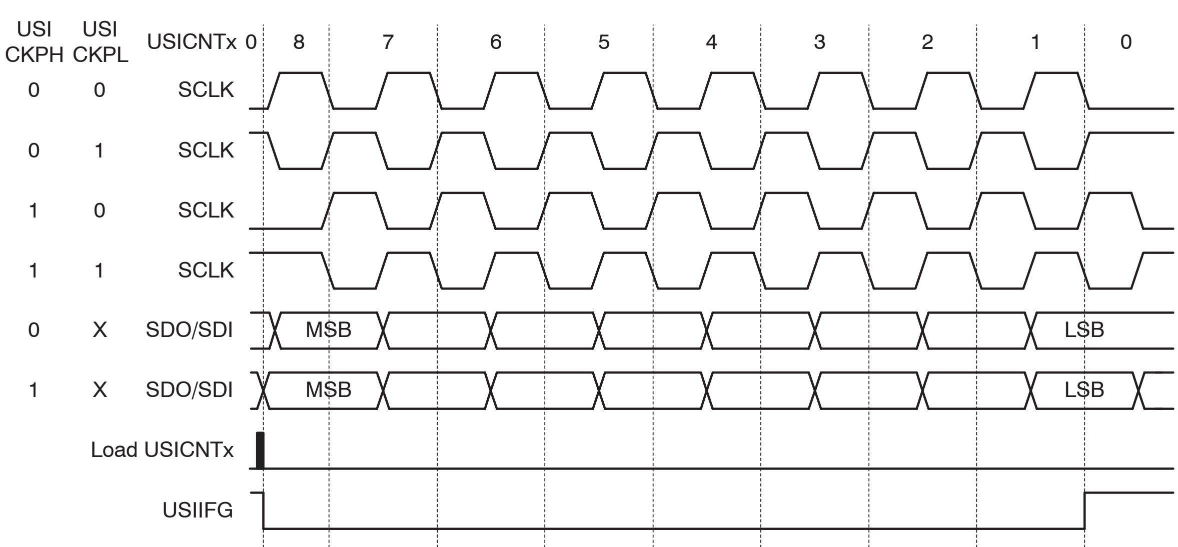
The USIDIVx bits can be used to divide the selected clock by a power of 2 up to 128. The generated clock, USICLK, is stopped when USIIFG = 1 or when the module operates in slave mode.

The USICKPL bit is used to select the polarity of USICLK. When USICKPL = 0, the inactive level of USICLK is low. When USICKPL = 1 the inactive level of USICLK is high.

## 10.2.3 SPI Mode

The USI module is configured in SPI mode when USII2C = 0. Control bit USICKPL selects the inactive level of the SPI clock while USICKPH selects the clock edge on which SDO is updated and SDI is sampled. Figure 10−3 shows the clock/data relationship for an 8-bit, MSB-first transfer. USIPE5, USIPE6, and USIPE7 must be set to enable the SCLK, SDO, and SDI port functions.

*Figure 10−3. SPI Timing*



### SPI Master Mode

The USI module is configured as SPI master by setting the master bit USIMST and clearing the I2C bit USII2C. Since the master provides the clock to the slave(s) an appropriate clock source needs to be selected and SCLK configured as output. When USIPE5 = 1, SCLK is automatically configured as an output.

When USIIFG = 0 and USICNTx > 0, clock generation is enabled and the master will begin clocking in/out data using USISR.

Received data must be read from the shift register before new data is written into it for transmission. In a typical application, the USI software will read received data from USISR, write new data to be transmitted to USISR, and enable the module for the next transfer by writing the number of bits to be transferred to USICNTx.

### SPI Slave Mode

The USI module is configured as SPI slave by clearing the USIMST and the USII2C bits. In this mode, when USIPE5 = 1 SCLK is automatically configured as an input and the USI receives the clock externally from the master.

If the USI is to transmit data, the shift register must be loaded with the data before the master provides the first clock edge. The output must be enabled by setting USIOE. When USICKPH = 1, the MSB will be visible on SDO immediately after loading the shift register.

The SDO pin can be disabled by clearing the USIOE bit. This is useful if the slave is not addressed in an environment with multiple slaves on the bus.

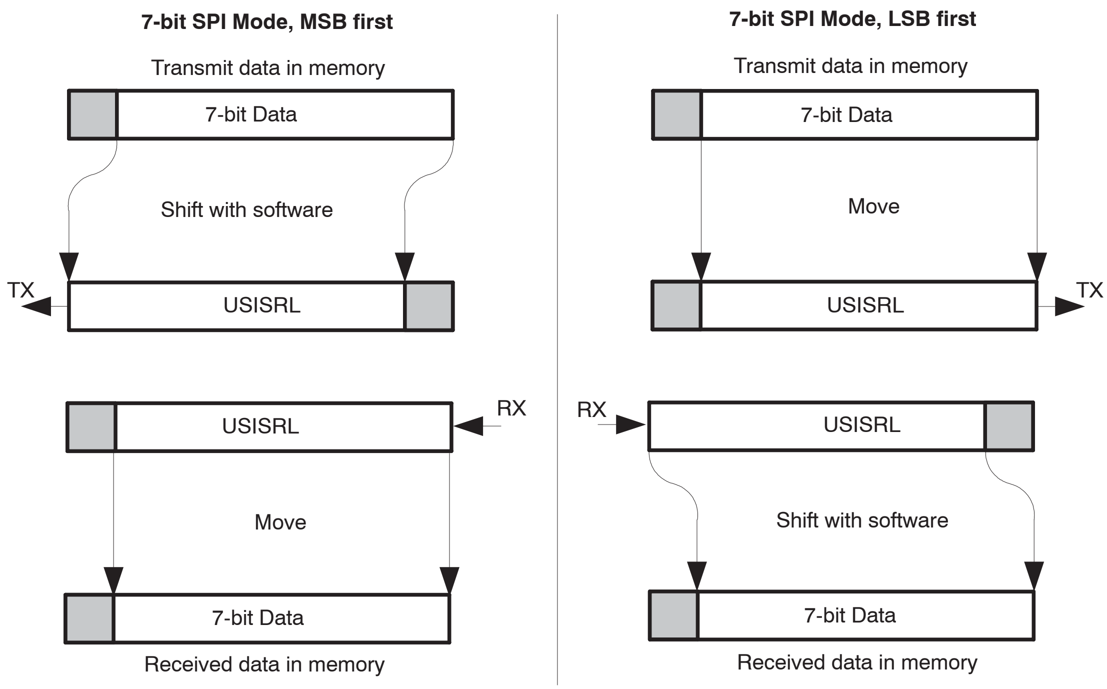
Once all bits are received, the data must be read from USISR and new data loaded into USISR before the next clock edge from the master. In a typical application, after receiving data, the USI software will read the USISR register, write new data to USISR to be transmitted, and enable the USI module for the next transfer by writing the number of bits to be transferred to USICNTx.

## *USISR Operation*

The 16-bit USISR is made up of two 8-bit registers, USISRL and USISRH. Control bit USI16B selects the number of bits of USISR that are used for data transmit and receive. When USI16B = 0, only the lower 8 bits, USISRL, are used.

To transfer < 8 bits, the data must be loaded into USISRL such that unused bits are not shifted out. The data must be MSB- or LSB-aligned depending on USILSB. Figure 10−4 shows an example of 7-bit data handling.

*Figure 10−4. Data adjustments for 7-bit SPI Data*




When USI16B = 1, all 16 bits are used for data handling. When using USISR to access both USISRL and USISRH, the data needs to be properly adjusted when < 16 bits are used in the same manner as shown in Figure 10−4.

## *SPI Interrupts*

There is one interrupt vector associated with the USI module, and one interrupt flag, USIIFG, relevant for SPI operation. When USIIE and the GIE bit are set, the interrupt flag will generate an interrupt request.

USIIFG is set when USICNTx becomes zero, either by counting or by directly writing 0 to the USICNTx bits. USIIFG is cleared by writing a value > 0 to the USICNTx bits when USIIFGCC = 0, or directly by software.

### 10.2.4 I²C Mode

The USI module is configured in I²C mode when USII2C =1, USICKPL = 1, and USICKPH = 0. For I²C data compatibility, USILSB and USI16B must be cleared. USIPE6 and USIPE7 must be set to enable the SCL and SDA port functions.

#### I²C *Master Mode*

To configure the USI module as an I²C master the USIMST bit must be set. In master mode, clocks are generated by the USI module and output to the SCL line while USIIFG = 0. When USIIFG = 1, the SCL will stop at the idle, or high, level. Multi-master operation is supported as described in the Arbitration section.

The master supports slaves that are holding the SCL line low only when USIDIVx > 0. When USIDIVx is set to /1 clock division (USIDIVx = 0), connected slaves must not hold the SCL line low during data transmission. Otherwise the communication may fail.

#### I²C *Slave Mode*

To configure the USI module as an I²C slave the USIMST bit must be cleared. In slave mode, SCL is held low if USIIFG = 1, USISTTIFG = 1 or if USICNTx = 0. USISTTIFG must be cleared by software after the slave is setup and ready to receive the slave address from a master.

#### I²C *Transmitter*

In transmitter mode, data is first loaded into USISRL. The output is enabled by setting USIOE and the transmission is started by writing 8 into USICNTx. This clears USIIFG and SCL is generated in master mode or released from being held low in slave mode. After the transmission of all 8 bits, USIIFG is set, and the clock signal on SCL is stopped in master mode or held low at the next low phase in slave mode.

To receive the I²C acknowledgement bit, the USIOE bit is cleared with software and USICNTx is loaded with 1. This clears USIIFG and one bit is received into USISRL. When USIIFG becomes set again, the LSB of USISRL is the received acknowledge bit and can be tested in software.

```
; Receive ACK/NACK
   BIC.B  #USIOE,&USICTL0     ; SDA input
   MOV.B  #01h,&USICNT        ; USICNTx = 1
TEST_USIIFG
   BIT.B  #USIIFG,&USICTL1    ; Test USIIFG
   JZ     TEST_USIIFG
   BIT.B  #01h,&USISRL        ; Test received ACK bit
   JNZ    HANDLE_NACK         ; Handle if NACK
...Else, handle ACK
```

### $I^2C$ *Receiver*

In $I^2C$ receiver mode the output must be disabled by clearing USIOE and the USI module is prepared for reception by writing 8 into USICNTx. This clears USIIFG and SCL is generated in master mode or released from being held low in slave mode. The USIIFG bit will be set after 8 clocks. This stops the clock signal on SCL in master mode or holds SCL low at the next low phase in slave mode.

To transmit an acknowledge or no-acknowledge bit, the MSB of the shift register is loaded with 0 or 1, the USIOE bit is set with software to enable the output, and 1 is written to the USICNTx bits. As soon as the MSB bit is shifted out, USIIFG will be become set and the module can be prepared for the reception of the next $I^2C$ data byte.

```
; Generate ACK
   BIS.B  #USIOE,&USICTL0     ; SDA output
   MOV.B  #00h,&USISRL        ; MSB = 0
   MOV.B  #01h,&USICNT        ; USICNTx = 1
TEST_USIIFG
   BIT.B  #USIIFG,&USICTL1    ; Test USIIFG
   JZ     TEST_USIIFG
...continue...

; Generate NACK
   BIS.B  #USIOE,&USICTL0     ; SDA output
   MOV.B #0FFh,&USISRL        ; MSB = 1
   MOV.B  #01h,&USICNT        ; USICNTx = 1
TEST_USIIFG
   BIT.B  #USIIFG,&USICTL1    ; Test USIIFG
   JZ     TEST_USIIFG
...continue...
```

### *START Condition*

A START condition is a high-to-low transition on SDA while SCL is high. The START condition can be generated by setting the MSB of the shift register to 0. Setting the USIGE and USIOE bits makes the output latch transparent and the MSB of the shift register is immediately presented to SDA and pulls the line low. Clearing USIGE resumes the clocked-latch function and holds the 0 on SDA until data is shifted out with SCL.

```
; Generate START
   MOV.B  #000h,&USISRL          ; MSB = 0
   BIS.B  #USIGE+USIOE,&USICTL0  ; Latch/SDA output enabled
   BIC.B  #USIGE,&USICTL0        ; Latch disabled
...continue...
```

### *STOP Condition*

A STOP condition is a low-to-high transition on SDA while SCL is high. To finish the acknowledgment bit and pull SDA low to prepare the STOP condition generation requires clearing the MSB in the shift register and loading 1 into USICNTx. This will generate a low pulse on SCL and during the low phase SDA is pulled low. SCL stops in the idle, or high, state since the module is in master mode. To generate the low-to-high transition, the MSB is set in the shift register and USICNTx is loaded with 1. Setting the USIGE and USIOE bits makes the output latch transparent and the MSB of USISRL releases SDA to the idle state. Clearing USIGE stores the MSB in the output latch and the output is disabled by clearing USIOE. SDA remains high until a START condition is generated because of the external pull-up..

```
; Generate STOP
   BIS.B  #USIOE,&USICTL0  ; SDA=output
   MOV.B  #000H,&USISRL    ; MSB = 0
   MOV.B  #001H,&USICNT    ; USICNT = 1 for one clock
TEST_USIIFG
   BIT.B  #USIIFG,&USICTL1 ; Test USIIFG
   JZ  TEST_USIIFG         ;
   MOV.B  #0FFH,&USISRL    ; USISRL = 1 to drive SDA high
   BIS.B  #USIGE,&USICTL0  ; Transparent latch enabled
   BIC.B  #USIGE+USIOE,&USICTL; Latch/SDA output disabled
...continue...
```

### *Releasing SCL*

Setting the USISCLREL bit will release SCL if it is being held low by the USI module without requiring USIIFG to be cleared. The USISCLREL bit will be cleared automatically if a START condition is received and the SCL line will be held low on the next clock.

In slave operation this bit should be used to prevent SCL from being held low when the slave has detected that it was not addressed by the master. On the next START condition USISCLREL will be cleared and the USISTTIFG will be set.

### *Arbitration*

The USI module can detect a lost arbitration condition in multi-master $I^2C$ systems. The $I^2C$ arbitration procedure uses the data presented on SDA by the competing transmitters. The first master transmitter that generates a logic high loses arbitration to the opposing master generating a logic low. The loss of arbitration is detected in the USI module by comparing the value presented to the bus and the value read from the bus. If the values are not equal arbitration is lost and the arbitration lost flag, USIAL, is set. This also clears the output enable bit USIOE and the USI module no longer drives the bus. In this case, user software must check the USIAL flag together with USIIFG and configure the USI to slave receiver when arbitration is lost. The USIAL flag must be cleared by software.

To prevent other faster masters from generating clocks during the arbitration procedure SCL is held low if another master on the bus drives SCL low and USIIFG or USISTTIFG is set, or if USICNTx = 0.

### $I^2C$ *Interrupts*

There is one interrupt vector associated with the USI module with two interrupt flags relevant for I2C operation, USIIFG and USISTTIFG. Each interrupt flag has its own interrupt enable bit, USIIE and USISTTIE. When an interrupt is enabled, and the GIE bit is set, a set interrupt flag will generate an interrupt request.

USIIFG is set when USICNTx becomes zero, either by counting or by directly writing 0 to the USICNTx bits. USIIFG is cleared by writing a value > 0 to the USICNTx bits when USIIFGCC = 0, or directly by software.

USISTTIFG is set when a START condition is detected. The USISTTIFG flag must be cleared by software.

The reception of a STOP condition is indicated with the USISTP flag but there is no interrupt function associated with the USISTP flag. USISTP is cleared by writing a value > 0 to the USICNTx bits when USIIFGCC = 0 or directly by software.

## 10.3 USI Registers

The USI registers are listed in Table 10−1:

*Table 10−1. USI Registers*

| Register | Short Form | Register Type | Address | Initial State |
|---|---|---|---|---|
| USI control register 0 | USICTL0 | Read/write | 078h | 01h with PUC |
| USI control register 1 | USICTL1 | Read/write | 079h | 01h with PUC |
| USI clock control | USICKCTL | Read/write | 07Ah | Reset with PUC |
| USI bit counter | USICNT | Read/write | 07Bh | Reset with PUC |
| USI low byte shift register | USISRL | Read/write | 07Ch | Unchanged |
| USI high byte shift register | USISRH | Read/write | 07Dh | Unchanged |

The USI registers can be accessed with word instructions as shown in Table 10−2:

*Table 10−2. Word Access to USI Registers*

| Register | Short Form | High−Byte Register | Low−Byte Register | Address |
|---|---|---|---|---|
| USI control register | USICTL | USICTL1 | USICTL0 | 078h |
| USI clock and counter control register | USICCTL | USICNT | USICKCTL | 07Ah |
| USI shift register | USISR | USISRH | USISRL | 07Ch |

## USICTL0, USI Control Register 0

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **USIPE7** | **USIPE6** | **USIPE5** | **USILSB** | **USIMST** | **USIGE** | **USIOE** | **USISWRST** |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–1 |

| | | |
|---|---|---|
| **USIPE7** | Bit 7 | USI SDI/SDA port enable<br>Input in SPI mode, input or open drain output in I2C mode.<br>0 USI function disabled<br>1 USI function enabled |
| **USIPE6** | Bit 6 | USI SDO/SCL port enable<br>Output in SPI mode, input or open drain output in I2C mode.<br>0 USI function disabled<br>1 USI function enabled |
| **USIPE5** | Bit 5 | USI SCLK port enable<br>Input in SPI slave mode, or I2C mode, output in SPI master mode.<br>0 USI function disabled<br>1 USI function enabled |
| **USILSB** | Bit 4 | LSB first select. This bit controls the direction of the receive and transmit shift register.<br>0 MSB first<br>1 LSB first |
| **USIMST** | Bit 3 | Master select<br>0 Slave mode<br>1 Master mode |
| **USIGE** | Bit 2 | Output latch control<br>0 Output latch enable depends on shift clock<br>1 Output latch always enabled and transparent |
| **USIOE** | Bit 1 | Data output enable<br>0 Output disabled<br>1 Output enabled |
| **USISWRST** | Bit 0 | USI software reset<br>0 USI released for operation.<br>1 USI logic held in reset state. |

## USICTL1, USI Control Register 1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **USICKPH** | **USII2C** | **USISTTIE** | **USIIE** | **USIAL** | **USISTP** | **USISTTIFG** | **USIIFG** |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–1 |

| | | |
|---|---|---|
| **USICKPH** | Bit 7 | Clock phase select<br>0 Data is changed on the first SCLK edge and captured on the following edge.<br>1 Data is captured on the first SCLK edge and changed on the following edge. |
| **USII2C** | Bit 6 | I2C mode enable<br>0 I2C mode disabled<br>1 I2C mode enabled |
| **USISTTIE** | Bit 5 | START condition interrupt-enable<br>0 Interrupt on START condition disabled<br>1 Interrupt on START condition enabled |
| **USIIE** | Bit 4 | USI counter interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **USIAL** | Bit 3 | Arbitration lost<br>0 No arbitration lost condition<br>1 Arbitration lost |
| **USISTP** | Bit 2 | STOP condition received. USISTP is automatically cleared if USICNTx is loaded with a value > 0 when USIIFGCC = 0.<br>0 No STOP condition received<br>1 STOP condition received |
| **USISTTIFG** | Bit 1 | START condition interrupt flag<br>0 No START condition received. No interrupt pending.<br>1 START condition received. Interrupt pending. |
| **USIIFG** | Bit 1 | USI counter interrupt flag. Set when the USICNTx = 0. Automatically cleared if USICNTx is loaded with a value > 0 when USIIFGCC = 0.<br>0 No interrupt pending<br>1 Interrupt pending |

## USICKCTL, USI Clock Control Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| USIDIVx | | | USISSELx | | | USICKPL | USISWCLK |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 |

| | | |
|---|---|---|
| **USIDIVx** | Bits 7–5 | Clock divider select<br>000 Divide by 1<br>001 Divide by 2<br>010 Divide by 4<br>011 Divide by 8<br>100 Divide by 16<br>101 Divide by 32<br>110 Divide by 64<br>111 Divide by 128 |
| **USISSELx** | Bits 4–2 | Clock source select. Not used in slave mode.<br>000 SCLK (Not used in SPI mode)<br>001 ACLK<br>010 SMCLK<br>011 SMCLK<br>100 USISWCLK bit<br>101 TACCR0<br>110 TACCR1<br>111 TACCR2 (Reserved on MSP430F20xx devices) |
| **USICKPL** | Bit 1 | Clock polarity select<br>0 Inactive state is low<br>1 Inactive state is high |
| **USISWCLK** | Bit 0 | Software clock<br>0 Input clock is low<br>1 Input clock is high |

### USICNT, USI Bit Counter Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **USISCLREL** | **USI16B** | **USIIFGCC** | **USICNTx** | | | | |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 |

| | | |
|---|---|---|
| **USISCLREL** | Bit 7 | SCL release. The SCL line is released from low to idle. USISCLREL is cleared if a START condition is detected.<br>0 SCL line is held low if USIIFG is set<br>1 SCL line is released |
| **USI16B** | Bit 6 | 16-bit shift register enable<br>0 8-bit shift register mode. Low byte register USISRL is used.<br>1 16-bit shift register mode. Both high and low byte registers USISRL and USISRH are used. USISR addresses all 16 bits simultaneously. |
| **USIIFGCC** | Bit 5 | USI interrupt flag clear control. When USIIFGCC = 1 the USIIFG will not be cleared automatically when USICNTx is written with a value > 0.<br>0 USIIFG automatically cleared on USICNTx update<br>1 USIIFG is not cleared automatically |
| **USICNTx** | Bits 4–0 | USI bit count<br>The USICNTx bits set the number of bits to be received or transmitted. |

## USISRL, USI Low Byte Shift Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| USISRLx | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

| | | |
|---|---|---|
| **USISRLx** | Bits 7−0 | Contents of the USI low byte shift register |

## USISRH, USI High Byte Shift Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| USISRHx | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

| | | |
|---|---|---|
| **USISRHx** | Bits 7−0 | Contents of the USI high byte shift register. Ignored when USI16B = 0. |

Chapter 11

# Universal Serial Communication Interface, UART Mode

The universal serial communication interface (USCI) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the asynchronous UART mode.

**Topic** **Page**

## 11.1 USCI Overview

The universal serial communication interface (USCI) modules support multiple serial communication modes. Different USCI modules support different modes. Each different USCI module is named with a different letter. For example, USCI_A is different from USCI_B, etc. If more than one identical USCI module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two USCI_A modules, they are named USCI_A0 and USCI_A1. See the device-specific datasheet to determine which USCI modules, if any, are implemented on which devices.

The USCI_Ax modules support:

- UART mode
- Pulse shaping for IrDA communications
- Automatic baud rate detection for LIN communications
- SPI mode

The USCI_Bx modules support:

- $I^2C$ mode
- SPI mode

## 11.2 USCI Introduction: UART Mode

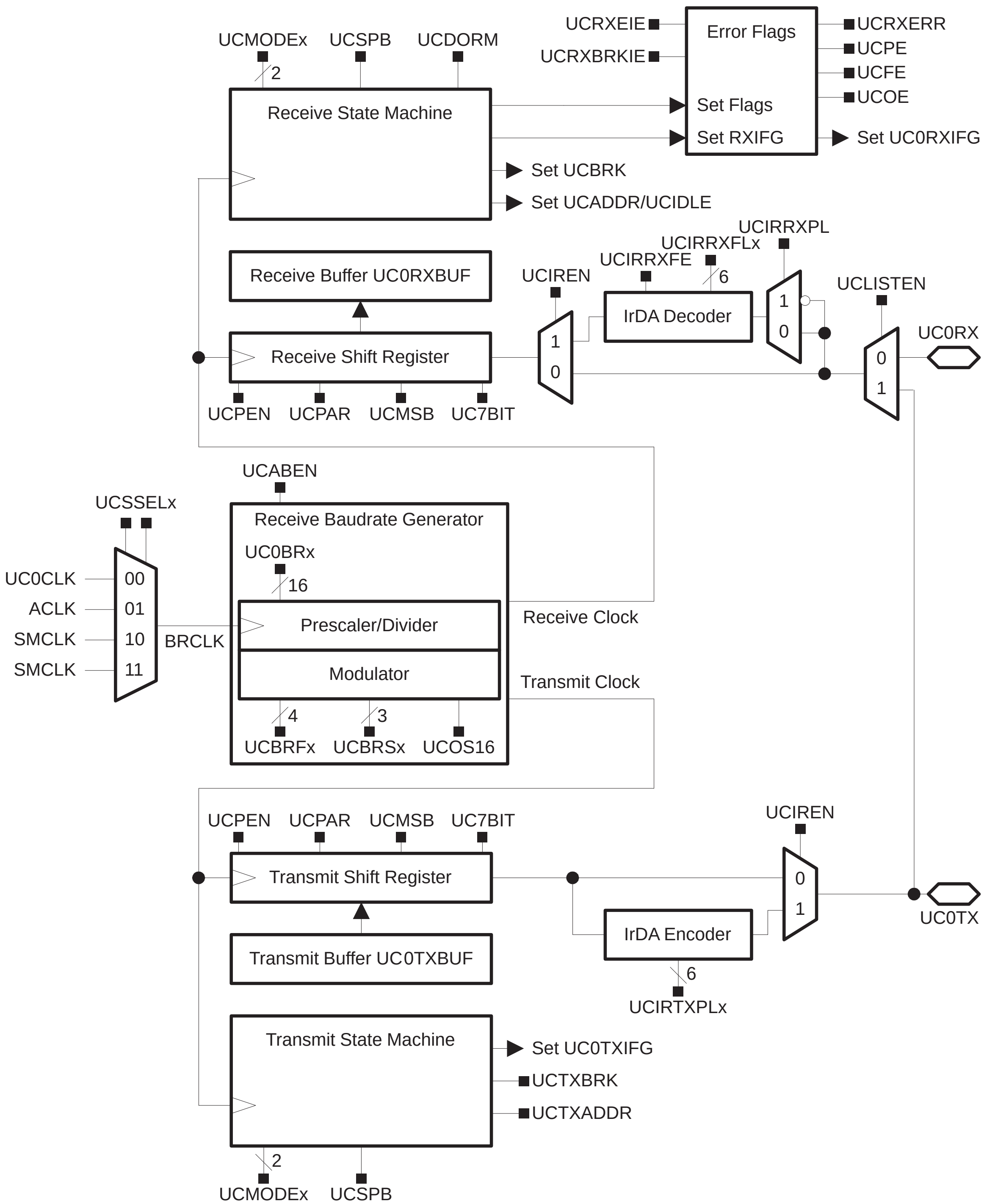
In asynchronous mode, the USCI_Ax modules connect the MSP430 to an external system via two external pins, UCAxRXD and UCAxTXD. UART mode is selected when the UCSYNC bit is cleared.

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 or MSB-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
- Status flags for address detection
- Independent interrupt capability for receive and transmit

Figure 11–1 shows the USCI_Ax when configured for UART mode.

*Figure 11–1. USCI_Ax Block Diagram: UART Mode (UCSYNC = 0)*



## 11.3 USCI Operation: UART Mode

In UART mode, the USCI 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 USCI. The transmit and receive functions use the same baud rate frequency.

### 11.3.1 USCI Initialization and Reset

The USCI is reset by a PUC or by setting the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the USCI in a reset condition. When set, the UCSWRST bit resets the UCAxRXIE, UCAxTXIE, UCAxRXIFG, UCRXERR, UCBRK, UCPE, UCOE, UCFE, UCSTOE and UCBTOE bits and sets the UCAxTXIFG bit. Clearing UCSWRST releases the USCI for operation.

**Note: Initializing or Re-Configuring the USCI Module**

The recommended USCI initialization/re-configuration process is:

1) Set UCSWRST (`BIS.B   #UCSWRST,&UCAxCTL1`)

2) Initialize all USCI registers with UCSWRST = 1 (including UCAxCTL1)

3) Configure ports.

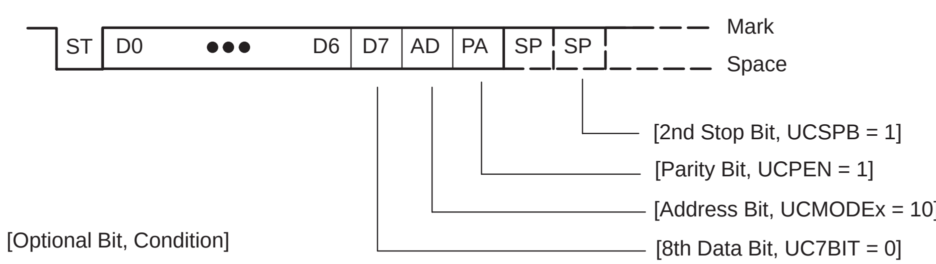
4) Clear UCSWRST via software (`BIC.B   #UCSWRST,&UCAxCTL1`)

5) Enable interrupts (optional) via UCAxRXIE and/or UCAxTXIE

### 11.3.2 Character Format

The UART character format, shown in Figure 11–2, 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 UCMSB bit controls the direction of the transfer and selects LSB or MSB first. LSB-first is typically required for UART communication.

*Figure 11–2. Character Format*



### 11.3.3 Asynchronous Communication Formats

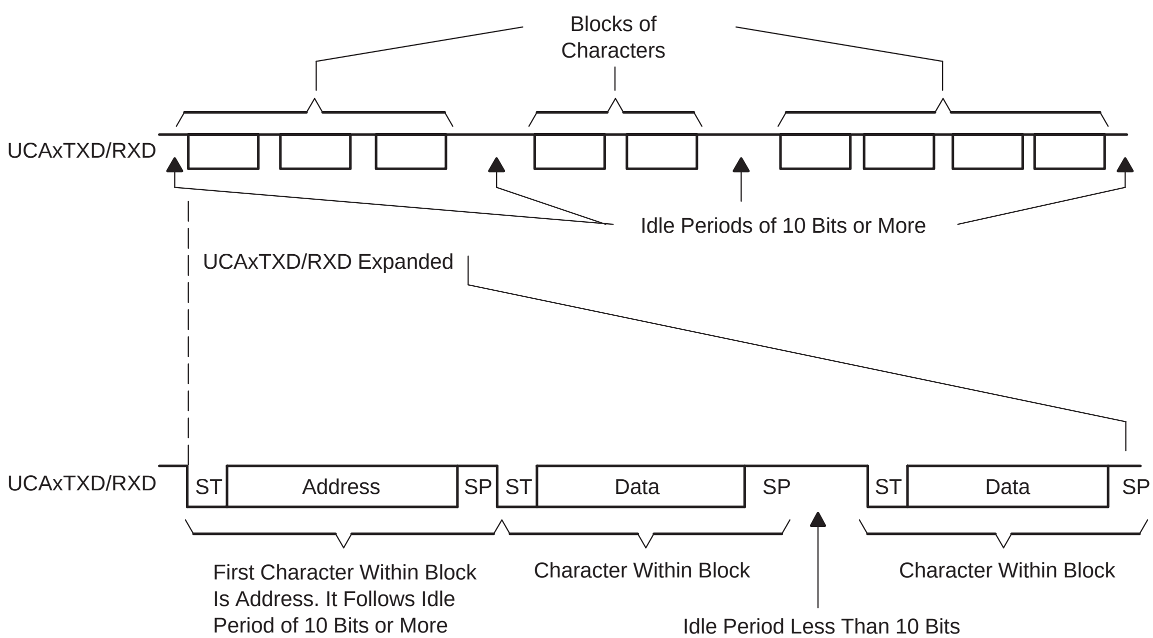
When two devices communicate asynchronously, no multiprocessor format is required for the protocol. When three or more devices communicate, the USCI supports the idle-line and address-bit multiprocessor communication formats.

#### Idle-Line Multiprocessor Format

When UCMODEx = 01, 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 11–3. An idle receive line is detected when 10 or more continuous ones (marks) are received after the one or two stop bits of a character. The baud rate generator is switched off after reception of an idle line until the next start edge is detected. When an idle line is detected the UCIDLE bit is set.

The first character received after an idle period is an address character. The UCIDLE bit is used as an address tag for each block of characters. In idle-line multiprocessor format, this bit is set when a received character is an address

*Figure 11–3. Idle-Line Format*



The UCDORM bit is used to control data reception in the idle-line multiprocessor format. When UCDORM = 1, all non-address characters are assembled but not transferred into the UCAxRXBUF, and interrupts are not generated. When an address character is received, the character is transferred into UCAxRXBUF, UCAxRXIFG is set, and any applicable error flag is set when UCRXEIE = 1. When UCRXEIE = 0 and an address character is recieved but has a framing error or parity error, the character is not transferred into UCAxRXBUF and UCAxRXIFG is not set.

If an address is received, user software can validate the address and must reset UCDORM to continue receiving data. If UCDORM remains set, only address characters will be received. When UCDORM is cleared during the reception of a character the receive interrupt flag will be set after the reception completed. The UCDORM bit is not modified by the USCI hardware automatically.

For address transmission in idle-line multiprocessor format, a precise idle period can be generated by the USCI to generate address character identifiers on UCAxTXD. The double-buffered UCTXADDR flag indicates if the next character loaded into UCAxTXBUF is preceded by an idle line of 11 bits. UCTXADDR is automatically cleared when the start bit is generated.

### Transmitting an Idle Frame

The following procedure sends out an idle frame to indicate an address character followed by associated data:

1) Set UCTXADDR, then write the address character to UCAxTXBUF. UCAxTXBUF must be ready for new data (UCAxTXIFG = 1).

   This generates an idle period of exactly 11 bits followed by the address character. UCTXADDR is reset automatically when the address character is transferred from UCAxTXBUF into the shift register.

2) Write desired data characters to UCAxTXBUF. UCAxTXBUF must be ready for new data (UCAxTXIFG = 1).

   The data written to UCAxTXBUF is transferred to the shift register and transmitted as soon as the shift register is ready for new data.

   The idle-line time must not be exceeded between address and data transmission or between data transmissions. Otherwise, the transmitted data will be misinterpreted as an address.

## Address-Bit Multiprocessor Format

When UCMODEx = 10, the address-bit multiprocessor format is selected. Each processed character contains an extra bit used as an address indicator shown in Figure 11–4. The first character in a block of characters carries a set address bit which indicates that the character is an address. The USCI UCADDR bit is set when a received character has its address bit set and is transferred to UCAxRXBUF.

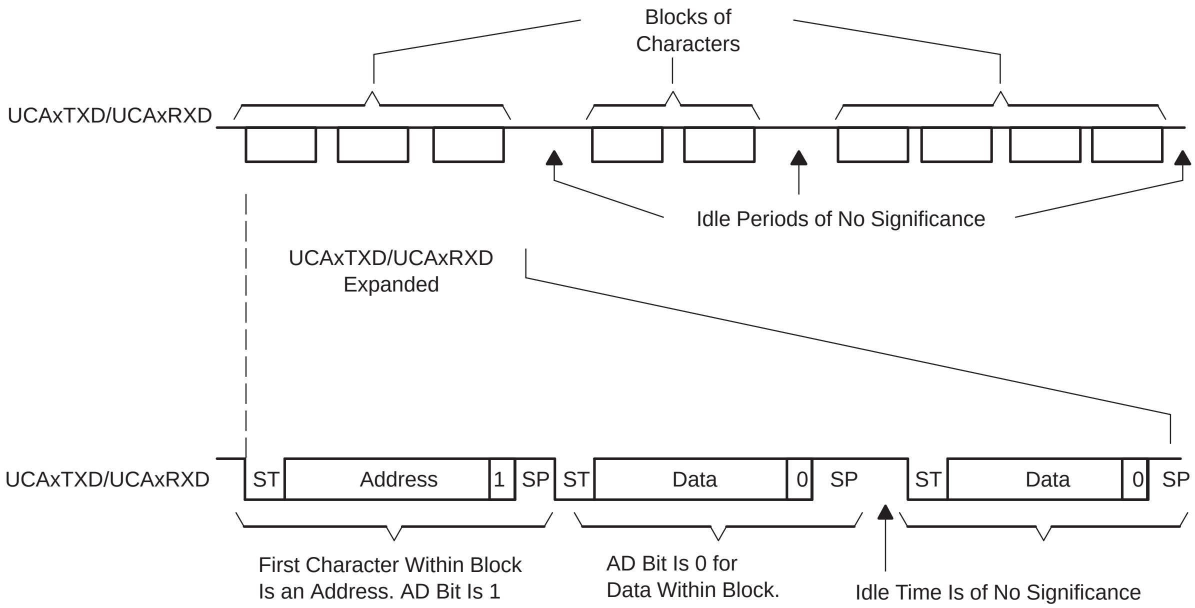
The UCDORM bit is used to control data reception in the address-bit multiprocessor format. When UCDORM is set, data characters with address bit = 0 are assembled by the receiver but are not transferred to UCAxRXBUF and no interrupts are generated. When a character containing a set address bit is received, the character is transferred into UCAxRXBUF, UCAxRXIFG is set, and any applicable error flag is set when UCRXEIE = 1. When UCRXEIE = 0 and a character containing a set address bit is received, but has a framing error or parity error, the character is not transferred into UCAxRXBUF and UCAxRXIFG is not set.

If an address is received, user software can validate the address and must reset UCDORM to continue receiving data. If UCDORM remains set, only address characters with address bit = 1 will be received. The UCDORM bit is not modified by the USCI hardware automatically.

When UCDORM = 0 all received characters will set the receive interrupt flag UCAxRXIFG. If UCDORM is cleared during the reception of a character the receive interrupt flag will be set after the reception is completed.

For address transmission in address-bit multiprocessor mode, the address bit of a character is controlled by the UCTXADDR bit. The value of the UCTXADDR bit is loaded into the address bit of the character transferred from UCAxTXBUF to the transmit shift register. UCTXADDR is automatically cleared when the start bit is generated.

*Figure 11–4. Address-Bit Multiprocessor Format*




## Break Reception and Generation

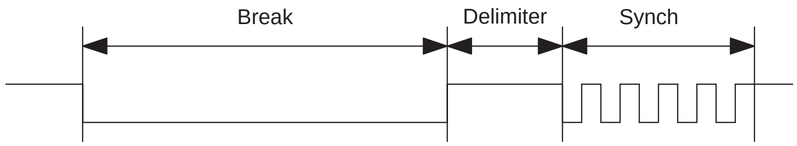
When UCMODEx = 00, 01, or 10 the receiver detects a break when all data, parity, and stop bits are low, regardless of the parity, address mode, or other character settings. When a break is detected, the UCBRK bit is set. If the break interrupt enable bit, UCBRKIE, is set, the receive interrupt flag UCAxRXIFG will also be set. In this case, the value in UCAxRXBUF is 0h since all data bits were zero.

To transmit a break set the UCTXBRK bit, then write 0h to UCAxTXBUF. UCAxTXBUF must be ready for new data (UCAxTXIFG = 1). This generates a break with all bits low. UCTXBRK is automatically cleared when the start bit is generated.

### 11.3.4 Automatic Baud Rate Detection

When UCMODEx = 11 UART mode with automatic baud rate detection is selected. For automatic baud rate detection, a data frame is preceded by a synchronization sequence that consists of a break and a synch field. A break is detected when 11 or more continuous zeros (spaces) are received. If the length of the break exceeds 22 bit times the break timeout error flag UCBTOE is set. The synch field follows the break as shown in Figure 11–5.

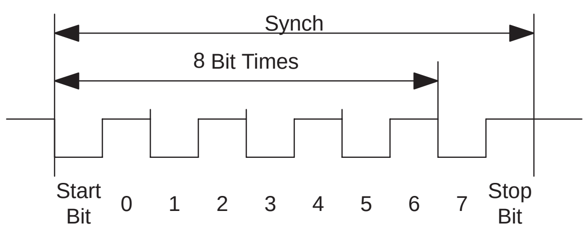
*Figure 11–5. Auto Baud Rate Detection – Break/Synch Sequence*




For LIN conformance the character format should be set to 8 data bits, LSB first, no parity and one stop bit. No address bit is available.

The synch field consists of the data 055h inside a byte field as shown in Figure 11–6. The synchronization is based on the time measurement between the first falling edge and the last falling edge of the pattern. The transmit baud rate generator is used for the measurement if automatic baud rate detection is enabled by setting UCABDEN. Otherwise, the pattern is received but not measured. The result of the measurement is transferred into the baud rate control registers UCAxBR0, UCAxBR1, and UCAxMCTL. If the length of the synch field exceeds the measurable time the synch timeout error flag UCSTOE is set.

*Figure 11–6. Auto Baud Rate Detection – Synch Field*




The UCDORM bit is used to control data reception in this mode. When UCDORM is set, all characters are received but not transferred into the UCAxRXBUF, and interrupts are not generated. When a break/synch field is detected the UCBRK flag is set. The character following the break/synch field is transferred into UCAxRXBUF and the UCAxRXIFG interrupt flag is set. Any applicable error flag is also set. If the UCBRKIE bit is set, reception of the break/synch sets the UCAxRXIFG. The UCBRK bit is reset by user software or by reading the receive buffer UCAxRXBUF.

When a break/synch field is received, user software must reset UCDORM to continue receiving data. If UCDORM remains set, only the character after the next reception of a break/synch field will be received. The UCDORM bit is not modified by the USCI hardware automatically.

When UCDORM = 0 all received characters will set the receive interrupt flag UCAxRXIFG. If UCDORM is cleared during the reception of a character the receive interrupt flag will be set after the reception is complete.

## Transmitting a Break/Synch Field

The following procedure transmits a break/synch field:

1) Set UCTXBRK with UMODEx = 11.

2) Write 055h to UCAxTXBUF. UCAxTXBUF must be ready for new data (UCAxTXIFG = 1).

   This generates a break field of 13 bits followed by a break delimiter and the synch character. The length of the break delimiter is controlled with the UCDELIMx bits. UCTXBRK is reset automatically when the synch character is transferred from UCAxTXBUF into the shift register.

3) Write desired data characters to UCAxTXBUF. UCAxTXBUF must be ready for new data (UCAxTXIFG = 1).

   The data written to UCAxTXBUF is transferred to the shift register and transmitted as soon as the shift register is ready for new data.

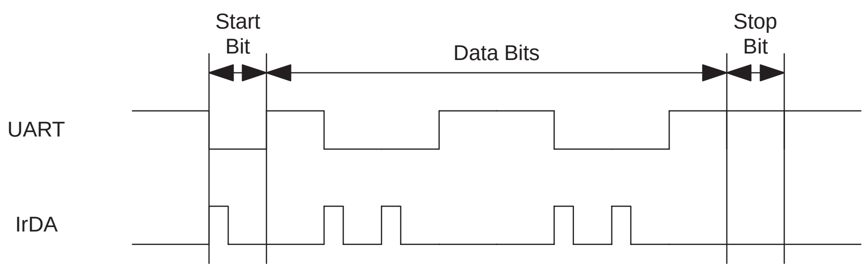
### 11.3.5 IrDA Encoding and Decoding

When UCIREN is set the IrDA encoder and decoder are enabled and provide hardware bit shaping for IrDA communication.

#### IrDA Encoding

The encoder sends a pulse for every zero bit in the transmit bit stream coming from the UART as shown in Figure 11–7. The pulse duration is defined by UCIRTXPLx bits specifying the number of half clock periods of the clock selected by UCIRTXCLK.

*Figure 11–7. UART vs. IrDA Data Format*




To set the pulse time of 3/16 bit period required by the IrDA standard the BITCLK16 clock is selected with UCIRTXCLK = 1 and the pulse length is set to 6 half clock cycles with UCIRTXPLx = 6 – 1 = 5.

When UCIRTXCLK = 0, the pulse length $t_{PULSE}$ is based on BRCLK and is calculated as follows:

$$UCIRTXPLx = t_{PULSE} \cdot 2 \cdot f_{BRCLK} - 1$$

When UCIRTXCLK = 0 the prescaler UCBRx must to be set to a value greater or equal to 5.

#### IrDA Decoding

The decoder detects high pulses when UCIRRXPL = 0. Otherwise it detects low pulses. In addition to the analog deglitch filter an additional programmable digital filter stage can be enabled by setting UCIRRXFE. When UCIRRXFE is set, only pulses longer than the programmed filter length are passed. Shorter pulses are discarded. The equation to program the filter length UCIRRXFLx is:

$$UCIRRXFLx = (t_{PULSE} - t_{WAKE}) \cdot 2 \cdot f_{BRCLK} - 4$$

where:

$t_{PULSE}$: Minimum receive pulse width

$t_{WAKE}$: Wake time from any low power mode. Zero when MSP430 is in active mode.

### 11.3.6 Automatic Error Detection

Glitch suppression prevents the USCI from being accidentally started. Any pulse on UCAxRXD shorter than the deglitch time $t_{\tau}$ (approximately 150 ns) will be ignored. See the device-specific datasheet for parameters.

When a low period on UCAxRXD exceeds $t_{\tau}$ a majority vote is taken for the start bit. If the majority vote fails to detect a valid start bit the USCI halts character reception and waits for the next low period on UCAxRXD. The majority vote is also used for each bit in a character to prevent bit errors.

The USCI module automatically detects framing errors, parity errors, overrun errors, and break conditions when receiving characters. The bits UCFE, UCPE, UCOE, and UCBRK are set when their respective condition is detected. When the error flags UCFE, UCPE or UCOE are set, UCRXERR is also set. The error conditions are described in Table 11–1.

*Table 11–1. Receive Error Conditions*

| Error Condition | Error Flag | Description |
|---|---|---|
| Framing error | UCFE | A framing error occurs when a low stop bit is detected. When two stop bits are used, both stop bits are checked for framing error. When a framing error is detected, the UCFE bit is set. |
| Parity error | UCPE | A parity error is a mismatch between the number of 1s in a character and the value of the parity bit. When an address bit is included in the character, it is included in the parity calculation. When a parity error is detected, the UCPE bit is set. |
| Receive overrun | UCOE | An overrun error occurs when a character is loaded into UCAxRXBUF before the prior character has been read. When an overrun occurs, the UCOE bit is set. |
| Break condition | UCBRK | When not using automatic baud rate detection, a break is detected when all data, parity, and stop bits are low. When a break condition is detected, the UCBRK bit is set. A break condition can also set the interrupt flag UCAxRXIFG if the break interrupt enable UCBRKIE bit is set. |

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

When UCFE, UCPE, UCOE, UCBRK, or UCRXERR is set, the bit remains set until user software resets it or UCAxRXBUF is read. UCOE must be reset by reading UCAxRXBUF. Otherwise it will not function properly.

### 11.3.7 USCI Receive Enable

The USCI module is enabled by clearing the UCSWRST bit and the receiver is ready and in an idle state. The receive baud rate generator is in a ready state but is not clocked nor producing any clocks.

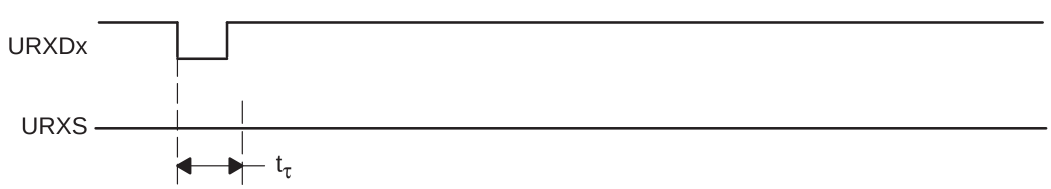
The falling edge of the start bit enables the baud rate generator and the UART state machine checks for a valid start bit. If no valid start bit is detected the UART state machine returns to its idle state and the baud rate generator is turned off again. If a valid start bit is detected a character will be received.

When the idle-line multiprocessor mode is selected with UCMODEx = 01 the UART state machine checks for an idle line after receiving a character. If a start bit is detected another character is received. Otherwise the UCIDLE flag is set after 10 ones are received and the UART state machine returns to its idle state and the baud rate generator is turned off.

#### Receive Data Glitch Suppression

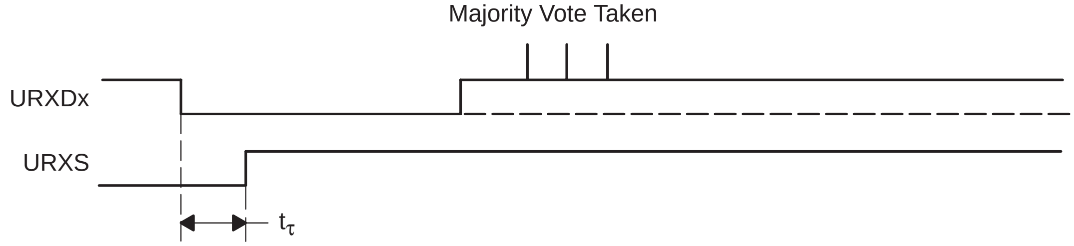
Glitch suppression prevents the USCI from being accidentally started. Any glitch on UCAxRXD shorter than the deglitch time $t_\tau$ (approximately 150 ns) will be ignored by the USCI and further action will be initiated as shown in Figure 11–8. See the device-specific datasheet for parameters.

*Figure 11–8. Glitch Suppression, USCI Receive Not Started*




When a glitch is longer than $t_\tau$, or a valid start bit occurs on UCAxRXD, the USCI receive operation is started and a majority vote is taken as shown in Figure 11–9. If the majority vote fails to detect a start bit the USCI halts character reception.

*Figure 11–9. Glitch Suppression, USCI Activated*



### 11.3.8 USCI Transmit Enable

The USCI module is enabled by clearing the UCSWRST bit and the transmitter is ready and in an idle state. The transmit baud rate generator is ready but is not clocked nor producing any clocks.

A transmission is initiated by writing data to UCAxTXBUF. When this occurs, the baud rate generator is enabled and the data in UCAxTXBUF is moved to the transmit shift register on the next BITCLK after the transmit shift register is empty. UCAxTXIFG is set when new data can be written into UCAxTXBUF.

Transmission continues as long as new data is available in UCAxTXBUF at the end of the previous byte transmission. If new data is not in UCAxTXBUF when the previous byte has transmitted, the transmitter returns to its idle state and the baud rate generator is turned off.

### 11.3.9 UART Baud Rate Generation

The USCI baud rate generator is capable of producing standard baud rates from non-standard source frequencies. It provides two modes of operation selected by the UCOS16 bit.

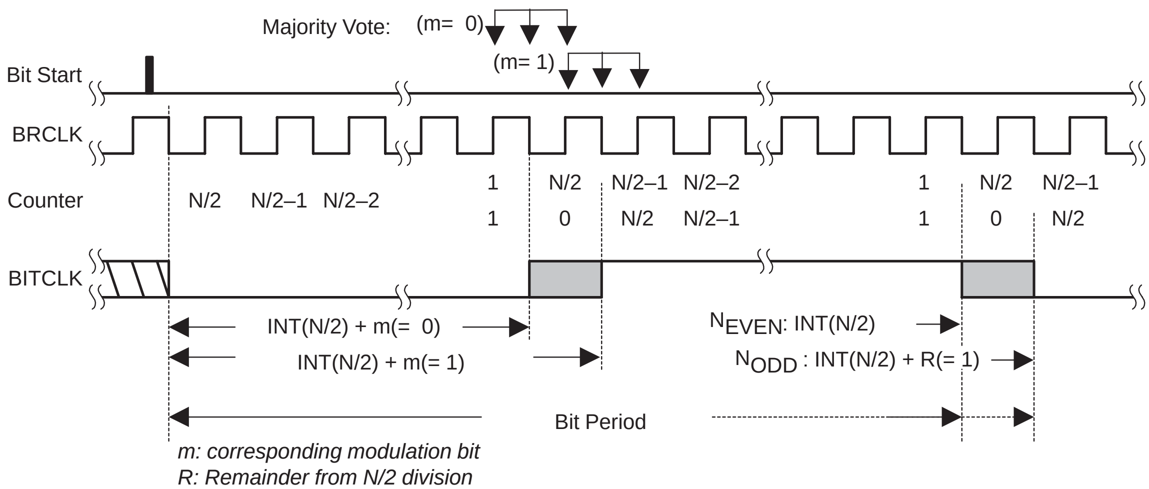
#### Low-Frequency Baud Rate Generation

The low-frequency mode is selected when UCOS16 = 0. This mode allows generation of baud rates from low frequency clock sources (e.g. 9600 baud from a 32768Hz crystal). By using a lower input frequency the power consumption of the module is reduced. Using this mode with higher frequencies and higher prescaler settings will cause the majority votes to be taken in an increasingly smaller window and thus decrease the benefit of the majority vote.

In low-frequency mode the baud rate generator uses one prescaler and one modulator to generate bit clock timing. This combination supports fractional divisors for baud rate generation. In this mode, the maximum USCI baud rate is one-third the UART source clock frequency BRCLK.

Timing for each bit is shown in Figure 11–10. For each bit received, a majority vote is taken to determine the bit value. These samples occur at the N/2 – 1/2, N/2, and N/2 + 1/2 BRCLK periods, where N is the number of BRCLKs per BITCLK.

*Figure 11–10. BITCLK Baud Rate Timing with UCOS16 = 0*




Modulation is based on the UCBRSx setting as shown in Table 11–2. A 1 in the table indicates that m = 1 and the corresponding BITCLK period is one BRCLK period longer than a BITCLK period with m = 0. The modulation wraps around after 8 bits but restarts with each new start bit.

*Table 11–2. BITCLK Modulation Pattern*

| UCBRSx | Bit 0 (Start Bit) | Bit 1 | Bit 2 | Bit 3 | Bit 4 | Bit 5 | Bit 6 | Bit 7 |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| **3** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **4** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| **5** | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| **6** | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| **7** | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

## Oversampling Baud Rate Generation

The oversampling mode is selected when UCOS16 = 1. This mode supports sampling a UART bit stream with higher input clock frequencies. This results in majority votes that are always 1/16 of a bit clock period apart. This mode also easily supports IrDA pulses with a 3/16 bit-time when the IrDA encoder and decoder are enabled.

This mode uses one prescaler and one modulator to generate the BITCLK16 clock that is 16 times faster than the BITCLK. An additional divider and modulator stage generates BITCLK from BITCLK16. This combination supports fractional divisions of both BITCLK16 and BITCLK for baud rate generation. In this mode, the maximum USCI baud rate is 1/16 the UART source clock frequency BRCLK. When UCBRx is set to 0 or 1 the first prescaler and modulator stage is bypassed and BRCLK is equal to BITCLK16.

Modulation for BITCLK16 is based on the UCBRFx setting as shown in Table 11–3. A 1 in the table indicates that the corresponding BITCLK16 period is one BRCLK period longer than the periods m=0. The modulation restarts with each new bit timing.

Modulation for BITCLK is based on the UCBRSx setting as shown in Table 11–2 as previously described.

*Table 11–3. BITCLK16 Modulation Pattern*

| UCBRFx | No. of BITCLK16 Clocks after last falling BITCLK edge | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **00h** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **01h** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **02h** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **03h** | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **04h** | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **05h** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **06h** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| **07h** | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| **08h** | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| **09h** | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| **0Ah** | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| **0Bh** | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| **0Ch** | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| **0Dh** | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| **0Eh** | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **0Fh** | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

### 11.3.10 Setting a Baud Rate

For a given BRCLK clock source, the baud rate used determines the required division factor N:

$$N = \frac{f_{BRCLK}}{Baudrate}$$

The division factor N is often a non-integer value thus at least one divider and one modulator stage is used to meet the factor as closely as possible.

If N is equal or greater than 16 the oversampling baud rate generation mode can be chosen by setting UCOS16.

#### Low–Frequency Baud Rate Mode Setting

In the low-frequency mode, the integer portion of the divisor is realized by the prescaler:

UCBRx = INT(N)

and the fractional portion is realized by the modulator with the following nominal formula:

UCBRSx = round( ( N – INT(N) ) * 8 )

Incrementing or decrementing the UCBRSx setting by one count may give a lower maximum bit error for any given bit. To determine if this is the case, a detailed error calculation must be performed for each bit for each UCBRSx setting.

#### Oversampling Baud Rate Mode Setting

In the oversampling mode the prescaler is set to:

UCBRx = INT(N/16).

and the first stage modulator is set to:

UCBRFx = round( ( (N/16) – INT(N/16) ) * 16 )

When greater accuracy is required, the UCBRSx modulator can also be implemented with values from 0 – 7. To find the setting that gives the lowest maximum bit error rate for any given bit, a detailed error calculation must be performed for all settings of UCBRSx from 0 – 7 with the initial UCBRFx setting and with the UCBRFx setting incremented and decremented by one.

### 11.3.11 Transmit Bit Timing

The timing for each character is the sum of the individual bit timings. Using the modulation features of the baud rate generator reduces the cumulative bit error. The individual bit error can be calculated using the following steps.

#### Low–Frequency Baud Rate Mode Bit Timing

In low-frequency mode, calculate the length of bit i $T_{bit,TX}[i]$ based on the UCBRx and UCBRSx settings:

$$T_{bit,TX}[i] = \frac{1}{f_{BRCLK}}\left(UCBRx + m_{UCBRSx}[i]\right)$$

where:

$m_{UCBRSx}[i]$: Modulation of bit i from Table 11–2

#### Oversampling Baud Rate Mode Bit Timing

In oversampling baud rate mode calculate the length of bit i $T_{bit,TX}[i]$ based on the baud rate generator UCBRx, UCBRFx and UCBRSx settings:

$$T_{bit,TX}[i] = \frac{1}{f_{BRCLK}}\left(\left(16 + m_{UCBRSx}[i]\right) \cdot UCBRx + \sum_{j=0}^{15} m_{UCBRFx}[j]\right)$$

where:

$\sum_{j=0}^{15} m_{UCBRFx}[j]$: Sum of ones from the corresponding row in Table 11–3

$m_{UCBRSx}[i]$: Modulation of bit i from Table 11–2

This results in an end-of-bit time $t_{bit,TX}[i]$ equal to the sum of all previous and the current bit times:

$$t_{bit,TX}[i] = \sum_{j=0}^{i} T_{bit,TX}[j]$$

To calculate bit error, this time is compared to the ideal bit time $t_{bit,ideal,TX}[i]$:

$$t_{bit,ideal,TX}[i] = \frac{1}{Baudrate}(i + 1)$$

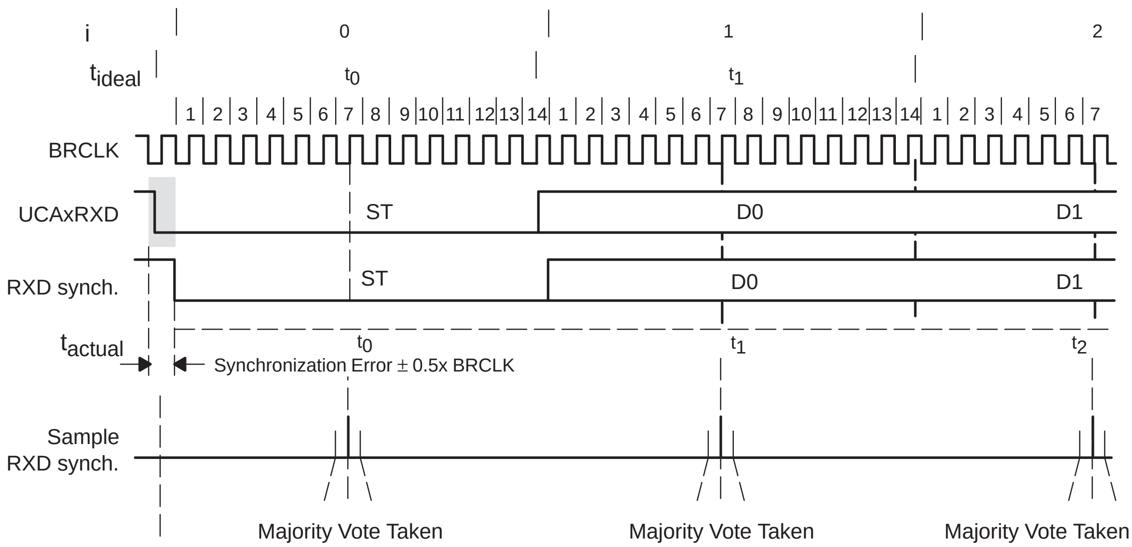
This results in an error normalized to one ideal bit time (1/baudrate):

$$Error_{TX}[i] = \left(t_{bit,TX}[i] - t_{bit,ideal,TX}[i]\right) \cdot Baudrate \cdot 100\%$$

### 11.3.12 Receive Bit Timing

Receive timing error consists of two error sources. The first is the bit-to-bit timing error similar to the transmit bit timing error. The second is the error between a start edge occurring and the start edge being accepted by the USCI module. Figure 11–11 shows the asynchronous timing errors between data on the UCAxRXD pin and the internal baud-rate clock. This results in an additional synchronization error. The synchronization error $t_{SYNC}$ is between –0.5 BRCLKs and +0.5 BRCLKs independent of the selected baud rate generation mode.

*Figure 11–11. Receive Error*




The ideal sampling time $t_{bit,ideal,RX}[i]$ is in the middle of a bit period:

$$t_{bit,ideal,RX}[i] = \frac{1}{\text{Baudrate}}(i + 0.5)$$

The real sampling time $t_{bit,RX}[i]$ is equal to the sum of all previous bits according to the formulas shown in the transmit timing section, plus one half BITCLK for the current bit i, plus the synchronization error $t_{SYNC}$.

This results in the following $t_{bit,RX}[i]$ for the low-frequency baud rate mode

$$t_{bit,RX}[i] = t_{SYNC} + \sum_{j=0}^{i-1} T_{bit,RX}[j] + \frac{1}{f_{BRCLK}}\left(\text{INT}\left(\frac{1}{2}\text{UCBRx}\right) + m_{UCBRSx}[i]\right)$$

where:

$$T_{bit,RX}[i] = \frac{1}{f_{BRCLK}}\left(\text{UCBRx} + m_{UCBRSx}[i]\right)$$

$m_{UCBRSx}[i]$: Modulation of bit i from Table 11–2

For the oversampling baud rate mode the sampling time $t_{bit,RX}[i]$ of bit i is calculated by:

$$t_{bit,RX}[i] = t_{SYNC} + \sum_{j=0}^{i-1} T_{bit,RX}[j] + \frac{1}{f_{BRCLK}}\left(\left(8 + m_{UCBRSx}[i]\right) \cdot UCBRx + \sum_{j=0}^{7+m_{UCBRSx}[i]} m_{UCBRFx}[j]\right)$$

where:

$$T_{bit,RX}[i] = \frac{1}{f_{BRCLK}}\left(\left(16 + m_{UCBRSx}[i]\right) \cdot UCBRx + \sum_{j=0}^{15} m_{UCBRFx}[j]\right)$$

$\sum_{j=0}^{7+m_{UCBRSx}[i]} m_{UCBRFx}[j]$: Sum of ones from columns $0 - 7 + m_{UCBRSx}[i]$ from the corresponding row in Table 11–3

$m_{UCBRSx}[i]$: Modulation of bit i from Table 11–2

This results in an error normalized to one ideal bit time (1/baudrate) according to the following formula:

$$Error_{RX}[i] = \left(t_{bit,RX}[i] - t_{bit,ideal,RX}[i]\right) \cdot Baudrate \cdot 100\%$$

### 11.3.13 Typical Baud Rates and Errors

Standard baud rate data for UCBRx, UCBRSx and UCBRFx are listed in Table 11–4 and Table 11–5 for a 32,768 Hz crystal sourcing ACLK and typical SMCLK frequencies.

The receive error is the accumulated time versus the ideal scanning time in the middle of each bit. The worst case error is given for the reception of an 8-bit character with parity and one stop bit including synchronization error.

The transmit error is the accumulated timing error versus the ideal time of the bit period. The worst case error is given for the transmission of an 8-bit character with parity and stop bit.

*Table 11–4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0*

| BRCLK frequency [Hz] | Baud Rate [Baud] | UCBRx | UCBRSx | UCBRFx | Max. TX Error [%] | | Max. RX Error [%] | |
|---|---|---|---|---|---|---|---|---|
| **32,768** | **1200** | 27 | 2 | 0 | –2.8 | 1.4 | –5.9 | 2.0 |
| **32,768** | **2400** | 13 | 6 | 0 | –4.8 | 6.0 | –9.7 | 8.3 |
| **32,768** | **4800** | 6 | 7 | 0 | –12.1 | 5.7 | –13.4 | 19.0 |
| **32,768** | **9600** | 3 | 3 | 0 | –21.1 | 15.2 | –44.3 | 21.3 |
| **1,048,576** | **9600** | 109 | 2 | 0 | –0.2 | 0.7 | –1.0 | 0.8 |
| **1,048,576** | **19200** | 54 | 5 | 0 | –1.1 | 1.0 | –1.5 | 2.5 |
| **1,048,576** | **38400** | 27 | 2 | 0 | –2.8 | 1.4 | –5.9 | 2.0 |
| **1,048,576** | **56000** | 18 | 6 | 0 | –3.9 | 1.1 | –4.6 | 5.7 |
| **1,048,576** | **115200** | 9 | 1 | 0 | –1.1 | 10.7 | –11.5 | 11.3 |
| **1,048,576** | **128000** | 8 | 1 | 0 | –8.9 | 7.5 | –13.8 | 14.8 |
| **1,048,576** | **256000** | 4 | 1 | 0 | –2.3 | 25.4 | –13.4 | 38.8 |
| **1,000,000** | **9600** | 104 | 1 | 0 | –0.5 | 0.6 | –0.9 | 1.2 |
| **1,000,000** | **19200** | 52 | 0 | 0 | –1.8 | 0 | –2.6 | 0.9 |
| **1,000,000** | **38400** | 26 | 0 | 0 | –1.8 | 0 | –3.6 | 1.8 |
| **1,000,000** | **56000** | 17 | 7 | 0 | –4.8 | 0.8 | –8.0 | 3.2 |
| **1,000,000** | **115200** | 8 | 6 | 0 | –7.8 | 6.4 | –9.7 | 16.1 |
| **1,000,000** | **128000** | 7 | 7 | 0 | –10.4 | 6.4 | –18.0 | 11.6 |
| **1,000,000** | **256000** | 3 | 7 | 0 | –29.6 | 0 | –43.6 | 5.2 |
| **4,000,000** | **9600** | 416 | 6 | 0 | –0.2 | 0.2 | –0.2 | 0.4 |
| **4,000,000** | **19200** | 208 | 3 | 0 | –0.2 | 0.5 | –0.3 | 0.8 |
| **4,000,000** | **38400** | 104 | 1 | 0 | –0.5 | 0.6 | –0.9 | 1.2 |
| **4,000,000** | **56000** | 71 | 4 | 0 | –0.6 | 1.0 | –1.7 | 1.3 |
| **4,000,000** | **115200** | 34 | 6 | 0 | –2.1 | 0.6 | –2.5 | 3.1 |
| **4,000,000** | **128000** | 31 | 2 | 0 | –0.8 | 1.6 | –3.6 | 2.0 |
| **4,000,000** | **256000** | 15 | 5 | 0 | –4.0 | 3.2 | –8.4 | 5.2 |
| **8,000,000** | **9600** | 833 | 2 | 0 | –0.1 | 0 | –0.2 | 0.1 |
| **8,000,000** | **19200** | 416 | 6 | 0 | –0.2 | 0.2 | –0.2 | 0.4 |
| **8,000,000** | **38400** | 208 | 3 | 0 | –0.2 | 0.5 | –0.3 | 0.8 |
| **8,000,000** | **56000** | 142 | 7 | 0 | –0.6 | 0.1 | –0.7 | 0.8 |
| **8,000,000** | **115200** | 69 | 4 | 0 | –0.6 | 0.8 | –1.8 | 1.1 |
| **8,000,000** | **128000** | 62 | 4 | 0 | –0.8 | 0 | –1.2 | 1.2 |
| **8,000,000** | **256000** | 31 | 2 | 0 | –0.8 | 1.6 | –3.6 | 2.0 |

Table 11–4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0 (Continued)

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **12,000,000** | **9600** | 1250 | 0 | 0 | 0 | 0 | –0.05 | 0.05 |
| **12,000,000** | **19200** | 625 | 0 | 0 | 0 | 0 | –0.2 | 0 |
| **12,000,000** | **38400** | 312 | 4 | 0 | –0.2 | 0 | –0.2 | 0.2 |
| **12,000,000** | **56000** | 214 | 2 | 0 | –0.3 | 0.2 | –0.4 | 0.5 |
| **12,000,000** | **115200** | 104 | 1 | 0 | –0.5 | 0.6 | –0.9 | 1.2 |
| **12,000,000** | **128000** | 93 | 6 | 0 | –0.8 | 0 | –1.5 | 0.4 |
| **12,000,000** | **256000** | 46 | 7 | 0 | –1.9 | 0 | –2.0 | 2.0 |
| **16,000,000** | **9600** | 1666 | 6 | 0 | –0.05 | 0.05 | –0.05 | 0.1 |
| **16,000,000** | **19200** | 833 | 2 | 0 | –0.1 | 0.05 | –0.2 | 0.1 |
| **16,000,000** | **38400** | 416 | 6 | 0 | –0.2 | 0.2 | –0.2 | 0.4 |
| **16,000,000** | **56000** | 285 | 6 | 0 | –0.3 | 0.1 | –0.5 | 0.2 |
| **16,000,000** | **115200** | 138 | 7 | 0 | –0.7 | 0 | –0.8 | 0.6 |
| **16,000,000** | **128000** | 125 | 0 | 0 | 0 | 0 | –0.8 | 0 |
| **16,000,000** | **256000** | 62 | 4 | 0 | –0.8 | 0 | –1.2 | 1.2 |

Table 11–5. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 1

| BRCLK frequency [Hz] | Baud Rate [Baud] | UCBRx | UCBRSx | UCBRFx | Max. TX Error [%] | | Max. RX Error [%] | |
|---|---|---|---|---|---|---|---|---|
| **32,768** | **1200** | 1 | 2 | 11 | –2.8 | 1.4 | –2.2 | 5.6 |
| **1,048,576** | **9600** | 6 | 0 | 13 | –2.3 | 0 | –2.2 | 0.8 |
| **1,048,576** | **19200** | 3 | 1 | 6 | –4.6 | 3.2 | –5.0 | 4.7 |
| **1,048,576** | **38400** | 1 | 2 | 11 | –2.8 | 1.4 | –2.2 | 5.6 |
| **1,048,576** | **56000** | 1 | 6 | 2 | –3.9 | 1.1 | –4.6 | 5.7 |
| **1,000,000** | **9600** | 6 | 0 | 8 | –1.8 | 0 | –2.2 | 0.4 |
| **1,000,000** | **19200** | 3 | 0 | 4 | –1.8 | 0 | –2.6 | 0.9 |
| **1,000,000** | **38400** | 1 | 0 | 10 | –1.8 | 0 | –3.6 | 1.8 |
| **1,000,000** | **56000** | 1 | 7 | 1 | –4.8 | 0.8 | –2.4 | 8.8 |
| **4,000,000** | **9600** | 26 | 0 | 1 | 0 | 0.9 | 0 | 1.1 |
| **4,000,000** | **19200** | 13 | 0 | 0 | –1.8 | 0 | –1.9 | 0.2 |
| **4,000,000** | **38400** | 6 | 0 | 8 | –1.8 | 0 | –2.2 | 0.4 |
| **4,000,000** | **56000** | 4 | 5 | 5 | –3.4 | 3.2 | –1.7 | 6.3 |
| **4,000,000** | **115200** | 2 | 3 | 2 | –2.1 | 4.8 | –2.5 | 7.3 |
| **4,000,000** | **128000** | 1 | 2 | 15 | –0.8 | 1.6 | –3.6 | 5.2 |
| **8,000,000** | **9600** | 52 | 0 | 1 | –0.4 | 0 | –0.4 | 0.1 |
| **8,000,000** | **19200** | 26 | 0 | 1 | 0 | 0.9 | 0 | 1.1 |
| **8,000,000** | **38400** | 13 | 0 | 0 | –1.8 | 0 | –1.9 | 0.2 |
| **8,000,000** | **56000** | 8 | 0 | 15 | 0 | 1.1 | –0.7 | 1.1 |
| **8,000,000** | **115200** | 4 | 5 | 3 | –3.5 | 3.2 | –1.8 | 6.4 |
| **8,000,000** | **128000** | 3 | 4 | 13 | –2.4 | 0 | –1.2 | 2.8 |
| **8,000,000** | **256000** | 1 | 2 | 15 | –0.8 | 1.6 | –3.6 | 5.2 |
| **12,000,000** | **9600** | 78 | 0 | 2 | 0 | 0 | –0.05 | 0.05 |
| **12,000,000** | **19200** | 39 | 0 | 1 | 0 | 0 | 0 | 0.2 |
| **12,000,000** | **38400** | 19 | 0 | 8 | –1.8 | 0 | –1.8 | 0.1 |
| **12,000,000** | **56000** | 13 | 0 | 6 | –1.5 | 0 | –1.6 | 0.2 |
| **12,000,000** | **115200** | 6 | 0 | 8 | –1.8 | 0 | –2.2 | 0.4 |
| **12,000,000** | **128000** | 5 | 0 | 14 | 0 | 2.9 | –0.4 | 3.3 |
| **12,000,000** | **256000** | 2 | 4 | 14 | –1.9 | 2.7 | –2.0 | 4.7 |
| **16,000,000** | **9600** | 104 | 0 | 3 | 0 | 0.2 | 0 | 0.3 |
| **16,000,000** | **19200** | 52 | 0 | 1 | –0.4 | 0 | –0.4 | 0.1 |
| **16,000,000** | **38400** | 26 | 0 | 1 | 0 | 0.9 | 0 | 1.1 |
| **16,000,000** | **56000** | 17 | 0 | 14 | 0 | 1.1 | –0.1 | 1.2 |
| **16,000,000** | **115200** | 8 | 0 | 11 | 0 | 0.9 | 0 | 1.6 |
| **16,000,000** | **128000** | 7 | 0 | 13 | 0 | 0 | 0 | 0.8 |
| **16,000,000** | **256000** | 3 | 4 | 13 | –2.4 | 0 | –1.2 | 2.8 |

### 11.3.14 Using the USCI Module in UART Mode with Low Power Modes

The USCI module provides automatic clock activation for SMCLK for use with low-power modes. When SMCLK is the USCI clock source, and is inactive because the device is in a low-power mode, the USCI module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the USCI module returns to its idle condition. After the USCI module returns to the idle condition, control of the clock source reverts to the settings of its control bits. Automatic clock activation is not provided for ACLK.

When the USCI module activates an inactive clock source, the clock source becomes active for the whole device and any peripheral configured to use the clock source may be affected. For example, a timer using SMCLK will increment while the USCI module forces SMCLK active.

### 11.3.15 USCI Interrupts

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

#### USCI Transmit Interrupt Operation

The UCAxTXIFG interrupt flag is set by the transmitter to indicate that UCAxTXBUF is ready to accept another character. An interrupt request is generated if UCAxTXIE and GIE are also set. UCAxTXIFG is automatically reset if a character is written to UCAxTXBUF.

UCAxTXIFG is set after a PUC or when UCSWRST = 1. UCAxTXIE is reset after a PUC or when UCSWRST = 1.

#### USCI Receive Interrupt Operation

The UCAxRXIFG interrupt flag is set each time a character is received and loaded into UCAxRXBUF. An interrupt request is generated if UCAxRXIE and GIE are also set. UCAxRXIFG and UCAxRXIE are reset by a system reset PUC signal or when UCSWRST = 1. UCAxRXIFG is automatically reset when UCAxRXBUF is read.

Additional interrupt control features include:

- When UCAxRXEIE = 0 erroneous characters will not set UCAxRXIFG.
- When UCDORM = 1, non-address characters will not set UCAxRXIFG in multiprocessor modes.
- When UCBRKIE = 1 a break condition will set the UCBRK bit and the UCAxRXIFG flag.

## USCI Interrupt Usage

USCI_Ax and USCI_Bx share the same interrupt vectors. The receive interrupt flags UCAxRXIFG and UCBxRXIFG are routed to one interrupt vector, the transmit interrupt flags UCAxTXIFG and UCBxTXIFG share another interrupt vector.

### *Shared Interrupt Vectors Software Example*

The following software example shows an extract of an interrupt service routine to handle data receive interrupts from USCI_A0 in either UART or SPI mode and USCI_B0 in SPI mode.

```
USCIA0_RX_USCIB0_RX_ISR
   BIT.B  #UCA0RXIFG, &IFG2  ; USCI_A0 Receive Interrupt?
   JNZ    USCIA0_RX_ISR
USCIB0_RX_ISR?
   ; Read UCB0RXBUF (clears UCB0RXIFG)
   ...
   RETI
USCIA0_RX_ISR
   ; Read UCA0RXBUF (clears UCA0RXIFG)
   ...
   RETI
```

The following software example shows an extract of an interrupt service routine to handle data transmit interrupts from USCI_A0 in either UART or SPI mode and USCI_B0 in SPI mode.

```
USCIA0_TX_USCIB0_TX_ISR
   BIT.B  #UCA0TXIFG, &IFG2  ; USCI_A0 Transmit Interrupt?
   JNZ    USCIA0_TX_ISR
USCIB0_TX_ISR
   ; Write UCB0TXBUF (clears UCB0TXIFG)
   ...
   RETI
USCIA0_TX_ISR
   ; Write UCA0TXBUF  (clears UCA0TXIFG)
   ...
   RETI
```

## 11.4 USCI Registers: UART Mode

The USCI registers applicable in UART mode are listed in Table 11–6 and Table 11–7.

*Table 11–6. USCI_A0 Control and Status Registers*

| Register | Short Form | Register Type | Address | Initial State |
|---|---|---|---|---|
| USCI_A0 control register 0 | UCA0CTL0 | Read/write | 060h | Reset with PUC |
| USCI_A0 control register 1 | UCA0CTL1 | Read/write | 061h | 001h with PUC |
| USCI_A0 Baud rate control register 0 | UCA0BR0 | Read/write | 062h | Reset with PUC |
| USCI_A0 Baud rate control register 1 | UCA0BR1 | Read/write | 063h | Reset with PUC |
| USCI_A0 modulation control register | UCA0MCTL | Read/write | 064h | Reset with PUC |
| USCI_A0 status register | UCA0STAT | Read/write | 065h | Reset with PUC |
| USCI_A0 Receive buffer register | UCA0RXBUF | Read | 066h | Reset with PUC |
| USCI_A0 Transmit buffer register | UCA0TXBUF | Read/write | 067h | Reset with PUC |
| USCI_A0 Auto Baud control register | UCA0ABCTL | Read/write | 05Dh | Reset with PUC |
| USCI_A0 IrDA Transmit control register | UCA0IRTCTL | Read/write | 05Eh | Reset with PUC |
| USCI_A0 IrDA Receive control register | UCA0IRRCTL | Read/write | 05Fh | Reset with PUC |
| SFR interrupt enable register 2 | IE2 | Read/write | 001h | Reset with PUC |
| SFR interrupt flag register 2 | IFG2 | Read/write | 003h | 00Ah with 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.

*Table 11–7. USCI_A1 Control and Status Registers*

| Register | Short Form | Register Type | Address | Initial State |
|---|---|---|---|---|
| USCI_A1 control register 0 | UCA1CTL0 | Read/write | 0D0h | Reset with PUC |
| USCI_A1 control register 1 | UCA1CTL1 | Read/write | 0D1h | 001h with PUC |
| USCI_A1 Baud rate control register 0 | UCA1BR0 | Read/write | 0D2h | Reset with PUC |
| USCI_A1 Baud rate control register 1 | UCA1BR1 | Read/write | 0D3h | Reset with PUC |
| USCI_A1 modulation control register | UCA10MCTL | Read/write | 0D4h | Reset with PUC |
| USCI_A1 status register | UCA1STAT | Read/write | 0D5h | Reset with PUC |
| USCI_A1 Receive buffer register | UCA1RXBUF | Read | 0D6h | Reset with PUC |
| USCI_A1 Transmit buffer register | UCA1TXBUF | Read/write | 0D7h | Reset with PUC |
| USCI_A1 Auto Baud control register | UCA1ABCTL | Read/write | 0CDh | Reset with PUC |
| USCI_A1 IrDA Transmit control register | UCA1IRTCTL | Read/write | 0CEh | Reset with PUC |
| USCI_A1 IrDA Receive control register | UCA1IRRCTL | Read/write | 0CFh | Reset with PUC |
| USCI_A1/B1 interrupt enable register | UC1IE | Read/write | 006h | Reset with PUC |
| USCI_A1/B1 interrupt flag register | UC1IFG | Read/write | 007h | 00Ah with PUC |

## UCAxCTL0, USCI_Ax Control Register 0

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCPEN | UCPAR | UCMSB | UC7BIT | UCSPB | UCMODEx | | UCSYNC=0 |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 |

| | | |
|---|---|---|
| **UCPEN** | Bit 7 | Parity enable<br>0 Parity disabled.<br>1 Parity enabled. Parity bit is generated (UCAxTXD) and expected (UCAxRXD). In address-bit multiprocessor mode, the address bit is included in the parity calculation. |
| **UCPAR** | Bit 6 | Parity select. UCPAR is not used when parity is disabled.<br>0 Odd parity<br>1 Even parity |
| **UCMSB** | Bit 5 | MSB first select. Controls the direction of the receive and transmit shift register.<br>0 LSB first<br>1 MSB first |
| **UC7BIT** | Bit 4 | Character length. Selects 7-bit or 8-bit character length.<br>0 8-bit data<br>1 7-bit data |
| **UCSPB** | Bit 3 | Stop bit select. Number of stop bits.<br>0 One stop bit<br>1 Two stop bits |
| **UCMODEx** | Bits 2–1 | USCI mode. The UCMODEx bits select the asynchronous mode when UCSYNC = 0.<br>00 UART Mode.<br>01 Idle-Line Multiprocessor Mode.<br>10 Address-Bit Multiprocessor Mode.<br>11 UART Mode with automatic baud rate detection. |
| **UCSYNC** | Bit 0 | Synchronous mode enable<br>0 Asynchronous mode<br>1 Synchronous Mode |

## UCAxCTL1, USCI_Ax Control Register 1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCSSELx | | UCRXEIE | UCBRKIE | UCDORM | UCTXADDR | UCTXBRK | UCSWRST |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–1 |

| | | |
|---|---|---|
| **UCSSELx** | Bits 7-6 | USCI clock source select. These bits select the BRCLK source clock.<br>00 UCLK<br>01 ACLK<br>10 SMCLK<br>11 SMCLK |
| **UCRXEIE** | Bit 5 | Receive erroneous-character interrupt-enable<br>0 Erroneous characters rejected and UCAxRXIFG is not set<br>1 Erroneous characters received will set UCAxRXIFG |
| **UCBRKIE** | Bit 4 | Receive break character interrupt-enable<br>0 Received break characters do not set UCAxRXIFG.<br>1 Received break characters set UCAxRXIFG. |
| **UCDORM** | Bit 3 | Dormant. Puts USCI into sleep mode.<br>0 Not dormant. All received characters will set UCAxRXIFG.<br>1 Dormant. Only characters that are preceded by an idle-line or with address bit set will set UCAxRXIFG. In UART mode with automatic baud rate detection only the combination of a break and synch field will set UCAxRXIFG. |
| **UCTXADDR** | Bit 2 | Transmit address. Next frame to be transmitted will be marked as address depending on the selected multiprocessor mode.<br>0 Next frame transmitted is data<br>1 Next frame transmitted is an address |
| **UCTXBRK** | Bit 1 | Transmit break. Transmits a break with the next write to the transmit buffer. In UART mode with automatic baud rate detection 055h must be written into UCAxTXBUF to generate the required break/synch fields. Otherwise 0h must be written into the transmit buffer.<br>0 Next frame transmitted is not a break<br>1 Next frame transmitted is a break or a break/synch |
| **UCSWRST** | Bit 0 | Software reset enable<br>0 Disabled. USCI reset released for operation.<br>1 Enabled. USCI logic held in reset state. |

## UCAxBR0, USCI_Ax Baud Rate Control Register 0

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCBRx | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

## UCAxBR1, USCI_Ax Baud Rate Control Register 1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCBRx | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

| | |
|---|---|
| **UCBRx** | Clock prescaler setting of the Baud rate generator. |

## UCAxMCTL, USCI_Ax Modulation Control Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCBRFx | | | | UCBRSx | | | UCOS16 |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 |

| | | |
|---|---|---|
| **UCBRFx** | Bits 7–4 | First modulation stage select. These bits determine the modulation pattern for BITCLK16 when UCOS16 = 1. Ignored with UCOS16 = 0. Table 11–3 shows the modulation pattern. |
| **UCBRSx** | Bits 3–1 | Second modulation stage select. These bits determine the modulation pattern for BITCLK. Table 11–2 shows the modulation pattern. |
| **UCOS16** | Bit 0 | Oversampling mode enabled<br>0 Disabled<br>1 Enabled |

## UCAxSTAT, USCI_Ax Status Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCLISTEN | UCFE | UCOE | UCPE | UCBRK | UCRXERR | UCADDR UCIDLE | UCBUSY |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | r–0 |

| | | |
|---|---|---|
| **UCLISTEN** | Bit 7 | Listen enable. The UCLISTEN bit selects loopback mode.<br>0 Disabled<br>1 Enabled. UCAxTXD is internally fed back to the receiver. |
| **UCFE** | Bit 6 | Framing error flag<br>0 No error<br>1 Character received with low stop bit |
| **UCOE** | Bit 5 | Overrun error flag. This bit is set when a character is transferred into UCAxRXBUF before the previous character was read. UCOE is cleared automatically when UCxRXBUF is read, and must not be cleared by software. Otherwise, it will not function correctly.<br>0 No error<br>1 Overrun error occurred |
| **UCPE** | Bit 4 | Parity error flag. When UCPEN = 0, UCPE is read as 0.<br>0 No error<br>1 Character received with parity error |
| **UCBRK** | Bit 3 | Break detect flag<br>0 No break condition<br>1 Break condition occurred |
| **UCRXERR** | Bit 2 | Receive error flag. This bit indicates a character was received with error(s). When UCRXERR = 1, on or more error flags (UCFE, UCPE, UCOE) is also set. UCRXERR is cleared when UCAxRXBUF is read.<br>0 No receive errors detected<br>1 Receive error detected |
| **UCADDR** | Bit 1 | Address received in address-bit multiprocessor mode.<br>0 Received character is data<br>1 Received character is an address |
| **UCIDLE** | | Idle line detected in idle-line multiprocessor mode.<br>0 No idle line detected<br>1 Idle line detected |
| **UCBUSY** | Bit 0 | USCI busy. This bit indicates if a transmit or receive operation is in progress.<br>0 USCI inactive<br>1 USCI transmitting or receiving |

### UCAxRXBUF, USCI_Ax Receive Buffer Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **UCRXBUFx** | | | | | | | |
| r | r | r | r | r | r | r | r |

| | | |
|---|---|---|
| **UCRXBUFx** | Bits 7–0 | The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCAxRXBUF resets the receive-error bits, the UCADDR or UCIDLE bit, and UCAxRXIFG. In 7-bit data mode, UCAxRXBUF is LSB justified and the MSB is always reset. |

### UCAxTXBUF, USCI_Ax Transmit Buffer Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **UCTXBUFx** | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

| | | |
|---|---|---|
| **UCTXBUFx** | 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 UCAxTXD. Writing to the transmit data buffer clears UCAxTXIFG. The MSB of UCAxTXBUF is not used for 7-bit data and is reset. |

## UCAxIRTCTL, USCI_Ax IrDA Transmit Control Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCIRTXPLx | | | | | | UCIRTXCLK | UCIREN |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 |

**UCIRTXPLx** Bits 7–2 Transmit pulse length
Pulse Length $t_{PULSE}$ = (UCIRTXPLx + 1) / (2 * $f_{IRTXCLK}$)

**UCIRTXCLK** Bit 1 IrDA transmit pulse clock select
- 0 BRCLK
- 1 BITCLK16 when UCOS16 = 1. Otherwise, BRCLK

**UCIREN** Bit 0 IrDA encoder/decoder enable.
- 0 IrDA encoder/decoder disabled
- 1 IrDA encoder/decoder enabled

## UCAxIRRCTL, USCI_Ax IrDA Receive Control Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCIRRXFLx | | | | | | UCIRRXPL | UCIRRXFE |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 |

**UCIRRXFLx** Bits 7–2 Receive filter length. The minimum pulse length for receive is given by:
$t_{MIN}$ = (UCIRRXFLx + 4) / (2 * $f_{IRTXCLK}$)

**UCIRRXPL** Bit 1 IrDA receive input UCAxRXD polarity
- 0 IrDA transceiver delivers a high pulse when a light pulse is seen
- 1 IrDA transceiver delivers a low pulse when a light pulse is seen

**UCIRRXFE** Bit 0 IrDA receive filter enabled
- 0 Receive filter disabled
- 1 Receive filter enabled

## UCAxABCTL, USCI_Ax Auto Baud Rate Control Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | UCDELIMx | | UCSTOE | UCBTOE | Reserved | UCABDEN |
| r–0 | r–0 | rw–0 | rw–0 | rw–0 | rw–0 | r–0 | rw–0 |

| | | |
|---|---|---|
| **Reserved** | Bits 7-6 | Reserved |
| **UCDELIMx** | Bits 5–4 | Break/synch delimiter length<br>00 1 bit time<br>01 2 bit times<br>10 3 bit times<br>11 4 bit times |
| **UCSTOE** | Bit 3 | Synch field time out error<br>0 No error<br>1 Length of synch field exceeded measurable time. |
| **UCBTOE** | Bit 2 | Break time out error<br>0 No error<br>1 Length of break field exceeded 22 bit times. |
| **Reserved** | Bit 1 | Reserved |
| **UCABDEN** | Bit 0 | Automatic baud rate detect enable<br>0 Baud rate detection disabled. Length of break and synch field is not measured.<br>1 Baud rate detection enabled. Length of break and synch field is measured and baud rate settings are changed accordingly. |

## IE2, Interrupt Enable Register 2

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | | | **UCA0TXIE** | **UCA0RXIE** |
| | | | | | | rw–0 | rw–0 |

| | | |
|---|---|---|
| | Bits 7-2 | These bits may be used by other modules. See device-specific datasheet. |
| **UCA0TXIE** | Bit 1 | USCI_A0 transmit interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **UCA0RXIE** | Bit 0 | USCI_A0 receive interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |

## IFG2, Interrupt Flag Register 2

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | | | **UCA0 TXIFG** | **UCA0 RXIFG** |
| | | | | | | rw–1 | rw–0 |

| | | |
|---|---|---|
| | Bits 7-2 | These bits may be used by other modules. See device-specific datasheet. |
| **UCA0 TXIFG** | Bit 1 | USCI_A0 transmit interrupt flag. UCA0TXIFG is set when UCA0TXBUF is empty.<br>0 No interrupt pending<br>1 Interrupt pending |
| **UCA0 RXIFG** | Bit 0 | USCI_A0 receive interrupt flag. UCA0RXIFG is set when UCA0RXBUF has received a complete character.<br>0 No interrupt pending<br>1 Interrupt pending |

## UC1IE, USCI_A1 Interrupt Enable Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **Unused** | **Unused** | **Unused** | **Unused** | | | **UCA1TXIE** | **UCA1RXIE** |
| rw–0 | rw–0 | rw–0 | rw–0 | | | rw–0 | rw–0 |

| | | |
|---|---|---|
| **Unused** | Bits 7-4 | Unused |
| | Bits 3-2 | These bits may be used by other USCI modules. See device-specific datasheet. |
| **UCA1TXIE** | Bit 1 | USCI_A1 transmit interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **UCA1RXIE** | Bit 0 | USCI_A1 receive interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |

## UC1IFG, USCI_A1 Interrupt Flag Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **Unused** | **Unused** | **Unused** | **Unused** | | | **UCA1 TXIFG** | **UCA1 RXIFG** |
| rw–0 | rw–0 | rw–0 | rw–0 | | | rw–1 | rw–0 |

| | | |
|---|---|---|
| **Unused** | Bits 7-4 | Unused |
| | Bits 3-2 | These bits may be used by other USCI modules. See device-specific datasheet. |
| **UCA1 TXIFG** | Bit 1 | USCI_A1 transmit interrupt flag. UCA1TXIFG is set when UCA1TXBUF is empty.<br>0 No interrupt pending<br>1 Interrupt pending |
| **UCA1 RXIFG** | Bit 0 | USCI_A1 receive interrupt flag. UCA1RXIFG is set when UCA1RXBUF has received a complete character.<br>0 No interrupt pending<br>1 Interrupt pending |

Chapter 12

# Universal Serial Communication Interface, SPI Mode

The universal serial communication interface (USCI) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the synchronous peripheral interface or SPI mode.

**Topic** **Page**

## 12.1 USCI Overview

The universal serial communication interface (USCI) modules support multiple serial communication modes. Different USCI modules support different modes. Each different USCI module is named with a different letter. For example, USCI_A is different from USCI_B, etc. If more than one identical USCI module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two USCI_A modules, they are named USCI_A0 and USCI_A1. See the device-specific datasheet to determine which USCI modules, if any, are implemented on which devices.

The USCI_Ax modules support:

- UART mode
- Pulse shaping for IrDA communications
- Automatic baud rate detection for LIN communications
- SPI mode

The USCI_Bx modules support:

- $I^2C$ mode
- SPI mode

## 12.2 USCI Introduction: SPI Mode

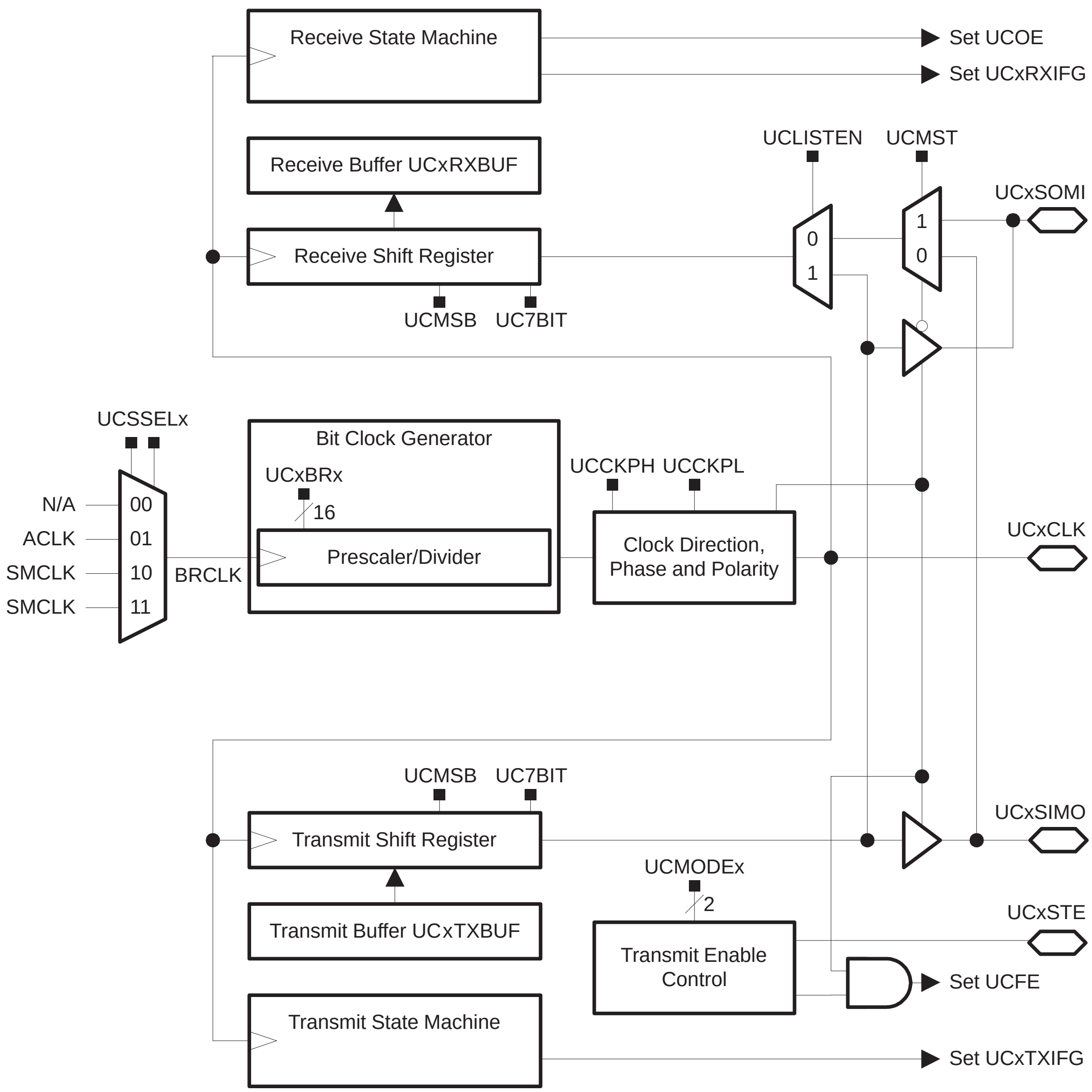
In synchronous mode, the USCI connects the MSP430 to an external system via three or four pins: UCxSIMO, UCxSOMI, UCxCLK, and UCxSTE. SPI mode is selected when the UCSYNC bit is set and SPI mode (3-pin or 4-pin) is selected with the UCMODEx bits.

SPI mode features include:

- 7- or 8-bit data length
- LSB-first or MSB-first data transmit and receive
- 3-pin and 4-pin SPI operation
- Master or slave modes
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- Continuous transmit and receive operation
- Selectable clock polarity and phase control
- Programmable clock frequency in master mode
- Independent interrupt capability for receive and transmit
- Slave operation in LPM4

Figure 12–1 shows the USCI when configured for SPI mode.

*Figure 12–1. USCI Block Diagram: SPI Mode*



## 12.3 USCI 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, UCxSTE, is provided 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:

- UCxSIMO  Slave in, master out
  Master mode: UCxSIMO is the data output line.
  Slave mode: UCxSIMO is the data input line.
- UCxSOMI  Slave out, master in
  Master mode: UCxSOMI is the data input line.
  Slave mode: UCxSOMI is the data output line.
- UCxCLK  USCI SPI clock
  Master mode: UCxCLK is an output.
  Slave mode: UCxCLK is an input.
- UCxSTE  Slave transmit enable. Used in 4-pin mode to allow multiple masters on a single bus. Not used in 3-pin mode. Table 12–1 describes the UCxSTE operation.

*Table 12–1. UCxSTE Operation*

| UCMODEx | UCxSTE Active State | UCxSTE | Slave | Master |
|---|---|---|---|---|
| 01 | high | 0 | inactive | active |
| | | 1 | active | inactive |
| 10 | low | 0 | active | inactive |
| | | 1 | inactive | active |

### 12.3.1 USCI Initialization and Reset

The USCI is reset by a PUC or by the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the USCI in a reset condition. When set, the UCSWRST bit resets the UCxRXIE, UCxTXIE, UCxRXIFG, UCOE, and UCFE bits and sets the UCxTXIFG flag. Clearing UCSWRST releases the USCI for operation.

**Note: Initializing or Re-Configuring the USCI Module**

The recommended USCI initialization/re-configuration process is:

1) Set UCSWRST (`BIS.B  #UCSWRST,&UCxCTL1`)

2) Initialize all USCI registers with UCSWRST=1 (including UCxCTL1)

3) Configure ports.

4) Clear UCSWRST via software (`BIC.B  #UCSWRST,&UCxCTL1`)

5) Enable interrupts (optional) via UCxRXIE and/or UCxTXIE

### 12.3.2 Character Format

The USCI module in SPI mode supports 7- and 8-bit character lengths selected by the UC7BIT bit. In 7-bit data mode, UCxRXBUF is LSB justified and the MSB is always reset. The UCMSB bit controls the direction of the transfer and selects LSB or MSB first.

**Note: Default Character Format**

The default SPI character transmission is LSB first. For communication with other SPI interfaces it MSB-first mode may be required.

**Note: Character Format for Figures**

Figures throughout this chapter use MSB first format.

### 12.3.3 Master Mode

*Figure 12–2. USCI Master and External Slave*

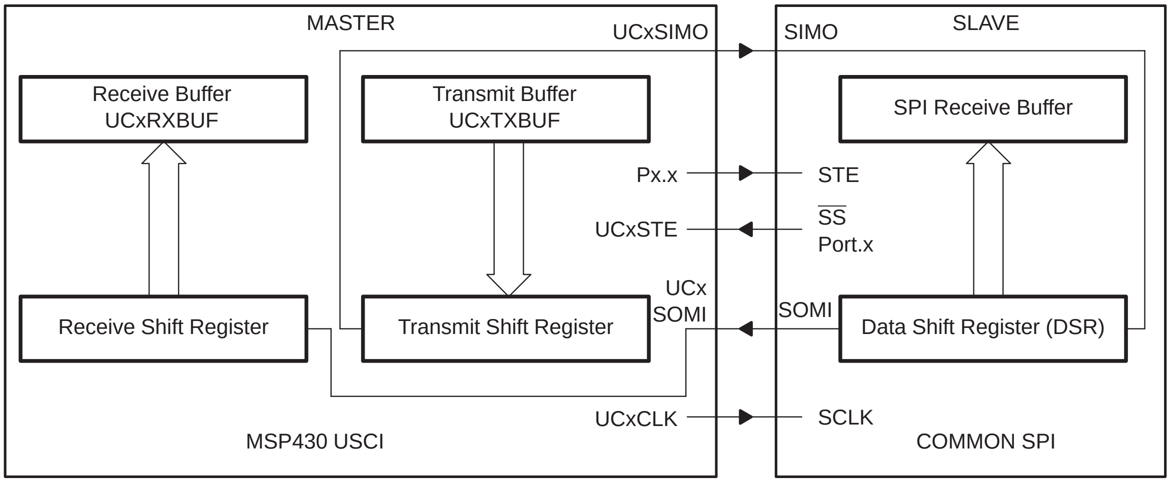



Figure 12–2 shows the USCI as a master in both 3-pin and 4-pin configurations. The USCI initiates data transfer when data is moved to the transmit data buffer UCxTXBUF. The UCxTXBUF data is moved to the TX shift register when the TX shift register is empty, initiating data transfer on UCxSIMO starting with either the most-significant or least-significant bit depending on the UCMSB setting. Data on UCxSOMI is shifted into the receive shift register on the opposite clock edge. When the character is received, the receive data is moved from the RX shift register to the received data buffer UCxRXBUF and the receive interrupt flag, UCxRXIFG, is set, indicating the RX/TX operation is complete.

A set transmit interrupt flag, UCxTXIFG, indicates that data has moved from UCxTXBUF to the TX shift register and UCxTXBUF is ready for new data. It does not indicate RX/TX completion.

To receive data into the USCI in master mode, data must be written to UCxTXBUF because receive and transmit operations operate concurrently.

## Four-Pin SPI Master Mode

In 4-pin master mode, UCxSTE is used to prevent conflicts with another master and controls the master as described in Table 12–1. When UCxSTE is in the master-inactive state:

- UCxSIMO and UCxCLK are set to inputs and no longer drive the bus
- The error bit UCFE is set indicating a communication integrity violation to be handled by the user.
- The internal state machines are reset and the shift operation is aborted.

If data is written into UCxTXBUF while the master is held inactive by UCxSTE, it will be transmit as soon as UCxSTE transitions to the master-active state. If an active transfer is aborted by UCxSTE transitioning to the master-inactive state, the data must be re-written into UCxTXBUF to be transferred when UCxSTE transitions back to the master-active state. The UCxSTE input signal is not used in 3-pin master mode.

### 12.3.4 Slave Mode

*Figure 12–3. USCI Slave and External Master*

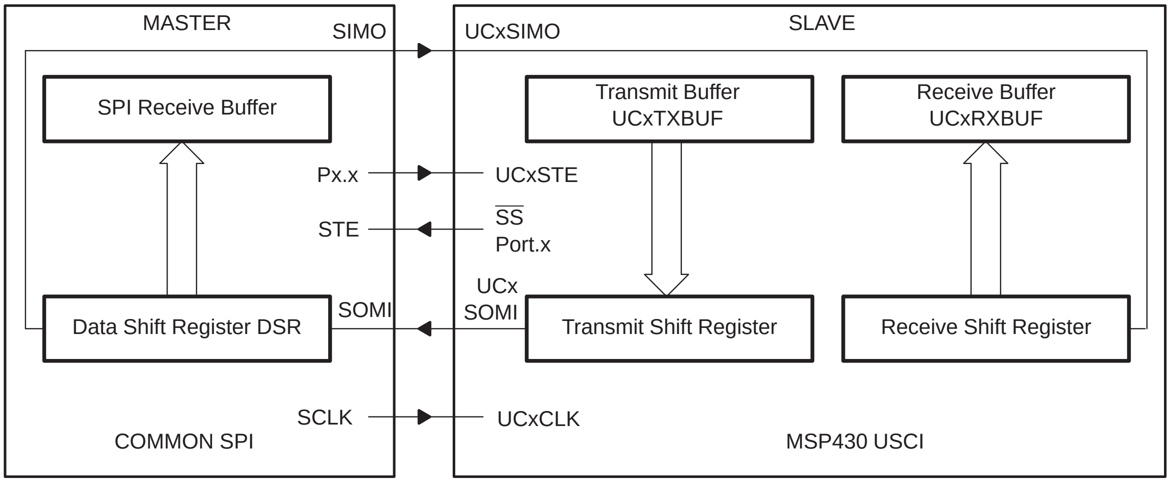



Figure 12–3 shows the USCI as a slave in both 3-pin and 4-pin configurations. UCxCLK is used as the input for the SPI clock and must be supplied by the external master. The data-transfer rate is determined by this clock and not by the internal bit clock generator. Data written to UCxTXBUF and moved to the TX shift register before the start of UCxCLK is transmitted on UCxSOMI. Data on UCxSIMO is shifted into the receive shift register on the opposite edge of UCxCLK and moved to UCxRXBUF when the set number of bits are received. When data is moved from the RX shift register to UCxRXBUF, the UCxRXIFG interrupt flag is set, indicating that data has been received. The overrun error bit, UCOE, is set when the previously received data is not read from UCxRXBUF before new data is moved to UCxRXBUF.

#### Four-Pin SPI Slave Mode

In 4-pin slave mode, UCxSTE is used by the slave to enable the transmit and receive operations and is provided by the SPI master. When UCxSTE is in the slave-active state, the slave operates normally. When UCxSTE is in the slave-inactive state:

- Any receive operation in progress on UCxSIMO is halted
- UCxSOMI is set to the input direction
- The shift operation is halted until the UCxSTE line transitions into the slave transmit active state.

The UCxSTE input signal is not used in 3-pin slave mode.

### 12.3.5 SPI Enable

When the USCI module is enabled by clearing the UCSWRST bit it is ready to receive and transmit. In master mode the bit clock generator is ready, but is not clocked nor producing any clocks. In slave mode the bit clock generator is disabled and the clock is provided by the master.

A transmit or receive operation is indicated by UCBUSY = 1.

A PUC or set UCSWRST bit disables the USCI immediately and any active transfer is terminated.

#### Transmit Enable

In master mode, writing to UCxTXBUF activates the bit clock generator and the data will begin to transmit.

In slave mode, transmission begins when a master provides a clock and, in 4-pin mode, when the UCxSTE is in the slave-active state.

#### Receive Enable

The SPI receives data when a transmission is active. Receive and transmit operations operate concurrently.

### 12.3.6 Serial Clock Control

UCxCLK is provided by the master on the SPI bus. When UCMST = 1, the bit clock is provided by the USCI bit clock generator on the UCxCLK pin. The clock used to generate the bit clock is selected with the UCSSELx bits. When UCMST = 0, the USCI clock is provided on the UCxCLK pin by the master, the bit clock generator is not used, and the UCSSELx bits are don't care. The SPI receiver and transmitter operate in parallel and use the same clock source for data transfer.

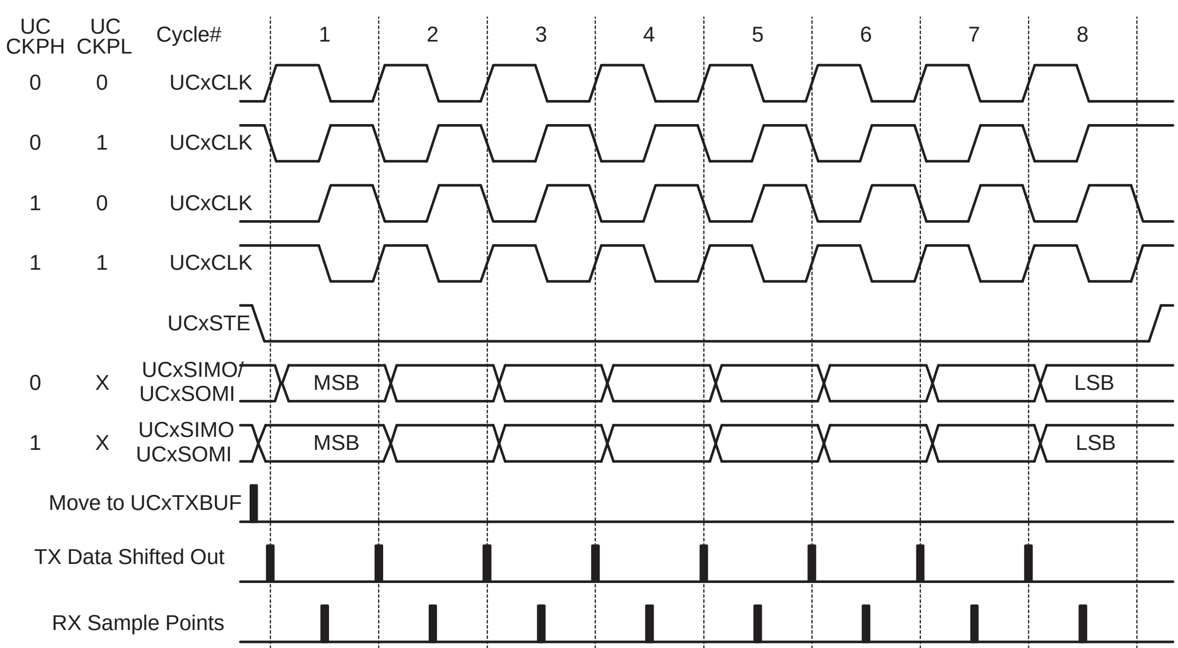
The 16-bit value of UCBRx in the bit rate control registers UCxxBR1 and UCxxBR0 is the division factor of the USCI clock source, BRCLK. The maximum bit clock that can be generated in master mode is BRCLK. Modulation is not used in SPI mode and UCAxMCTL should be cleared when using SPI mode for USCI_A. The UCAxCLK/UCBxCLK frequency is given by:

$$f_{BitClock} = \frac{f_{BRCLK}}{UCBRx}$$

#### Serial Clock Polarity and Phase

The polarity and phase of UCxCLK are independently configured via the UCCKPL and UCCKPH control bits of the USCI. Timing for each case is shown in Figure 12–4.

*Figure 12–4. USCI SPI Timing with UCMSB = 1*



### 12.3.7 Using the SPI Mode with Low Power Modes

The USCI module provides automatic clock activation for SMCLK for use with low-power modes. When SMCLK is the USCI clock source, and is inactive because the device is in a low-power mode, the USCI module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the USCI module returns to its idle condition. After the USCI module returns to the idle condition, control of the clock source reverts to the settings of its control bits. Automatic clock activation is not provided for ACLK.

When the USCI module activates an inactive clock source, the clock source becomes active for the whole device and any peripheral configured to use the clock source may be affected. For example, a timer using SMCLK will increment while the USCI module forces SMCLK active.

In SPI slave mode no internal clock source is required because the clock is provided by the external master. It is possible to operate the USCI in SPI slave mode while the device is in LPM4 and all clock sources are disabled. The receive or transmit interrupt can wake up the CPU from any low power mode.

### 12.3.8 SPI Interrupts

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

#### SPI Transmit Interrupt Operation

The UCxTXIFG interrupt flag is set by the transmitter to indicate that UCxTXBUF is ready to accept another character. An interrupt request is generated if UCxTXIE and GIE are also set. UCxTXIFG is automatically reset if a character is written to UCxTXBUF. UCxTXIFG is set after a PUC or when UCSWRST = 1. UCxTXIE is reset after a PUC or when UCSWRST = 1.

**Note: Writing to UCxTXBUF in SPI Mode**

Data written to UCxTXBUF when UCxTXIFG = 0 may result in erroneous data transmission.

#### SPI Receive Interrupt Operation

The UCxRXIFG interrupt flag is set each time a character is received and loaded into UCxRXBUF. An interrupt request is generated if UCxRXIE and GIE are also set. UCxRXIFG and UCxRXIE are reset by a system reset PUC signal or when UCSWRST = 1. UCxRXIFG is automatically reset when UCxRXBUF is read.

## USCI Interrupt Usage

USCI_Ax and USCI_Bx share the same interrupt vectors. The receive interrupt flags UCAxRXIFG and UCBxRXIFG are routed to one interrupt vector, the transmit interrupt flags UCAxTXIFG and UCBxTXIFG share another interrupt vector.

### *Shared Interrupt Vectors Software Example*

The following software example shows an extract of an interrupt service routine to handle data receive interrupts from USCI_A0 in either UART or SPI mode and USCI_B0 in SPI mode.

```
USCIA0_RX_USCIB0_RX_ISR
   BIT.B  #UCA0RXIFG, &IFG2  ; USCI_A0 Receive Interrupt?
   JNZ    USCIA0_RX_ISR
USCIB0_RX_ISR?
   ; Read UCB0RXBUF (clears UCB0RXIFG)
   ...
   RETI
USCIA0_RX_ISR
   ; Read UCA0RXBUF (clears UCA0RXIFG)
   ...
   RETI
```

The following software example shows an extract of an interrupt service routine to handle data transmit interrupts from USCI_A0 in either UART or SPI mode and USCI_B0 in SPI mode.

```
USCIA0_TX_USCIB0_TX_ISR
   BIT.B  #UCA0TXIFG, &IFG2  ; USCI_A0 Transmit Interrupt?
   JNZ    USCIA0_TX_ISR
USCIB0_TX_ISR
   ; Write UCB0TXBUF (clears UCB0TXIFG)
   ...
   RETI
USCIA0_TX_ISR
   ; Write UCA0TXBUF  (clears UCA0TXIFG)
   ...
   RETI
```

## 12.4 USCI Registers: SPI Mode

The USCI registers applicable in SPI mode for USCI_A0 and USCI_B0 are listed in Table 12–2. Registers applicable in SPI mode for USCI_A1 and USCI_B1 are listed in Table 12–3.

*Table 12–2. USCI_A0 and USCI_B0 Control and Status Registers*

| Register | Short Form | Register Type | Address | Initial State |
|---|---|---|---|---|
| USCI_A0 control register 0 | UCA0CTL0 | Read/write | 060h | Reset with PUC |
| USCI_A0 control register 1 | UCA0CTL1 | Read/write | 061h | 001h with PUC |
| USCI_A0 Baud rate control register 0 | UCA0BR0 | Read/write | 062h | Reset with PUC |
| USCI_A0 Baud rate control register 1 | UCA0BR1 | Read/write | 063h | Reset with PUC |
| USCI_A0 modulation control register | UCA0MCTL | Read/write | 064h | Reset with PUC |
| USCI_A0 status register | UCA0STAT | Read/write | 065h | Reset with PUC |
| USCI_A0 Receive buffer register | UCA0RXBUF | Read | 066h | Reset with PUC |
| USCI_A0 Transmit buffer register | UCA0TXBUF | Read/write | 067h | Reset with PUC |
| USCI_B0 control register 0 | UCB0CTL0 | Read/write | 068h | 001h with PUC |
| USCI_B0 control register 1 | UCB0CTL1 | Read/write | 069h | 001h with PUC |
| USCI_B0 Bit rate control register 0 | UCB0BR0 | Read/write | 06Ah | Reset with PUC |
| USCI_B0 Bit rate control register 1 | UCB0BR1 | Read/write | 06Bh | Reset with PUC |
| USCI_B0 status register | UCB0STAT | Read/write | 06Dh | Reset with PUC |
| USCI_B0 Receive buffer register | UCB0RXBUF | Read | 06Eh | Reset with PUC |
| USCI_B0 Transmit buffer register | UCB0TXBUF | Read/write | 06Fh | Reset with PUC |
| SFR interrupt enable register 2 | IE2 | Read/write | 001h | Reset with PUC |
| SFR interrupt flag register 2 | IFG2 | Read/write | 003h | 00Ah with 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.

*Table 12–3. USCI_A1 and USCI_B1 Control and Status Registers*

| Register | Short Form | Register Type | Address | Initial State |
|---|---|---|---|---|
| USCI_A1 control register 0 | UCA1CTL0 | Read/write | 0D0h | Reset with PUC |
| USCI_A1 control register 1 | UCA1CTL1 | Read/write | 0D1h | 001h with PUC |
| USCI_A1 Baud rate control register 0 | UCA1BR0 | Read/write | 0D2h | Reset with PUC |
| USCI_A1 Baud rate control register 1 | UCA1BR1 | Read/write | 0D3h | Reset with PUC |
| USCI_A1 modulation control register | UCA10MCTL | Read/write | 0D4h | Reset with PUC |
| USCI_A1 status register | UCA1STAT | Read/write | 0D5h | Reset with PUC |
| USCI_A1 Receive buffer register | UCA1RXBUF | Read | 0D6h | Reset with PUC |
| USCI_A1 Transmit buffer register | UCA1TXBUF | Read/write | 0D7h | Reset with PUC |
| USCI_B1 control register 0 | UCB1CTL0 | Read/write | 0D8h | 001h with PUC |
| USCI_B1 control register 1 | UCB1CTL1 | Read/write | 0D9h | 001h with PUC |
| USCI_B1 Bit rate control register 0 | UCB1BR0 | Read/write | 0DAh | Reset with PUC |
| USCI_B1 Bit rate control register 1 | UCB1BR1 | Read/write | 0DBh | Reset with PUC |
| USCI_B1 status register | UCB1STAT | Read/write | 0DDh | Reset with PUC |
| USCI_B1 Receive buffer register | UCB1RXBUF | Read | 0DEh | Reset with PUC |
| USCI_B1 Transmit buffer register | UCB1TXBUF | Read/write | 0DFh | Reset with PUC |
| USCI_A1/B1 interrupt enable register | UC1IE | Read/write | 006h | Reset with PUC |
| USCI_A1/B1 interrupt flag register | UC1IFG | Read/write | 007h | 00Ah with PUC |

## UCAxCTL0, USCI_Ax Control Register 0
## UCBxCTL0, USCI_Bx Control Register 0

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCCKPH | UCCKPL | UCMSB | UC7BIT | UCMST | UCMODEx | | UCSYNC=1 |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | |

**UCCKPH** Bit 7 Clock phase select.
- 0 Data is changed on the first UCLK edge and captured on the following edge.
- 1 Data is captured on the first UCLK edge and changed on the following edge.

**UCCKPL** Bit 6 Clock polarity select.
- 0 The inactive state is low.
- 1 The inactive state is high.

**UCMSB** Bit 5 MSB first select. Controls the direction of the receive and transmit shift register.
- 0 LSB first
- 1 MSB first

**UC7BIT** Bit 4 Character length. Selects 7-bit or 8-bit character length.
- 0 8-bit data
- 1 7-bit data

**UCMST** Bit 3 Master mode select
- 0 Slave mode
- 1 Master mode

**UCMODEx** Bits 2-1 USCI Mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1.
- 00 3-Pin SPI
- 01 4-Pin SPI with UCxSTE active high: slave enabled when UCxSTE = 1
- 10 4-Pin SPI with UCxSTE active low: slave enabled when UCxSTE = 0
- 11 $I^2C$ Mode

**UCSYNC** Bit 0 Synchronous mode enable
- 0 Asynchronous mode
- 1 Synchronous Mode

## UCAxCTL1, USCI_Ax Control Register 1
## UCBxCTL1, USCI_Bx Control Register 1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **UCSSELx** | | **Unused** | | | | | **UCSWRST** |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-1 |

| | | |
|---|---|---|
| **UCSSELx** | Bits 7-6 | USCI clock source select. These bits select the BRCLK source clock in master mode. UCxCLK is always used in slave mode.<br>00 NA<br>01 ACLK<br>10 SMCLK<br>11 SMCLK |
| **Unused** | Bits 5-1 | Unused |
| **UCSWRST** | Bit 0 | Software reset enable<br>0 Disabled. USCI reset released for operation.<br>1 Enabled. USCI logic held in reset state. |

## UCAxBR0, USCI_Ax Bit Rate Control Register 0
## UCBxBR1, USCI_Bx Bit Rate Control Register 0

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **UCBRx** | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

## UCAxBR1, USCI_Ax Bit Rate Control Register 1
## UCBxBR1, USCI_Bx Bit Rate Control Register 1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **UCBRx** | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

| | |
|---|---|
| **UCBRx** | Bit clock prescaler. The 16-bit value of {UCxxBR0+UCxxBR1} form the prescaler value. |

## UCAxSTAT, USCI_Ax Status Register
## UCBxSTAT, USCI_Bx Status Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **UCLISTEN** | **UCFE** | **UCOE** | **Unused** | **Unused** | **Unused** | **Unused** | **UCBUSY** |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | r-0 |

| | | |
|---|---|---|
| **UCLISTEN** | Bit 7 | Listen enable. The UCLISTEN bit selects loopback mode.<br>0 Disabled<br>1 Enabled. The transmitter output is internally fed back to the receiver. |
| **UCFE** | Bit 6 | Framing error flag. This bit indicates a bus conflict in 4-wire master mode. UCFE is not used in 3-wire master or any slave mode.<br>0 No error<br>1 Bus conflict occurred |
| **UCOE** | Bit 5 | Overrun error flag. This bit is set when a character is transferred into UCxRXBUF before the previous character was read. UCOE is cleared automatically when UCxRXBUF is read, and must not be cleared by software. Otherwise, it will not function correctly.<br>0 No error<br>1 Overrun error occurred |
| **Unused** | Bits 4–1 | Unused |
| **UCBUSY** | Bit 0 | USCI busy. This bit indicates if a transmit or receive operation is in progress.<br>0 USCI inactive<br>1 USCI transmitting or receiving |

## UCAxRXBUF, USCI_Ax Receive Buffer Register
## UCBxRXBUF, USCI_Bx Receive Buffer Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **UCRXBUFx** | | | | | | | |
| r | r | r | r | r | r | r | r |

| | | |
|---|---|---|
| **UCRXBUFx** | Bits 7-0 | The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCxRXBUF resets the receive-error bits, and UCxRXIFG. In 7-bit data mode, UCxRXBUF is LSB justified and the MSB is always reset. |

## UCAxTXBUF, USCI_Ax Transmit Buffer Register
## UCBxTXBUF, USCI_Bx Transmit Buffer Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **UCTXBUFx** | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

| | | |
|---|---|---|
| **UCTXBUFx** | 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. Writing to the transmit data buffer clears UCxTXIFG. The MSB of UCxTXBUF is not used for 7-bit data and is reset. |

## IE2, Interrupt Enable Register 2

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | **UCB0TXIE** | **UCB0RXIE** | **UCA0TXIE** | **UCA0RXIE** |
| | | | | rw-0 | rw-0 | rw-0 | rw-0 |

| | | |
|---|---|---|
| | Bits 7-4 | These bits may be used by other modules. See device-specific datasheet. |
| **UCB0TXIE** | Bit 3 | USCI_B0 transmit interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **UCB0RXIE** | Bit 2 | USCI_B0 receive interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **UCA0TXIE** | Bit 1 | USCI_A0 transmit interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **UCA0RXIE** | Bit 0 | USCI_A0 receive interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |

## IFG2, Interrupt Flag Register 2

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | **UCB0 TXIFG** | **UCB0 RXIFG** | **UCA0 TXIFG** | **UCA0 RXIFG** |
| | | | | rw-1 | rw-0 | rw-1 | rw-0 |

| | | |
|---|---|---|
| | Bits 7-4 | These bits may be used by other modules. See device-specific datasheet. |
| **UCB0 TXIFG** | Bit 3 | USCI_B0 transmit interrupt flag. UCB0TXIFG is set when UCB0TXBUF is empty.<br>0 No interrupt pending<br>1 Interrupt pending |
| **UCB0 RXIFG** | Bit 2 | USCI_B0 receive interrupt flag. UCB0RXIFG is set when UCB0RXBUF has received a complete character.<br>0 No interrupt pending<br>1 Interrupt pending |
| **UCA0 TXIFG** | Bit 1 | USCI_A0 transmit interrupt flag. UCA0TXIFG is set when UCA0TXBUF empty.<br>0 No interrupt pending<br>1 Interrupt pending |
| **UCA0 RXIFG** | Bit 0 | USCI_A0 receive interrupt flag. UCA0RXIFG is set when UCA0RXBUF has received a complete character.<br>0 No interrupt pending<br>1 Interrupt pending |

## UC1IE, USCI_A1/USCI_B1 Interrupt Enable Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **Unused** | **Unused** | **Unused** | **Unused** | **UCB1TXIE** | **UCB1RXIE** | **UCA1TXIE** | **UCA1RXIE** |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 |

| | | |
|---|---|---|
| **Unused** | Bits 7-4 | Unused |
| **UCB1TXIE** | Bit 3 | USCI_B1 transmit interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **UCB1RXIE** | Bit 2 | USCI_B1 receive interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **UCA1TXIE** | Bit 1 | USCI_A1 transmit interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **UCA1RXIE** | Bit 0 | USCI_A1 receive interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |

## UC1IFG, USCI_A1/USCI_B1 Interrupt Flag Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **Unused** | **Unused** | **Unused** | **Unused** | **UCB1 TXIFG** | **UCB1 RXIFG** | **UCA1 TXIFG** | **UCA1 RXIFG** |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–1 | rw–0 | rw–1 | rw–0 |

| | | |
|---|---|---|
| **Unused** | Bits 7-4 | Unused |
| **UCB1 TXIFG** | Bit 3 | USCI_B1 transmit interrupt flag. UCB1TXIFG is set when UCB1TXBUF is empty.<br>0 No interrupt pending<br>1 Interrupt pending |
| **UCB1 RXIFG** | Bit 2 | USCI_B1 receive interrupt flag. UCB1RXIFG is set when UCB1RXBUF has received a complete character.<br>0 No interrupt pending<br>1 Interrupt pending |
| **UCA1 TXIFG** | Bit 1 | USCI_A1 transmit interrupt flag. UCA1TXIFG is set when UCA1TXBUF empty.<br>0 No interrupt pending<br>1 Interrupt pending |
| **UCA1 RXIFG** | Bit 0 | USCI_A1 receive interrupt flag. UCA1RXIFG is set when UCA1RXBUF has received a complete character.<br>0 No interrupt pending<br>1 Interrupt pending |

Chapter 13

# Universal Serial Communication Interface, $I^2C$ Mode

The universal serial communication interface (USCI) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the $I^2C$ mode.

**Topic** **Page**

## 13.1 USCI Overview

The universal serial communication interface (USCI) modules support multiple serial communication modes. Different USCI modules support different modes. Each different USCI module is named with a different letter. For example, USCI_A is different from USCI_B, etc. If more than one identical USCI module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two USCI_A modules, they are named USCI_A0 and USCI_A1. See the device-specific datasheet to determine which USCI modules, if any, are implemented on which devices.

The USCI_Ax modules support:

- UART mode
- Pulse shaping for IrDA communications
- Automatic baud rate detection for LIN communications
- SPI mode

The USCI_Bx modules support:

- $I^2C$ mode
- SPI mode

## 13.2 USCI Introduction: I2C Mode

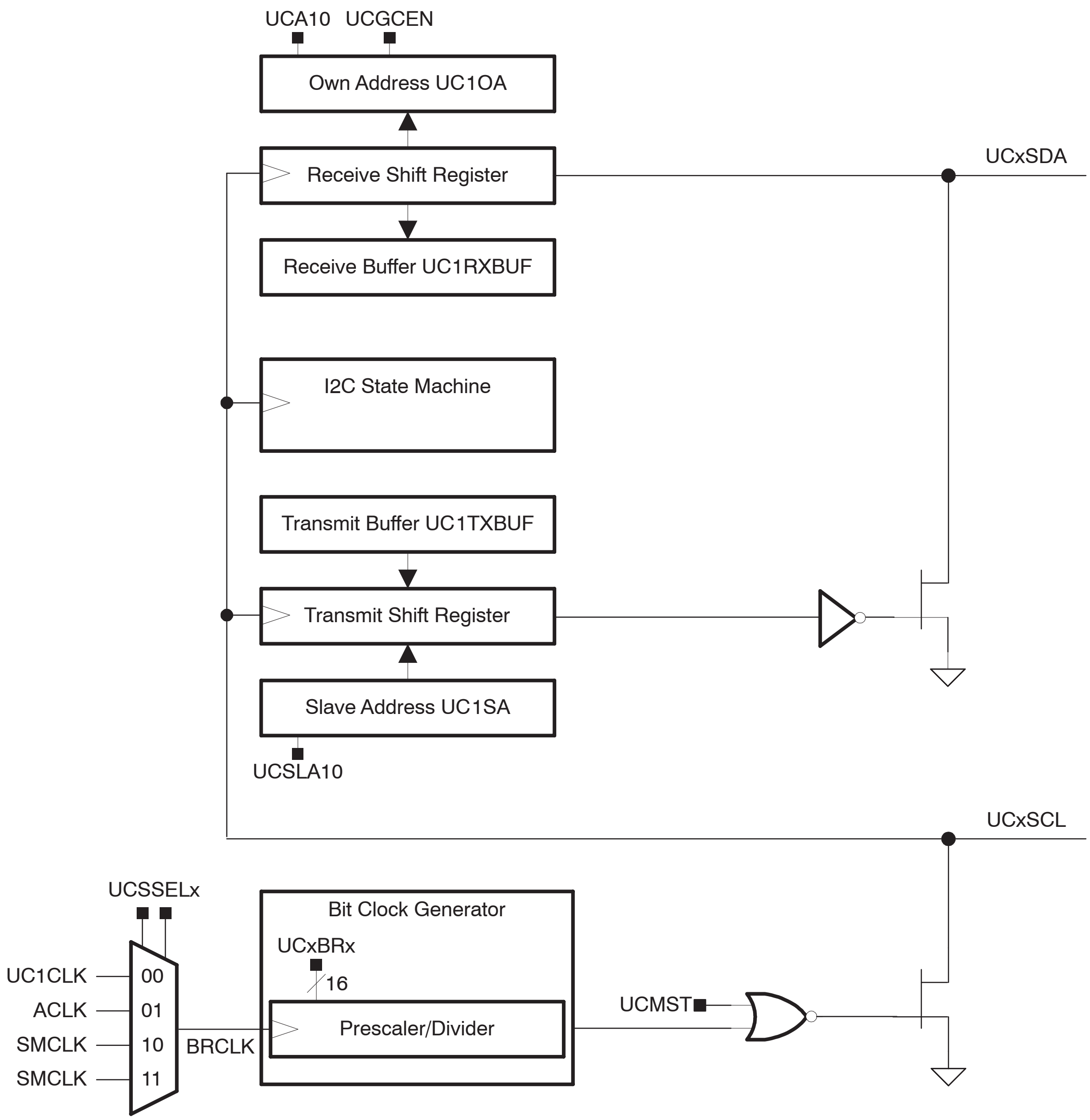
In $I^2C$ mode, the USCI module provides an interface between the MSP430 and $I^2C$-compatible devices connected by way of the two-wire $I^2C$ serial bus. External components attached to the $I^2C$ bus serially transmit and/or receive serial data to/from the USCI module through the 2-wire $I^2C$ interface.

The $I^2C$ mode features include:

- Compliance to the Philips Semiconductor $I^2C$ specification v2.1
  - 7-bit and 10-bit device addressing modes
  - General call
  - START/RESTART/STOP
  - Multi-master transmitter/receiver mode
  - Slave receiver/transmitter mode
  - Standard mode up to100 kbps and fast mode up to 400 kbps support
- Programmable UCxCLK frequency in master mode
- Designed for low power
- Slave receiver START detection for auto-wake up from LPMx modes
- Slave operation in LPM4

Figure 13−1 shows the USCI when configured in $I^2C$ mode.

*Figure 13−1. USCI Block Diagram: I$^2$C Mode*

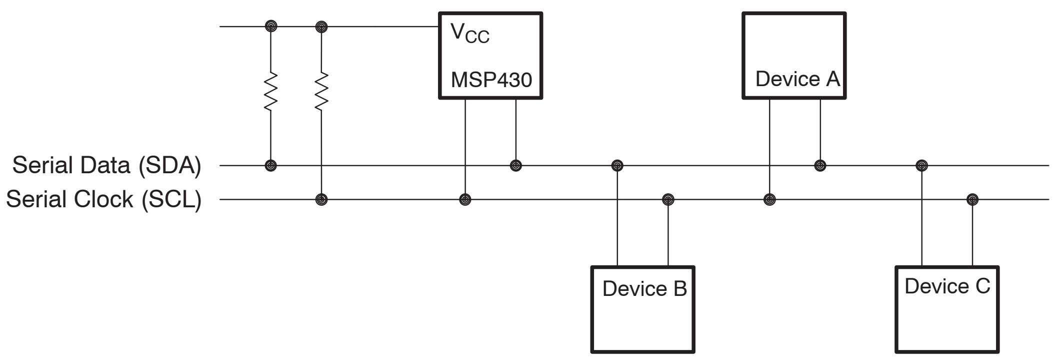


## 13.3 USCI Operation: I2C Mode

The I2C mode supports any slave or master I2C-compatible device. Figure 13−2 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.

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.

*Figure 13−2. I2C Bus Connection Diagram*




**Note: SDA and SCL Levels**

The MSP430 SDA and SCL pins must not be pulled up above the MSP430 $V_{CC}$ level.

### 13.3.1 USCI Initialization and Reset

The USCI is reset by a PUC or by setting the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the USCI in a reset condition. To select $I^2C$ operation the UCMODEx bits must be set to 11. After module initialization, it is ready for transmit or receive operation. Clearing UCSWRST releases the USCI for operation.

Configuring and re-configuring the USCI module should be done when UCSWRST is set to avoid unpredictable behavior. Setting UCSWRST in $I^2C$ mode has the following effects:

- $I^2C$ communication stops
- SDA and SCL are high impedance
- UCBxI2CSTAT, bits 6-0 are cleared
- UCBxTXIE and UCBxRXIE are cleared
- UCBxTXIFG and UCBxRXIFG are cleared
- All other bits and registers remain unchanged.

**Note: Initializing or Re-Configuring the USCI Module**

The recommended USCI initialization/re-configuration process is:

1) Set UCSWRST (`BIS.B   #UCSWRST,&UCxCTL1`)

2) Initialize all USCI registers with UCSWRST=1 (including UCxCTL1)

3) Configure ports.

4) Clear UCSWRST via software (`BIC.B   #UCSWRST,&UCxCTL1`)

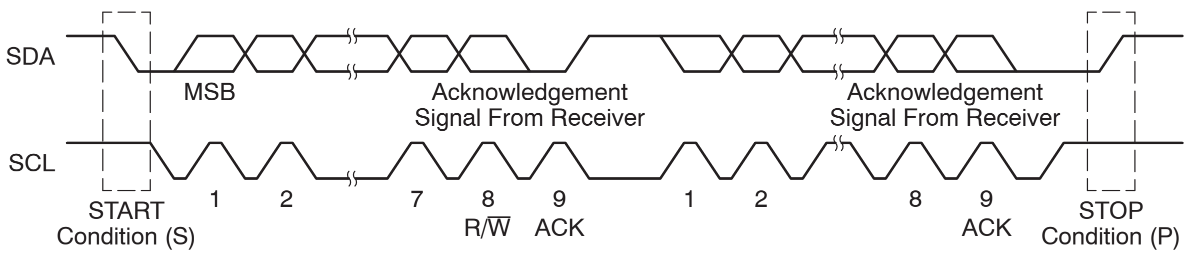
5) Enable interrupts (optional) via UCxRXIE and/or UCxTXIE

### 13.3.2 I2C Serial Data

One clock pulse is generated by the master device for each data bit transferred. The I2C mode operates with byte data. Data is transferred most significant bit first as shown in Figure 13−3.

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.

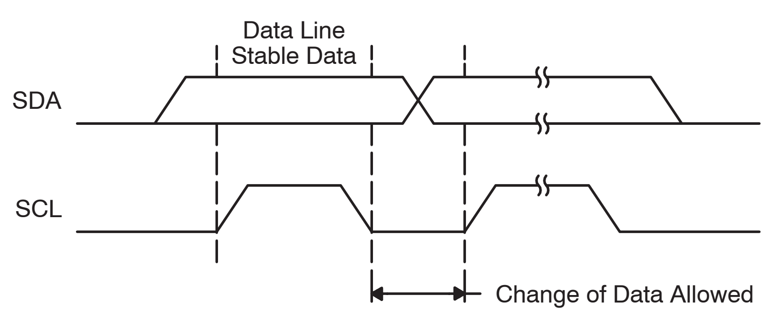
*Figure 13−3. I2C Module Data Transfer*




START and STOP conditions are generated by the master and are shown in Figure 13−3. 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 bus busy bit, UCBBUSY, is set after a START and cleared after a STOP.

Data on SDA must be stable during the high period of SCL as shown in Figure 13−4. The high and low state of SDA can only change when SCL is low, otherwise START or STOP conditions will be generated.

*Figure 13−4. Bit Transfer on the I2C Bus*



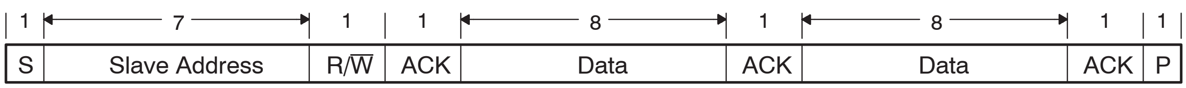
### 13.3.3 I$^2$C Addressing Modes

The I$^2$C mode supports 7-bit and 10-bit addressing modes.

#### 7-Bit Addressing

In the 7-bit addressing format, shown in Figure 13−5, 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.

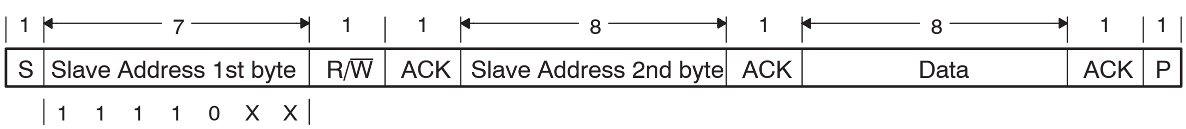
*Figure 13−5. I$^2$C Module 7-Bit Addressing Format*




#### 10-Bit Addressing

In the 10-bit addressing format, shown in Figure 13−6, 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.

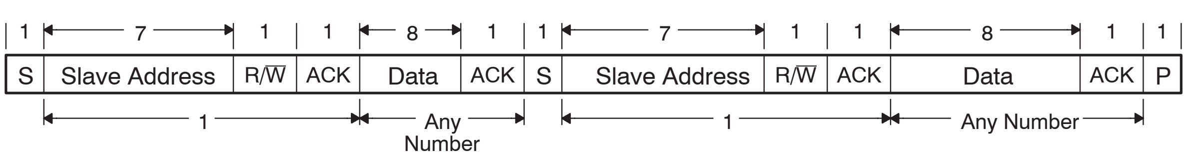
*Figure 13−6. I$^2$C Module 10-Bit Addressing Format*




#### 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 13−7.

*Figure 13−7. I$^2$C Module Addressing Format with Repeated START Condition*



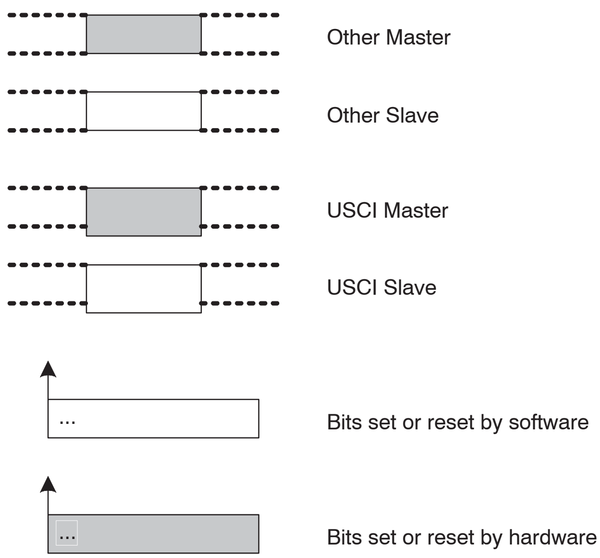
### 13.3.4 I$^2$C Module Operating Modes

In I$^2$C mode the USCI module can operate in master transmitter, master receiver, slave transmitter, or slave receiver mode. The modes are discussed in the following sections. Time lines are used to illustrate the modes.

Figure 13−8 shows how to interpret the time line figures. Data transmitted by the master is represented by grey rectangles, data transmitted by the slave by white rectangles. Data transmitted by the USCI module, either as master or slave, is shown by rectangles that are taller than the others.

Actions taken by the USCI module are shown in grey rectangles with an arrow indicating where in the the data stream the action occurs. Actions that must be handled with software are indicated with white rectangles with an arrow pointing to where in the data stream the action must take place.

*Figure 13−8. I$^2$C Time line Legend*



## Slave Mode

The USCI module is configured as an I2C slave by selecting the I2C mode with UCMODEx = 11 and UCSYNC = 1 and clearing the UCMST bit.

Initially the USCI module must to be configured in receiver mode by clearing the UCTR bit to receive the I2C address. Afterwards, transmit and receive operations are controlled automatically depending on the R/W bit received together with the slave address.

The USCI slave address is programmed with the UCBxI2COA register. When UCA10 = 0, 7-bit addressing is selected. When UCA10 = 1, 10-bit addressing is selected. The UCGCEN bit selects if the slave responds to a general call.

When a START condition is detected on the bus, the USCI module will receive the transmitted address and compare it against its own address stored in UCBxI2COA. The UCSTTIFG flag is set when address received matches the USCI slave address.

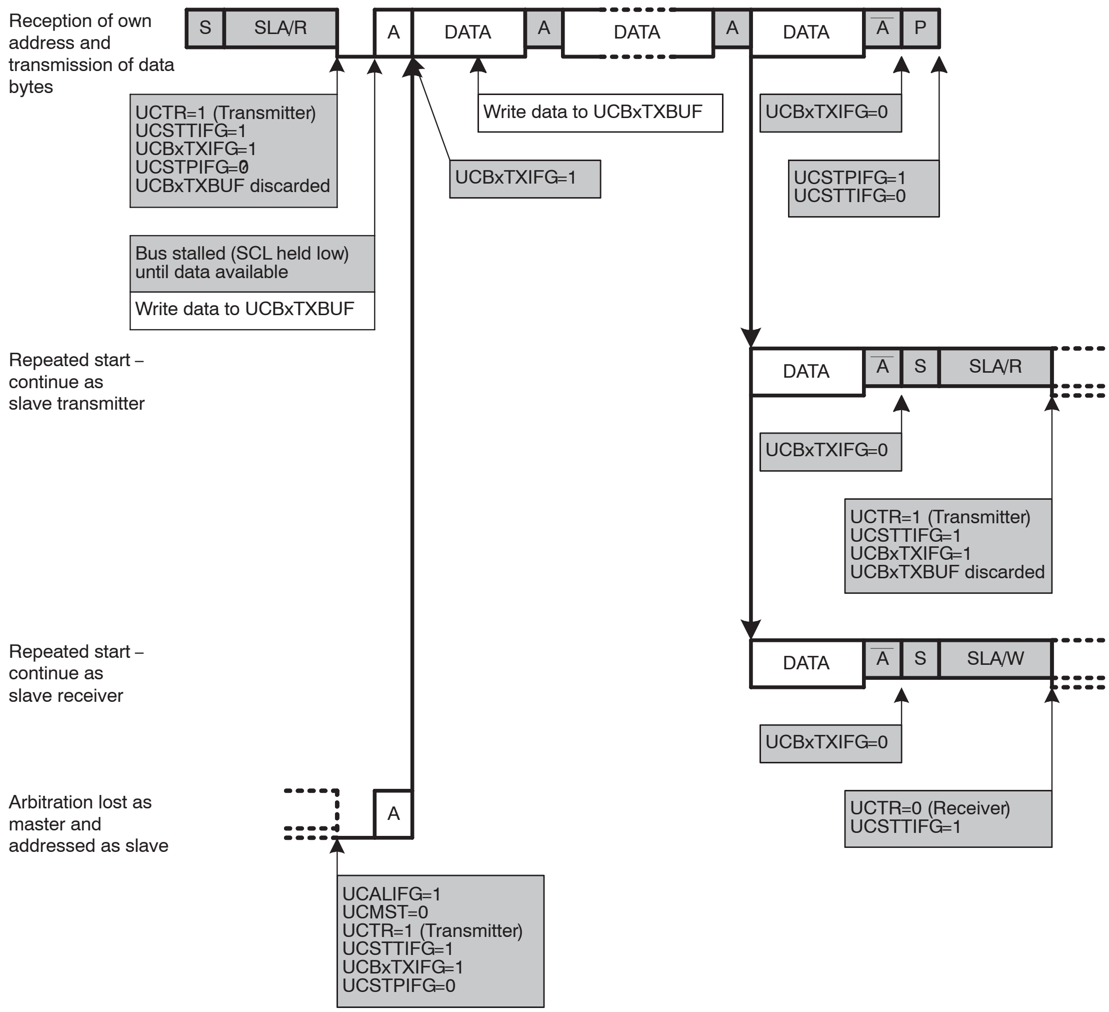
### *I2C Slave Transmitter Mode*

Slave transmitter mode is entered when the slave address transmitted by the master is identical to its own address with a set R/$\overline{W}$ bit. 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 will hold SCL low while intervention of the CPU is required after a byte has been transmitted.

If the master requests data from the slave the USCI module is automatically configured as a transmitter and UCTR and UCBxTXIFG become set. The SCL line is held low until the first data to be sent is written into the transmit buffer UCBxTXBUF. Then the address is acknowledged, the UCSTTIFG flag is cleared, and the data is transmitted. As soon as the data is transferred into the shift register the UCBxTXIFG is set again. After the data is acknowledged by the master the next data byte written into UCBxTXBUF is transmitted or if the buffer is empty the bus is stalled during the acknowledge cycle by holding SCL low until new data is written into UCBxTXBUF. If the master sends a NACK succeeded by a STOP condition the UCSTPIFG flag is set. If the NACK is succeeded by a repeated START condition the USCI I2C state machine returns to its address-reception state.

Figure 13−9 illustrates the slave transmitter operation.

*Figure 13−9. I2C Slave Transmitter Mode*



### *I2C Slave Receiver Mode*

Slave receiver mode is entered when the slave address transmitted by the master is identical to its own address and a cleared R/$\overline{W}$ bit is received. 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.

If the slave should receive data from the master the USCI module is automatically configured as a receiver and UCTR is cleared. After the first data byte is received the receive interrupt flag UCBxRXIFG is set. The USCI module automatically acknowledges the received data and can receive the next data byte.

If the previous data wasn not read from the receive buffer UCBxRXBUF at the end of a reception, the bus is stalled by holding SCL low. As soon as UCBxRXBUF is read the new data is transferred into UCBxRXBUF, an acknowledge is sent to the master, and the next data can be received.

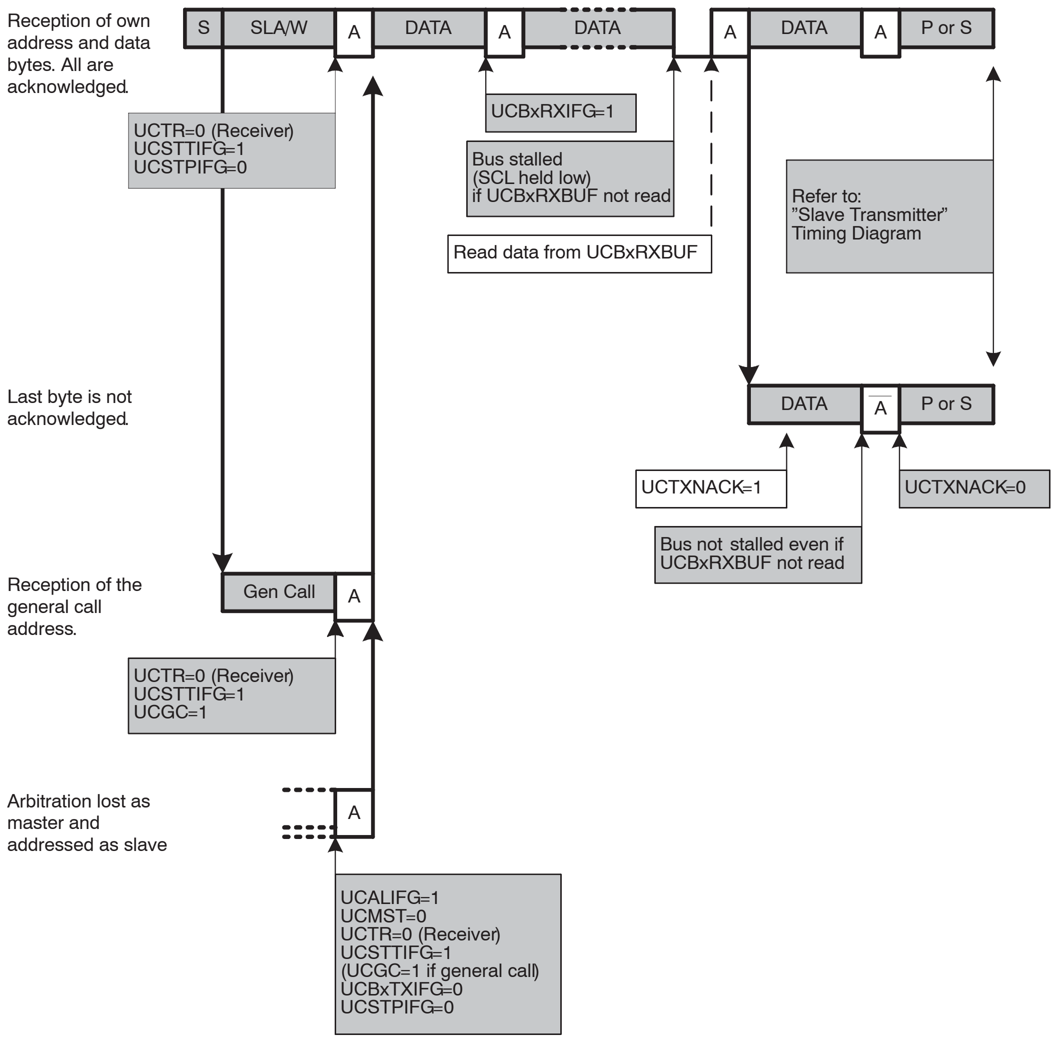
Setting the UCTXNACK bit causes a NACK to be transmitted to the master during the next acknowledgment cycle. A NACK is sent even if UCBxRXBUF is not ready to receive the latest data. If the UCTXNACK bit is set while SCL is held low the bus will be released, a NACK is transmitted immediately, and UCBxRXBUF is loaded with the last received data. Since the previous data was not read that data will be lost. To avoid loss of data the UCBxRXBUF needs to be read before UCTXNACK is set.

When the master generates a STOP condition the UCSTPIFG flag is set.

If the master generates a repeated START condition the USCI $I^2C$ state machine returns to its address reception state.

Figure 13−10 illustrates the the $I^2C$ slave receiver operation.

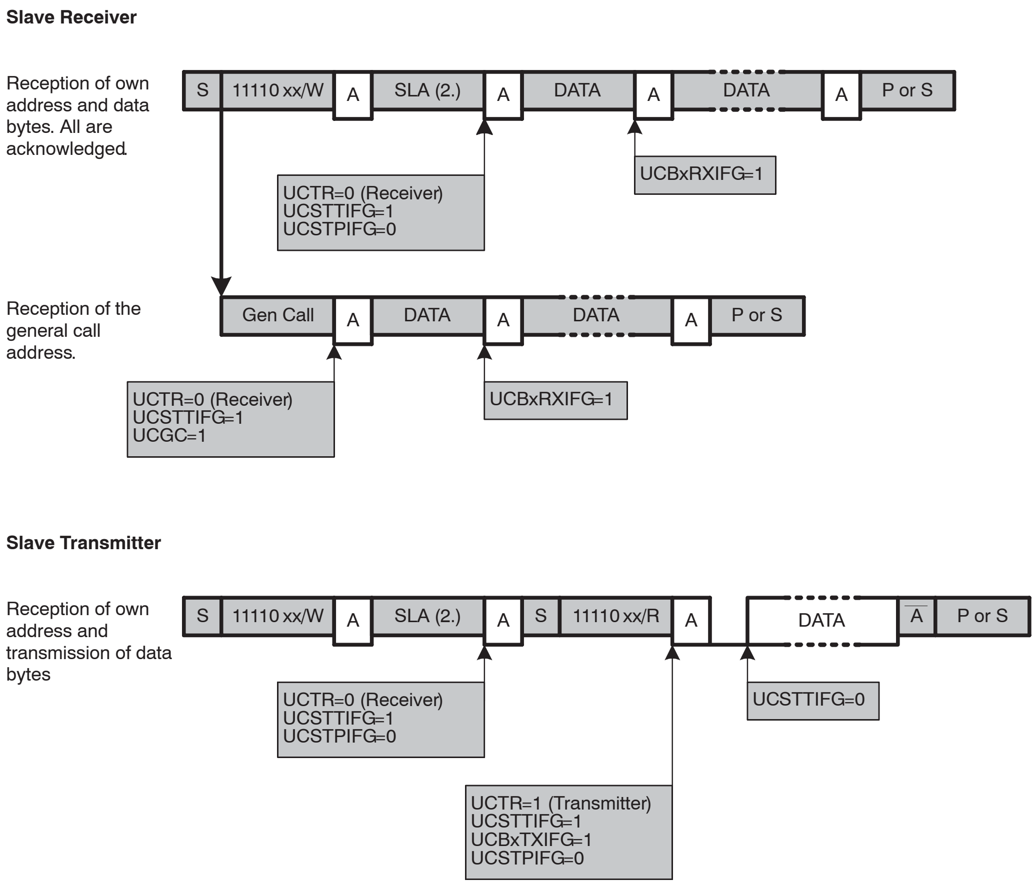
*Figure 13−10. I2C Slave Receiver Mode*



## I2C Slave 10-bit Addressing Mode

The 10-bit addressing mode is selected when UCA10 = 1 and is as shown in Figure 13−11. In 10-bit addressing mode, the slave is in receive mode after the full address is received. The USCI module indicates this by setting the UCSTTIFG flag while the UCTR bit is cleared. To switch the slave into transmitter mode the master sends a repeated START condition together with the first byte of the address but with the R/W bit set. This will set the UCSTTIFG flag if it was previously cleared by software and the USCI modules switches to transmitter mode with UCTR = 1.

*Figure 13−11. I2C Slave 10-bit Addressing Mode*



## Master Mode

The USCI module is configured as an $I^2C$ master by selecting the $I^2C$ mode with UCMODEx = 11 and UCSYNC = 1 and setting the UCMST bit. When the master is part of a multi-master system, UCMM must be set and its own address must be programmed into the UCBxI2COA register. When UCA10 = 0, 7-bit addressing is selected. When UCA10 = 1, 10-bit addressing is selected. The UCGCEN bit selects if the USCI module responds to a general call.

### *$I^2C$ Master Transmitter Mode*

After initialization, master transmitter mode is initiated by writing the desired slave address to the UCBxI2CSA register, selecting the size of the slave address with the UCSLA10 bit, setting UCTR for transmitter mode, and setting UCTXSTT to generate a START condition.

The USCI module checks if the bus is available, generates the START condition, and transmits the slave address. The UCBxTXIFG bit is set when the START condition is generated and the first data to be transmitted can be written into UCBxTXBUF. As soon as the slave acknowledges the address the UCTXSTT bit is cleared.

The data written into UCBxTXBUF is transmitted if arbitration is not lost during transmission of the slave address. UCBxTXIFG is set again as soon as the data is transferred from the buffer into the shift register. If there is no data loaded to UCBxTXBUF before the acknowledge cycle, the bus is held during the acknowledge cycle with SCL low until data is written into UCBxTXBUF. Data is transmitted or the bus is held as long as the UCTXSTP bit or UCTXSTT bit is not set.

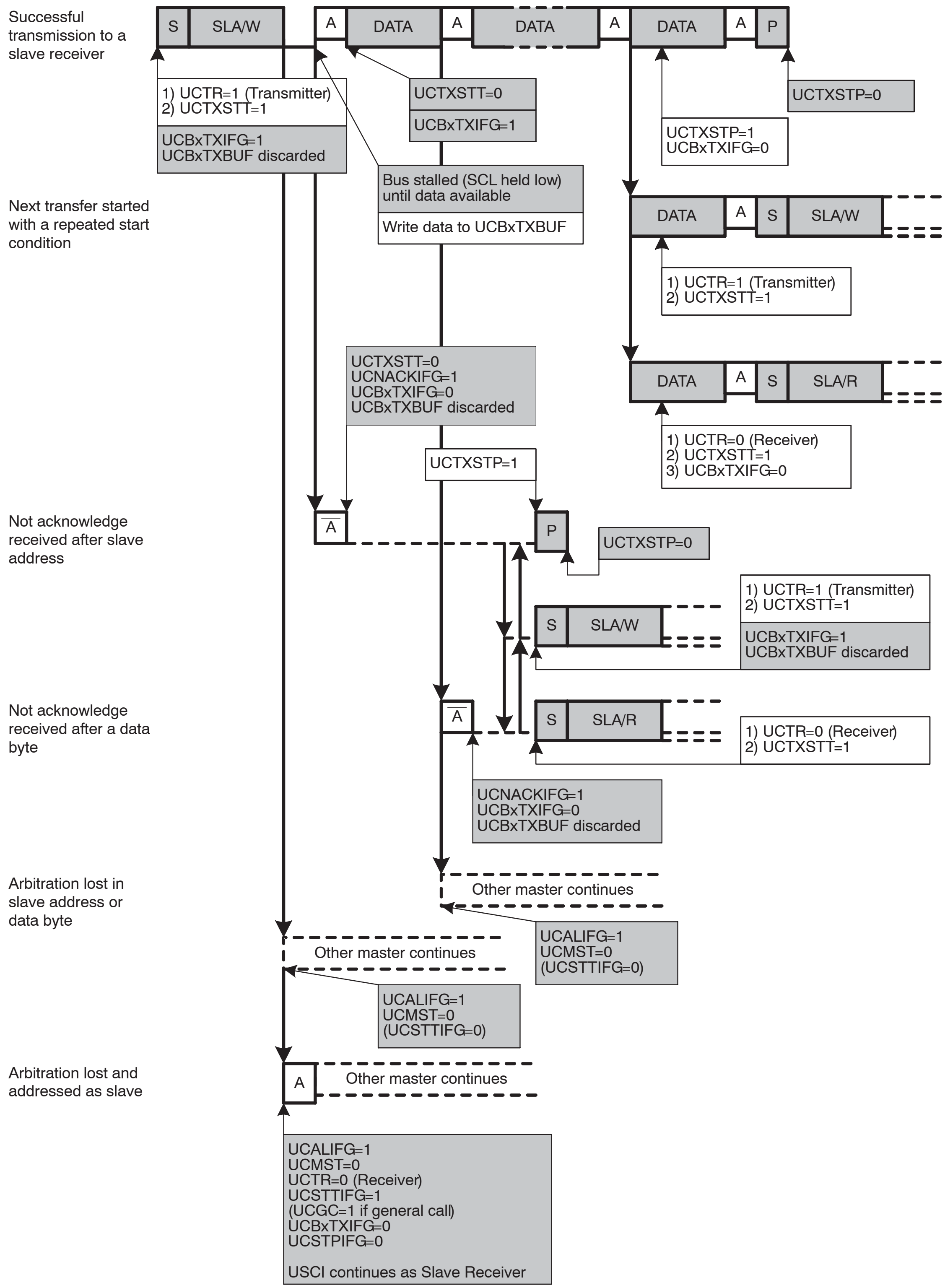
Setting UCTXSTP will generate a STOP condition after the next acknowledge from the slave. If UCTXSTP is set during the transmission of the slave's address or while the USCI module waits for data to be written into UCBxTXBUF, a STOP condition is generated even if no data was transmitted to the slave. When transmitting a single byte of data, the UCTXSTP bit must be set while the byte is being transmitted, or anytime after transmission begins, without writing new data into UCBxTXBUF. Otherwise, only the address will be transmitted. When the data is transferred from the buffer to the shift register, UCBxTXIFG will become set indicating data transmission has begun and the UCTXSTP bit may be set.

Setting UCTXSTT will generate a repeated START condition. In this case, UCTR may be set or cleared to configure transmitter or receiver, and a different slave address may be written into UCBxI2CSA if desired.

If the slave does not acknowledge the transmitted data the not-acknowledge interrupt flag UCNACKIFG is set. The master must react with either a STOP condition or a repeated START condition. If data was already written into UCBxTXBUF it will be discarded. If this data should be transmitted after a repeated START it must be written into UCBxTXBUF again.

Figure 13−12 illustrates the I2C master transmitter operation.

*Figure 13−12. I2C Master Transmitter Mode*



## $I^2C$ Master Receiver Mode

After initialization, master receiver mode is initiated by writing the desired slave address to the UCBxI2CSA register, selecting the size of the slave address with the UCSLA10 bit, clearing UCTR for receiver mode, and setting UCTXSTT to generate a START condition.

The USCI module checks if the bus is available, generates the START condition, and transmits the slave address. As soon as the slave acknowledges the address the UCTXSTT bit is cleared.

After the acknowledge of the address from the slave the first data byte from the slave is received and acknowledged and the UCBxRXIFG flag is set. Data is received from the slave ss long as UCTXSTP or UCTXSTT is not set. If UCBxRXBUF is not read the master holds the bus during reception of the last data bit and until the UCBxRXBUF is read.

If the slave does not acknowledge the transmitted address the not-acknowledge interrupt flag UCNACKIFG is set. The master must react with either a STOP condition or a repeated START condition.

Setting the UCTXSTP bit will generate a STOP condition. After setting UCTXSTP, a NACK followed by a STOP condition is generated after reception of the data from the slave, or immediately if the USCI module is currently waiting for UCBxRXBUF to be read.

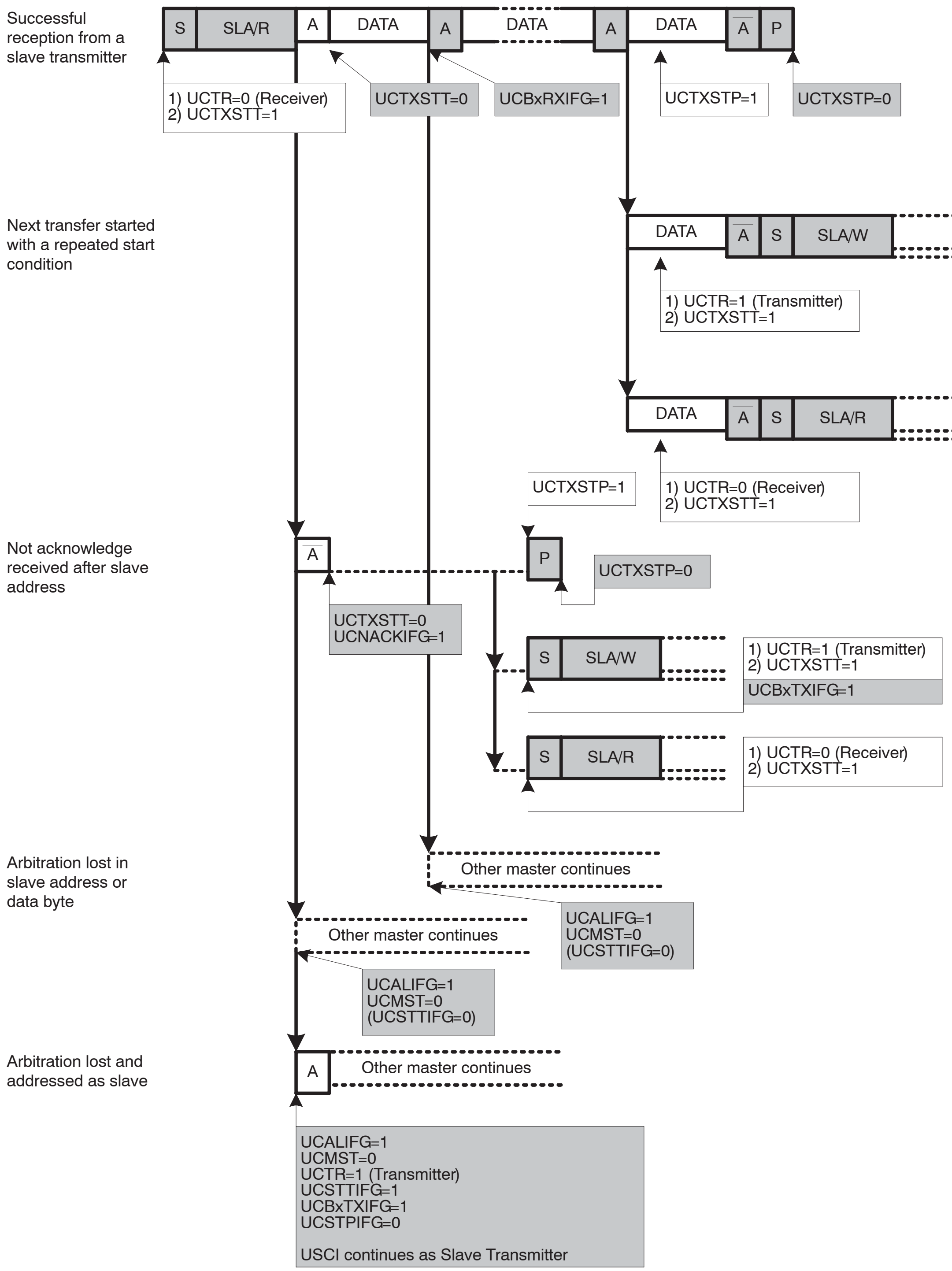
If a master wants to receive a single byte only, the UCTXSTP bit must be set while the byte is being received. For this case, the UCTXSTT may be polled to determine when it is cleared:

```
          BIS.B  #UCTXSTT,&UCBOCTL1 ;Transmit START cond.
POLL_STT  BIT.B  #UCTXSTT,&UCBOCTL1 ;Poll UCTXSTT bit
          JC     POLL_STT           ;When cleared,
          BIS.B  #UCTXSTP,&UCB0CTL1 ;transmit STOP cond.
```

Setting UCTXSTT will generate a repeated START condition. In this case, UCTR may be set or cleared to configure transmitter or receiver, and a different slave address may be written into UCBxI2CSA if desired.

Figure 13−13 illustrates the $I^2C$ master receiver operation.

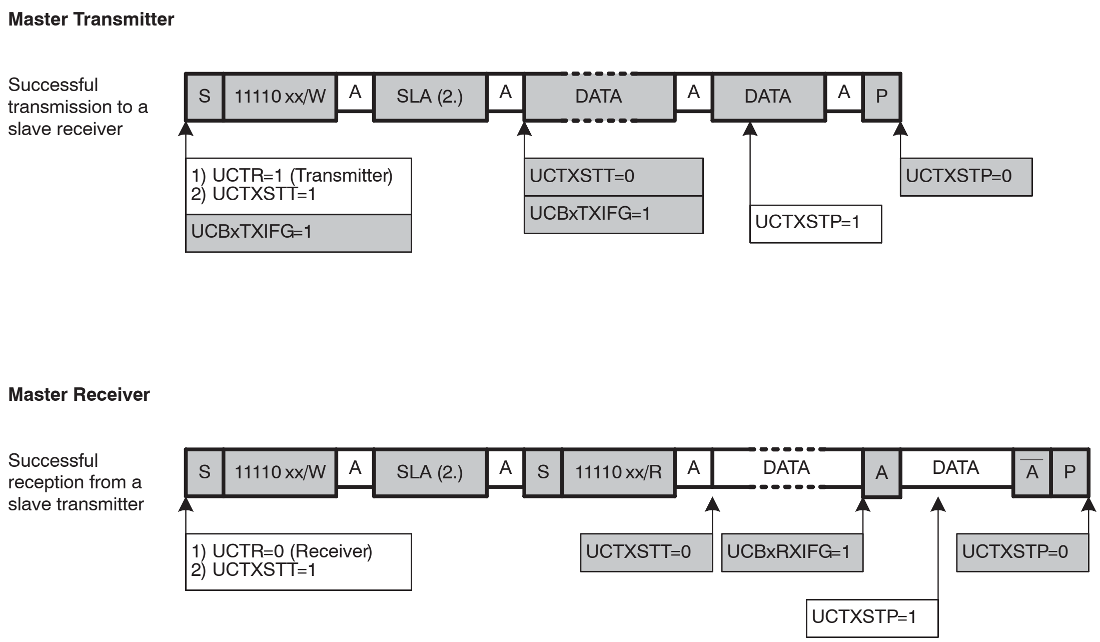
*Figure 13−13. I2C Master Receiver Mode*



## I2C Master 10-bit Addressing Mode

The 10-bit addressing mode is selected when UCSLA10 = 1 and is shown in Figure 13−14.

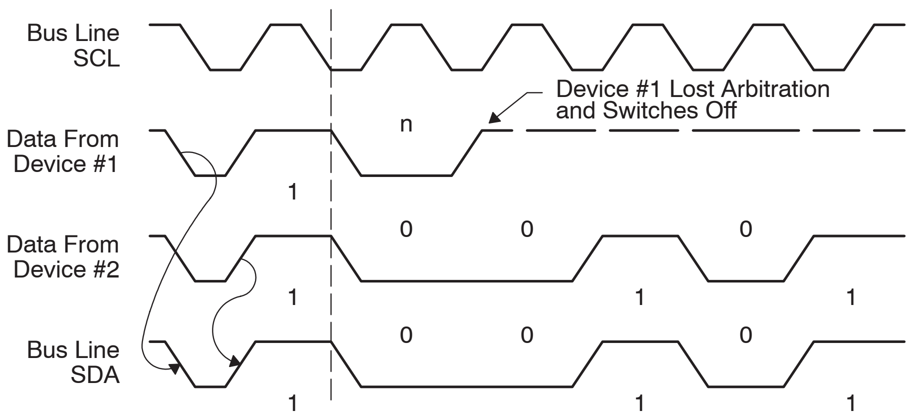
*Figure 13−14. I2C Master 10-bit Addressing Mode*



### Arbitration

If two or more master transmitters simultaneously start a transmission on the bus, an arbitration procedure is invoked. Figure 13−15 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 UCALIFG. If two or more devices send identical first bytes, arbitration continues on the subsequent bytes.

*Figure 13−15. 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

### 13.3.5 I2C Clock Generation and Synchronization

The I2C clock SCL is provided by the master on the I2C bus. When the USCI is in master mode, BITCLK is provided by the USCI bit clock generator and the clock source is selected with the UCSSELx bits. In slave mode the bit clock generator is not used and the UCSSELx bits are don't care.

The 16-bit value of UCBRx in registers UCBxBR1 and UCBxBR0 is the division factor of the USCI clock source, BRCLK. The maximum bit clock that can be used in single master mode is $f_{BRCLK}/4$. In multi-master mode the maximum bit clock is $f_{BRCLK}/8$. The BITCLK frequency is given by:

$$f_{BitClock} = \frac{f_{BRCLK}}{UCBRx}$$

The minimum high and low periods of the generated SCL are

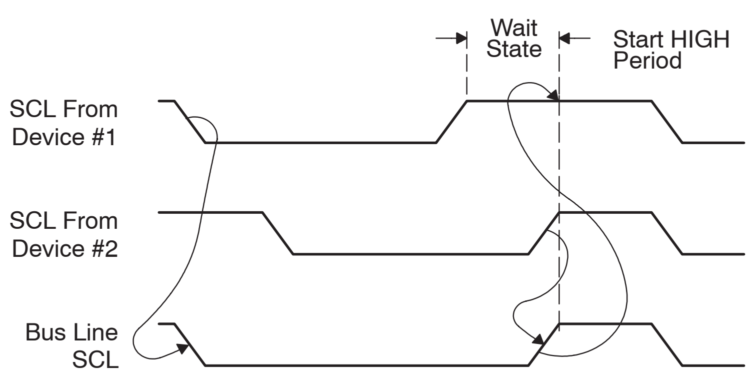
$$t_{LOW,MIN} = t_{HIGH,MIN} = \frac{UCBRx/2}{f_{BRCLK}} \quad \text{when UCBRx is even and}$$

$$t_{LOW,MIN} = t_{HIGH,MIN} = \frac{UCBRx/2}{f_{BRCLK}} \quad \text{when UCBRx is odd.}$$

The USCI clock source frequency and the prescaler setting UCBRx must to be chosen such that the minimum low and high period times of the I2C specification are met.

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 13−16 illustrates the clock synchronization. This allows a slow slave to slow down a fast master.

*Figure 13−16. Synchronization of Two I2C Clock Generators During Arbitration*



### 13.3.6 Using the USCI Module in I2C Mode with Low Power Modes

The USCI module provides automatic clock activation for SMCLK for use with low-power modes. When SMCLK is the USCI clock source, and is inactive because the device is in a low-power mode, the USCI module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the USCI module returns to its idle condition. After the USCI module returns to the idle condition, control of the clock source reverts to the settings of its control bits. Automatic clock activation is not provided for ACLK.

When the USCI module activates an inactive clock source, the clock source becomes active for the whole device and any peripheral configured to use the clock source may be affected. For example, a timer using SMCLK will increment while the USCI module forces SMCLK active.

In I2C slave mode no internal clock source is required because the clock is provided by the external master. It is possible to operate the USCI in I2C slave mode while the device is in LPM4 and all internal clock sources are disabled. The receive or transmit interrupts can wake up the CPU from any low power mode.

## 13.3.7 USCI Interrupts in I2C Mode

Their are two interrupt vectors for the USCI module in I2C mode. One interrupt vector is associated with the transmit and receive interrupt flags. The other interrupt vector is associated with the four state change 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. DMA transfers are controlled by the UCBxTXIFG and UCBxRXIFG flags on devices with a DMA controller.

### I2C Transmit Interrupt Operation

The UCBxTXIFG interrupt flag is set by the transmitter to indicate that UCBxTXBUF is ready to accept another character. An interrupt request is generated if UCBxTXIE and GIE are also set. UCBxTXIFG is automatically reset if a character is written to UCBxTXBUF or if a NACK is received. UCBxTXIFG is set when UCSWRST = 1 and the I2C mode is selected. UCBxTXIE is reset after a PUC or when UCSWRST = 1.

### I2C Receive Interrupt Operation

The UCBxRXIFG interrupt flag is set when a character is received and loaded into UCBxRXBUF. An interrupt request is generated if UCBxRXIE and GIE are also set. UCBxRXIFG and UCBxRXIE are reset after a PUC signal or when UCSWRST = 1. UCxRXIFG is automatically reset when UCxRXBUF is read.

### I2C State Change Interrupt Operation.

Table 13−1 Describes the I2C state change interrupt flags.

*Table 13−1. I2C State Change Interrupt Flags*

| Interrupt Flag | Interrupt Condition |
|---|---|
| UCALIFG | Arbitration-lost. Arbitration can be lost when two or more transmitters start a transmission simultaneously, or when the USCI operates as master but is addressed as a slave by another master in the system. The UCALIFG flag is set when arbitration is lost. When UCALIFG is set the UCMST bit is cleared and the I2C controller becomes a slave. |
| UCNACKIFG | Not-acknowledge interrupt. This flag is set when an acknowledge is expected but is not received. UCNACKIFG is automatically cleared when a START condition is received. |
| UCSTTIFG | Start condition detected interrupt. This flag is set when the I2C module detects a START condition together with its own address while in slave mode. UCSTTIFG is used in slave mode only and is automatically cleared when a STOP condition is received. |
| UCSTPIFG | Stop condition detected interrupt. This flag is set when the I2C module detects a STOP condition while in slave mode. UCSTPIFG is used in slave mode only and is automatically cleared when a START condition is received. |

## Interrupt Vector Assignment

USCI_Ax and USCI_Bx share the same interrupt vectors. In I2C mode the state change interrupt flags UCSTTIFG, UCSTPIFG, UCIFG, UCALIFG from USCI_Bx and UCAxRXIFG from USCI_Ax are routed to one interrupt vector. The I2C transmit and receive interrupt flags UCBxTXIFG and UCBxRXIFG from USCI_Bx and UCAxTXIFG from USCI_Ax share another interrupt vector.

### *Shared Interrupt Vectors Software Example*

The following software example shows an extract of the interrupt service routine to handle data receive interrupts from USCI_A0 in either UART or SPI mode and state change interrupts from USCI_B0 in I2C mode.

```
USCIA0_RX_USCIB0_I2C_STATE_ISR
   BIT.B  #UCA0RXIFG, &IFG2  ; USCI_A0 Receive Interrupt?
   JNZ    USCIA0_RX_ISR
USCIB0_I2C_STATE_ISR
   ; Decode I2C state changes ...
   ; Decode I2C state changes ...
   ...
   RETI
USCIA0_RX_ISR
   ; Read UCA0RXBUF ... - clears UCA0RXIFG
   ...
   RETI
```

The following software example shows an extract of the interrupt service routine that handles data transmit interrupts from USCI_A0 in either UART or SPI mode and the data transfer interrupts from USCI_B0 in I2C mode.

```
USCIA0_TX_USCIB0_I2C_DATA_ISR
   BIT.B  #UCA0TXIFG, &IFG2  ; USCI_A0 Transmit Interrupt?
   JNZ    USCIA0_TX_ISR
USCIB0_I2C_DATA_ISR
   BIT.B  #UCB0RXIFG, &IFG2
   JNZ    USCIB0_I2C_RX
USCIB0_I2C_TX
   ; Write UCB0TXBUF... - clears UCB0TXIFG
   ...
   RETI
USCIB0_I2C_RX
   ; Read UCB0RXBUF... - clears UCB0RXIFG
   ...
   RETI
USCIA0_TX_ISR
   ; Write UCA0TXBUF ... - clears UCA0TXIFG
   ...
   RETI
```

## 13.4 USCI Registers: I$^2$C Mode

The USCI registers applicable in I$^2$C mode for USCI_B0 are listed in Table 13−2 and for USCI_B1 in Table 13−3.

*Table 13−2.USCI_B0 Control and Status Registers*

| Register | Short Form | Register Type | Address | Initial State |
|---|---|---|---|---|
| USCI_B0 control register 0 | UCB0CTL0 | Read/write | 068h | 001h with PUC |
| USCI_B0 control register 1 | UCB0CTL1 | Read/write | 069h | 001h with PUC |
| USCI_B0 Bit rate control register 0 | UCB0BR0 | Read/write | 06Ah | Reset with PUC |
| USCI_B0 Bit rate control register 1 | UCB0BR1 | Read/write | 06Bh | Reset with PUC |
| USCI_B0 I2C Interrupt enable register | UCB0I2CIE | Read/write | 06Ch | Reset with PUC |
| USCI_B0 status register | UCB0STAT | Read/write | 06Dh | Reset with PUC |
| USCI_B0 Receive buffer register | UCB0RXBUF | Read | 06Eh | Reset with PUC |
| USCI_B0 Transmit buffer register | UCB0TXBUF | Read/write | 06Fh | Reset with PUC |
| USCI_B0 I2C Own Address register | UCB0I2COA | Read/write | 0118h | Reset with PUC |
| USCI_B0 I2C Slave Address register | UCB0I2CSA | Read/write | 011Ah | Reset with PUC |
| SFR interrupt enable register 2 | IE2 | Read/write | 001h | Reset with PUC |
| SFR interrupt flag register 2 | IFG2 | Read/write | 003h | 00Ah with 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.

*Table 13−3.USCI_B1 Control and Status Registers*

| Register | Short Form | Register Type | Address | Initial State |
|---|---|---|---|---|
| USCI_B1 control register 0 | UCB1CTL0 | Read/write | 0D8h | Reset with PUC |
| USCI_B1 control register 1 | UCB1CTL1 | Read/write | 0D9h | 001h with PUC |
| USCI_B1 Baud rate control register 0 | UCB1BR0 | Read/write | 0DAh | Reset with PUC |
| USCI_B1 Baud rate control register 1 | UCB1BR1 | Read/write | 0DBh | Reset with PUC |
| USCI_B1 I2C Interrupt enable register | UCB1I2CIE | Read/write | 0DCh | Reset with PUC |
| USCI_B1 status register | UCB1STAT | Read/write | 0DDh | Reset with PUC |
| USCI_B1 Receive buffer register | UCB1RXBUF | Read | 0DEh | Reset with PUC |
| USCI_B1 Transmit buffer register | UCB1TXBUF | Read/write | 0DFh | Reset with PUC |
| USCI_B1 I2C Own Address register | UCB1I2COA | Read/write | 017Ch | Reset with PUC |
| USCI_B1 I2C Slave Address register | UCB1I2CSA | Read/write | 017Eh | Reset with PUC |
| USCI_A1/B1 interrupt enable register | UC1IE | Read/write | 006h | Reset with PUC |
| USCI_A1/B1 interrupt flag register | UC1IFG | Read/write | 007h | 00Ah with PUC |

## UCBxCTL0, USCI_Bx Control Register 0

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCA10 | UCSLA10 | UCMM | Unused | UCMST | UCMODEx=11 | | UCSYNC=1 |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | r–1 |

| | | |
|---|---|---|
| **UCA10** | Bit 7 | Own addressing mode select<br>0 Own address is a 7-bit address<br>1 Own address is a 10-bit address |
| **UCSLA10** | Bit 6 | Slave addressing mode select<br>0 Address slave with 7-bit address<br>1 Address slave with 10-bit address |
| **UCMM** | Bit 5 | Multi-master environment select<br>0 Single master environment. There is no other master in the system. The address compare unit is disabled.<br>1 Multi master environment |
| **Unused** | Bit 4 | Unused |
| **UCMST** | Bit 3 | Master mode select. When a master looses arbitration in a multi-master environment (UCMM = 1) the UCMST bit is automatically cleared and the module acts as slave.<br>0 Slave mode<br>1 Master mode |
| **UCMODEx** | Bits 2–1 | USCI Mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1.<br>00 3–Pin SPI<br>01 4–Pin SPI (master/slave enabled if STE = 1)<br>10 4–Pin SPI (master/slave enabled if STE = 0)<br>11 I²C Mode |
| **UCSYNC** | Bit 0 | Synchronous mode enable<br>0 Asynchronous mode<br>1 Synchronous Mode |

## UCBxCTL1, USCI_Bx Control Register 1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCSSELx | | Unused | UCTR | UCTXNACK | UCTXSTP | UCTXSTT | UCSWRST |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–1 |

| | | |
|---|---|---|
| **UCSSELx** | Bits 7-6 | USCI clock source select. These bits select the BRCLK source clock.<br>00 UCLKI<br>01 ACLK<br>10 SMCLK<br>11 SMCLK |
| **Unused** | Bit 5 | Unused |
| **UCTR** | Bit 4 | Transmitter/Receiver<br>0 Receiver<br>1 Transmitter |
| **UCTXNACK** | Bit 3 | Transmit a NACK. UCTXNACK is automatically cleared after a NACK is transmitted.<br>0 Acknowledge normally<br>1 Generate NACK |
| **UCTXSTP** | Bit 2 | Transmit STOP condition in master mode. Ignored in slave mode. In master receiver mode the STOP condition is preceded by a NACK. UCTXSTP is automatically cleared after STOP is generated.<br>0 No STOP generated<br>1 Generate STOP |
| **UCTXSTT** | Bit 1 | Transmit START condition in master mode. Ignored in slave mode. In master receiver mode a repeated START condition is preceded by a NACK. UCTXSTT is automatically cleared after START condition and address information is transmitted.<br>Ignored in slave mode.<br>0 Do not generate START condition<br>1 Generate START condition |
| **UCSWRST** | Bit 0 | Software reset enable<br>0 Disabled. USCI reset released for operation.<br>1 Enabled. USCI logic held in reset state. |

## UCBxBR0, USCI_Bx Baud Rate Control Register 0

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCBRx | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

## UCBxBR1, USCI_Bx Baud Rate Control Register 1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCBRx | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

**UCBRx** Bit clock prescaler. The 16-bit value of {UCxxBR0+UCxxBR1} form the prescaler value.

## UCBxSTAT, USCI_Bx Status Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Unused | UC SCLLOW | UCGC | UCBBUSY | UCNACK IFG | UCSTPIFG | UCSTTIFG | UCALIFG |
| rw–0 | r–0 | rw–0 | r–0 | rw–0 | rw–0 | rw–0 | rw–0 |

| | | |
|---|---|---|
| **Unused** | Bit 7 | Unused. |
| **UC SCLLOW** | Bit 6 | SCL low<br>0 SCL is not held low<br>1 SCL is held low |
| **UCGC** | Bit 5 | General call address received. UCGC is automatically cleared when a START condition is received.<br>0 No general call address received<br>1 General call address received |
| **UCBBUSY** | Bit 4 | Bus busy<br>0 Bus inactive<br>1 Bus busy |
| **UCNACK IFG** | Bit 3 | Not-acknowledge received interrupt flag. UCNACKIFG is automatically cleared when a START condition is received.<br>0 No interrupt pending<br>1 Interrupt pending |
| **UCSTPIFG** | Bit 2 | Stop condition interrupt flag. UCSTPIFG is automatically cleared when a START condition is received.<br>0 No interrupt pending<br>1 Interrupt pending |
| **UCSTTIFG** | Bit 1 | Start condition interrupt flag. UCSTTIFG is automatically cleared if a STOP condition is received.<br>0 No interrupt pending<br>1 Interrupt pending |
| **UCALIFG** | Bit 0 | Arbitration lost interrupt flag<br>0 No interrupt pending<br>1 Interrupt pending |

## UCBxRXBUF, USCI_Bx Receive Buffer Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCRXBUFx | | | | | | | |
| r | r | r | r | r | r | r | r |

| | | |
|---|---|---|
| **UCRXBUFx** | Bits 7−0 | The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCBxRXBUF resets UCBxRXIFG. |

## UCBxTXBUF, USCI_Bx Transmit Buffer Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCTXBUFx | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

| | | |
|---|---|---|
| **UCTXBUFx** | 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. Writing to the transmit data buffer clears UCBxTXIFG. |

## UCBxI2COA, USCIBx I$^2$C Own Address Register

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| **UCGCEN** | **0** | **0** | **0** | **0** | **0** | **I2COAx** | |
| rw–0 | r0 | r0 | r0 | r0 | r0 | rw–0 | rw–0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **I2COAx** | | | | | | | |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 |

| | | |
|---|---|---|
| **UCGCEN** | Bit 15 | General call response enable<br>0 Do not respond to a general call<br>1 Respond to a general call |
| **I2COAx** | Bits 9-0 | I$^2$C own address. The I2COAx bits contain the local address of the USCI_Bx I$^2$C controller. The address is right-justified. In 7-bit addressing mode Bit 6 is the MSB, Bits 9-7 are ignored. In 10-bit addressing mode Bit 9 is the MSB. |

## UCBxI2CSA, USCI_Bx I$^2$C Slave Address Register

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **I2CSAx** | |
| r0 | r0 | r0 | r0 | r0 | r0 | rw–0 | rw–0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **I2CSAx** | | | | | | | |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 |

| | | |
|---|---|---|
| **I2CSAx** | Bits 9-0 | I$^2$C slave address. The I2CSAx bits contain the slave address of the external device to be addressed by the USCI_Bx module. It is only used in master mode. The address is right-justified. In 7-bit slave addressing mode Bit 6 is the MSB, Bits 9-7 are ignored. In 10-bit slave addressing mode Bit 9 is the MSB. |

## UCBxI2CIE, USCI_Bx $I^2C$ Interrupt Enable Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | UCNACKIE | UCSTPIE | UCSTTIE | UCALIE |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 |

| | | |
|---|---|---|
| **Reserved** | Bits 7–4 | Reserved |
| **UCNACKIE** | Bit 3 | Not-acknowledge interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **UCSTPIE** | Bit 2 | Stop condition interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **UCSTTIE** | Bit 1 | Start condition interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **UCALIE** | Bit 0 | Arbitration lost interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |

## IE2, Interrupt Enable Register 2

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | UCB0TXIE | UCB0RXIE | | |
| | | | | rw–0 | rw–0 | | |

| | | |
|---|---|---|
| | Bits 7-4 | These bits may be used by other modules. See device-specific datasheet. |
| **UCB0TXIE** | Bit 3 | USCI_B0 transmit interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **UCB0RXIE** | Bit 2 | USCI_B0 receive interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| | Bits 1-0 | These bits may be used by other modules. See device-specific datasheet. |

## IFG2, Interrupt Flag Register 2

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| | | | | UCB0 TXIFG | UCB0 RXIFG | | |
| | | | | rw–1 | rw–0 | | |

| | | |
|---|---|---|
| | Bits 7-4 | These bits may be used by other modules. See device-specific datasheet. |
| **UCB0 TXIFG** | Bit 3 | USCI_B0 transmit interrupt flag. UCB0TXIFG is set when UCB0TXBUF is empty.<br>0 No interrupt pending<br>1 Interrupt pending |
| **UCB0 RXIFG** | Bit 2 | USCI_B0 receive interrupt flag. UCB0RXIFG is set when UCB0RXBUF has received a complete character.<br>0 No interrupt pending<br>1 Interrupt pending |
| | Bits 1-0 | These bits may be used by other modules. See device-specific datasheet. |

## UC1IE, USCI_B1 Interrupt Enable Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **Unused** | **Unused** | **Unused** | **Unused** | **UCB1TXIE** | **UCB1RXIE** | | |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | | |

| | | |
|---|---|---|
| **Unused** | Bits 7-4 | Unused |
| **UCB1TXIE** | Bit 3 | USCI_B1 transmit interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| **UCB1RXIE** | Bit 2 | USCI_B1 receive interrupt enable<br>0 Interrupt disabled<br>1 Interrupt enabled |
| | Bits 1-0 | These bits may be used by other USCI modules. See device-specific datasheet. |

## UC1IFG, USCI_B1 Interrupt Flag Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **Unused** | **Unused** | **Unused** | **Unused** | **UCB1 TXIFG** | **UCB1 RXIFG** | | |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–1 | rw–0 | | |

| | | |
|---|---|---|
| **Unused** | Bits 7-4 | Unused. |
| **UCB1 TXIFG** | Bit 3 | USCI_B1 transmit interrupt flag. UCB1TXIFG is set when UCB1TXBUF is empty.<br>0 No interrupt pending<br>1 Interrupt pending |
| **UCB1 RXIFG** | Bit 2 | USCI_B1 receive interrupt flag. UCB1RXIFG is set when UCB1RXBUF has received a complete character.<br>0 No interrupt pending<br>1 Interrupt pending |
| | Bits 1-0 | These bits may be used by other modules. See device-specific datasheet. |

Chapter 14

# OA

The OA is a general purpose operational amplifier. This chapter describes the OA. Two OA modules are implemented in the MSP430x22x4 devices.

| Topic | Page |
|---|---|

## 14.1 OA Introduction

The OA op amps support front-end analog signal conditioning prior to analog-to-digital conversion.

Features of the OA include:

- ❑ Single supply, low-current operation
- ❑ Rail-to-rail output
- ❑ Programmable settling time vs. power consumption
- ❑ Software selectable configurations
- ❑ Software selectable feedback resistor ladder for PGA implementations

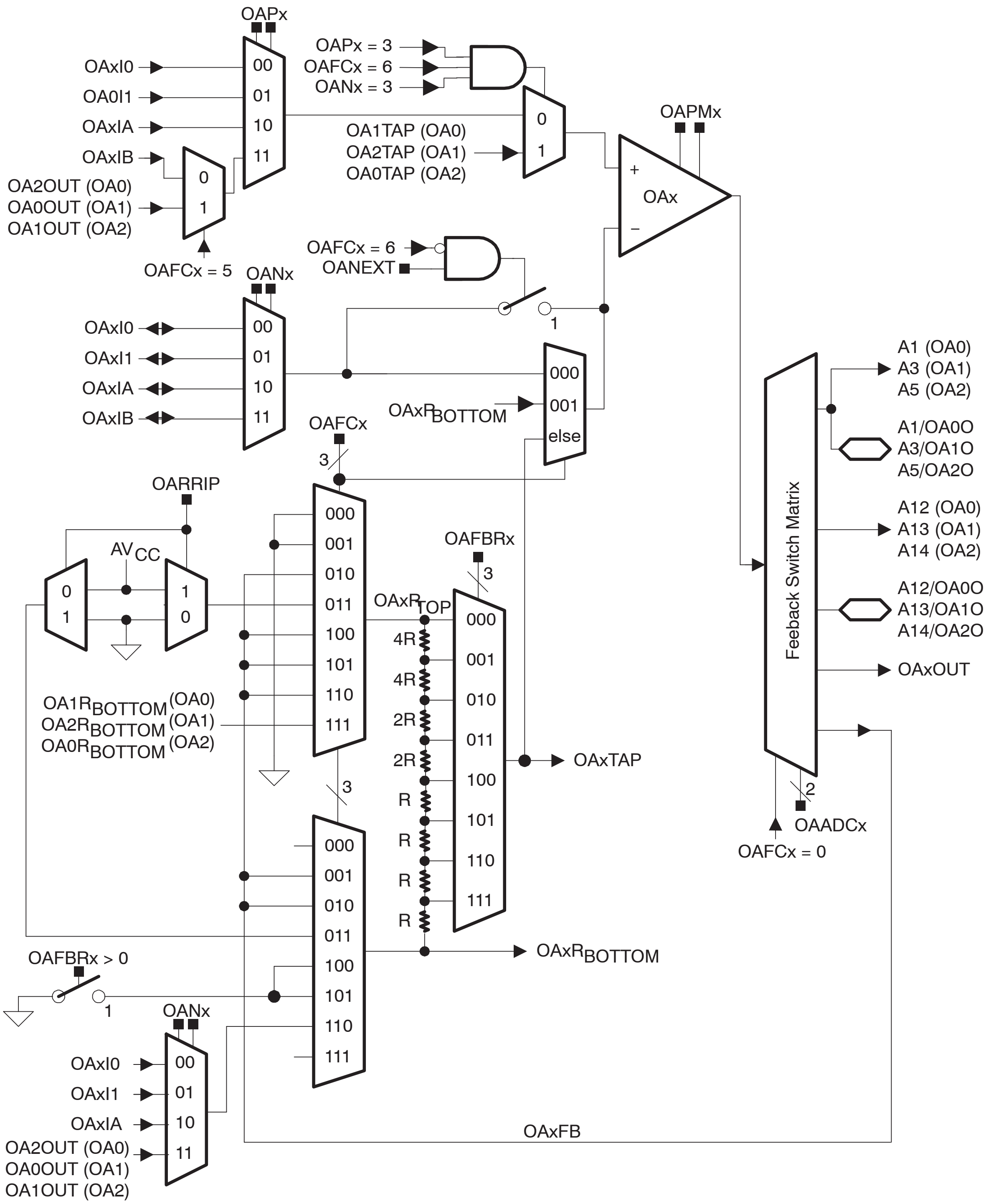
**Note: Multiple OA Modules**

Some devices may integrate more than one OA module. In the case where more than one OA is present on a device, the multiple OA modules operate identically.

Throughout this chapter, nomenclature appears such as OAxCTL0 to describe register names. When this occurs, the x is used to indicate which OA module is being discussed. In cases where operation is identical, the register is simply referred to as OAxCTL0.

The block diagram of the OA module is shown in Figure 14−1.

*Figure 14−1. OA Block Diagram*



## 14.2 OA Operation

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

### 14.2.1 OA Amplifier

The OA is a configurable, low-current, rail-to-rail output operational amplifier. It can be configured as an inverting amplifier, or a non-inverting amplifier, or can be combined with other OA modules to form differential amplifiers. The output slew rate of the OA can be configured for optimized settling time vs. power consumption with the OAPMx bits. When OAPMx = 00 the OA is off and the output is high-impedance. When OAPMx > 0, the OA is on. See the device-specific datasheet for parameters.

### 14.2.2 OA Input

The OA has configurable input selection. The signals for the + and − inputs are individually selected with the OANx and OAPx bits and can be selected as external signals or internal signals. OAxI0 and OAxI1 are external signals provided for each OA module. OA0I1 provides a non-inverting input that is tied together internally for all OA modules. OAxIA and OAxIB provide device-dependent inputs. Refer to the device datasheet for signal connections.

When the external inverting input is not needed for a mode, setting the OANEXT bit makes the internal inverting input externally available.

### 14.2.3 OA Output and Feedback Routing

The OA has configurable output selection controlled by the OAADCx bits and the OAFCx bits. The OA output signals can be routed to ADC12 inputs A12 (OA0), A13 (OA1), or A14 (OA2) internally, or can be routed to these ADC inputs and their external pins. The OA output signals can also be routed to ADC inputs A1 (OA0), A3 (OA1), or A5 (OA2) and the corresponding external pin. The OA output is also connected to an internal R-ladder with the OAFCx bits. The R-ladder tap is selected with the OAFBRx bits to provide programmable gain amplifier functionality.

Table 14−1 shows the OA output and feedback routing configurations. When OAFCx = 0 the OA is in general-purpose mode and feedback is achieved externally to the device. When OAFCx > 0 and when OAADCx = 00 or 11, the output of the OA is kept internal to the device. When OAFCx > 0 and OAADCx = 01 or 10, the OA output is routed both internally and externally.

*Table 14−1. OA Output Configurations*

| OAFCx | OAADCx | OA Output and Feedback Routing |
|---|---|---|
| = 0 | x0 | OAxOUT connected to external pins and ADC input A1, A3, or A5. |
| = 0 | x1 | OAxOUT connected to external pins and ADC input A12, A13, or A14. |
| > 0 | 00 | OAxOUT used for internal routing only. |
| > 0 | 01 | OAxOUT connected to external pins and ADC input A12, A13, or A14. |
| > 0 | 10 | OAxOUT connected to external pins and ADC input A1, A3, or A5. |
| > 0 | 11 | OAxOUT connected internally to ADC input A12, A13 , or A14. External A12, A13, or A14 pin connections are disconnected from the ADC. |

### 14.2.4 OA Configurations

The OA can be configured for different amplifier functions with the OAFCx bits as listed in Table 14−2.

*Table 14−2. OA Mode Select*

| OAFCx | OA Mode |
|---|---|
| 000 | General-purpose opamp |
| 001 | Unity gain buffer for three-opamp differential amplifier |
| 010 | Unity gain buffer |
| 011 | Comparator |
| 100 | Non-inverting PGA amplifier |
| 101 | Cascaded non-inverting PGA amplifier |
| 110 | Inverting PGA amplifier |
| 111 | Differential amplifier |

#### General Purpose Opamp Mode

In this mode the feedback resistor ladder is isolated from the OAx and the OAxCTL0 bits define the signal routing. The OAx inputs are selected with the OAPx and OANx bits. The OAx output is connected to the ADC12 input channel as selected by the OAxCTL0 bits.

#### Unity Gain Mode for Differential Amplifier

In this mode the output of the OAx is connected to the inverting input of the OAx providing a unity gain buffer. The non-inverting input is selected by the OAPx bits. The external connection for the inverting input is disabled and the OANx bits are don't care. The output of the OAx is also routed through the resistor ladder as part of the three-opamp differential amplifier. This mode is only for construction of the three-opamp differential amplifier.

#### Unity Gain Mode

In this mode the output of the OAx is connected to the inverting input of the OAx providing a unity gain buffer. The non-inverting input is selected by the OAPx bits. The external connection for the inverting input is disabled and the OANx bits are don't care. The OAx output is connected to the ADC12 input channel as selected by the OAxCTL0 bits.

### Comparator Mode

In this mode the output of the OAx is isolated from the resistor ladder. $R_{TOP}$ is connected to $AV_{SS}$ and $R_{BOTTOM}$ is connected to $AV_{CC}$ when OARRIP = 0. When OARRIP = 1, the connection of the resistor ladder is reversed. $R_{TOP}$ is connected to $AV_{CC}$ and $R_{BOTTOM}$ is connected to $AV_{SS.}$ The OAxTAP signal is connected to the inverting input of the OAx providing a comparator with a programmable threshold voltage selected by the OAFBRx bits. The non-inverting input is selected by the OAPx bits. Hysteresis can be added by an external positive feedback resistor. The external connection for the inverting input is disabled and the OANx bits are don't care. The OAx output is connected to the ADC12 input channel as selected by the OAxCTL0 bits.

### Non-Inverting PGA Mode

In this mode the output of the OAx is connected to $R_{TOP}$ and $R_{BOTTOM}$ is connected to $AV_{SS}$. The OAxTAP signal is connected to the inverting input of the OAx providing a non-inverting amplifier configuration with a programmable gain of [1+OAxTAP ratio]. The OAxTAP ratio is selected by the OAFBRx bits. If the OAFBRx bits = 0, the gain is unity. The non-inverting input is selected by the OAPx bits. The external connection for the inverting input is disabled and the OANx bits are don't care. The OAx output is connected to the ADC12 input channel as selected by the OAxCTL0 bits.

### Cascaded Non-Inverting PGA Mode

This mode allows internal routing of the OA signals to cascade two or three OA in non-inverting mode. In this mode the non-inverting input of the OAx is connected to OA2OUT (OA0), OA0OUT (OA1), or OA1OUT (OA2) when OAPx = 11. The OAx outputs are connected to the ADC12 input channel as selected by the OAxCTL0 bits.

### Inverting PGA Mode

In this mode the output of the OAx is connected to $R_{TOP}$ and $R_{BOTTOM}$ is connected to an analog multiplexer that multiplexes the OAxI0, OAxI1, OAxIA, or the output of one of the remaining OAs, selected with the OANx bits. The OAxTAP signal is connected to the inverting input of the OAx providing an inverting amplifier with a gain of −OAxTAP ratio. The OAxTAP ratio is selected by the OAFBRx bits. The non-inverting input is selected by the OAPx bits. The OAx output is connected to the ADC12 input channel as selected by the OAxCTL0 bits.

**Note: Using OAx Negative Input Simultaneously as ADC Input**

When the pin connected to the negative input multiplexer is also used as an input to the ADC, conversion errors up to 5mV may be observed due to internal wiring voltage drops.

## Differential Amplifier Mode

This mode allows internal routing of the OA signals for a two-opamp or three-opamp instrumentation amplifier. Figure 14−2 shows a two-opamp configuration with OA0 and OA1. In this mode the output of the OAx is connected to $R_{TOP}$ by routing through another OAx in the Inverting PGA mode. $R_{BOTTOM}$ is unconnected providing a unity gain buffer. This buffer is combined with one or two remaining OAx to form the differential amplifier. The OAx output is connected to the ADC12 input channel as selected by the OAxCTL0 bits.

Figure 14−2 shows an example of a two-opamp differential amplifier using OA0 and OA1. The control register settings and are shown in Table 14−3. The gain for the amplifier is selected by the OAFBRx bits for OA1 and is shown in Table 14−4. The OAx interconnections are shown in Figure 14−3.

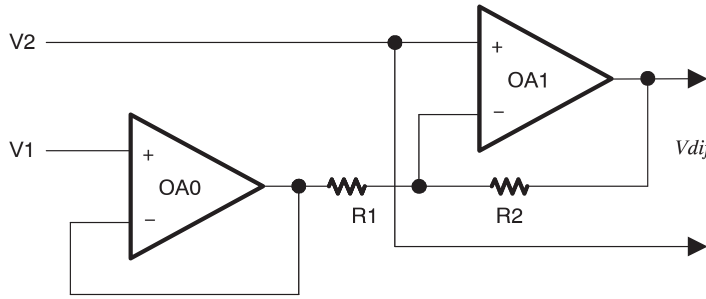
*Table 14−3. Two-Opamp Differential Amplifier Control Register Settings*

| Register | Settings (binary) |
|---|---|
| OA0CTL0 | xx xx xx 0 0 |
| OA0CTL1 | 000 111 0 x |
| OA1CTL0 | 11 xx xx x x |
| OA1CTL1 | xxx 110 0 x |

*Table 14−4. Two-Opamp Differential Amplifier Gain Settings*

| OA1 OAFBRx | Gain |
|---|---|
| 000 | 0 |
| 001 | 1/3 |
| 010 | 1 |
| 011 | 1 2/3 |
| 100 | 3 |
| 101 | 4 1/3 |
| 110 | 7 |
| 111 | 15 |

*Figure 14−2. Two Opamp Differential Amplifier*




$$Vdiff = \frac{(V2 - V1)xR2}{R1}$$

*Figure 14–3. Two Opamp Differential Amplifier OAx Interconnections*

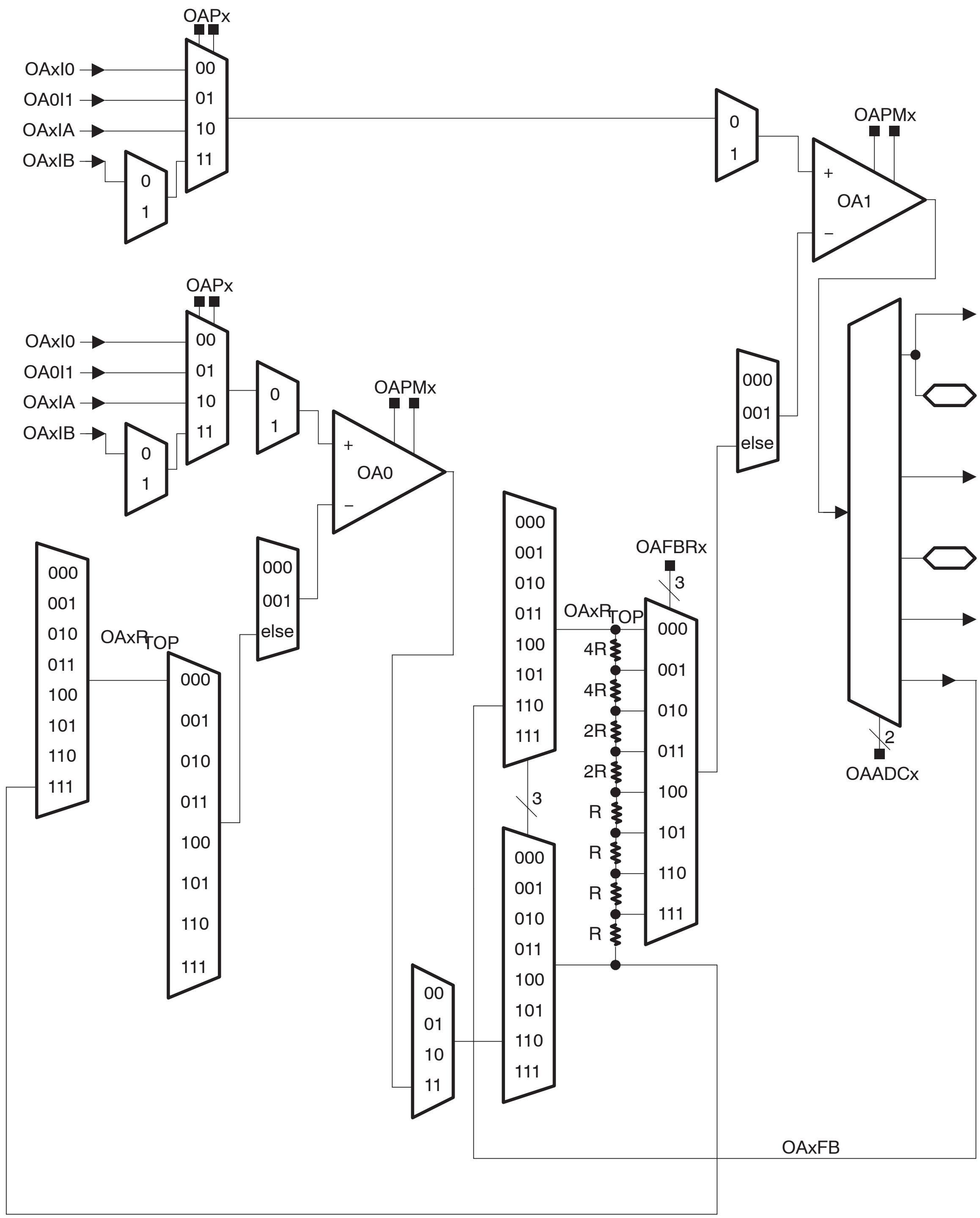


Figure 14−4 shows an example of a three-opamp differential amplifier using OA0, OA1 and OA2 (Three opamps are not available on all devices. See device-specific datasheet for implementation.). The control register settings are shown in Table 14−5. The gain for the amplifier is selected by the OAFBRx bits of OA0 and OA2. The OAFBRx settings for both OA0 and OA2 must be equal. The gain settings are shown in Table 14−6. The OAx interconnections are shown in Figure 14−5.

*Table 14−5. Three-Opamp Differential Amplifier Control Register Settings*

| Register | Settings (binary) |
|---|---|
| OA0CTL0 | xx xx xx 0 0 |
| OA0CTL1 | xxx 001 0 x |
| OA1CTL0 | xx xx xx 0 0 |
| OA1CTL1 | 000 111 0 x |
| OA2CTL0 | 11 11 xx x x |
| OA2CTL1 | xxx 110 0 x |

*Table 14−6. Three-Opamp Differential Amplifier Gain Settings*

| OA0/OA2 OAFBRx | Gain |
|---|---|
| 000 | 0 |
| 001 | 1/3 |
| 010 | 1 |
| 011 | 1 2/3 |
| 100 | 3 |
| 101 | 4 1/3 |
| 110 | 7 |
| 111 | 15 |

*Figure 14−4. Three Opamp Differential Amplifier*

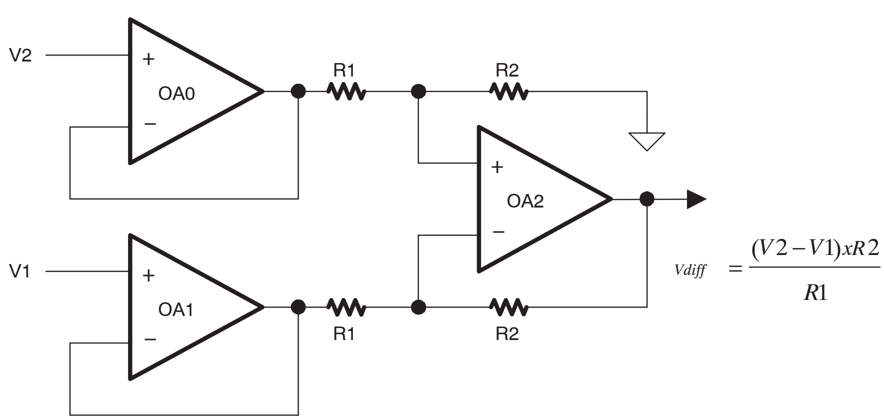
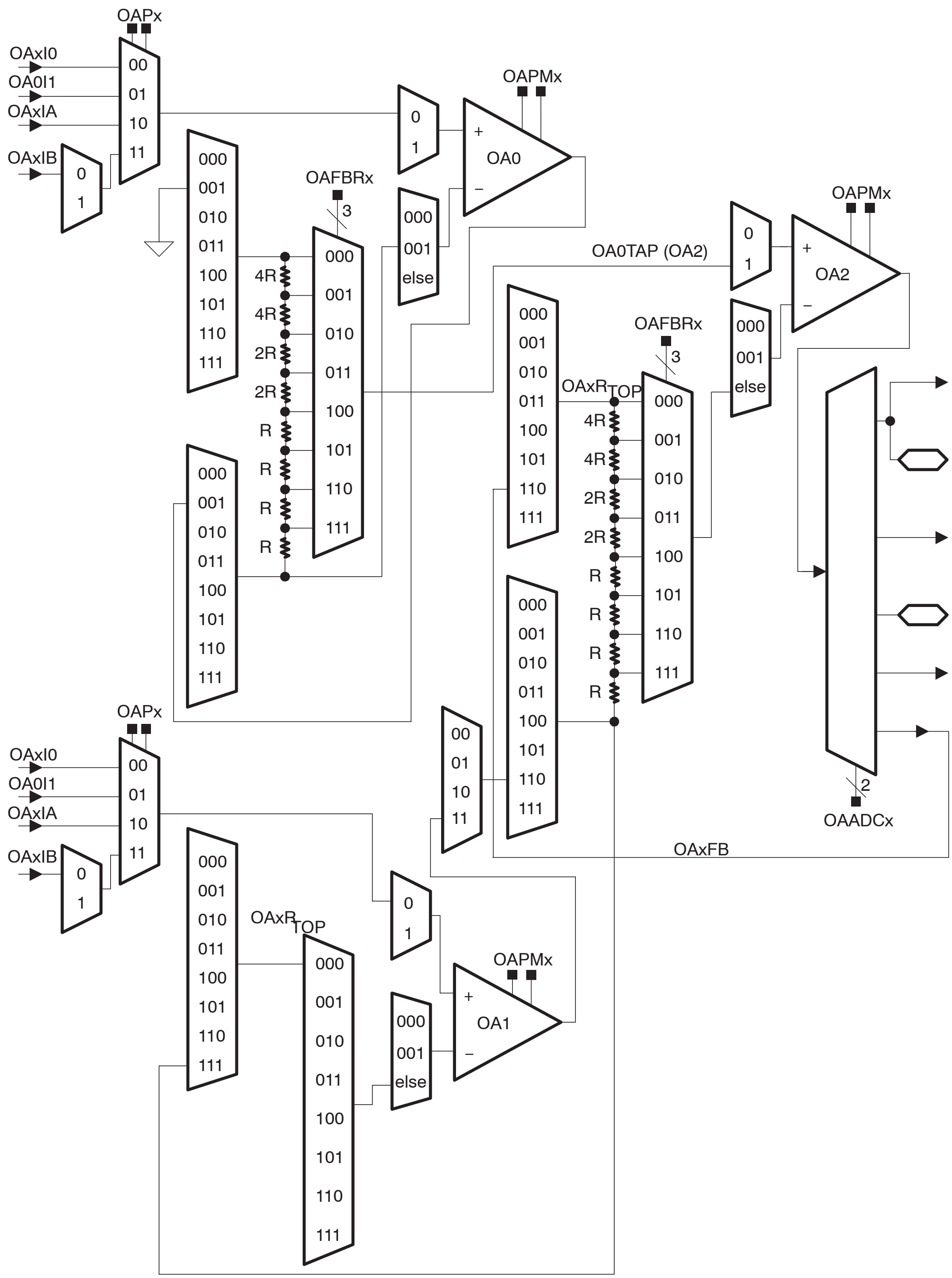


Figure 14−5. Three Opamp Differential Amplifier OAx Interconnections



## 14.3 OA Registers

The OA registers are listed in Table 14−7.

*Table 14−7.*

| Register | Short Form | Register Type | Address | Initial State |
|---|---|---|---|---|
| OA0 Control Register 0 | OA0CTL0 | Read/write | 0C0h | Reset with POR |
| OA0 Control Register 1 | OA0CTL1 | Read/write | 0C1h | Reset with POR |
| OA1 Control Register 0 | OA1CTL0 | Read/write | 0C2h | Reset with POR |
| OA1 Control Register 1 | OA1CTL1 | Read/write | 0C3h | Reset with POR |
| OA2 Control Register 0 | OA2CTL0 | Read/write | 0C4h | Reset with POR |
| OA2 Control Register 1 | OA2CTL1 | Read/write | 0C5h | Reset with POR |

## OAxCTL0, Opamp Control Register 0

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| OANx | | OAPx | | OAPMx | | OAADCx | |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 |

**OANx** Bits 7-6 Inverting input select. These bits select the input signal for the OA inverting input.
00 OAxI0
01 OAxI1
10 OAxIA – refer to device datasheet for connected signal
11 OAxIB – refer to device datasheet for connected signal

**OAPx** Bits 5-4 Non-inverting input select. These bits select the input signal for the OA non-inverting input.
00 OAxI0
01 OA0I1
10 OAxIA – refer to device datasheet for connected signal
11 OAxIB – refer to device datasheet for connected signal

**OAPMx** Bits 3-2 Slew rate select. These bits select the slew rate vs. current consumption for the OA.
00 Off, output high Z
01 Slow
10 Medium
11 Fast

**OAADCx** Bits 1–0 OA output select. These bits, together with the OAFCx bits, control the routing of the OAx output when OAPMx > 0.
When OAFCx = 0:
00 OAxOUT connected to external pins and ADC input A1, A3, or A5
01 OAxOUT connected to external pins and ADC input A12, A13, or A14
10 OAxOUT connected to external pins and ADC input A1, A3, or A5
11 OAxOUT connected to external pins and ADC input A12, A13, or A14
When OAFCx > 0:
00 OAxOUT used for internal routing only
01 OAxOUT connected to external pins and ADC input A12, A13, or A14
10 OAxOUT connected to external pins and ADC input A1, A3, or A5
11 OAxOUT connected internally to ADC input A12, A13 , or A14. External A12, A13, or A14 pin connections are disconnected from the ADC.

## OAxCTL1, Opamp Control Register 1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| OAFBRx | | | OAFCx | | | OANEXT | OARRIP† |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 |

| | | |
|---|---|---|
| **OAFBRx** | Bits 7-5 | OAx feedback resistor select<br>000 Tap 0 – 0R/16R<br>001 Tap 1 – 4R/12R<br>010 Tap 2 – 8R/8R<br>011 Tap 3 – 10R/6R<br>100 Tap 4 – 12R/4R<br>101 Tap 5 – 13R/3R<br>110 Tap 6 – 14R/2R<br>111 Tap 7 – 15R/1R |
| **OAFCx** | Bits 4-2 | OAx function control. This bit selects the function of OAx<br>000 General purpose opamp<br>001 Unity gain buffer for three-opamp differential amplifier<br>010 Unity gain buffer<br>011 Comparator<br>100 Non-inverting PGA amplifier<br>101 Cascaded non-inverting PGA amplifier<br>110 Inverting PGA amplifier<br>111 Differential amplifier |
| **OANEXT** | Bit 1 | OAx Inverting input externally available. This bit, when set, connects the inverting OAx input to the external pin when the integrated resistor network is used.<br>0 OAx inverting input not externally available<br>1 OAx inverting input externally available |
| **OARRIP** | Bit 0 | OAx reverse resistor connection in comparator mode<br>0 $R_{TOP}$ is connected to $AV_{SS}$ and $R_{BOTTOM}$ is connected to $AV_{CC}$ when OAFCx = 3<br>1 $R_{TOP}$ is connected to $AV_{CC}$ and $R_{BOTTOM}$ is connected to $AV_{SS}$ when OAFCx = 3. |

## Chapter 15

# Comparator_A+

Comparator_A+ is an analog voltage comparator. This chapter describes Comparator_A+. The Comparator_A+ is implemented in all MSP430x2xx devices except MSP430x20x0 and MSP430x22x4 devices.

| Topic | Page |
|---|---|

## 15.1 Comparator_A+ Introduction

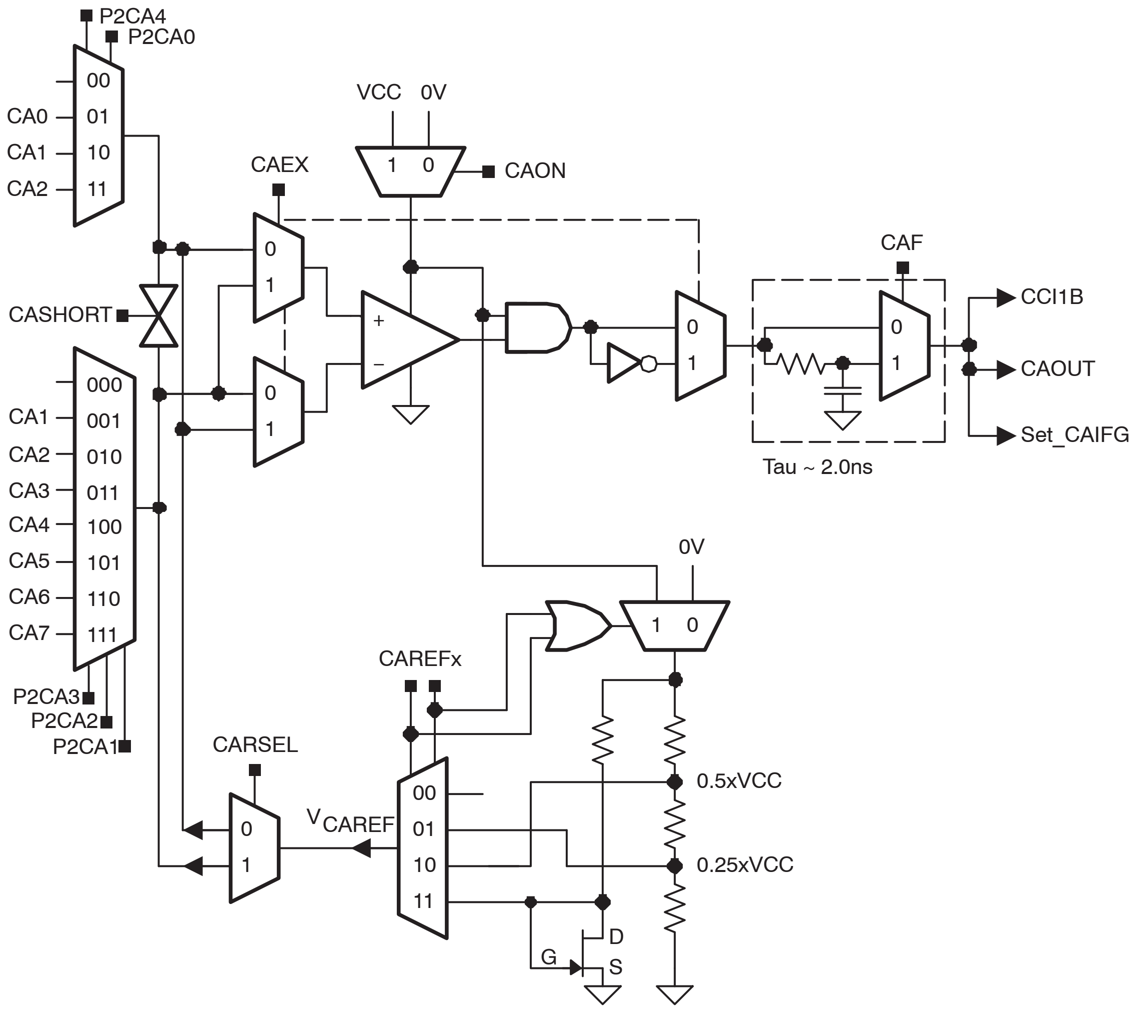
The Comparator_A+ module supports precision slope analog-to-digital conversions, supply voltage supervision, and monitoring of external analog signals.

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
- Input Multiplexer

The Comparator_A+ block diagram is shown in Figure 15−1.

*Figure 15−1. Comparator_A+ Block Diagram*



## 15.2 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.

### 15.2.1 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.

### 15.2.2 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**

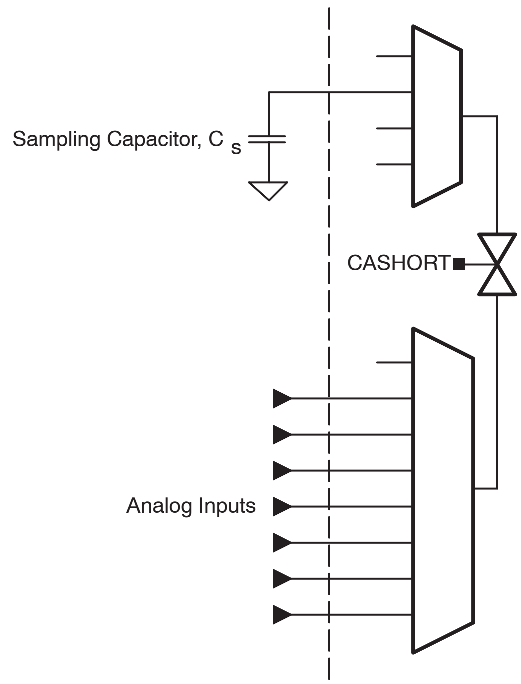
When the comparator is on, the input terminals should be connected to a signal, power, or ground. Otherwise, floating levels may cause unexpected interrupts and increased current consumption.

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.

### 15.2.3 Input Short Switch

The CASHORT bit shorts the comparator_A+ inputs. This can be used to build a simple sample-and-hold for the comparator as shown in Figure 15−2.

*Figure 15−2. Comparator_A+ Sample−And−Hold*




The required sampling time is proportional to the size of the sampling capacitor ($C_S$), the resistance of the input switches in series with the short switch ($R_i$), and the resistance of the external source ($R_S$). The total internal resistance ($R_I$) is typically in the range of 2 − 10 kΩ. The sampling capacitor $C_S$ should be greater than 100pF. The time constant, Tau, to charge the sampling capacitor $C_S$ can be calculated with the following equation:

$Tau = (R_I + R_S) \times C_S$

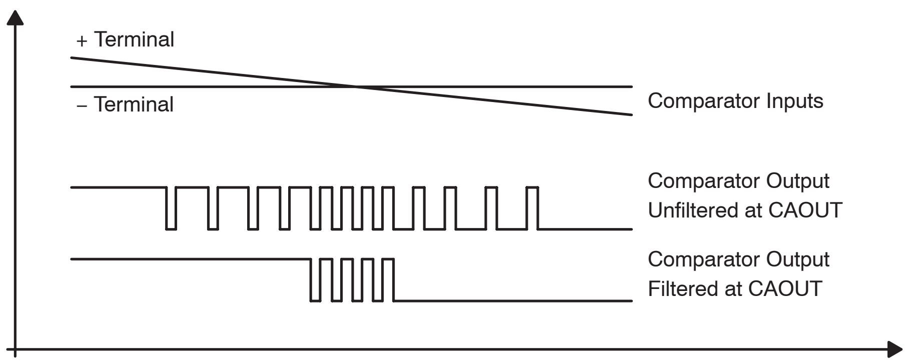
Depending on the required accuracy 3 to 10 Tau should be used as a sampling time. With 3 Tau the sampling capacitor is charged to approximately 95% of the input signals voltage level, with 5 Tau it is charge to more than 99% and with 10 Tau the sampled voltage is sufficient for 12−bit accuracy.

### 15.2.4 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 15−3. The comparator output oscillation reduces accuracy and resolution of the comparison result. Selecting the output filter can reduce errors associated with comparator oscillation.

*Figure 15−3. RC-Filter Response at the Output of the Comparator*




### 15.2.5 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.

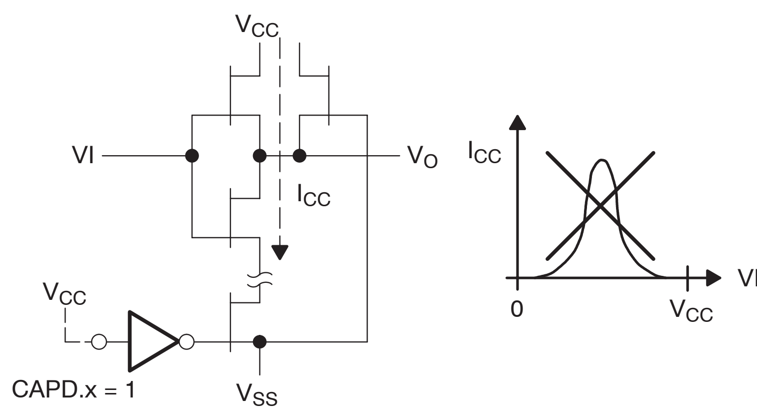
### 15.2.6 Comparator_A+, Port Disable Register CAPD

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 P1 input and output buffers as shown in Figure 15−4. When current consumption is critical, any port pin connected to analog signals should be disabled with its CAPDx bit.

Selecting an input pin to the comparator multiplexer with the P2CAx bits automatically disables the input and output buffers for that pin, regardless of the state of the associated CAPDx bit.

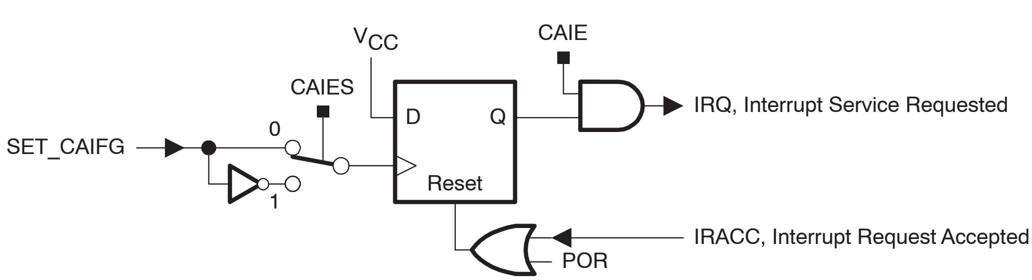
*Figure 15−4. Transfer Characteristic and Power Dissipation in a CMOS Inverter/Buffer*




### 15.2.7 Comparator_A+ Interrupts

One interrupt flag and one interrupt vector are associated with the Comparator_A+ as shown in Figure 15−5. 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.

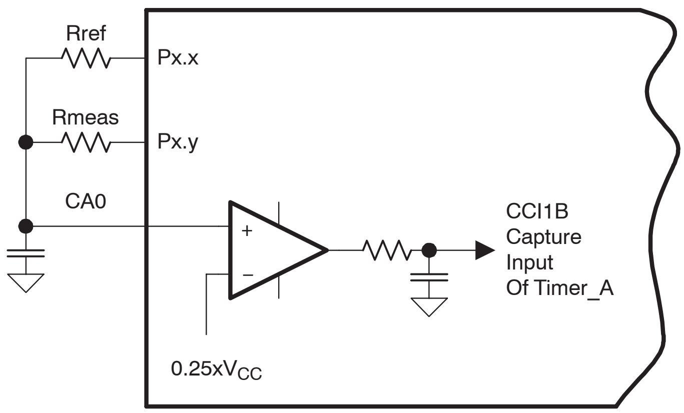
*Figure 15−5. Comparator_A+ Interrupt System*



### 15.2.8 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 15−6. A reference resister Rref is compared to Rmeas.

*Figure 15−6. Temperature Measurement System*



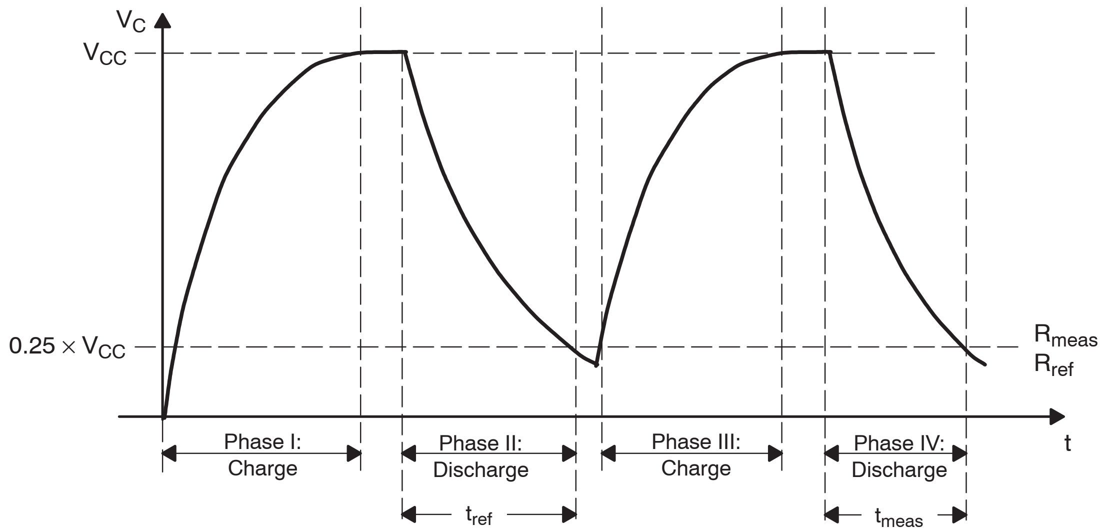

The MSP430 resources used to calculate the temperature sensed by Rmeas are:

- ❑ 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 15−7.

*Figure 15−7. 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:

$$\frac{N_{meas}}{N_{ref}} = \frac{-R_{meas} \times C \times ln\,\frac{V_{ref}}{V_{CC}}}{-R_{ref} \times C \times ln\,\frac{V_{ref}}{V_{CC}}}$$

$$\frac{N_{meas}}{N_{ref}} = \frac{R_{meas}}{R_{ref}}$$

$$R_{meas} = R_{ref} \times \frac{N_{meas}}{N_{ref}}$$

## 15.3 Comparator_A+ Registers

The Comparator_A+ registers are listed in Table 15−1:

*Table 15−1. 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.<br>When CAEX = 0:<br>0 $V_{CAREF}$ is applied to the + terminal<br>1 $V_{CAREF}$ is applied to the – terminal<br>When CAEX = 1:<br>0 $V_{CAREF}$ is applied to the – terminal<br>1 $V_{CAREF}$ is applied to the + terminal |
| **CAREF** | Bits 5-4 | Comparator_A+ reference. These bits select the reference voltage $V_{CAREF}$.<br>00 Internal reference off. An external reference can be applied.<br>01 0.25*$V_{CC}$<br>10 0.50*$V_{CC}$<br>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.<br>0 Off<br>1 On |
| **CAIES** | Bit 2 | Comparator_A+ interrupt edge select<br>0 Rising edge<br>1 Falling edge |
| **CAIE** | Bit 1 | Comparator_A+ interrupt enable<br>0 Disabled<br>1 Enabled |
| **CAIFG** | Bit 0 | The Comparator_A+ interrupt flag<br>0 No interrupt pending<br>1 Interrupt pending |

## CACTL2, Comparator_A+, Control Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **CASHORT** | **P2CA4** | **P2CA3** | **P2CA2** | **P2CA1** | **P2CA0** | **CAF** | **CAOUT** |
| rw−(0) | rw−(0) | rw−(0) | rw−(0) | rw−(0) | rw−(0) | rw−(0) | r−(0) |

| | | |
|---|---|---|
| **CASHORT** | Bit 7 | Input short. This bit shorts the + and − input terminals.<br>0 Inputs not shorted.<br>1 Inputs shorted. |
| **P2CA4** | Bit 6 | Input select. This bit together with P2CA0 selects the + terminal input when CAEX = 0 and the − terminal input when CAEX = 1. |
| **P2CA3**<br>**P2CA2**<br>**P2CA1** | Bits 5-3 | Input select. These bits select the − terminal input when CAEX = 0 and the + terminal input when CAEX = 1.<br>000 No connection<br>001 CA1<br>010 CA2<br>011 CA3<br>100 CA4<br>101 CA5<br>110 CA6<br>111 CA7 |
| **P2CA0** | Bit 2 | Input select. This bit, together with P2CA4, selects the + terminal input when CAEX = 0 and the − terminal input when CAEX = 1.<br>00 No connection<br>01 CA0<br>10 CA1<br>11 CA2 |
| **CAF** | Bit 1 | Comparator_A+ output filter<br>0 Comparator_A+ output is not filtered<br>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. |

## CAPD, Comparator_A+, Port Disable Register

| 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 CA0 is on pin P2.3, 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.<br>0 The input buffer is enabled.<br>1 The input buffer is disabled. |

Chapter 16

# 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 MSP430x20x2 and MSP430x22x4 devices.

**Topic** **Page**

## 16.1 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).

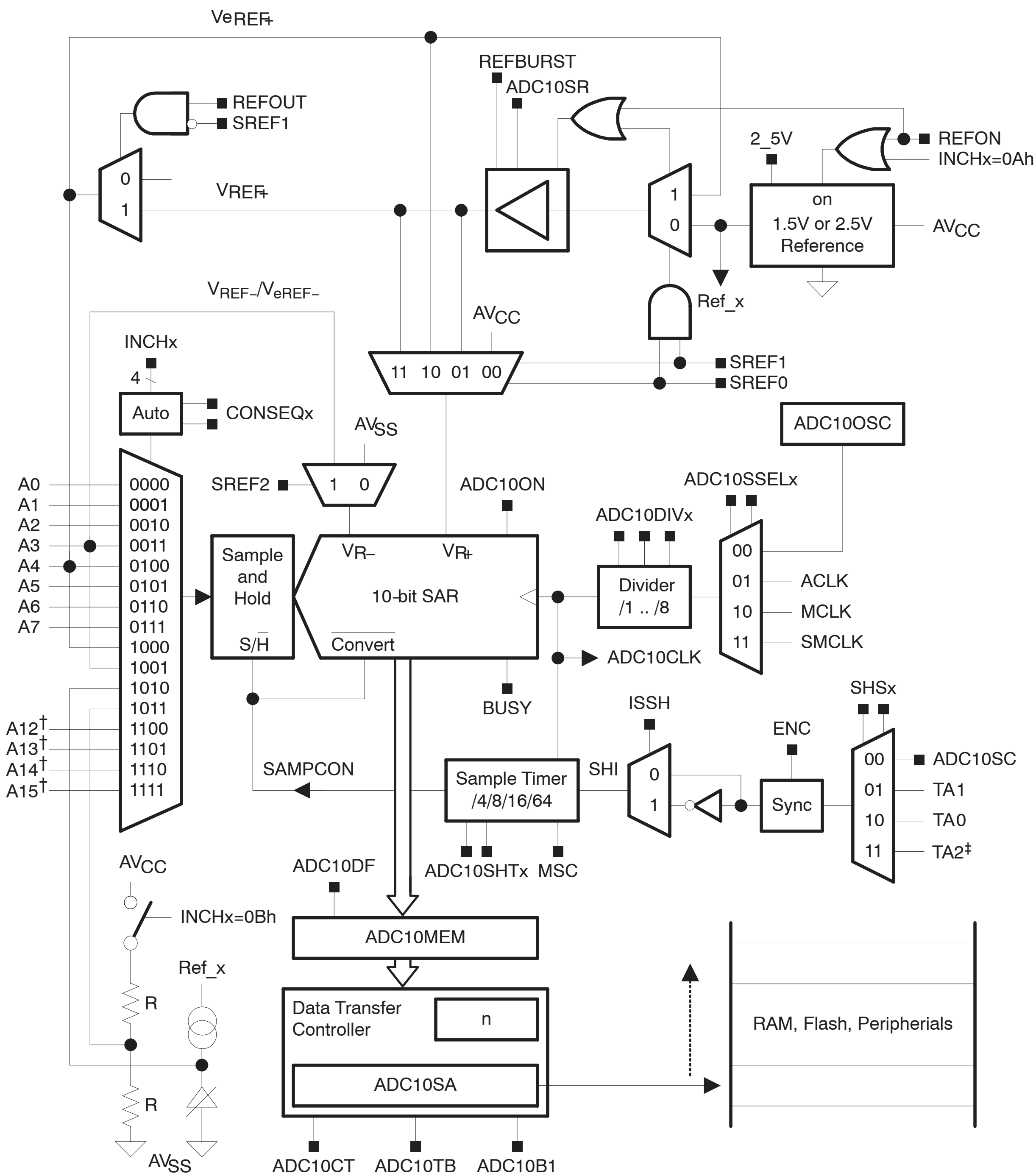
The DTC allows ADC10 samples to be converted and stored anywhere in memory without CPU intervention. The module can be configured with user software to support a variety of applications.

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 (twelve on MSP430x22x4 devices)
- Conversion channels for internal temperature sensor, $V_{CC}$, 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 16−1.

*Figure 16−1. ADC10 Block Diagram*




†MSP430x22x4 devices only. Channels A12-A15 tied to channel A11 in other devices
‡TA1 on MSP43020x2 devices

## 16.2 ADC10 Operation

The ADC10 module is configured with user software. The setup and operation of the ADC10 is discussed in the following sections.

### 16.2.1 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 datasheet for the ADC10OSC specification.

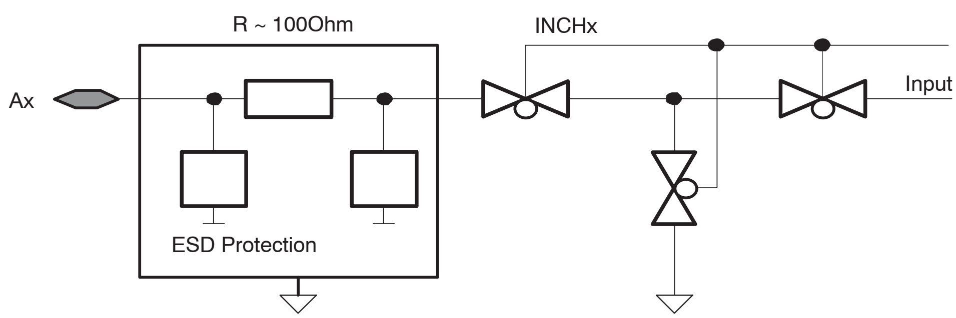
The user must ensure that the clock chosen for ADC10CLK remains active until the end of a conversion. If the clock is removed during a conversion, the operation will not complete, and any result will be invalid.

### 16.2.2 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 16−2. 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 ($V_{SS}$) 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 16−2. Analog Multiplexer*




#### *Analog Port Selection*

The ADC10 external inputs Ax, $Ve_{REF+}$, and $V_{REF-}$ share terminals with general purpose I/O ports, which are digital CMOS gates (see device-specific datasheet). 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 and output buffers.

```
; P2.3 on MSP430x22x4 device configured for analog input
   BIS.B  #08h,&ADC10AE0  ; P2.3 ADC10 function and enable
```

### 16.2.3 Voltage Reference Generator

The ADC10 module contains a built-in voltage reference with two selectable voltage levels. Setting REFON = 1 enables the internal reference. When REF2_5V = 1, the internal reference is 2.5 V. When REF2_5V = 0, the reference is 1.5 V. The internal reference voltage may be used internally and, when REFOUT = 0, externally on pin $V_{REF+}$.

External references may be supplied for $V_{R+}$ and $V_{R-}$ through pins A4 and A3 respectively. When external references are used, or when $V_{CC}$ is used as the reference, the internal reference may be turned off to save power.

An external positive reference $Ve_{REF+}$ can be buffered by setting SREF0 = 1 and SREF1 = 1. This allows using an external reference with a large internal resistance at the cost of the buffer current. When REFBURST = 1 the increased current consumption is limited to the sample and conversion period.

External storage capacitance is not required for the ADC10 reference source as on the ADC12.

#### Internal Reference Low-Power Features

The ADC10 internal reference generator is designed for low power applications. The reference generator includes a band-gap voltage source and a separate buffer. The current consumption of each is specified separately in the device-specific datasheet. When REFON = 1, both are enabled and when REFON = 0 both are disabled. The total settling time when REFON becomes set is $\leq$ 30 $\mu$s.

When REFON = 1, but no conversion is active, the buffer is automatically disabled and automatically re-enabled when needed. When the buffer is disabled, it consumes no current. In this case, the band-gap voltage source remains enabled.

When REFOUT = 1, the REFBURST bit controls the operation of the internal reference buffer. When REFBURST = 0, the buffer will be on continuously, allowing the reference voltage to be present outside the device continuously. When REFBURST = 1, the buffer is automatically disabled when the ADC10 is not actively converting, and automatically re-enabled when needed.

The internal reference buffer also has selectable speed vs. power settings. When the maximum conversion rate is below 50 ksps, setting ADC10SR = 1 reduces the current consumption of the buffer approximately 50%.

### 16.2.4 Auto Power-Down

The ADC10 is designed for low power applications. When the ADC10 is not actively converting, the core is automatically disabled and automatically re-enabled when needed The ADC10OSC is also automatically enabled when needed and disabled when not needed. When the core or oscillator are disabled, they consume no current.

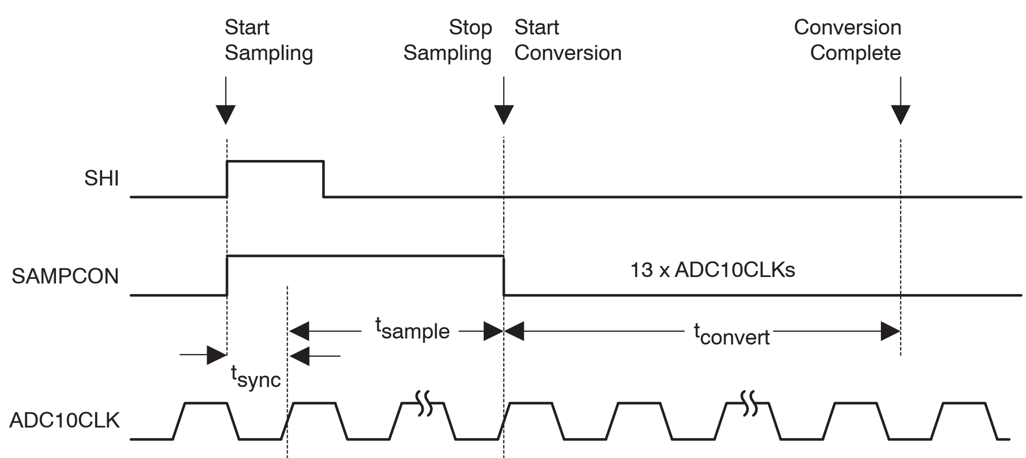
### 16.2.5 Sample and Conversion Timing

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 16−3.

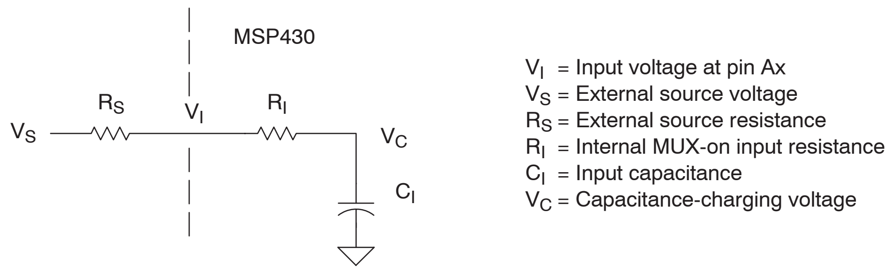
*Figure 16−3. Sample Timing*




### 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 16−4. An internal MUX-on input resistance $R_I$ (max. 2 k$\Omega$) in series with capacitor $C_I$ (max. 27 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 16−4. Analog Input Equivalent Circuit*




The resistance of the source $R_S$ and $R_I$ affect $t_{sample}$.The following equations can be used to calculate the minimum sampling time for a 10-bit conversion.

$$t_{sample} > (R_S + R_I) \times \ln(2^{11}) \times C_I$$

Substituting the values for $R_I$ and $C_I$ given above, the equation becomes:

$$t_{sample} > (R_S + 2k) \times 7.625 \times 27pF$$

For example, if $R_S$ is 10 kΩ, $t_{sample}$ must be greater than 2.47 µs.

When the reference buffer is used in burst mode, the sampling time must be greater than the sampling time calculated and the settling time of the buffer, $t_{REFBURST}$:

$$t_{sample} > \begin{cases} (R_S + R_I) \times \ln(2^{11}) \times C_I \\ t_{REFBURST} \end{cases}$$

For example, if $V_{Ref}$ is 1.5 V and $R_S$ is 10 kΩ, $t_{sample}$ must be greater than 2.47 µs when ADC10SR = 0, or 2.5 µs when ADC10SR = 1. See the device-specific datasheet for parameters.

To calculate the buffer settling time when using an external reference, the formula is:

$$t_{REFBURST} = SR \times V_{Ref} - 0.5\mu s$$

Where:

*SR:* Buffer slew rate
(~1 µs/V when ADC10SR = 0 and ~2 µs/V when ADC10SR = 1)
Vref: External reference voltage

### 16.2.6 Conversion Modes

The ADC10 has four operating modes selected by the CONSEQx bits as discussed in Table 16−1.

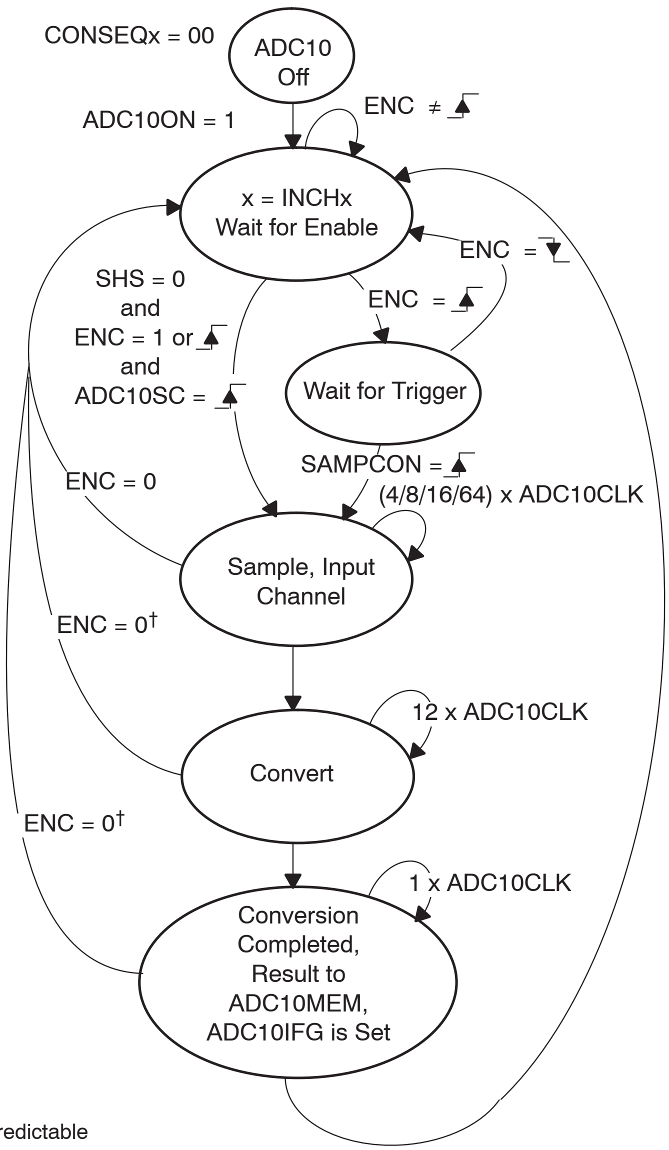
*Table 16−1. Conversion Mode Summary*

| 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 16−5 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 16−5. Single-Channel Single-Conversion Mode*

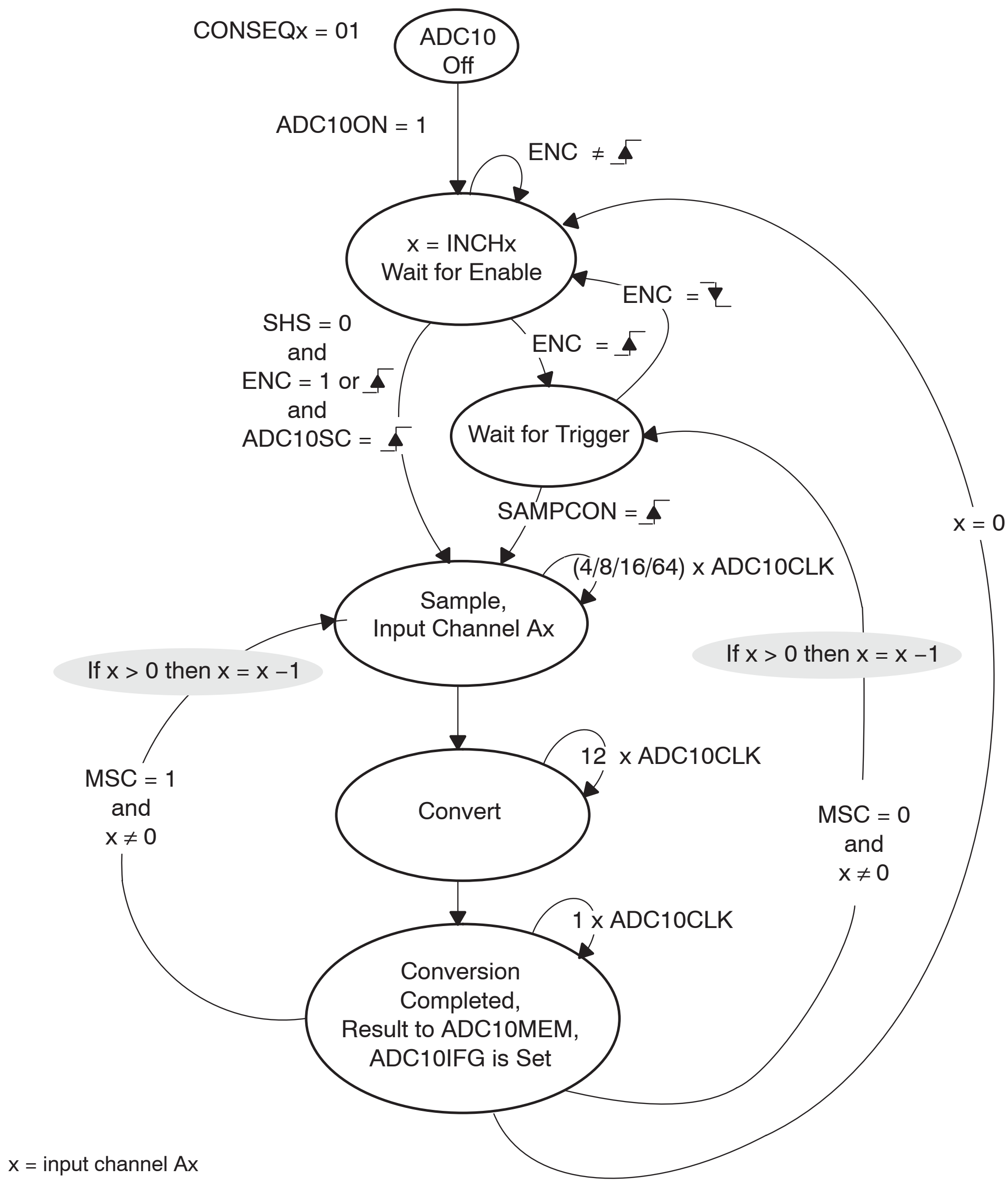



x = input channel Ax
† Conversion result is unpredictable

### 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 16−6 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.

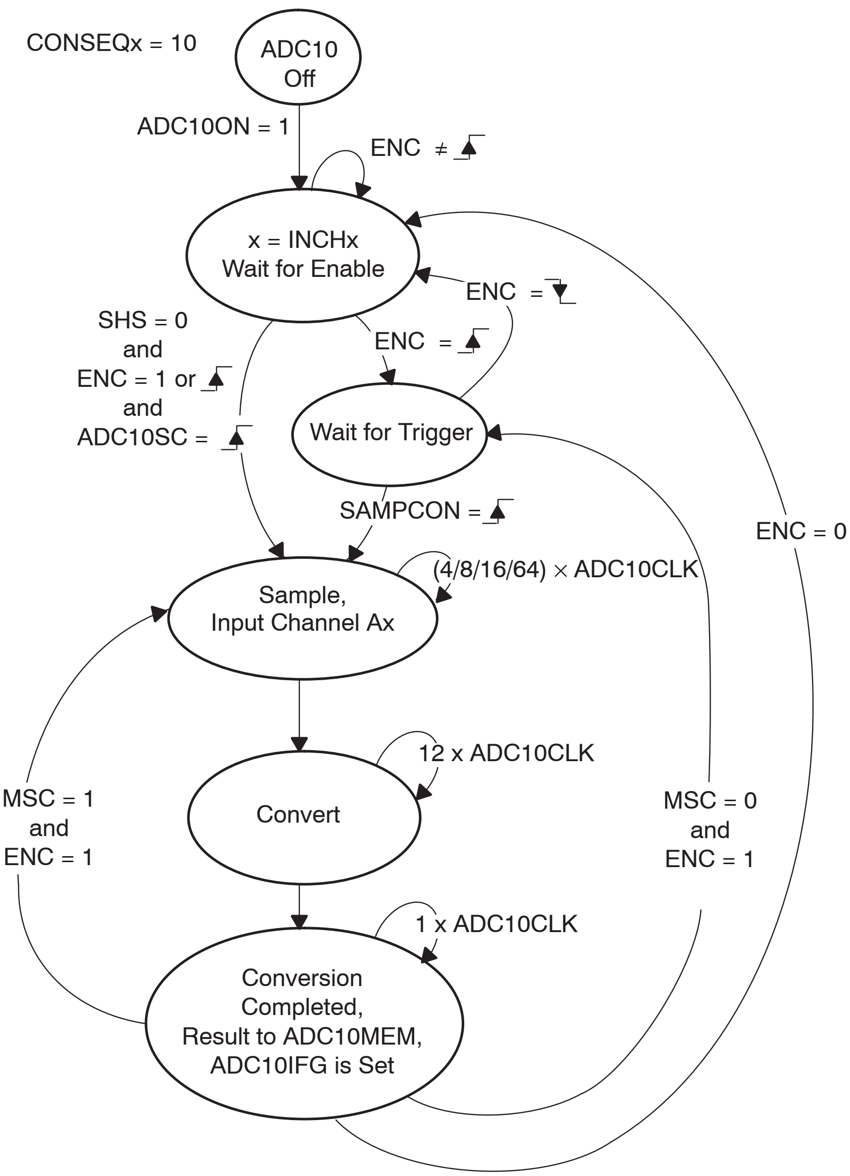
*Figure 16−6. Sequence-of-Channels Mode*



## Repeat-Single-Channel Mode

A single channel selected by INCHx is sampled and converted continuously. Each ADC result is written to ADC10MEM. Figure 16−7 shows the repeat-single-channel mode.

*Figure 16−7. Repeat-Single-Channel Mode*

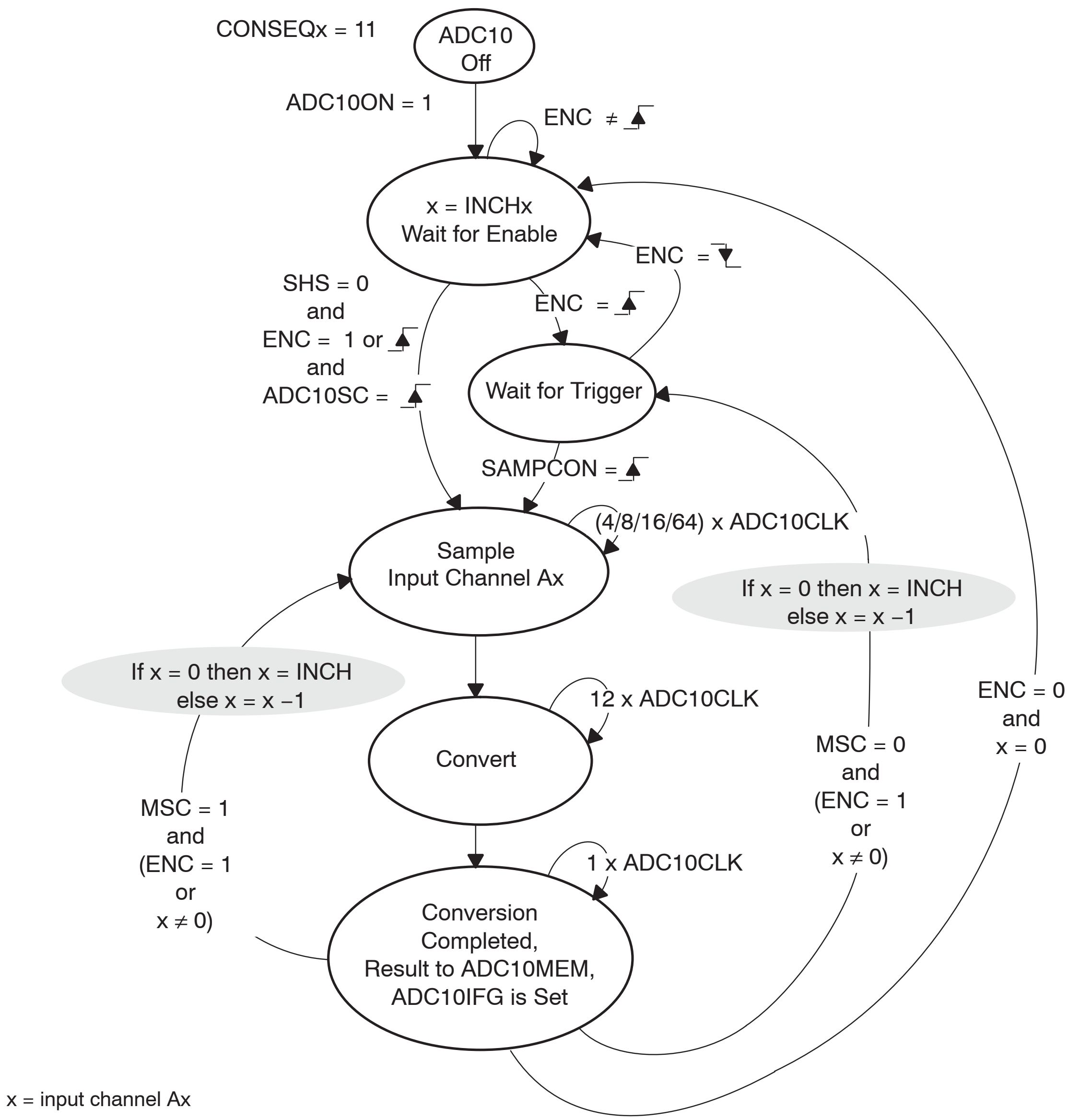



x = input channel Ax

## 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 16−8 shows the repeat-sequence-of-channels mode.

*Figure 16−8. 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.

### 16.2.7 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.

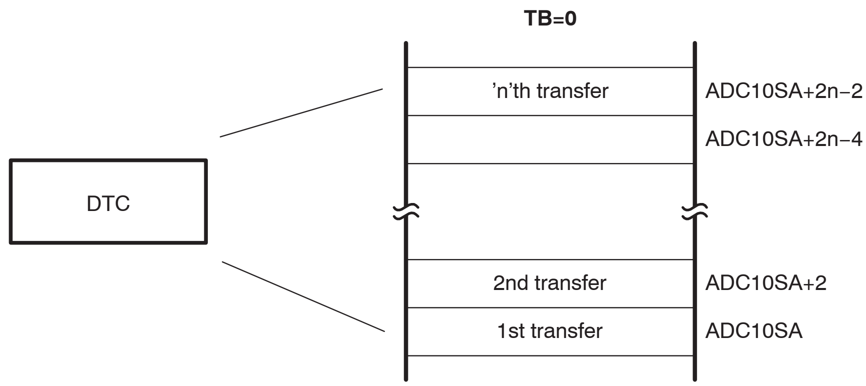
A DTC transfer must not be initiated while the ADC10 is busy. Software must ensure that no active conversion or sequence is in progress when the DTC is configured:

```
; 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 16−9.

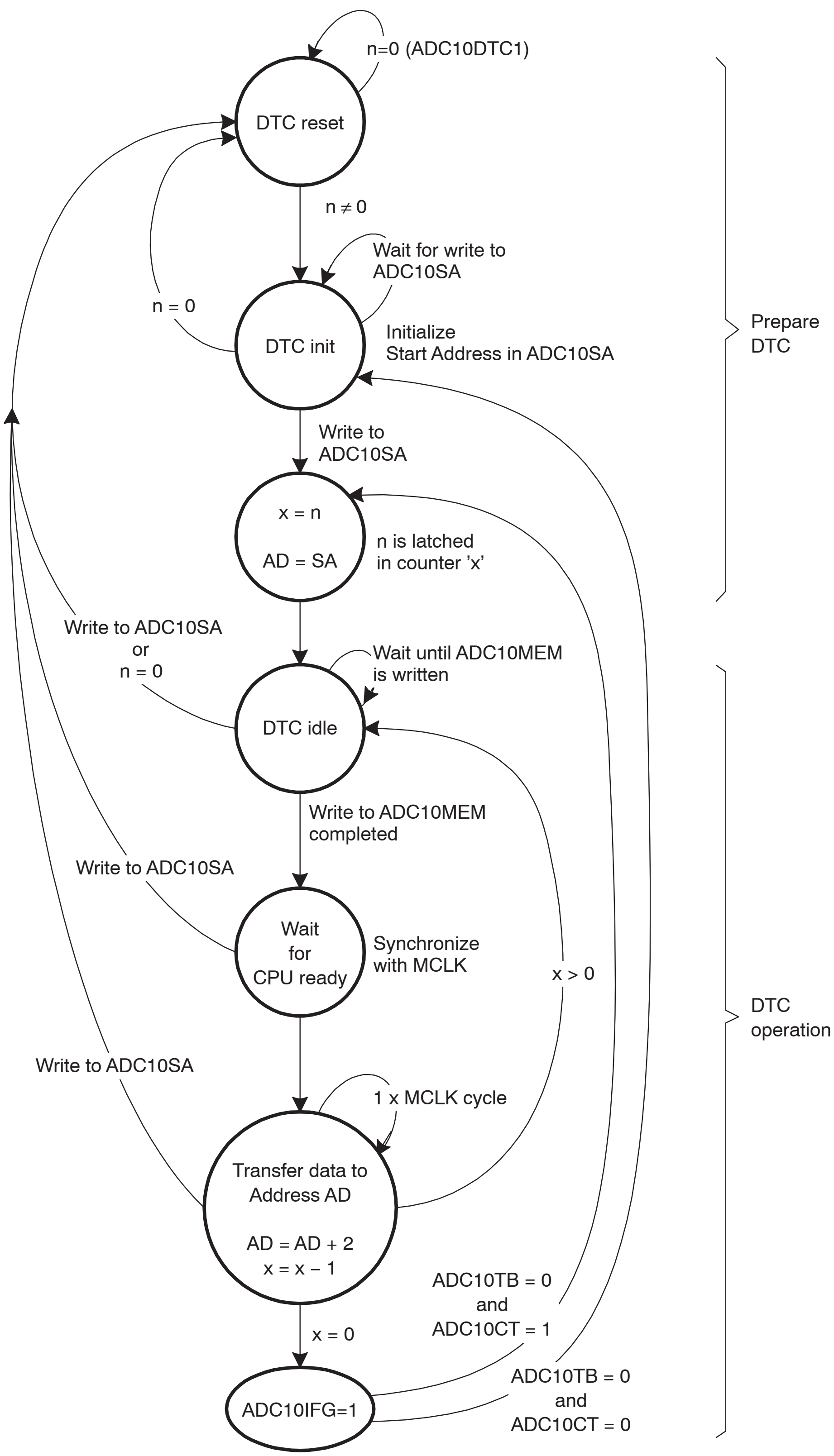
*Figure 16−9. One-Block Transfer*




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 16−10 shows a state diagram of the one-block mode.

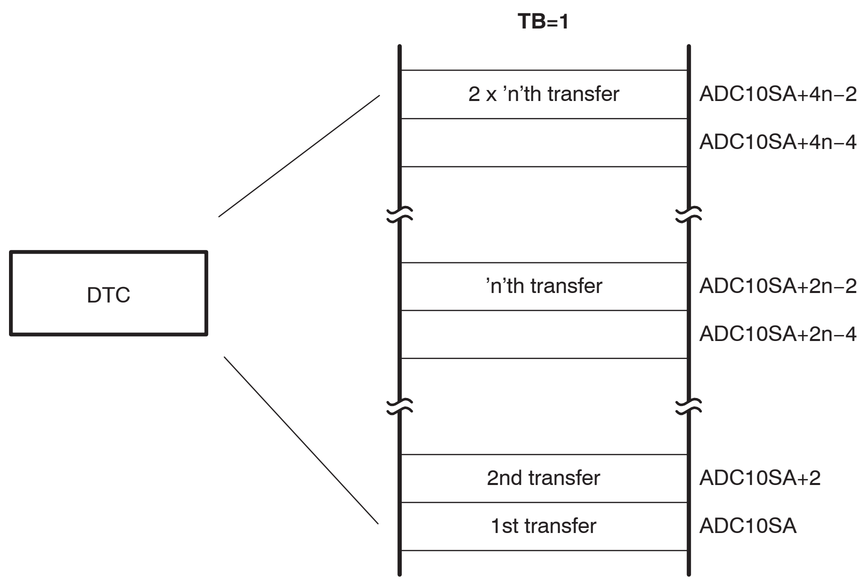
*Figure 16−10. State Diagram for Data Transfer Control in One-Block Transfer Mode*



## 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 16−11**.**

*Figure 16−11. Two-Block Transfer*

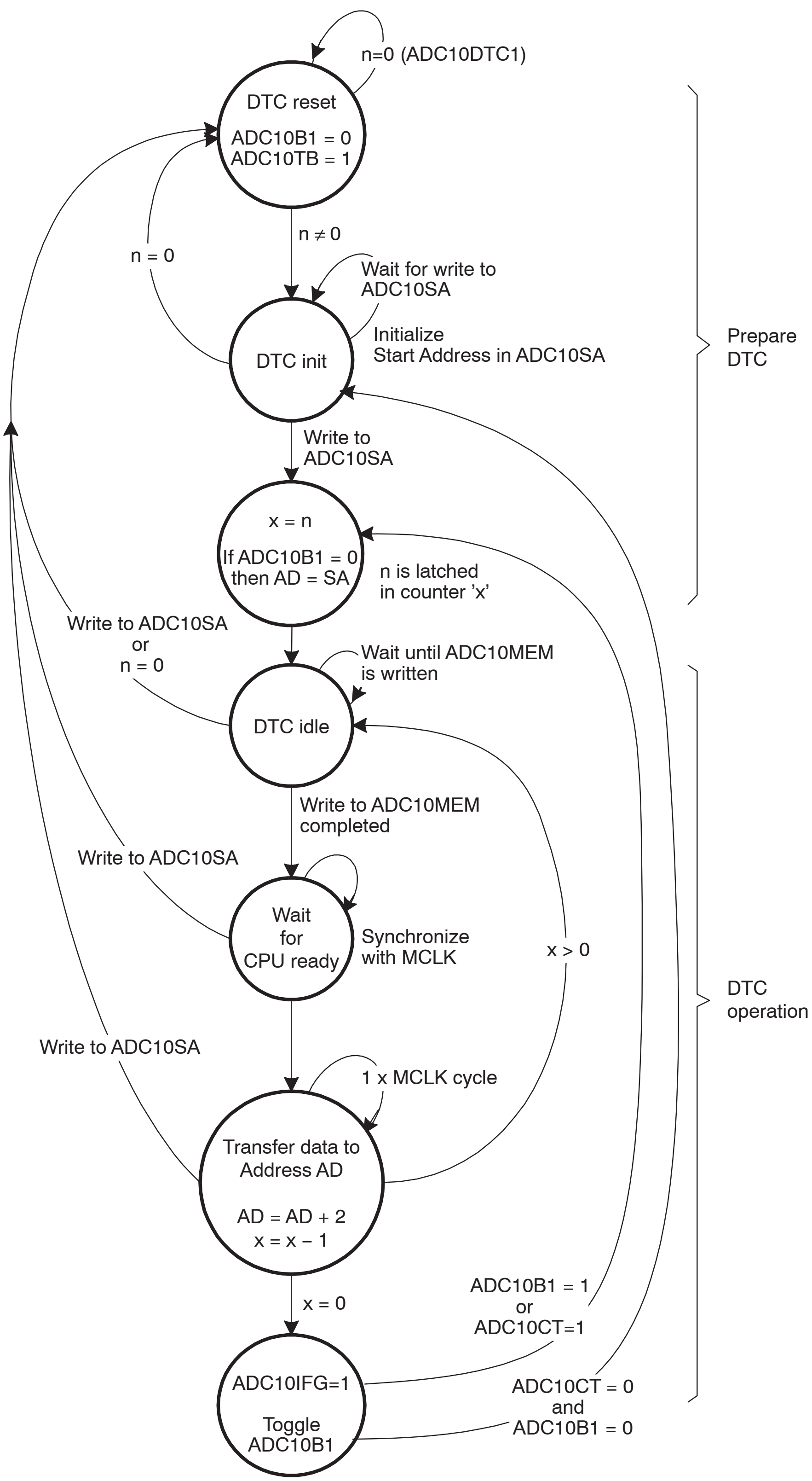



The internal address pointer is initially equal to ADC10SA and the internal transfer counter is initially equal to ‘n’. The internal pointer and counter are not visible to software. The DTC transfers the word-value of ADC10MEM to the address pointer ADC10SA. After each DTC transfer the internal address 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 16−12 shows a state diagram of the two-block mode.

*Figure 16−12. 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 16−2.

*Table 16−2. 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 + 2 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 + 2 μs† |

† The additional 2 μs are needed to start the DCOCLK. See device-datasheet for parameters.

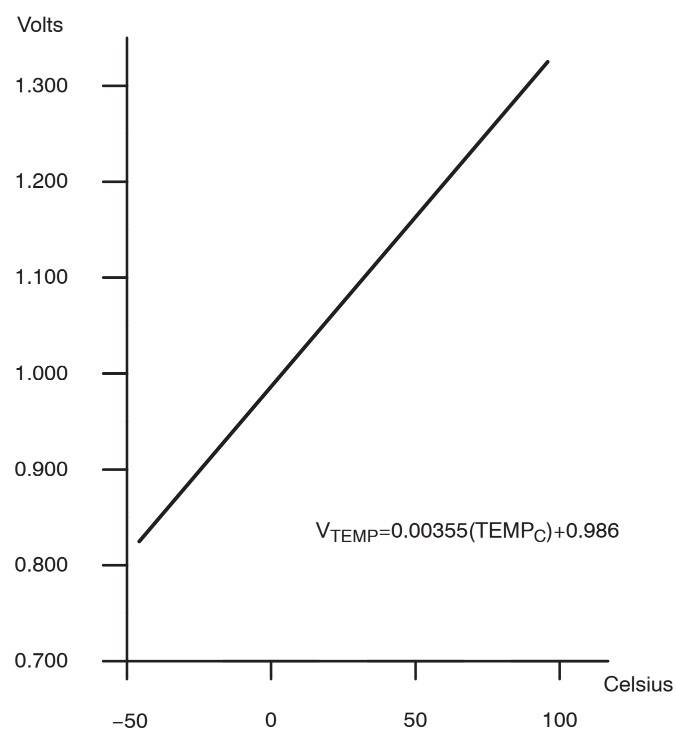
### 16.2.8 Using the Integrated Temperature Sensor

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 16−13. 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 datasheet 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 16−14. Typical Temperature Sensor Transfer Function*



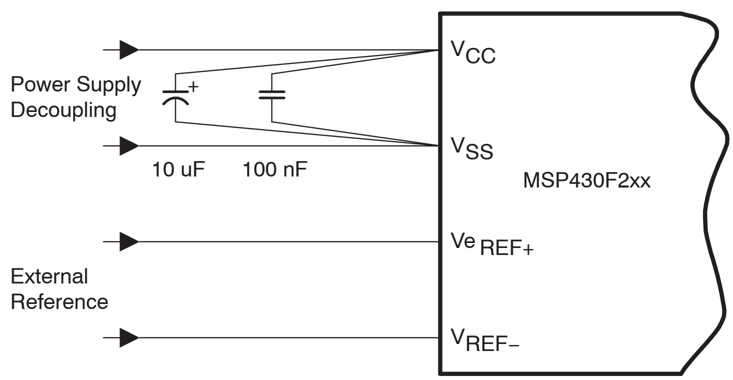
### 16.2.9 ADC10 Grounding and Noise Considerations

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 16−15 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.

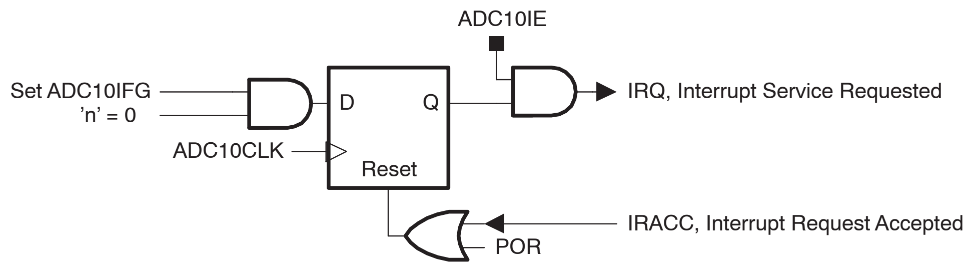
*Figure 16−16. ADC10 Grounding and Noise Considerations*



### 16.2.10 ADC10 Interrupts

One interrupt and one interrupt vector are associated with the ADC10 as shown in Figure 16−17. 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 16−17. ADC10 Interrupt System*



## 16.3 ADC10 Registers

The ADC10 registers are listed in Table 16−3.

*Table 16−3. ADC10 Registers*

| Register | Short Form | Register Type | Address | Initial State |
|---|---|---|---|---|
| ADC10 Input enable register 0 | ADC10AE0 | Read/write | 04Ah | Reset with POR |
| ADC10 Input enable register 1 | ADC10AE1 | Read/write | 04Bh | 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 $V_{R+} = V_{CC}$ and $V_{R-} = V_{SS}$
- 001 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{SS}$
- 010 $V_{R+} = Ve_{REF+}$ and $V_{R-} = V_{SS}$
- 011 $V_{R+}$ = Buffered $Ve_{REF+}$ and $V_{R-} = V_{SS}$
- 100 $V_{R+} = V_{CC}$ 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 $V_{R+}$ = Buffered $Ve_{REF+}$ 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 reference buffer drive capability for the maximum sampling rate. Setting ADC10SR reduces the current consumption of the reference buffer.
- 0 Reference buffer supports up to ~200 ksps
- 1 Reference buffer supports up to ~50 ksps

**REFOUT** Bit 9 Reference output
- 0 Reference output off
- 1 Reference output on

**REFBURST** Bit 8 Reference burst.
- 0 Reference buffer on continuously
- 1 Reference buffer on only during sample-and-conversion

| | | |
|---|---|---|
| **MSC** | Bit 7 | Multiple sample and conversion. Valid only for sequence or repeated modes.<br>0 The sampling requires a rising edge of the SHI signal to trigger each sample-and-conversion.<br>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.<br>0 1.5 V<br>1 2.5 V |
| **REFON** | Bit 5 | Reference generator on<br>0 Reference off<br>1 Reference on |
| **ADC10ON** | Bit 4 | ADC10 on<br>0 ADC10 off<br>1 ADC10 on |
| **ADC10IE** | Bit 3 | ADC10 interrupt enable<br>0 Interrupt disabled<br>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.<br>0 No interrupt pending<br>1 Interrupt pending |
| **ENC** | Bit 1 | Enable conversion<br>0 ADC10 disabled<br>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.<br>0 No sample-and-conversion start<br>1 Start sample-and-conversion |

## ADC10CTL1, ADC10 Control Register 1

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| INCHx | | | | SHSx | | ADC10DF | ISSH |
| rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ADC10DIVx | | | ADC10SSELx | | CONSEQx | | ADC10 BUSY |
| rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | r–0 |

Modifiable only when ENC = 0

**INCHx** Bits 15-12 Input channel select. These bits select the channel for a single-conversion or the highest channel for a sequence of conversions.
- 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 sensor
- 1011 $(V_{CC} - V_{SS}) / 2$
- 1100 $(V_{CC} - V_{SS}) / 2$, A12 on MSP430x22x4 devices
- 1101 $(V_{CC} - V_{SS}) / 2$, A13 on MSP430x22x4 devices
- 1110 $(V_{CC} - V_{SS}) / 2$, A14 on MSP430x22x4 devices
- 1111 $(V_{CC} - V_{SS}) / 2$, A15 on MSP430x22x4 devices

**SHSx** Bits 11-10 Sample-and-hold source select
- 00 ADC10SC bit
- 01 Timer_A.OUT1
- 10 Timer_A.OUT0
- 11 Timer_A.OUT2 (Timer_A.OUT1 on MSP43020x2 devices)

**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<br>000 /1<br>001 /2<br>010 /3<br>011 /4<br>100 /5<br>101 /6<br>110 /7<br>111 /8 |
| **ADC10 SSELx** | Bits 4-3 | ADC10 clock source select<br>00 ADC10OSC<br>01 ACLK<br>10 MCLK<br>11 SMCLK |
| **CONSEQx** | Bits 2-1 | Conversion sequence mode select<br>00 Single-channel-single-conversion<br>01 Sequence-of-channels<br>10 Repeat-single-channel<br>11 Repeat-sequence-of-channels |
| **ADC10 BUSY** | Bit 0 | ADC10 busy. This bit indicates an active sample or conversion operation<br>0 No operation is active.<br>1 A sequence, sample, or conversion is active. |

## ADC10AE0, Analog (Input) Enable Control Register 0

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **ADC10AE0x** | | | | | | | |
| rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) |

| | | |
|---|---|---|
| **ADC10AE0x** | Bits 7-0 | ADC10 analog enable. These bits enable the corresponding pin for analog input. BIT0 corresponds to A0, BIT1 corresponds to A1, etc.<br>0 Analog input disabled<br>1 Analog input enabled |

## ADC10AE1, Analog (Input) Enable Control Register 1 (MSP430x22x4 only)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **ADC10AE1x** | | | | **Reserved** | **Reserved** | **Reserved** | **Reserved** |
| rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) | rw–(0) |

| | | |
|---|---|---|
| **ADC10AE1x** | Bits 7-4 | ADC10 analog enable. These bits enable the corresponding pin for analog input. BIT4 corresponds to A12, BIT5 corresponds to A13, BIT6 corresponds to A14, and BIT7 corresponds to A15.<br>0 Analog input disabled<br>1 Analog input enabled |

### ADC10MEM, Conversion-Memory Register, Binary Format

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| **0** | **0** | **0** | **0** | **0** | **0** | **Conversion Results** | |
| r0 | r0 | r0 | r0 | r0 | r0 | r | r |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **Conversion Results** | | | | | | | |
| r | r | r | r | r | r | r | r |

| | | |
|---|---|---|
| **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

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| **Conversion Results** | | | | | | | |
| r | r | r | r | r | r | r | r |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **Conversion Results** | | **0** | **0** | **0** | **0** | **0** | **0** |
| r | r | r0 | r0 | r0 | r0 | r0 | r0 |

| | | |
|---|---|---|
| **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.<br>0 One-block transfer mode<br>1 Two-block transfer mode |
| **ADC10CT** | Bit 2 | ADC10 continuous transfer.<br>0 Data transfer stops when one block (one-block mode) or two blocks (two-block mode) have completed.<br>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<br>0 Block 2 is filled<br>1 Block 1 is filled |
| **ADC10 FETCH** | Bit 0 | This bit should normally be reset. |

## ADC10DTC1, Data Transfer Control Register 1

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **DTC Transfers** | | | | | | | |
| rw−(0) | rw−(0) | rw−(0) | rw−(0) | rw−(0) | rw−(0) | rw−(0) | rw−(0) |

| | | |
|---|---|---|
| **DTC Transfers** | Bits 7-0 | DTC transfers. These bits define the number of transfers in each block.<br>0 DTC is disabled<br>01h-0FFh Number of transfers per block |

## ADC10SA, Start Address Register for Data Transfer

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| **ADC10SAx** | | | | | | | |
| rw−(0) | rw−(0) | rw−(0) | rw−(0) | rw−(0) | rw−(0) | rw−(1) | rw−(0) |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| **ADC10SAx** | | | | | | | 0 |
| rw−(0) | rw−(0) | rw−(0) | rw−(0) | rw−(0) | rw−(0) | rw−(0) | r0 |

| | | |
|---|---|---|
| **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 17

# SD16_A

The SD16_A module is a single-converter 16-bit, sigma-delta analog-to-digital conversion module with high impedance input buffer. This chapter describes the SD16_A. The SD16_A module is implemented in the MSP430x20x3 devices.

**Topic** **Page**

## 17.1 SD16_A Introduction

The SD16_A module consists of one sigma-delta analog-to-digital converter with an high impedance input buffer and an internal voltage reference. It has up to 8 fully differential multiplexed inputs including a built-in temperature sensor. The converter is based on a second-order oversampling sigma-delta modulator and digital decimation filter. The decimation filter is a comb type filter with selectable oversampling ratios of up to 1024. Additional filtering can be done in software.

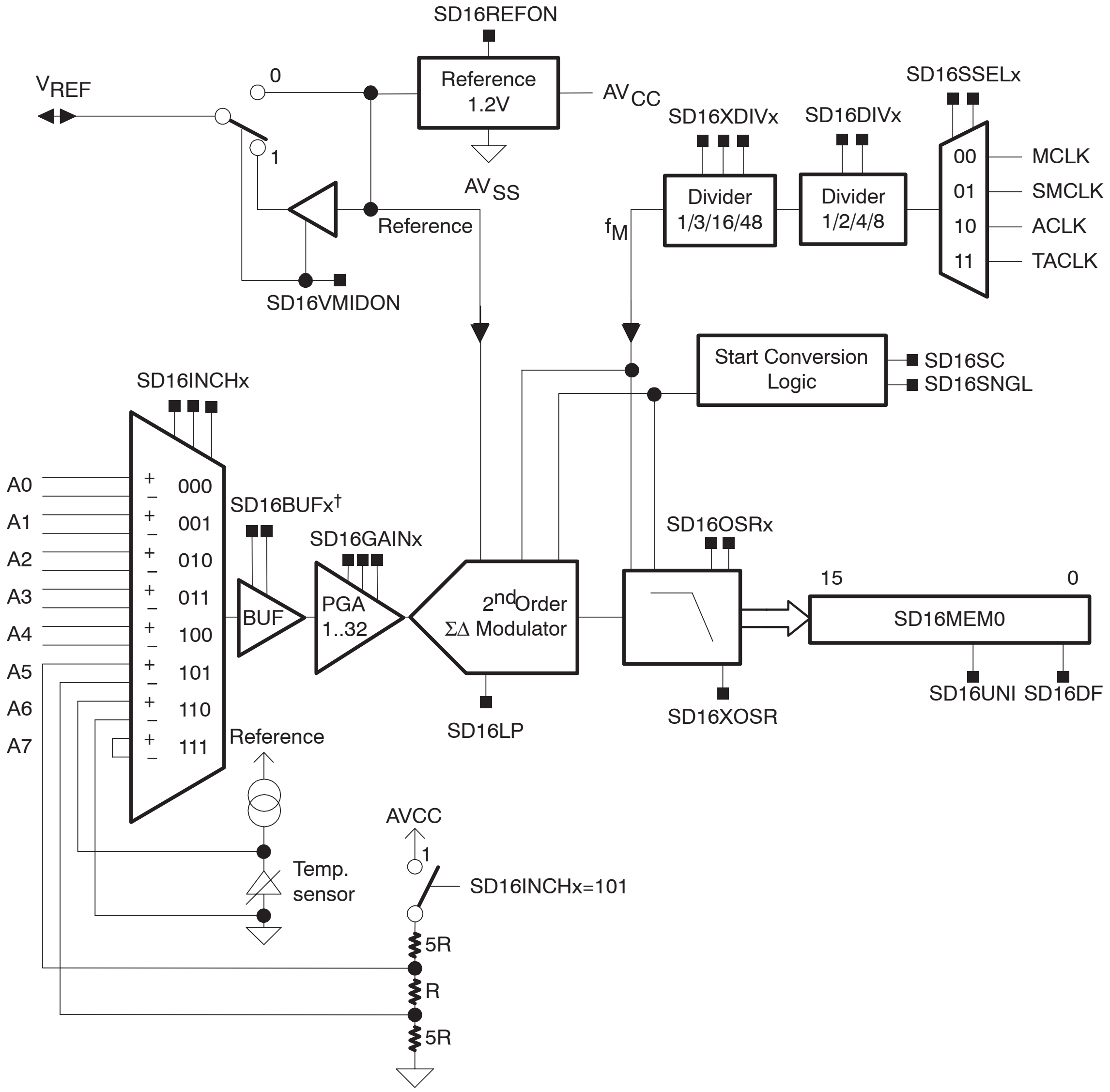
The high impedance input buffer is not implemented in MSP430x20x3 devices.

Features of the SD16_A include:

- 16-bit sigma-delta architecture
- Up to 8 multiplexed differential analog inputs per channel
- Software selectable on-chip reference voltage generation (1.2V)
- Software selectable internal or external reference
- Built-in temperature sensor
- Up to 1.1 MHz modulator input frequency
- High impedance input buffer
- Selectable low-power conversion mode

The block diagram of the SD16_A module is shown in Figure 17−1.

*Figure 17−1. SD16_A Block Diagram*




†Not Implemented in MSP430x20x3 devices

## 17.2 SD16_A Operation

The SD16_A module is configured with user software. The setup and operation of the SD16_A is discussed in the following sections.

### 17.2.1 ADC Core

The analog-to-digital conversion is performed by a 1-bit, second-order sigma-delta modulator. A single-bit comparator within the modulator quantizes the input signal with the modulator frequency $f_M$. The resulting 1-bit data stream is averaged by the digital filter for the conversion result.

### 17.2.2 Analog Input Range and PGA

The full-scale input voltage range for each analog input pair is dependent on the gain setting of the programmable gain amplifier of each channel. The maximum full-scale range is $\pm V_{FSR}$ where $V_{FSR}$ is defined by:

$$V_{FSR} = \frac{V_{REF}/2}{GAIN_{PGA}}$$

For a 1.2V reference, the maximum full-scale input range for a gain of 1 is:

$$\pm\ V_{FSR} = \frac{1.2V/2}{1} = \pm\ 0.6V$$

Refer to the device-specific data sheet for full-scale input specifications.

### 17.2.3 Voltage Reference Generator

The SD16_A module has a built-in 1.2V reference. It is enabled by the SD16REFON bit. When using the internal reference an external 100nF capacitor connected from $V_{REF}$ to $AV_{SS}$ is recommended to reduce noise. The internal reference voltage can be used off-chip when SD16VMIDON = 1. The buffered output can provide up to 1mA of drive. When using the internal reference off-chip, a 470nF capacitor connected from $V_{REF}$ to $AV_{SS}$ is required. See device-specific data sheet for parameters.

An external voltage reference can be applied to the $V_{REF}$ input when SD16REFON and SD16VMIDON are both reset.

### 17.2.4 Auto Power-Down

The SD16_A is designed for low power applications. When the SD16_A is not actively converting, it is automatically disabled and automatically re-enabled when a conversion is started. The reference is not automatically disabled, but can be disabled by setting SD16REFON = 0. When the SD16_A or reference are disabled, they consume no current.

### 17.2.5 Channel Selection

The SD16_A can convert up to 8 differential pair inputs multiplexed into the PGA. Up to five input pairs (A0-A4) are available externally on the device. A resistive divider to measure the supply voltage is available using the A5 multiplexer input. An internal temperature sensor is available using the A6 multiplexer input. Input A7 is a shorted connection between the + and - input pair and can be used to calibrate the offset of the SD16_A input stage.

#### Analog Input Setup

The analog input is configured using the SD16INCTL0 and the SD16AE registers. The SD16INCHx bits select one of eight differential input pairs of the analog multiplexer. The gain for the PGA is selected by the SD16GAINx bits. A total of six gain settings are available. The SD16AEx bits enable or disable the analog input pin. Setting any SD16AEx bit disables the multiplexed digital circuitry for the associated pin. See the device-specific datasheet for pin diagrams.

During conversion any modification to the SD16INCHx and SD16GAINx bits will become effective with the next decimation step of the digital filter. After these bits are modified, the next three conversions may be invalid due to the settling time of the digital filter. This can be handled automatically with the SD16INTDLYx bits. When SD16INTDLY = 00h, conversion interrupt requests will not begin until the 4$^{th}$ conversion after a start condition.

The high impedance input buffer can be enabled using the SD16BUFx bits. The speed settings are selected based on the SD16_A modulator frequency as shown in Table 17−1.

*Table 17−1. High Input Impedance Buffer*

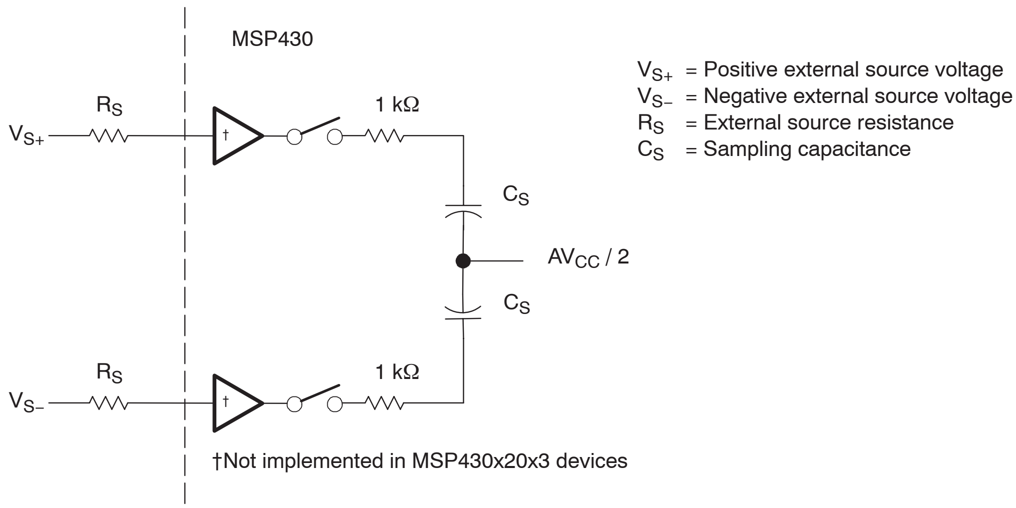
| SD16BUFx | Buffer | SD16 Modulator Frequency $f_M$ |
|---|---|---|
| 00 | Buffer disabled | |
| 01 | Low speed/current | $f_M$ < 200kHz |
| 10 | Medium speed/current | 200kHz < $f_M$ < 700kHz |
| 11 | High speed/current | 700kHz < $f_M$ < 1.1MHz |

An external R-C anti-aliasing filter is recommended for the SD16_A to prevent aliasing of the input signal. The cutoff frequency should be < 10 kHz for a 1 Mhz modulator clock and OSR = 256. The cutoff frequency may set to a lower frequency for applications that have lower bandwidth requirements.

### 17.2.6 Analog Input Characteristics

The SD16_A uses a switched-capacitor input stage that appears as an impedance to external circuitry as shown in Figure 17−2.

*Figure 17−2. Analog Input Equivalent Circuit*




When the buffers are used, $R_S$ does not affect the sampling frequency. However, when the buffers are not used or are not present on the device, the maximum sampling frequency may be calculated from the minimum settling time of the sampling circuit given by:

$$t_S \leqslant (R_S + 1k\Omega) \times C_S \times \ln\left(\frac{V_{REF}}{GAIN \times 2^{17} \times V_{Ax}}\right)$$

where

$$f_S = \frac{1}{2 \times t_S} \qquad \text{and} \qquad V_{Ax} = \left|\frac{AV_{CC}}{2} - V_S\right|$$

$C_S$ varies with the gain setting as shown in Table 17−2.

*Table 17−2. Sampling Capacitance*

| PGA Gain | Sampling Capacitance $C_S$ |
|---|---|
| 1 | 1.25 pF |
| 2, 4 | 2.5 pF |
| 8 | 5 pF |
| 16, 32 | 10 pF |

### 17.2.7 Digital Filter

The digital filter processes the 1-bit data stream from the modulator using a SINC$^3$ comb filter. The transfer function is described in the z-Domain by:

$$H(z) = \left(\frac{1}{OSR} \times \frac{1 - z^{-OSR}}{1 - z^{-1}}\right)^3$$

and in the frequency domain by:

$$H(f) = \left[\frac{sinc\left(OSR\pi\frac{f}{f_M}\right)}{sinc\left(\pi\frac{f}{f_M}\right)}\right]^3 = \left[\frac{1}{OSR} \times \frac{\sin\left(OSR \times \pi \times \frac{f}{f_M}\right)}{\sin\left(\pi \times \frac{f}{f_M}\right)}\right]^3$$

where the oversampling rate, OSR, is the ratio of the modulator frequency $f_M$ to the sample frequency $f_S$. Figure 17−3 shows the filter's frequency response for an OSR of 32. The first filter notch is at $f_S = f_M/OSR$. The notch's frequency can be adjusted by changing the modulator's frequency, $f_M$, using SD16SSELx and SD16DIVx and the oversampling rate using the SD16OSRx and SD16XOSR bits .

The digital filter for each enabled ADC channel completes the decimation of the digital bit-stream and outputs new conversion results to the SD16MEM0 register at the sample frequency $f_S$.

*Figure 17−3. Comb Filter's Frequency Response with OSR = 32*

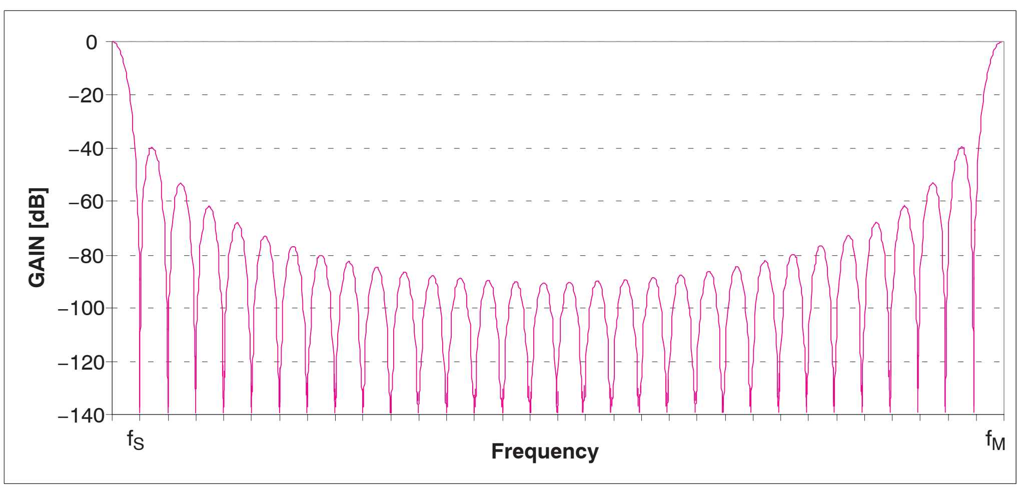
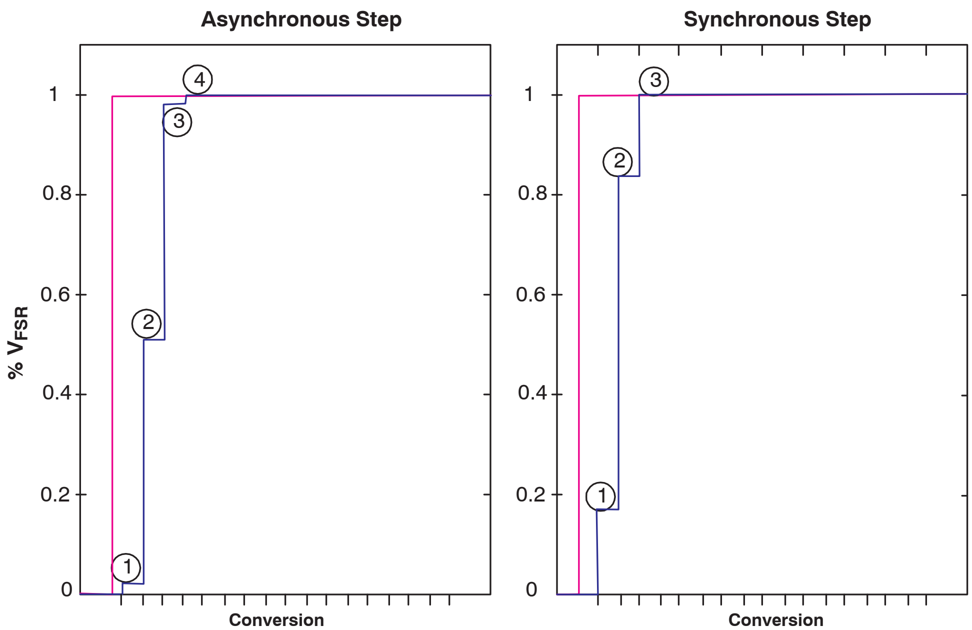


Figure 17−4 shows the digital filter step response and conversion points. For step changes at the input after start of conversion a settling time must be allowed before a valid conversion result is available. The SD16INTDLYx bits can provide sufficient filter settling time for a full-scale change at the ADC input. If the step occurs synchronously to the decimation of the digital filter the valid data will be available on the third conversion. An asynchronous step will require one additional conversion before valid data is available.

*Figure 17−4. Digital Filter Step Response and Conversion Points*



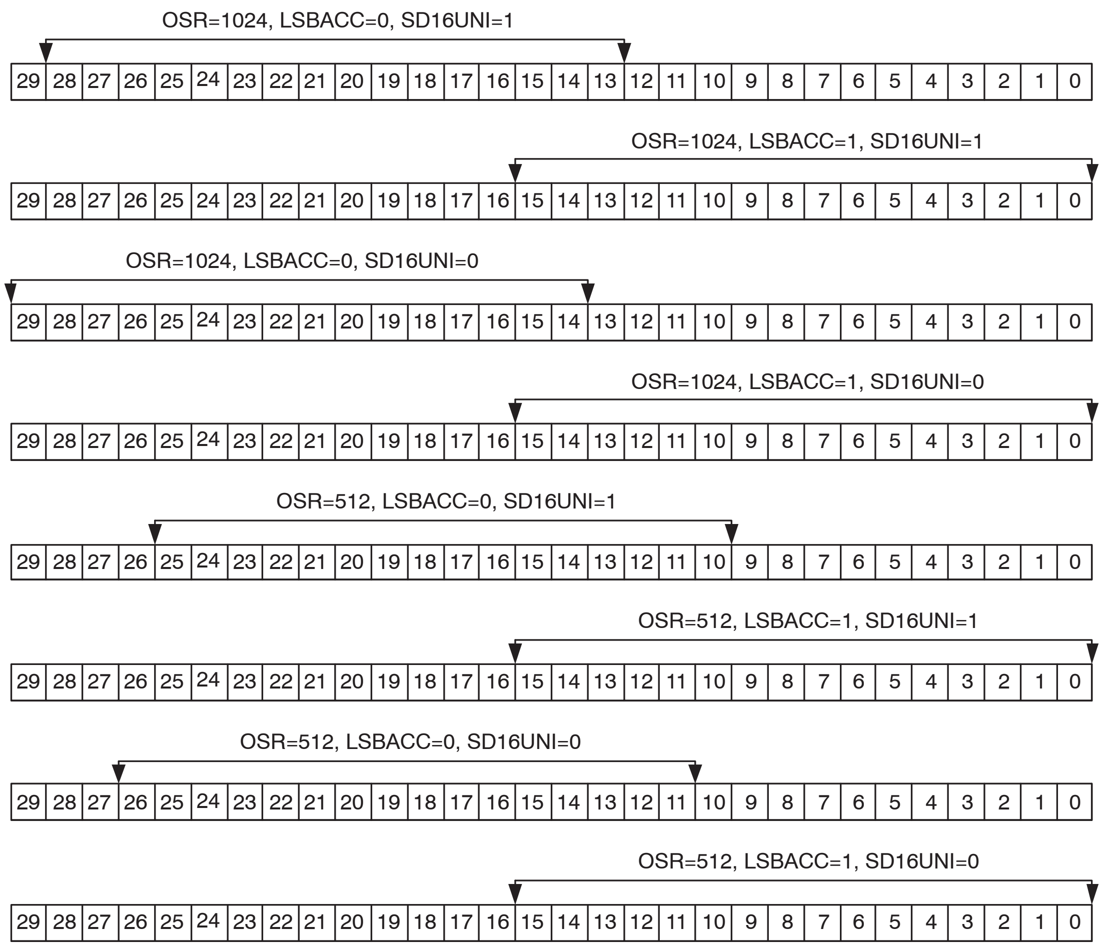
## Digital Filter Output

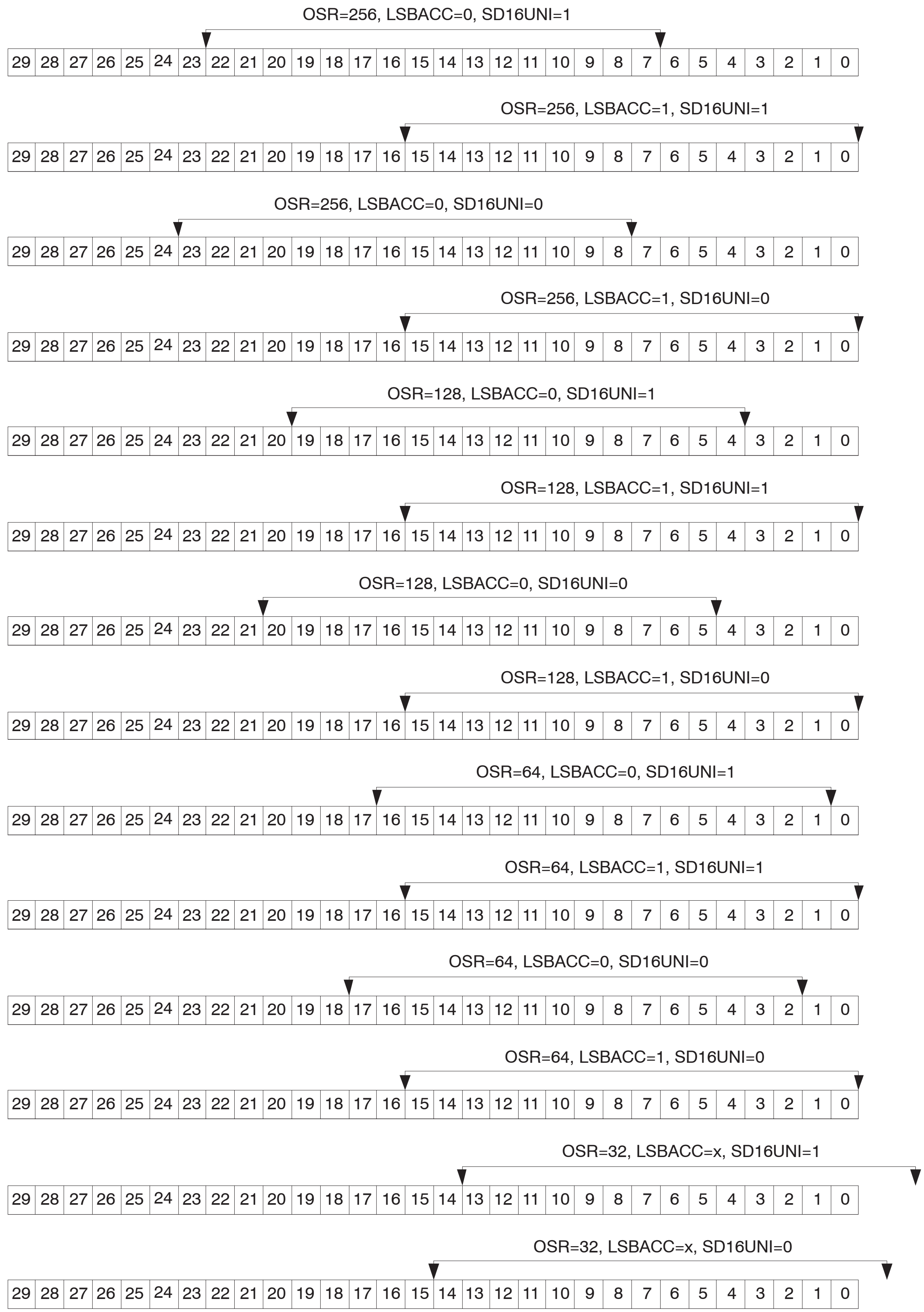
The number of bits output by the digital filter is dependent on the oversampling ratio and ranges from 15 to 30 bits. Figure 17−5 shows the digital filter output and their relation to SD16MEM0 for each OSR, LSBACC, and SD16UNI setting. For example, for OSR = 1024, LSBACC = 0, and SD16UNI = 1, the SD16MEM0 register contains bits 28 − 13 of the digital filter output. When OSR = 32, the one (SD16UNI = 0) or two (SD16UNI=1) LSBs are always zero.

The SD16LSBACC and SD16LSBTOG bits give access to the least significant bits of the digital filter output. When SD16LSBACC = 1 the 16 least significant bits of the digital filter's output are read from SD16MEM0 using word instructions. The SD16MEM0 register can also be accessed with byte instructions returning only the 8 least significant bits of the digital filter output.

When SD16LSBTOG = 1 the SD16LSBACC bit is automatically toggled each time SD16MEM0 is read. This allows the complete digital filter output result to be read with two reads of SD16MEM0. Setting or clearing SD16LSBTOG does not change SD16LSBACC until the next SD16MEM0 access.

*Figure 17−5. Used Bits of Digital Filter Output*




OSR=256, LSBACC=0, SD16UNI=1
OSR=256, LSBACC=1, SD16UNI=1
OSR=256, LSBACC=0, SD16UNI=0
OSR=256, LSBACC=1, SD16UNI=0
OSR=128, LSBACC=0, SD16UNI=1
OSR=128, LSBACC=1, SD16UNI=1
OSR=128, LSBACC=0, SD16UNI=0
OSR=128, LSBACC=1, SD16UNI=0
OSR=64, LSBACC=0, SD16UNI=1
OSR=64, LSBACC=1, SD16UNI=1
OSR=64, LSBACC=0, SD16UNI=0
OSR=64, LSBACC=1, SD16UNI=0
OSR=32, LSBACC=x, SD16UNI=1
OSR=32, LSBACC=x, SD16UNI=0

### 17.2.8 Conversion Memory Register: SD16MEM0

The SD16MEM0 register is associated with the SD16_A channel. Conversion results are moved to the SD16MEM0 register with each decimation step of the digital filter. The SD16IFG bit is set when new data is written to SD16MEM0. SD16IFG is automatically cleared when SD16MEM0 is read by the CPU or may be cleared with software.

#### Output Data Format

The output data format is configurable in two's complement, offset binary or unipolar mode as shown in Table 17−3.The data format is selected by the SD16DF and SD16UNI bits.

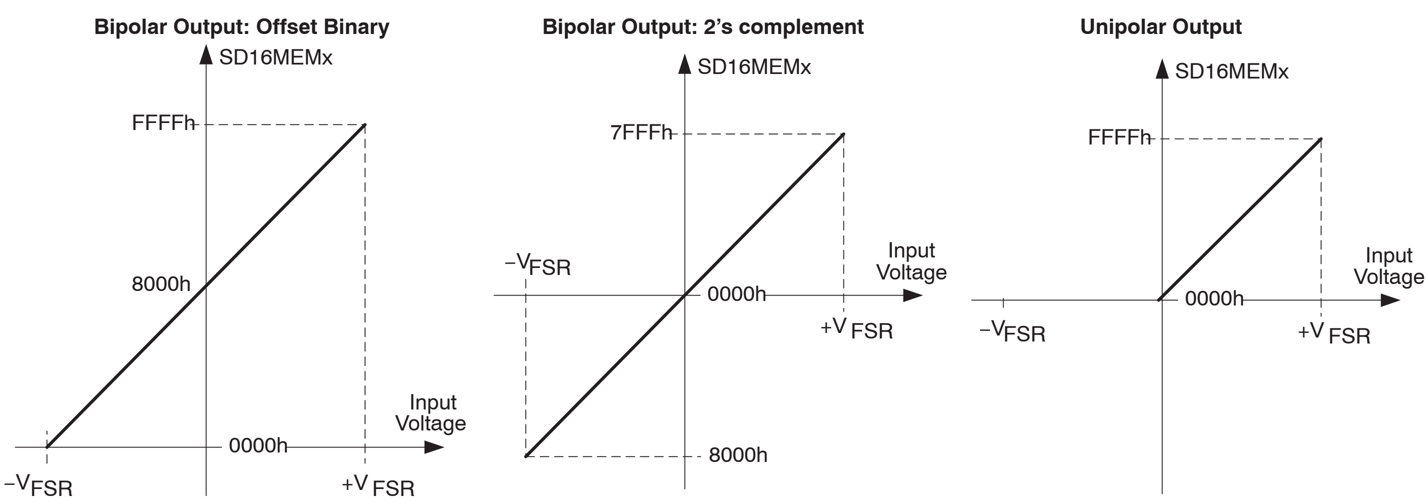
*Table 17−3.Data Format*

| SD16UNI | SD16DF | Format | Analog Input | SD16MEM0† | Digital Filter Output (OSR =256) |
|---|---|---|---|---|---|
| 0 | 0 | Bipolar Offset Binary | +FSR | FFFF | FFFFFF |
| | | | ZERO | 8000 | 800000 |
| | | | −FSR | 0000 | 000000 |
| 0 | 1 | Bipolar Two's compliment | +FSR | 7FFF | 7FFFFF |
| | | | ZERO | 0000 | 000000 |
| | | | −FSR | 8000 | 800000 |
| 1 | 0 | Unipolar | +FSR | FFFF | FFFFFF |
| | | | ZERO | 0000 | 800000 |
| | | | −FSR | 0000 | 000000 |

† Independent of SD16OSRx and SD16XOSR settings; SD16LSBACC = 0.

Figure 17−6 shows the relationship between the full-scale input voltage range from $-V_{FSR}$ to $+V_{FSR}$ and the conversion result. The data formats are illustrated.

*Figure 17−6. Input Voltage vs. Digital Output*



### 17.2.9 Conversion Modes

The SD16_A module can be configured for two modes of operation, listed in Table 17−4. The SD16SNGL bit selects the conversion mode.

*Table 17−4. Conversion Mode Summary*

| SD16SNGL | Mode | Operation |
|---|---|---|
| 1 | Single conversion | The channel is converted once. |
| 0 | Continuous conversion | The channel is converted continuously. |

#### Single Conversion

Setting the SD16SC bit of the channel initiates one conversion on that channel when SD16SNGL = 1. The SD16SC bit will automatically be cleared after conversion completion.

Clearing SD16SC before the conversion is completed immediately stops conversion of the channel, the channel is powered down and the corresponding digital filter is turned off. The value in SD16MEM0 can change when SD16SC is cleared. It is recommended that the conversion data in SD16MEM0 be read prior to clearing SD16SC to avoid reading an invalid result.

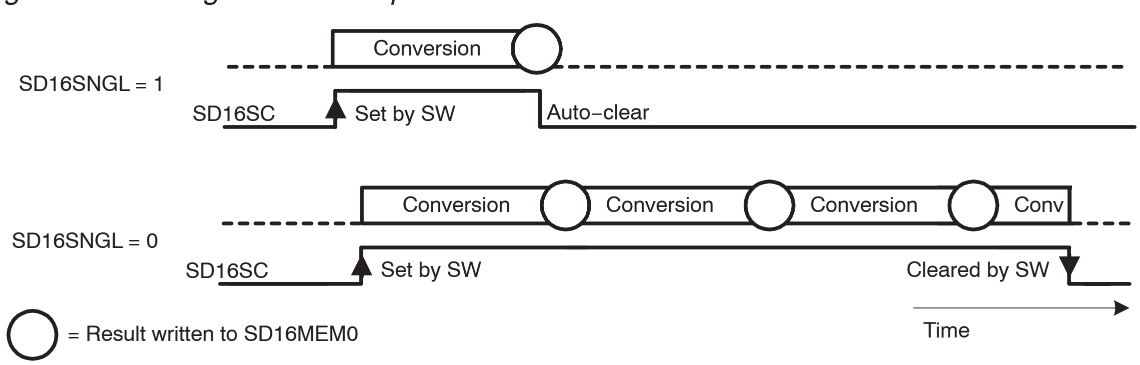
#### Continuous Conversion

When SD16SNGL = 0 continuous conversion mode is selected. Conversion of the channel will begin when SD16SC is set and continue until the SD16SC bit is cleared by software.

Clearing SD16SC immediately stops conversion of the selected channel, the channel is powered down and the corresponding digital filter is turned off. The value in SD16MEM0 can change when SD16SC is cleared. It is recommended that the conversion data in SD16MEM0 be read prior to clearing SD16SC to avoid reading an invalid result.

Figure 17−7 shows conversion operation.

*Figure 17−7. Single Channel Operation*

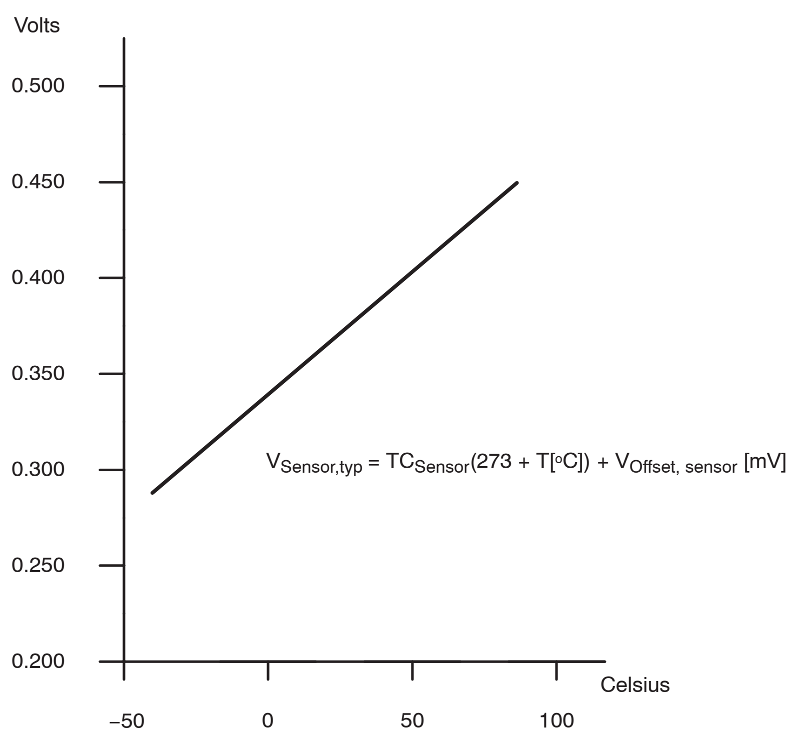


### 17.2.10 Using the Integrated Temperature Sensor

To use the on-chip temperature sensor, the user selects the analog input channel SD16INCHx = 110 and sets SD16REFON = 1. Any other configuration is done as if an external channel was selected, including SD16INTDLYx and SD16GAINx settings. Because the internal reference must be on to use the temperature sensor, it is not possible to use an external reference for the conversion of the temperature sensor voltage. Also, the internal reference will be in contention with any used external reference. In this case, the SD16VMIDON bit may be set to minimize the affects of the contention on the conversion.

The typical temperature sensor transfer function is shown in Figure 17−8. When switching inputs of an SD16_A channel to the temperature sensor, adequate delay must be provided using SD16INTDLYx to allow the digital filter to settle and assure that conversion results are valid. The temperature sensor offset error can be large, and may need to be calibrated for most applications. See device-specific data sheet for temperature sensor parameters.

*Figure 17−8. Typical Temperature Sensor Transfer Function*



### 17.2.11 Interrupt Handling

The SD16_A has 2 interrupt sources for its ADC channel:

- ❑ SD16IFG
- ❑ SD16OVIFG

The SD16IFG bit is set when the SD16MEM0 memory register is written with a conversion result. An interrupt request is generated if the corresponding SD16IE bit and the GIE bit are set. The SD16_A overflow condition occurs when a conversion result is written to SD16MEM0 location before the previous conversion result was read.

#### SD16IV, Interrupt Vector Generator

All SD16_A interrupt sources are prioritized and combined to source a single interrupt vector. SD16IV is used to determine which enabled SD16_A interrupt source requested an interrupt. The highest priority SD16_A interrupt request that is enabled generates a number in the SD16IV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled SD16_A interrupts do not affect the SD16IV value.

Any access, read or write, of the SD16IV register has no effect on the SD16OVIFG or SD16IFG flags. The SD16IFG flags are reset by reading the SD16MEM0 register or by clearing the flags in software. SD16OVIFG bits can only be reset with software.

If another interrupt is pending after servicing of an interrupt, another interrupt is generated. For example, if the SD16OVIFG and one or more SD16IFG interrupts are pending when the interrupt service routine accesses the SD16IV register, the SD16OVIFG interrupt condition is serviced first and the corresponding flag(s) must be cleared in software. After the RETI instruction of the interrupt service routine is executed, the highest priority SD16IFG pending generates another interrupt request.

#### Interrupt Delay Operation

The SD16INTDLYx bits control the timing for the first interrupt service request for the corresponding channel. This feature delays the interrupt request for a completed conversion by up to four conversion cycles allowing the digital filter to settle prior to generating an interrupt request. The delay is applied each time the SD16SC bit is set or when the SD16GAINx or SD16INCHx bits for the channel are modified. SD16INTDLYx disables overflow interrupt generation for the channel for the selected number of delay cycles. Interrupt requests for the delayed conversions are not generated during the delay.

## 17.3 SD16_A Registers

The SD16_A registers are listed in Table 17−5:

*Table 17−5. SD16_A Registers*

| Register | Short Form | Register Type | Address | Initial State |
|---|---|---|---|---|
| SD16_A Control | SD16CTL | Read/write | 0100h | Reset with PUC |
| SD16_A Interrupt Vector | SD16IV | Read/write | 0110h | Reset with PUC |
| SD16_A Channel 0 Control | SD16CCTL0 | Read/write | 0102h | Reset with PUC |
| SD16_A Conversion Memory | SD16MEM0 | Read/write | 0112h | Reset with PUC |
| SD16_A Input Control | SD16INCTL0 | Read/write | 0B0h | Reset with PUC |
| SD16_A Analog Enable | SD16AE | Read/write | 0B7h | Reset with PUC |

## SD16CTL, SD16_A Control Register

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | SD16XDIVx | | | SD16LP |
| r0 | r0 | r0 | r0 | rw–0 | rw–0 | rw–0 | rw–0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SD16DIVx | | SD16SSELx | | SD16 VMIDON | SD16 REFON | SD16OVIE | Reserved |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | r0 |

| | | |
|---|---|---|
| **Reserved** | Bits 15-12 | Reserved |
| **SD16XDIVx** | Bits 11-9 | SD16_A clock divider<br>000 /1<br>001 /3<br>010 /16<br>011 /48<br>1xx Reserved |
| **SD16LP** | Bit 8 | Low power mode. This bit selects a reduced speed, reduced power mode<br>0 Low-power mode is disabled<br>1 Low-power mode is enabled. The maximum clock frequency for the SD16_A is reduced. |
| **SD16DIVx** | Bits 7-6 | SD16_A clock divider<br>00 /1<br>01 /2<br>10 /4<br>11 /8 |
| **SD16SSELx** | Bits 5-4 | SD16_A clock source select<br>00 MCLK<br>01 SMCLK<br>10 ACLK<br>11 External TACLK |
| **SD16 VMIDON** | Bit 3 | $V_{MID}$ buffer on<br>0 Off<br>1 On |
| **SD16 REFON** | Bit 2 | Reference generator on<br>0 Reference off<br>1 Reference on |
| **SD16OVIE** | Bit 1 | SD16_A overflow interrupt enable. The GIE bit must also be set to enable the interrupt.<br>0 Overflow interrupt disabled<br>1 Overflow interrupt enabled |
| **Reserved** | Bit 0 | Reserved |

## SD16CCTL0, SD16_A Control Register 0

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | SD16BUFx† | | SD16UNI | SD16XOSR | SD16SNGL | SD16OSRx | |
| r0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SD16 LSBTOG | SD16 LSBACC | SD16 OVIFG | SD16DF | SD16IE | SD16IFG | SD16SC | Reserved |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | r–0 |

†Reserved in MSP430x20x3 devices

**Reserved** Bit 15 Reserved

**SD16BUFx** Bits 14–13 High impedance input buffer mode
00 Buffer disabled
01 Slow speed/current
10 Medium speed/current
11 High speed/current

**SD16UNI** Bit 12 Unipolar mode select
0 Bipolar mode
1 Unipolar mode

**SD16XOSR** Bit 11 Extended oversampling ratio. This bit, along with the SD16OSRx bits, select the oversampling ratio. See SD16OSRx bit description for settings.

**SD16SNGL** Bit 10 Single conversion mode select
0 Continuous conversion mode
1 Single conversion mode

**SD16OSRx** Bits 9-8 Oversampling ratio
When SD16XOSR = 0
00 256
01 128
10 64
11 32
When SD16XOSR = 1
00 512
01 1024
10 Reserved
11 Reserved

**SD16 LSBTOG** Bit 7 LSB toggle. This bit, when set, causes SD16LSBACC to toggle each time the SD16MEM0 register is read.
0 SD16LSBACC does not toggle with each SD16MEM0 read
1 SD16LSBACC toggles with each SD16MEM0 read

| | | |
|---|---|---|
| **SD16 LSBACC** | Bit 6 | LSB access. This bit allows access to the upper or lower 16-bits of the SD16_A conversion result.<br>0 SD16MEMx contains the most significant 16-bits of the conversion.<br>1 SD16MEMx contains the least significant 16-bits of the conversion. |
| **SD16OVIFG** | Bit 5 | SD16_A overflow interrupt flag<br>0 No overflow interrupt pending<br>1 Overflow interrupt pending |
| **SD16DF** | Bit 4 | SD16_A data format<br>0 Offset binary<br>1 2's complement |
| **SD16IE** | Bit 3 | SD16_A interrupt enable<br>0 Disabled<br>1 Enabled |
| **SD16IFG** | Bit 2 | SD16_A interrupt flag. SD16IFG is set when new conversion results are available. SD16IFG is automatically reset when the corresponding SD16MEMx register is read, or may be cleared with software.<br>0 No interrupt pending<br>1 Interrupt pending |
| **SD16SC** | Bit 1 | SD16_A start conversion<br>0 No conversion start<br>1 Start conversion |
| **Reserved** | Bit 0 | Reserved |

## SD16INCTL0, SD16_A Input Control Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SD16INTDLYx | | SD16GAINx | | | SD16INCHx | | |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 |

**SD16 INTDLYx** Bits 7-6
Interrupt delay generation after conversion start. These bits select the delay for the first interrupt after conversion start.
00 Fourth sample causes interrupt
01 Third sample causes interrupt
10 Second sample causes interrupt
11 First sample causes interrupt

**SD16GAINx** Bits 5-3
SD16_A preamplifier gain
000 x1
001 x2
010 x4
011 x8
100 x16
101 x32
110 Reserved
111 Reserved

**SD16INCHx** Bits 2-0
SD16_A channel differential pair input
000 A0
001 A1
010 A2
011 A3
100 A4
101 A5– ($AV_{CC}$ – $AV_{SS}$) / 11
110 A6- Temperature Sensor
111 A7- Short for PGA offset measurement

## SD16MEM0, SD16_A Conversion Memory Register

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Conversion Results | | | | | | | |
| r | r | r | r | r | r | r | r |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Conversion Results | | | | | | | |
| r | r | r | r | r | r | r | r |

| | | |
|---|---|---|
| **Conversion Result** | Bits 15-0 | Conversion Results. The SD16MEMx register holds the upper or lower 16-bits of the digital filter output, depending on the SD16LSBACC bit. |

## SD16AE, SD16_A Analog Input Enable Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SD16AE7 | SD16AE6 | SD16AE5 | SD16AE4 | SD16AE3 | SD16AE2 | SD16AE1 | SD16AE0 |
| rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 | rw–0 |

| | | |
|---|---|---|
| **SD16AEx** | Bits 7-0 | SD16_A analog enable<br>0 External input disabled. Negative inputs are internally connected to VSS.<br>1 External input enabled. |

## SD16IV, SD16_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** | **SD16IVx** | | | | **0** |
| r0 | r0 | r0 | r–0 | r–0 | r–0 | r–0 | r0 |

**SD16IVx** Bits 15-0 SD16_A interrupt vector value

| SD16IV Contents | Interrupt Source | Interrupt Flag | Interrupt Priority |
|---|---|---|---|
| 000h | No interrupt pending | – | |
| 002h | SD16MEMx overflow | SD16CCTLx SD16OVIFG | Highest |
| 004h | SD16_A Interrupt | SD16CCTL0 SD16IFG | |
| 006h | Reserved | – | |
| 008h | Reserved | – | |
| 00Ah | Reserved | – | |
| 00Ch | Reserved | – | |
| 00Eh | Reserved | – | |
| 010h | Reserved | – | Lowest |