MSP430x5xx Family

User's Guide

Literature Number: SLAU208

June 2008

Contents

List of Figures

List of Tables

Read This First

About This Manual

This manual describes the modules and peripherals of the MSP430x5xx family of devices. Each description presents the module or peripheral in a general sense. Not all features and functions of all modules or peripherals may be 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 parameters differ from device to device. The user should consult the device-specific data sheet 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
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 FLL+ Module
dst	Destination; see RISC 16-Bit CPU
FLL	Frequency Locked Loop; see FLL+ Module
GIE Modes	General Interrupt Enable; see System Resets Interrupts and Operating
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; also named PM for Power Mode
MAB	Memory Address Bus
MCLK	Master Clock
MDB	Memory Data Bus
MSB	Most-Significant Bit
MSD	Most-Significant Digit
NMI	(Non)-Maskable Interrupt; see System Resets Interrupts and Operating Modes; also split to UNMI and SNMI
PC	Program Counter; see RISC 16-Bit CPU
PM	Power Mode See; system Resets Interrupts and Operating Modes
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; see System Resets, Interrupts, and Operating Modes
SMCLK	Sub-System Master Clock
SNMI	System NMI; see System Resets, Interrupts, and Operating Modes
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
UNMI	User NMI; see System Resets, Interrupts, and Operating Modes
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
-[0],-[1]	Condition after BOR
-{0},-{1}	Condition after Brownout

Chapter 1
SLAU208–June 2008

System Resets, Interrupts, and Operating Modes, System Control Module (SYS)

The System Control Module (SYS) is integrated into various devices with different feature sets. It provides public services like Device-ID and TI-private services.

The following list shows the basic feature set of SYS.

- Power on reset (BOR/POR) handling
- Power up clear (PUC) handling
- NMI (SNMI/UNMI) event source selection and management
- Address decoding
- Providing an user data exchange mechanism via the JTAG Mailbox (JMB)
- Boot strap loader (BSL) entry mechanism
- Configuration management (device descriptors)
- Providing interrupt vector generators for Reset and NMIs
- Watch dog timer (WDT_A)

Topic **Page**

1.1 System Control Module Introduction

The SYS module is responsible for interaction between various modules throughout the system. The functions SYS provides for are not inherent to the modules themselves. Address decoding, bus arbitration, interrupt event collection/prioritization, and reset generation are some of the many functions that SYS provides.

1.2 Principle of Operation

The SYS module provides a series of services that can be used by the application program. Some of these services however can be locked to fulfill code protection requirements. Some bit fields used for common functions are defined as reserved when not implemented on a particular device; this allows a maximum of compatibility among the devices within the MSP430 microcontroller family with SYS modules.

1.2.1 Device Descriptor Table

Each MSP430 provides a data structure in memory that allows an unambiguous identification of the device. Device adaptive SW-tools and libraries need a more detailed description of the available modules on a given device. The SYS module provides this information and can be used by device adaptive SW tools and libraries to clearly identify a particular device and all modules/capabilities contained within it. The validity of the device descriptor can be verified by CRC (cyclic redundancy check).

1.2.1.1 Identifying the Device type

The value read at address location 00FF0h identifies the family branch of the device. All values starting with 80h indicate a hierarchical structure consisting of the info block and a TLV (tag-length-value) structure containing the various descriptors. The info block contains the device ID, die revisions, SW revisions of boot code, and other manufacturer and tool related information. The descriptors contains information about the available peripherals, their subtypes and addresses. This allows to build adaptive HW drivers for operating systems.

Any other value than 80h read at address location 00FF0h indicates the device is of an older family and contains a flat descriptor beginning at location 0FF0h.

1.2.1.2 MSP430 Calibration Descriptors

The MSP430 features a common data structure for calibration data. This structure starts with a predefined header of constant length that simplifies extracting some basic information like Chip_ID, hardware revisions, etc., and is followed by a flexible TLV list containing various calibration information required by the device.

1.2.2 Boot Code

The boot code will always be executed after a BOR. The boot performs calibration of the oscillator and reference voltages. In addition, it checks for existing signatures (predefined data pattern) that indicate the presence of a customer definable boot strap loader (BSL).

1.2.3 Boot Strap Loader (BSL)

The MSP430 bootstrap loader (BSL) is software that is executed after startup when a certain bootstrap loader entry condition is applied. A BSL enables the user to communicate with embedded memory in the MSP430 microcontroller during the prototyping phase, final production, and in service. All memory mapped resources, the programmable memory (flash memory), the data memory (RAM) and the peripherals, can be modified by the BSL as required. The user can define its own BSL-Code for flash based devices and protect it against erasure and unintentional or unauthorized access.

A basic BSL program is provided by TI. This supports the commonly used UART protocol with RS232 interfacing, allowing flexible use of both hardware and software. To use the bootstrap loader, a specific BSL entry sequence has to be applied to specific device pins. An added sequence of commands initiates the desired function. A boot loading session can be exited by continuing operation at a defined user program address, or by the reset condition. Access to the MSP430 memory via the bootstrap loader is protected against misuse by a user-defined password.

1.2.4 JTAG Mailbox System (JMB)

The SYS module provides the capability to exchange user data via the regular JTAG test/debug interface. The idea behind the JTAG mailbox system is to have a direct interface to the CPU during debugging, programming and test that is identical for all ‘430 devices of this family and uses only few or no user application resources. The JTAG interface was chosen because it is available on all ‘430 devices and is a dedicated resource for debugging, programming and test.

Applications of the JTAG Mailbox System are:

- Fast flash programming
- Providing entry password for software security fuse
- Run-time data exchange (RTDX)

1.3 Memory Map–Uses and Abilities

This memory map represents the MSP430F5438 device. Though the address ranges differs from device to device, overall behavior remains the same.

Address	Name/Purpose	Segment erase by user able from	Block erase by user able from	Mass erase by user able from	Always able to access PMM registers from[(1)]; Mass erase by user able from	Protectable for read/write accesses	Generates PUC on fetch access	Can Generates NMI on read/write/fetch
Address	Name/Purpose	Properties						
00000h-00FFFh	Peripherals with gaps							
00000h-000FFh	Reserved for system-extension							
00100h-00FEFh	Peripherals						x	
00FF0h-00FF3h	Descriptor type						x	
00FF4h-00FF7h	start address of descriptor structure						x	
01000h-011FFh	BSL_Seg_0	x				x		
01200h-013FFh	BSL_Seg_1	x				x		
01400h-015FFh	BSL_Seg_2	x				x		
01600h-017FFh	BSL_Seg_3	x			x	x		
017FCh-017FFh	BSL Signature Location							
01800h-0187Fh	User_Info_D	x						
01880h-018FFh	User_Info_C	x						
01900h-0197Fh	User_Info_B	x						
01980h-019FFh	User_Info_A	x						
01A00h-01A7Fh	Calibration					x	x	
01A80h-01AFFh	Info-Bock, Device ID, Descriptor							
01C00h-05BFFh	RAM 16k							
05B80-05BFFh	Alternate Interrupt Vectors							
05C00h-0FFFFh	Program_lo (64-x^5)k	x	x[(1)]	x				
0FF7Ch-0FF7Fh	Application Signature Location							
0FF80h-0FFFFh	Interrupt Vectors							
10000h-45BFFh	Program_hi (192+x^5)k	x	x	x				
45C00h-FFFFFh	Vacant							x[(2)]

[(1)] Access rights are separately programmable for SYS and PMM.
[(2)] On vacant memory space, the value 03FFFh will be driven on the data bus.

1.3.1 Vacant Memory Space

Accesses to vacant memory space will generate a NMI interrupt. Reads from vacant memory results in the value 3FFFh. In the case of a fetch, this is taken as JMP $. Fetch accesses from vacant peripheral space will result in a PUC. After the Boot code is executed, it behaves like vacant memory space and causes a NMI on access.

1.3.2 JTAG Lock Mechanism

After a BOR the memory location 01BFEh will be taken as the reset-vector to start the boot code. The Boot code evaluates the signatures of an optional boot strap loader (BSL) and the application is able to lock or unlock JTAG for debugging, all that depending on the signatures.

1.3.3 SYS Interrupt Vector Generators

The SYS module collects all user NMI (UNMI) sources, system NMI (SNMI) sources, and BOR/POR/PUC sources of all the other modules. They are combined into three interrupt vectors. The interrupt vector registers SYSRSTIV, SYSSNIV, SYSUNIV are used to determine which flags requested an interrupt or a BOR/POR/PUC reset. The interrupt with the highest priority of a group, when enabled, generates a number in the corresponding SYSRSTIV, SYSSNIV, SYSUNIV register. This number can be directly added to the program counter, causing a branch to the appropriate portion of the interrupt service routine. Disabled interrupts do not affect the SYSRSTIV, SYSSNIV, SYSUNIV values. A read access, read to the SYSRSTIV, SYSSNIV, SYSUNIV register automatically resets the highest pending interrupt flag of that register. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. A write access to the SYSRSTIV, SYSSNIV, SYSUNIV register automatically resets all pending interrupt flags of the group.

1.3.3.1 SYSSNIV Software Example

The following software example shows the recommended use of SYSSNIV. The SYSSNIV value is added to the PC to automatically jump to the appropriate routine. For SYSRSTIV and SYSUNIV a similar SW approach can be chosen. The following is an example for a generic MSP430x5xx device. Vectors can change in priority for a given device. The device specific data sheet should be referenced for the vector locations. All vectors should be coded symbolically to allow for easy portability of code.

```
SNI_ISR:    ADD      &SYSSNIV,PC ; Add offset to jump table
         RETI              ; Vector 0: No interrupt
         JMP      SVML_ISR       ; Vector 2: SVMLIFG
         JMP      SVMH_ISR       ; Vector 4: SVMHIFG
         JMP      DLYL_ISR       ; Vector 6: DLYLIFG
         JMP      DLYH_ISR       ; Vector 8: DLYHIFG
         JMP      VMA_ISR        ; Vector 10: VMAIFG
         JMP      JMBI_ISR       ; Vector 12: JMBINIFG
JMBO_ISR:                           ; Vector 14: JMBOUTIFG
        ...                         ; Task_E starts here
        RETI                        ; Return
SVML_ISR:                           ; Vector 2
        ...                         ; Task_2 starts here
        RETI                        ; Return
SVMH_ISR:                           ; Vector 4
        ...                   ; Task_4 starts here
        RETI                        ; Return
DELL_ISR:                           ; Vector 6
        ...                         ; Task_6 starts here
        RETI                        ; Return
DELH_ISR:                           ; Vector 8
        ...                         ; Task_8 starts here
        RETI                        ; Return
VMA_ISR:                               ; Vector A
       ...                          ; Task_A starts here
          RETI                      ; Return
JMBI_ISR:                           ; Vector C
       ...                          ; Task_C starts here
       RETI                         ; Return
```

1.4 Interrupts

Interrupt priorities are fixed and defined by the arrangement of the modules in the connection chain as shown in Figure 1-1. 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

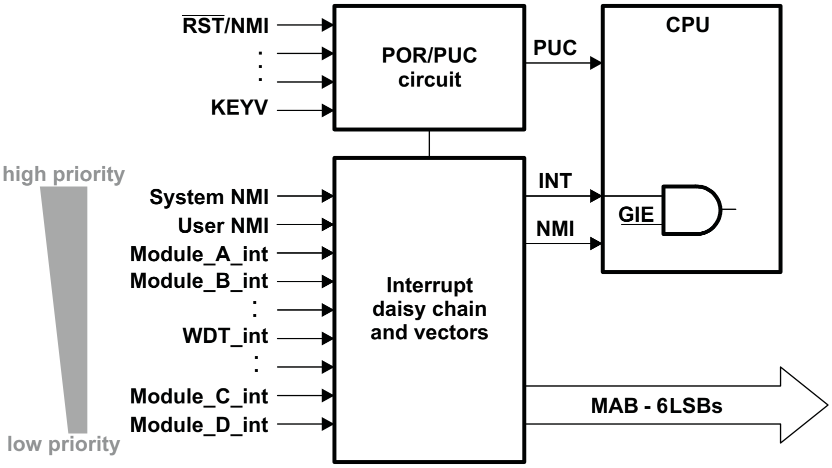


Figure 1-1. Interrupt Priority

1.4.1 (Non)-Maskable Interrupts (NMI)

The MSP430x5xx family supports two levels of NMI interrupts, system NMI (SNMI) and user NMI (UNMI). In general, (Non)-maskable NMI interrupts are not masked by the general interrupt enable bit (GIE). The user NMI sources are enabled by individual interrupt enable bits (NMIIE, ACCVIE, OFIE). When a user NMI interrupt is accepted, other NMIs of that level are automatically disabled to prevent nesting of consecutive NMIs of the same level. Program execution begins at the address stored in the (non)-maskable interrupt vector as shown in Table 1-2. To allow software backward compatibility to users of earlier MSP430 families, the software may, but does not need to re-enable user NMI sources. The block diagram for NMI sources is shown in Figure 1-2.

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

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

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

- Power Management Module (PMM) SVML/SVMH supply voltage fault
- PMM time out
- Vacant memory access
- JTAG mailbox event

1.4.2 SNMI Timing

Consecutive system NMIs that are fired in a higher rate than they can be handled (interrupt storm) allow the main program to execute one instruction after the system NMI handler is finished with an RETI instruction, before the system NMI handler is executed again. Consecutive system NMIs are not interrupted by user NMIs in this case. This avoids a blocking behavior on high SNMI rates.

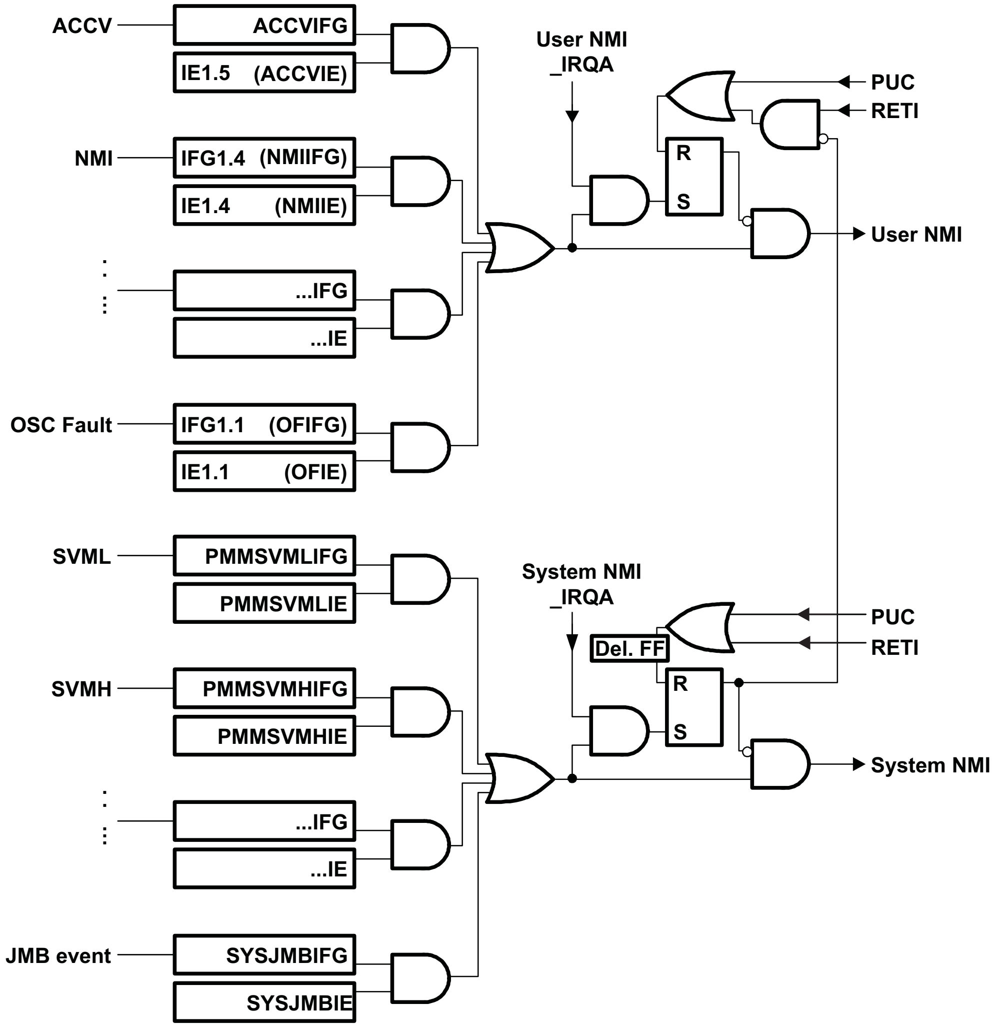


Figure 1-2. NMI Interrupts With Reentrance Protection

1.4.3 Maskable Interrupts

Maskable interrupts are caused by peripherals with interrupt capability. 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 its respective module chapter of this manual.

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.

1.4.4.1 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 1-3. 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.

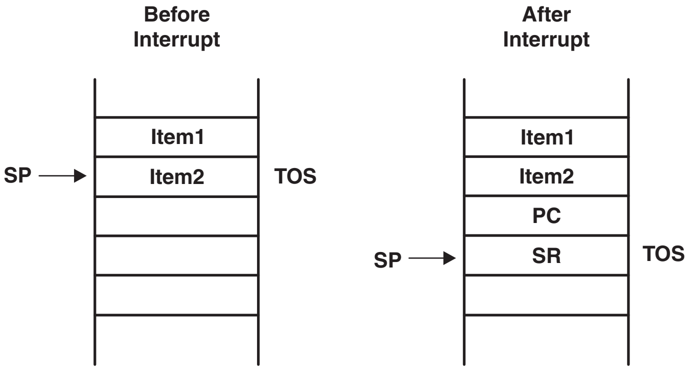


Figure 1-3. Interrupt Processing

1.4.4.2 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 1-4.

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.

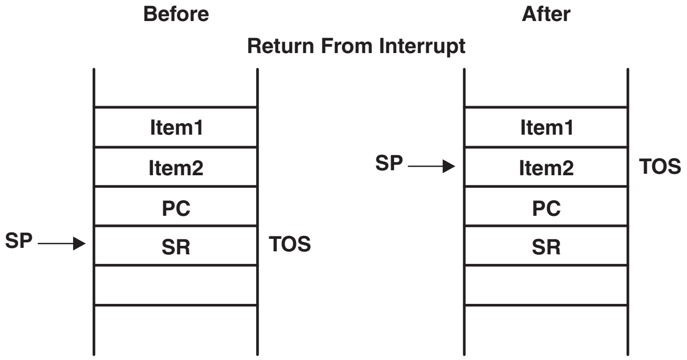


Figure 1-4. Return From Interrupt

1.4.4.3 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.

1.5 Operating Modes

The MSP430 family is designed for ultralow-power applications and uses different operating modes shown in Figure 1-5.

The operating modes take into account three different needs:

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

The low-power modes LPM0 through LPM4 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.

When LPM5 is entered, the voltage regulator of the Power Management Module (PMM) is disabled. All RAM and register contents are lost, as well as, I/O configuration. Wake-up is possible via a power sequence or an RST/NMI event. On some devices, wake-up from I/O is also possible. Please refer to the device specific datasheet.

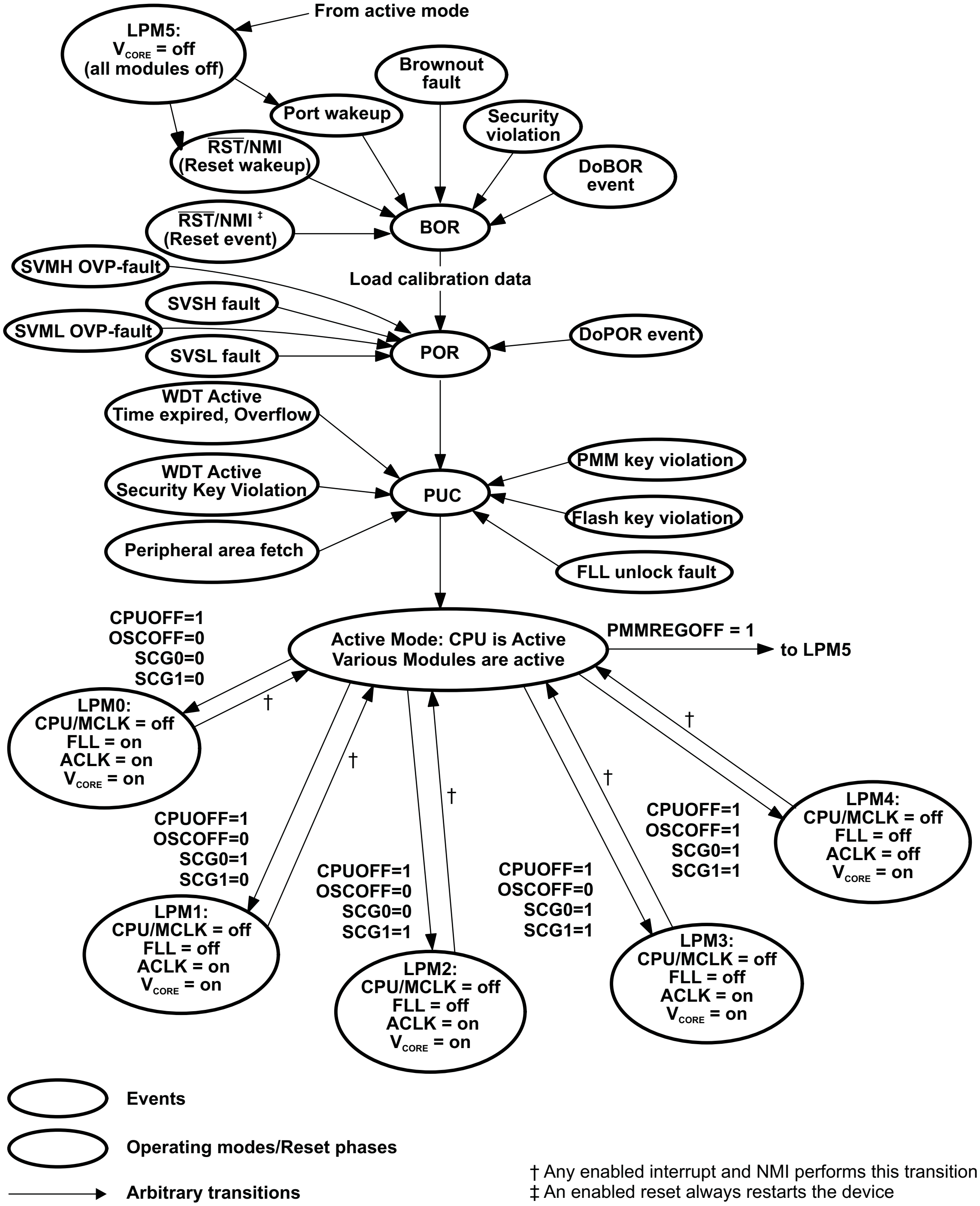


Figure 1-5. Operation Modes

SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU, MCLK are active. ACLK is active. SMCLK optionally active (SMCLKOFF = 0).
0	0	0	1	LPM0	CPU, MCLK are disabled. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO enabled if sources ACLK, MCLK, or SMCLK (SMCLKOFF = 0). FLL enabled if DCO enabled.
0	1	0	1	LPM1	CPU, MCLK are disabled. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO enabled if sources ACLK or SMCLK (SMCLKOFF = 0). FLL disabled.
1	0	0	1	LPM2	CPU, MCLK are disabled. ACLK is active. SMCLK is disabled. DCO enabled if sources ACLK. FLL disabled.
1	1	0	1	LPM3	CPU, MCLK are disabled. ACLK is active. SMCLK is disabled. DCO enabled if sources ACLK. FLL disabled.
1	1	1	1	LPM4	CPU and all clocks disabled
1	1	1	1	LPM5	When PMMREGOFF = 1, regulator disabled. No memory retention.

1.5.1 Entering and Exiting Low-Power Modes

An enabled interrupt event wakes the MSP430 from low-power operating modes LPM0 through LPM4. LPM5 exit is only possible via a power cycle or a RST/NMI event or wakeup from I/O on when available on some devices. The program flow entering and exiting LPM0 through LPM4 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
```

```
; Enter LPM4 Example
   BIS   #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR     ; Enter LPM4
;  ...                                       ; Program stops here
;
; Exit LPM4 Interrupt Service Routine
   BIC   #CPUOFF+OSCOFF+SCG1+SCG0,0(SP)      ; Exit LPM4 on RETI
   RETI
```

The following code example shows how to enter LPM5 mode. Exit from LPM5 is only possible with a RST/NMI event, a power on cycle, or if available on some devices via specific I/O. Upon exit from the device, a complete reset sequence is performed. Please refer to the Power Management Module Chapter for further details.

```
; Enter LPM5 Example
   BIS   #PMMREGOFF, &PMMCTL0                ;
   BIS   #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR     ;Enter LPM5 when PMMREGOFF is set.
```

1.5.1.1 Extended Time in Low-Power Modes

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

```
; Enter LPM4 Example with lowest DCO Setting
   BIC   #SCG0, SR                             ; Disable FLL
   MOV   #0100h, &UCSCTL0                      ; Set DCO tap to first tap, clear
modulation.
   BIC   #DCORSEL2+DCORSEL1+DCORSEL0,&UCSCTL1 ; Lowest DCORSEL
   BIS   #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR      ; Enter LPM4
;  ...                                         ; Program stops
;

; Interrupt Service Routine
   BIC   #CPUOFF+OSCOFF+SCG1+SCG0,0(SR)       ; Exit LPM4 on RETI
   RETI
```

1.6 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 or LPM4 modes whenever possible.

- 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.

1.7 Connection of Unused Pins

The correct termination of all unused pins is listed in Table 1-1.

Table 1-1. Connection of Unused Pins

Pin	Potential	Comment
AV_{CC}	DV_{CC}	
AV_{SS}	DV_{SS}	
Px.0 to Px.7	Open	Switched to port function, output direction
$\overline{RST}$/NMI	DV_{CC} or V_{CC}	47-kΩ pullup or internal pullup selected with 10-nF pulldown
TDO/TDI/TMS/TCK	Open	
TEST	Open	

1.8 Reset and Subtypes

BOR, POR, and PUC can be seen as a special type of a non-maskable interrupt with restart behavior of the complete system. BOR (brownout reset), POR (power on reset) and PUC (power up clear) are subtypes of it. Figure 1-6 shows their dependencies; A BOR reset represents the highest impacts to HW and causes a reload of device dependent HW while a PUC only resets the CPU and starts over with program execution.

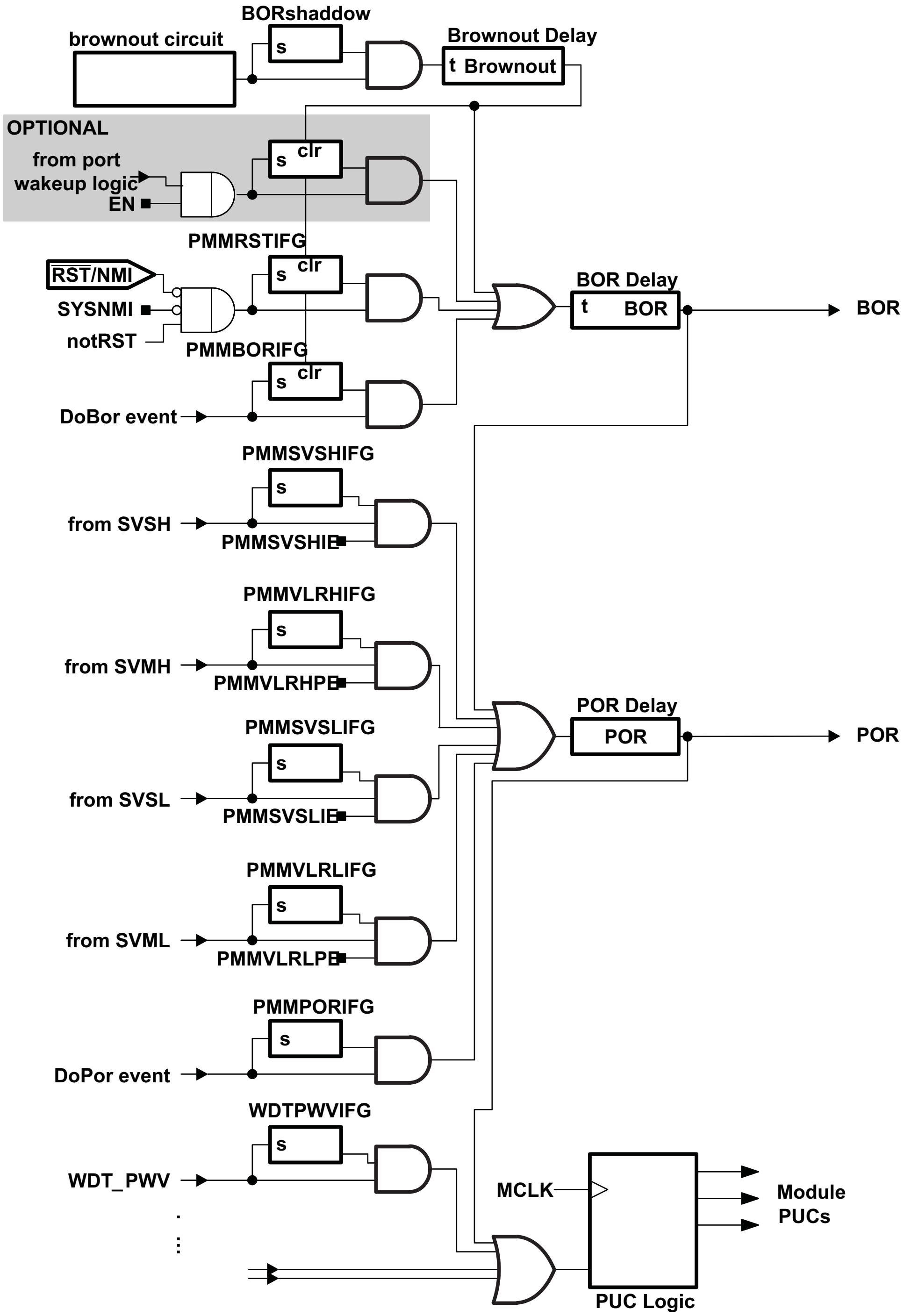


Figure 1-6. BOR/POR/PUC Reset Circuit

1.9 Interrupt Vectors

The interrupt vectors and the power-up starting address are located in the address range 0FFFFh to 0FF80h, for a maximum of 64 interrupt sources. A vector is programmed by the user this vector points to the start of the corresponding interrupt service routine. See the device-specific data sheet for the complete interrupt vector list.

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

Interrupt Source	Interrupt Flag	System Interrupt	Word Address	Priority
Reset: Power up, external reset, watchdog, flash password	... WDTIFG KEYV	... Reset	... 0FFFEh	... highest
System NMI: PSS		(non)-maskable	0FFFCh	…
User NMI: NMI, oscillator fault, flash memory access violation	... NMIIFG OFIFG ACCIFG	... (non)-maskable (non)-maskable (non)-maskable	... 0FFFAh	... …
device specific			0FFF8h	…
...			...	...
Watchdog timer	WDTIFG	maskable	...	...
...			...	...
device specific			…	…
reserved		(maskable)	…	lowest

Some interrupt enable bits, and interrupt flags and control bits for the $\overline{\text{RST}}$/NMI pin are located in the Special Function Registers (SFRs). The SFRs are located in the peripheral address range and are byte and word accessible. See the device-specific data sheet for the SFR configuration.

1.10 Special Function Registers

The special function registers, SFR, are listed in Table 1-4. The base address for the SFR registers is listed in Table 1-3.

Table 1-3. SFR Base Address

Module	Base address
SFR	00100h

Table 1-4. Special Function Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Interrupt enable register	SFRIE1 SFRIE1_L (IE1) SFRIE1_H (IE2)	read/write read/write read/write	word byte byte	00h 00h 01h	0000h 00h 00h
Interrupt flag register	SFRIFG1 SFRIFG1_L (IFG1) SFRIFG1_H (IFG2)	read/write read/write read/write	word byte byte	02h 02h 03h	0082h 82h 00h
Reset pin control register	SFRRPCR SFRRPCR_L SFRRPCR_H	read/write read/write read/write	word byte byte	04h 04h 05h	0000h 00h 00h

SFRIFG1, SFRIFG1_L, SFRIFG1_H, Interrupt Flag Register

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
JMBOUTIFG	**JMBINIFG**	**Reserved**	**NMIIFG**	**VMAIFG**	**Reserved**	**OFIFG**	**WDTIFG**
rw-(1)	rw-(0)	rw-0	rw-0	rw-0	r0	rw-(1)	rw-0

Reserved Bit 15–8 Reserved. Reads back 0

JMBOUTIFG Bit 7 JTAG mailbox output interrupt flag

0 no interrupt pending. When in 16-bit mode (JMBMODE = 0), this bit is cleared automatically when JMBO0 has been written by the CPU. When in 32-bit mode (JMBMODE = 1), this bit is cleared automatically when both JMBO0 and JMBO1 have been written by the CPU. This bit is also cleared when the associated vector in SYSUNIV has been read.

1 interrupt pending, JMBO registers are ready for new messages. In 16-bit mode (JMBMODE=0) JMBO0 has been received by JTAG. In 32-bit mode (JMBMODE=1) , JMBO0 and JMBO1 have been received by JTAG.

JMBINIFG Bit 6 JTAG mailbox input interrupt flag

0 no interrupt pending. When in 16-bit mode (JMBMODE = 0), this bit is cleared automatically when JMBI0 is read by the CPU. When in 32-bit mode (JMBMODE = 1), this bit is cleared automatically when both JMBI0 and JMBI1 have been read by the CPU. This bit is also cleared when the associated vector in SYSUNIV has been read

1 interrupt pending, a message is waiting in the JMBIN registers. In 16-bit mode (JMBMODE = 0) when JMBI0 has been written by JTAG. In 32 bit mode (JMBMODE = 1) when JMBI0 and JMBI1 have been written by JTAG.

Reserved Bit 5 Reserved. Reads back 0

NMIIFG Bit 4 NMI pin interrupt flag

0 no interrupt pending

1 interrupt pending

VMAIFG Bit 3 Vacant memory access interrupt flag

0 no interrupt pending

1 interrupt pending

Reserved Bit 2 Reserved. Reads back 0

OFIFG Bit 1 Oscillator fault interrupt flag

0 no interrupt pending

1 interrupt pending

WDTIFG Bit 0 Watchdog timer interrupt flag. In watchdog mode, WDTIFG remains set until reset by software. In interval mode, WDTIFG is reset automatically by servicing the interrupt, or can be reset by software. Because other bits in ~IFG1 may be used for other modules, it is recommended to clear WDTIFG by using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instructions.

0 no interrupt pending

1 interrupt pending

SFRIE1, SFRIE1_L, SFRIE1_H, Interrupt Enable Register

15 7	14 6	13 5	12 4	11 3	10 2	9 1	8 0
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
JMBOUTIE	**JMBINIE**	**ACCVIE**	**NMIIE**	**VMAIE**	**Reserved**	**OFIE**	**WDTIE**
rw-0	rw-0	rw-0	rw-0	rw-0	r0	rw-0	rw-0

Reserved	Bit 15-8	Reserved. Reads back 0.
JMBOUTIE	Bit 7	JTAG mailbox output interrupt enable flag 0 interrupts disabled 1 interrupts enabled
JMBINIE	Bit 6	JTAG mailbox input interrupt enable flag 0 interrupts disabled 1 interrupts enabled
ACCVIE	Bit 5	Flash controller access violation interrupt enable flag 0 interrupts disabled 1 interrupts enabled
NMIIE	Bit 4	NMI pin interrupt enable flag 0 interrupts disabled 1 interrupts enabled
VMAIE	Bit 3	Vacant memory access interrupt enable flag 0 interrupts disabled 1 interrupts enabled
Reserved	Bit 2	Reserved. Reads back 0.
OFIE	Bit 1	Oscillator fault interrupt enable flag 0 interrupts disabled 1 interrupts enabled
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 instruction 0 interrupts disabled 1 interrupts enabled

SFRRPCR, SFRRPCR_H, SFRRPCR_L, Reset Pin Control Register

15 7	14 6	13 5	12 4	11 3	10 2	9 1	8 0
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**Reserved**	**SYSRSTRE**	**SYSRSTUP**	**SYSNMIIES**	**SYSNMI**
r0	r0	r0	r0	rw-0	rw-0	rw-0	rw-0

Reserved Bit 15-5 Reserved. Reads back 0.

SYSRSTRE Bit 3 Reset pin resistor Enable.

0 Pullup/pulldown resistor at the $\overline{\text{RST}}$/NMI pin is disabled.

1 Pullup/pulldown resistor at the $\overline{\text{RST}}$/NMI pin is enabled.

SYSRSTUP Bit 2 Reset resistor pin pullup/pulldown.

0 Pulldown is selected.

1 Pullup is selected.

SYSNMIIES Bit 1 NMI edge select. This bit selects the interrupt edge for the NMI interrupt when SYSNMI = 1. Modifying this bit can trigger an NMI. Modify this bit when SYSNMI = 0 to avoid triggering an accidental NMI.

0 NMI on rising edge

1 NMI on falling edge

SYSNMI Bit 0 NMI select. This bit selects the function for the RST/NMI pin.

0 Reset function

1 NMI function

1.11 SYS Registers

The SYS registers are listed in Table 1-8 and Table 6. A detailed description of each register and its bits is also provided. Each register starts at a word boundary. Both, word or byte data can be written to the SYS registers.

Table 1-8. SYS Base Address

Module	Base address
SYS	00180h

Table 1-9. SYS Configuration Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
System Control Register	SYSCTL SYSCTL_L SYSCTL_H	read/write read/write read/write	word byte byte	00h 00h 01h	0000h 00h 00h
Boot strap loader configuration register	SYSBSLC SYSBSLC_L SYSBSLC_H	read/write read/write read/write	word byte byte	02h 02h 03h	0003h 03h 00h
Arbitration configuration Register	SYSARB SYSARB_L SYSARB_H	read/write read/write read/write	word byte byte	04h 04h 05h	0000h 00h 00h
JTAG Mailbox Control Register	SYSJMBC SYSJMBC_L SYSJMB_H	read/write read/write read/write	word byte byte	06h 06h 07h	0000h 0Ch 00h
JTAG Mailbox Input Register #0	SYSJMBI0 SYSJMBI0_L SYSJMBI0_H	read/write read/write read/write	word byte byte	08h 08h 09h	0000h 00h 00h
JTAG Mailbox Input Register #1	SYSJMBI1 SYSJMBI1_L SYSJMBI1_H	read/write read/write read/write	word byte byte	0Ah 0Ah 0Bh	0000h 00h 00h
JTAG Mailbox Output Register #0	SYSJMBO0 SYSJMBO0_L SYSJMBO0_H	read/write read/write read/write	word byte byte	0Ch 0Ch 0Dh	0000h 00h 00h
JTAG Mailbox Output Register #1	SYSJMBO1 SYSJMBO1_L SYSJMBO1_H	read/write read/write read/write	word byte byte	0Eh 0Eh 0Fh	0000h 00h 00h
reserved for future use				10h 17h	
Bus error vector generator	SYSBERRIV	read only	word	18h	0000h
User NMI vector generator	SYSUNIV	read only	word	1Ah	0000h
System NMI vector gen.	SYSSNIV	read only	word	1Ch	0000h
Reset vector generator	SYSRSTIV	read only	word	1Eh	0002h

Access to some SYS registers is allowed only if the SYSLOCK bit is reset.

SYSCTL, SYSCTL_L, SYSCTL_H, SYS Control Register

15 7	14 6	13 5	12 4	11 3	10 2	9 1	8 0
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Reserved	**Reserved**	**SYSJTAGPIN**	**SYSBSLIND**	**Reserved**	**SYSPMMPE**	**Reserved**	**SYSRIVECT**
r0	r0	rw-(0)	r-0	r0	rw-(0)	r0	rw-(0)

Reserved Bits 15-8 Reserved. Reads back 0.

SYSJTAGPIN Bit 5 Dedicated JTAG pins enable. Setting this bit disables the shared functionality of the JTAG pins and permanently enables the JTAG function. This bit can only be set once. Once it is set it will remain set until a BOR occurs.

0 shared JTAG pins (JTAG mode selectable via SBW sequence)

1 Dedicated JTAG pins (explicit 4 wire JTAG mode selection)

SYSBSLIND Bit 4 TCK/RST entry BSL indication detected to allow writing a backward compatible BSL to early ‘430 families. See BSL entry in Spy-Bi –Wire

0 No BSL indication

1 BSL entry detected

Reserved Bit 3 Reserved. Reads back 0.

SYSPMMPE Bit 2 PMM access protect. The control register of the PMM module can be accessed by a program running … (If this bit is set to one it only can be cleared again by a BOR)

0 … anywhere in memory

1 … only from boot code area (01B00h-01BFFh) and the protected BSL segments.

Reserved Bit 1 Reserved. Reads back 0.

SYSRIVECT Bit 0 RAM based Interrupt Vectors

0 Interrupt Vectors generated with end address: TOP of lower 64k Flash FFFFh

1 Interrupt Vectors generated with end address TOP of RAM

SYSBSLC, SYSBSLC_L, SYSBSLC_H, BSL Configuration Register

15 7	14 6	13 5	12 4	11 3	10 2	9 1	8 0
SYSBSLPE	**SYSBSLOFF**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**
rw-(0)	rw-(0)	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**SYSBSLR**	**SYSBSLSIZE**	
r0	r0	r0	r0	r0	rw-(0)	rw-(1)	rw-(1)

SYSBSLPE Bit 15-7 Boot strap loader (BSL) memory protection enable for the size covered in SYSBSLSIZE

0 area not protected read, program and erase of memory is possible

1 area protected

SYSBSLOFF Bit 14-6 Boot strap loader (BSL) memory disable for the size covered in SYSBSLSIZE

0 BSL memory is addressed when this area is read

1 BSL memory behaves like vacant memory

Reserved Bit 13-3 Reserved. Reads back 0.

SYSBSLR Bit 2 RAM assigned to BSL

0 no RAM assigned to BSL area

1 lowest 16 bytes of RAM assigned to BSL

SYSBSLSIZE Bit 1-0 **BOOT Strap Loader Size**
This defines the space and size of Flash that is reserved for the Boot Strap Loader.

00 Size: 512B BSL_SEG_3

01 Size: 1024B BSL_SEG_2,3

10 Size: 1536B BSL_SEG_1,2,3

11 Size: 2048B BSL_SEG_0,1,2,3 **(value after BOR!)**

SYSJMBC, SYSJMBC_L, SYSBMBC_H, JTAG Mailbox Control Register

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**
r0	r0	r0	r0r	r0	r0	r0	r0

7	6	5	4	3	2	1	0
JMBCLR10FF	**JMBCLR0OFF**	**Reserved**	**JMBM0DE**	**JMBOUT1FG**	**JMBOUT0FG**	**JMBIN1FG**	**JMBIN0FG**
rw-(0)	rw-(0)	r0	rw-0	r-(1)	r-(1)	rw-(0)	rw-(0)

Reserved Bit 15-8 Reserved. Reads back 0.

JMBCLR1OFF Bit 7 Incoming JTAG Mailbox 1 flag auto-clear disable.

0 JMBIN1FG cleared on read of JMB1IN register

1 JMBIN1FG cleared by SW

JMBCLR0OFF Bit 6 Incoming JTAG Mailbox 0 flag auto-clear disable

0 JMBIN0FG cleared on read of JMB0IN register

1 JMBIN0FG cleared by SW

Reserved Bit 5 Reserved. Reads back 0.

JMBMODE Bit 4 This bit defined the operation mode of JMB for JMBI0/1 and JMBO0/1. Before switching this bit pad and flush out any partial content to avoid data drops.

0 16 bit transfers using JMBO0 and JMBI0 only

1 32 bit transfers using JMBO0/1 and JMBI0/1

JMBOUT1FG Bit 3 Outgoing JTAG Mailbox 1 flag. This bit is cleared automatically when a message is written to the upper byte of JMBO1 or as word access (by the CPU, DMA,…) and is set after the message was read via JTAG.

0 JMBO1 is not ready to receive new data

1 JMBO1 is ready to receive new data

JMBOUT0FG Bit 2 Outgoing JTAG Mailbox 0 flag. This bit is cleared automatically when a message is written to the upper byte of JMBO0 or as word access (by the CPU, DMA,…) and is set after the message was read via JTAG.

0 JMBO0 is not ready to receive new data

1 JMBO0 is ready to receive new data

JMBIN1FG Bit 1 Incoming JTAG Mailbox 1 flag. This bit is set when a new message (provided via JTAG) is available in JMBI1. This flag is cleared automatically on read of JMBI1 when JMBCLR1OFF=0 (auto clear mode). On JMBCLR1OFF=1 JMBIN1FG needs to be cleared by SW.

0 JMBI1 has no new data

1 JMBI1 has new data available

JMBIN0FG Bit 0 Incoming JTAG Mailbox 0 flag. This bit is set when a new message (provided via JTAG) is available in JMBI0. This flag is cleared automatically on read of JMBI0 when JMBCLR0OFF=0 (auto clear mode). On JMBCLR0OFF=1 JMBIN0FG needs to be cleared by SW.

0 JMBI1 has no new data

1 JMBI1 has new data available

SYSJMBI0, SYSJMBI0_L, SYSJMBI0_H, JTAG Mailbox Input 0 Register
SYSJMBI1, SYSJMBI1_L, SYSJMBI1_H, JTAG Mailbox Input 1 Register

15 7	14 6	13 5	12 4	11 3	10 2	9 1	8 0
MSGHI							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0

7	6	5	4	3	2	1	0
MSGL0							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0

MSGHI	Bit 15-8	JTAG mailbox incoming message high byte
MSGLO	Bit 7-0	JTAG mailbox incoming message low byte

SYSJMBO0, SYSJMBO0_L, SYSJMBO0_H, JTAG Mailbox Out 0
Register SYSJMBO1, SYSJMBO1_L, SYSJMBO1_H, JTAG Mailbox Out 1 Register

15 7	14 6	13 5	12 4	11 3	10 2	9 1	8 0
MSGHI							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
MSGL0							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

MSGHI	Bit 15-8	JTAG Mailbox outgoing message high byte
MSGLO	Bit 7-0	JTAG Mailbox outgoing message low byte

SYSUNIV, SYSUNIV_H, SYSUNIV_L, User NMI Interrupt Vector Register

15 7	14 6	13 5	12 4	11 3	10 2	9 1	8 0
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**	**SYSUNVEC**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

SYSUNIV Bit 15-0 User NMI interrupt vector. It generates an value that can be used as address offset for fast interrupt service routine handling. Writing to this register clears all pending user NMI interrupt flags.

Value	**Interrupt Type**
0000h	No interrupt pending
0002h	NMIIFG interrupt pending (highest priority)
0004h	OFIFG interrupt pending
0006h	ACCVIFG interrupt pending
0008h	reserved for future extensions

Note: Additional events for more complex devices will be appended to this table; Sources that are removed will reduce the length of this table. The vectors are expected to be accessed symbolic only with the corresponding include file of the used device.

SYSSNIV, SYSSNIV_H, SYSSNIV_L, SYS NMI Interrupt Vector Register

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
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**	**SYSSNVEC**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

SYSSNIV Bit 15-0 System NMI interrupt vector. It generates an value that can be used as address offset for fast interrupt service routine handling. Writing to this register clears all pending system NMI interrupt flags.

Value	Interrupt Type
0000h	No interrupt pending
0002h	SVMLIFG interrupt pending (highest priority)
0004h	SVMHIFG interrupt pending
0006h	DLYLIFG interrupt pending
0008h	DLYHIFG interrupt pending
000Ah	VMAIFG interrupt pending
000Ch	JMBINIFG interrupt pending
000Eh	JMBOUTIFG interrupt pending
0010h	VLRLIFG interrupt pending
0012h	VLRHIFG interrupt pending
0014h	Reserved for future extensions

Note: Additional events for more complex devices will be appended to this table; Sources that are removed will reduce the length of this table. The vectors are expected to be accessed symbolic only with the corresponding include file of the used device.

SYSRSTIV, SYSRSTIV_H, SYSRSTIV_L, SYS Reset Interrupt Vector Register

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
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**	**SYSRSTVEC**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

SYSRSTIV Bit 15-0 Reset interrupt vector. It generates an value that can be used as address offset for fast interrupt service routine handling to identify the last cause of an Reset (BOR, POR, PUC) . Writing to this register clears all pending reset source flags.

Value	Interrupt Type
0000h	No interrupt pending
0002h	Brownout (BOR) (highest priority)
0004h	RST/NMI (POR) (also RST wakes up)
0006h	DoBOR (BOR)
0008h	Port wakeup (BOR)
000Ah	Security violation (BOR)
000Ch	SVSL (POR)
000Eh	SVSH (POR)
0010h	SVML_OVP (POR)
0012h	SVMH_OVP (POR)
0014h	DoPOR (POR)
0016h	WDT time out (PUC)
0018h	WDT keyviol (PUC)
001Ah	KEYV flash keyviol (PUC)
001Ch	PLL unlock (PUC)
001Eh	PERF peripheral/configuration area fetch (PUC)
0020h	PMM key violation (PUC)
0022h	Reserved for future extensions

Note: Additional events for more complex devices will be appended to this table; Sources that are removed will reduce the length of this table. The vectors are expected to be accessed symbolic only with the corresponding include file of the used device

Chapter 2

SLAU208–June 2008

Watchdog Timer (WDT_A)

The watchdog timer is a 32-bit timer that can be used as a watchdog or as an interval timer. This chapter describes the watchdog timer. The enhanced watchdog timer, WDT_A, is implemented in all MSP430x5xx devices.

Topic **Page**

2.1 Watchdog Timer Introduction

The primary function of the watchdog timer (WDT_A) module is to perform a controlled system restart after a software problem occurs. If the selected time interval expires, a system reset is generated. If the watchdog function is not needed in an application, the module can be configured as an interval timer and can generate interrupts at selected time intervals.

Features of the watchdog timer module include:

- Eight software-selectable time intervals
- Watchdog mode
- Interval mode
- Access to WDT control register is password protected
- Selectable clock source
- Can be stopped to conserve power
- Clock fail-safe feature

The WDT block diagram is shown in Figure 2-1.

Note: Watchdog Timer Powers Up Active

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

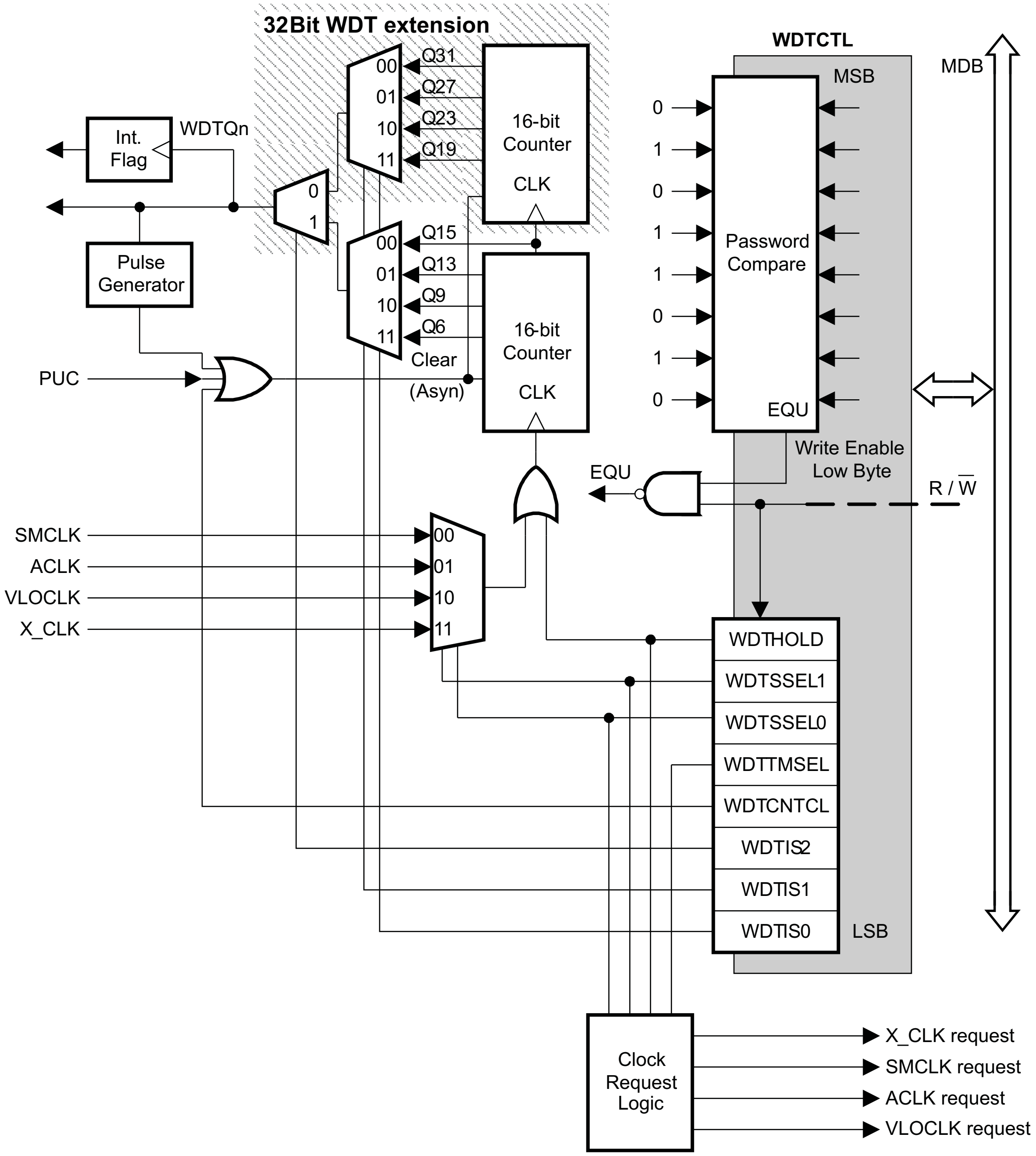


Figure 2-1. Watchdog Timer Block Diagram

2.2 Watchdog Timer Block Diagram

The WDT module can be configured as either a watchdog or interval timer with the WDTCTL register. 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. Byte reads on WDTCTL high or low part will result the value of the low byte. Writing byte wide to upper or lower part of WDTCTL results into a PUC.

2.2.1 Watchdog Timer Counter

The watchdog timer counter (WDTCNT) is a 32-bit up-counter that is not directly accessible by software. The WDTCNT is controlled and its time intervals selected through the watchdog timer control register WDTCTL. The WDTCNT can be sourced from SMCLK, ACLK, VLOCLK and X_CLK on some devices. The clock source is selected with the WDTSSEL bits.

2.2.2 Watchdog Mode

After a PUC condition, the WDT module is configured in the watchdog mode with an initial ~32-ms reset interval using the SMCLK. 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.

2.2.3 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.

2.2.4 Watchdog Timer Interrupts

The WDT uses two bits in the SFRs for interrupt control

- The WDT interrupt flag, WDTIFG, located in SFRIFG1.0
- The WDT interrupt enable, WDTIE, located in SFRIE1.0

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.

2.2.5 Clock Fail-Safe Feature

The WDT_A provides a fail-safe clocking feature assuring the clock to the WDT_A cannot be disabled while in watchdog mode. This means the low-power modes may be affected by the choice for the WDT_A clock.

If SMCLK or ACLK fails as WDT_A clock source then VLOCLK is automatically selected as WDT_A clock source.

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

2.2.6 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_A should be configured. For example, the WDT_A should not be configured in watchdog mode with a clock source that is originally sourced from DCO, XT1 in high frequency mode, or XT2 via SMCLK or ACLK if the user wants to use low power mode 3. In this case, SMCLK or ACLK would remain enabled 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.

2.2.7 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+WDTCNTCL+SSEL,&WDTCTL
;
; Stop the watchdog
MOV #WDTPW+WDTHOLD,&WDTCTL
;
; Change WDT to interval timer mode, clock/8192 interval
MOV #WDTPW+WDTCNTCL+WDTTMSEL+WDTIS2+WDTIS0,&WDTCTL
```

2.3 Watchdog Timer Registers

The watchdog timer module registers are listed in Table 2-2. The base register or the watchdog timer module registers and special function registers (SFR) can be found in the device specific data sheet. The address offset is given in Table 2-1.

Table 2-1. Watchdog Timer Base Register

Module	Base address
WDT_A	00150h

Table 2-2. Watchdog Timer Registers

Register	Short Form	Register Type	REG Access	Address	Initial State
Watchdog timer control register	WDTCTL	read/write	word	0Ch	6904h

WDTCTL, Watchdog Timer Register

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Read as 069h **WDTPW, must be written as 05Ah**							

7	6	5	4	3	2	1	0
WDTHOLD	**WDTSSELx**		**WDTTMSEL**	**WDTCNTCL**	**WDTISx**		
rw-0	rw-0	rw-0	rw-0	r0(w)	rw-1	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

WDTSSEL Bit 6-5 Watchdog timer clock source select

00 SMCLK

01 ACLK

10 VLOCLK

11 X_CLK , same as VLOCLK if not defined differently in data sheet

WDTTMSEL Bit 4 Watchdog timer mode select

0 Watchdog mode

1 Interval timer mode

WDTCNTL 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

WDTISx Bit 2-0 Watchdog timer interval select. These bits select the watchdog timer interval to set the WDTIFG flag and/or generate a PUC.

000 Watchdog clock source /2G (18:12:16 at 32 kHz)

001 Watchdog clock source /128M (01:08:16 at 32 kHz

010 Watchdog clock source /8192k (00:04:16 at 32 kHz)

011 Watchdog clock source /512k (00:00:16 at 32 kHz)

100 Watchdog clock source /32k (1 s at 32 kHz)

101 Watchdog clock source /8192 (250 ms at 32 kHz)

110 Watchdog clock source /512 (15,6 ms at 32 kHz)

111 Watchdog clock source /64 (1.95 ms at 32 kHz)

Chapter 3

SLAU208–June 2008

Unified Clock System (UCS)

The Unified Clock System module provides the clocks for MSP430x5xx devices. This chapter describes the operation of the Unified Clock System module. The Unified Clock System module is implemented in all MSP430x5xx devices.

Topic **Page**

3.1 Unified Clock System Introduction

The Unified Clock System (UCS) module supports low system cost and ultra-low power consumption. Using three internal clock signals, the user can select the best balance of performance and low power consumption. The Unified Clock System module can be configured to operate without any external components, with one or two external crystals, or with resonators, under full software control.

The Unified Clock System module includes up to five clock sources:

- XT1CLK: Low-frequency/high-frequency oscillator that can be used either with low-frequency 32768-Hz watch crystals, standard crystals, resonators, or external clock sources in the 4-MHz to 32-MHz range.
- VLOCLK: Internal very low power, low frequency oscillator with 12 kHz typical frequency.
- REFOCLK: Internal, trimmed, low frequency oscillator with 32768 Hz typical frequency, with the ability to be used as a clock reference into the FLL.
- DCOCLK: Internal digitally controlled oscillator (DCO) that can be stabilized by the FLL.
- XT2CLK: Optional high-frequency oscillator that can be used with standard crystals, resonators, or external clock sources in the 4-MHz to 40-MHz range.

Three clock signals are available from the Unified Clock System module:

- ACLK: Auxiliary clock. The ACLK is software selectable as XT1CLK, REFOCLK, VLOCLK, DCOCLK, DCOCLKDIV, and when available, XT2CLK. DCOCLKDIV is the DCOCLK frequency divided by 1, 2, 4, 8, 16, or 32 within the FLL block. ACLK is software selectable for individual peripheral modules. ACLK is divided by 1, 2, 4, 8, 16 or 32. ACLK/n is ACLK divided by 1, 2, 4, 8, 16, or 32 and is available externally at a pin.
- MCLK: Master clock. MCLK is software selectable as XT1CLK, REFOCLK, VLOCLK, DCOCLK, DCOCLKDIV, and when available, XT2CLK. DCOCLKDIV is the DCOCLK frequency divided by 1, 2, 4, 8, 16, or 32 within the FLL block. MCLK is divided by 1, 2, 4, 8. 16, or 32 MCLK is used by the CPU and system.
- SMCLK: Sub-system master clock. SMCLK is software selectable as XT1CLK, REFOCLK, VLOCLK, DCOCLK, DCOCLKDIV, and when available, XT2CLK. DCOCLKDIV is the DCOCLK frequency divided by 1, 2, 4, 8, 16, or 32 within the FLL block. SMCLK is divided by 1, 2, 4, 8, 16, or 32. SMCLK is software selectable for individual peripheral modules.

A peripheral module may request its clock sources automatically if required for its proper operation. The block diagram of the Unified Clock System module is shown in Figure 3-1.

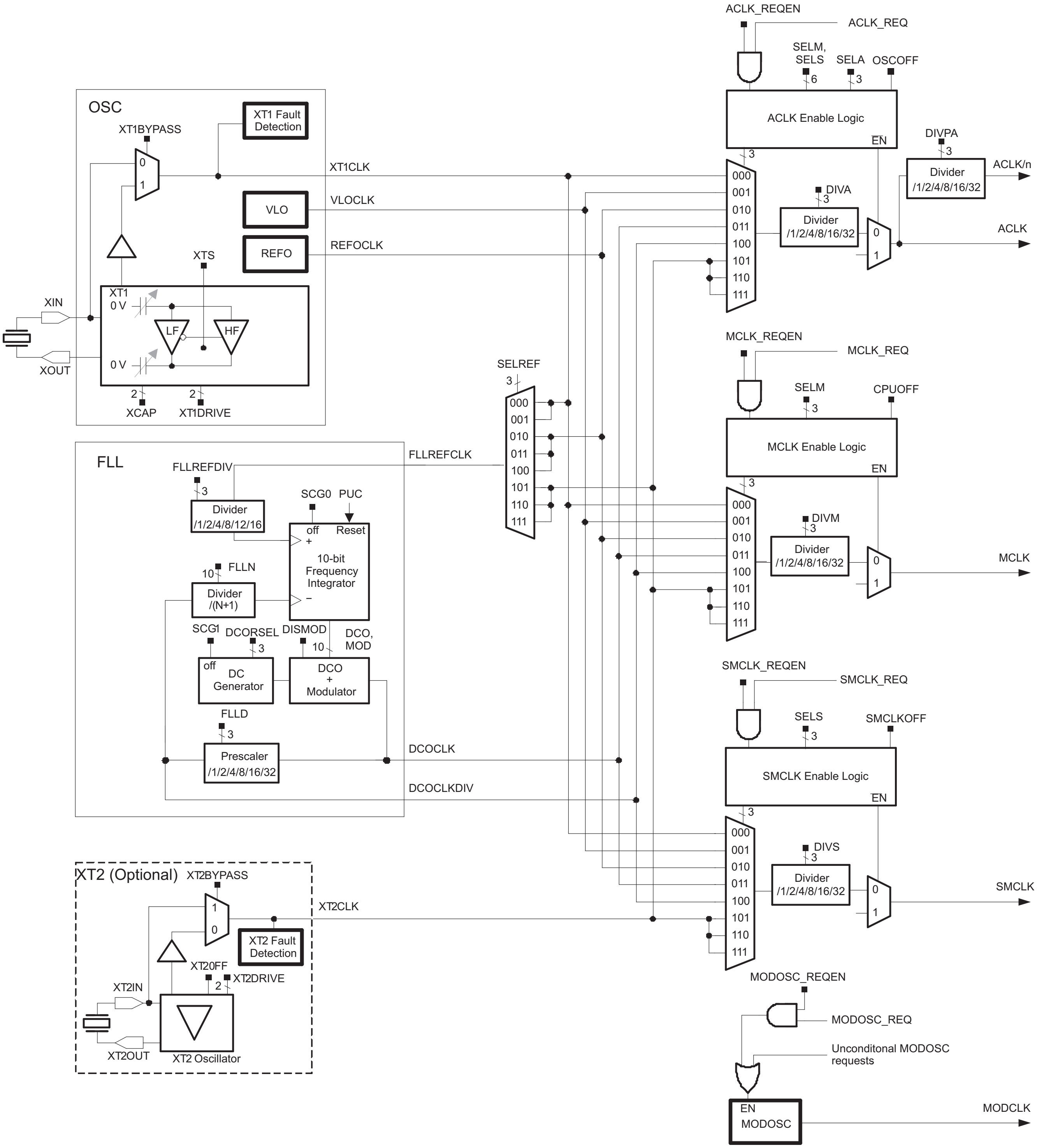


Figure 3-1. Unified Clock System Block Diagram

3.2 Unified Clock System Module Operation

After a PUC, the Unified Clock System module's default configuration is as follows:

- XT1 in LF mode is selected as the oscillator source for XT1CLK. XT1CLK selected for ACLK.
- DCOCLKDIV selected for MCLK.
- DCOCLKDIV selected for SMCLK.
- FLL operation is enabled. XT1CLK is selected as FLL reference clock, FLLREFCLK.
- XIN and XOUT pins set to general purpose I/O, XT1 remains disabled until I/O ports are configured for XT1 operation.
- When available, XT2IN and XT2OUT pins set to general purpose I/O, XT2 disabled.

As shown above, FLL operation with XT1 is enabled by default. On MSP430x5xx devices, the crystal pins (XIN, XOUT) are shared with general-purpose I/O. To enable XT1, the PSEL bits associated with the crystal pins must be set. When a 32,768 Hz crystal is used for XT1CLK, the fault control logic will immediately cause ACLK to be sourced by the REFOCLK since XT1 will not be stable immediately. See Section 3.2.12 for further details. Once the crystal startup is obtained and settled, the FLL stabilizes MCLK and SMCLK to 1.048576 MHz and f_{DCO} = 2.097152 MHz.

Status register control bits SCG0, SCG1, OSCOFF, and CPUOFF configure the MSP430 operating modes and enable or disable portions of the Unified Clock System module. See Chapter System Resets, Interrupts and Operating Modes. The UCSCTL0, UCSCTL1, UCSCTL2, UCSCTL3, UCSCTL4, UCSCTL5, UCSCTL6, UCSCTL7, and UCSCTL8 registers configure the Unified Clock System module.

The Unified Clock System module can be configured or reconfigured by software at any time during program execution.

3.2.1 Unified Clock System Module Features for Low-Power Applications

Conflicting requirements typically exist in battery-powered MSP430x5xx applications:

- Low clock frequency for energy conservation and time keeping
- High clock frequency for fast response times and fast burst processing capabilities
- Clock stability over operating temperature and supply voltage
- Low cost applications with less constrained clock accuracy requirements

The Unified Clock System module addresses the above conflicting requirements by allowing the user to select from the three available clock signals: ACLK, MCLK, and SMCLK.

All three available clock signals can be sourced via any of the available clock sources, (XT1CLK, VLOCLK, REFOCLK, or XT2CLK) giving complete flexibility in the system clock configuration.

For optimal low-power performance, ACLK can be sourced from a low-power 32,786-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. A flexible clock distribution and divider system is provided to fine tune the individual clock requirements. ACLK can be sourced via any of the available clock sources (XT1CLK, VLOCLK, REFOCLK, DCO, or XT2CLK) .

MCLK can be configured to operate from the on-chip DCO, optionally stabilized by the FLL, that can be activated when requested by interrupt-driven events. A flexible clock distribution and divider system is provided to fine tune the individual clock requirements. MCLK can be sourced via any of the available clock sources (XT1CLK, VLOCLK, REFOCLK, DCO, or XT2CLK).

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. SMCLK can be sourced via any of the available clock sources (XT1CLK, VLOCLK, REFOCLK, DCO, or XT2CLK).

3.2.2 Internal Very-Low-Power Low-Frequency Oscillator (VLO)

The internal VLO provides a typical frequency of 12 kHz (see device-specific data sheet for parameters) without requiring a crystal. The VLO provides for a low-cost ultra-low power clock source for applications that do not require an accurate time base.

The VLO is selected when it is used to source ACLK, MCLK, or SMCLK (SELA = 1 or SELM = 1 or SELS = 1).

3.2.3 Internal Trimmed Low-Frequency Reference Oscillator(REFO)

The internal trimmed reference oscillator (REFO) can be used for cost sensitive applications where a crystal is not required or desired. The reference oscillator is internally trimmed to 32.768 kHz typical and provides for a stable reference frequency that can be used as FLLREFCLK. The REFOCL , combined with the FLL, provides for a flexible range of system clock settings without the need for a crystal. The REFO consumes no power when not being used.

The REFO is selected when it is used to source ACLK, MCLK, or SMCLK (SELA = 2 or SELM = 2 or SELS = 2) or sources the FLL (SELREF = 2). The REFO oscillator can be disabled with software by setting OSCOFF, if the REFO oscillator is not used to source MCLK, SMCLK. or FLLREFCLK. The OSCOFF bit disables REFO for LPM4.

3.2.4 XT1 Oscillator

The XT1 oscillator supports ultra low-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 XCAP bits configure the internally provided load capacitance for the XT1 crystal in LF mode. This capacitance can be selected as 2 pF, 6 pF, 9 pF, or 12 pF (typical). Additional external capacitors can be added if necessary.

The XT1 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.

The drive settings of XT1 in LF mode can be increased with the XT1DRIVE bits. At power up, the XT1 starts with the highest drive settings for fast, reliable startup. If needed, user software can reduce the drive strength to further reduce power. In HF mode, different crystal or resonator ranges are supported by choosing the proper XT1DRIVE settings.

XT1 may be used with an external clock signal on the XIN pin in either LF or HF mode by setting XT1BYPASS. When used with an external signal, the external frequency must meet the datasheet parameters for the chosen mode. XT1 is powered down when used in bypass mode.

The XT1 pins are shared with general-purpose I/O ports. At power up, the default operation is XT1, LF mode of operation. However, XT1 will remain disabled until the ports shared with XT1 are configured for XT1 operation. The configuration of the shared I/O is determined by the PSEL bit associated with XIN and the XT1BYPASS bit. Setting the PSEL bit will cause the XIN and XOUT ports to be configured for XT1 operation. If XT1BYPASS is also set, XT1 is configured for bypass mode of operation. In bypass mode of operation, XIN can accept an external clock input signal and XOUT is configured as general-purpose I/O. The PSEL bit associated with XOUT is a do not care.

If the PSEL bit associated with XIN is cleared, both XIN and XOUT ports are configured as general-purpose I/O and XT1 will be disabled.

XT1 is enabled when it is used to source ACLK, MCLK, or SMCLK (SELA = 0 or SELM = 0 or SELS = 0) or FLLREFCLK (SELREF = 0) and (XT1OFF = 1) in all power modes AM through LPM3, otherwise it is disabled. Setting OSCOFF (LPM4) while (XT1OFF = 1), will disable XT1. If an application wishes to have XT1 enabled regardless of the OSCOFF setting, clearing XT1OFF will enable XT1 continuously. This will cause XT1 to be enabled in power modes AM through LPM4.

3.2.5 XT2 Oscillator

Some devices have a second crystal oscillator, XT2. XT2 sources XT2CLK and its characteristics are identical to XT1 in HF mode. The XT2DRIVE bits select the frequency range of operation of XT2.

XT2 may be used with external clock signals on the XT2IN pin by setting XT2BYPASS. When used with an external signal, the external frequency must meet the datasheet parameters for XT2. XT2 is powered down when used in bypass mode.

The XT2 pins are shared with general-purpose I/O ports. At power up, the default operation is XT2. However, XT2 will remain disabled until the ports shared with XT2 are configured for XT2 operation.The configuration of the shared I/O is determined by the PSEL bit associated with XT2IN and the XT2BYPASS bit. Setting the PSEL bit will cause the XT2IN and XT2OUT ports to be configured for XT2 operation. If XT2BYPASS is also set, XT2 is configured for bypass mode of operation. In bypass mode of operation, XT2IN can accept an external clock input signal and XT2OUT is configured as general-purpose I/O. The PSEL bit associated with XT2OUT is a do not care.

If the PSEL bit associated with XT2IN is cleared, both XT2IN and XT2OUT ports are configured as general-purpose I/O and XT2 will be disabled.

XT2 is enabled when it is used to source ACLK, MCLK, or SMCLK (SELA = 5 or SELM = 5 or SELS = 5) or FLLREFCLK (SELREF = 5) and (XT2OFF = 1) in all power modes AM through LPM3, otherwise it is disabled. Setting OSCOFF (LPM4) while (XT2OFF = 1), will disable XT2. If an application wishes to have XT2 enabled regardless of the OSCOFF setting, clearing XT2OFF will enable XT2 continuously. This will cause XT2 to be enabled in power modes AM through LPM4.

3.2.6 Digitally-Controlled Oscillator (DCO)

The DCO is an integrated digitally controlled oscillator. The DCO frequency can be adjusted by software using the DCORSEL, DCO, and MOD bits. The DCO frequency can be optionally stabilized by the FLL to a multiple frequency of FLLREFCLK / n . The FLL can accept different reference sources selectable via the SELREF bits. Reference sources include XT1CLK, REFOCLK, or XT2CLK (if available) The value of n is defined by the FLLREFDIVx (n = 1, 2, 4, 8, 12, or 16). The default is n = 1.

The FLLD bits configure the FLL prescaler divider value D to 1, 2, 4, 8, 16, or 32. By default, D = 2, MCLK and SMCLK are sourced from DCOCLKDIV, providing a clock frequency DCOCLK/2.

The divider (N + 1) and the divider value D define the DCOCLK and DCOCLKDIV frequencies, where N > 0. Writing N = 0 causes the divider to be set to 2.

$$f_{DCOCLK} = D \times (N + 1) \times (f_{FLLREFCLK} \div n)$$

$$f_{DCOCLKDIV} = (N + 1) \times (f_{FLLREFCLK} \div n)$$

Adjusting the DCO Frequency

By default, FLL operation is enabled. FLL operation can be disabled by setting SCG0. Once disabled, the DCO will continue to operate at the current settings defined in UCSCTL0 and UCSCTL1. The DCO frequency can be adjusted manually if desired. Otherwise, the DCO frequency will be stabilized by the FLL operation.

After a PUC, DCORSELx = 2 and DCOx = 0. MCLK and SMCLK are sourced from DCOCLKDIV. Because the CPU executes code from MCLK, which is sourced from the fast-starting DCO, code execution begins from PUC in less than 5 µs.

The frequency of DCOCLK is set by the following functions:

- The three DCORSELx bits select one of eight nominal frequency ranges for the DCO. These ranges are defined for an individual device in the device-specific data sheet.
- The five DCOx bits divide the DCO range selected by the DCORSELx bits into 32 frequency steps, separated by approximately 8%.
- The five MODx bits, switch between the frequency selected by the DCOx bits and the next higher frequency set by DCOx + 1. When DCOx = 31, the MODx bits have no effect, because the DCO is already at the highest setting for the selected DCORSELx range.

3.2.7 Frequency Locked Loop (FLL)

The FLL continuously counts up or down a frequency integrator. The output of the frequency integrator that drives the DCO can be read in UCSCTL0, UCSCTL1 (bits MODx and DCOx). The count is adjusted +1 with the frequency $f_{FLLREFCLK}/n$ (n = 1, 2, 4, 8, 12, or 16) or –1 with the frequency $f_{DCOCLK}/(D \times (N+1))$.

Note: **Reading MODx and DCOx**

The integrator is updated via the DCOCLK which may differ in frequency of operation of MCLK. It is possible that immediate reads of a previously written value are not visible to the user since the update to the integrator has not occurred. This is normal. Once the integrator is updated at the next successive DCOCLK, the correct value can be read.

In addition, since the MCLK can be asynchronous to the integrator updates, reading the values may be cause a corrupted value to be read under this condition. In this case, a majority vote method should be performed.

Five of the integrator bits, UCSCTL0 bits 12 to 8, set the DCO frequency tap. Thirty-two taps are implemented for the DCO, and each is approximately 8% higher than the previous. The modulator mixes two adjacent DCO frequencies to produce fractional taps.

For a given DCO bias range setting, time must be allowed for the DCO to settle on the proper tap for normal operation. $(n \times 32)\ f_{FLLREFCLK}$ cycles are required between taps requiring a worst case of $(n \times 32 \times 32)\ f_{FLLREFCLK}$ cycles for the DCO to settle. The value n is defined by the FLLREFDIVx bits (n = 1, 2, 4, 8, 12, or 16).

3.2.8 DCO Modulator

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

The modulator mixing formula is:

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

Figure 3-2 illustrates the modulator operation.

When FLL operation is enabled, the modulator settings and DCO are controlled by the FLL hardware. If FLL operation is not desired, the modulator settings and DCO control can be configured with software.

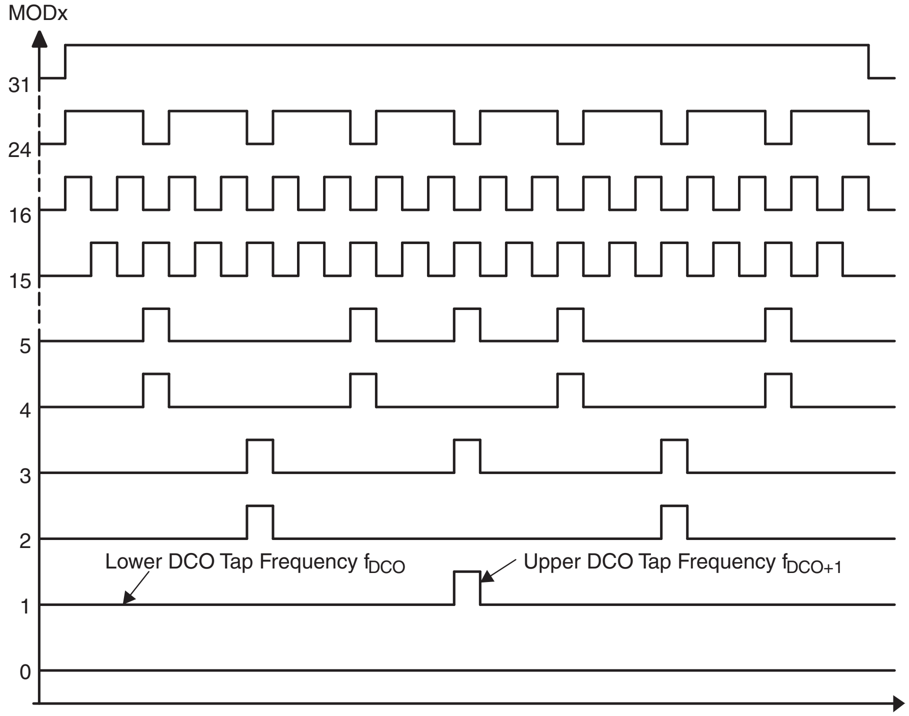

MODx
31
24
16
15
5
4
3
2
1
0
Lower DCO Tap Frequency f_{DCO}
Upper DCO Tap Frequency f_{DCO+1}

Figure 3-2. Modulator Patterns

3.2.9 Disabling the FLL Hardware and Modulator

The FLL is disabled when the status register bits SCG0 or SCG1 are set. When the FLL is disabled, the DCO runs at the previously selected tap and DCOCLK is not automatically stabilized.

The DCO modulator is disabled when DISMOD is set. When the DCO modulator is disabled, the DCOCLK is adjusted to the DCO tap selected by the DCOx bits.

Note: DCO Operation without FLL

When FLL operation is disabled, the DCO will continue to operate at the current settings. Since it is not stabilized by the FLL, temperature and voltage variations will influence the frequency of operation. Please refer to the device specific data sheet for voltage and temperature coefficients to ensure reliable operation.

3.2.10 FLL Operation from Low-Power Modes

An interrupt service request clears SCG1, CPUOFF and OSCOFF if set but does not clear SCG0. This means that FLL operation from within an interrupt service routine entered from LPM1, 2, 3 or 4, the FLL remains disabled and the DCO operates at the previous setting as defined in UCSCTL0 and UCSCTL1. SCG0 can be cleared by user software if FLL operation is required.

3.2.11 Operation from Low-Power Modes, Requested by Peripheral Modules

Peripheral modules can request a clock from the Unified Clock System module if their state of operation still requires an operational clock as shown in Figure 3-3.

A peripheral module asserts one of three possible clock request signals, ACLK_REQ, MCLK_REQ, or SMCLK_REQ. If the requested source is not active, the software NMI handler must take care of the required actions.

The watchdog, due to its security requirement, actively selects the VLOCLK source if the originally selected clock source is not available.

Any clock request from a peripheral module will cause its respective clock off signal to be overridden, but does not change the setting of clock off control bit. For example, a peripheral module may require the MCLK source which is currently disabled by the CPUOFF bit. The module can request the MCLK source by setting the MCLK_REQ bit. This causes the CPUOFF bit to have no effect, thereby allowing the MCLK to be sourced to the requesting peripheral module.

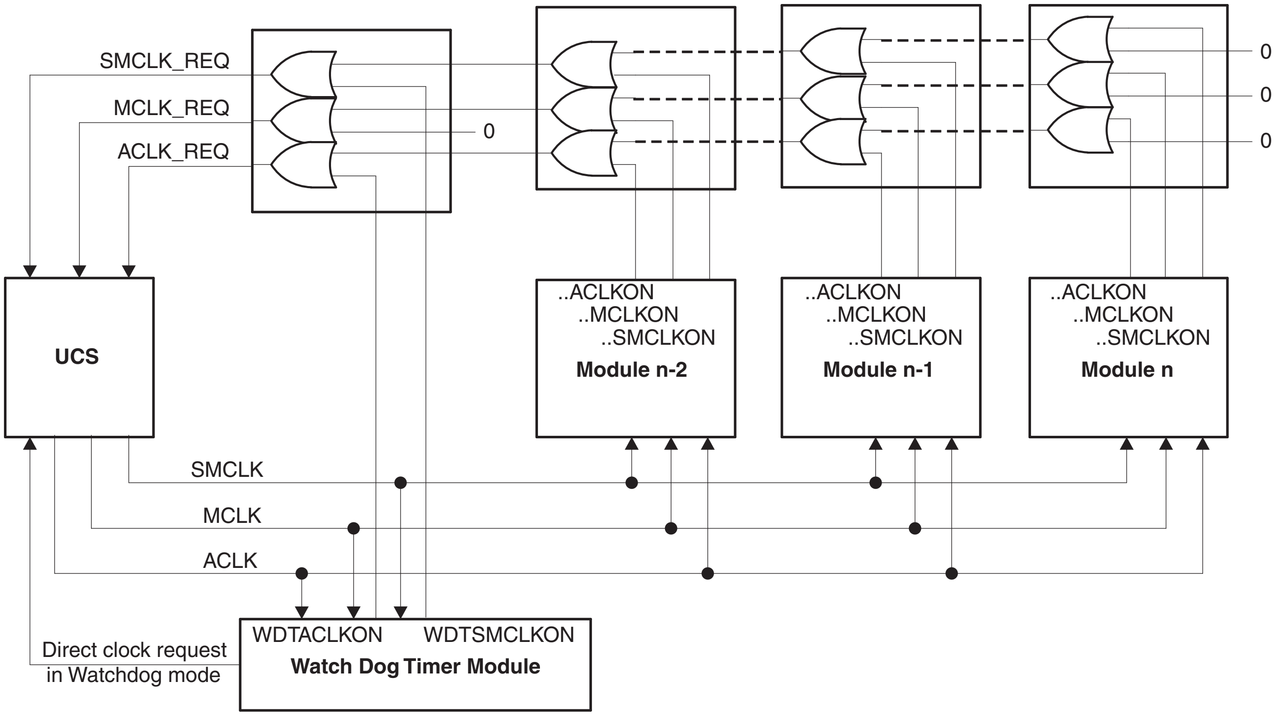


Figure 3-3. Module Request Clock System

3.2.12 *Unified Clock System Module Fail-Safe Operation*

The Unified Clock System module incorporates an oscillator-fault fail-safe feature. This feature detects an oscillator fault for XT1, DCO and XT2 as shown in Figure 3-4. The available fault conditions are:

- Low-frequency oscillator fault (XT1LFOFFG) for XT1 in LF mode
- High-frequency oscillator fault (XT1HFOFFG) for XT1 in HF mode
- High-frequency oscillator fault (XT2OFFG) for XT2
- DCO fault flag (DCOFFG) for the DCO

The crystal oscillator fault bits XT1LFOFFG, XT1HFOFFG and XT2OFFG are set if the corresponding crystal oscillator is turned on and not operating properly. Once set, the fault bits remain set regardless if the fault condition no longer exists. If the user clears the fault bits, and the fault condition still exists, the fault bits will automatically be set, otherwise they remain cleared.

When using XT1 operation in LF mode as the reference source into the FLL (SELREFx = 0), a crystal fault will automatically cause the FLL reference source, FLLREFCLK, to be sourced by REFO. XT1LFOFFG will be set. When using XT1 operation in HF mode as the reference source into the FLL, a crystal fault causes no FLLREFCLK signal to be generated and the FLL continues to count down to zero in an attempt to lock FLLREFCLK and DCOCLK/(D×[N+1]). The DCO tap moves to the lowest position (DCOx are cleared) and the DCOFFG is set. DCOFFG is also set if the N-multiplier value is set too high for the selected DCO frequency range resulting the DCO tap to move to the highest position (UCSCTL0.12 to UCSCTL0.8 are set). The DCOFFG will remain set until cleared by the user. If the user clears the DCOFFG and the fault condition remains, it will automatically be set, otherwise it remains cleared. XT1HFOFFG will be set.

When using XT2 as the reference source into the FLL, a crystal fault causes no FLLREFCLK signal to be generated and the FLL continues to count down to zero in an attempt to lock FLLREFCLK and DCOCLK/(D×[N+1]). The DCO tap moves to the lowest position (DCOx are cleared) and the DCOFFG is set. DCOFFG is also set if the N-multiplier value is set too high for the selected DCO frequency range resulting the DCO tap to move to the highest position (UCSCTL0.12 to UCSCTL0.8 are set). The DCOFFG will remain set until cleared by the user. If the user clears the DCOFFG and the fault condition remains, it will automatically be set, otherwise it will remain cleared. XT2OFFG will be set.

The OFIFG oscillator-fault interrupt flag is set and latched at POR or when any oscillator fault (XT1LFOFFG, XT1HFOFFG, XT2OFFG, or DCOFFG) is detected. When OFIFG is set, and OFIE is set, the OFIFG requests an NMI interrupt. When the interrupt is granted, the OFIE is not reset automatically as in previous MSP430 families. It is no longer required to reset the OFIE. NMI entry/exit circuitry removes this requirement. 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 oscillator sourcing MCLK, MCLK is automatically switched to the DCO for its clock source (DCOCLKDIV) for all clock sources except XT1 LF mode. If MCLK is sourced from XT1 in LF mode, an oscillator fault will cause MCLK to be automatically switched to the REFO for its clock source (REFOCLK). This does not change the SELMx bit settings. This condition must be handled by user software.

If a fault is detected for the oscillator sourcing SMCLK, SMCLK is automatically switched to the DCO for its clock source (DCOCLKDIV) for all clock sources except XT1 LF mode. If SMCLK is sourced from XT1 in LF mode, an oscillator fault will cause SMCLK to be automatically switched to the REFO for its clock source (REFOCLK). This does not change the SELSx bit settings. This condition must be handled by user software.

If a fault is detected for the oscillator sourcing ACLK, ACLK is automatically switched to the DCO for its clock source (DCOCLKDIV) for all clock sources except XT1 LF mode. If ACLK is sourced from XT1 in LF mode, an oscillator fault will cause ACLK to be automatically switched to the REFO for its clock source (REFOCLK). This does not change the SELAx bit settings. This condition must be handled by user software.

Note: DCO Active During Oscillator Fault

DCOCLKDIV is active even at the lowest DCO tap. The clock signal is available for the CPU to execute code and service an NMI during an oscillator fault.

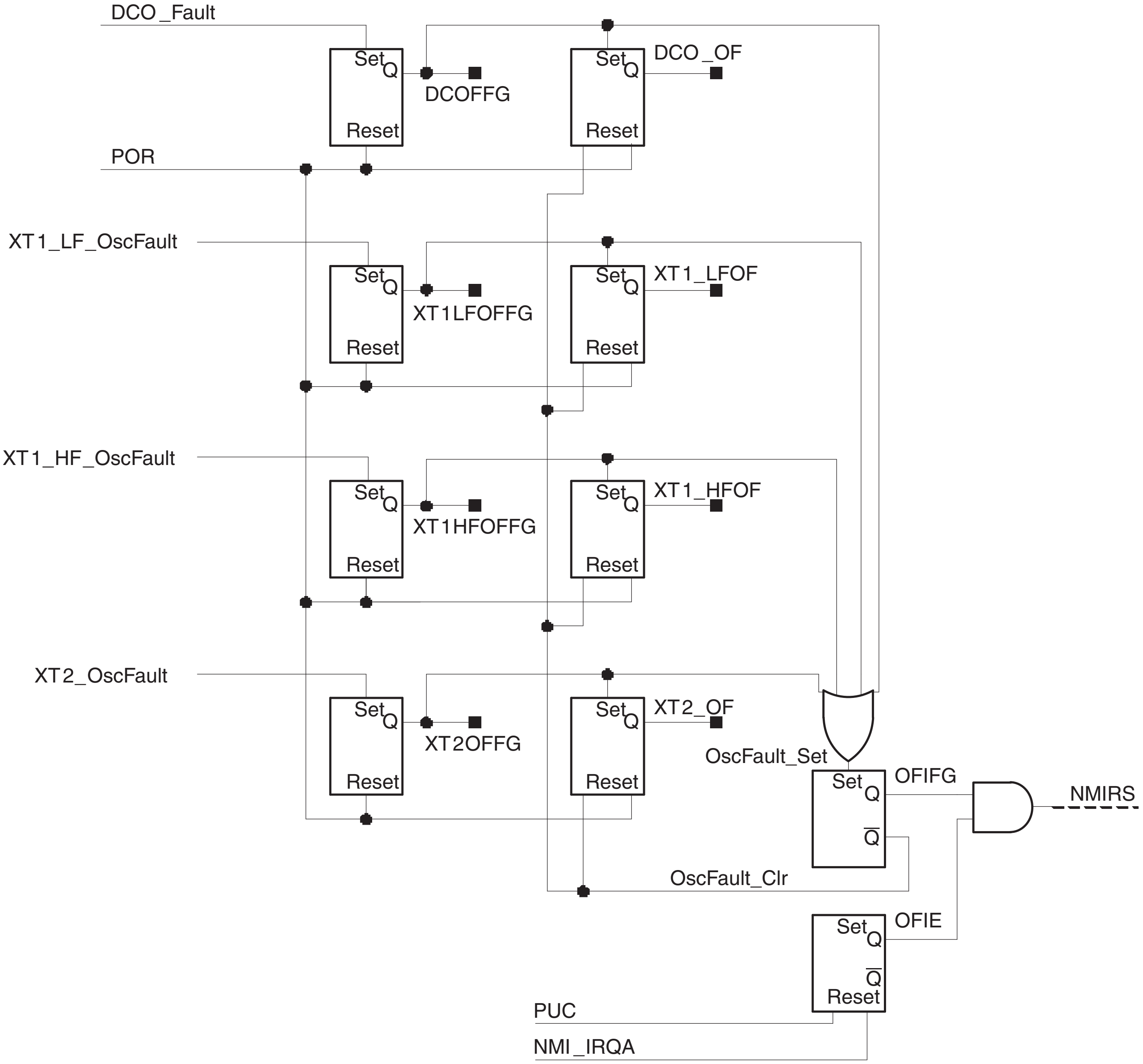


Figure 3-4. Oscillator Fault Logic

Note: Fault Conditions

DCO_Fault: DCOFFG is set if DCOx bits in UCSCTL0 register value equals 0 or 31.

XT1_LF_OscFault: This signal is set after the XT1 (LF mode) oscillator has stopped operation and cleared after operation resumes. The fault condition will cause XT1LFOFFG to be set and will remain set. If the user clears XT1LFOFFG and the fault condition still exists, XT1LFOFFG will remain set.

XT1_HF_OscFault: This signal is set after the XT1 (HF mode) oscillator has stopped operation and cleared after operation resumes. The fault condition will cause XT1HFOFFG to be set and will remain set. If the user clears XT1HFOFFG and the fault condition still exists, XT1HFOFFG will remain set.

XT2_OscFault: This signal is set after the XT2 oscillator has stopped operation and cleared after operation resumes. The fault condition will cause XT2OFFG to be set and will remain set. If the user clears XT2OFFG and the fault condition still exists, XT2OFFG will remain set.

Note: Fault Logic

Please note that as long as a fault condition still exists, the OFIFG will remain set. The application must take special care when clearing the OFIFG signal. If no fault condition remains when the OFIFG signal is cleared, the clock logic will switch back to the original user settings prior to the fault condition.

3.2.13 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 3-5:

- The current clock cycle continues until the next rising edge.
- The clock remains high until the next rising edge of the new clock.
- The new clock source is selected and continues with a full high period.

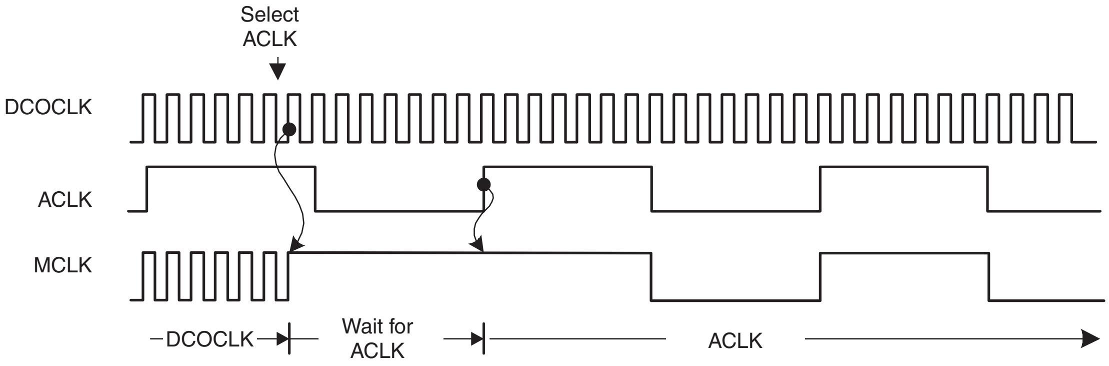


Figure 3-5. Switch MCLK from DCOCLK to ACLK

3.3 MODOSC Module Oscillator

The Unified Clock System module also supports an internal oscillator, MODOSC that is used by the Flash Memory Controller module, and optionally, by other modules in the system. The MODOSC sources MODCLK.

3.3.1 MODOSC Operation

To conserve power, MODOSC is powered down when not needed and enabled only when required. When the MODOSC source is required, the respective module requests it. The MODOSC is enabled based on unconditional and conditional requests. Setting MODOSCREQEN will enable conditional requests. Unconditional requests are always enabled. It is not necessary to set the MODOSCREQEN for modules that utilize unconditional requests e.g. Flash controller, ADC12_A.

The Flash Memory Controller only requires MODCLK when performing write or erase operations. When performing such operations, the Flash Memory Controller issues an unconditional request for the MODOSC source. Upon doing so, the MODOSC source will be enabled, if not already enabled from other modules' previous requests.

The ADC12_A may optionally use the MODOSC as a clock source for its conversion clock. The user chooses the ADC12OSC as the conversion clock source. During a conversion, the ADC12_A module issues an unconditional request for the ADC12OSC clock source. Upon doing so, the MODOSC source will be enabled, if not already enabled form other modules' previous requests.

3.4 Unified Clock System Module Registers

The Unified Clock System module registers are listed in Table 3-1. The base address can be found in the device specific datasheet. The address offset is listed in Table 3-1.

Table 3-1. Unified Clock System Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
UCS Control register 0	UCSCTL0	Read/write	Word	00h	0000h
	UCSCTL0_L	Read/write	Byte	00h	00h
	UCSCTL0_H	Read/write	Byte	01h	00h
UCS Control register 1	UCSCTL1	Read/write	Word	02h	0020h
	UCSCTL1_L	Read/write	Byte	02h	20h
	UCSCTL1_H	Read/write	Byte	03h	00h
UCS Control register 2	UCSCTL2	Read/write	Word	04h	101Fh
	UCSCTL2_L	Read/write	Byte	04h	1Fh
	UCSCTL2_H	Read/write	Byte	05h	10h
UCS Control register 3	UCSCTL3	Read/write	Word	06h	0000h
	UCSCTL3_L	Read/write	Byte	06h	00h
	UCSCTL3_H	Read/write	Byte	07h	00h
UCS Control register 4	UCSCTL4	Read/write	Word	08h	0044h
	UCSCTL4_L	Read/write	Byte	08h	44h
	UCSCTL4_H	Read/write	Byte	09h	00h
UCS Control register 5	UCSCTL5	Read/write	Word	0Ah	0000h
	UCSCTL5_L	Read/write	Byte	0Ah	00h
	UCSCTL5_H	Read/write	Byte	0Bh	00h
UCS Control register 6	UCSCTL6	Read/write	Word	0Ch	C1CDh
	UCSCTL6_L	Read/write	Byte	0Ch	CDh
	UCSCTL6_H	Read/write	Byte	0Dh	C1h
UCS Control register 7	UCSCTL7	Read/write	Word	0Eh	0703h
	UCSCTL7_L	Read/write	Byte	0Eh	03h
	UCSCTL7_H	Read/write	Byte	0Fh	07h
UCS Control register 8	UCSCTL8	Read/write	Word	10h	0307h
	UCSCTL8_L	Read/write	Byte	10h	07h
	UCSCTL8_H	Read/write	Byte	11h	03h

UCSCTL0, UCSCTL0_H, UCSCTL0_L, Unified Clock System Control Register 0

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved			**DCO**				
r0	r0	r0	rw-0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
MOD					**Reserved**		
rw-0	rw-0	rw-0	rw-0	rw-0	r0	r0	r0

Reserved Reserved. Reads back as 0.
UCSCTL0 Bits 15-13
UCSCTL0_H Bits 7-5

DCO DCO tap selection. These bits select the DCO tap and are modified automatically during FLL operation.
UCSCTL0 Bits 12-8
UCSCTL0_H Bits 4-0

MOD Modulation bit counter. These bits select the modulation pattern. All MOD bits are modified automatically during FLL operation. The DCO register value is incremented when the modulation bit counter rolls over from 31 to 0. If the modulation bit counter decrements from 0 to the maximum count, the DCO register value is also decremented.
UCSCTL0 Bits 7-3
UCSCTL0_L Bits 7-3

Reserved Reserved. Reads back as 0.
UCSCTL0 Bits 2-0
UCSCTL0_L Bits 2-0

UCSCTL1, UCSCTL1_H, UCSCTL1_L, Unified Clock System Control Register 1

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Reserved	**DCORSEL**			**Reserved**		**Reserved**	**DISMOD**
r0	rw-0	rw-1	rw-0	r0	r0	rw-0	rw-0

Reserved Reserved. Reads back as 0.
UCSCTL1 Bits 15-8
UCSCTL1_H Bits 7-0

Reserved Reserved. Reads back as 0.
UCSCTL1 Bit 7
UCSCTL1_L Bit 7

DCORSEL DCO frequency range select. These bits select the DCO frequency range of operation.
UCSCTL1 Bits 6-4
UCSCTL1_L Bits 6-4

Reserved Reserved. Reads back as 0.
UCSCTL1 Bits 3-2
UCSCTL1_L Bits 3-2

Reserved Reserved. Reads back as 0.
UCSCTL1 Bit 1
UCSCTL1_L Bit 1

DISMOD Modulation. This bit enables/disables the modulation.
UCSCTL1 Bit 0
UCSCTL1_L Bit 0

0 Modulation enabled

1 Modulation disabled

UCSCTL2, UCSCTL2_H, UCSCTL2_L, Unified Clock System Control Register 2

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved	**FLLD**			**Reserved**		**FLLN**	
r0	rw-0	rw-0	rw-1	r0	r0	rw-0	rw-0

7	6	5	4	3	2	1	0
FLLN							
rw-0	rw-0	rw-0	rw-1	rw-1	rw-1	rw-1	rw-1

Reserved
UCSCTL2 Bit 15
UCSCTL2_H Bit 7

Reserved. Reads back as 0.

FLLD
UCSCTL2 Bits 14-12
UCSCTL2_H Bits 6-4

FLL loop divider. These bits select the DCO frequency range of operation.

000	f_{DCOCLK}/1
001	f_{DCOCLK}/2
010	f_{DCOCLK}/4
011	f_{DCOCLK}/8
100	f_{DCOCLK}/16
101	f_{DCOCLK}/32
110	Reserved for future use. Defaults to f_{DCOCLK}/32.
111	Reserved for future use. Defaults to f_{DCOCLK}/32.

Reserved
UCSCTL2 Bits 11-10
UCSCTL2_H Bits 3-2

Reserved. Reads back as 0.

FLLN
UCSCTL2 Bits 9-0
UCSCTL2_H Bits 1-0
UCSCTL2_L Bits 7-0

Multiplier bits. These bits set the multiplier value N of the DCO. N must be greater than zero. Writing zero to FLLN causes N to be set to one.

UCSCTL3, UCSCTL3_H, UCSCTL3_L, Unified Clock System Control Register 3

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Reserved	**SELREF**			**Reserved**	**FLLREFDIV**		
r0	rw-0	rw-0	rw-0	r0	rw-0	rw-0	rw-0

Reserved
UCSCTL3 Bits 15-8
UCSCTL3_H Bits 7-0

Reserved. Reads back as 0.

Reserved
UCSCTL3 Bit 7
UCSCTL3_L Bit 7

Reserved. Reads back as 0.

SELREF
UCSCTL3 Bits 6-4
UCSCTL3_L Bits 6-4

FLL reference select. These bits select the FLL reference clock source.

000 XT1CLK
001 Reserved for future use. Defaults to XT1CLK.
010 REFOCLK
011 Reserved for future use. Defaults to REFOCLK.
100 Reserved for future use. Defaults to REFOCLK.
101 XT2CLK when available, otherwise REFOCLK.
110 Reserved for future use. XT2CLK when available, otherwise REFOCLK.
111 Reserved for future use. XT2CLK when available, otherwise REFOCLK.

Reserved
UCSCTL3 Bit 3
UCSCTL3_L Bit 3

Reserved. Reads back as 0.

FLLREFDIV
UCSCTL3 Bits 2-0
UCSCTL3_L Bits 2-0

FLL reference divider. These bits define the divide factor for $f_{FLLREFCLK}$. The divided frequency is used as the FLL reference frequency.

000 $f_{FLLREFCLK}/1$
001 $f_{FLLREFCLK}/2$
010 $f_{FLLREFCLK}/4$
011 $f_{FLLREFCLK}/8$
100 $f_{FLLREFCLK}/12$
101 $f_{FLLREFCLK}/16$
110 Reserved for future use. Defaults to $f_{FLLREFCLK}/16$.
111 Reserved for future use. Defaults to $f_{FLLREFCLK}/16$.

UCSCTL4, UCSCTL4_H, UCSCTL4_L, Unified Clock System Control Register 4

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved					**SELA**		
r0	r0	r0	r0	r0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
Reserved	**SELS**			**Reserved**	**SELM**		
r0	rw-1	rw-0	rw-0	r0	rw-1	rw-0	rw-0

Reserved
UCSCTL4 Bits 15-11
UCSCTL4_H Bits 7-3

Reserved. Reads back as 0.

SELA
UCSCTL4 Bits 10-8
UCSCTL4_H Bits 2-0

Selects the ACLK source

000	XT1CLK
001	VLOCLK
010	REFOCLK
011	DCOCLK
100	DCOCLKDIV
101	XT2CLK when available, otherwise DCOCLKDIV
110	Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV.
111	Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV.

Reserved
UCSCTL4 Bit 7
UCSCTL4_L Bit 7

Reserved. Reads back as 0.

SELS
UCSCTL4 Bits 6-4
UCSCTL4_L Bits 6-4

Selects the SMCLK source

000	XT1CLK
001	VLOCLK
010	REFOCLK
011	DCOCLK
100	DCOCLKDIV
101	XT2CLK when available, otherwise DCOCLKDIV
110	Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV.
111	Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV.

Reserved
UCSCTL4 Bit 3
UCSCTL4_L Bit 3

Reserved. Reads back as 0.

SELM
UCSCTL4 Bits 2-0
UCSCTL4_L Bits 2-0

Selects the MCLK source

000	XT1CLK
001	VLOCLK
010	REFOCLK
011	DCOCLK
100	DCOCLKDIV
101	XT2CLK when available, otherwise DCOCLKDIV
110	Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV.
111	Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLKDIV.

UCSCTL5, UCSCTL5_H, UCSCTL5_L, Unified Clock System Control Register 5

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved	**DIVPA**			**Reserved**	**DIVA**		
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
Reserved	**DIVS**			**Reserved**	**DIVM**		
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Reserved
UCSCTL5 Bit 15
UCSCTL5_H Bit 7

Reserved. Reads back as 0.

DIVPA
UCSCTL5 Bits 14-12
UCSCTL5_H Bits 6-4

ACLK source divider available at external pin

000	f_{ACLK}/1
001	f_{ACLK}/2
010	f_{ACLK}/4
011	f_{ACLK}/8
100	f_{ACLK}/16
101	f_{ACLK}/32
110	Reserved for future use. Defaults to f_{ACLK}/32.
111	Reserved for future use. Defaults to f_{ACLK}/32.

Reserved
UCSCTL5 Bit 11
UCSCTL5_H Bit 3

Reserved. Reads back as 0.

DIVA
UCSCTL5 Bits 10-8
UCSCTL5_H Bits 2-0

ACLK source divider

000	f_{ACLK}/1
001	f_{ACLK}/2
010	f_{ACLK}/4
011	f_{ACLK}/8
100	f_{ACLK}/16
101	f_{ACLK}/32
110	Reserved for future use. Defaults to f_{ACLK}/32.
111	Reserved for future use. Defaults to f_{ACLK}/32.

Reserved
UCSCTL5 Bit 7
UCSCTL5_L Bit 7

Reserved. Reads back as 0.

DIVS
UCSCTL5 Bits 6-4
UCSCTL5_L Bits 6-4

SMCLK source divider

000	f_{ACLK}/1
001	f_{ACLK}/2
010	f_{ACLK}/4
011	f_{ACLK}/8
100	f_{ACLK}/16
101	f_{ACLK}/32
110	Reserved for future use. Defaults to f_{ACLK}/32.
111	Reserved for future use. Defaults to f_{ACLK}/32.

Reserved UCSCTL5 UCSCTL5_L	 Bit 3 Bit 3	Reserved. Reads back as 0.
DIVM UCSCTL5 UCSCTL5_L	 Bits 2-0 Bits 2-0	MCLK source divider 000 $f_{ACLK}/1$ 001 $f_{ACLK}/2$ 010 $f_{ACLK}/4$ 011 $f_{ACLK}/8$ 100 $f_{ACLK}/16$ 101 $f_{ACLK}/32$ 110 Reserved for future use. Defaults to $f_{ACLK}/32$. 111 Reserved for future use. Defaults to $f_{ACLK}/32$.

UCSCTL6, UCSCTL6_H, UCSCTL6_L, Unified Clock System Control Register 6

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
XT2DRIVE		**Reserved**	**XT2BYPASS**	**Reserved**			**XT2OFF**
rw-1	rw-1	r0	rw-0	r0	r0	r0	rw-1
7	6	5	4	3	2	1	0
XT1DRIVE		**XTS**	**XT1BYPASS**	**XCAP**		**SMCLKOFF**	**XT1OFF**
rw-1	rw-1	rw-0	rw-0	rw-1	rw-1	rw-0	rw-1

XT2DRIVE
UCSCTL6 Bits 15-14
UCSCTL6_H Bits 7-6

The XT2 oscillator current can be adjusted to its drive needs. Initially, it starts with the highest supply current for reliable and quick startup. If needed, user software can reduce the drive strength.

00 Lowest current consumption. XT2 oscillator operating range is 4 MHz to 8 MHz.

01 Increased drive strength XT2 oscillator. XT2 oscillator operating range is 8 MHz to 16 MHz.

10 Increased drive capability XT2 oscillator. XT2 oscillator operating range is 16 MHz to 24 MHz.

11 Maximum drive capability and maximum current consumption for both XT2 oscillator. XT2 oscillator operating range is 24 MHz to 32 MHz.

Reserved
UCSCTL6 Bit 13
UCSCTL6_H Bit 5

Reserved. Reads back as 0.

XT2BYPASS
UCSCTL6 Bit 12
UCSCTL6_H Bit 4

XT2 bypass select

0 XT2 sourced internally

1 XT2 sourced externally from pin

Reserved
UCSCTL6 Bits 11-9
UCSCTL6_H Bits 3-1

Reserved. Reads back as 0.

XT2OFF
UCSCTL6 Bit 8
UCSCTL6_H Bit 0

Turns off the XT2 oscillator.

0 XT2 is on if XT2 is selected via the port selection and XT2 is not in bypass mode of operation.

1 XT2 is off if it is not used as a source for ACLK, MCLK, or SMCLK or is not used as a reference source required for FLL operation.

XT1DRIVE
UCSCTL6 Bits 7-6
UCSCTL6_L Bits 7-6

The XT1 oscillator current can be adjusted to its drive needs. Initially, it starts with the highest supply current for reliable and quick startup. If needed, user software can reduce the drive strength.

00 Lowest current consumption for XT1 LF mode. XT1 oscillator operating range in HF mode is 4 MHz to 8 MHz.

01 Increased drive strength for XT1 LF mode. XT1 oscillator operating range in HF mode is 8 MHz to 16 MHz.

10 Increased drive capability for XT1 LF mode. XT1 oscillator operating range in HF mode is 16 MHz to 24 MHz.

11 Maximum drive capability and maximum current consumption for XT1 LF mode. XT1 oscillator operating range in HF mode is 24 MHz to 32 MHz.

XTS
UCSCTL6 Bit 5
UCSCTL6_L Bit 5

XT1 mode select

0 Low frequency mode. XCAP bits define the capacitance at the XIN and XOUT pins.

1 High frequency mode. XCAP bits are not used.

XTS
UCSCTL6 Bit 4
UCSCTL6_L Bit 4

XT1 bypass select

0 XT1 sourced internally

1 XT1 sourced externally from pin

XCAP
UCSCTL6 Bits 3-2
UCSCTL6_L Bits 3-2

Oscillator capacitor selection. These bits select the capacitors applied to the LF crystal or resonator in the low-frequency mode (XTS = 0). The effective capacitance (seen by the crystal) is $C_{eff} \approx (C_{XIN} + 2 \text{ pF})/2$. It is assumed, that $C_{XIN} = C_{XOUT}$ and that a parasitic capacitance of 2 pF is added by the package and the printed circuit board. For details about the typical internal and the effective capacitors refer to the device specific datasheet.

SMCLKOFF UCSCTL6 UCSCTL6_L	 Bit 1 Bit 1	SMCLK off. This bit turns off the SMCLK. 0 SMCLK off 1 SMCLK on
XT1OFF UCSCTL6 UCSCTL6_L	 Bit 0 Bit 0	XT1 off. This bit turns off the XT1. 0 XT1 is on if XT1 is selected via the port selection and XT1 is not in bypass mode of operation. 1 XT1 is off if it is not used as a source for ACLK, MCLK, or SMCLK or is not used as a reference source required for FLL operation.

UCSCTL7, UCSCTL7_H, UCSCTL7_L, Unified Clock System Control Register 7

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved		**FLLWARNEN**	**FLLULIE**	**FLLUNLOCKHIS**		**FLLUNLOCK**	
r0	r0	rw-0	rw-(0)	rw-(1)	rw-(1)	r-1	r-1

7	6	5	4	3	2	1	0
Reserved			**FLLULIFG**	**XT2OFFG**	**XT1HFOFFG**	**XT1LFOFFG**	**DCOFFG**
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(1)	rw-(1)

Reserved
UCSCTL7 Bit 15
UCSCTL7_H Bit 7

Reserved. Reads back as 0.

FLLWARNEN
UCSCTL7 Bit 13
UCSCTL7_H Bit 5

Warning enable. If this bit is set then an interrupt is generated based on the FLLUNLOCKHIS bits. If FLLUNLOCKHIS is not equal to 00 then an OFIFG is generated.

0 The FLLUNLOCKHIS status cannot set OFIFG.

1 The FLLUNLOCKHIS status can set OFIFG.

FLLULIE
UCSCTL7 Bit 12
UCSCTL7_H Bit 4

FLL unlock interrupt enable. If the FLLUIE bit is set a reset (PUC) is triggered if the FLLULIFG is set. The FLLULIFG indicates when FLLUNLOCK bits equal to 10. The FLLULIE is automatically cleared upon servicing the event. If FLLUIE is cleared (0), no PUC can be triggered by the FLLULIFG.

FLLUNLOCKHIS
UCSCTL7 Bits 11-10
UCSCTL7_H Bits 3-2

Unlock history bits. These bits indicate the FLL unlock condition history. As soon as any unlock condition happens the respective bits are set and remain set until cleared by software by writing 0 to it or by a POR.

00 FLL is locked. No unlock situation has been detected since the last reset of these bits.

01 DCOCLK has been too low since the bits were cleared.

10 DCOCLK has been too fast since the bits were cleared.

11 DCOCLK has been both too fast and too slow since the bits were cleared.

FLLUNLOCK
UCSCTL7 Bits 9-8
UCSCTL7_H Bits 1-0

Unlock. These bits indicate the current FLL unlock condition. These bits are both set as long as the DCOFFG flag is set.

00 FLL is locked. No unlock condition currently active.

01 DCOCLK is currently too low.

10 DCOCLK is currently too fast.

11 DCOERROR. DCO out of range.

Reserved
UCSCTL7 Bits 7-5
UCSCTL7_L Bits 7-5

Reserved. Reads back as 0.

FLLULIFG
UCSCTL7 Bit 4
UCSCTL7_L Bit 4

FLL unlock interrupt flag. This flag is set when the FLLUNLOCK bits equal 10b (DCO too fast) If the FLLUIFE is also set, a PUC will be triggered when FLLUIFG is set.

0 FLLUNLOCK bits not equal to 10b

1 FLLUNLOCK bits equal to 10b

XT2OFFG
UCSCTL7 Bit 3
UCSCTL7_L Bit 3

XT2 oscillator fault flag. If this bit is set, the OFIFG flag is also set. XT2OFFG is set if a XT2 fault condition exists. The XT2OFFG can be cleared via software. If the XT2 fault condition still remains, the XT2OFFG is set.

0 No fault condition occurred after the last reset.

1 XT2 fault. An XT2 fault occurred after the last reset.

XT1HFOFFG
UCSCTL7 Bit 2
UCSCTL7_L Bit 2

XT1 oscillator fault flag (HF mode). If this bit is set, the OFIFG flag is also set. XT1HFOFFG is set if a XT1 fault condition exists. The XT1HFOFFG can be cleared via software. If the XT1 fault condition still remains, the XT1HFOFFG is set.

0 No fault condition occurred after the last reset.

1 XT1 fault. An XT1 fault occurred after the last reset.

XT1LFOFFG
UCSCTL7 Bit 1
UCSCTL7_L Bit 1

XT1 oscillator fault flag (LF mode). If this bit is set, the OFIFG flag is also set. XT1LFOFFG is set if a XT1 fault condition exists. The XT1LFOFFG can be cleared via software. If the XT1 fault condition still remains, the XT1LFOFFG is set.

0 No fault condition occurred after the last reset.

1 XT1 fault (LF mode). A XT1 fault occurred after the last reset.

DCOFFG
UCSCTL7 Bit 0
UCSCTL7_L Bit 0

DCO fault flag. If this bit is set, the OFIFG flag is also set. The DCOFFG bit is set if DCOx = 0 or DCOx = 31. The DCOOFFG can be cleared via software. If the DCO fault condition still remains, the DCOOFFG is set. As long as DCOFFG is set, FLLUNLOCK shows the DCOERROR condition.

0 No fault condition occurred after the last reset.

1 DCO fault. A DCO fault occurred after the last reset.

Note: The FLLWAREN, FLLUIE, FLLUNLOCKHIS, and FLLUNLOCK bits and features are currently under evaluation and may not be present in the final product.

UCSCTL8, UCSCTL8_H, UCSCTL8_L, Unified Clock System Control Register 8

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved						**Reserved**	**Reserved**
r0	r0	r0	r0	r0	r0	rw-(1)	rw-(1)

7	6	5	4	3	2	1	0
Reserved			**Reserved**	**MODOSCREQEN**	**Reserved**	**Reserved**	**Reserved**
r0	r0	r0	rw-(1)	rw-(0)	rw-(1)	rw-(1)	rw-(1)

Reserved
UCSCTL8 Bits 15-10
UCSCTL8_H Bits 7-2

Reserved. Reads back as 0.

Reserved
UCSCTL8 Bit 9
UCSCTL8_H Bit 1

Reserved. Must always be written as 1.

Reserved
UCSCTL8 Bit 8
UCSCTL8_H Bit 0

Reserved. Must always be written as 1.

Reserved
UCSCTL8 Bits 7-5
UCSCTL8_L Bits 7-5

Reserved. Reads back as 0.

Reserved
UCSCTL8 Bit 4
UCSCTL8_L Bit 4

Reserved. Must always be written as 1.

MODOSCREQEN
UCSCTL8 Bit 3
UCSCTL8_L Bit 3

MODOSC clock request enable. Setting this enables module requests for the MODOSC.

0 MODOSC requests are disabled.

1 MODOSC requests are enabled.

Reserved
UCSCTL8 Bit 2
UCSCTL8_L Bit 2

Reserved. Must always be written as 1.

Reserved
UCSCTL8 Bit 1
UCSCTL8_L Bit 1

Reserved. Must always be written as 1.

Reserved
UCSCTL8 Bit 0
UCSCTL8_L Bit 0

Reserved. Must always be written as 1.

Chapter 4

SLAU208–June 2008

Power Management Module and Supply Voltage Supervisor

This chapter describes the operation of the Power Management Module (PMM) and the Supply Voltage Supervisors (SVS) of the MSP430x5xx devices.

Topic **Page**

4.1 PMM Introduction

The PMM features include:

- Wide supply voltage (DV_{CC}) range: 1.8 V to 3.6 V
- Core voltage (V_{CORE}) generation: 1.4 V, 1.6 V, 1.8 V, and 1.9 V (typical)
- Brown-out-reset (BOR)
- Supply voltage supervisor for DV_{CC} and V_{CORE}
- Supply voltage monitor for DV_{CC} and V_{CORE} with eight programmable levels
- Software accessible power-fail conditions
- Software selectable power-on-reset at power-fail condition
- I/O protection at power-fail condition
- Software selectable supervisor or monitor state output (optional)

The main digital logic of the MSP430 device requires a voltage that is lower than the range allowed by DV_{CC}. For this reason, the PMM incorporates an integrated low-dropout voltage regulator (LDO) that generates a secondary core voltage rail, V_{CORE}. The core voltage is programmable in four steps to allow power consumption optimization. The required minimum voltage for the core depends on the selected MCLK rate, as shown inFigure 4-1

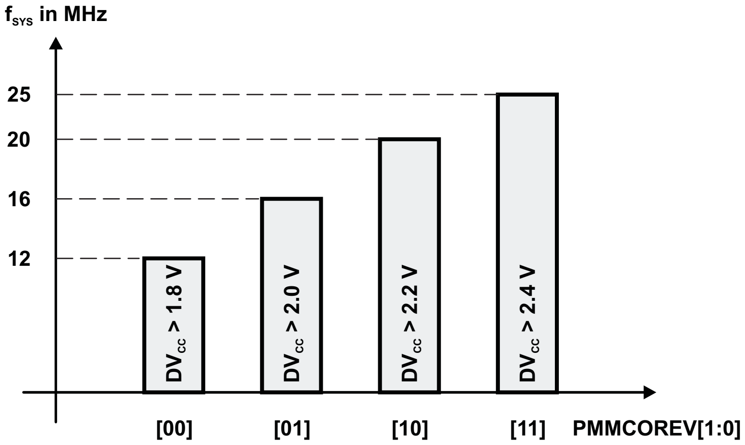


Figure 4-1. System Frequency and Supply/Core Voltages

DV_{CC} and V_{CORE} can be supervised and monitored. Both supervision and monitoring detect when a voltage has fallen under a specific threshold. Generally speaking, supervision results in a power-on reset (POR) event, while monitoring results in the generation of an interrupt flag, which software can then handle. As such, DV_{CC} (the high-side of the LDO) is supervised and monitored by the high-side supervisor (SVS_H) and high-side monitor (SVM_H), respectively. V_{CORE} (the low-side of the LDO) is supervised and monitored by the low-side supervisor (SVS_L) and low-side monitor SVM_L), respectively. The block diagram of the PMM is shown in Figure 4-2.

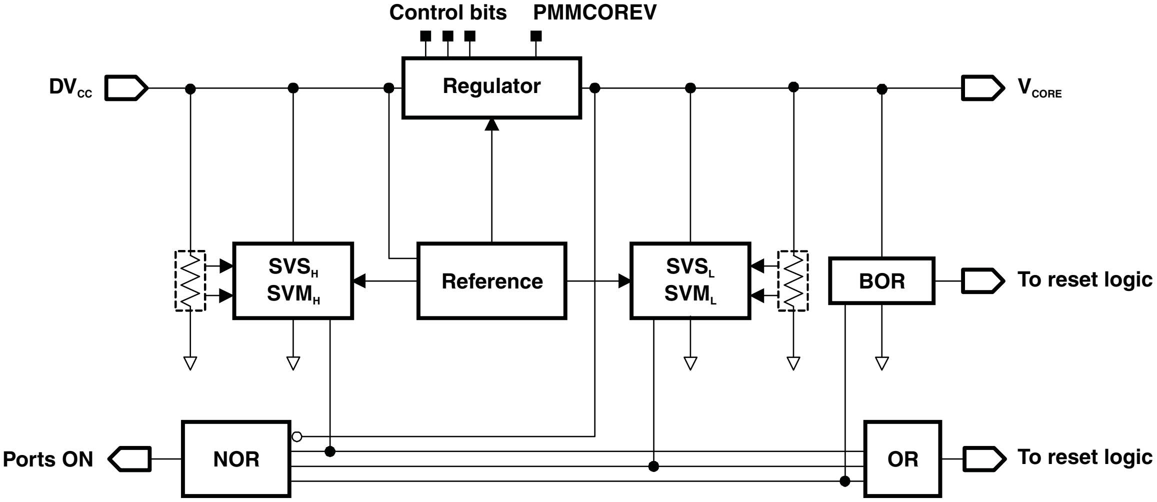


Figure 4-2. PMM Block Diagram

The I/Os and all analog modules including the oscillators are supplied by DV_{CC}. Memories (Flash and RAM) and the digital modules are supplied by the core voltage (V_{CORE}).

4.2 PMM Operation

The PMM can be configured for four possible levels of core voltage, correlating with four system speed levels. For a given core voltage, there is an associated set of thresholds.

The PMM regulator supports two different load settings. The low current mode can be used if the system consumes less than I(V_{CORE}) ≤ 30 μA (see device specific datasheet). Higher system currents are supported by the full-performance mode. The full-performance mode is required if:

- any internal high frequency clock (>32 kHz) is used by any module
- an interrupt is executed
- JTAG is active
- in active mode, LPM0, or LPM1

The PMM supports four system speed levels by adjusting the core voltage.

The selected core voltage level remains unchanged when entering a low-power mode. During the system start-up the SVS_H and SVS_L functions are enabled. The typical values of are shown in Table 4-1 for DV_{CC} (high voltage) domain and Table 4-2 for the V_{CORE} (low voltage) domain. Figure 4-3 shows how the system behaves during power-up. If both the high side and the low side voltage supervisors levels are met the system reset is released.

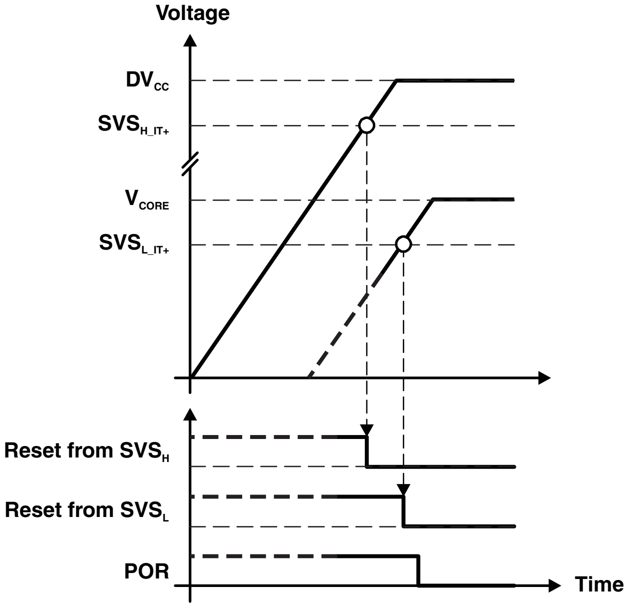


Figure 4-3. Powering Up the System

Once the system is up and running, both voltage domains are supervised and monitored as long as the respective modules are enabled. The PMM supply voltage supervisor levels selected after reset are 1.74 V (typical) for the high side and 1.34 V (typical) for the low side. Once both levels are exceeded the system starts operation. The device specific values can be found in the device specific data sheet.

A power-fail at the high or low side voltage domains may cause system failures. Both high and low side voltage levels are monitored by the supply voltage monitors. If DV_{CC} falls below the supply voltage monitor level for the high side, the supply voltage monitor interrupt flag for the high side, SVMHIFG, is set. Similary, if V_{CORE} falls below the supply voltage monitor level of the low side, the supply voltage monitor interrupt flag for the low side, SVMLIFG, is set.

When DV_{CC} rises above the supply voltage monitor level of the high side, the supply voltage reached interrupt flag SVMHVLRIFG is set. Similary, if V_{CORE} rises above the supply voltage monitor level of the low side, the supply voltage reached interrupt flag SVMLVLRIFG is set. When both the high side and low side levels have been reached, the system can continue to operate normally.

Supply voltages below the supply voltage supervisor levels cause a system reset (POR) if enabled. Setting SVSHPE will cause a POR when SVSHIFG is set. Similarly, setting SVSLPE will cause a POR when SVSLIFG is set.

Both the supply voltage supervisor and monitor interrupt flags remain set unless cleared by a BOR or by software to allow the application software to determine the latest reset condition.

Figure 4-4 explains the high and low side power fails with respect to the supply voltage supervisor and monitor levels and the respective interrupt flags.

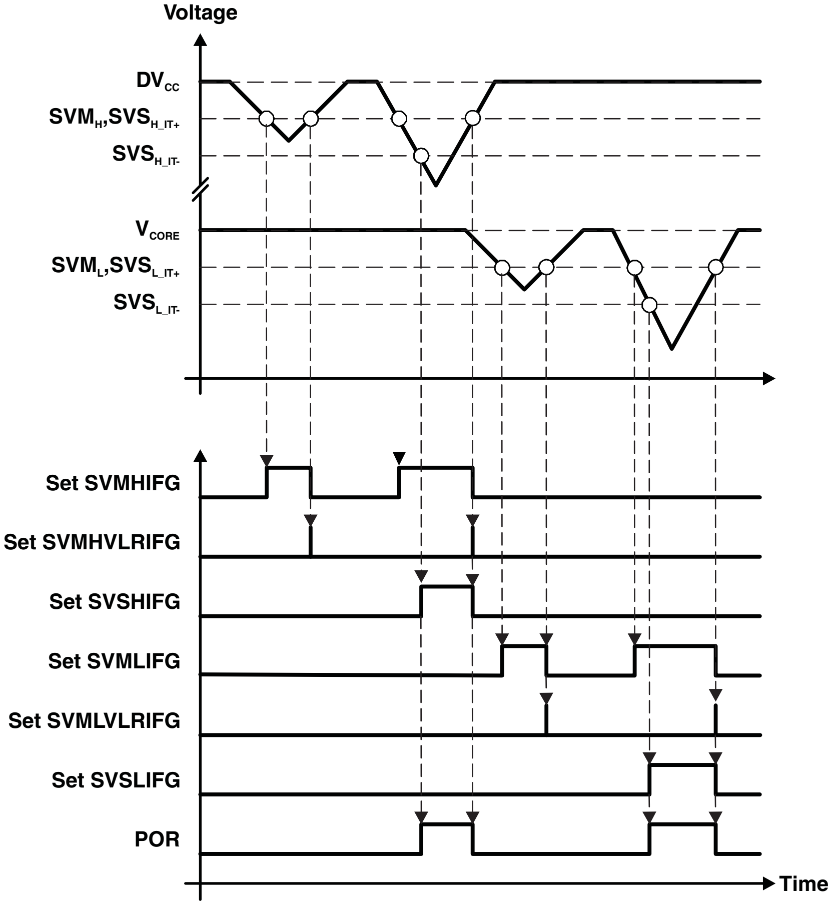


Figure 4-4. High-Side and Low-Side Voltage Failure

Table 4-1. High-Side Supply Voltage Supervisor and Monitor Levels (see the device-specific datasheet)

Parameter	High Side (DV_{CC}) Voltage			
$DV_{CC(min)}$ in V	≥1.8[1]	≥2.0	≥2.2	≥2.4
SVM_H – $V_{(SVMH\_IT+,typ)}$ in V	1.74[1]	1.94	2.14	2.26
SVM_H – $V_{(SVMH\_IT-,typ)}$ in V	1.74[1]	1.94	2.14	2.26
SVS_H – $V_{(SVSH\_IT+,max)}$ in V	1.79[1]	1.99	2.19	2.31
SVS_H – $V_{(SVSH\_IT+,min)}$ in V	1.69[1]	1.89	2.09	2.21
SVS_H – $V_{(SVSH\_IT-,max)}$ in V	1.69[1]	1.89	2.09	2.21
SVS_H – $V_{(SVSH\_IT-,min)}$ in V	1.59[1]	1.79	1.99	2.11

[1] Default value after reset

Table 4-2. Low-Side Supply Voltage Supervisor and Monitor Levels (see the device specific datasheet)

Parameter	Low Side (V_{CORE}) Voltage			
PMMCOREV	0[1]	1	2	3
$V_{CORE(typ)}$ in V	1.40[1]	1.60	1.80	1.92
SVM_L – $V_{(SVML\_IT+,typ)}$ in V	1.34[1]	1.54	1.74	1.84
SVM_L – $V_{(SVML\_IT-,typ)}$ in V	1.34[1]	1.54	1.74	1.84
SVS_L – $V_{(SVSL\_IT+,max)}$ in V	1.39[1]	1.59	1.79	1.89
SVS_L – $V_{(SVSL\_IT+,min)}$ in V	1.29[1]	1.49	1.69	1.79
SVS_L – $V_{(SVSL\_IT-,max)}$ in V	1.32[1]	1.52	1.72	1.82
SVS_L – $V_{(SVSL\_IT-,min)}$ in V	1.22[1]	1.42	1.62	1.72

[1] Default value after reset

4.2.1 Supply Voltage Supervisor and Monitor – High Side

The high side supply voltage supervisor/monitor operates in active mode and in the low-power modes. To save power the operation speed can be reduced (default: SVMHFP=0, SVSHFP=0). The blockdiagram is shown in Figure 4-5.

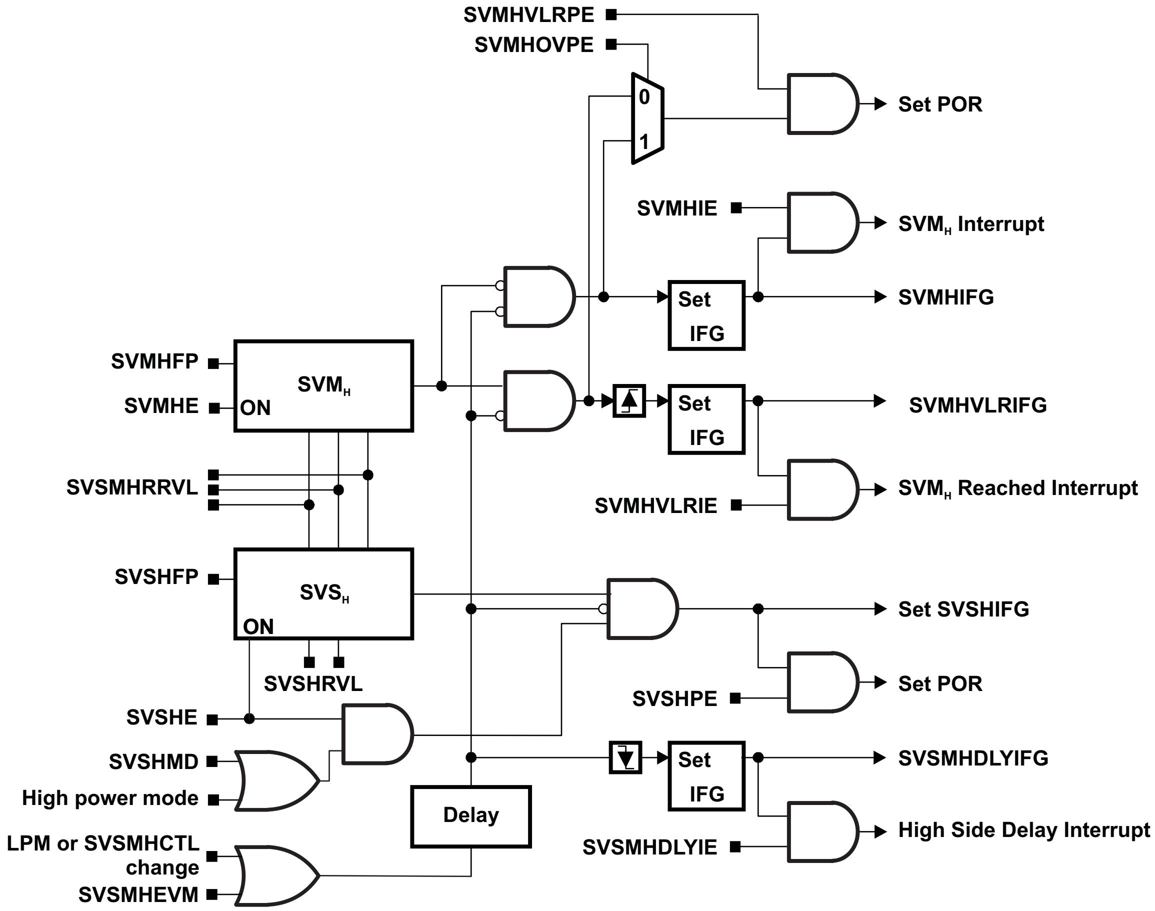


Figure 4-5. High-Side Supply Voltage Supervisor and Monitor

The SVM_H module is enabled by setting SVMHE=1. Its power consumption can be reduced by setting SVMHFP=0. The voltage reset release level is defined by SVSMHRRVL. A rising DV_{CC} level crossing the SVM_H level sets the SVMHVLRIFG interrupt flag. An interrupt is also triggered if SVMHVLRIE = 1. A falling DV_{CC} level crossing the SVMH level sets the SVMHIFG interrupt flag. An interrupt is also triggered if SVMHIE = 1. When DV_{CC} remains lower than the SVM_H level and SVMHIFG is cleared by software, then it is immediately set again by hardware. If desired, a POR can also be generated if SVMHVLRPE =1 and SVMHOVPE = 0. The SVM_H module also contains overvoltage detection. If DV_{CC} exceeds safe device operation, a POR will be generated when SVMHOVPE = 1 and SVMHVLRPE = 1.

The SVS_H module is enabled by setting SVSHE=1. Its power consumption can be reduced by setting SVSHFP=0. The voltage reset release level is defined by SVSHRVL. A falling DV_{CC} level crossing the SVS_H level sets the SVSHIFG interrupt flag, as well as causes a POR if SVSHPE = 1. When DV_{CC} remains lower than the SVS_H level and SVSHIFG is cleared by software, then it is immediately set again by hardware. The SVS_H is disabled in low-power modes 2, 3, and 4 unless the SVSHMD forces the SVS_H circuit on.

If the power mode of the SVM_H or SVS_H or a voltage level is altered, a delay element masks the interrupts and POR sources until the SVM_H and SVS_H circuits have settled. SVSMHDLYIFG is set indicating when the delay has completed. An interrupt can also be generated if SVSMHDLYIE = 1.

4.2.2 Supply Voltage Supervisor and Monitor – Low Side

The low side supply voltage supervisor/monitor operates in active mode and in the low-power modes. To save power the operation speed can be reduced (default: SVMLFP=0, SVSLFP=0). The blockdiagram is shown in Figure 4-6.

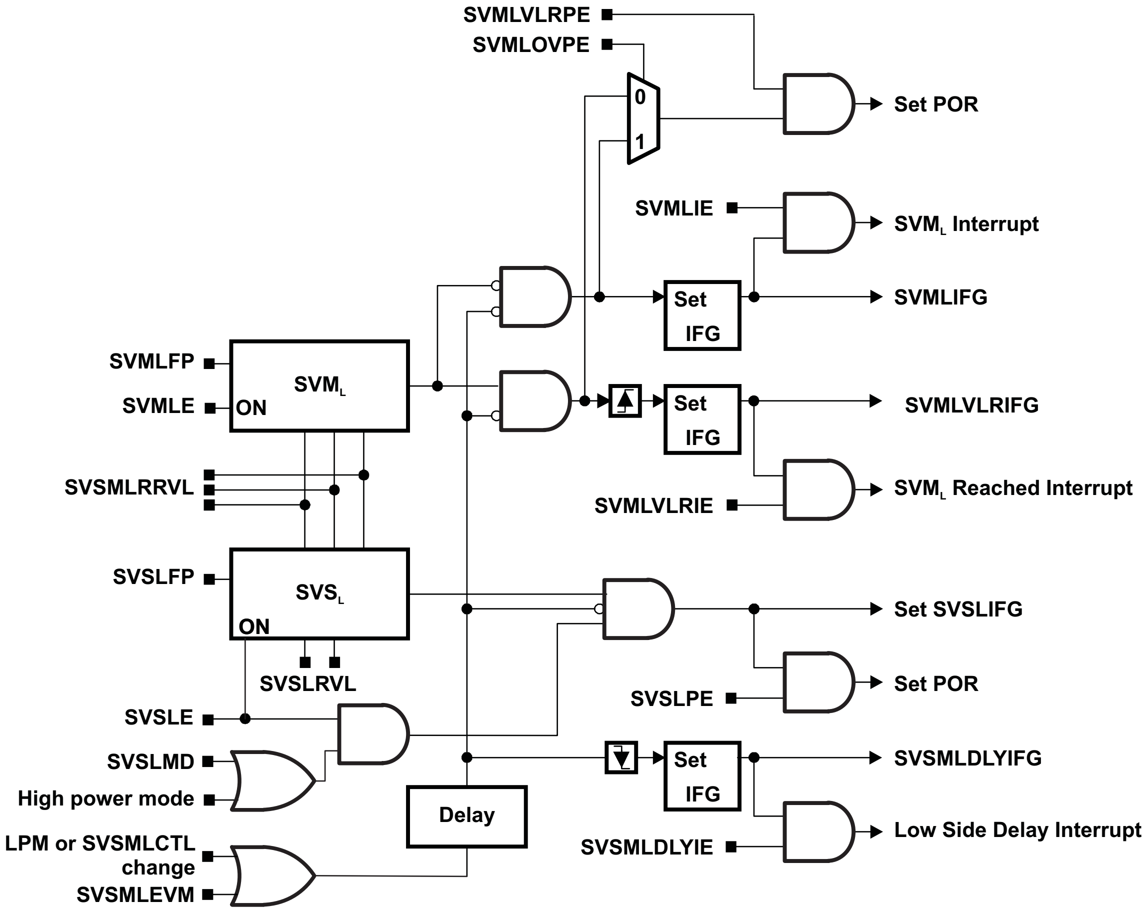


Figure 4-6. Low Side Supply Voltage Supervisor and Monitor

The SVM_L module is enabled by setting SVMLE=1. Its power consumption can be reduced by setting SVMLFP=0. The voltage reset release level is defined by SVSMLRRVL. A rising V_{CORE} level crossing the SVM_L level sets the SVMLVLRIFG interrupt flag. An interrupt is also triggered if SVMLVLRIE = 1. A falling

V_{CORE} level crossing the SVM_L level sets the SVMLIFG interrupt flag. An interrupt is also triggered if SVMLIE = 1. When V_{CORE} remains lower than the SVM_L level and SVMLIFG is cleared by software, then it is immediately set again by hardware. If desired, a POR can also be generated if SVMLVLRPE =1 and SVMLOVPE = 0. The SVM_L module also contains overvoltage detection. If V_{CORE} exceeds safe device operation, a POR will be generated when SVMLOVPE = 1 and SVMLVLRPE = 1.

The SVS_L module is enabled by setting SVSLE=1. Its power consumption can be reduced by setting SVSLFP=0. The voltage reset release level is defined by SVSLRVL. A falling V_{CORE} level crossing the SVS_L level sets the SVSLIFG interrupt flag, as well as, causes a POR if SVSLPE = 1. When V_{CORE}remains lower than the SVSL level and SVSLIFG is cleared by software then it is immediately set again by hardware. The SVS_L is disabled in low-power modes 2, 3, and 4 unless the SVSLMD forces the SVS_L circuit on.

If the power mode of the SVM_L or SVS_L or a voltage level is altered a delay element masks the interrupts and POR sources until the SVM_L and SVS_L circuits have settled. SVSMLDLYIFG is set indicating when the delay has completed. An interrupt can also be generated if SVSMLDLYIE = 1.

4.2.3 Supply Voltage Monitor Output (SVMOUT, Optional)

The state of the SVMLIFG, SVMLVLRIFG, SVMHIFG and SVMLVLRIFG can be monitored on the external SVMOUT pin. Each of these interrupt flags can be enabled (SVMLOE, SVMLVLROE, SVMHOE, SVMLVLROE) to generate an output signal. The polarity of the output is selected by the SVMOUTPOL bit. If SVMOUTPOL is set then the output is set to 1 if an enabled interrupt flag is set.

4.2.4 Performance Optimization

The CPU and the digital modules are supplied by the regulated core voltage (V_{CORE}). If the CPU has to run at full speed the core voltage has to be programmed to the highest level (see Figure 4-1). If the full CPU performance is not required the core voltage can be reduced to the desired level to save considerable power. During reset the core voltage defaults to the lowest voltage of 1.4 V (typical). The SVM_L and SVS_L levels are selected accordingly during reset. Figure 4-7 shows how the core voltage can be programmed from one level to another using the built-in supply voltage monitor and supervisor for safe operation.

Steps 1 to 4 show the sequence how the core voltage is increased while Steps 5 and 6 show how the core voltage is decreased.

Step 1: Program the SVM_L to the new level and wait for (SVSMLDLYIFG) to be set.

Step 2: Program PMMCOREV to the new V_{CORE} level.

Step 3: Wait for the voltage level reached (SVMLVLRIFG) interrupt.

Step 4: Program the SVS_L to the new level.

The desired core voltage level is reached and both the supply voltage monitor and the supply voltage supervisor levels are programmed accordingly. The system speed can now be increased.

Decreasing the core voltage level:

Step 5: Decrease the system speed to the target speed. Program the SVS_L and SVM_L level to the target values.

Step 6: Program V_{CORE} to the new level.

The delay element shown in Figure 4-6 is triggered if the configuration registers for the high- or low-side SVS or SVM is changed or if the power-mode (active mode, LPMx) is changed.

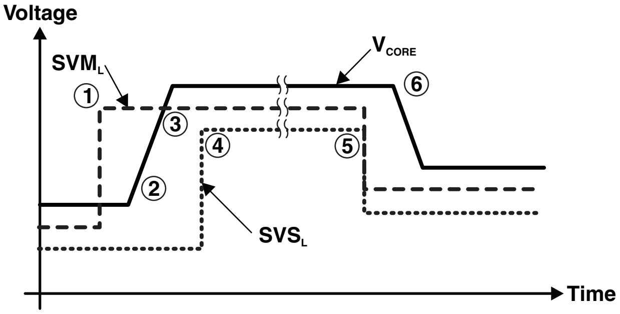


Figure 4-7. Changing V_{CORE} and the SVM_L and SVS_L Levels

4.2.5 Voltage Reference

The voltage reference supplies the voltage regulator, the supply voltage supervisors and the supply voltage monitors. In low-power modes 2, 3, and 4 the reference is clocked by a PWM signal (switched mode) to save power. In LPM5 the reference is switched off. In the other modes the reference is in static mode. In the static mode the reference is more accurate than in switched mode. In switched mode the power consumption and the accuracy of the reference can be further reduced by setting the (PMMREFACC) bit.

4.2.6 Brown-Out Reset (BOR)

The BOR circuit generates a brown-out reset signal which initializes the system at power-up and starts the supply voltage supervisors. The brown-out reset always triggers a POR followed by a PUC.

4.2.7 Manual Control of the Power Management Module

PMM operation requires minimal software involvement. The core voltage and the supply voltage supervisor and monitor of DV_{CC} and V_{CORE} are selected by the user, while the hardware manages proper operation. If the application allows, the user can manually switch off or degrade functionality to save power.

4.2.7.1 Manual Control of the Voltage Regulator

The regulator current mode (full performance or low current) is selected by the hardware. The application software can also manually select the current mode by setting voltage regulator current mode bits (PMMCMD).

Table 4-3. Power Mode Overwrite (see also device specific datasheet)

PMMCMD		$I(V_{CORE})$	Description
[1]	[0]		
0	0 or 1	0 to 25 mA	Hardware controlled performance mode
1	0	≤30 µA	Manually selected low-current mode
1	1	≤25 mA	Manually selected full-performance mode

4.2.7.2 Controlling the $SVS_{H,L}$ and $SVM_{H,L}$ Performance

The supply voltage supervisors and supply voltage monitors are detecting supply voltage changes. If the application allows, the power consumption of the $SVM_{H,L}$ and $SVS_{H,L}$ can be reduced by lowering their reaction speed (low power mode). $SVM_{H,L}$ and $SVS_{H,L}$ can be disabled separately by clearing the respective enable bits. A predefined performance selection can be enabled by setting SVSHACE=1 (SVSLACE=1) for the supply voltage supervisor and the supply voltage monitor of the high (low) voltage side. If the SVSHACE (SVSLACE) bit is not set, the SVS_H and SVM_H (SVS_L and SVM_L) operation mode is controlled only by SVSHFP (SVSLFP) and can be disabled by clearing the enable bits SVSHE and SVMHE (SVSLE and SVMLE).

Table 4-4. $SVS_{H,L}$ and $SVM_{H,L}$ Performance When SVSHACE = SVSLACE = 0

Control Bit Setting		Active mode, LPM0, LPM1	LPM2, LPM3, LPM4	LPM5
SVSHFP, SVMHFP, SVSLFP, SVMLFP	0	Slow	Slow	Off
	1	Fast	Fast	Off

Table 4-5. $SVS_{H,L}$ and $SVM_{H,L}$ Performance When SVSHACE = SVSLACE = 1

Control Bit Setting		Active mode, LPM0, LPM1	LPM2, LPM3, LPM4	LPM5
SVSHFP, SVMHFP, SVSLFP, SVMLFP	0	Slow	Off	Off
	1	Fast	Slow	Off

4.2.7.3 Disabling the Core Voltage Regulator – LPM5

The voltage regulator is disabled by setting the PMMREGOFF bit to 1 and entering LPM4. The current consumption is reduced below ~100 nA (see device specific datasheet). Device wake-up is done through the $\overline{RST}$/NMI pin or any other wake-up capable enabled I/O-pin (see device specific datasheet).

```
; Code Sequence to enter LPM5.
   MOV   #PMMPW+REGOFF,&PMMCTL0     ; Set REGOFF
   BIS   #LPM4,SR                   ; Enter LPM4
```

The voltage regulator is turned off when LPM4 is entered while the REGOFF bit is set. An active clock request prevents turning off the voltage regulator. Once the clock request is deasserted the device turns off the voltage regulator and enters LPM5. If an interrupt request clears the REGOFF bit before the voltage regulator is turned off the device enters active mode immediately.

4.2.8 I/O-Port Control

As long as the system is not powered up completely or during a low-voltage condition, the digital input path of the digital I/O is disabled by locking the latest logical level. The data-in registers keep their values and the interrupts associated with digital inputs are not detected. Digital outputs stop driving and weak pullup/pulldown resistors are disabled.

4.2.9 PMM Interrupts

The PMM module generates reset signals and interrupt requests. The reset signals and the interrupt flags are routed to the system control module (SYS) and are together with other reset and interrupt sources making up the reset vector word and the system NMI vector word. For the priorities and the details of these vector words, see the System Control Module chapter.

4.3 PMM Registers

The PMM registers are listed in Table 4-6. The base address of the PMM module can be found in the devices specific datasheet. The address offset of each PMM register is given in Table 4-6. The password defined in the PMMCTL0 register controls access to all PMM, SVS, and SVM registers. Once the correct password is written the write access is enabled. The write access is disabled by writing a wrong password in byte mode to the PMMCTL0 upper byte. Word accesses to PMMCTL0 with a wrong password triggers a PUC. A write access to a register other than PMMCTL0 while write access is not enabled causes a PUC.

Table 4-6. PMM Registers

Register	Short Form	Register Type	Address	Initial State
PMM control register 0	PMMCTL0	Read/write	00h	0000h
PMM control register 1	PMMCTL1	Read/write	02h	0000h
SVS and SVM high side control register	SVSMHCTL	Read/write	04h	4400h
SVS and SVM low side control register	SVSMLCTL	Read/write	06h	4400h
SVSIN ans SVMOUT control register (optional)	SVSMIO	Read/write	08h	0020h
PMM interrupt flag register	PMMIFG	Read/write	0Ch	0000h
PMM interrupt enable register	PMMRIE	Read/write	0Eh	0000h

PMMCTL0, Power Management System Control Register 0

15	14	13	12	11	10	9	8
PMMKEY, Read as 96h, Must be written as A5h							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
PMMHPMRE	**Reserved**		**PMMREGOFF**	**PMMSWPOR**	**PMMSWBOR**	**PMMCOREV**	
rw-0	r-0	r-0	rw-0	rw-0	rw-0	rw-[0]	rw-[0]

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

PMMHPMRE Bit 7 Global High Power Module Request Enable. If the PMMHPMRE bit is set any module is able to request the PMM high power mode.

Reserved Bits 6-5 Reserved. Always read 0.

PMMREGOFF Bit 4 Regulator off. See chapter "Disabling the Core Voltage Regulator - LPM5"

PMMSWPOR Bit 3 Software POR. Setting this bit to 1 triggers a POR. This bit is self-clearing.

PMMSWBOR Bit 2 Software BOR. Setting this bit to 1 triggers a BOR. This bit is self-clearing.

PMMCOREV Bits 1-0 Core voltage. For details please refer to the devices specific datasheet.

00 V_{CORE} is typical at 1.4 V.

01 V_{CORE} is typical at 1.6 V.

10 V_{CORE} is typical at 1.8 V.

11 V_{CORE} is typical at 1.9 V.

PMMCTL1, Power Management System Control Register 1

15	14	13	12	11	10	9	8
Reserved							
r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0

7	6	5	4	3	2	1	0
Reserved		**PMMCMD**		**Reserved**		**PMMREFACC**	**PMMREFMD**
r-0	r-0	rw-[0]	rw-[0]	r-0	r-0	rw-0	rw-0

Reserved Bits 15-6 Reserved. Always read 0.

PMMCMD Bits 5-4 Voltage regulator current mode

00 The voltage regulator current range is defined by the low-power mode

01 The voltage regulator current range is defined by the low-power mode.

10 The voltage regulator is forced into low-current mode.

11 The voltage regulator is forced into full-performance mode.

Reserved Bits 3-2 Reserved. Always read 0.

PMMREFACC Bit 1 PMM reference accuracy. If PMMREFACC is set to 1 the power consumption of the voltage reference is reduced. The accuracy of the voltage reference decreases especially at higher temperatures.

PMMREFMD Bit 0 PMM reference mode. If the voltage regulator is in full performance mode the voltage reference operates in continuous (static) mode. If PMMREFMD is set and the voltage regulator is in full-performance mode the voltage reference current consumption is reduced. The accuracy of the voltage reference decreases.

SVSMHCTL, High Side Supply Voltage Supervisor and Monitor Control Register

15	14	13	12	11	10	9	8
SVMHFP	**SVMHE**	**Reserved**	**SVMHOVPE**	**SVSHFP**	**SVSHE**	**SVSHRVL**	
rw-[0]	rw-1	r-0	rw-[0]	rw-[0]	rw-1	rw-[0]	rw-[0]

7	6	5	4	3	2	1	0
SVSMHACE	**SVSMHEVM**	**Reserved**	**SVSHMD**	**SVSMHDLYST**	**SVSMHRRVL**		
rw-[0]	rw-0	r-0	rw-0	rw-0	rw-[0]	rw-[0]	rw-[0]

SVMHFP Bit 15 SVM high side full-performance mode. If this bit is set the SVM$_H$ operates in full-performance mode.

0 Normal mode. The propagation delay is typical 150us. See device specific datasheet.

1 Full performance mode. The propagation delay is typical 1us. See device specific datasheet.

SVMHE Bit 14 SVM high side enable. If this bit is set the SVM$_H$ is enabled.

Reserved Bit 13 Reserved. Always read 0.

SVMHOVPE Bit 12 SVM high side over-voltage enable. If this bit is set the SVM$_H$ overvoltage detection is enabled.

SVSHFP Bit 11 SVS high side full-performance mode. If this bit is set the SVS$_H$ operates in full-performance mode.

0 Normal mode. The propagation delay is typical 150us. See device specific datasheet.

1 Full performance mode. The propagation delay is typical 1us. See device specific datasheet.

SVSHE Bit 10 SVS high side enable. If this bit is set the SVS$_H$ is enabled.

SVSHRVL Bits 9-8 SVS high side reset voltage level. If DV$_{CC}$ falls short of the SVS$_H$ voltage level selected by SVSHRVL a reset is triggered (if SVS$_L$ is enabled). The voltage levels are defined in the device specific datasheet.

SVSMHACE Bit 7 SVS and SVM high side automatic control enable. If this bit is set the low-power mode of the SVS$_H$ and SVM$_H$ circuits is under hardware control.

SVSMHEVM Bit 6 SVS and SVM high side event mask. If this bit is set the SVS$_H$ and SVM$_H$ events are masked.

0 No events are masked

1 All events are masked.

Reserved Bit 5 Reserved. Always read 0.

SVSHMD Bit 4 SVS high side mode. If this bit is set the SVS$_H$ interrupt flag is set in LPM2, LPM3, and LPM4 in case of power fail conditions. If this bit is not set the SVS$_H$ interrupt is not set in LPM2, LPM3, and LPM4.

SVSMHDLYST Bit 3 SVS and SVM high side delay status. If this bit is set the SVS$_H$ and SVM$_H$ events are masked for some delay time. The delay time depends on the power-mode of the SVS$_H$ and SVM$_H$. If SVMHFP = 1 and SVSHFP = 1 it is ~2 µs in all other cases it is ~150 µs. See the device-specific data sheet for details. The bit is cleared by hardware if the delay has expired.

SVSMHRRVL Bits 2-0 SVS and SVM high side reset release voltage level. These bits define the reset release voltage level of the SVS$_H$. It is also used for the SVM$_H$ to define the voltage reached level. The voltage levels are defined in the device specific datasheet.

SVSMLCTL, Low Side Supply Voltage Supervisor and Monitor Control Register

15	14	13	12	11	10	9	8
SVMLFP	**SVMLE**	**Reserved**	**SVMLOVPE**	**SVSLFP**	**SVSLE**	**SVSLRVL**	
rw-[0]	rw-1	r-0	rw-[0]	rw-[0]	rw-1	rw-[0]	rw-[0]

7	6	5	4	3	2	1	0
SVSMLACE	**SVSMLEVM**	**Reserved**	**SVSLMD**	**SVSMLDLYST**	**SVSMLRRVL**		
rw-[0]	rw-0	r-0	rw-0	rw-0	rw-[0]	rw-[0]	rw-[0]

SVMLFP Bit 15 SVM low side full-performance mode. If this bit is set the SVM_L operates in full-performance mode.

0 Normal mode. The propagation delay is typical 150us. See device specific datasheet.

1 Full performance mode. The propagation delay is typical 1us. See device specific datasheet.

SVMLE Bit 14 SVM low side enable. If this bit is set the SVM_L is enabled.

Reserved Bit 13 Reserved. Always read 0.

SVMLOVPE Bit 12 SVM low side over-voltage enable. If this bit is set the SVM_L overvoltage detection is enabled.

SVSLFP Bit 11 SVS low side full-performance mode. If this bit is set the SVS_L operates in full-performance mode.

0 Normal mode. The propagation delay is typical 150us. See device specific datasheet.

1 Full performance mode. The propagation delay is typical 1us. See device specific datasheet.

SVSLE Bit 10 SVS low side enable. If this bit is set the SVS_L is enabled.

SVSLRVL Bits 9-8 SVS low side reset voltage level. If DV_{CC} falls short of the SVS_L voltage level selected by SVSHRVL a reset is triggered (if SVS_L is enabled). The voltage levels are defined in the device specific datasheet.

SVSMLACE Bit 7 SVS and SVM low side automatic control enable. If this bit is set the low-power mode of the SVS_L and SVM_L circuits is under hardware control.

SVSMLEVM Bit 6 SVS and SVM low side event mask. If this bit is set the SVS_L and SVM_L events are masked.

0 No events are masked.

1 All events are masked.

Reserved Bit 5 Reserved. Always read 0.

SVSLMD Bit 4 SVS low side mode. If this bit is set the SVS_L interrupt flag is set in LPM2, LPM3 and LPM4 in case of a power fail conditions. If this bit is not set the SVS_L interrupt is not set in LPM2, LPM3, and LPM4.

SVSMLDLYST Bit 3 SVS and SVM low side delay status. If this bit is set the SVS_L and SVM_L events are masked for some delay time. The delay time depends on the power-mode of the SVS_L and SVM_L. If SVMLFP = 1 and SVSLFP = 1 it is ~2 µs in all other cases it is ~150 µs. The bit is cleared by hardware if the delay has expired.

SVSMLRRVL Bits 2-0 SVS and SVM low side reset release voltage level. These bits define the reset release voltage level of the SVS_L. It is also used for the SVM_L to define the voltage reached level. The voltage levels are defined in the device specific datasheet.

SVSMIO, SVSIN, and SVMOUT Control Register

15	14	13	12	11	10	9	8
Reserved			**SVMLVLROE**	**SVMHOE**	**Reserved**		
r-0	r-0	r-0	rw-[0]	rw-[0]	r-0	r-0	r-0

7	6	5	4	3	2	1	0
Reserved		**SVMOUTPOL**	**SVMLVLROE**	**SVMLOE**	**Reserved**		
r-0	r-0	rw-[1]	rw-[0]	rw-[0]	r-0	r-0	r-0

Reserved	Bits 15-13	Reserved. Always read 0.
SVMLVLROE	Bit 12	SVM high side voltage level reached output enable. If this bit is set the SVMLVLRIFG bit is output to the device SVMOUT pin. The device specific port logic has to be configured accordingly.
SVMHOE	Bit 11	SVM high side output enable. If this bit is set the SVMHIFG bit is output to the device SVMOUT pin. The device specific port logic has to be configured accordingly.
Reserved	Bits 10-6	Reserved. Always read 0.
SVMOUTPOL	Bit 5	SVMOUT pin polarity. If this bit is set SVMOUT is active high. An error condition is signaled by a 1 at SVMOUT. If SVMOUTPOL is cleared the error condition is signaled by a 0 at the SVMOUT pin.
SVMLVLROE	Bit 4	SVM low side voltage level reached output enable. If this bit is set the SVMLVLRIFG bit is output to the device SVMOUT pin. The device specific port logic has to be configured accordingly.
SVMLOE	Bit 3	SVM low side output enable. If this bit is set the SVMLIFG bit is output to the device SVMOUT pin. The device specific port logic has to be configured accordingly.
Reserved	Bits 2-0	Reserved. Always read 0.

PMMIFG, Power Management System and Supply Voltage Supervisor and Monitor Interrupt Flag Register

15	14	13	12	11	10	9	8
PMMRSTLPM5IF1G	**Reserved**	**SVSLIFG**[1]	**SVSHIFG**[1]	**Reserved**	**PMMPORIFG**	**PMMRSTIFG**	**PMMBORIFG**
rw-[0]	r-0	rw-[0]	rw-[0]	r-0	rw-[0]	rw-[0]	rw-[0]

7	6	5	4	3	2	1	0
Reserved	**SVMHVLRIFG**[1]	**SVMHIFG**	**SVSMHDLYIFG**	**Reserved**	**SVMLVLRIFG**[1]	**SVMLIFG**	**SVSMLDLYIFG**
r-0	rw-[0]	rw-[0]	rw-0	r-0	rw-[0]	rw-[0]	rw-0

[1] After power up the reset value depends
on the power sequence.

PMMRSTLPM5IFG Bit 15 LPM5 Flag. This bit is set if the system was in LPM5 before. The bit is cleared by software or by reading the reset vector word. A power-failure on the DV_{CC} domain clears the bit.

0 No interrupt pending

1 Interrupt pending

Reserved Bit 14 Reserved. Always read 0.

SVSLIFG Bit 13 SVS low side interrupt flag. The bit is cleared by software or by reading the reset vector word.

0 No interrupt pending

1 Interrupt pending

SVSHIFG Bit 12 SVS high side interrupt flag. The bit is cleared by software or by reading the reset vector word.

0 No interrupt pending

1 Interrupt pending

Reserved Bit 11 Reserved. Always read 0.

PMMPORIFG Bit 10 PMM software POR interrupt flag. This interrupt flag is set if a software POR is triggered. The bit is cleared by software or by reading the reset vector word.

0 No interrupt pending

1 Interrupt pending

PMMRSTIFG Bit 9 PMM RST pin interrupt flag. This interrupt flag is set if the $\overline{\text{RST}}$/NMI pin is the reset source. The bit is cleared by software or by reading the reset vector word.

0 No interrupt pending

1 Interrupt pending

PMMBORIFG Bit 8 PMM software BOR interrupt flag. This interrupt flag is set if a software BOR (PMMSWBOR) is triggered. The bit is cleared by software or by reading the reset vector word.

0 No interrupt pending

1 Interrupt pending

Reserved Bit 7 Reserved. Always read 0.

SVMHVLRIFG Bit 6 SVM high side voltage level reached interrupt flag. The bit is cleared by software or by reading the reset vector (SVSHPE = 1) word or by reading the interrupt vector (SVSHPE = 0) word.

0 No interrupt pending

1 Interrupt pending

SVMHIFG Bit 5 SVM high side interrupt flag. The bit is cleared by software.

0 No interrupt pending

1 Interrupt pending

SVSMHDLYIFG Bit 4 SVS and SVM high side delay expired interrupt flag. This interrupt flag is set if the delay element expired. The bit is cleared by software or by reading the interrupt vector word.

0 No interrupt pending

1 Interrupt pending

Reserved Bit 3 Reserved. Always read 0.

SVMLVLRIFG Bit 2 SVM low side voltage level reached interrupt flag. The bit is cleared by software or by reading the reset vector (SVSLPE = 1) word or by reading the interrupt vector (SVSLPE = 0) word.

0 No interrupt pending

1 Interrupt pending

SVMLIFG Bit 1 SVM low side interrupt flag. The bit is cleared by software.

0 No interrupt pending

1 Interrupt pending

SVSMLDLYIFG Bit 0 SVS and SVM low side delay expired interrupt flag. This interrupt flag is set if the delay element expired. The bit is cleared by software or by reading the interrupt vector word.

0 No interrupt pending

1 Interrupt pending

PMMRIE, Power Management System Reset Enable and Interrupt Enable Register

15	14	13	12	11	10	9	8
Reserved		**SVMHVLRPE**	**SVSHPE**	**Reserved**		**SVMLVLRPE**	**SVSLPE**
r-0	r-0	rw-[0]	rw-[0]	r-0	r-0	rw-[0]	rw-[0]

7	6	5	4	3	2	1	0
Reserved	**SVMHVLRIE**	**SVMHIE**	**SVSMHDLYIE**	**Reserved**	**SVMLVLRIE**	**SVMLIE**	**SVSMLDLYIE**
r-0	rw-0	rw-0	rw-0	r-0	rw-0	rw-0	rw-0

Reserved Bits 15-14 Reserved. Always read 0.

SVMHVLRPE Bit 13 SVM high side voltage level reached POR enable. If this bit is set, exceeding the SVM_H voltage level triggers a POR.

SVSHPE Bit 12 SVS high side POR enable. If this bit is set, falling below the SVS_H voltage level triggers a POR.

Reserved Bits 11-10 Reserved. Always read 0.

SVMLVLRPE Bit 9 SVM low side voltage level reached por enable. If this bit is set, exceeding the SVM_L voltage level triggers a POR.

SVSLPE Bit 8 SVS low side POR enable. If this bit is set, falling below the SVS_L voltage level triggers a POR.

Reserved Bit 7 Reserved. Always read 0.

SVMHVLRIE Bit 6 SVM high side reset voltage level interrupt enable

SVMHIE Bit 5 SVM high side interrupt enable. This bit is cleared by software or if the interrupt vector word is read.

SVSMHDLYIE Bit 4 SVS and SVM high side delay expired interrupt enable

Reserved Bit 3 Reserved. Always read 0.

SVMLVLRIE Bit 2 SVM low side reset voltage level interrupt enable

SVMLIE Bit 1 SVM low side interrupt enable. This bit is cleared by software or if the interrupt vector word is read.

SVSMLDLYIE Bit 0 SVS and SVM low side delay expired interrupt enable

Chapter 5

SLAU208–June 2008

CPUX

This chapter describes the extended MSP430X 16-bit RISC CPU with 1-MB memory access, its addressing modes, and instruction set. The MSP430X CPU is implemented in the MSP430F5xx devices.

Note: The MSP430X CPU implemented on MSP430F5xx devices has, in some cases, slightly different cycle counts from the MSP430X CPU implemented on the 2xx and 4xx families.

Topic **Page**

5.1 CPU Introduction

The MSP430X 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 MSP430X CPU can address a 1-MB address range without paging. The MSP430X CPU is completely backwards compatible with the MSP430 CPU.

The MSP430X CPU features include:

- RISC architecture
- Orthogonal architecture
- Full register access including program counter, status register and stack pointer
- Single-cycle register operations
- Large register file reduces fetches to memory.
- 20-bit address bus allows direct access and branching throughout the entire memory range without paging.
- 16-bit data bus allows direct manipulation of word-wide arguments.
- Constant generator provides the six most often used immediate values and reduces code size.
- Direct memory-to-memory transfers without intermediate register holding.
- Byte, word, and 20-bit address-word addressing

The block diagram of the MSP430X CPU is shown in Figure 5-1.

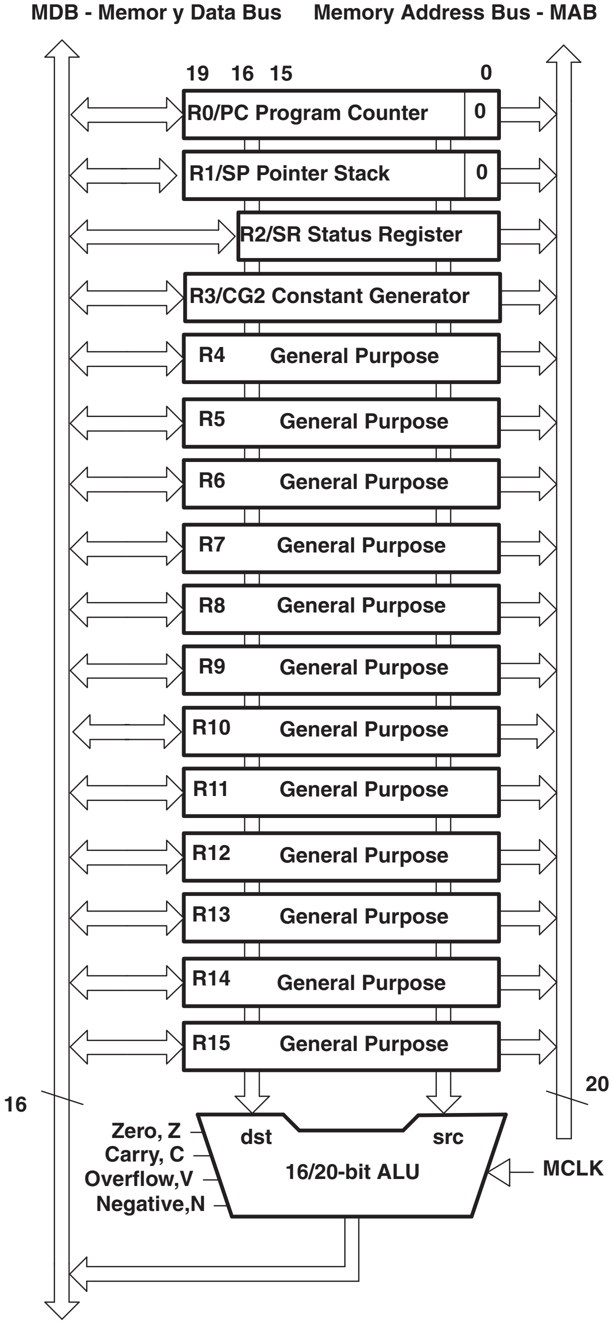


Figure 5-1. MSP430X CPU Block Diagram

5.2 Interrupts

The MSP430X has the following interrupt structure:

- Vectored interrupts with no polling necessary
- Interrupt vectors are located downward from address 0FFFEh.

The interrupt vectors contain 16-bit addresses that point into the lower 64-KB memory. This means all interrupt handlers must start in the lower 64-KB memory.

During an interrupt, the program counter and the status register are pushed onto the stack as shown in Figure 5-2. The MSP430X architecture stores the complete 20-bit PC value efficiently by appending the PC bits 19:16 to the stored SR value automatically on the stack. When the RETI instruction is executed, the full 20-bit PC is restored making return from interrupt to any address in the memory range possible.

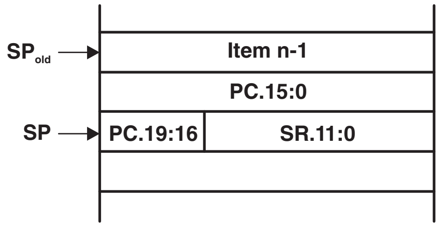


Figure 5-2. Program Counter Storage on the Stack for Interrupts

5.3 CPU Registers

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

5.3.1 *Program Counter (PC)*

The 20-bit program counter (PC/R0) points to the next instruction to be executed. Each instruction uses an even number of bytes (two, four, six, or eight bytes), and the PC is incremented accordingly. Instruction accesses are performed on word boundaries, and the PC is aligned to even addresses. Figure 5-3 shows the program counter.

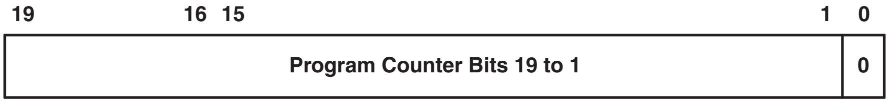


Figure 5-3. Program Counter

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

```
MOV.W  #LABEL,PC  ; Branch to address LABEL (lower 64 KB)

MOVA   #LABEL,PC  ; Branch to address LABEL (1MB memory)

MOV.W  LABEL,PC   ; Branch to address in word LABEL
                  ; (lower 64 KB)

MOV.W  @R14,PC    ; Branch indirect to address in
                  ; R14 (lower 64 KB)

ADDA   #4,PC      ; Skip two words (1 MB memory)
```

The BR and CALL instructions reset the upper four PC bits to 0. Only addresses in the lower 64-KB address range can be reached with the BR or CALL instruction. When branching or calling, addresses beyond the lower 64-KB range can only be reached using the BRA or CALLA instructions. Also, any instruction to directly modify the PC does so according to the used addressing mode. For example, `MOV.W #value,PC` will clear the upper four bits of the PC because it is a .W instruction.

The program counter is automatically stored on the stack with CALL, or CALLA instructions, and during an interrupt service routine. Figure 5-4 shows the storage of the program counter with the return address after a CALLA instruction. A CALL instruction stores only bits 15:0 of the PC.

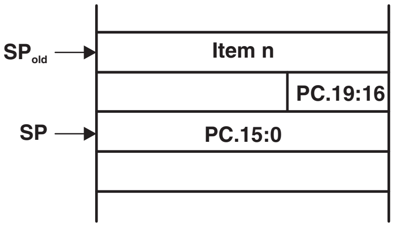


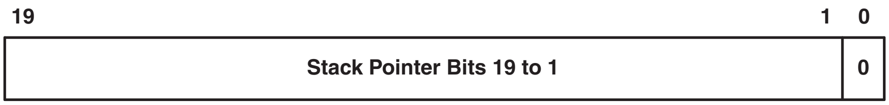
Figure 5-4. Program Counter Storage on the Stack for CALLA

The RETA instruction restores bits 19:0 of the program counter and adds 4 to the stack pointer. The RET instruction restores bits 15:0 to the program counter and adds 2 to the stack pointer.

5.3.2 Stack Pointer (SP)

The 20-bit 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 5-5 shows the SP. The SP is initialized into RAM by the user, and is always aligned to even addresses.

Figure 5-6 shows the stack usage. Figure 5-7 shows the stack usage when 20-bit address-words are pushed.



```
MOV.W    2(SP),R6       ; Copy Item I2 to R6
MOV.W    R7,0(SP)       ; Overwrite TOS with R7
PUSH     #0123h         ; Put 0123h on stack
POP      R8             ; R8 = 0123h
```

Figure 5-5. Stack Pointer

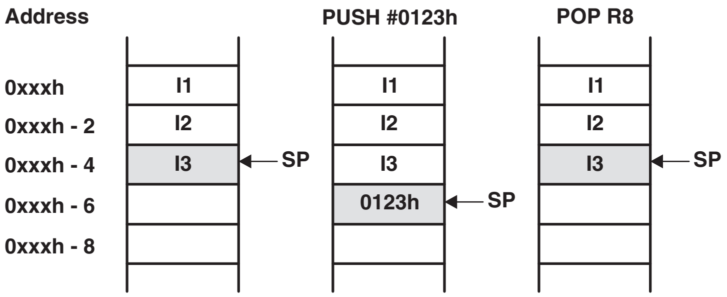


Figure 5-6. Stack Usage

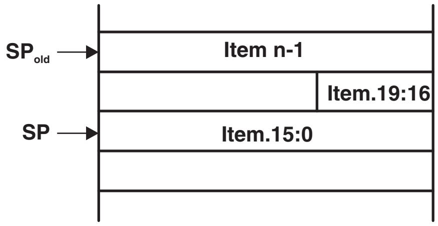
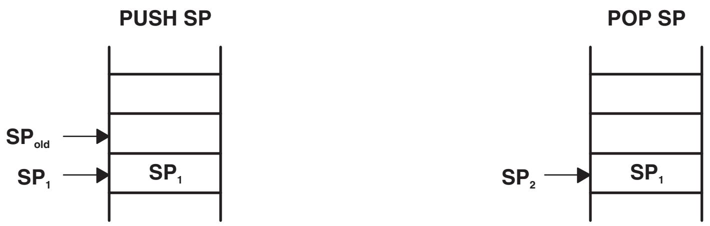


Figure 5-7. PUSHX.A Format on the Stack

The special cases of using the SP as an argument to the PUSH and POP instructions are described and shown in Figure 5-8.



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)

Figure 5-8. PUSH SP, POP SP Sequence

5.3.3 Status Register (SR)

The 16-bit status register (SR/R2), used as a source or destination register, can only be used in register mode addressed with word instructions. The remaining combinations of addressing modes are used to support the constant generator. Figure 5-9 shows the SR bits. Do not write 20-bit values to the SR. Unpredictable operation can result.

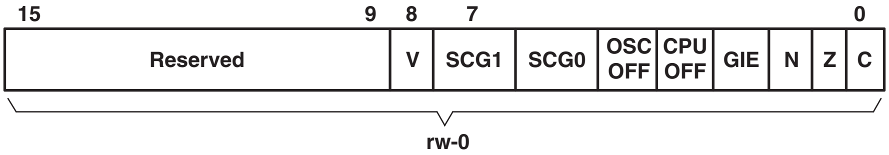


Figure 5-9. Status Register Bits

Table 5-1 describes the status register bits.

Table 5-1. Description of Status Register Bits

Bit	Description	
Reserved	Reserved	
V	Overflow bit. This bit is set when the result of an arithmetic operation overflows the signed-variable range.	

	`ADD(.B), ADDX(.B,.A),`
	`ADDC(.B), ADDCX(.B.A), ADDA`
		Set when:
		positive + positive = negative
		negative + negative = positive
		otherwise reset

	`SUB(.B), SUBX(.B,.A),`
	`SUBC(.B),SUBCX(.B,.A), SUBA,`
	`CMP(.B), CMPX(.B,.A), CMPA`
		Set when:
		positive – negative = negative
		negative – positive = positive
		otherwise reset

SCG1	System clock generator 1. This bit, when set, turns off the DCO dc generator, if DCOCLK is not used for MCLK or SMCLK.	
SCG0	System clock generator 0. This bit, when set, turns off the FLL+ loop control.	
OSCOFF	Oscillator off. This bit, when set, turns off the LFXT1 crystal oscillator, when LFXT1CLK is not used 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 an operation is negative and cleared when the result is positive.	
Z	Zero bit. This bit is set when the result of an operation is zero and cleared when the result is not zero.	
C	Carry bit. This bit is set when the result of an operation produced a carry and cleared when no carry occurred.	

5.3.4 Constant Generator Registers (CG1 and CG2)

Six commonly used constants are generated with the constant generator registers R2 (CG1) and R3 (CG2), 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 5-2.

Table 5-2. 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	FFh, FFFFh, FFFFFh	–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
```

5.3.5 General Purpose Registers R4 to R15

The twelve CPU registers R4 to R15, contain 8-bit, 16-bit, or 20-bit values. Any byte-write to a CPU register clears bits 19:8. Any word-write to a register clears bits 19:16. The only exception is the SXT instruction. The SXT instruction extends the sign through the complete 20-bit register.

The following figures show the handling of byte, word and address-word data. Note the reset of the leading MSBs, if a register is the destination of a byte or word instruction.

Figure 5-10 shows byte handling (8-bit data, .B suffix). The handling is shown for a source register and a destination memory byte and for a source memory byte and a destination register.

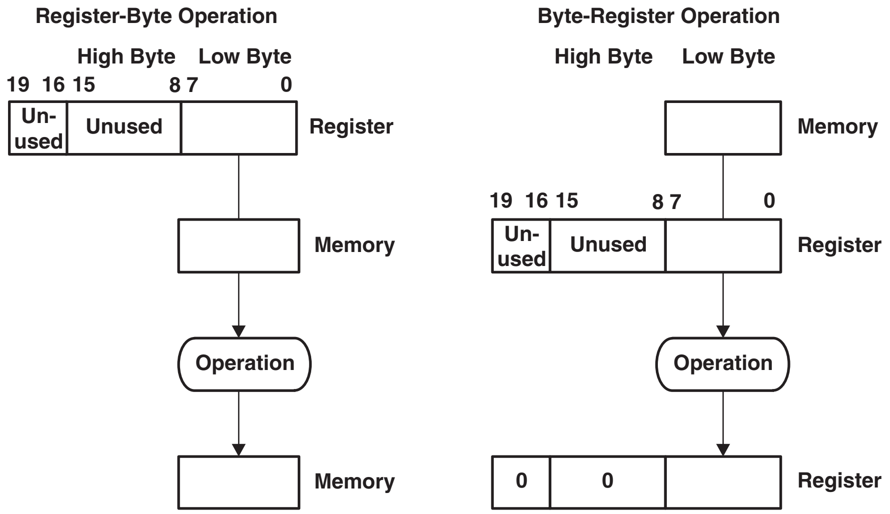


Figure 5-10. Register-Byte/Byte-Register Operation

Figure 5-11 and Figure 5-12 show 16-bit word handling (.W suffix). The handling is shown for a source register and a destination memory word and for a source memory word and a destination register.

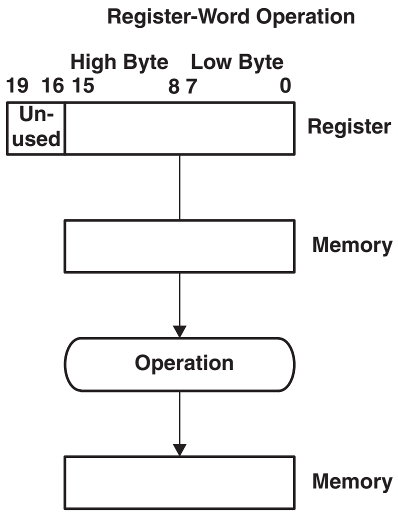


Figure 5-11. Register-Word Operation

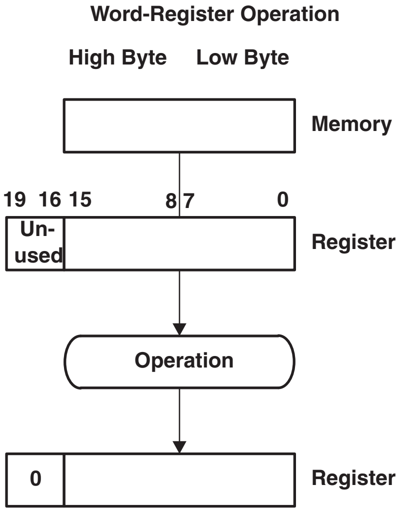


Figure 5-12. Word-Register Operation

Figure 5-13 and Figure 5-14 show 20-bit address-word handling (.A suffix). The handling is shown for a source register and a destination memory address-word and for a source memory address-word and a destination register.

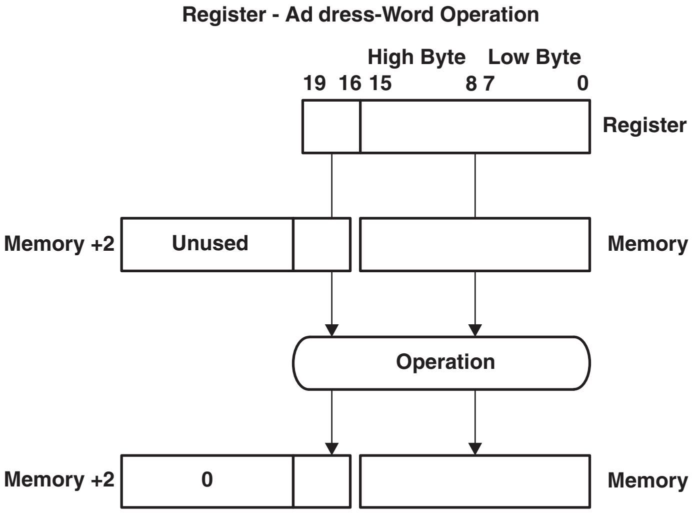


Figure 5-13. Register – Address-Word Operation

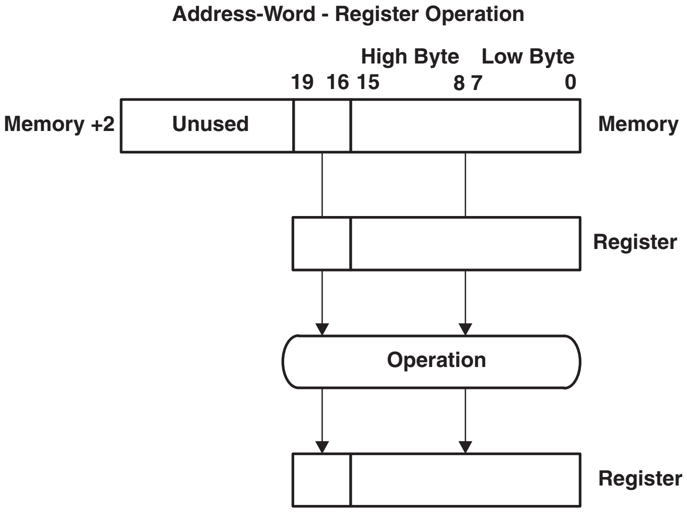


Figure 5-14. Address-Word – Register Operation

5.4 Addressing Modes

Seven addressing modes for the source operand and four addressing modes for the destination operand use 16-bit or 20-bit addresses (see Table 5-3). The MSP430 and MSP430X instructions are usable throughout the entire 1-MB memory range.

Table 5-3. Source/Destination Addressing

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, or stored in combination of the preceding extension word and the next word.
01/1	Symbolic mode	ADDR	(PC + X) points to the operand. X is stored in the next word, or stored in combination of the preceding extension word and 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, or stored in combination of the preceding extension word and 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. by 2 for .W instructions, and by 4 for .A instructions.
11/–	Immediate mode	#N	N is stored in the next word, or stored in combination of the preceding extension word and the next word. 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.

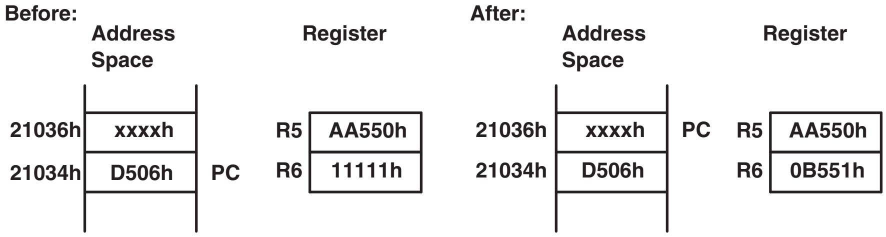
5.4.1 Register Mode

Operation:	The operand is the 8-, 16-, or 20-bit content of the used CPU register.
Length:	One, two, or three words
Comment:	Valid for source and destination
Byte operation:	Byte operation reads only the 8 LSBs of the source register Rsrc and writes the result to the 8 LSBs of the destination register Rdst. The bits Rdst.19:8 are cleared. The register Rsrc is not modified.
Word operation:	Word operation reads the 16 LSBs of the source register Rsrc and writes the result to the 16 LSBs of the destination register Rdst. The bits Rdst.19:16 are cleared. The register Rsrc is not modified.
Address-word operation:	Address-word operation reads the 20 bits of the source register Rsrc and writes the result to the 20 bits of the destination register Rdst. The register Rsrc is not modified
SXT exception:	The SXT instruction is the only exception for register operation. The sign of the low byte in bit 7 is extended to the bits Rdst.19:8.

Example:

```
BIS.W R5,R6 ;
```

This instruction logically ORs the 16-bit data contained in R5 with the 16-bit contents of R6. R6.19:16 is cleared.

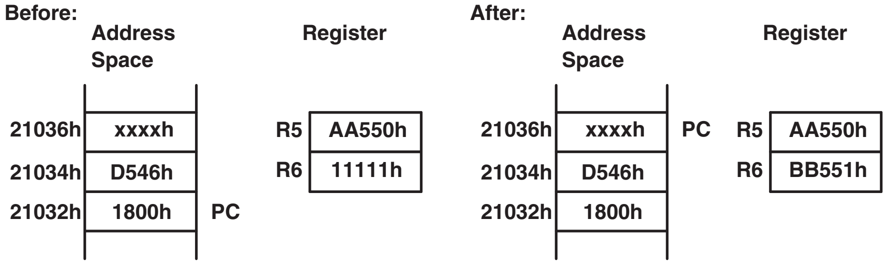


A550h.or.1111h = B551h

Example:

```
BISX.A R5,R6 ;
```

This instruction logically ORs the 20-bit data contained in R5 with the 20-bit contents of R6.
The extension word contains the A/L-bit for 20-bit data. The instruction word uses byte mode with bits A/L:B/W = 01. The result of the instruction is:



AA550h.or.11111h = BB551h

5.4.2 Indexed Mode

The Indexed mode calculates the address of the operand by adding the signed index to a CPU register. The Indexed mode has three addressing possibilities:

- Indexed mode in lower 64-KB memory
- MSP430 instruction with Indexed mode addressing memory above the lower 64-KB memory
- MSP430X instruction with Indexed mode

Indexed Mode in Lower 64-KB Memory

If the CPU register Rn points to an address in the lower 64 KB of the memory range, the calculated memory address bits 19:16 are cleared after the addition of the CPU register Rn and the signed 16-bit index. This means, the calculated memory address is always located in the lower 64 KB and does not overflow or underflow out of the lower 64-KB memory space. The RAM and the peripheral registers can be accessed this way and existing MSP430 software is usable without modifications as shown in Figure 5-15.

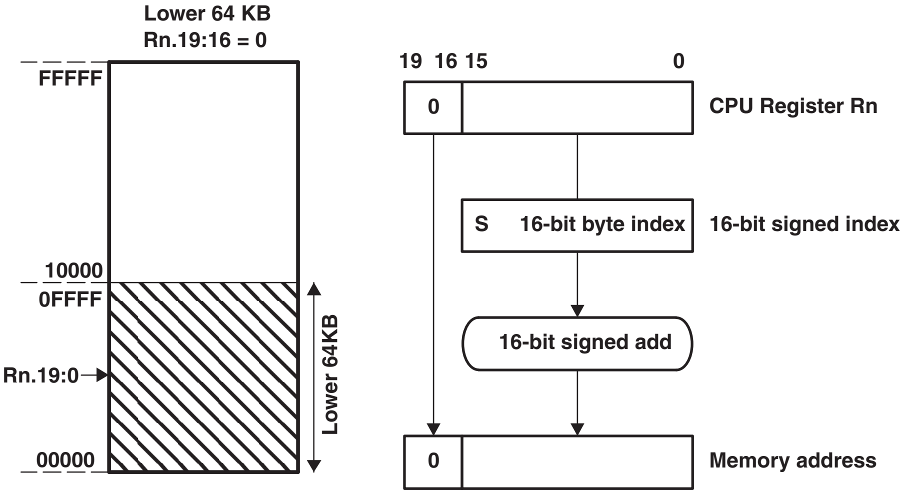


Figure 5-15. Indexed Mode in Lower 64 KB

Length: Two or three words

Operation: The signed 16-bit index is located in the next word after the instruction and is added to the CPU register Rn. The resulting bits 19:16 are cleared giving a truncated 16-bit memory address, which points to an operand address in the range 00000h to 0FFFFh. The operand is the content of the addressed memory location.

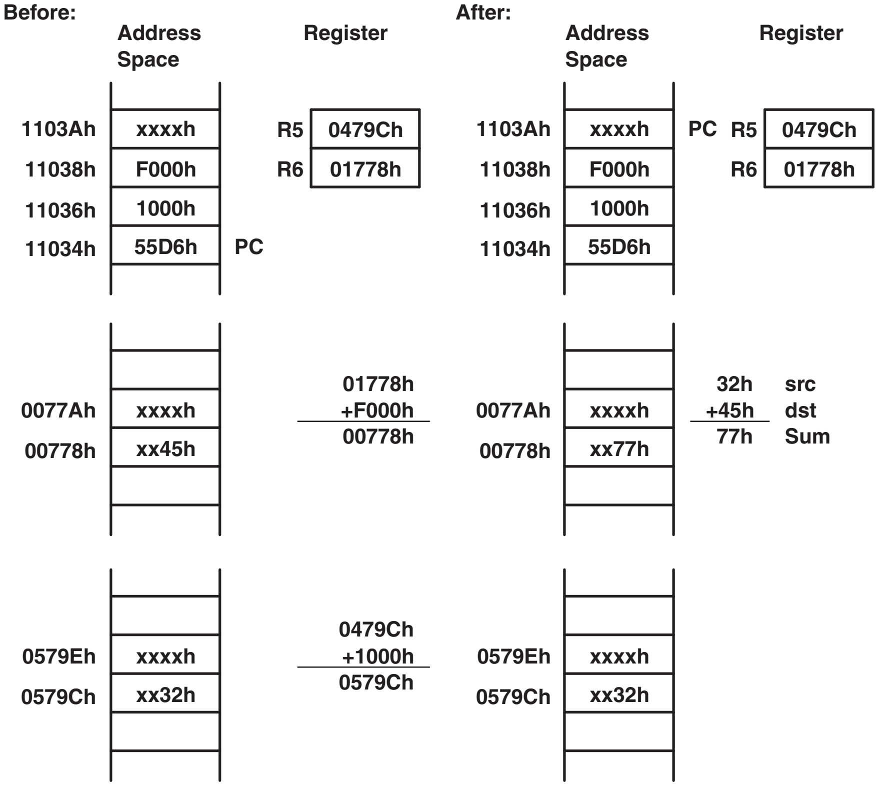
Comment: Valid for source and destination. The assembler calculates the register index and inserts it.

Example: `ADD.B 1000h(R5),0F000h(R6);`

This instruction adds the 8-bit data contained in source byte 1000h(R5) and the destination byte 0F000h(R6) and places the result into the destination byte. Source and destination bytes are both located in the lower 64 KB due to the cleared bits 19:16 of registers R5 and R6.

Source: The byte pointed to by R5 + 1000h results in address 0479Ch + 1000h = 0579Ch after truncation to a 16-bit address.

Destination: The byte pointed to by R6 + F000h results in address 01778h + F000h = 00778h after truncation to a 16-bit address.



MSP430 Instruction With Indexed Mode in Upper Memory

If the CPU register Rn points to an address above the lower 64-KB memory, the Rn bits 19:16 are used for the address calculation of the operand. The operand may be located in memory in the range Rn +32 KB, because the index, X, is a signed 16-bit value. In this case, the address of the operand can overflow or underflow into the lower 64-KB memory space (see Figure 5-16 and Figure 5-17).

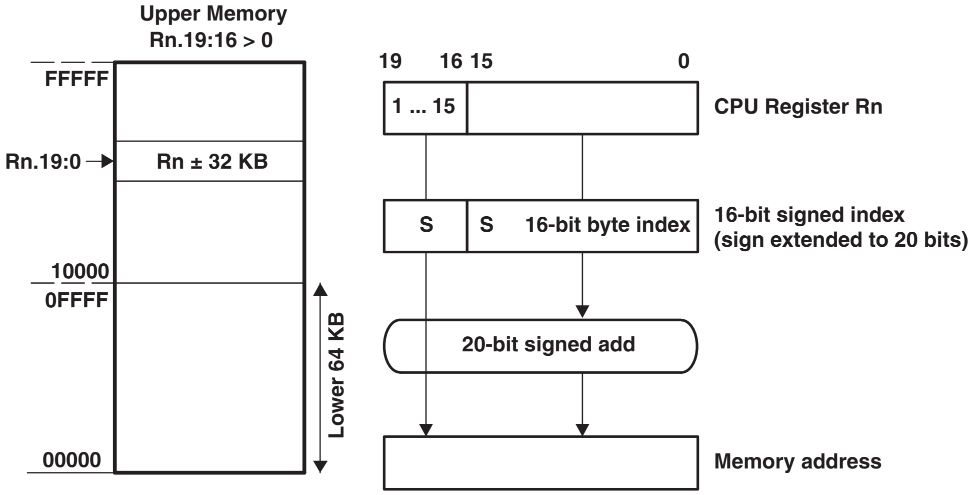


Figure 5-16. Indexed Mode in Upper Memory

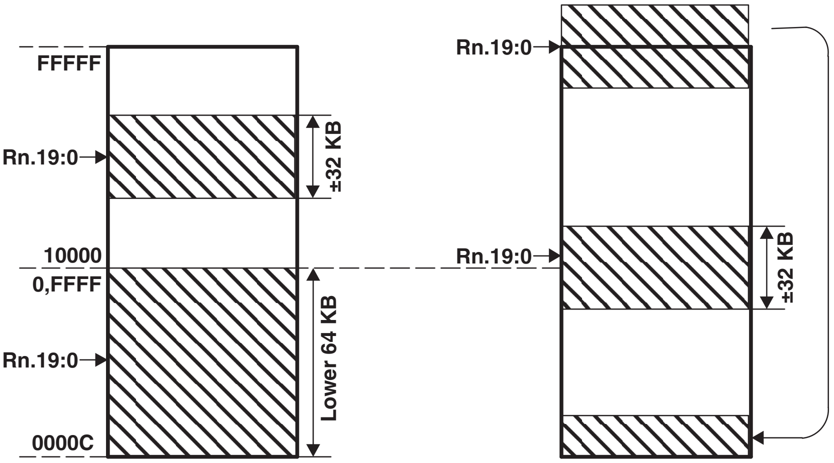


Figure 5-17. Overflow and Underflow for the Indexed Mode

Length: Two or three words

Operation: The sign-extended 16-bit index in the next word after the instruction is added to the 20 bits of the CPU register Rn. This delivers a 20-bit address, which points to an address in the range 0 to FFFFFh. The operand is the content of the addressed memory location.

Comment: Valid for source and destination. The assembler calculates the register index and inserts it.

Example: `ADD.W 8346h(R5),2100h(R6) ;`

This instruction adds the 16-bit data contained in the source and the destination addresses and places the 16-bit result into the destination. Source and destination operand can be located in the entire address range.

Source: The word pointed to by R5 + 8346h. The negative index 8346h is sign-extended, which results in address 23456h + F8346h = 1B79Ch.

Destination: The word pointed to by R6 + 2100h results in address 15678h + 2100h = 17778h.

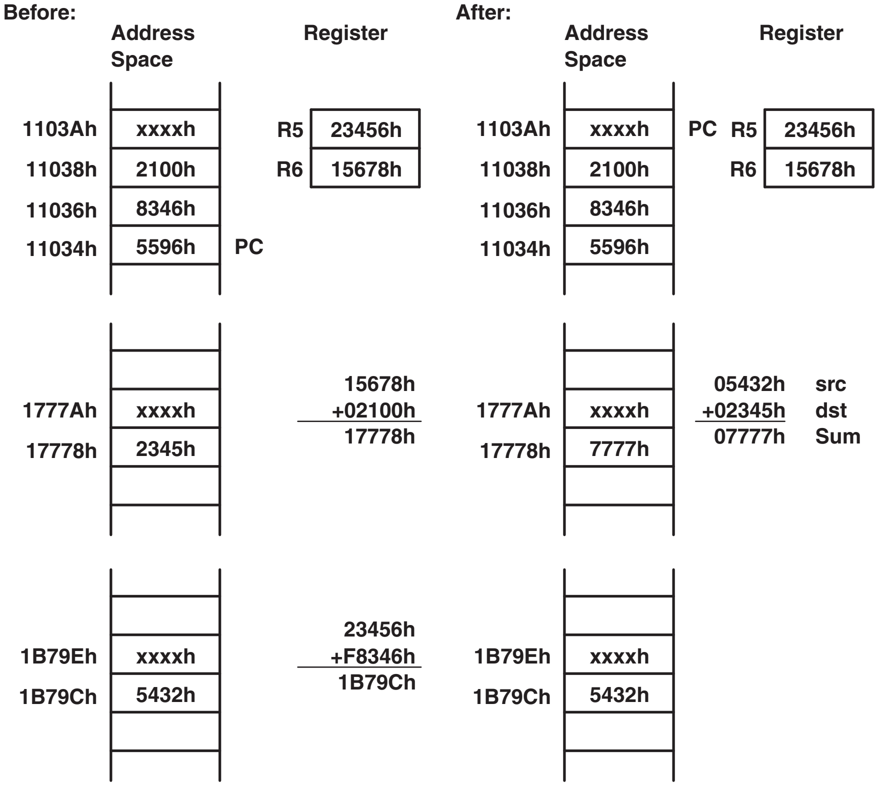


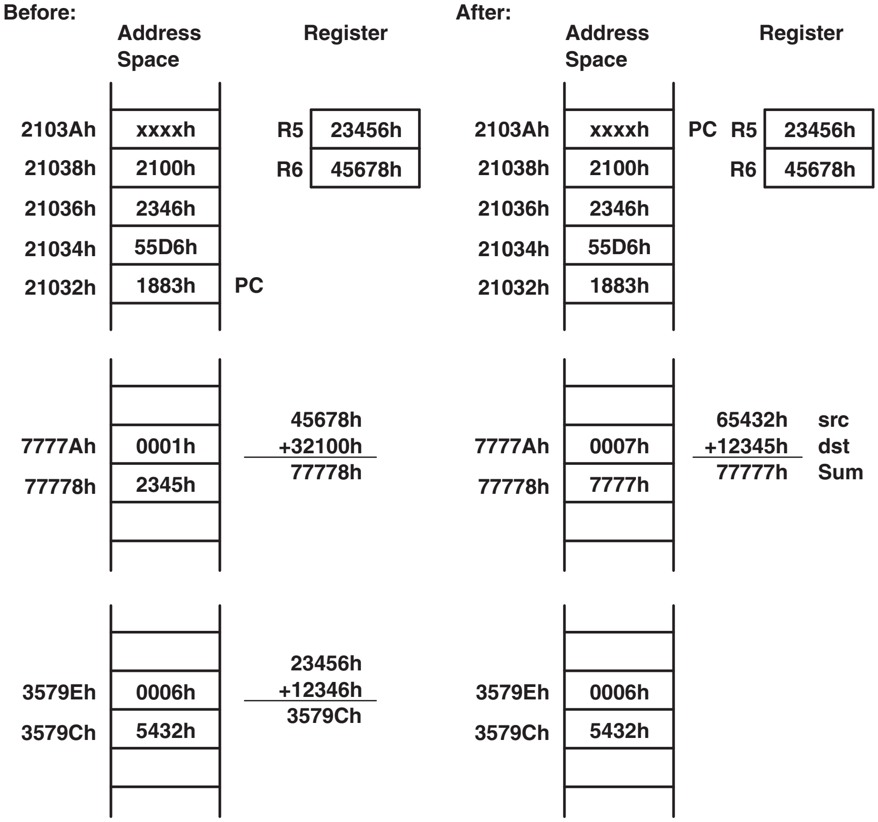
Figure 5-18. Example for the Indexed Mode

MSP430X Instruction With Indexed Mode

When using an MSP430X instruction with Indexed mode, the operand can be located anywhere in the range of Rn + 19 bits.

Length:	Three or four words
Operation:	The operand address is the sum of the 20-bit CPU register content and the 20-bit index. The four MSBs of the index are contained in the extension word, the 16 LSBs are contained in the word following the instruction. The CPU register is not modified
Comment:	Valid for source and destination. The assembler calculates the register index and inserts it.
Example:	`ADDX.A 12346h(R5),32100h(R6) ;` This instruction adds the 20-bit data contained in the source and the destination addresses and places the result into the destination.
Source:	Two words pointed to by R5 + 12346h which results in address 23456h + 12346h = 3579Ch.
Destination:	Two words pointed to by R6 + 32100h which results in address 45678h + 32100h = 77778h.

The extension word contains the MSBs of the source index and of the destination index and the A/L-bit for 20-bit data. The instruction word uses byte mode due to the 20-bit data length with bits A/L:B/W = 01.



5.4.3 Symbolic Mode

The Symbolic mode calculates the address of the operand by adding the signed index to the program counter. The Symbolic mode has three addressing possibilities:

- Symbolic mode in lower 64-KB memory
- MSP430 instruction with symbolic mode addressing memory above the lower 64-KB memory.
- MSP430X instruction with symbolic mode

Symbolic Mode in Lower 64 KB

If the PC points to an address in the lower 64 KB of the memory range, the calculated memory address bits 19:16 are cleared after the addition of the PC and the signed 16-bit index. This means, the calculated memory address is always located in the lower 64 KB and does not overflow or underflow out of the lower 64-KB memory space. The RAM and the peripheral registers can be accessed this way and existing MSP430 software is usable without modifications as shown in Figure 5-19.

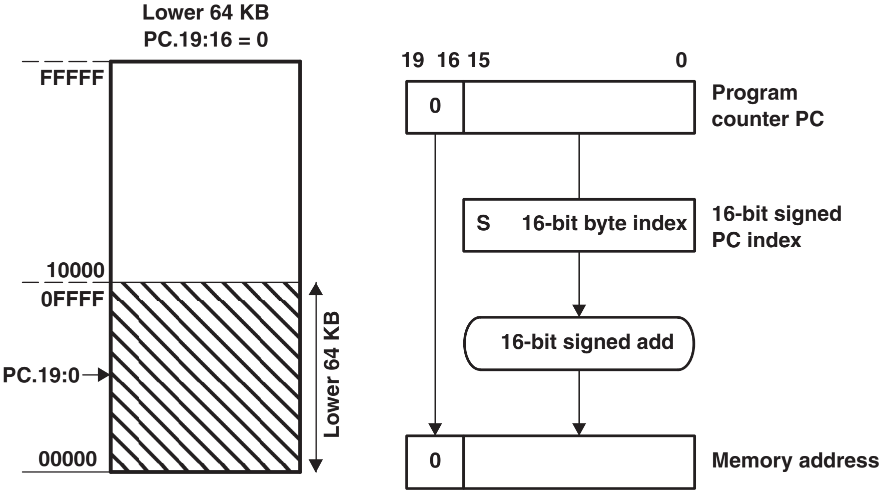


Figure 5-19. Symbolic Mode Running in Lower 64 KB

Operation: The signed 16-bit index in the next word after the instruction is added temporarily to the PC. The resulting bits 19:16 are cleared giving a truncated 16-bit memory address, which points to an operand address in the range 00000h, to 0FFFFh. The operand is the content of the addressed memory location.

Length: Two or three words

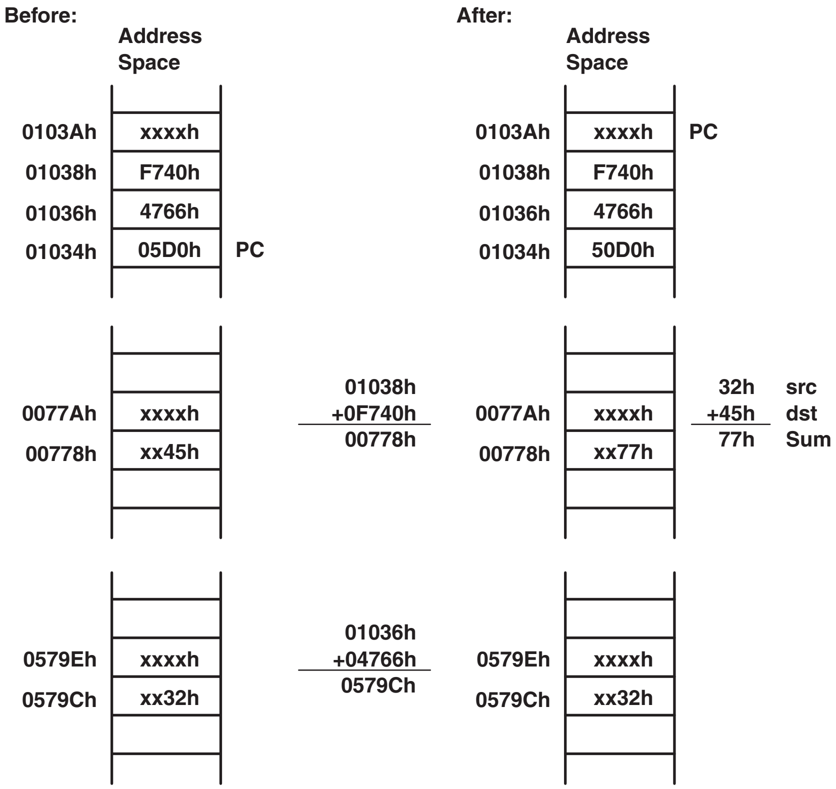
Comment: Valid for source and destination. The assembler calculates the PC index and inserts it.

Example: `ADD.B EDE,TONI ;`

This instruction adds the 8-bit data contained in source byte EDE and destination byte TONI and places the result into the destination byte TONI. Bytes EDE and TONI and the program are located in the lower 64 KB.

Source: Byte EDE located at address 0,579Ch, pointed to by PC + 4766h where the PC index 4766h is the result of 0579Ch - 01036h = 04766h. Address 01036h is the location of the index for this example.

Destination: Byte TONI located at address 00778h, pointed to by PC + F740h, is the truncated 16-bit result of 00778h – 1038h = FF740h. Address 01038h is the location of the index for this example.



MSP430 Instruction with Symbolic Mode in Upper Memory

If the PC points to an address above the lower 64-KB memory, the PC bits 19:16 are used for the address calculation of the operand. The operand may be located in memory in the range PC +32 KB, because the index, X, is a signed 16-bit value. In this case, the address of the operand can overflow or underflow into the lower 64-KB memory space as shown in Figure 5-20 and Figure 5-21.

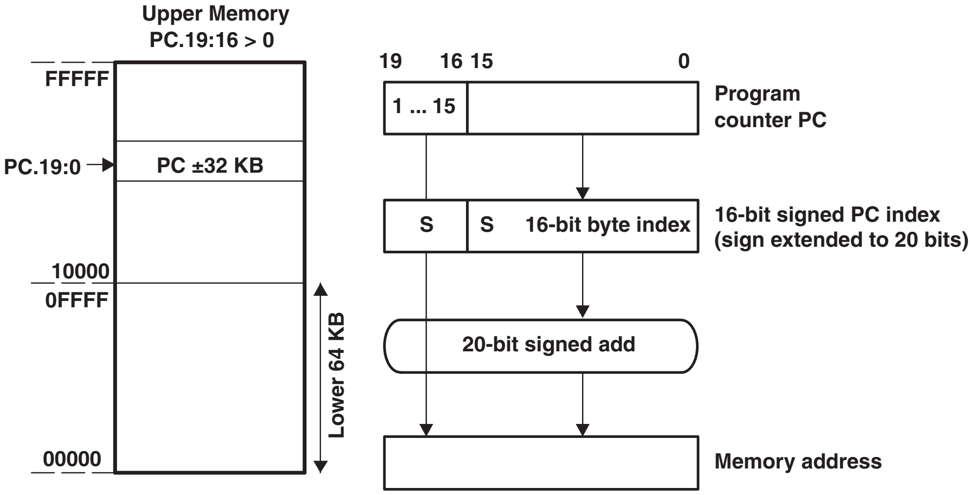


Figure 5-20. Symbolic Mode Running in Upper Memory

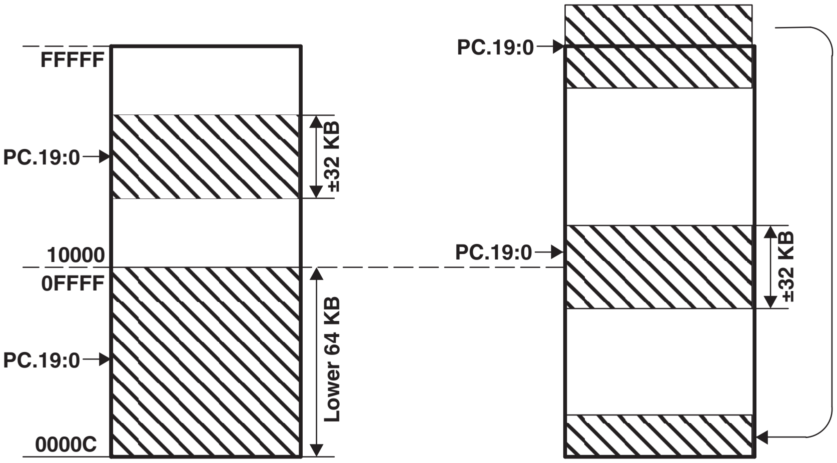


Figure 5-21. Overflow and Underflow for the Symbolic Mode

Length: Two or three words

Operation: The sign-extended 16-bit index in the next word after the instruction is added to the 20 bits of the PC. This delivers a 20-bit address, which points to an address in the range 0 to FFFFFh. The operand is the content of the addressed memory location.

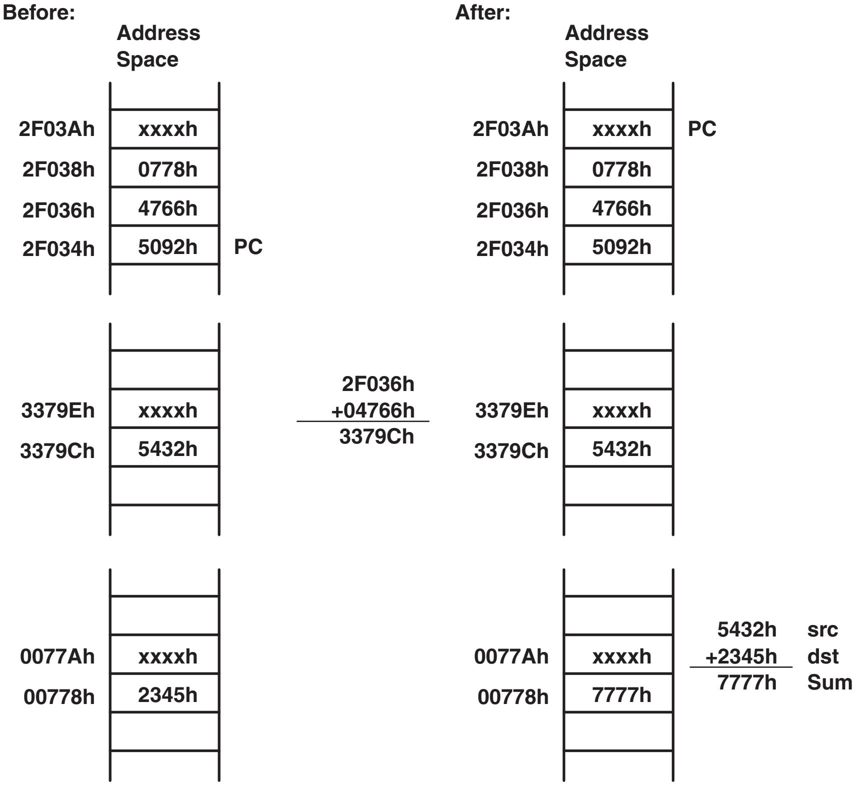
Comment: Valid for source and destination. The assembler calculates the PC index and inserts it

Example: `ADD.W EDE,&TONI ;`

This instruction adds the 16-bit data contained in source word EDE and destination word TONI and places the 16-bit result into the destination word TONI. For this example, the instruction is located at address 2,F034h.

Source: Word EDE at address 3379Ch, pointed to by PC + 4766h which is the 16-bit result of 3379Ch - 2F036h = 04766h. Address 2F036h is the location of the index for this example.

Destination: Word TONI located at address 00778h pointed to by the absolute address 00778h.



MSP430X Instruction with Symbolic Mode

When using an MSP430X instruction with Symbolic mode, the operand can be located anywhere in the range of PC + 19 bits.

Length: Three or four words

Operation: The operand address is the sum of the 20-bit PC and the 20-bit index. The four MSBs of the index are contained in the extension word, the 16 LSBs are contained in the word following the instruction.

Comment: Valid for source and destination. The assembler calculates the register index and inserts it.

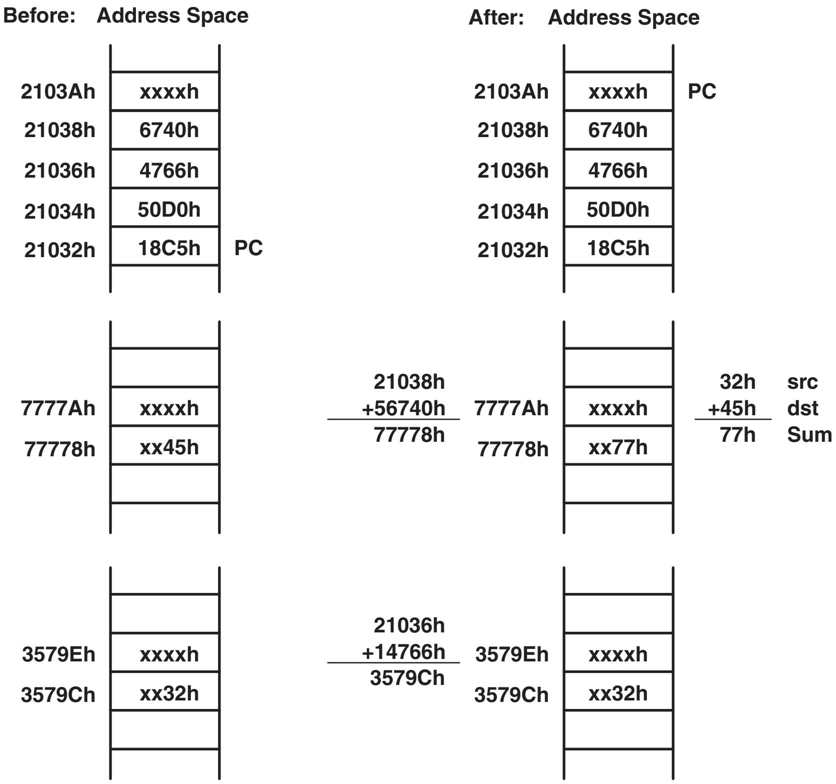
Example:

```
ADDX.B EDE,TONI ;
```

This instruction adds the 8-bit data contained in source byte EDE and destination byte TONI and places the result into the destination byte TONI.

Source: Byte EDE located at address 3579Ch, pointed to by PC + 14766h, is the 20-bit result of 3579Ch - 21036h = 14766h. Address 21036h is the address of the index in this example.

Destination: Byte TONI located at address 77778h, pointed to by PC + 56740h, is the 20-bit result of 77778h - 21038h = 56740h. Address 21038h is the address of the index in this example.



5.4.4 Absolute Mode

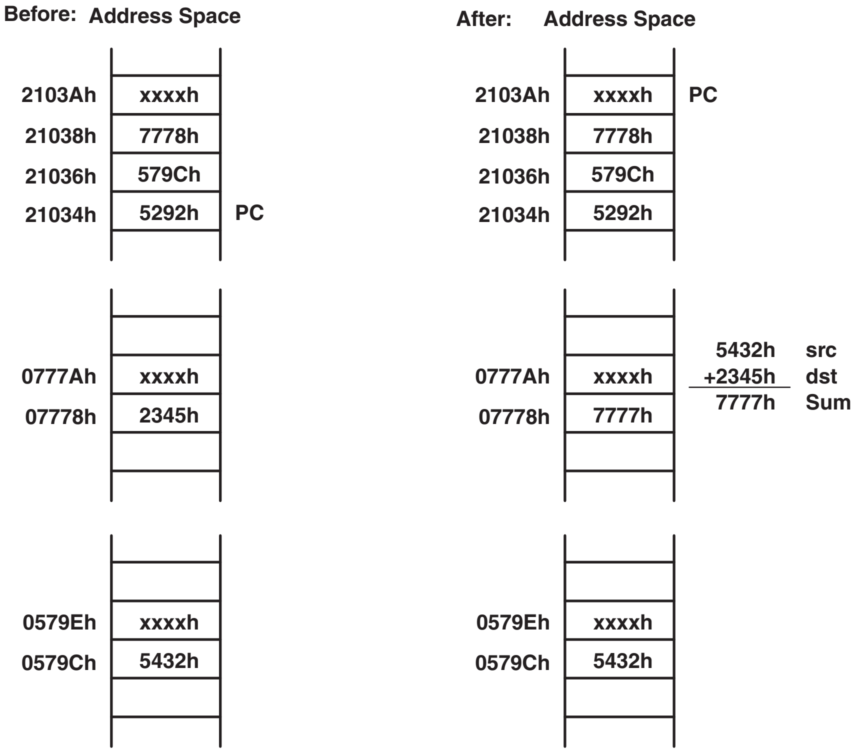
The Absolute mode uses the contents of the word following the instruction as the address of the operand. The Absolute mode has two addressing possibilities:

- Absolute mode in lower 64-KB memory
- MSP430X instruction with Absolute mode

Absolute Mode in Lower 64 KB

If an MSP430 instruction is used with Absolute addressing mode, the absolute address is a 16-bit value and therefore points to an address in the lower 64 KB of the memory range. The address is calculated as an index from 0 and is stored in the word following the instruction The RAM and the peripheral registers can be accessed this way and existing MSP430 software is usable without modifications.

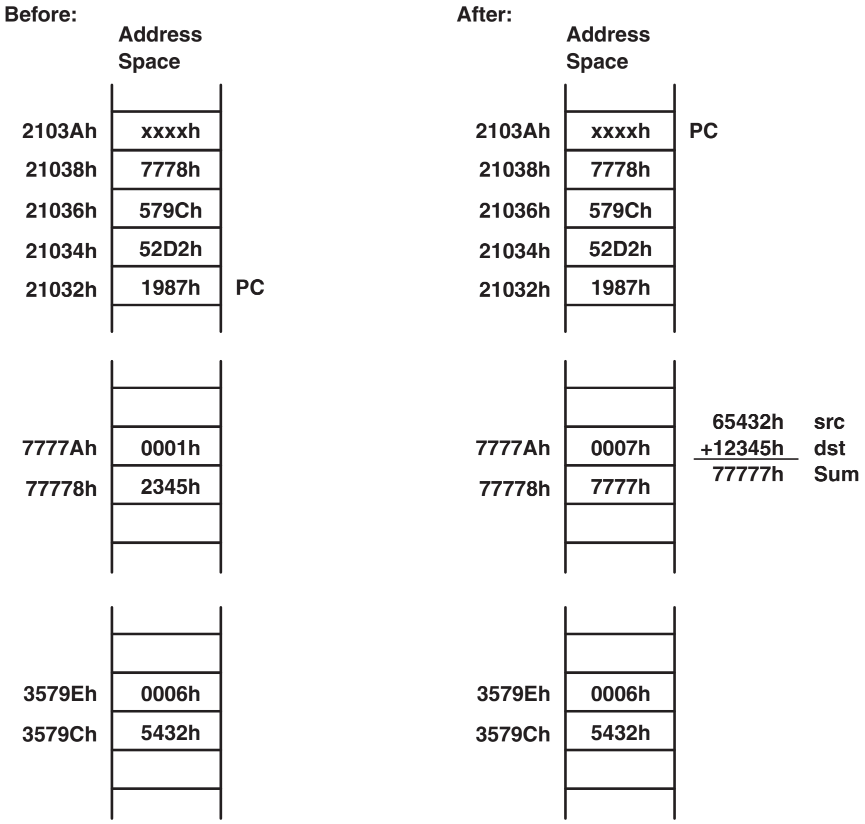
Length:	Two or three words
Operation:	The operand is the content of the addressed memory location.
Comment:	Valid for source and destination. The assembler calculates the index from 0 and inserts it.
Example:	`ADD.W &EDE,&TONI ;` This instruction adds the 16-bit data contained in the absolute source and destination addresses and places the result into the destination.
Source:	Word at address EDE
Destination:	Word at address TONI



MSP430X Instruction with Absolute Mode

If an MSP430X instruction is used with Absolute addressing mode, the absolute address is a 20-bit value and therefore points to any address in the memory range. The address value is calculated as an index from 0. The four MSBs of the index are contained in the extension word, and the 16 LSBs are contained in the word following the instruction.

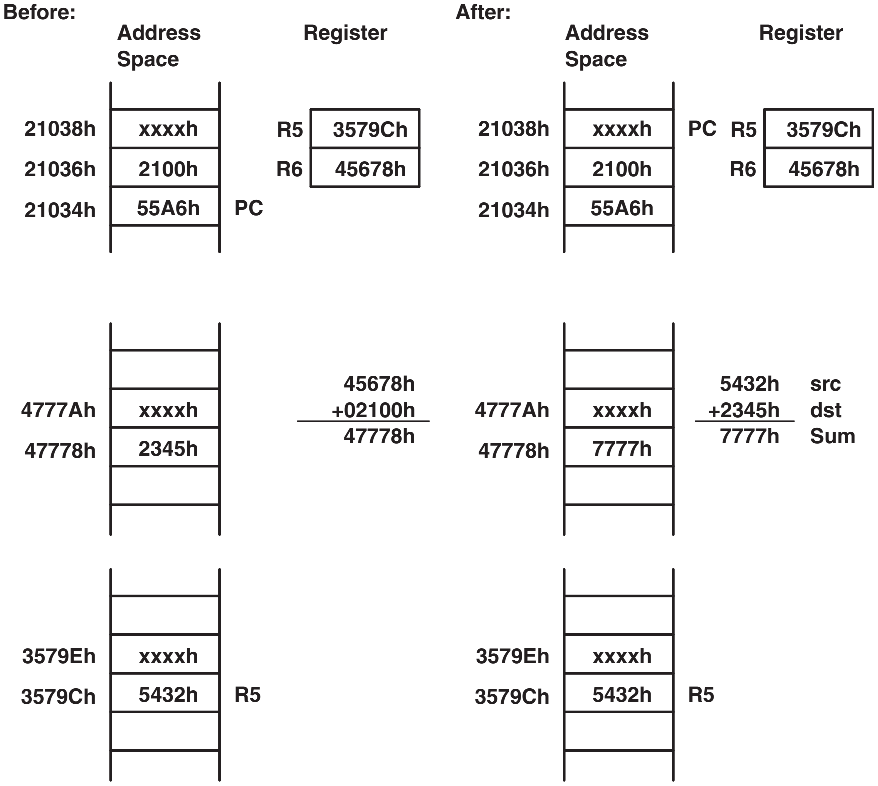
Length:	Three or four words
Operation:	The operand is the content of the addressed memory location.
Comment:	Valid for source and destination. The assembler calculates the index from 0 and inserts it.
Example:	`ADDX.A &EDE,&TONI ;`
	This instruction adds the 20-bit data contained in the absolute source and destination addresses and places the result into the destination.
Source:	Two words beginning with address EDE
Destination:	Two words beginning with address TONI



5.4.5 Indirect Register Mode

The Indirect Register mode uses the contents of the CPU register Rsrc as the source operand. The Indirect Register mode always uses a 20-bit address.

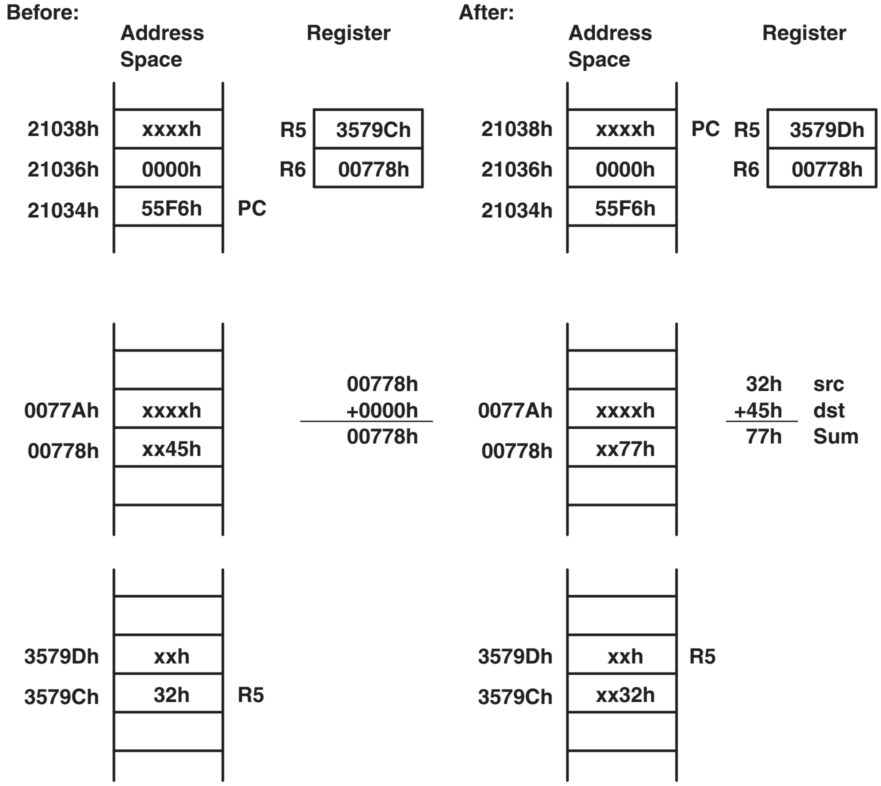
Length:	One, two, or three words
Operation:	The operand is the content the addressed memory location. The source register Rsrc is not modified.
Comment:	Valid only for the source operand. The substitute for the destination operand is 0(Rdst).
Example:	`ADDX.W @R5,2100h(R6)`
	This instruction adds the two 16-bit operands contained in the source and the destination addresses and places the result into the destination.
Source:	Word pointed to by R5. R5 contains address 3579Ch for this example.
Destination:	Word pointed to by R6 + 2100h which results in address 45678h + 2100h = 7778h.



5.4.6 Indirect, Autoincrement Mode

The Indirect Autoincrement mode uses the contents of the CPU register Rsrc as the source operand. Rsrc is then automatically incremented by 1 for byte instructions, by 2 for word instructions, and by 4 for address-word instructions immediately after accessing the source operand. If the same register is used for source and destination, it contains the incremented address for the destination access. Indirect Autoincrement mode always uses 20-bit addresses.

Length:	One, two, or three words
Operation:	The operand is the content of the addressed memory location.
Comment:	Valid only for the source operand
Example:	`ADD.B @R5+,0(R6)`
	This instruction adds the 8-bit data contained in the source and the destination addresses and places the result into the destination.
Source:	Byte pointed to by R5. R5 contains address 3,579Ch for this example
Destination:	Byte pointed to by R6 + 0h which results in address 0778h for this example



5.4.7 Immediate Mode

odeThe Immediate mode allows accessing constants as operands by including the constant in the memory location following the instruction. The program counter PC is used with the Indirect Autoincrement mode. The PC points to the immediate value contained in the next word. After the fetching of the immediate operand, the PC is incremented by 2 for byte, word, or address-word instructions. The Immediate mode has two addressing possibilities:

- 8-bit or 16-bit constants with MSP430 instructions
- 20-bit constants with MSP430X instruction

MSP430 Instructions With Immediate Mode

If an MSP430 instruction is used with Immediate addressing mode, the constant is an 8- or 16-bit value and is stored in the word following the instruction.

Length: Two or three words. One word less if a constant of the constant generator can be used for the immediate operand.

Operation: The 16-bit immediate source operand is used together with the 16-bit destination operand.

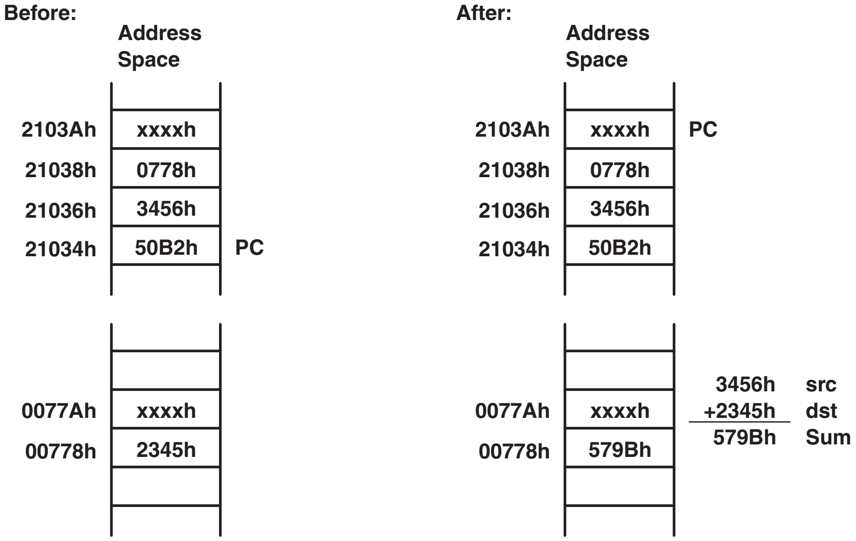
Comment: Valid only for the source operand

Example: `ADD #3456h,&TONI`

This instruction adds the 16-bit immediate operand 3456h to the data in the destination address TONI.

Source: 16-bit immediate value 3456h

Destination: Word at address TONI



MSP430X Instructions With Immediate Mode

If an MSP430X instruction is used with immediate addressing mode, the constant is a 20-bit value. The 4 MSBs of the constant are stored in the extension word and the 16 LSBs of the constant are stored in the word following the instruction.

Length: Three or four words. One word less if a constant of the constant generator can be used for the immediate operand.

Operation: The 20-bit immediate source operand is used together with the 20-bit destination operand.

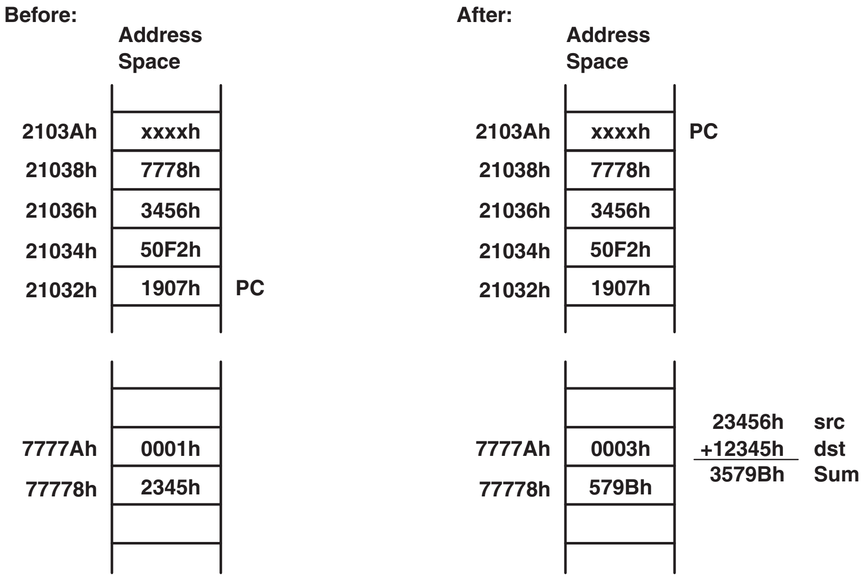
Comment: Valid only for the source operand

Example: `ADDX.A #23456h,&TONI ;`

This instruction adds the 20-bit immediate operand 23456h to the data in the destination address TONI.

Source: 20-bit immediate value 23456h

Destination: Two words beginning with address TONI


Before:
Address Space
2103Ah xxxxh
21038h 7778h
21036h 3456h
21034h 50F2h
21032h 1907h PC
7777Ah 0001h
77778h 2345h
After:
Address Space
2103Ah xxxxh PC
21038h 7778h
21036h 3456h
21034h 50F2h
21032h 1907h
7777Ah 0003h
77778h 579Bh
23456h src
+12345h dst
3579Bh Sum

5.5 MSP430 and MSP430X Instructions

MSP430 instructions are the 27 implemented instructions of the MSP430 CPU. These instructions are used throughout the 1-MB memory range unless their 16-bit capability is exceeded. The MSP430X instructions are used when the addressing of the operands or the data length exceeds the 16-bit capability of the MSP430 instructions.

There are three possibilities when choosing between an MSP430 and MSP430X instruction:

- To use only the MSP430 instructions: The only exceptions are the CALLA and the RETA instruction. This can be done if a few, simple rules are met:
  - Placement of all constants, variables, arrays, tables, and data in the lower 64 KB. This allows the use of MSP430 instructions with 16-bit addressing for all data accesses. No pointers with 20-bit addresses are needed.
  - Placement of subroutine constants immediately after the subroutine code. This allows the use of the symbolic addressing mode with its 16-bit index to reach addresses within the range of PC +32 KB.
- To use only MSP430X instructions: The disadvantages of this method are the reduced speed due to the additional CPU cycles and the increased program space due to the necessary extension word for any double operand instruction.
- Use the best fitting instruction where needed

The following sections list and describe the MSP430 and MSP430X instructions.

5.5.1 *MSP430 Instructions*

The MSP430 instructions can be used, regardless if the program resides in the lower 64 KB or beyond it. The only exceptions are the instructions CALL and RET which are limited to the lower 64 KB address range. CALLA and RETA instructions have been added to the MSP430X CPU to handle subroutines in the entire address range with no code size overhead.

MSP430 Double Operand (Format I) Instructions

Figure 5-22 shows the format of the MSP430 double operand instructions. Source and destination words are appended for the Indexed, Symbolic, Absolute and Immediate modes. Table 5-4 lists the twelve MSP430 double operand instructions.

15 12	11 8	7	6	5 4	0
Op-code	Rsrc	Ad	B/W	As	Rdst
Source or Destination 15:0					
Destination 15:0					

Figure 5-22. MSP430 Double Operand Instruction Format

Table 5-4. MSP430 Double Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits[1]			
			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	*	*	*	*

[1] * = The status bit is affected.
– = The status bit is not affected.
0 = The status bit is cleared.
1 = The status bit is set.

Table 5-4. MSP430 Double Operand Instructions (continued)

Mnemonic	S-Reg, D-Reg	Operation	Status Bits[1]			
			V	N	Z	C
`CMP(.B)`	src,dst	dst - src	*	*	*	*
`DADD(.B)`	src,dst	src + dst + C → dst (decimally)	*	*	*	*
`BIT(.B)`	src,dst	src .and. dst	0	*	*	Z
`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	*	*	*	Z
`AND(.B)`	src,dst	src .and. dst → dst	0	*	*	Z

MSP430 Single Operand (Format II) Instructions

Figure 5-23 shows the format for MSP430 single operand instructions, except RETI. The destination word is appended for the Indexed, Symbolic, Absolute and Immediate modes. Table 5-5 lists the seven single operand instructions.

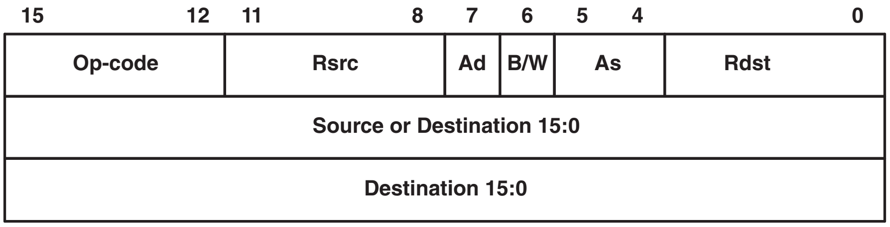


Figure 5-23. MSP430 Single Operand Instructions

Table 5-5. MSP430 Single Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits[1]			
			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	bit 15...bit 8 ↔ bit 7...bit 0	–	–	–	–
`CALL`	dst	Call subroutine in lower 64 KB	–	–	–	–
`RETI`		TOS → SR, SP + 2 → SP	*	*	*	*
		TOS → PC,SP + 2 → SP				
`SXT`	dst	Register mode: bit 7 → bit 8...bit 19 Other modes: bit 7 → bit 8...bit 15	0	*	*	Z

[1] * = The status bit is affected.
– = The status bit is not affected.
0 = The status bit is cleared.
1 = The status bit is set.

Jumps

Figure 5-24 shows the format for MSP430 and MSP430X jump instructions. The signed 10-bit word offset of the jump instruction is multiplied by two, sign-extended to a 20-bit address, and added to the 20-bit program counter. This allows jumps in a range of -511 to +512 words relative to the program counter in the full 20-bit address space Jumps do not affect the status bits. Table 5-6 lists and describes the eight jump instructions.

15 13	12 10	9	8 0
Op-Code	Condition	S	10-Bit Signed PC Offset

Figure 5-24. Format of the Conditional Jump Instructions

Table 5-6. Conditional Jump Instructions

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

Emulated Instructions

In addition to the MSP430 and MSP430X instructions, 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 a core instruction. There is no code or performance penalty for using emulated instructions. The emulated instructions are listed in Table 5-7.

Table 5-7. Emulated Instructions

Instruction	Explanation	Emulation	Status Bits[1]			
			V	N	Z	C
ADC(.B) dst	Add Carry to dst	ADDC(.B) #0,dst	*	*	*	*
BR dst	Branch indirectly dst	MOV dst,PC	–	–	–	–
CLR(.B) dst	Clear dst	MOV(.B) #0,dst	–	–	–	–
CLRC	Clear Carry bit	BIC #1,SR	–	–	–	0
CLRN	Clear Negative bit	BIC #4,SR	–	0	–	–
CLRZ	Clear Zero bit	BIC #2,SR	–	–	0	–
DADC(.B) dst	Add Carry to dst decimally	DADD(.B) #0,dst	*	*	*	*
DEC(.B) dst	Decrement dst by 1	SUB(.B) #1,dst	*	*	*	*
DECD(.B) dst	Decrement dst by 2	SUB(.B) #2,dst	*	*	*	*
DINT	Disable interrupt	BIC #8,SR	–	–	–	–
EINT	Enable interrupt	BIS #8,SR	–	–	–	–
INC(.B) dst	Increment dst by 1	ADD(.B) #1,dst	*	*	*	*
INCD(.B) dst	Increment dst by 2	ADD(.B) #2,dst	*	*	*	*
INV(.B) dst	Invert dst	XOR(.B) #–1,dst	*	*	*	*
NOP	No operation	MOV R3,R3	–	–	–	–
POP dst	Pop operand from stack	MOV @SP+,dst	–	–	–	–
RET	Return from subroutine	MOV @SP+,PC	–	–	–	–
RLA(.B) dst	Shift left dst arithmetically	ADD(.B) dst,dst	*	*	*	*
RLC(.B) dst	Shift left dst logically through Carry	ADDC(.B) dst,dst	*	*	*	*
SBC(.B) dst	Subtract Carry from dst	SUBC(.B) #0,dst	*	*	*	*

[1] * = The status bit is affected.
– = The status bit is not affected.
0 = The status bit is cleared.
1 = The status bit is set.

Table 5-7. Emulated Instructions (continued)

Instruction	Explanation	Emulation	Status Bits[1]			
			V	N	Z	C
`SETC`	Set Carry bit	`BIS #1,SR`	–	–	–	1
`SETN`	Set Negative bit	`BIS #4,SR`	–	1	–	–
`SETZ`	Set Zero bit	`BIS #2,SR`	–	–	1	–
`TST(.B) dst`	Test dst (compare with 0)	`CMP(.B) #0,dst`	0	*	*	1

MSP430 Instruction Execution

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 MCLK.

Instruction Cycles and Length for Interrupt, Reset, and Subroutines

Table 5-8 lists the length and the CPU cycles for reset, interrupts, and subroutines.

Table 5-8. Interrupt, Return, and Reset Cycles and Length

Action	Execution Time (MCLK Cycles)	Length of Instruction (Words)
Return from interrupt RETI	5	1
Return from subroutine RET	4	1
Interrupt request service (cycles needed before first instruction)	6	–
WDT reset	4	–
Reset ($\overline{\text{RST}}$/NMI)	4	–

Format-II (Single Operand) Instruction Cycles and Lengths

Table 5-9 lists the length and the CPU cycles for all addressing modes of the MSP430 single operand instructions.

Table 5-9. MSP430 Format-II Instruction Cycles and Length

Addressing Mode	No. of Cycles			Length of Instruction	Example
	RRA, RRC SWPB, SXT	PUSH	CALL		
Rn	1	3	4	1	`SWPB R5`
@Rn	3	3	4	1	`RRC @R9`
@Rn+	3	3	4	1	`SWPB @R10+`
#N	N/A	3	4	2	`CALL #LABEL`
X(Rn)	4	4	5	2	`CALL 2(R7)`
EDE	4	4	5	2	`PUSH EDE`
&EDE	4	4	6	2	`SXT &EDE`

Jump Instructions 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 5-10 lists the length and CPU cycles for all addressing modes of the MSP430 format-I instructions.

Table 5-10. MSP430 Format-I Instructions Cycles and Length

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

[1] MOV, BIT, and CMP instructions execute in one fewer cycle.

5.5.2 *MSP430X Extended Instructions*

The extended MSP430X instructions give the MSP430X CPU full access to its 20-bit address space. Most MSP430X instructions require an additional word of op-code called the extension word. Some extended instructions do not require an additional word and are noted in the instruction description. All addresses, indexes and immediate numbers have 20-bit values, when preceded by the extension word.

There are two types of extension word:

- Register/register mode for Format-I instructions and register mode for Format-II instructions
- Extension word for all other address mode combinations

Register Mode Extension Word

The register mode extension word is shown in Figure 5-25 and described in Table 5-11. An example is shown in Figure 5-27.

15 12	11	10 9	8	7	6	5	4	3 0
0001	1	00	ZC	#	A/L	0	0	(n-1)/Rn

Figure 5-25. Extension Word for Register Modes

Table 5-11. Description of the Extension Word Bits for Register Mode

Bit	Description
15:11	Extension word op-code. Op-codes 1800h to 1FFFh are extension words.
10:9	Reserved
ZC	Zero carry bit
	0 The executed instruction uses the status of the carry bit C.
	1 The executed instruction uses the carry bit as 0. The carry bit will be defined by the result of the final operation after instruction execution.
#	Repetition bit
	0 The number of instruction repetitions is set by extension-word bits 3:0.
	1 The number of instructions repetitions is defined by the value of the four LSBs of Rn. See description for bits 3:0.
A/L	Data length extension bit. Together with the B/W bits of the following MSP430 instruction, the AL bit defines the used data length of the instruction.
	A/L **B/W** **Comment**
	0 0 Reserved
	0 1 20-bit address word
	1 0 16-bit word
	1 1 8-bit byte
5:4	Reserved
3:0	Repetition count
	# = 0 These four bits set the repetition count n. These bits contain n – 1.
	# = 1 These four bits define the CPU register whose bits 3:0 set the number of repetitions. Rn.3:0 contain n – 1.

Non-Register Mode Extension Word

The extension word for non-register modes is shown in Figure 5-26 and described in Table 5-12. An example is shown in Figure 5-28.

15 12	11	10 7	6	5	4	3 0
0 0 0 1	1	Source bits 19:16	A/L	0	0	Destination bits 19:16

Figure 5-26. Extension Word for Non-Register Modes

Table 5-12. Description of the Extension Word Bits for Non-Register Modes

Bit	Description
15:11	Extension word op-code. Op-codes 1800h to 1FFFh are extension words.
Source Bits 19:16	The four MSBs of the 20-bit source. Depending on the source addressing mode, these four MSBs may belong to an immediate operand, an index or to an absolute address.

Table 5-12. Description of the Extension Word Bits for Non-Register Modes (continued)

Bit	Description
A/L	Data length extension bit. Together with the B/W-bits of the following MSP430 instruction, the AL bit defines the used data length of the instruction.
5:4	Reserved
Destination Bits 19:16	The four MSBs of the 20-bit destination. Depending on the destination addressing mode, these four MSBs may belong to an index or to an absolute address.

A/L	B/W	Comment
0	0	Reserved
0	1	20-bit address word
1	0	16-bit word
1	1	8-bit byte

Note: B/W and A/L Bit Settings for SWPBX and SXTX

The B/W and A/L bit settings for SWPBX and SXTX are:

A/L	B/W	
0	0	SWPBX.A, SXTX.A
0	1	N/A
1	0	SWPB.W, SXTX.W
1	1	N/A

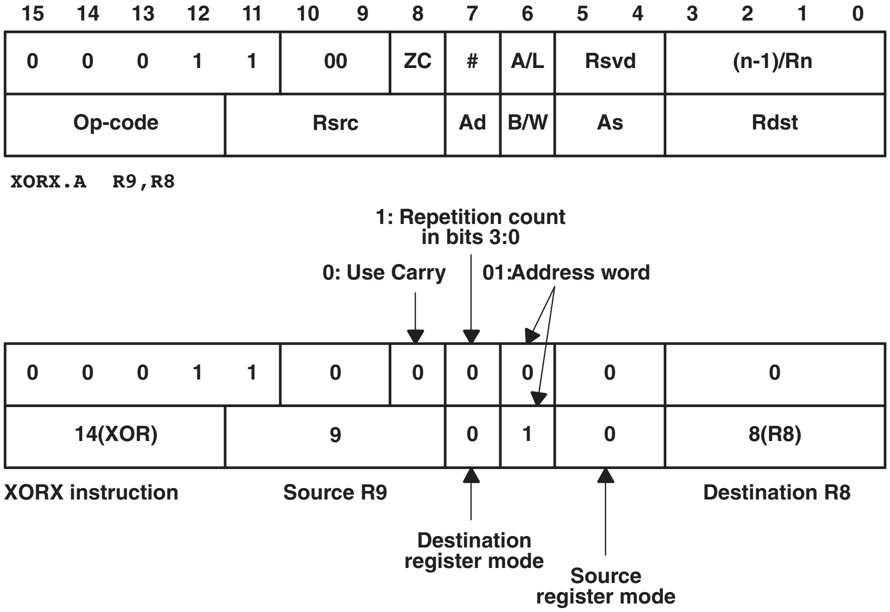


Figure 5-27. Example for an Extended Register/Register Instruction

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	Source 19:16				A/L	Rsvd		Destination 19:16			
Op-code				Rsrc				Ad	B/W	As		Rdst			
Source 15:0															
Destination 15:0															

```
XORX.A #12345h, 45678h(R15)
```

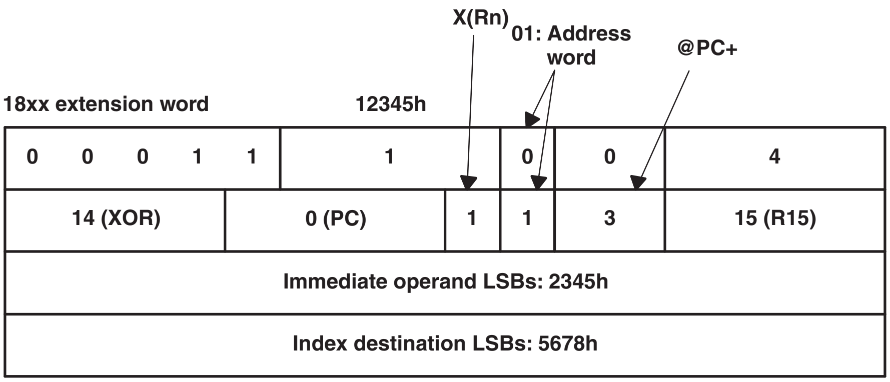


Figure 5-28. Example for an Extended Immediate/Indexed Instruction

Extended Double Operand (Format-I) Instructions

All twelve double-operand instructions have extended versions as listed in Table 1-13.

Table 5-13. Extended Double Operand Instructions

Mnemonic	Operands	Operation	Status Bits[1]			
			V	N	Z	C
`MOVX(.B,.A)`	src,dst	src → dst	–	–	–	–
`ADDX(.B,.A)`	src,dst	src + dst → dst	*	*	*	*
`ADDCX(.B,.A)`	src,dst	src + dst + C → dst	*	*	*	*
`SUBX(.B,.A)`	src,dst	dst + .not.src + 1 → dst	*	*	*	*
`SUBCX(.B,.A)`	src,dst	dst + .not.src + C → dst	*	*	*	*
`CMPX(.B,.A)`	src,dst	dst – src	*	*	*	*
`DADDX(.B,.A)`	src,dst	src + dst + C → dst (decimal)	*	*	*	*
`BITX(.B,.A)`	src,dst	src .and. dst	0	*	*	Z
`BICX(.B,.A)`	src,dst	.not.src .and. dst → dst	–	–	–	–
`BISX(.B,.A)`	src,dst	src .or. dst → dst	–	–	–	–
`XORX(.B,.A)`	src,dst	src .xor. dst → dst	*	*	*	Z
`ANDX(.B,.A)`	src,dst	src .and. dst → dst	0	*	*	Z

[1] * = The status bit is affected.
– = The status bit is not affected.
0 = The status bit is cleared.
1 = The status bit is set.

The four possible addressing combinations for the extension word for format-I instructions are shown in Figure 5-29.

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th></th><th></th><th>0</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>ZC</td><td>#</td><td>A/L</td><td>0</td><td>0</td><td colspan="4">n-1/Rn</td></tr>
<tr><td colspan="4">Op-code</td><td colspan="4">src</td><td>0</td><td>B/W</td><td>0</td><td>0</td><td colspan="4">dst</td></tr>
</table>

<table>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td colspan="4">src.19:16</td><td>A/L</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="4">Op-code</td><td colspan="4">src</td><td>Ad</td><td>B/W</td><td colspan="2">As</td><td colspan="4">dst</td></tr>
<tr><td colspan="16">src.15:0</td></tr>
</table>

<table>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>A/L</td><td>0</td><td>0</td><td colspan="4">dst.19:16</td></tr>
<tr><td colspan="4">Op-code</td><td colspan="4">src</td><td>Ad</td><td>B/W</td><td colspan="2">As</td><td colspan="4">dst</td></tr>
<tr><td colspan="16">dst.15:0</td></tr>
</table>

<table>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td colspan="4">src.19:16</td><td>A/L</td><td>0</td><td>0</td><td colspan="4">dst.19:16</td></tr>
<tr><td colspan="4">Op-code</td><td colspan="4">src</td><td>Ad</td><td>B/W</td><td colspan="2">As</td><td colspan="4">dst</td></tr>
<tr><td colspan="16">src.15:0</td></tr>
<tr><td colspan="16">dst.15:0</td></tr>
</table>

Figure 5-29. Extended Format-I Instruction Formats

If the 20-bit address of a source or destination operand is located in memory, not in a CPU register, then two words are used for this operand as shown in Figure 5-30.

<table>
<tr><th></th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>Address+2</td><td colspan="12">0 ....0</td><td colspan="4">19:16</td></tr>
<tr><td>Address</td><td colspan="16">Operand LSBs 15:0</td></tr>
</table>

Figure 5-30. 20-Bit Addresses in Memory

Extended Single Operand (Format-II) Instructions

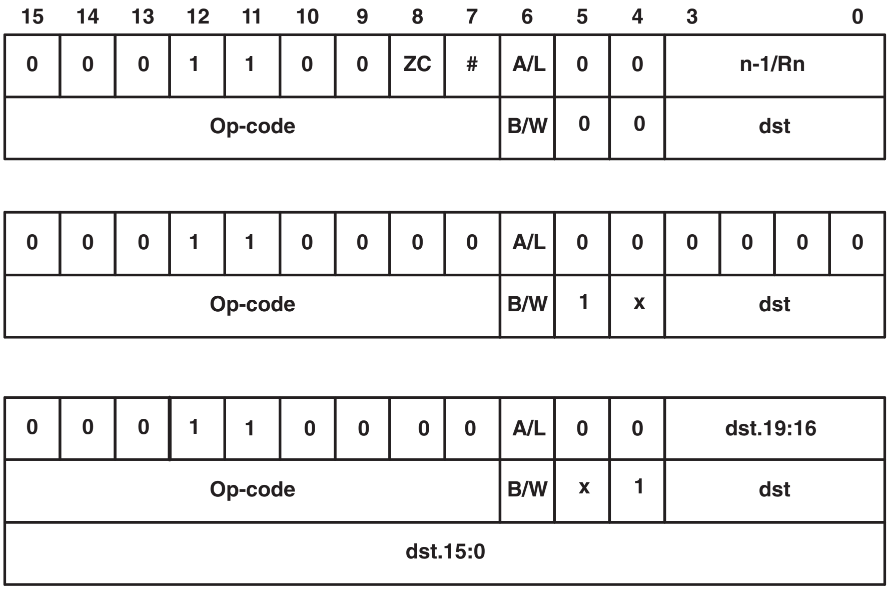
Extended MSP430X Format-II instructions are listed in Table 5-14.

Table 5-14. Extended Single-Operand Instructions

Mnemonic	Operands	Operation	n	Status Bits[(1)] V	N	Z	C
CALLA	dst	Call indirect to subroutine (20-bit address)		–	–	–	–
POPM.A	#n,Rdst	Pop n 20-bit registers from stack	1 to 16	*	*	*	*
POPM.W	#n,Rdst	Pop n 16-bit registers from stack	1 to 16	*	*	*	*
PUSHM.A	#n,Rsrc	Push n 20-bit registers to stack	1 to 16	*	*	*	*
PUSHM.W	#n,Rsrc	Push n 16-bit registers to stack	1 to 16	*	*	*	*
PUSHX(.B,.A)	src	Push 8/16/20-bit source to stack		*	*	*	*
RRCM(.A)	#n,Rdst	Rotate right Rdst n bits through carry (16-/20-bit register)	1 to 4	*	*	*	*
RRUM(.A)	#n,Rdst	Rotate right Rdst n bits unsigned (16-/20-bit register)	1 to 4	0	*	*	Z
RRAM(.A)	#n,Rdst	Rotate right Rdst n bits arithmetically (16-/20-bit register)	1 to 4	–	–	–	–
RLAM(.A)	#n,Rdst	Rotate left Rdst n bits arithmetically (16-/20-bit register)	1 to 4	–	–	–	–
RRCX(.B,.A)	dst	Rotate right dst through carry (8-/16-/20-bit data)	1	*	*	*	Z
RRUX(.B,.A)	dst	Rotate right dst unsigned (8-/16-/20-bit)	1	0	*	*	Z
RRAX(.B,.A)	dst	Rotate right dst arithmetically	1				
SWPBX(.A)	dst	Exchange low byte with high byte	1				
SXTX(.A)	Rdst	Bit7 → bit8 ... bit19	1				
SXTX(.A)	dst	Bit7 → bit8 ... MSB	1				

[(1)] * = The status bit is affected.
– = The status bit is not affected.
0 = The status bit is cleared.
1 = The status bit is set.

The three possible addressing mode combinations for format-II instructions are shown in Figure 5-31.



15	14	13	12	11	10	9	8	7	6	5	4	3			0
0	0	0	1	1	0	0	ZC	#	A/L	0	0	n-1/Rn			
Op-code									B/W	0	0	dst			

0	0	0	1	1	0	0	0	0	A/L	0	0	0	0	0	0
Op-code									B/W	1	x	dst			

0	0	0	1	1	0	0	0	0	A/L	0	0	dst.19:16			
Op-code									B/W	x	1	dst			
dst.15:0															

Figure 5-31. Extended Format-II Instruction Format

Extended Format II Instruction Format Exceptions

Exceptions for the Format II instruction formats are shown in Figure 5-32 through Figure 5-35.

15 8	7 4	3 0
Op-code	n-1	Rdst - n+1

Figure 5-32. PUSHM/POPM Instruction Format

15 12	11 10	9 4	3 0
C	n-1	Op-code	Rdst

Figure 5-33. RRCM, RRAM, RRUM and RLAM Instruction Format

15 12	11 8	7 4	3 0
C	Rsrc	Op-code	0(PC)

C	#imm/abs19:16	Op-code	0(PC)
#imm15:0 / &abs15:0			

C	Rsrc	Op-code	0(PC)
index15:0			

Figure 5-34. BRA Instruction Format

15 4	3 0
Op-code	Rdst

Op-code	Rdst
index15:0	

Op-code	#imm/ix/abs19:16
#imm15:0 / index15:0 / &abs15:0	

Figure 5-35. CALLA Instruction Format

Extended Emulated Instructions

The extended instructions together with the constant generator form the extended emulated instructions. Table 5-15 lists the emulated instructions.

Table 5-15. Extended Emulated Instructions

Instruction	Explanation	Emulation
`ADCX(.B,.A) dst`	Add carry to dst	`ADDCX(.B,.A) #0,dst`
`BRA dst`	Branch indirect dst	`MOVA dst,PC`
`RETA`	Return from subroutine	`MOVA @SP+,PC`
`CLRA Rdst`	Clear Rdst	`MOV #0,Rdst`
`CLRX(.B,.A) dst`	Clear dst	`MOVX(.B,.A) #0,dst`
`DADCX(.B,.A) dst`	Add carry to dst decimally	`DADDX(.B,.A) #0,dst`
`DECX(.B,.A) dst`	Decrement dst by 1	`SUBX(.B,.A) #1,dst`
`DECDA Rdst`	Decrement Rdst by 2	`SUBA #2,Rdst`
`DECDX(.B,.A) dst`	Decrement dst by 2	`SUBX(.B,.A) #2,dst`
`INCX(.B,.A) dst`	Increment dst by 1	`ADDX(.B,.A) #1,dst`
`INCDA Rdst`	Increment Rdst by 2	`ADDA #2,Rdst`
`INCDX(.B,.A) dst`	Increment dst by 2	`ADDX(.B,.A) #2,dst`
`INVX(.B,.A) dst`	Invert dst	`XORX(.B,.A) #-1,dst`
`RLAX(.B,.A) dst`	Shift left dst arithmetically	`ADDX(.B,.A) dst,dst`
`RLCX(.B,.A) dst`	Shift left dst logically through carry	`ADDCX(.B,.A) dst,dst`
`SBCX(.B,.A) dst`	Subtract carry from dst	`SUBCX(.B,.A) #0,dst`
`TSTA Rdst`	Test Rdst (compare with 0)	`CMPA #0,Rdst`
`TSTX(.B,.A) dst`	Test dst (compare with 0)	`CMPX(.B,.A) #0,dst`
`POPX dst`	Pop to dst	`MOVX(.B, .A) @SP+,dst`

MSP430X Address Instructions

MSP430X address instructions are instructions that support 20-bit operands but have restricted addressing modes. The addressing modes are restricted to the register mode and the Immediate mode, except for the MOVA instruction as listed in Table 5-16. Restricting the addressing modes removes the need for the additional extension-word op-code improving code density and execution time. Address instructions should be used any time an MSP430X instruction is needed with the corresponding restricted addressing mode.

Table 5-16. Address Instructions, Operate on 20-Bit Register Data

Mnemonic	Operands	Operation	Status Bits[1]			
			V	N	Z	C
`ADDA`	`Rsrc,Rdst`	Add source to destination register	*	*	*	*
	`#imm20,Rdst`					
`MOVA`	`Rsrc,Rdst`	Move source to destination	–	–	–	–
	`#imm20,Rdst`					
	`z16(Rsrc),Rdst`					
	`EDE,Rdst`					
	`&abs20,Rdst`					
	`@Rsrc,Rdst`					
	`@Rsrc+,Rdst`					
	`Rsrc,z16(Rdst)`					
	`Rsrc,&abs20`					
`CMPA`	ADDA	Compare source to destination register	*	*	*	*
	ADDA					
`SUBA`	ADDA	Subtract source from destination register	*	*	*	*
	ADDA					

[1] * = The status bit is affected.
– = The status bit is not affected.
0 = The status bit is cleared.
1 = The status bit is set.

MSP430X Instruction Execution

The number of CPU clock cycles required for an MSP430X instruction depends on the instruction format and the addressing modes used, not the instruction itself. The number of clock cycles refers to MCLK.

MSP430X Format-II (Single-Operand) Instruction Cycles and Lengths

Table 5-17 lists the length and the CPU cycles for all addressing modes of the MSP430X extended single-operand instructions.

Table 5-17. MSP430X Format II Instruction Cycles and Length

Instruction	Execution Cycles/Length of Instruction (Words)						
	Rn	@Rn	@Rn+	#N	X(Rn)	EDE	&EDE
RRAM	n/1	–	–	–	–	–	–
RRCM	n/1	–	–	–	–	–	–
RRUM	n/1	–	–	–	–	–	–
RLAM	n/1	–	–	–	–	–	–
PUSHM	2+n/1	–	–	–	–	–	–
PUSHM.A	2+2n/1	–	–	–	–	–	–
POPM	2+n/1	–	–	–	–	–	–
POPM.A	2+2n/1	–	–	–	–	–	–
CALLA	5/1	6/1	6/1	5/2	5[1]/2	7/2	7/2
RRAX(.B)	1+n/2	4/2	4/2	–	5/3	5/3	5/3
RRAX.A	1+n/2	6/2	6/2	–	7/3	7/3	7/3
RRCX(.B)	1+n/2	4/2	4/2	–	5/3	5/3	5/3
RRCX.A	1+n/2	6/2	6/2	–	7/3	7/3	7/3
PUSHX(.B)	4/2	4/2	4/2	4/3	5[1]/3	5/3	5/3
PUSHX.A	5/2	6/2	6/2	5/3	7[1]/3	7/3	7/3
POPX(.B)	3/2	–	–	–	5/3	5/3	5/3
POPX.A	4/2	–	–	–	7/3	7/3	7/3

[1] Add one cycle when Rn = SP.

MSP430X Format-I (Double-Operand) Instruction Cycles and Lengths

Table 5-18 lists the length and CPU cycles for all addressing modes of the MSP430X extended format-I instructions.

Table 5-18. MSP430X Format-I Instruction Cycles and Length

Addressing Mode		No. of Cycles		Length of Instruction	Examples
Source	Destination	.B/.W	.A	.B/.W/.A	
Rn	Rm[1]	2	2	2	`BITX.B R5,R8`
	PC	4	4	2	`ADDX R9,PC`
	x(Rm)	5[2]	7[3]	3	`ANDX.A R5,4(R6)`
	EDE	5[2]	7[3]	3	`XORX R8,EDE`
	&EDE	5[2]	7[3]	3	`BITX.W R5,&EDE`
@Rn	Rm	3	4	2	`BITX @R5,R8`
	PC	5	6	2	`ADDX @R9,PC`
	x(Rm)	6[2]	9[3]	3	`ANDX.A @R5,4(R6)`
	EDE	6[2]	9[3]	3	`XORX @R8,EDE`
	&EDE	6[2]	9[3]	3	`BITX.B @R5,&EDE`
@Rn+	Rm	3	4	2	`BITX @R5+,R8`
	PC	5	6	2	`ADDX.A @R9+,PC`
	x(Rm)	6[2]	9[3]	3	`ANDX @R5+,4(R6)`
	EDE	6[2]	9[3]	3	`XORX.B @R8+,EDE`
	&EDE	6[2]	9[3]	3	`BITX @R5+,&EDE`
#N	Rm	3	3	33	`BITX #20,R8`
	PC[4]	4	4	3	`ADDX.A #FE000h,PC`
	x(Rm)	6[2]	8[3]	4	`ANDX #1234,4(R6)`
	EDE	6[2]	8[3]	4	`XORX #A5A5h,EDE`
	&EDE	6[2]	8[3]	4	`BITX.B #12,&EDE`
x(Rn)	Rm	4	5	3	`BITX 2(R5),R8`
	PC[4]	6	7	3	`SUBX.A 2(R6),PC`
	TONI	7[2]	10[3]	4	`ANDX 4(R7),4(R6)`
	x(Rm)	7[2]	10[3]	4	`XORX.B 2(R6),EDE`
	&TONI	7[2]	10[3]	4	`BITX 8(SP),&EDE`
EDE	Rm	4	5	3	`BITX.B EDE,R8`
	PC[4]	6	7	3	`ADDX.A EDE,PC`
	TONI	7[2]	10[3]	4	`ANDX EDE,4(R6)`
	x(Rm)	7[2]	10[3]	4	`ANDX EDE,TONI`
	&TONI	7[2]	10[3]	4	`BITX EDE,&TONI`
&EDE	Rm	4	5	3	`BITX &EDE,R8`
	PC[4]	6	7	3	`ADDX.A &EDE,PC`
	TONI	7[2]	10[3]	4	`ANDX.B &EDE,4(R6)`
	x(Rm)	7[2]	10[3]	4	`XORX &EDE,TONI`
	&TONI	7[2]	10[3]	4	`BITX &EDE,&TONI`

[1] Repeat instructions require n+1 cycles where n is the number of times the instruction is executed.
[2] Reduce the cycle count by one for MOV, BIT, and CMP instructions.
[3] Reduce the cycle count by two for MOV, BIT, and CMP instructions.
[4] Reduce the cycle count by one for MOV, ADD, and SUB instructions.

MSP430X Address Instruction Cycles and Lengths

Table 5-19 lists the length and the CPU cycles for all addressing modes of the MSP430X address instructions.

Table 5-19. Address Instruction Cycles and Length

Addressing Mode		Execution Time (MCLK Cycles)		Length of Instruction (Words)		Example
Source	Destination	MOVA BRA	CMPA ADDA SUBA	MOVA	CMPA ADDA SUBA	
Rn	Rn	1	1	1	1	`CMPA R5,R8`
	PC	3	3	1	1	`SUBA R9,PC`
	x(Rm)	4	–	2	–	`MOVA R5,4(R6)`
	EDE	4	–	2	–	`MOVA R8,EDE`
	&EDE	4	–	2	–	`MOVA R5,&EDE`
@Rn	Rm	3	–	1	–	`MOVA @R5,R8`
	PC	5	–	1	–	`MOVA @R9,PC`
@Rn+	Rm	3	–	1	–	`MOVA @R5+,R8`
	PC	5	–	1	–	`MOVA @R9+,PC`
#N	Rm	2	3	2	2	`CMPA #20,R8`
	PC	3	3	2	2	`SUBA #FE000h,PC`
x(Rn)	Rm	4	–	2	–	`MOVA 2(R5),R8`
	PC	6	–	2	–	`MOVA 2(R6),PC`
EDE	Rm	4	–	2	–	`MOVA EDE,R8`
	PC	6	–	2	–	`MOVA EDE,PC`
&EDE	Rm	4	–	2	–	`MOVA &EDE,R8`
	PC	6	–	2	–	`MOVA &EDE,PC`

5.6 Instruction Set Description

Table 5-20 shows all available instructions:

Table 5-20. Instruction Map of MSP430X

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx	MOVA, CMPA, ADDA, SUBA, RRCM, RRAM, RLAM, RRUM															
10xx	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI	CALLA		
14xx	PUSHM.A, POPM.A, PUSHM.W, POPM.W															
18xx	Extension Word For Format I and Format II Instructions															
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															

5.6.1 Extended Instruction Binary Descriptions

Detailed MSP430X instruction binary descriptions are shown in the following tables.

Instruction	Instruction Group				src or data.19:16	Instruction Identifier				dst	
	15			12	11 8	7			4	3 0	
MOVA	0	0	0	0	src	0	0	0	0	dst	`MOVA @Rsrc,Rdst`
	0	0	0	0	src	0	0	0	1	dst	`MOVA @Rsrc+,Rdst`
	0	0	0	0	&abs.19:16	0	0	1	0	dst	`MOVA &abs20,Rdst`
	&abs.15:0										
	0	0	0	0	src	0	0	1	1	dst	`MOVA x(Rsrc),Rdst`
	x.15:0										±15-bit index x
	0	0	0	0	src	0	1	1	0	&abs.19:16	`MOVA Rsrc,&abs20`
	&abs.15:0										
	0	0	0	0	src	0	1	1	1	dst	`MOVA Rsrc,X(Rdst)`
	x.15:0										±15-bit index x
	0	0	0	0	imm.19:16	1	0	0	0	dst	`MOVA #imm20,Rdst`
	imm.15:0										
CMPA	0	0	0	0	imm.19:16	1	0	0	1	dst	`CMPA #imm20,Rdst`
	imm.15:0										
ADDA	0	0	0	0	imm.19:16	1	0	1	0	dst	`ADDA #imm20,Rdst`
	imm.15:0										
SUBA	0	0	0	0	imm.19:16	1	0	1	1	dst	`SUBA #imm20,Rdst`
	imm.15:0										
MOVA	0	0	0	0	src	1	1	0	0	dst	`MOVA Rsrc,Rdst`
CMPA	0	0	0	0	src	1	1	0	1	dst	`CMPA Rsrc,Rdst`
ADDA	0	0	0	0	src	1	1	1	0	dst	`ADDA Rsrc,Rdst`
SUBA	0	0	0	0	src	1	1	1	1	dst	`SUBA Rsrc,Rdst`

Instruction	Instruction Group				Bit Loc.	Inst. ID		Instruction Identifier				dst	
	15			12	11 10	9	8	7			4	3 0	
RRCM.A	0	0	0	0	n – 1	0	0	0	1	0	0	dst	`RRCM.A #n,Rdst`
RRAM.A	0	0	0	0	n – 1	0	1	0	1	0	0	dst	`RRAM.A #n,Rdst`
RLAM.A	0	0	0	0	n – 1	1	0	0	1	0	0	dst	`RLAM.A #n,Rdst`
RRUM.A	0	0	0	0	n – 1	1	1	0	1	0	0	dst	`RRUM.A #n,Rdst`
RRCM.W	0	0	0	0	n – 1	0	0	0	1	0	1	dst	`RRCM.W #n,Rdst`
RRAM.W	0	0	0	0	n – 1	0	1	0	1	0	1	dst	`RRAM.W #n,Rdst`
RLAM.W	0	0	0	0	n – 1	1	0	0	1	0	1	dst	`RLAM.W #n,Rdst`
RRUM.W	0	0	0	0	n – 1	1	1	0	1	0	1	dst	`RRUM.W #n,Rdst`

Instruction	Instruction Identifier												dst				
	15			12	11			8	7	6	5	4	3			0	
RETI	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	
CALLA	0	0	0	1	0	0	1	1	0	1	0	0	dst				`CALLA Rdst`
	0	0	0	1	0	0	1	1	0	1	0	1	dst				`CALLA x(Rdst)`
	x.15:0																
	0	0	0	1	0	0	1	1	0	1	1	0	dst				`CALLA @Rdst`
	0	0	0	1	0	0	1	1	0	1	1	1	dst				`CALLA @Rdst+`
	0	0	0	1	0	0	1	1	1	0	0	0	&abs.19:16				`CALLA &abs20`
	&abs.15:0																
	0	0	0	1	0	0	1	1	1	0	0	1	x.19:16				`CALLA EDE`
	x.15:0																`CALLA x(PC)`
	0	0	0	1	0	0	1	1	1	0	1	1	imm.19:16				`CALLA #imm20`
	imm.15:0																
Reserved	0	0	0	1	0	0	1	1	1	0	1	0	x	x	x	x	
Reserved	0	0	0	1	0	0	1	1	1	1	x	x	x	x	x	x	
PUSHM.A	0	0	0	1	0	1	0	0	n – 1				dst				`PUSHM.A #n,Rdst`
PUSHM.W	0	0	0	1	0	1	0	1	n – 1				dst				`PUSHM.W #n,Rdst`
POPM.A	0	0	0	1	0	1	1	0	n – 1				dst – n + 1				`POPM.A #n,Rdst`
POPM.W	0	0	0	1	0	1	1	1	n – 1				dst – n + 1				`POPM.W #n,Rdst`

5.6.2 MPS430 Instructions

The MSP430 instructions are listed and described on the following pages.

*** 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 word to destination word

ADD.B Add source byte to destination byte

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 previous content of the destination is lost.

Status Bits

N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)

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 the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise

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

Example Ten is added to the 16-bit counter CNTR located in lower 64 K.

```
ADD.W   #10,&CNTR      ; Add 10 to 16-bit counter
```

Example A table word pointed to by R5 (20-bit address in R5) is added to R6. The jump to label TONI is performed on a carry.

```
ADD.W    @R5,R6        ; Add table word to R6. R6.19:16 = 0
JC       TONI          ; Jump if carry
...                    ; No carry
```

Example A table byte pointed to by R5 (20-bit address) is added to R6. The jump to label TONI is performed if no carry occurs. The table pointer is auto-incremented by 1. R6.19:8 = 0

```
ADD.B    @R5+,R6       ; Add byte to R6. R5 + 1. R6: 000xxh
JNC      TONI          ; Jump if no carry
...                    ; Carry occurred
```

ADDC[.W] Add source word and carry to destination word

ADDC.B Add source byte and carry to destination byte

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 previous content of the destination is lost.

Status Bits N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)

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 the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise

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

Example Constant value 15 and the carry of the previous instruction are added to the 16-bit counter CNTR located in lower 64 K.

```
ADDC.W    #15,&CNTR     ; Add 15 + C to 16-bit CNTR
```

Example A table word pointed to by R5 (20-bit address) and the carry C are added to R6. The jump to label TONI is performed on a carry. R6.19:16 = 0

```
ADDC.W    @R5,R6        ; Add table word + C to R6
JC        TONI          ; Jump if carry
...                     ; No carry
```

Example A table byte pointed to by R5 (20-bit address) and the carry bit C are added to R6. The jump to label TONI is performed if no carry occurs. The table pointer is auto-incremented by 1. R6.19:8 = 0

```
ADDC.B    @R5+,R6       ; Add table byte + C to R6. R5 + 1
JNC       TONI          ; Jump if no carry
...                     ; Carry occurred
```

AND[.W] Logical AND of source word with destination word

AND.B Logical AND of source byte with destination byte

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. The source operand is not affected.

Status Bits

N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)

Z: Set if result is zero, reset otherwise

C: Set if the result is not zero, reset otherwise. C = (.not. Z)

V: Reset

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

Example The bits set in R5 (16-bit data) are used as a mask (AA55h) for the word TOM located in the lower 64 K. If the result is zero, a branch is taken to label TONI. R5.19:16 = 0

```
MOV     #AA55h,R5      ; Load 16-bit mask to R5
AND     R5,&TOM        ; TOM .and. R5 -> TOM
JZ      TONI           ; Jump if result 0
...                    ; Result > 0
```

or shorter:

```
AND     #AA55h,&TOM    ; TOM .and. AA55h -> TOM
JZ      TONI           ; Jump if result 0
```

Example A table byte pointed to by R5 (20-bit address) is logically ANDed with R6. R5 is incremented by 1 after the fetching of the byte. R6.19:8 = 0

```
AND.B   @R5+,R6        ; AND table byte with R6. R5 + 1
```

BIC[.W] Clear bits set in source word in destination word

BIC.B Clear bits set in source byte in destination byte

Syntax `BIC src,dst` or `BIC.W src,dst`

`BIC.B src,dst`

Operation (.not. src) .and. dst → dst

Description The inverted source operand and the destination operand are logically ANDed. The result is placed into the destination. The source operand is not affected.

Status Bits

N:	Not affected
Z:	Not affected
C:	Not affected
V:	Not affected

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

Example The bits 15:14 of R5 (16-bit data) are cleared. R5.19:16 = 0

```
BIC     #0C000h,R5     ; Clear R5.19:14 bits
```

Example A table word pointed to by R5 (20-bit address) is used to clear bits in R7. R7.19:16 = 0

```
BIC.W   @R5,R7         ; Clear bits in R7 set in @R5
```

Example A table byte pointed to by R5 (20-bit address) is used to clear bits in Port1.

```
BIC.B   @R5,&P1OUT     ; Clear I/O port P1 bits set in @R5
```

BIS[.W] Set bits set in source word in destination word

BIS.B Set bits set in source byte in destination byte

Syntax `BIS src,dst` or `BIS.W src,dst`

`BIS.B src,dst`

Operation src .or. dst → dst

Description The source operand and the destination operand are logically ORed. The result is placed into the destination. The source operand is not affected.

Status Bits N: Not affected

Z: Not affected

C: Not affected

V: Not affected

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

Example Bits 15 and 13 of R5 (16-bit data) are set to one. R5.19:16 = 0

```
    BIS      #A000h,R5      ; Set R5 bits
```

Example A table word pointed to by R5 (20-bit address) is used to set bits in R7. R7.19:16 = 0

```
    BIS.W   @R5,R7          ; Set bits in R7
```

Example A table byte pointed to by R5 (20-bit address) is used to set bits in Port1. R5 is incremented by 1 afterwards.

```
    BIS.B   @R5+,&P1OUT     ; Set I/O port P1 bits. R5 + 1
```

BIT[.W] Test bits set in source word in destination word

BIT.B Test bits set in source byte in destination byte

Syntax `BIT src,dst` or `BIT.W src,dst`

`BIT.B src,dst`

Operation src .and. dst

Description The source operand and the destination operand are logically ANDed. The result affects only the status bits in SR.

Register mode: the register bits Rdst.19:16 (.W) resp. Rdst. 19:8 (.B) are not cleared!

Status Bits

N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)

Z: Set if result is zero, reset otherwise

C: Set if the result is not zero, reset otherwise. C = (.not. Z)

V: Reset

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

Example Test if one (or both) of bits 15 and 14 of R5 (16-bit data) is set. Jump to label TONI if this is the case. R5.19:16 are not affected.

```
BIT     #C000h,R5          ; Test R5.15:14 bits
JNZ     TONI               ; At least one bit is set in R5
...                        ; Both bits are reset
```

Example A table word pointed to by R5 (20-bit address) is used to test bits in R7. Jump to label TONI if at least one bit is set. R7.19:16 are not affected.

```
BIT.W   @R5,R7             ; Test bits in R7
JC      TONI               ; At least one bit is set
...                        ; Both are reset
```

Example A table byte pointed to by R5 (20-bit address) is used to test bits in output Port1. Jump to label TONI if no bit is set. The next table byte is addressed.

```
BIT.B   @R5+,&P1OUT        ; Test I/O port P1 bits. R5 + 1
JNC     TONI               ; No corresponding bit is set
...                        ; At least one bit is set
```

* BR, BRANCH

Branch to destination in lower 64K address space

Syntax `BR dst`

Operation dst → PC

Emulation `MOV dst,PC`

Description An unconditional branch is taken to an address anywhere in the lower 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 Call a subroutine in lower 64 K

Syntax `CALL dst`

Operation dst → PC 16-bit dst is evaluated and stored

SP – 2 → SP

PC → @SP updated PC with return address to TOS

tmp → PC saved 16-bit dst to PC

Description A subroutine call is made from an address in the lower 64 K to a subroutine address in the lower 64 K. All seven source addressing modes can be used. The call instruction is a word instruction. The return is made with the RET instruction.

Status Bits Status bits are not affected.
PC.19:16 cleared (address in lower 64 K)

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

Examples Examples for all addressing modes are given.

Immediate Mode: Call a subroutine at label EXEC (lower 64 K) or call directly to address.

```
CALL   #EXEC              ; Start address EXEC
CALL   #0AA04h            ; Start address 0AA04h
```

Symbolic Mode: Call a subroutine at the 16-bit address contained in address EXEC. EXEC is located at the address (PC + X) where X is within PC + 32 K.

```
CALL   EXEC               ; Start address at @EXEC. z16(PC)
```

Absolute Mode: Call a subroutine at the 16-bit address contained in absolute address EXEC in the lower 64 K.

```
CALL   &EXEC              ; Start address at @EXEC
```

Register mode: Call a subroutine at the 16-bit address contained in register R5.15:0.

```
CALL   R5                 ; Start address at R5
```

Indirect Mode: Call a subroutine at the 16-bit address contained in the word pointed to by register R5 (20-bit address).

```
CALL   @R5                ; Start address at @R5
```

*** CLR[.W]** Clear destination

*** CLR.B** Clear destination

Syntax `CLR dst` or `CLR.W dst`

`CLR.B dst`

Operation 0 → dst

Emulation `MOV #0,dst`
`MOV.B #0,dst`

Description The destination operand is cleared.

Status Bits Status bits are not affected.

Example RAM word TONI is cleared.

```
CLR     TONI      ; 0 -> TONI
```

Example Register R5 is cleared.

```
CLR     R5
```

Example RAM byte TONI is cleared.

```
CLR.B   TONI      ; 0 -> TONI
```

*** CLRC** Clear carry bit

Syntax `CLRC`

Operation 0 → C

Emulation `BIC #1,SR`

Description The carry bit (C) is cleared. The clear carry instruction is a word instruction.

Status Bits N: Not affected

Z: Not affected

C: Cleared

V: Not affected

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

Example The 16-bit decimal counter pointed to by R13 is added to a 32-bit counter pointed to by R12.

```
CLRC                    ; C=0: defines start
DADD   @R13,0(R12)      ; add 16-bit counter to low word of 32-bit counter
DADC   2(R12)           ; add carry to high word of 32-bit counter
```

*** CLRN** Clear negative bit

Syntax `CLRN`

Operation 0 → N

or

(.NOT.src .AND. dst → dst)

Emulation `BIC #4,SR`

Description The constant 04h is inverted (0FFFBh) and is logically ANDed with the destination operand. The result is placed into the destination. The clear negative bit instruction is a word instruction.

Status Bits

N: Reset to 0

Z: Not affected

C: Not affected

V: Not affected

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

Example The Negative bit in the status register is cleared. This avoids special treatment with negative numbers of the subroutine called.

```
            CLRN
            CALL   SUBR
            ......
            ......
SUBR        JN     SUBRET     ; If input is negative: do nothing and return
            ......
            ......
            ......
SUBRET      RET
```

*** CLRZ** Clear zero bit

Syntax `CLRZ`

Operation 0 → Z

or

(.NOT.src .AND. dst → dst)

Emulation `BIC #2,SR`

Description The constant 02h is inverted (0FFFDh) and logically ANDed with the destination operand. The result is placed into the destination. The clear zero bit instruction is a word instruction.

Status Bits

N:	Not affected
Z:	Reset to 0
C:	Not affected
V:	Not affected

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

Example The zero bit in the status register is cleared.

```
CLRZ
```

Indirect, Auto-Increment mode: Call a subroutine at the 16-bit address contained in the word pointed to by register R5 (20-bit address) and increment the 16-bit address in R5 afterwards by 2. The next time the software uses R5 as a pointer, it can alter the program execution due to access to the next word address in the table pointed to by R5.

```
CALL   @R5+          ; Start address at @R5. R5 + 2
```

Indexed mode: Call a subroutine at the 16-bit address contained in the 20-bit address pointed to by register (R5 + X), e.g. a table with addresses starting at X. The address is within the lower 64 KB. X is within +32 KB.

```
CALL   X(R5)         ; Start address at @(R5+X). z16(R5)
```

CMP[.W] Compare source word and destination word

CMP.B Compare source byte and destination byte

Syntax `CMP src,dst` or `CMP.W src,dst`

`CMP.B src,dst`

Operation (.not.src) + 1 + dst

or

dst – src

Emulation `BIC #2,SR`

Description The source operand is subtracted from the destination operand. This is made by adding the 1's complement of the source + 1 to the destination. The result affects only the status bits in SR.

Register mode: the register bits Rdst.19:16 (.W) resp. Rdst. 19:8 (.B) are not cleared.

Status Bits

N: Set if result is negative (src > dst), reset if positive (src = dst)

Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst)

C: Set if there is a carry from the MSB, reset otherwise

V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow).

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

Example Compare word EDE with a 16-bit constant 1800h. Jump to label TONI if EDE equals the constant. The address of EDE is within PC + 32 K.

```
CMP      #01800h,EDE       ; Compare word EDE with 1800h
JEQ      TONI              ; EDE contains 1800h
...                        ; Not equal
```

Example A table word pointed to by (R5 + 10) is compared with R7. Jump to label TONI if R7 contains a lower, signed 16-bit number. R7.19:16 is not cleared. The address of the source operand is a 20-bit address in full memory range.

```
CMP.W    10(R5),R7         ; Compare two signed numbers
JL       TONI              ; R7 < 10(R5)
...                        ; R7 >= 10(R5)
```

Example A table byte pointed to by R5 (20-bit address) is compared to the value in output Port1. Jump to label TONI if values are equal. The next table byte is addressed.

```
CMP.B    @R5+,&P1OUT       ; Compare P1 bits with table. R5 + 1
JEQ      TONI              ; Equal contents
...                        ; Not equal
```

*** DADC[.W]** Add carry decimally to destination

*** DADC.B** Add carry decimally to destination

Syntax `DADC dst` or `DADC.W 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]** Add source word and carry decimally to destination word

*** DADD.B** Add source byte and carry decimally to destination byte

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 two (.B) or four (.W) 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 content of the destination is lost. The result is not defined for non-BCD numbers.

Status Bits N: Set if MSB of result is 1 (word > 7999h, byte > 79h), reset if MSB is 0.

Z: Set if result is zero, reset otherwise

C: Set if the BCD result is too large (word > 9999h, byte > 99h), reset otherwise

V: Undefined

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

Example Decimal 10 is added to the 16-bit BCD counter DECCNTR.

```
DADD   #10h,&DECCNTR   ; Add 10 to 4-digit BCD counter
```

Example The eight-digit BCD number contained in 16-bit RAM addresses BCD and BCD+2 is added decimally to an eight-digit BCD number contained in R4 and R5 (BCD+2 and R5 contain the MSDs). The carry C is added, and cleared.

```
CLRC                    ; Clear carry
DADD.W   &BCD,R4        ; Add LSDs. R4.19:16 = 0
DADD.W   &BCD+2,R5      ; Add MSDs with carry. R5.19:16 = 0
JC       OVERFLOW       ; Result >9999,9999: go to error routine
...                     ; Result ok
```

Example The two-digit BCD number contained in word BCD (16-bit address) is added decimally to a two-digit BCD number contained in R4. The carry C is added, also. R4.19:8 = 0CLRC ; Clear carryDADD.B &BCD,R4 ; Add BCD to R4 decimally. R4: 0,00ddh

```
CLRC                    ; Clear carry
DADD.B   &BCD,R4        ; Add BCD to R4 decimally.
                          R4: 0,00ddh
```

*** 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`

`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     #510,R10
L$1     MOV     @R6+,TONI-EDE-1(R6)
        DEC     R10
        JNZ     L$1
```

Do not transfer tables using the routine above with the overlap shown in Figure 5-36.

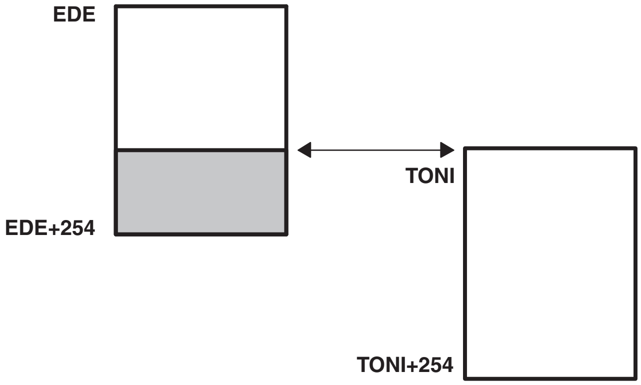


Figure 5-36. Decrement Overlap

*** 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`

`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 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-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, DINT should be executed at least one instruction before the beginning of the uninterruptible sequence, or it 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.

```
         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 enabled.

*** INC[.W]** Increment destination

*** INC.B** Increment destination

Syntax `INC dst` or `INC.W dst`

`INC.B dst`

Operation dst + 1 → dst

Emulation `ADD #1,dst`

Description The destination operand is incremented by one. The original contents are lost.

Status Bits N: Set if result is negative, reset if positive

Z: Set if dst contained 0FFFFh, reset otherwise

Set if dst contained 0FFh, reset otherwise

C: Set if dst contained 0FFFFh, reset otherwise

Set if dst contained 0FFh, reset otherwise

V: Set if dst contained 07FFFh, reset otherwise

Set if dst contained 07Fh, reset otherwise

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

Example The status byte, 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`

Description 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` or `INV.W dst`

`INV.B dst`

Operation .not.dst → dst

Emulation `XOR #0FFFFh,dst`

`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)
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

JHS Jump if higher or same (unsigned)

Syntax `JC label`

`JHS label`

Operation If C = 1: PC + (2 × Offset) → PC
If C = 0: execute the following instruction

Description The carry bit C in the status register is tested. If it is set, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit program counter PC. This means a jump in the range –511 to +512 words relative to the PC in the full memory range. If C is reset, the instruction after the jump is executed.
JC is used for the test of the carry bit C.
JHS is used for the comparison of unsigned numbers.

Status Bits Status bits are not affected

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

Example The state of the port 1 pin P1IN.1 bit defines the program flow.

```
BIT.B   #2,&P1IN        ; Port 1, bit 1 set? Bit -> C
JC      Label1          ; Yes, proceed at Label1
...                     ; No, continue
```

Example If R5 ≥ R6 (unsigned) the program continues at Label2

```
CMP     R6,R 5          ; Is R5 >= R6? Info to C
JHS     Label2          ; Yes, C = 1
...                     ; No, R5 < R6. Continue
```

Example If R5 ≥ 12345h (unsigned operands) the program continues at Label2

```
CMPA    #12345h,R5      ; Is R5 >= 12345h? Info to C
JHS     Label2          ; Yes, 12344h < R5 <= F,FFFFh. C = 1
...                     ; No, R5 < 12345h. Continue
```

JEQ Jump if equal

JZ Jump if zero

Syntax `JEQ label`

`JZ label`

Operation If Z = 1: PC + (2 × Offset) → PC
If Z = 0: execute following instruction

Description The zero bit Z in the status register is tested. If it is set, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit program counter PC. This means a jump in the range –511 to +512 words relative to the PC in the full memory range. If Z is reset, the instruction after the jump is executed.
JZ is used for the test of the zero bit Z.
JEQ is used for the comparison of operands.

Status Bits Status bits are not affected

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

Example The state of the P2IN.0 bit defines the program flow.

```
BIT.B   #1,&P2IN      ; Port 2, bit 0 reset?
JZ      Label1        ; Yes, proceed at Label1
...                   ; No, set, continue
```

Example If R5 = 15000h (20-bit data) the program continues at Label2.

```
CMPA    #15000h,R5    ; Is R5 = 15000h? Info to SR
JEQ     Label2        ; Yes, R5 = 15000h. Z = 1
...                   ; No, R5 not equal 15000h. Continue
```

Example R7 (20-bit counter) is incremented. If its content is zero, the program continues at Label4.

```
ADDA    #1,R7         ; Increment R7
JZ      Label4        ; Zero reached: Go to Label4
...                   ; R7 not equal 0. Continue here.
```

JGE Jump if greater or equal (signed)

Syntax `JGE label`

Operation If (N .xor. V) = 0: PC + (2 × Offset) → PC
If (N .xor. V) = 1: execute following instruction

Description The negative bit N and the overflow bit V in the status register are tested. If both bits are set or both are reset, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit program counter PC. This means a jump in the range -511 to +512 words relative to the PC in full Memory range. If only one bit is set, the instruction after the jump is executed.
JGE is used for the comparison of signed operands: also for incorrect results due to overflow, the decision made by the JGE instruction is correct.
Note that JGE emulates the non-implemented JP (jump if positive) instruction if used after the instructions AND, BIT, RRA, SXTX and TST. These instructions clear the V bit.

Status Bits Status bits are not affected

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

Example If byte EDE (lower 64 K) contains positive data, go to Label1. Software can run in the full memory range.

```
TST.B   &EDE            ; Is EDE positive? V <- 0
JGE     Label1          ; Yes, JGE emulates JP
...                     ; No, 80h <= EDE <= FFh
```

Example If the content of R6 is greater than or equal to the memory pointed to by R7, the program continues a Label5. Signed data. Data and program in full memory range.

```
CMP     @R7,R6          ; Is R6 >= @R7?
JGE     Label5          ; Yes, go to Label5
...                     ; No, continue here
```

Example If R5 ≥ 12345h (signed operands) the program continues at Label2. Program in full memory range.

```
CMPA    #12345h,R5      ; Is R5 >= 12345h?
JGE     Label2          ; Yes, 12344h < R5 <= 7FFFFh
...                     ; No, 80000h <= R5 < 12345h
```

JL Jump if greater or equal (signed)

Syntax `JL label`

Operation If (N .xor. V) = 1: PC + (2 × Offset) → PC
If (N .xor. V) = 0: execute following instruction

Description The negative bit N and the overflow bit V in the status register are tested. If only one is set, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit program counter PC. This means a jump in the range –511 to +512 words relative to the PC in full memory range. If both bits N and V are set or both are reset, the instruction after the jump is executed.
JL is used for the comparison of signed operands: also for incorrect results due to overflow, the decision made by the JL instruction is correct.

Status Bits Status bits are not affected

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

Example If byte EDE contains a smaller, signed operand than byte TONI, continue at Label1. The address EDE is within PC ± 32 K.

```
CMP.B   &TONI,EDE     ; Is EDE < TONI
JL      Label1        ; Yes
...                   ; No, TONI <= EDE
```

Example If the signed content of R6 is less than the memory pointed to by R7 (20-bit address) the program continues at Label Label5. Data and program in full memory range.

```
CMP     @R7,R6        ; Is R6 < @R7?
JL      Label5        ; Yes, go to Label5
...                   ; No, continue here
```

Example If R5 < 12345h (signed operands) the program continues at Label2. Data and program in full memory range.

```
CMPA    #12345h,R5    ; Is R5 < 12345h?
JL      Label2        ; Yes, 80000h =< R5 < 12345h
...                   ; No, 12344h < R5 <= 7FFFFh
```

JMP Jump unconditionally

Syntax `JMP label`

Operation PC + (2 × Offset) → PC

Description The signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit program counter PC. This means an unconditional jump in the range -511 to +512 words relative to the PC in the full memory. The JMP instruction may be used as a BR or BRA instruction within its limited range relative to the program counter.

Status Bits Status bits are not affected

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

Example The byte STATUS is set to 10. Then a jump to label MAINLOOP is made. Data in lower 64 K, program in full memory range.

```
MOV.B   #10,&STATUS     ; Set STATUS to 10
JMP     MAINLOOP        ; Go to main loop
```

Example The interrupt vector TAIV of Timer_A3 is read and used for the program flow. Program in full memory range, but interrupt handlers always starts in lower 64K.

```
ADD     &TAIV,PC        ; Add Timer_A interrupt vector to PC
RETI                    ; No Timer_A interrupt pending
JMP     IHCCR1          ; Timer block 1 caused interrupt
JMP     IHCCR2          ; Timer block 2 caused interrupt
RETI                    ; No legal interrupt, return
```

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 in the status register is tested. If it is set, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit program counter PC. This means a jump in the range -511 to +512 words relative to the PC in the full memory range. If N is reset, the instruction after the jump is executed.

Status Bits Status bits are not affected

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

Example The byte COUNT is tested. If it is negative, program execution continues at Label0. Data in lower 64 K, program in full memory range.

```
TST.B   &COUNT     ; Is byte COUNT negative?
JN      Label0     ; Yes, proceed at Label0
...                ; COUNT >= 0
```

Example R6 is subtracted from R5. If the result is negative, program continues at Label2. Program in full memory range.

```
SUB     R6,R5      ; R5 - R6 -> R5
JN      Label2     ; R5 is negative: R6 > R5 (N = 1)
...                ; R5 >= 0. Continue here.
```

Example R7 (20-bit counter) is decremented. If its content is below zero, the program continues at Label4. Program in full memory range.

```
SUBA    #1,R7      ; Decrement R7
JN      Label4     ; R7 < 0: Go to Label4
...                ; R7 >= 0. Continue here.
```

JNC Jump if no carry

JLO Jump if lower (unsigned)

Syntax `JNC label`

`JLO label`

Operation If C = 0: PC + (2 × Offset) → PC
If C = 1: execute following instruction

Description The carry bit C in the status register is tested. If it is reset, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit program counter PC. This means a jump in the range –511 to +512 words relative to the PC in the full memory range. If C is set, the instruction after the jump is executed.
JNC is used for the test of the carry bit C.
JLO is used for the comparison of unsigned numbers.

Status Bits Status bits are not affected

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

Example If byte EDE < 15 the program continues at Label2. Unsigned data. Data in lower 64 K, program in full memory range.

```
CMP.B   #15,&EDE     ; Is EDE < 15? Info to C
JLO     Label2       ; Yes, EDE < 15. C = 0
...                  ; No, EDE >= 15. Continue
```

Example The word TONI is added to R5. If no carry occurs, continue at Label0. The address of TONI is within PC ± 32 K.

```
ADD     TONI,R5      ; TONI + R5 -> R5. Carry -> C
JNC     Label0       ; No carry
...                  ; Carry = 1: continue here
```

JNZ Jump if not zero

JNE Jump if not equal

Syntax `JNZ label`

`JNE label`

Operation If Z = 0: PC + (2 × Offset) → PC
If Z = 1: execute following instruction

Description The zero bit Z in the status register is tested. If it is reset, the signed 10-bit word offset contained in the instruction is multiplied by two, sign extended, and added to the 20-bit program counter PC. This means a jump in the range –511 to +512 words relative to the PC in the full memory range. If Z is set, the instruction after the jump is executed.
JNZ is used for the test of the zero bit Z.
JNE is used for the comparison of operands.

Status Bits Status bits are not affected

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

Example The byte STATUS is tested. If it is not zero, the program continues at Label3. The address of STATUS is within PC ± 32 K.

```
TST.B   STATUS         ; Is STATUS = 0?
JNZ     Label3         ; No, proceed at Label3
...                    ; Yes, continue here
```

Example If word EDE ≠ 1500 the program continues at Label2. Data in lower 64 K, program in full memory range.

```
CMP     #1500,&EDE     ; Is EDE = 1500? Info to SR
JNE     Label2         ; No, EDE not equal 1500.
...                    ; Yes, R5 = 1500. Continue
```

Example R7 (20-bit counter) is decremented. If its content is not zero, the program continues at Label4. Program in full memory range.

```
SUBA     #1,R7         ; Decrement R7
JNZ      Label4        ; Zero not reached: Go to Label4
...                    ; Yes, R7 = 0. Continue here.
```

MOV[.W] Move source word to destination word

MOV.B Move source byte to destination byte

Syntax `MOV src,dst` or `MOV.W src,dst`

`MOV.B src,dst`

Operation src → dst

Description The source operand is copied to the destination. The source operand is not affected.

Status Bits
N: Not affected
Z: Not affected
C: Not affected
V: Not affected

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

Example Move a 16-bit constant 1800h to absolute address-word EDE (lower 64 K).

```
MOV     #01800h,&EDE            ; Move 1800h to EDE
```

Example The contents of table EDE (word data, 16-bit addresses) are copied to table TOM. The length of the tables is 030h words. Both tables reside in the lower 64K.

```
        MOV     #EDE,R10                ; Prepare pointer (16-bit address)
Loop    MOV     @R10+,TOM-EDE-2(R10)    ; R10 points to both tables.
                                        ; R10+2
        CMP     #EDE+60h,R10            ; End of table reached?
        JLO     Loop                    ; Not yet
        ...                             ; Copy completed
```

Example The contents of table EDE (byte data, 16-bit addresses) are copied to table TOM. The length of the tables is 020h bytes. Both tables may reside in full memory range, but must be within R10 ± 32 K.

```
        MOVA    #EDE,R10                ; Prepare pointer (20-bit)
        MOV     #20h,R9                 ; Prepare counter
Loop    MOV.B   @R10+,TOM-EDE-1(R10)    ; R10 points to both tables.
                                        ; R10+1
        DEC     R9                      ; Decrement counter
        JNZ     Loop                    ; Not yet done
        ...                             ; Copy 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.

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

*** POP.B** Pop byte from stack to destination

Syntax `POP dst`

`POP.B dst`

Operation @SP → temp
SP + 2 → SP
temp → dst

Emulation `MOV @SP+,dst` or `MOV.W @SP+,dst`

`MOV.B @SP+,dst`

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

Status Bits Status bits are not affected.

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

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

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

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

Example The contents of R7 is restored from the stack.

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

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

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

PUSH.B Save a byte on the stack

Syntax `PUSH dst` or `PUSH.W dst`

`PUSH.B dst`

Operation SP – 2 → SP
dst → @SP

Description The 20-bit stack pointer SP is decremented by two. The operand is then copied to the RAM word addressed by the SP. A pushed byte is stored in the low byte, the high byte is not affected.

Status Bits Status bits are not affected.

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

Example Save the two 16-bit registers R9 and R10 on the stack.

```
PUSH     R9       ; Save R9 and R10 XXXXh
PUSH     R10      ; YYYYh
```

Example Save the two bytes EDE and TONI on the stack. The addresses EDE and TONI are within PC ± 32 K.

```
PUSH.B   EDE      ; Save EDE   xxXXh
PUSH.B   TONI     ; Save TONI   xxYYh
```

RET Return from subroutine

Syntax RET

Operation @SP →PC.15:0 Saved PC to PC.15:0. PC.19:16 ← 0
SP + 2 → SP

Description The 16-bit return address (lower 64 K), pushed onto the stack by a CALL instruction is restored to the PC. The program continues at the address following the subroutine call. The four MSBs of the program counter PC.19:16 are cleared.

Status Bits Status bits are not affected.
PC.19:16: Cleared

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

Example Call a subroutine SUBR in the lower 64 K and return to the address in the lower 64K after the CALL.

```
        CALL    #SUBR       ; Call subroutine starting at SUBR
        ...                 ; Return by RET to here
SUBR    PUSH    R14         ; Save R14 (16 bit data)
        ...                 ; Subroutine code
        POP     R14         ; Restore R14
        RET                 ; Return to lower 64 K
```

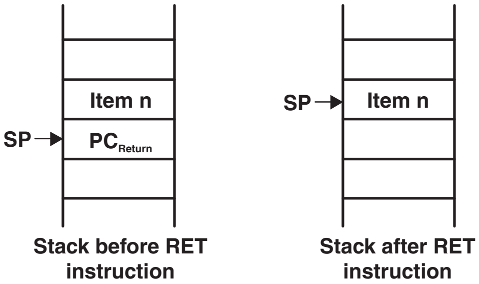


Figure 5-37. Stack After a RET Instruction

RETI	Return from interrupt	
Syntax	`RETI`	
Operation	@SP → SR.15:0 SP + 2 → SP	Restore saved status register SR with PC.19:16
	@SP → PC.15:0 SP + 2 → SP	Restore saved program counter PC.15:0 House keeping

Description The status register is restored to the value at the beginning of the interrupt service routine. This includes the four MSBs of the program counter PC.19:16. The stack pointer is incremented by two afterward.

The 20-bit PC is restored from PC.19:16 (from same stack location as the status bits) and PC.15:0. The 20-bit program counter is restored to the value at the beginning of the interrupt service routine. The program continues at the address following the last executed instruction when the interrupt was granted. The stack pointer is incremented by two afterward.

Status Bits

N:	Restored from stack
C:	Restored from stack
Z:	Restored from stack
V:	Restored from stack

Mode Bits OSCOFF, CPUOFF, and GIE are restored from stack.

Example Interrupt handler in the lower 64 K. A 20-bit return address is stored on the stack.

```
INTRPT  PUSHM.A   #2,R14    ; Save R14 and R13 (20-bit data)
        ...                 ; Interrupt handler code
        POPM.A    #2,R14    ; Restore R13 and R14 (20-bit data)
        RETI                ; Return to 20-bit address in full memory range
```

*** 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 5-38. 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.

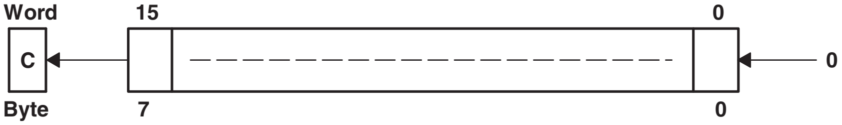


Figure 5-38. Destination Operand—Arithmetic Shift Left

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  (x 2)
```

Example The low byte of R7 is multiplied by 4.

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

Note: RLA Substitution

The assembler does not recognize the instructions:

```
RLA   @R5+           RLA.B  @R5+           RLA(.B) @R5
```

They must be substituted by:

```
ADD   @R5+,-2(R5)    ADD.B  @R5+,-1(R5)    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 5-39. The carry bit (C) is shifted into the LSB and the MSB is shifted into the carry bit (C).

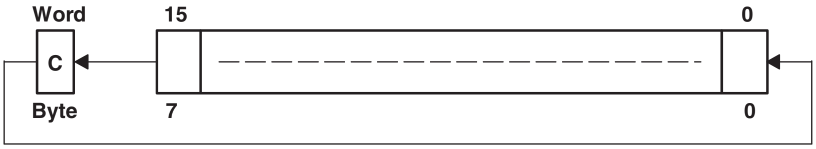


Figure 5-39. Destination Operand—Carry Left Shift

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: RLA Substitution

The assembler does not recognize the instructions:

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

They must be substituted by:

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

RRA[.W] Rotate right arithmetically destination word

RRA.B Rotate right arithmetically destination byte

Syntax `RRA.B dst` or `RRA.W dst`

Operation MSB → MSB → MSB–1 → ... LSB+1 → LSB → C

Description The destination operand is shifted right arithmetically by one bit position as shown in Figure 5-40. The MSB retains its value (sign). RRA operates equal to a signed division by 2. The MSB is retained and shifted into the MSB–1. The LSB+1 is shifted into the LSB. The previous LSB is shifted into the carry bit C.

Status Bits

N:	Set if result is negative (MSB = 1), reset otherwise (MSB = 0)
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 The signed 16-bit number in R5 is shifted arithmetically right one position.

```
RRA     R5              ; R5/2 -> R5
```

Example The signed RAM byte EDE is shifted arithmetically right one position.

```
RRA.B   EDE             ; EDE/2 -> EDE
```

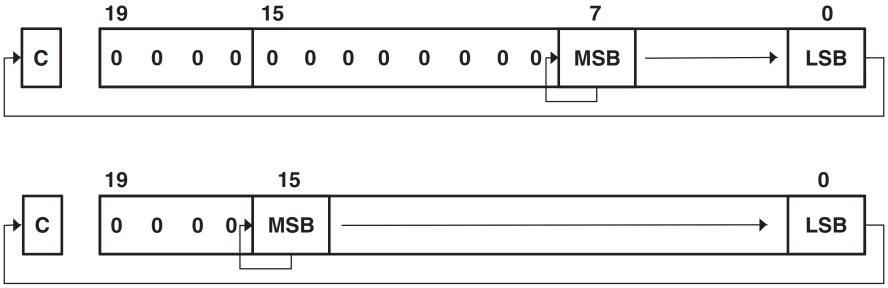


Figure 5-40. Rotate Right Arithmetically RRA.B and RRA.W

RRC[.W] Rotate right through carry destination word

RRC.B Rotate right through carry destination byte

Syntax `RRC dst` or `RRC.W dst`

`RRC.B dst`

Operation C → MSB → MSB–1 → ... LSB+1 → LSB → C

Description The destination operand is shifted right by one bit position as shown in Figure 5-41. The carry bit C is shifted into the MSB and the LSB is shifted into the carry bit C.

Status Bits N: Set if result is negative (MSB = 1), reset otherwise (MSB = 0)

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 RAM word EDE is shifted right one bit position. The MSB is loaded with 1.

```
SETC                ; Prepare carry for MSB
RRC     EDE         ; EDE = EDE >> 1 + 8000h
```

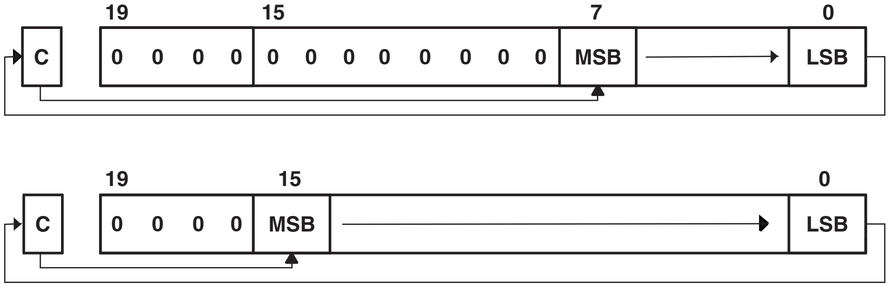


Figure 5-41. Rotate Right Through Carry RRC.B and RRC.W

*** SBC[.W]** Subtract source and borrow/.NOT. carry from destination

*** SBC.B** Subtract source and borrow/.NOT. carry from destination

Syntax `SBC dst` or `SBC.W dst`

`SBC.B dst`

Operation dst + 0FFFFh + C → dst

dst + 0FFh + C → dst

Emulation `SUBC #0,dst`

`SUBC.B #0,dst`

Description The carry bit (C) is added to the destination operand minus one. The previous contents of the destination are lost.

Status Bits N: Set if result is negative, reset if positive

Z: Set if result is zero, reset otherwise

C: Set if there is a carry from the MSB of the result, reset otherwise

Set to 1 if no borrow, reset if borrow

V: Set if an arithmetic overflow occurs, reset otherwise

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

Example The 16-bit counter pointed to by R13 is subtracted from a 32-bit counter pointed to by R12.

```
SUB     @R13,0(R12)     ; Subtract LSDs
SBC     2(R12)          ; Subtract carry from MSD
```

Example The 8-bit counter pointed to by R13 is subtracted from a 16-bit counter pointed to by R12.

```
SUB.B   @R13,0(R12)     ; Subtract LSDs
SBC.B   1(R12)          ; Subtract carry from MSD
```

Note: Borrow Implementation

The borrow is treated as a .NOT. carry:

Borrow	Carry Bit
Yes	0
No	1

*** SETC** Set carry bit

Syntax `SETC`

Operation 1 → C

Emulation `BIS #1,SR`

Description The carry bit (C) is set.

Status Bits
N: Not affected
Z: Not affected
C: Set
V: Not affected

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

Example Emulation of the decimal subtraction:

Subtract R5 from R6 decimally.

Assume that R5 = 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 carry bit
Syntax	`SETN`
Operation	1 → N
Emulation	`BIS #4,SR`
Description	The negative bit (N) is set.
Status Bits	N: Set
	Z: Not affected
	C: Not affected
	V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.

*** SETZ** Set zero bit

Syntax `SETZ`

Operation 1 → N

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 word from destination word

SUB.B Subtract source byte from destination byte

Syntax `SUB src,dst` or `SUB.W src,dst`

`SUB.B src,dst`

Operation (.not.src) + 1 + dst → dst or dst – src → dst

Description The source operand is subtracted from the destination operand. This is made by adding the 1's complement of the source + 1 to the destination. The source operand is not affected, the result is written to the destination operand.

Status Bits

N: Set if result is negative (src > dst), reset if positive (src ≤ dst)

Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst)

C: Set if there is a carry from the MSB, reset otherwise

V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow).

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

Example A 16-bit constant 7654h is subtracted from RAM word EDE.

```
SUB     #7654h,&EDE     ; Subtract 7654h from EDE
```

Example A table word pointed to by R5 (20-bit address) is subtracted from R7. Afterwards, if R7 contains zero, jump to label TONI. R5 is then auto-incremented by 2. R7.19:16 = 0.

```
SUB     @R5+,R7        ; Subtract table number from R7. R5 + 2
JZ      TONI           ; R7 = @R5 (before subtraction)
...                    ; R7 <> @R5 (before subtraction)
```

Example Byte CNT is subtracted from byte R12 points to. The address of CNT is within PC ± 32 K. The address R12 points to is in full memory range.

```
SUB.B   CNT,0(R12)     ; Subtract CNT from @R12
```

SUBC[.W] Subtract source word with carry from destination word

SUBC.B Subtract source byte with carry from destination byte

Syntax `SUBC src,dst` or `SUBC.W src,dst`

`SUBC.B src,dst`

Operation (.not.src) + C + dst → dst or dst – (src – 1) + C → dst

Description The source operand is subtracted from the destination operand. This is done by adding the 1's complement of the source + carry to the destination. The source operand is not affected, the result is written to the destination operand. Used for 32, 48, and 64-bit operands.

Status Bits N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)

Z: Set if result is zero, reset otherwise

C: Set if there is a carry from the MSB, reset otherwise

V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)

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

Example A 16-bit constant 7654h is subtracted from R5 with the carry from the previous instruction. R5.19:16 = 0

```
SUBC.W   #7654h,R5       ; Subtract 7654h + C from R5
```

Example A 48-bit number (3 words) pointed to by R5 (20-bit address) is subtracted from a 48-bit counter in RAM, pointed to by R7. R5 points to the next 48-bit number afterwards. The address R7 points to is in full memory range.

```
SUB      @R5+,0(R7)      ; Subtract LSBs. R5 + 2
SUBC     @R5+,2(R7)      ; Subtract MIDs with C. R5 + 2
SUBC     @R5+,4(R7)      ; Subtract MSBs with C. R5 + 2
```

Example Byte CNT is subtracted from the byte, R12 points to. The carry of the previous instruction is used. The address of CNT is in lower 64 K.

```
SUBC.B   &CNT,0(R12)     ; Subtract byte CNT from @R12
```

SWPB Swap bytes

Syntax `SWPB dst`

Operation dst.15:8 ↔ dst.7:0

Description The high and the low byte of the operand are exchanged. PC.19:16 bits are cleared in register mode.

Status Bits Status bits are not affected

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

Example Exchange the bytes of RAM word EDE (lower 64 K).

```
MOV     #1234h,&EDE       ; 1234h -> EDE
SWPB    &EDE              ; 3412h -> EDE
```

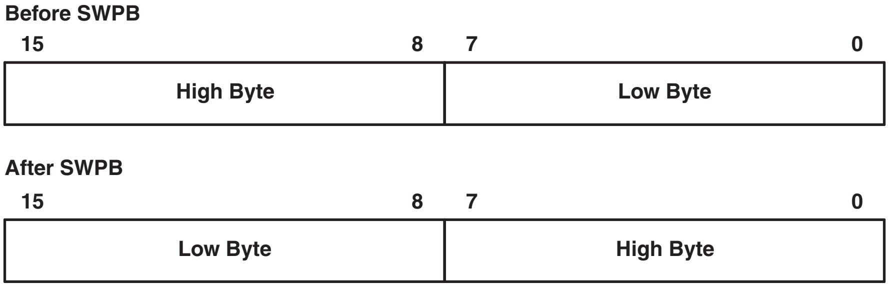


Figure 5-42. Swap Bytes in Memory

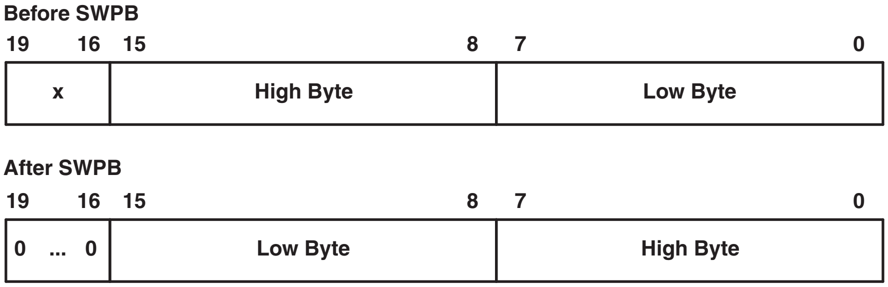


Figure 5-43. Swap Bytes in a Register

SXT Extend sign

Syntax `SXT dst`

Operation dst.7 → dst.15:8, dst.7 → dst.19:8 (register mode)

Description Register mode: the sign of the low byte of the operand is extended into the bits Rdst.19:8.

Rdst.7 = 0: Rdst.19:8 = 000h afterwards

Rdst.7 = 1: Rdst.19:8 = FFFh afterwards

Other Modes: the sign of the low byte of the operand is extended into the high byte.

dst.7 = 0: high byte = 00h afterwards

dst.7 = 1: high byte = FFh afterwards

Status Bits

N: Set if result is negative, reset otherwise

Z: Set if result is zero, reset otherwise

C: Set if result is not zero, reset otherwise (C = .not.Z)

V: Reset

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

Example The signed 8-bit data in EDE (lower 64 K) is sign extended and added to the 16-bit signed data in R7.

```
MOV.B   &EDE,R5       ; EDE -> R5. 00XXh
SXT     R5            ; Sign extend low byte to R5.19:8
ADD     R5,R7         ; Add signed 16-bit values
```

Example The signed 8-bit data in EDE (PC +32 K) is sign extended and added to the 20-bit data in R7.

```
MOV.B   EDE,R5        ; EDE -> R5. 00XXh
SXT     R5            ; Sign extend low byte to R5.19:8
ADDA    R5,R7         ; Add signed 20-bit values
```

*** 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 source word with destination word

XOR.B Exclusive OR source byte with destination byte

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 exclusively ORed. The result is placed into the destination. The source operand is not affected. The previous content of the destination is lost.

Status Bits N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)

Z: Set if result is zero, reset otherwise

C: Set if result is not zero, reset otherwise (C = .not. Z)

V: Set if both operands are negative before execution, reset otherwise

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

Example Toggle bits in word CNTR (16-bit data) with information (bit = 1) in address-word TONI. Both operands are located in lower 64 K.

```
XOR     &TONI,&CNTR     ; Toggle bits in CNTR
```

Example A table word pointed to by R5 (20-bit address) is used to toggle bits in R6. R6.19:16 = 0.

```
XOR     @R5,R6          ; Toggle bits in R6
```

Example Reset to zero those bits in the low byte of R7 that are different from the bits in byte EDE. R7.19:8 = 0. The address of EDE is within PC ± 32 K.

```
XOR.B   EDE,R7          ; Set different bits to 1 in R7.
INV.B   R7              ; Invert low byte of R7, high byte is 0h
```

5.6.3 Extended Instructions

The extended MSP430X instructions give the MSP430X CPU full access to its 20-bit address space. MSP430X instructions require an additional word of op-code called the extension word. All addresses, indexes, and immediate numbers have 20-bit values, when preceded by the extension word. The MSP430X extended instructions are listed and described in the following pages.

*** ADCX.A** Add carry to destination address-word

*** ADCX.[W]** Add carry to destination word

*** ADCX.B** Add carry to destination byte

Syntax `ADCX.A dst`

`ADCX dst` or `ADCX.W dst`

`ADCX.B dst`

Operation dst + C → dst

Emulation `ADDCX.A #0,dst`

`ADDCX #0,dst`

`ADDCX.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 (MSB = 1), reset if positive (MSB = 0)

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 the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise

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

Example The 40-bit counter, pointed to by R12 and R13, is incremented.

```
INCX.A   @R12     ; Increment lower 20 bits
ADCX.A   @R13     ; Add carry to upper 20 bits
```

ADDX.A Add source address-word to destination address-word

ADDX.[W] Add source word to destination word

ADDX.B Add source byte to destination byte

Syntax `ADDX.A src,dst`

`ADDX src,dst` or `ADDX.W src,dst`

`ADDX.B src,dst`

Operation src + dst → dst

Description The source operand is added to the destination operand. The previous contents of the destination are lost. Both operands can be located in the full address space.

Status Bits N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)

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 the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise

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

Example Ten is added to the 20-bit pointer CNTR located in two words CNTR (LSBs) and CNTR+2 (MSBs).

```
ADDX.A   #10,CNTR    ; Add 10 to 20-bit pointer
```

Example A table word (16-bit) pointed to by R5 (20-bit address) is added to R6. The jump to label TONI is performed on a carry.

```
ADDX.W   @R5,R6      ; Add table word to R6
JC       TONI        ; Jump if carry
...                  ; No carry
```

Example A table byte pointed to by R5 (20-bit address) is added to R6. The jump to label TONI is performed if no carry occurs. The table pointer is auto-incremented by 1.

```
ADDX.B   @R5+,R6     ; Add table byte to R6. R5 + 1. R6: 000xxh
JNC      TONI        ; Jump if no carry
...                  ; Carry occurred
```

Note: Use ADDA for the following two cases for better code density and execution.

```
ADDX.A   Rsrc,Rdst
ADDX.A   #imm20,Rdst
```

ADDCX.A Add source address-word and carry to destination address-word

ADDCX.[W] Add source word and carry to destination word

ADDCX.B Add source byte and carry to destination byte

Syntax `ADDCX.A src,dst`

`ADDCX src,dst` or `ADDCX.W src,dst`

`ADDCX.B src,dst`

Operation src + dst + C → dst

Description The source operand and the carry bit C are added to the destination operand. The previous contents of the destination are lost. Both operands may be located in the full address space.

Status Bits N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)

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 the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise

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

Example Constant 15 and the carry of the previous instruction are added to the 20-bit counter CNTR located in two words.

```
ADDCX.A   #15,&CNTR    ; Add 15 + C to 20-bit CNTR
```

Example A table word pointed to by R5 (20-bit address) and the carry C are added to R6. The jump to label TONI is performed on a carry.

```
ADDCX.W   @R5,R6       ; Add table word + C to R6
JC        TONI         ; Jump if carry
...                    ; No carry
```

Example A table byte pointed to by R5 (20-bit address) and the carry bit C are added to R6. The jump to label TONI is performed if no carry occurs. The table pointer is auto-incremented by 1.

```
ADDCX.B   @R5+,R6      ; Add table byte + C to R6. R5 + 1
JNC       TONI         ; Jump if no carry
...                    ; Carry occurred
```

ANDX.A Logical AND of source address-word with destination address-word

ANDX.[W] Logical AND of source word with destination word

ANDX.B Logical AND of source byte with destination byte

Syntax `ANDX.A src,dst`

`ANDX src,dst` or `ANDX.W src,dst`

`ANDX.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. The source operand is not affected. Both operands may be located in the full address space.

Status Bits N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)

Z: Set if result is zero, reset otherwise

C: Set if the result is not zero, reset otherwise. C = (.not. Z)

V: Reset

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

Example The bits set in R5 (20-bit data) are used as a mask (AAA55h) for the address-word TOM located in two words. If the result is zero, a branch is taken to label TONI.

```
MOVA     #AAA55h,R5      ; Load 20-bit mask to R5
ANDX.A   R5,TOM          ; TOM .and. R5 -> TOM
JZ       TONI            ; Jump if result 0
...                      ; Result > 0
```

or shorter:

```
ANDX.A   #AAA55h,TOM     ; TOM .and. AAA55h -> TOM
JZ       TONI            ; Jump if result 0
```

Example A table byte pointed to by R5 (20-bit address) is logically ANDed with R6. R6.19:8 = 0. The table pointer is auto-incremented by 1.

```
ANDX.B   @R5+,R6         ; AND table byte with R6. R5 + 1
```

BICX.A Clear bits set in source address-word in destination address-word

BICX.[W] Clear bits set in source word in destination word

BICX.B Clear bits set in source byte in destination byte

Syntax `BICX.A src,dst`

`BICX src,dst` or `BICX.W src,dst`

`BICX.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. Both operands may be located in the full address space.

Status Bits N: Not affected

Z: Not affected

C: Not affected

V: Not affected

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

Example The bits 19:15 of R5 (20-bit data) are cleared.

```
BICX.A   #0F8000h,R5     ; Clear R5.19:15 bits
```

Example A table word pointed to by R5 (20-bit address) is used to clear bits in R7. R7.19:16 = 0.

```
BICX.W   @R5,R7          ; Clear bits in R7
```

Example A table byte pointed to by R5 (20-bit address) is used to clear bits in output Port1.

```
BICX.B   @R5,&P1OUT      ; Clear I/O port P1 bits
```

BISX.A Set bits set in source address-word in destination address-word

BISX.[W] Set bits set in source word in destination word

BISX.B Set bits set in source byte in destination byte

Syntax `BISX.A src,dst`

`BISX src,dst` or `BISX.W src,dst`

`BISX.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. Both operands may be located in the full address space.

Status Bits N: Not affected

Z: Not affected

C: Not affected

V: Not affected

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

Example Bits 16 and 15 of R5 (20-bit data) are set to one.

```
BISX.A    #018000h,R5     ; Set R5.16:15 bits
```

Example A table word pointed to by R5 (20-bit address) is used to set bits in R7.

```
BISX.W   @R5,R7           ; Set bits in R7
```

Example A table byte pointed to by R5 (20-bit address) is used to set bits in output Port1.

```
BISX.B   @R5,&P1OUT       ; Set I/O port P1 bits
```

BITX.A Test bits set in source address-word in destination address-word

BITX.[W] Test bits set in source word in destination word

BITX.B Test bits set in source byte in destination byte

Syntax `BITX.A src,dst`

`BITX src,dst` or `BITX.W src,dst`

`BITX.B src,dst`

Operation src .and. dst → dst

Description The source operand and the destination operand are logically ANDed. The result affects only the status bits. Both operands may be located in the full address space.

Status Bits N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)

Z: Set if result is zero, reset otherwise

C: Set if the result is not zero, reset otherwise. C = (.not. Z)

V: Reset

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

Example Test if bit 16 or 15 of R5 (20-bit data) is set. Jump to label TONI if so.

```
BITX.A   #018000h,R5     ; Test R5.16:15 bits
JNZ      TONI            ; At least one bit is set
...                      ; Both are reset
```

Example A table word pointed to by R5 (20-bit address) is used to test bits in R7. Jump to label TONI if at least one bit is set.

```
BITX.W   @R5,R7           ; Test bits in R7: C = .not.Z
JC       TONI             ; At least one is set
...                       ; Both are reset
```

Example A table byte pointed to by R5 (20-bit address) is used to test bits in input Port1. Jump to label TONI if no bit is set. The next table byte is addressed.

```
BITX.B   @R5+,&P1IN       ; Test input P1 bits. R5 + 1
JNC      TONI             ; No corresponding input bit is set
...                       ; At least one bit is set
```

*** CLRX.A** Clear destination address-word

*** CLRX.[W]** Clear destination word

*** CLRX.B** Clear destination byte

Syntax `CLRX.A dst`

`CLRX dst` or `CLRX.W dst`

`CLRX.B dst`

Operation 0 → dst

Emulation `MOVX.A #0,dst`

`MOVX #0,dst`

`MOVX.B #0,dst`

Description The destination operand is cleared.

Status Bits Status bits are not affected.

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

Example RAM address-word TONI is cleared.

```
      CLRX.A   TONI   ; 0 -> TONI
```

CMPX.A Compare source address-word and destination address-word

CMPX.[W] Compare source word and destination word

CMPX.B Compare source byte and destination byte

Syntax `CMPX.A src,dst`

`CMPX src,dst` or `CMPX.W src,dst`

`CMPX.B src,dst`

Operation (.not. src) + 1 + dst or dst – src

Description The source operand is subtracted from the destination operand by adding the 1's complement of the source + 1 to the destination. The result affects only the status bits. Both operands may be located in the full address space.

Status Bits N: Set if result is negative (src > dst), reset if positive (src ≤ dst)

Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst)

C: Set if there is a carry from the MSB, reset otherwise

V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)

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

Example Compare EDE with a 20-bit constant 18000h. Jump to label TONI if EDE equals the constant.

```
CMPX.A   #018000h,EDE     ; Compare EDE with 18000h
JEQ      TONI             ; EDE contains 18000h
...                       ; Not equal
```

Example A table word pointed to by R5 (20-bit address) is compared with R7. Jump to label TONI if R7 contains a lower, signed, 16-bit number.

```
CMPX.W   @R5,R7           ; Compare two signed numbers
JL       TONI             ; R7 < @R5
...                       ; R7 >= @R5
```

Example A table byte pointed to by R5 (20-bit address) is compared to the input in I/O Port1. Jump to label TONI if the values are equal. The next table byte is addressed.

```
CMPX.B   @R5+,&P1IN       ; Compare P1 bits with table. R5 + 1
JEQ      TONI             ; Equal contents
...                       ; Not equal
```

Note: Use CMPA for the following two cases for better density and execution.

```
CMPA     Rsrc,Rdst
CMPA     #imm20,Rdst
```

*** DADCX.A** Add carry decimally to destination address-word

*** DADCX.[W]** Add carry decimally to destination word

*** DADCX.B** Add carry decimally to destination byte

Syntax `DADCX.A dst`

`DADCX dst` or `DADCX.W dst`

`DADCX.B dst`

Operation dst + C → dst (decimally)

Emulation `DADDX.A #0,dst`

`DADDX #0,dst`

`DADDX.B #0,dst`

Description The carry bit (C) is added decimally to the destination.

Status Bits N: Set if MSB of result is 1 (address-word > 79999h, word > 7999h, byte > 79h), reset if MSB is 0

Z: Set if result is zero, reset otherwise

C: Set if the BCD result is too large (address-word > 99999h, word > 9999h, byte > 99h), reset otherwise

V: Undefined

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

Example The 40-bit counter, pointed to by R12 and R13, is incremented decimally.

```
DADDX.A   #1,0(R12)     ; Increment lower 20 bits
DADCX.A   0(R13)        ; Add carry to upper 20 bits
```

DADDX.A Add source address-word and carry decimally to destination address-word

DADDX.[W] Add source word and carry decimally to destination word

DADDX.B Add source byte and carry decimally to destination byte

Syntax `DADDX.A src,dst`

`DADDX src,dst` or `DADDX.W src,dst`

`DADDX.B src,dst`

Operation src + dst + C → dst (decimally)

Description The source operand and the destination operand are treated as two (.B), four (.W), or five (.A) 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. Both operands may be located in the full address space.

Status Bits N: Set if MSB of result is 1 (address-word > 79999h, word > 7999h, byte > 79h), reset if MSB is 0.

Z: Set if result is zero, reset otherwise

C: Set if the BCD result is too large (address-word > 99999h, word > 9999h, byte > 99h), reset otherwise

V: Undefined

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

Example Decimal 10 is added to the 20-bit BCD counter DECCNTR located in two words.

```
DADDX.A   #10h,&DECCNTR     ; Add 10 to 20-bit BCD counter
```

Example The eight-digit BCD number contained in 20-bit addresses BCD and BCD+2 is added decimally to an eight-digit BCD number contained in R4 and R5 (BCD+2 and R5 contain the MSDs).

```
CLRC                        ; Clear carry
DADDX.W   BCD,R4            ; Add LSDs
DADDX.W   BCD+2,R5          ; Add MSDs with carry
JC        OVERFLOW          ; Result >99999999: go to error routine
...                         ;    Result ok
```

Example The two-digit BCD number contained in 20-bit address BCD is added decimally to a two-digit BCD number contained in R4.

```
CLRC                        ; Clear carry
DADDX.B   BCD,R4            ; Add BCD to R4 decimally.
                            ; R4: 000ddh
```

*** DECX.A** Decrement destination address-word

*** DECX.[W]** Decrement destination word

*** DECX.B** Decrement destination byte

Syntax `DECX.A dst`

`DECX dst` or `DECX.W dst`

`DECX.B dst`

Operation dst – 1 → dst

Emulation `SUBX.A #1,dst`

`SUBX #1,dst`

`SUBX.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

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

Example RAM address-word TONI is decremented by 1.

```
DECX.A   TONI     ; Decrement TONI
```

* **DECDX.A**	Double-decrement destination address-word	
* **DECDX.[W]**	Double-decrement destination word	
* **DECDX.B**	Double-decrement destination byte	
Syntax	`DECDX.A dst`	
	`DECDX dst` or `DECDX.W dst`	
	`DECDX.B dst`	
Operation	dst – 2 → dst	
Emulation	`SUBX.A #2,dst`	
	`SUBX #2,dst`	
	`SUBX.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
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.	
Example	RAM address-word TONI is decremented by 2.	

```
DECDX.A   TONI     ; Decrement TONI
```

*** INCX.A** Increment destination address-word

*** INCX.[W]** Increment destination word

*** INCX.B** Increment destination byte

Syntax `INCX.A dst`

`INCX dst` or `INCX.W dst`

`INCX.B dst`

Operation dst + 1 → dst

Emulation `ADDX.A #1,dst`

`ADDX #1,dst`

`ADDX.B #1,dst`

Description The destination operand is incremented by one. The original contents are lost.

Status Bits N: Set if result is negative, reset if positive

Z: Set if dst contained 0FFFFFh, reset otherwise

Set if dst contained 0FFFFh, reset otherwise

Set if dst contained 0FFh, reset otherwise

C: Set if dst contained 0FFFFFh, reset otherwise

Set if dst contained 0FFFFh, reset otherwise

Set if dst contained 0FFh, reset otherwise

V: Set if dst contained 07FFFh, reset otherwise

Set if dst contained 07FFFh, reset otherwise

Set if dst contained 07Fh, reset otherwise

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

Example RAM address-wordTONI is incremented by 1.

```
INCX.A   TONI     ; Increment TONI (20-bits)
```

*** INCDX.A** Double-increment destination address-word

*** INCDX.[W]** Double-increment destination word

*** INCDX.B** Double-increment destination byte

Syntax `INCDX.A dst`

`INCDX dst` or `INCDX.W dst`

`INCDX.B dst`

Operation dst + 2 → dst

Emulation `ADDX.A #2,dst`

`ADDX #2,dst`

`ADDX.B #2,dst`

Description 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 0FFFFEh, reset otherwise

Set if dst contained 0FFFEh, reset otherwise

Set if dst contained 0FEh, reset otherwise

C: Set if dst contained 0FFFFEh or 0FFFFFh, reset otherwise

Set if dst contained 0FFFEh or 0FFFFh, reset otherwise

Set if dst contained 0FEh or 0FFh, reset otherwise

V: Set if dst contained 07FFFEh or 07FFFFh, reset otherwise

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 RAM byte LEO is incremented by two; PC points to upper memory.

```
INCDX.B   LEO     ; Increment LEO by two
```

*** INVX.A** Invert destination

*** INVX.[W]** Invert destination

*** INVX.B** Invert destination

Syntax `INVX.A dst`

`INVX dst` or `INVX.W dst`

`INVX.B dst`

Operation .NOT.dst → dst

Emulation `XORX.A #0FFFFFh,dst`

`XORX #0FFFFh,dst`

`XORX.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 0FFFFFh, reset otherwise

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)

V: Set if initial destination operand was negative, otherwise reset

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

Example 20-bit content of R5 is negated (twos complement).

```
INVX.A   R5      ; Invert R5
INCX.A   R5      ; R5 is now negated
```

Example Content of memory byte LEO is negated. PC is pointing to upper memory.

```
INVX.B   LEO     ; Invert LEO
INCX.B   LEO     ; MEM(LEO) is negated
```

MOVX.A Move source address-word to destination address-word

MOVX.[W] Move source word to destination word

MOVX.B Move source byte to destination byte

Syntax `MOVX.A src,dst`

`MOVX src,dst` or `MOVX.W src,dst`

`MOVX.B src,dst`

Operation src → dst

Description The source operand is copied to the destination. The source operand is not affected. Both operands may be located in the full address space.

Status Bits N: Not affected

Z: Not affected

C: Not affected

V: Not affected

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

Example Move a 20-bit constant 18000h to absolute address-word EDE.

```
MOVX.A   #018000h,&EDE        ; Move 18000h to EDE
```

Example The contents of table EDE (word data, 20-bit addresses) are copied to table TOM. The length of the table is 030h words.

```
      MOVA     #EDE,R10               ; Prepare pointer (20-bit address)
Loop  MOVX.W   @R10+,TOM-EDE-2(R10)   ; R10 points to both tables.
                                      ; R10+2
      CMPA     #EDE+60h,R10           ; End of table reached?
      JLO      Loop                   ; Not yet
      ...                             ; Copy completed
```

Example The contents of table EDE (byte data, 20-bit addresses) are copied to table TOM. The length of the table is 020h bytes.

```
      MOVA     #EDE,R10               ; Prepare pointer (20-bit)
      MOV      #20h,R9                ; Prepare counter
Loop  MOVX.W   @R10+,TOM-EDE-2(R10)   ; R10 points to both tables.
                                      ; R10+1
      DEC      R9                     ; Decrement counter
      JNZ      Loop                   ; Not yet done
      ...                             ; Copy completed
```

Ten of the 28 possible addressing combinations of the MOVX.A instruction can use the MOVA instruction. This saves two bytes and code cycles. Examples for the addressing combinations are:

```
MOVX.A   Rsrc,Rdst          MOVA   Rsrc,Rdst         ; Reg/Reg
MOVX.A   #imm20,Rdst        MOVA   #imm20,Rdst       ; Immediate/Reg
MOVX.A   &abs20,Rdst        MOVA   &abs20,Rdst       ; Absolute/Reg
MOVX.A   @Rsrc,Rdst         MOVA   @Rsrc,Rdst        ; Indirect/Reg
MOVX.A   @Rsrc+,Rdst        MOVA   @Rsrc+,Rdst       ; Indirect,Auto/Reg
MOVX.A   Rsrc,&abs20        MOVA   Rsrc,&abs20       ; Reg/Absolute
```

The next four replacements are possible only if 16-bit indexes are sufficient for the addressing.

```
MOVX.A   z20(Rsrc),Rdst     MOVA   z16(Rsrc),Rdst   ; Indexed/Reg
MOVX.A   Rsrc,z20(Rdst)     MOVA   Rsrc,z16(Rdst)   ; Reg/Indexed
MOVX.A   symb20,Rdst        MOVA   symb16,Rdst      ; Symbolic/Reg
MOVX.A   Rsrc,symb20        MOVA   Rsrc,symb16      ; Reg/Symbolic
```

POPM.A Restore n CPU registers (20-bit data) from the stack

POPM.[W] Restore n CPU registers (16-bit data) from the stack

Syntax `POPM.A #n,Rdst` 1 ≤ n ≤ 16

`POPM.W #n,Rdst` or `POPM #n,Rdst` 1 ≤ n ≤ 16

Operation POPM.A: Restore the register values from stack to the specified CPU registers. The stack pointer SP is incremented by four for each register restored from stack. The 20-bit values from stack (2 words per register) are restored to the registers.

POPM.W: Restore the 16-bit register values from stack to the specified CPU registers. The stack pointer SP is incremented by two for each register restored from stack. The 16-bit values from stack (one word per register) are restored to the CPU registers.

Note : This instruction does not use the extension word.

Description POPM.A: The CPU registers pushed on the stack are moved to the extended CPU registers, starting with the CPU register (Rdst – n + 1). The stack pointer is incremented by (n y 4) after the operation.

POPM.W: The 16-bit registers pushed on the stack are moved back to the CPU registers, starting with CPU register (Rdst – n + 1). The stack pointer is incremented by (n y 2) after the instruction. The MSBs (Rdst.19:16) of the restored CPU registers are cleared.

Status Bits Status bits are not affected, except SR is included in the operation.

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

Example Restore the 20-bit registers R9, R10, R11, R12, R13 from the stack.

```
POPM.A   #5,R13     ; Restore R9, R10, R11, R12, R13
```

Example Restore the 16-bit registers R9, R10, R11, R12, R13 from the stack.

```
POPM.W   #5,R13     ; Restore R9, R10, R11, R12, R13
```

PUSHM.A Save n CPU registers (20-bit data) on the stack

PUSHM.[W] Save n CPU registers (16-bit words) on the stack

Syntax `PUSHM.A #n,Rdst` $1 \le n \le 16$

`PUSHM.W #n,Rdst` or `PUSHM #n,Rdst` $1 \le n \le 16$

Operation PUSHM.A: Save the 20-bit CPU register values on the stack. The stack pointer (SP) is decremented by four for each register stored on the stack. The MSBs are stored first (higher address).

PUSHM.W: Save the 16-bit CPU register values on the stack. The stack pointer is decremented by two for each register stored on the stack.

Description PUSHM.A: The n CPU registers, starting with Rdst backwards, are stored on the stack. The stack pointer is decremented by (n × 4) after the operation. The data (Rn.19:0) of the pushed CPU registers is not affected.

PUSHM.W: The n registers, starting with Rdst backwards, are stored on the stack. The stack pointer is decremented by (n × 2) after the operation. The data (Rn.19:0) of the pushed CPU registers is not affected.

Note : This instruction does not use the extension word.

Status Bits Status bits are not affected.

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

Example Save the five 20-bit registers R9, R10, R11, R12, R13 on the stack.

```
PUSHM.A   #5,R13     ; Save R13, R12, R11, R10, R9
```

Example Save the five 16-bit registers R9, R10, R11, R12, R13 on the stack.

```
PUSHM.W   #5,R13     ; Save R13, R12, R11, R10, R9
```

*** POPX.A** Restore single address-word from the stack

*** POPX.[W]** Restore single word from the stack

*** POPX.B** Restore single byte from the stack

Syntax `POPX.A dst`

`POPX dst` or `POPX.W dst`

`POPX.B dst`

Operation Restore the 8/16/20-bit value from the stack to the destination. 20-bit addresses are possible. The stack pointer SP is incremented by two (byte and word operands) and by four (address-word operand).

Emulation `MOVX(.B,.A) @SP+,dst`

Description The item on TOS is written to the destination operand. register mode, indexed mode, symbolic mode, and absolute mode are possible. The stack pointer is incremented by two or four.

Note: the stack pointer is incremented by two also for byte operations.

Status Bits Status bits are not affected.

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

Example Write the 16-bit value on TOS to the 20-bit address &EDE.

```
POPX.W   &EDE     ; Write word to address EDE
```

Example Write the 20-bit value on TOS to R9.

```
POPX.A   R9     ; Write address-word to R9
```

PUSHX.A	Restore single address-word from the stack
PUSHX.[W]	Restore single word from the stack
PUSHX.B	Restore single byte from the stack

Syntax	`PUSHX.A src`
	`PUSHX src` or `PUSHX.W src`
	`PUSHX.B src`

Operation	Save the 8/16/20-bit value of the source operand on the TOS. 20-bit addresses are possible. The stack pointer (SP) is decremented by two (byte and word operands) or by four (address-word operand) before the write operation.
Description	The stack pointer is decremented by two (byte and word operands) or by four (address-word operand). Then the source operand is written to the TOS. All seven addressing modes are possible for the source operand. Note : This instruction does not use the extension word.
Status Bits	Status bits are not affected.
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Save the byte at the 20-bit address &EDE on the stack.

```
PUSHX.B   &EDE     ; Save byte at address EDE
```

Example Save the 20-bit value in R9 on the stack.

```
PUSHX.A   R9       ; Save address-word in R9
```

RLAM.A Rotate left arithmetically the 20-bit CPU register content

RLAM.[W] Rotate left arithmetically the 16-bit CPU register content

Syntax `RLAM.A #n,Rdst` 1 ≤ n ≤ 4

`RLAM.W #n,Rdst` or `RLAM #n,Rdst` 1 ≤ n ≤ 4

Operation C ← MSB ← MSB-1 .... LSB+1 ← LSB ← 0

Description The destination operand is shifted arithmetically left one, two, three, or four positions as shown in Figure 5-44. RLAM works as a multiplication (signed and unsigned) with 2, 4, 8, or 16. The word instruction RLAM.W clears the bits Rdst.19:16.

Note : This instruction does not use the extension word.

Status Bits N: Set if result is negative

.A: Rdst.19 = 1, reset if Rdst.19 = 0

.W: Rdst.15 = 1, reset if Rdst.15 = 0

Z: Set if result is zero, reset otherwise

C: Loaded from the MSB (n = 1), MSB-1 (n = 2), MSB-2 (n = 3), MSB-3 (n = 4)

V: Undefined

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

Example The 20-bit operand in R5 is shifted left by three positions. It operates equal to an arithmetic multiplication by 8.

```
RLAM.A   #3,R5     ; R5 = R5 x 8
```

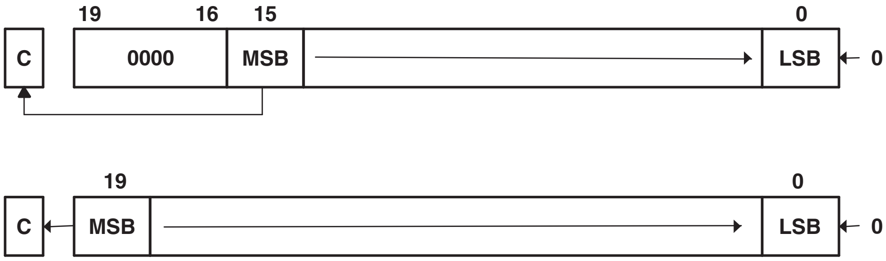


Figure 5-44. Rotate Left Arithmetically—RLAM[.W] and RLAM.A

*** RLAX.A** Rotate left arithmetically address-word

*** RLAX.[W]** Rotate left arithmetically word

*** RLAX.B** Rotate left arithmetically byte

Syntax `RLAX.A dst`

`RLAX dst` or `RLAX.W dst`

`RLAX.B dst`

Operation C ← MSB ← MSB-1 .... LSB+1 ← LSB ← 0

Emulation `ADDX.A dst,dst`

`ADDX dst,dst`

`ADDX.B dst,dst`

Description The destination operand is shifted left one position as shown in Figure 5-45. The MSB is shifted into the carry bit (C) and the LSB is filled with 0. The RLAX instruction acts as a signed multiplication by 2.

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 040000h ≤ dst < 0C0000h; reset otherwise

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 The 20-bit value in R7 is multiplied by 2.

```
RLAX.A   R7     ; Shift left R7 (20-bit)
```

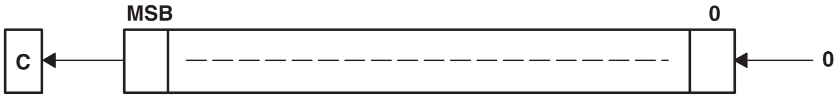


Figure 5-45. Destination Operand-Arithmetic Shift Left

*** RLCX.A** Rotate left through carry address-word

*** RLCX.[W]** Rotate left through carry word

*** RLCX.B** Rotate left through carry byte

Syntax `RLCX.A dst`

`RLCX dst` or `RLCX.W dst`

`RLCX.B dst`

Operation C ← MSB ← MSB-1 .... LSB+1 ← LSB ← C

Emulation `ADDCX.A dst,dst`

`ADDCX dst,dst`

`ADDCX.B dst,dst`

Description The destination operand is shifted left one position as shown in Figure 5-46. The carry bit (C) is shifted into the LSB and the MSB is shifted into the carry bit (C).

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 040000h ≤ dst < 0C0000h; reset otherwise

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 The 20-bit value in R5 is shifted left one position.

```
RLCX.A   R5     ; (R5 x 2) + C -> R5
```

Example The RAM byte LEO is shifted left one position. PC is pointing to upper memory.

```
RLCX.B   LEO    ; RAM(LEO) x 2 + C -> RAM(LEO)
```

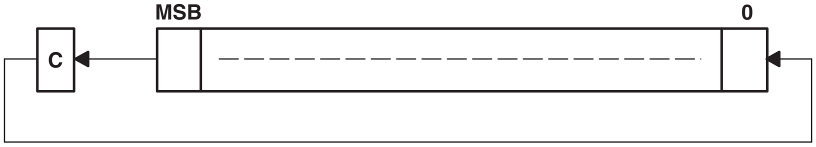


Figure 5-46. Destination Operand-Carry Left Shift

RRAM.A Rotate right arithmetically the 20-bit CPU register content

RRAM.[W] Rotate right arithmetically the 16-bit CPU register content

Syntax `RRAM.A #n,Rdst` 1 ≤ n ≤ 4

`RRAM.W #n,Rdst` or `RRAM #n,Rdst` 1 ≤ n ≤ 4

Operation MSB → MSB → MSB–1 ... LSB+1 → LSB → C

Description The destination operand is shifted right arithmetically by one, two, three, or four bit positions as shown in Figure 5-47. The MSB retains its value (sign). RRAM operates equal to a signed division by 2/4/8/16. The MSB is retained and shifted into MSB-1. The LSB+1 is shifted into the LSB, and the LSB is shifted into the carry bit C. The word instruction RRAM.W clears the bits Rdst.19:16.

Note : This instruction does not use the extension word.

Status Bits N: Set if result is negative

.A: Rdst.19 = 1, reset if Rdst.19 = 0

.W: Rdst.15 = 1, reset if Rdst.15 = 0

Z: Set if result is zero, reset otherwise

C: Loaded from the LSB (n = 1), LSB+1 (n = 2), LSB+2 (n = 3), or LSB+3 (n = 4)

V: Reset

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

Example The signed 20-bit number in R5 is shifted arithmetically right two positions.

```
RRAM.A   #2,R5          ; R5/4 -> R5
```

Example The signed 20-bit value in R15 is multiplied by 0.75. (0.5 + 0.25) × R15.

```
PUSHM.A   #1,R15        ; Save extended R15 on stack
RRAM.A    #1,R15        ; R15 y 0.5 -> R15
ADDX.A    @SP+,R15      ; R15 y 0.5 + R15 = 1.5 y R15 -> R15
RRAM.A    #1,R15        ; (1.5 y R15) y 0.5 = 0.75 y R15 -> R15
```

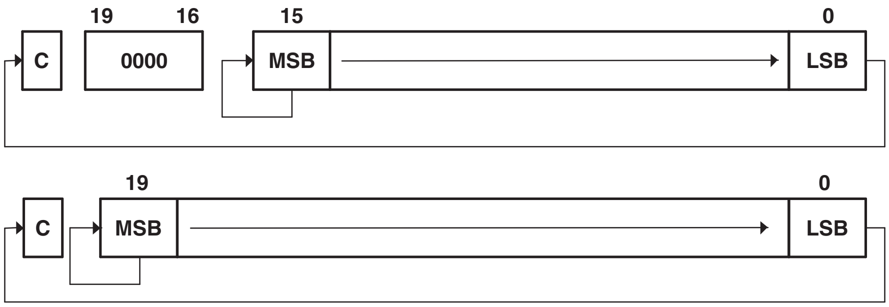


Figure 5-47. Rotate Right Arithmetically RRAM[.W] and RRAM.A

RRAX.A Rotate right arithmetically the 20-bit operand

RRAX.[W] Rotate right arithmetically the 16-bit operand

RRAX.B Rotate right arithmetically the 8-bit operand

Syntax

`RRAX.A Rdst`

`RRAX.W Rdst`

`RRAX Rdst`

`RRAX.B Rdst`

`RRAX.A dst`

`RRAX dst` or `RRAX.W dst`

`RRAX.B dst`

Operation MSB → MSB → MSB–1 ... LSB+1 → LSB → C

Description Register mode for the destination: the destination operand is shifted right by one bit position as shown in Figure 5-48. The MSB retains its value (sign). The word instruction RRAX.W clears the bits Rdst.19:16, the byte instruction RRAX.B clears the bits Rdst.19:8. The MSB retains its value (sign), the LSB is shifted into the carry bit. RRAX here operates equal to a signed division by 2.

All other modes for the destination: the destination operand is shifted right arithmetically by one bit position as shown in Figure 5-49. The MSB retains its value (sign), the LSB is shifted into the carry bit. RRAX here operates equal to a signed division by 2. All addressing modes - with the exception of the Immediate Mode - are possible in the full memory.

Status Bits

N: Set if result is negative, reset if positive
.A: dst.19 = 1, reset if dst.19 = 0
.W: dst.15 = 1, reset if dst.15 = 0
.B: dst.7 = 1, reset if dst.7 = 0

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 The signed 20-bit number in R5 is shifted arithmetically right four positions.

```
RPT      #4
RRAX.A   R5         ; R5/16 -> R5
```

Example The signed 8-bit value in EDE is multiplied by 0.5.

```
RRAX.B   &EDE       ; EDE/2 -> EDE
```

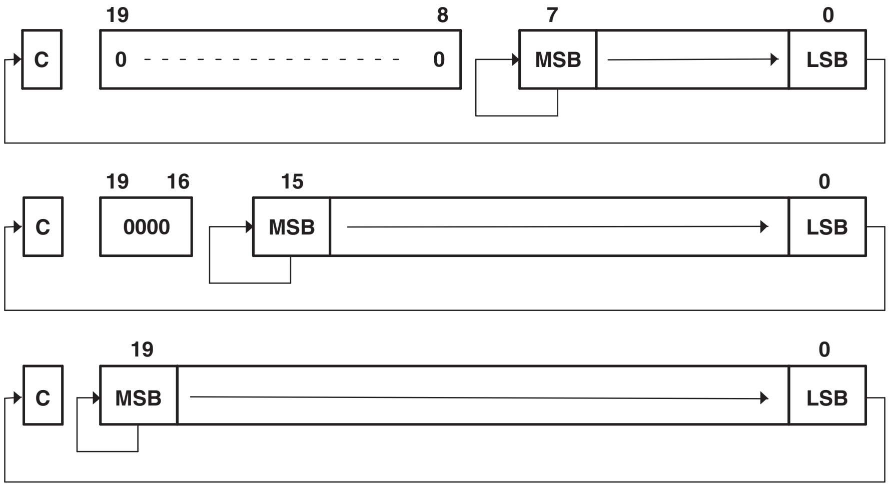


Figure 5-48. Rotate Right Arithmetically RRAX(.B,.A) – Register Mode

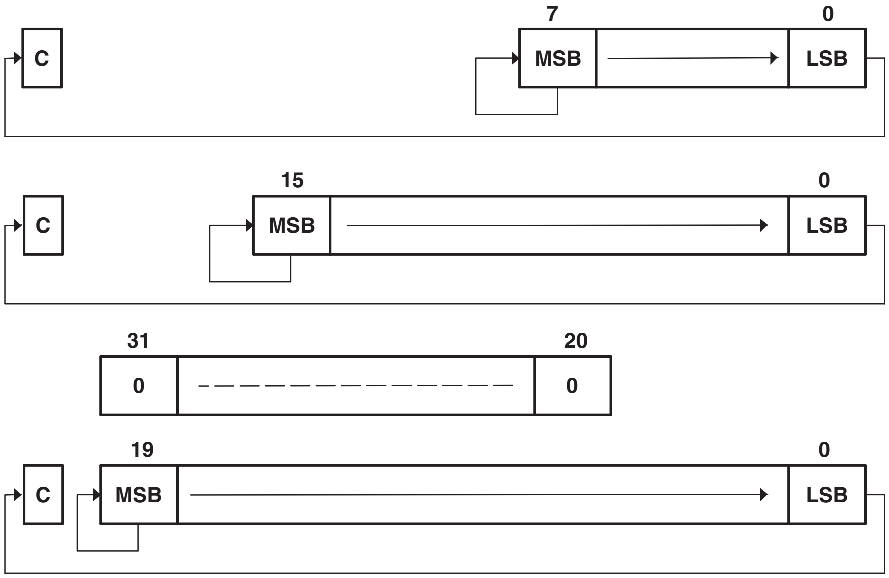


Figure 5-49. Rotate Right Arithmetically RRAX(.B,.A) – Non-Register Mode

RRCM.A Rotate right through carry the 20-bit CPU register content

RRCM.[W] Rotate right through carry the 16-bit CPU register content

Syntax `RRCM.A #n,Rdst` 1 ≤ n ≤ 4

`RRCM.W #n,Rdst` or `RRCM #n,Rdst` 1 ≤ n ≤ 4

Operation C → MSB → MSB–1 ... LSB+1 → LSB → C

Description The destination operand is shifted right by one, two, three, or four bit positions as shown in Figure 5-50. The carry bit C is shifted into the MSB, the LSB is shifted into the carry bit. The word instruction RRCM.W clears the bits Rdst.19:16.

Note : This instruction does not use the extension word.

Status Bits N: Set if result is negative

.A: Rdst.19 = 1, reset if Rdst.19 = 0

.W: Rdst.15 = 1, reset if Rdst.15 = 0

Z: Set if result is zero, reset otherwise

C: Loaded from the LSB (n = 1), LSB+1 (n = 2), LSB+2 (n = 3), or LSB+3 (n = 4)

V: Reset

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

Example The address-word in R5 is shifted right by three positions. The MSB-2 is loaded with 1.

```
SETC                 ; Prepare carry for MSB-2
RRCM.A   #3,R5       ; R5 = R5   3 + 20000h
```

Example The word in R6 is shifted right by two positions. The MSB is loaded with the LSB. The MSB–1 is loaded with the contents of the carry flag.

```
RRCM.W   #2,R6       ; R6 = R6   2. R6.19:16 = 0
```

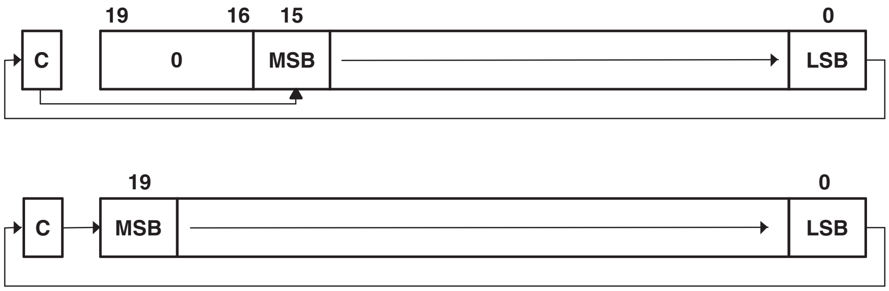


Figure 5-50. Rotate Right Through Carry RRCM[.W] and RRCM.A

RRCX.A Rotate right through carry the 20-bit operand

RRCX.[W] Rotate right through carry the 16-bit operand

RRCX.B Rotate right through carry the 8-bit operand

Syntax

`RRCX.A Rdst`

`RRCX.W Rdst`

`RRCX Rdst`

`RRCX.B Rdst`

`RRCX.A dst`

`RRCX dst` or `RRCX.W dst`

`RRCX.B dst`

Operation C → MSB → MSB–1 ... LSB+1 → LSB → C

Description Register mode for the destination: the destination operand is shifted right by one bit position as shown in Figure 5-51. The word instruction RRCX.W clears the bits Rdst.19:16, the byte instruction RRCX.B clears the bits Rdst.19:8. The carry bit C is shifted into the MSB, the LSB is shifted into the carry bit.

All other modes for the destination: the destination operand is shifted right by one bit position as shown in Figure 5-52. The carry bit C is shifted into the MSB, the LSB is shifted into the carry bit. All addressing modes - with the exception of the Immediate Mode - are possible in the full memory.

Status Bits

N: Set if result is negative

.A: dst.19 = 1, reset if dst.19 = 0

.W: dst.15 = 1, reset if dst.15 = 0

.B: dst.7 = 1, reset if dst.7 = 0

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 The 20-bit operand at address EDE is shifted right by one position. The MSB is loaded with 1.

```
SETC                ; Prepare carry for MSB
RRCX.A   EDE        ; EDE = EDE  1 + 80000h
```

Example The word in R6 is shifted right by twelve positions.

```
RPT       #12
RRCX.W    R6       ; R6 = R6  12. R6.19:16 = 0
```

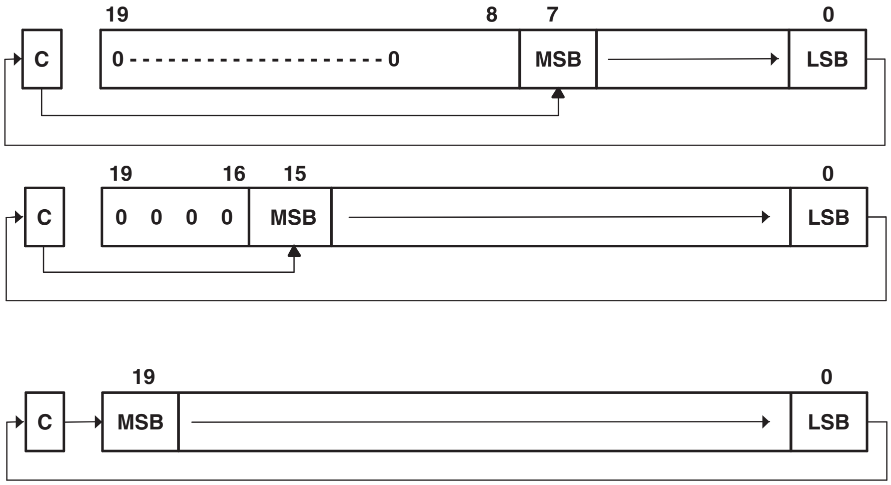


Figure 5-51. Rotate Right Through Carry RRCX(.B,.A) – Register Mode

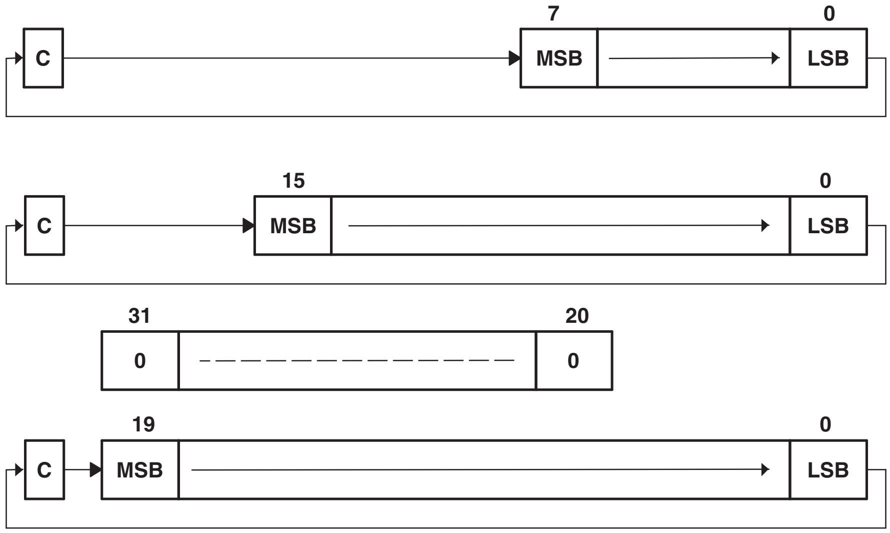


Figure 5-52. Rotate Right Through Carry RRCX(.B,.A) – Non-Register Mode

RRUM.A Rotate right through carry the 20-bit CPU register content

RRUM.[W] Rotate right through carry the 16-bit CPU register content

Syntax `RRUM.A #n,Rdst` 1 ≤ n ≤ 4

`RRUM.W #n,Rdst` or `RRUM #n,Rdst` 1 ≤ n ≤ 4

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

Description The destination operand is shifted right by one, two, three, or four bit positions as shown in Figure 5-53. Zero is shifted into the MSB, the LSB is shifted into the carry bit. RRUM works like an unsigned division by 2, 4, 8, or 16. The word instruction RRUM.W clears the bits Rdst.19:16.

Note : This instruction does not use the extension word.

Status Bits N: Set if result is negative

.A: Rdst.19 = 1, reset if Rdst.19 = 0

.W: Rdst.15 = 1, reset if Rdst.15 = 0

Z: Set if result is zero, reset otherwise

C: Loaded from the LSB (n = 1), LSB+1 (n = 2), LSB+2 (n = 3), or LSB+3 (n = 4)

V: Reset

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

Example The unsigned address-word in R5 is divided by 16.

```
RRUM.A   #4,R5      ; R5 = R5  4. R5/16
```

Example The word in R6 is shifted right by one bit. The MSB R6.15 is loaded with 0.

```
RRUM.W   #1,R6      ; R6 = R6/2. R6.19:15 = 0
```

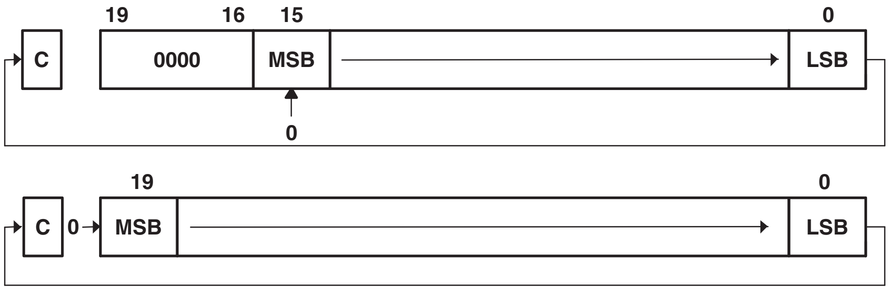


Figure 5-53. Rotate Right Unsigned RRUM[.W] and RRUM.A

RRUX.A Rotate right unsigned the 20-bit operand

RRUX.[W] Rotate right unsigned the 16-bit operand

RRUX.B Rotate right unsigned the 8-bit operand

Syntax `RRUX.A Rdst`

`RRUX.W Rdst`

`RRUX Rdst`

`RRUX.B Rdst`

Operation C=0 → MSB → MSB–1 ... LSB+1 → LSB → C

Description RRUX is valid for register mode only: the destination operand is shifted right by one bit position as shown in Figure 5-54. The word instruction RRUX.W clears the bits Rdst.19:16. The byte instruction RRUX.B clears the bits Rdst.19:8. Zero is shifted into the MSB, the LSB is shifted into the carry bit.

Status Bits N: Set if result is negative

.A: dst.19 = 1, reset if dst.19 = 0

.W: dst.15 = 1, reset if dst.15 = 0

.B: dst.7 = 1, reset if dst.7 = 0

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 The word in R6 is shifted right by twelve positions.

```
RPT      #12
RRUX.W   R6        ; R6 = R6  12. R6.19:16 = 0
```

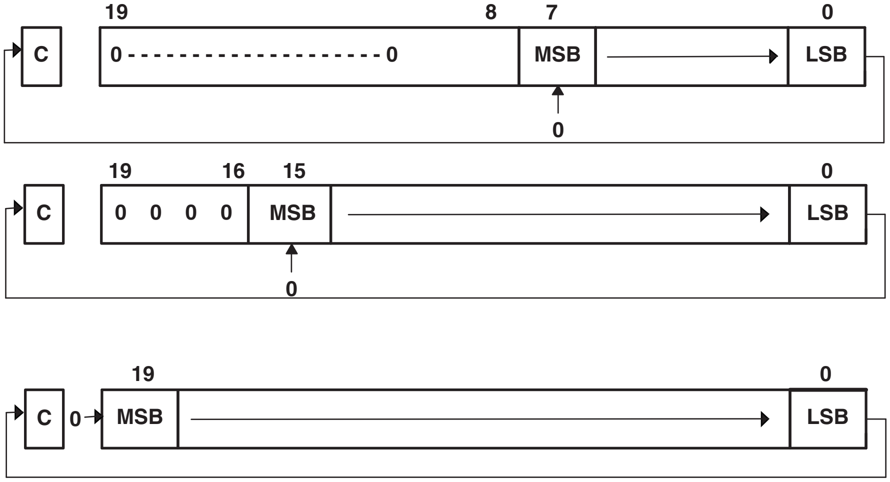


Figure 5-54. Rotate Right Unsigned RRUX(.B,.A) – Register Mode

*** SBCX.A** Subtract source and borrow/.NOT. carry from destination address-word

*** SBCX.[W]** Subtract source and borrow/.NOT. carry from destination word

*** SBCX.B** Subtract source and borrow/.NOT. carry from destination byte

Syntax `SBCX.A dst`

`SBCX dst` or `SBCX.W dst`

`SBCX.B dst`

Operation dst + 0FFFFFh + C → dst

dst + 0FFFFh + C → dst

dst + 0FFh + C → dst

Emulation `SBCX.A #0,dst`

`SBCX #0,dst`

`SBCX.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 8-bit counter pointed to by R13 is subtracted from a 16-bit counter pointed to by R12.

```
SUBX.B   @R13,0(R12)      ; Subtract LSDs
SBCX.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

SUBX.A Subtract source address-word from destination address-word

SUBX.[W] Subtract source word from destination word

SUBX.B Subtract source byte from destination byte

Syntax `SUBX.A src,dst`

`SUBX src,dst` or `SUBX.W src,dst`

`SUBX.B src,dst`

Operation (.not. src) + 1 + dst → dst or dst – src → dst

Description The source operand is subtracted from the destination operand. This is done by adding the 1's complement of the source + 1 to the destination. The source operand is not affected. The result is written to the destination operand. Both operands may be located in the full address space.

Status Bits N: Set if result is negative (src > dst), reset if positive (src ≤ dst)

Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst)

C: Set if there is a carry from the MSB, reset otherwise

V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow).

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

Example A 20-bit constant 87654h is subtracted from EDE (LSBs) and EDE+2 (MSBs).

```
SUBX.A   #87654h,EDE      ; Subtract 87654h from EDE+2|EDE
```

Example A table word pointed to by R5 (20-bit address) is subtracted from R7. Jump to label TONI if R7 contains zero after the instruction. R5 is auto-incremented by 2. R7.19:16 = 0.

```
SUBX.W   @R5+,R7          ; Subtract table number from R7. R5 + 2
JZ       TONI             ; R7 = @R5 (before subtraction)
...                       ; R7 <> @R5 (before subtraction)
```

Example Byte CNT is subtracted from the byte R12 points to in the full address space. Address of CNT is within PC ± 512 K.

```
SUBX.B   CNT,0(R12)       ; Subtract CNT from @R12
```

Note: Use SUBA for the following two cases for better density and execution.

```
SUBX.A   Rsrc,Rdst
SUBX.A   #imm20,Rdst
```

SUBCX.A Subtract source address-word with carry from destination address-word

SUBCX.[W] Subtract source word with carry from destination word

SUBCX.B Subtract source byte with carry from destination byte

Syntax `SUBCX.A src,dst`

`SUBCX src,dst` or `SUBCX.W src,dst`

`SUBCX.B src,dst`

Operation (.not. src) + C + dst → dst or dst – (src – 1) + C → dst

Description The source operand is subtracted from the destination operand. This is made by adding the 1's complement of the source + carry to the destination. The source operand is not affected, the result is written to the destination operand. Both operands may be located in the full address space.

Status Bits

N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)

Z: Set if result is zero, reset otherwise

C: Set if there is a carry from the MSB, reset otherwise

V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow).

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

Example A 20-bit constant 87654h is subtracted from R5 with the carry from the previous instruction.

```
SUBCX.A   #87654h,R5      ; Subtract 87654h + C from R5
```

Example A 48-bit number (3 words) pointed to by R5 (20-bit address) is subtracted from a 48-bit counter in RAM, pointed to by R7. R5 auto-increments to point to the next 48-bit number.

```
SUBX.W    @R5+,0(R7)      ; Subtract LSBs. R5 + 2
SUBCX.W   @R5+,2(R7)      ; Subtract MIDs with C. R5 + 2
SUBCX.W   @R5+,4(R7)      ; Subtract MSBs with C. R5 + 2
```

Example Byte CNT is subtracted from the byte R12 points to. The carry of the previous instruction is used. 20-bit addresses.

```
SUBCX.B   &CNT,0(R12)     ; Subtract byte CNT from @R12
```

SWPBX.A Swap bytes of lower word

SWPBX.[W] Swap bytes of word

Syntax `SWPBX.A dst`

`SWPBX dst` or `SWPBX.W dst`

Operation dst.15:8 ↔ dst.7:0

Description Register Mode: Rn.15:8 are swapped with Rn.7:0. When the .A extension is used, Rn.19:16 are unchanged. When the .W extension is used, Rn.19:16 are cleared.

Other Modes: When the .A extension is used, bits 31:20 of the destination address are cleared, bits 19:16 are left unchanged, and bits 15:8 are swapped with bits 7:0. When the .W extension is used, bits 15:8 are swapped with bits 7:0 of the addressed word.

Status Bits Status bits are not affected

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

Example Exchange the bytes of RAM address-word EDE.

```
MOVX.A    #23456h,&EDE      ; 23456h -> EDE
SWPBX.A   EDE               ; 25634h -> EDE
```

Example Exchange the bytes of R5.

```
MOVA      #23456h,R5        ; 23456h -> R5
SWPBX.W   R5                ; 05634h -> R5
```

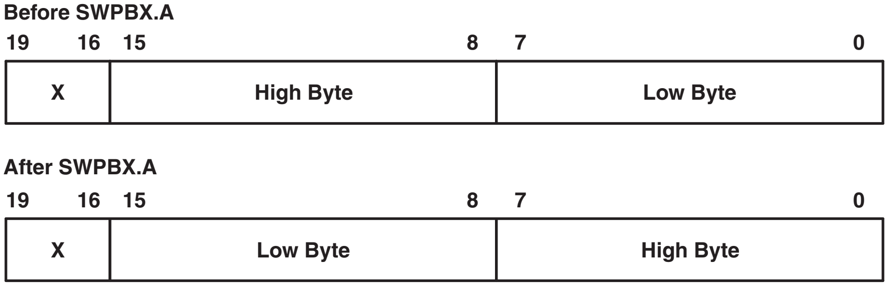


Figure 5-55. Swap Bytes SWPBX.A Register Mode

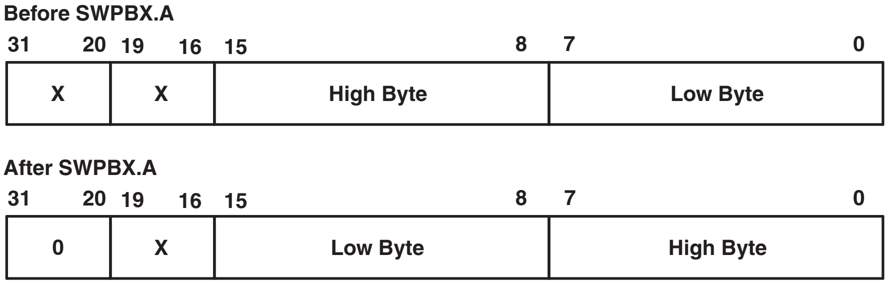


Figure 5-56. Swap Bytes SWPBX.A In Memory

Before SWPBX

19 16	15 8	7 0
X	High Byte	Low Byte

After SWPBX

19 16	15 8	7 0
0	Low Byte	High Byte

Figure 5-57. Swap Bytes SWPBX[.W] Register Mode

Before SWPBX

15 8	7 0
High Byte	Low Byte

After SWPBX

15 8	7 0
Low Byte	High Byte

Figure 5-58. Swap Bytes SWPBX[.W] In Memory

SXTX.A Extend sign of lower byte to address-word

SXTX.[W] Extend sign of lower byte to word

Syntax `SXTX.A dst`

`SXTX dst` or `SXTX.W dst`

Operation dst.7 → dst.15:8, Rdst.7 → Rdst.19:8 (Register Mode)

Description Register Mode: The sign of the low byte of the operand (Rdst.7) is extended into the bits Rdst.19:8.

Other Modes: SXTX.A: the sign of the low byte of the operand (dst.7) is extended into dst.19:8. The bits dst.31:20 are cleared.

SXTX[.W]: the sign of the low byte of the operand (dst.7) is extended into dst.15:8.

Status Bits N: Set if result is negative, reset otherwise

Z: Set if result is zero, reset otherwise

C: Set if result is not zero, reset otherwise (C = .not.Z)

V: Reset

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

Example The signed 8-bit data in EDE.7:0 is sign extended to 20 bits: EDE.19:8. Bits 31:20 located in EDE+2 are cleared.

```
SXTX.A      &EDE            ; Sign extended EDE -> EDE+2/EDE
```

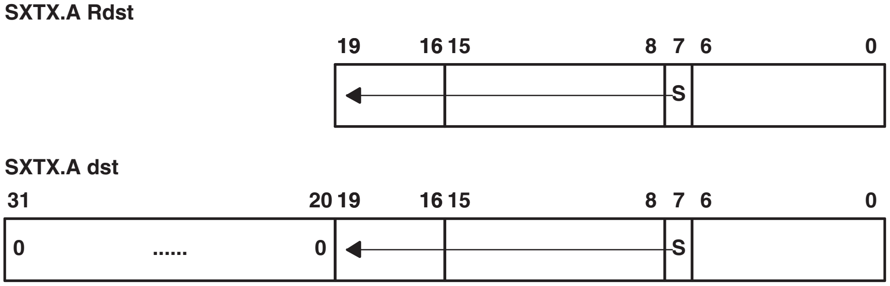


Figure 5-59. Sign Extend SXTX.A

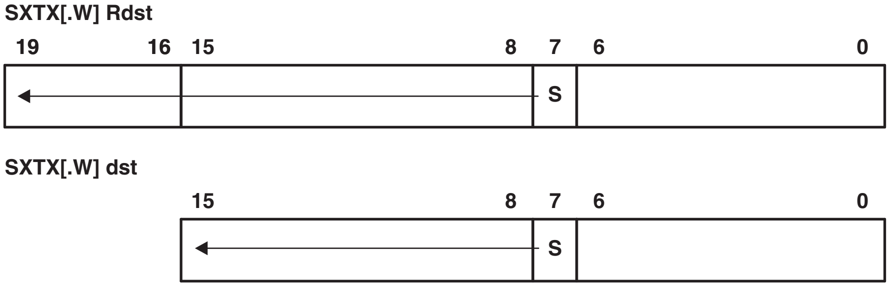


Figure 5-60. Sign Extend SXTX[.W]

*** TSTX.A** Test destination address-word

*** TSTX.[W]** Test destination word

*** TSTX.B** Test destination byte

Syntax `TSTX.A dst`

`TSTX dst` or `TSTX.W dst`

`TSTX.B dst`

Operation dst + 0FFFFFh + 1

dst + 0FFFFh + 1

dst + 0FFh + 1

Emulation `CMPX.A #0,dst`

`CMPX #0,dst`

`CMPX.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 RAM byte LEO is tested; PC is pointing to upper memory. If it is negative, continue at LEONEG; if it is positive but not zero, continue at LEOPOS.

```
          TSTX.B   LEO          ; Test LEO
          JN       LEONEG       ; LEO is negative
          JZ       LEOZERO      ; LEO is zero
LEOPOS    ......                ; LEO is positive but not zero
LEONEG    ......                ; LEO is negative
LEOZERO   ......                ; LEO is zero
```

XORX.A Exclusive OR source address-word with destination address-word

XORX.[W] Exclusive OR source word with destination word

XORX.B Exclusive OR source byte with destination byte

Syntax `XORX.A src,dst`

`XORX src,dst` or `XORX.W src,dst`

`XORX.B src,dst`

Operation src .xor. dst → dst

Description The source and destination operands are exclusively ORed. The result is placed into the destination. The source operand is not affected. The previous contents of the destination are lost. Both operands may be located in the full address space.

Status Bits N: Set if result is negative (MSB = 1), reset if positive (MSB = 0)

Z: Set if result is zero, reset otherwise

C: Set if result is not zero, reset otherwise (carry = .not. Zero)

V: Set if both operands are negative (before execution), reset otherwise

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

Example Toggle bits in address-word CNTR (20-bit data) with information in address-word TONI (20-bit address).

```
XORX.A   TONI,&CNTR      ; Toggle bits in CNTR
```

Example A table word pointed to by R5 (20-bit address) is used to toggle bits in R6.

```
XORX.W   @R5,R6          ; Toggle bits in R6. R6.19:16 = 0
```

Example Reset to zero those bits in the low byte of R7 that are different from the bits in byte EDE (20-bit address).

```
XORX.B   EDE,R7          ; Set different bits to 1 in R7
INV.B    R7              ; Invert low byte of R7. R7.19:8 = 0.
```

5.6.4 Address Instructions

MSP430X address instructions are instructions that support 20-bit operands but have restricted addressing modes. The addressing modes are restricted to the Register mode and the Immediate mode, except for the MOVA instruction. Restricting the addressing modes removes the need for the additional extension-word op-code improving code density and execution time. The MSP430X address instructions are listed and described in the following pages.

ADDA Add 20-bit source to a 20-bit destination register

Syntax `ADDA Rsrc,Rdst`

`ADDA #imm20,Rdst`

Operation src + Rdst → Rdst

Description The 20-bit source operand is added to the 20-bit destination CPU register. The previous contents of the destination are lost. The source operand is not affected.

Status Bits
N: Set if result is negative (Rdst.19 = 1), reset if positive (Rdst.19 = 0)
Z: Set if result is zero, reset otherwise
C: Set if there is a carry from the 20-bit result, reset otherwise
V: Set if the result of two positive operands is negative, or if the result of two negative numbers is positive, reset otherwise

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

Example R5 is increased by 0A4320h. The jump to TONI is performed if a carry occurs.

```
ADDA   #0A4320h,R5      ; Add A4320h to 20-bit R5
JC     TONI             ; Jump on carry
...                     ; No carry occurred
```

*** BRA** Branch to destination

Syntax `BRA dst`

Operation dst → PC

Emulation `MOVA dst,PC`

Description An unconditional branch is taken to a 20-bit address anywhere in the full address space. All seven source addressing modes can be used. The branch instruction is an address-word instruction. If the destination address is contained in a memory location X, it is contained in two ascending words: X (LSBs) and (X + 2) (MSBs).

Status Bits

N:	Not affected
Z:	Not affected
C:	Not affected
V:	Not affected

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

Examples Examples for all addressing modes are given.

Immediate Mode: Branch to label EDE located anywhere in the 20-bit address space or branch directly to address.

```
BRA       #EDE          ; MOVA   #imm20,PC
BRA       #01AA04h
```

Symbolic Mode: Branch to the 20-bit address contained in addresses EXEC (LSBs) and EXEC+2 (MSBs). EXEC is located at the address (PC + X) where X is within +32 K. Indirect addressing.

```
BRA       EXEC          ; MOVA   z16(PC),PC
```

Note: if the 16-bit index is not sufficient, a 20-bit index may be used with the following instruction.

```
MOVX.A    EXEC,PC       ; 1M byte range with 20-bit index
```

Absolute Mode: Branch to the 20-bit address contained in absolute addresses EXEC (LSBs) and EXEC+2 (MSBs). Indirect addressing.

```
BRA       &EXEC         ; MOVA   &abs20,PC
```

Register Mode: Branch to the 20-bit address contained in register R5. Indirect R5.

```
BRA       R5            ; MOVA   R5,PC
```

Indirect Mode: Branch to the 20-bit address contained in the word pointed to by register R5 (LSBs). The MSBs have the address (R5 + 2). Indirect, indirect R5.

```
BRA       @R5           ; MOVA   @R5,PC
```

Indirect, Auto-Increment Mode: Branch to the 20-bit address contained in the words pointed to by register R5 and increment the address in R5 afterwards by 4. The next time the S/W flow uses R5 as a pointer, it can alter the program execution due to access to the next address in the table pointed to by R5. Indirect, indirect R5.

```
BRA      @R5+         ; MOVA   @R5+,PC. R5 + 4
```

Indexed Mode: Branch to the 20-bit address contained in the address pointed to by register (R5 + X) (e.g., a table with addresses starting at X). (R5 + X) points to the LSBs, (R5 + X + 2) points to the MSBs of the address. X is within R5 + 32 K. Indirect, indirect (R5 + X).

```
BRA      X(R5)        ; MOVA   z16(R5),PC
```

Note: if the 16-bit index is not sufficient, a 20-bit index X may be used with the following instruction:

```
MOVX.A   X(R5),PC     ; 1M byte range with 20-bit index
```

CALLA Call a subroutine

Syntax `CALLA dst`

Operation

dst → tmp 20-bit dst is evaluated and stored

SP – 2 → SP

PC.19:16 → @SP updated PC with return address to TOS (MSBs)

SP – 2 → SP

PC.15:0 → @SP updated PC to TOS (LSBs)

tmp → PC saved 20-bit dst to PC

Description A subroutine call is made to a 20-bit address anywhere in the full address space. All seven source addressing modes can be used. The call instruction is an address-word instruction. If the destination address is contained in a memory location X, it is contained in two ascending words: X (LSBs) and (X + 2) (MSBs). Two words on the stack are needed for the return address. The return is made with the instruction RETA.

Status Bits

N: Not affected

Z: Not affected

C: Not affected

V: Not affected

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

Examples Examples for all addressing modes are given.

Immediate Mode: Call a subroutine at label EXEC or call directly an address.

```
CALLA   #EXEC         ; Start address EXEC
CALLA   #01AA04h      ; Start address 01AA04h
```

Symbolic Mode: Call a subroutine at the 20-bit address contained in addresses EXEC (LSBs) and EXEC+2 (MSBs). EXEC is located at the address (PC + X) where X is within +32 K. Indirect addressing.

```
CALLA   EXEC          ; Start address at @EXEC. z16(PC)
```

Absolute Mode: Call a subroutine at the 20-bit address contained in absolute addresses EXEC (LSBs) and EXEC+2 (MSBs). Indirect addressing.

```
CALLA   &EXEC         ; Start address at @EXEC
```

Register Mode: Call a subroutine at the 20-bit address contained in register R5. Indirect R5.

```
CALLA   R5            ; Start address at @R5
```

Indirect Mode: Call a subroutine at the 20-bit address contained in the word pointed to by register R5 (LSBs). The MSBs have the address (R5 + 2). Indirect, indirect R5.

```
CALLA   @R5           ; Start address at @R5
```

Indirect, Auto-Increment Mode: Call a subroutine at the 20-bit address contained in the words pointed to by register R5 and increment the 20-bit address in R5 afterwards by 4. The next time the S/W flow uses R5 as a pointer, it can alter the program execution due to access to the next word address in the table pointed to by R5. Indirect, indirect R5.

```
CALLA   @R5+         ; Start address at @R5. R5 + 4
```

Indexed Mode: Call a subroutine at the 20-bit address contained in the address pointed to by register (R5 + X); e.g., a table with addresses starting at X. (R5 + X) points to the LSBs, (R5 + X + 2) points to the MSBs of the word address. X is within R5 +32 K. Indirect, indirect (R5 + X).

```
CALLA   X(R5)        ; Start address at @(R5+X). z16(R5)
```

*** CLRA**	Clear 20-bit destination register
Syntax	`CLRA Rdst`
Operation	0 → Rdst
Emulation	`MOVA #0,Rdst`
Description	The destination register is cleared.
Status Bits	Status bits are not affected.
Example	The 20-bit value in R10 is cleared.

```
CLRA   R10      ; 0 -> R10
```

CMPA Compare the 20-bit source with a 20-bit destination register

Syntax `CMPA Rsrc,Rdst`

`CMPA #imm20,Rdst`

Operation (.not. src) + 1 + Rdst or Rdst – src

Description The 20-bit source operand is subtracted from the 20-bit destination CPU register. This is made by adding the 1's complement of the source + 1 to the destination register. The result affects only the status bits.

Status Bits N: Set if result is negative (src > dst), reset if positive (src ≤ dst)

Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst)

C: Set if there is a carry from the MSB, reset otherwise

V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)

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

Example A 20-bit immediate operand and R6 are compared. If they are equal the program continues at label EQUAL.

```
CMPA   #12345h,R6      ; Compare R6 with 12345h
JEQ    EQUAL           ; R5 = 12345h
...                    ; Not equal
```

Example The 20-bit values in R5 and R6 are compared. If R5 is greater than (signed) or equal to R6, the program continues at label GRE.

```
CMPA   R6,R5           ; Compare R6 with R5 (R5 - R6)
JGE    GRE             ; R5 >= R6
...                    ; R5 < R6
```

*** DECDA** Double-decrement 20-bit destination register

Syntax `DECDA Rdst`

Operation Rdst – 2 → Rdst

Emulation `SUBA #2,Rdst`

Description The destination register is decremented by two. The original contents are lost.

Status Bits N: Set if result is negative, reset if positive

Z: Set if Rdst contained 2, reset otherwise

C: Reset if Rdst contained 0 or 1, set otherwise

V: Set if an arithmetic overflow occurs, otherwise reset

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

Example The 20-bit value in R5 is decremented by 2.

```
DECDA   R5      ; Decrement R5 by two
```

*** INCDA** Double-increment 20-bit destination register

Syntax `INCDA Rdst`

Operation Rdst + 2 → Rdst

Emulation `ADDA #2,Rdst`

Description The destination register is incremented by two. The original contents are lost.

Status Bits

N: Set if result is negative, reset if positive

Z: Set if Rdst contained 0FFFFEh, reset otherwise
Set if Rdst contained 0FFFEh, reset otherwise
Set if Rdst contained 0FEh, reset otherwise

C: Set if Rdst contained 0FFFFEh or 0FFFFFh, reset otherwise
Set if Rdst contained 0FFFEh or 0FFFFh, reset otherwise
Set if Rdst contained 0FEh or 0FFh, reset otherwise

V: Set if Rdst contained 07FFFEh or 07FFFFh, reset otherwise
Set if Rdst contained 07FFEh or 07FFFh, reset otherwise
Set if Rdst contained 07Eh or 07Fh, reset otherwise

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

Example The 20-bit value in R5 is incremented by 2.

```
INCDA   R5      ; Increment R5 by two
```

MOVA Move the 20-bit source to the 20-bit destination

Syntax

```
MOVA Rsrc,Rdst
MOVA #imm20,Rdst
MOVA z16(Rsrc),Rdst
MOVA EDE,Rdst
MOVA &abs20,Rdst
MOVA @Rsrc,Rdst
MOVA @Rsrc+,Rdst
MOVA Rsrc,z16(Rdst)
MOVA Rsrc,&abs20
```

Operation

src → Rdst

Rsrc → dst

Description The 20-bit source operand is moved to the 20-bit destination. The source operand is not affected. The previous content of the destination is lost.

Status Bits

N: Not affected

Z: Not affected

C: Not affected

V: Not affected

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

Examples Copy 20-bit value in R9 to R8.

```
MOVA   R9,R8         ; R9 -> R8
```

Write 20-bit immediate value 12345h to R12.

```
MOVA #12345h,R12 ; 12345h -> R12
```

Copy 20-bit value addressed by (R9 + 100h) to R8. Source operand in addresses (R9 + 100h) LSBs and (R9 + 102h) MSBs.

```
MOVA   100h(R9),R8      ; Index: + 32 K. 2 words transferred
```

Move 20-bit value in 20-bit absolute addresses EDE (LSBs) and EDE+2 (MSBs) to R12.

```
MOVA   &EDE,R12         ; &EDE -> R12. 2 words transferred
```

Move 20-bit value in 20-bit addresses EDE (LSBs) and EDE+2 (MSBs) to R12. PC index ± 32 K.

```
MOVA   EDE,R12          ; EDE -> R12. 2 words transferred
```

Copy 20-bit value R9 points to (20 bit address) to R8. Source operand in addresses @R9 LSBs and @(R9 + 2) MSBs.

```
MOVA   @R9,R8           ; @R9 -> R8. 2 words transferred
```

Copy 20-bit value R9 points to (20 bit address) to R8. R9 is incremented by four afterwards. Source operand in addresses @R9 LSBs and @(R9 + 2) MSBs.

```
MOVA    @R9+,R8           ; @R9 -> R8. R9 + 4. 2 words transferred.
```

Copy 20-bit value in R8 to destination addressed by (R9 + 100h). Destination operand in addresses @(R9 + 100h) LSBs and @(R9 + 102h) MSBs.

```
MOVA    R8,100h(R9)       ; Index: +- 32 K. 2 words transferred
```

Move 20-bit value in R13 to 20-bit absolute addresses EDE (LSBs) and EDE+2 (MSBs).

```
MOVA    R13,&EDE          ; R13 -> EDE. 2 words transferred
```

Move 20-bit value in R13 to 20-bit addresses EDE (LSBs) and EDE+2 (MSBs). PC index ± 32 K.

```
MOVA    R13,EDE           ; R13 -> EDE. 2 words transferred
```

*** RETA** Return from subroutine

Syntax RETA

Operation @SP → PC.15:0 LSBs (15:0) of saved PC to PC.15:0

SP + 2 → SP

@SP → PC.19:16 MSBs (19:16) of saved PC to PC.19:16

SP + 2 → SP

Emulation `MOVA @SP+,PC`

Description The 20-bit return address information, pushed onto the stack by a CALLA instruction, is restored to the program counter PC. The program continues at the address following the subroutine call. The status register bits SR.11:0 are not affected. This allows the transfer of information with these bits.

Status Bits N: Not affected

Z: Not affected

C: Not affected

V: Not affected

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

Example Call a subroutine SUBR from anywhere in the 20-bit address space and return to the address after the CALLA.

```
        CALLA     #SUBR       ; Call subroutine starting at SUBR
        ...                   ; Return by RETA to here
SUBR    PUSHM.A   #2,R14      ; Save R14 and R13 (20 bit data)
        ...                   ; Subroutine code
        POPM.A    #2,R14      ; Restore R13 and R14 (20 bit data)
        RETA                  ; Return (to full address space)
```

*** TSTA** Test 20-bit destination register

Syntax `TSTA Rdst`

Operation dst + 0FFFFFh + 1

dst + 0FFFFh + 1

dst + 0FFh + 1

Emulation `CMPA #0,Rdst`

Description The destination register is compared with zero. The status bits are set according to the result. The destination register is not affected.

Status Bits N: Set if destination register is negative, reset if positive

Z: Set if destination register contains zero, reset otherwise

C: Set

V: Reset

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

Example The 20-bit value in R7 is tested. If it is negative, continue at R7NEG; if it is positive but not zero, continue at R7POS.

```
         TSTA   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
```

SUBA Subtract 20-bit source from 20-bit destination register

Syntax SUBA `Rsrc,Rdst`

SUBA `#imm20,Rdst`

Operation (.not.src) + 1 + Rdst → Rdst or Rdst – src → Rdst

Description The 20-bit source operand is subtracted from the 20-bit destination register. This is made by adding the 1's complement of the source + 1 to the destination. The result is written to the destination register, the source is not affected.

Status Bits N: Set if result is negative (src > dst), reset if positive (src ≤ dst)

Z: Set if result is zero (src = dst), reset otherwise (src ≠ dst)

C: Set if there is a carry from the MSB (Rdst.19), reset otherwise

V: Set if the subtraction of a negative source operand from a positive destination operand delivers a negative result, or if the subtraction of a positive source operand from a negative destination operand delivers a positive result, reset otherwise (no overflow)

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

Example The 20-bit value in R5 is subtracted from R6. If a carry occurs, the program continues at label TONI.

```
SUBA   R5,R6      ; R6 - R5 -> R6
JC    TONI        ; Carry occurred
...               ; No carry
```

Chapter 6

SLAU208–June 2008

Flash Memory Controller

This chapter describes the operation of the MSP430x5xx flash memory controller.

6.1 Flash Memory Introduction

The MSP430 flash memory is byte-, word- and long-word addressable and programmable. The flash memory module has an integrated controller that controls programming and erase operations. The module contains three registers, a timing generator, and a voltage generator to supply program and erase voltages. The cumulative high-voltage time must not be exceeded and each word can be written not more than twice before another erase cycle. See device specific datasheet for details.

The flash memory features include:

- Internal programming voltage generation
- Byte, Word (2 bytes), and Long (4 bytes) programmable
- Ultralow-power operation
- Segment erase, bank erase and mass erase
- Marginal 0 and marginal 1 read modes
- Each bank can be erased individually while program execution can proceed in a different flash bank. The bank sizes are in the device-specific data sheet.

The block diagram of the flash memory and controller is shown in Figure 6-1.

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

The minimum V_{CORE} voltage during a flash write or erase operation is 1.6 V. If V_{CORE} falls below 1.6 V during a write or erase, the result of the write or erase will be unpredictable.

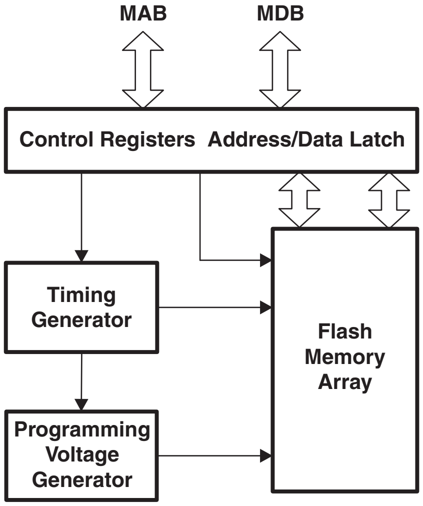


Figure 6-1. Flash Memory Module Block Diagram

6.2 Flash Memory Segmentation

The MSP430 flash main memory is partitioned into segments. Each bank contains 512-byte segments. Single bits, bytes or words can be written to flash memory, but a segment is the smallest size of the 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 and data can be located in either section. The difference between the sections is the segment size.

There are four information memory segments, A through D. Each information memory segment contains 128 bytes and can be erased individually.

The bootstrap loader memory consists of four segments, A through D. Each bootstrap loader memory segment contains 512 bytes and can be erased individually.

The main memory segment size is 512 byte. See the device-specific data sheet for the start and end addresses of each bank and for the complete memory map of a device.

Figure 6-2 shows the flash segmentation using an example of 256-KB flash that has four banks of 64 KB, the segments A through D, and the information memory.

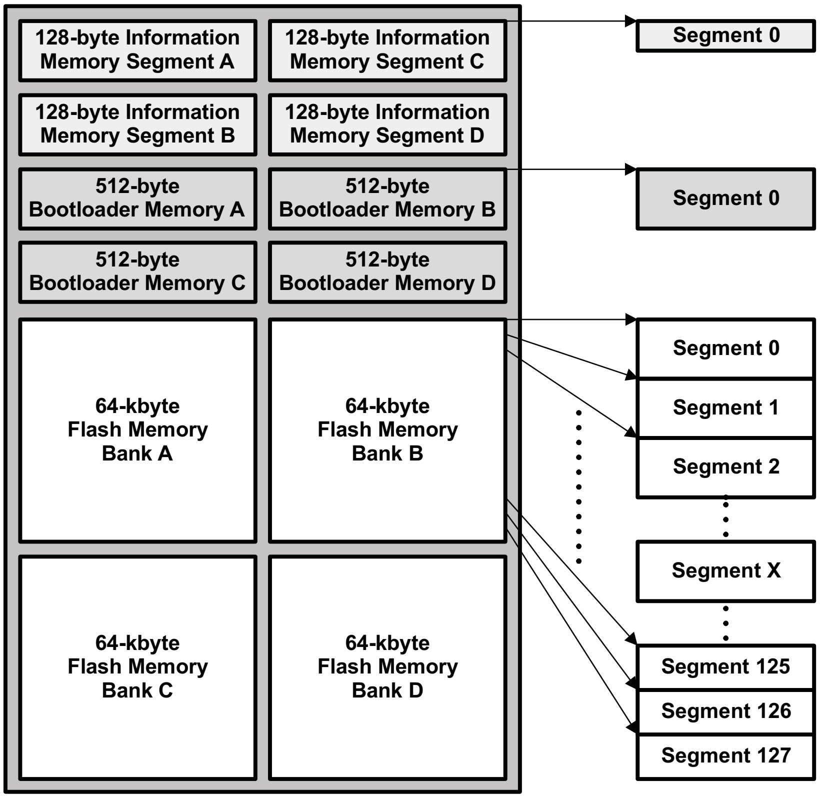

128-byte Information Memory Segment A
128-byte Information Memory Segment C
128-byte Information Memory Segment B
128-byte Information Memory Segment D
512-byte Bootloader Memory A
512-byte Bootloader Memory B
512-byte Bootloader Memory C
512-byte Bootloader Memory D
64-kbyte Flash Memory Bank A
64-kbyte Flash Memory Bank B
64-kbyte Flash Memory Bank C
64-kbyte Flash Memory Bank D
Segment 0
Segment 0
Segment 0
Segment 1
Segment 2
Segment X
Segment 125
Segment 126
Segment 127

Figure 6-2. Flash Memory Segments, 256-KB Example

6.2.1 Segment A

Segment A of the information memory is locked separately from all other segments with the LOCKA bit. If LOCKA = 1, segment A cannot be written or erased, and all information memory is protected from being segment erased. If LOCKA = 0, segment A can be erased and written like any other flash memory segment.

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

```
; Unlock Info Memory
   BIC      #FWKEY+LOCKINFO, &FCTL4  ; Clear LOCKINFO
; 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
; Lock Info Memory
   BIS      #FWKEY+LOCKINFO,&FCTL4   ; Set LOCKINFO
```

6.3 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.

Read and Fetch While Erase – The flash memory allows to execute a program from flash while a different flash bank is erased. Data reads are also possible from any flash bank not being erased.

Note: Read and Fetch While Erase

The read and fetch while erase feature is available in flash memory configurations where more than one flash bank is available. If there is one flash bank available, holding the complete flash program memory, the read from the program memory and information memory and bootstrap-loader memory during the erase is not provided.

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

- Byte/word/long-word (32-bit) write
- Block write
- Segment erase
- Bank erase (only main memory)
- Mass erase (all main memory banks)
- Read during bank erase (except for the one currently read from)

Reading or writing to flash memory while it is busy programming or erasing (page, mass or bank) from the same bank is prohibited. Any flash erase or programming can be initiated from within flash memory or RAM.

6.3.1 Erasing Flash Memory

The logical value of an erased 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 one segment. There are three erase modes selected by the ERASE and MERAS bits listed in Table 6-1.

Table 6-1. Erase Modes

MERAS	ERASE	Erase Mode
0	1	Segment erase
1	0	Bank erase (of one bank) selected by the dummy write address
1	1	Mass erase (all memory banks, information memory A to D and bootstrap loader segments A to D are not erased)

Erase Cycle

An erase cycle is initiated by a dummy write to the address range of the segment to be erased. The dummy write starts the erase operation. Figure 6-3 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 mass erase cycle timing is not dependent on the amount of flash memory present on a device. Erase cycle times are equivalent for all MSP430F5xx devices.

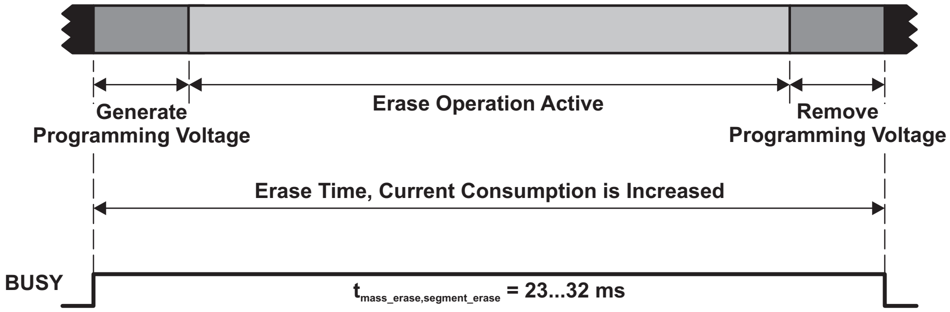

Generate
Programming Voltage
Erase Operation Active
Remove
Programming Voltage
Erase Time, Current Consumption is Increased
BUSY
$t_{mass\_erase,segment\_erase}$ = 23...32 ms

Figure 6-3. Erase Cycle Timing

Erasing Main Memory

The main memory consists of one or more banks. Each bank can be erased individually (bank erase). All main memory banks can be erased in the mass erase mode.

Erasing Information Memory or Flash Segments

The information memory A to D and the bootstrap loader segments A to D can be erased in segment erase mode. They are not erased during a bank erase or a mass erase.

Initiating Erase From Flash

An erase cycle can be initiated from within flash memory. Code can be executed from flash or RAM during a bank erase. The executed code cannot be located in a bank to be erased.

During a segment erase, the CPU is held until the erase cycle completes. After the erase cycle ends, 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 operation. If this occurs, CPU execution will be unpredictable after the erase cycle.

The flow to initiate an erase from flash is shown in Figure 6-4.

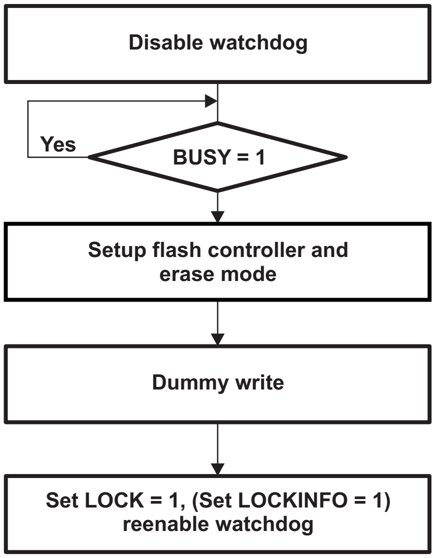

Disable watchdog
Yes
BUSY = 1
Setup flash controller and erase mode
Dummy write
Set LOCK = 1, (Set LOCKINFO = 1) reenable watchdog

Figure 6-4. Erase Cycle From Flash

```
; Segment Erase from flash.
; Assumes Program Memory. Information memory or BSL
; requires LOCKINFO to be cleared as well.
; Assumes ACCVIE = NMIIE = OFIE = 0.
    MOV   #WDTPW+WDTHOLD,&WDTCTL      ; Disable WDT
L1  BIT   #BUSY,&FCTL3                ; Test BUSY
    JNZ   L1                          ; Loop while busy
    MOV   #FWKEY,&FCTL3               ; Clear LOCK
    MOV   #FWKEY+ERASE,&FCTL1         ; Enable segment erase
    CLR   &0FC10h                     ; Dummy write
L2  BIT   #BUSY,&FCTL3                ; Test BUSY
    JNZ   L2                          ; Loop while busy
    MOV   #FWKEY+LOCK,&FCTL3          ; Done, set LOCK
    ...                               ; Re-enable WDT?
```

Initiating Erase From RAM

An erase cycle can be initiated from RAM. In this case, the CPU is not held and continues to execute code from RAM. The mass erase (all main memory banks) operation is initiated while executing from RAM. The BUSY bit is used to determine the end of the erase cycle. If the flash is busy completing a bank erase, flash addresses of a different bank can be used to read data or to fetch instructions. While the flash is BUSY, starting an erase cycle or a programming cycle causes an access violation, ACCIFG is set to 1, and the result of the erase operation is unpredictable.

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

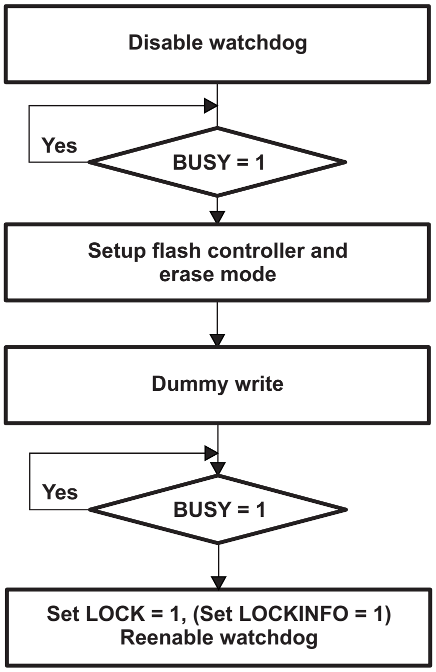

Disable watchdog
Yes
BUSY = 1
Setup flash controller and erase mode
Dummy write
Yes
BUSY = 1
Set LOCK = 1, (Set LOCKINFO = 1)
Reenable watchdog

Figure 6-5. Erase Cycle From RAM

```
; segment Erase from RAM.
; Assumes Program Memory. Information memory or BSL
; requires LOCKINFO to be cleared as well.
; Assumes ACCVIE = NMIIE = OFIE = 0.
    MOV   #WDTPW+WDTHOLD,&WDTCTL     ; Disable WDT
L1  BIT   #BUSY,&FCTL3               ; Test BUSY
    JNZ   L1                         ; Loop while busy
    MOV   #FWKEY,&FCTL3              ; Clear LOCK
    MOV   #FWKEY+ERASE,&FCTL1        ; Enable page erase
    CLR   &0FC10h                    ; Dummy write
L2  BIT   #BUSY,&FCTL3               ; Test BUSY
    JNZ   L2                         ; Loop while busy
    MOV   #FWKEY+LOCK,&FCTL3         ; Done, set LOCK
    ...                              ; Re-enable WDT?
```

6.3.2 Writing Flash Memory

The write modes, selected by the WRT and BLKWRT bits, are listed in Table 6-2.

Table 6-2. Write Modes

BLKWRT	WRT	Write Mode
0	1	Byte/word write
1	0	Long-word write
1	1	Long-word block write

The write modes use a sequence of individual write instructions. Using the long-word write mode is approximately twice as fast as the byte/word mode. Using the long-word block write mode is approximately four times faster than byte/word mode, because the voltage generator remains on for the complete block write, and long-words are written in parallel. Any instruction that modifies a destination can be used to modify a flash location in either byte/word write mode, long-word write mode, or block long-word write mode.

The BUSY bit is set while the 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 is set to 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 the CPU is held while the write completes. After the write completes, the CPU resumes code execution with the instruction following the write access. The byte/word write timing is shown in Figure 6-6.

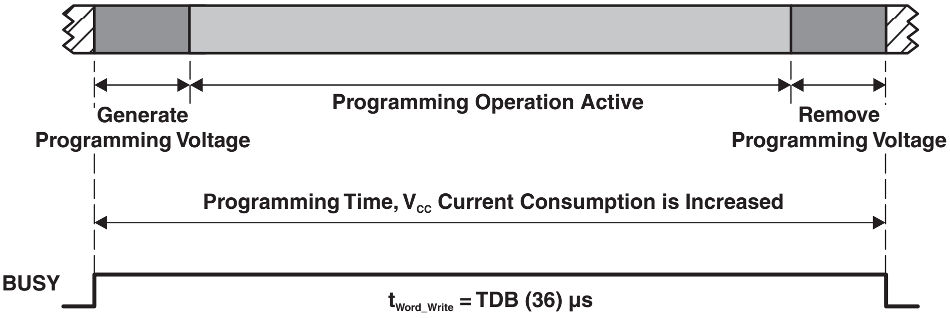

Generate
Programming Voltage
Programming Operation Active
Remove
Programming Voltage
Programming Time, V_{CC} Current Consumption is Increased
BUSY
t_{Word_Write} = TDB (36) µs

Figure 6-6. Byte/Word/Long-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 write mode, the internally-generated programming voltage is applied to the complete 128-byte block. The cumulative programming time, t_{CPT}, must not be exceeded for any block. Each byte or word write adds to the cumulative program time of a segment. If the maximum cumulative program time is reached or exceeded the segment must be erased. Further programming or using the data returns unpredictable results. See the device-specific data sheet for specifications.

Initiating Byte/Word Write From Flash

The flow to initiate a byte/word write from flash is shown in Figure 6-7.

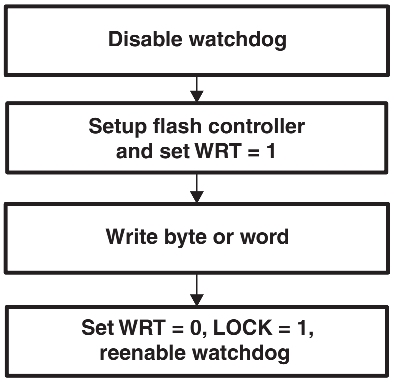

Disable watchdog
Setup flash controller and set WRT = 1
Write byte or word
Set WRT = 0, LOCK = 1, reenable watchdog

Figure 6-7. Initiating a Byte/Word Write From Flash

```
; Byte/word write from flash.
; Assumes 0x0FF1E is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
   MOV   #WDTPW+WDTHOLD,&WDTCTL     ; Disable WDT
   MOV   #FWKEY,&FCTL3              ; Clear LOCK
   MOV   #FWKEY+WRT,&FCTL1          ; Enable write
   MOV   #0123h,&0FF1Eh             ; 0123h -> 0x0FF1E
   MOV   #FWKEY,&FCTL1              ; Done. Clear WRT
   MOV   #FWKEY+LOCK,&FCTL3         ; Set LOCK
   ...                              ; Re-enable WDT?
```

Initiating Byte/Word Write From RAM

The flow to initiate a byte/word write from RAM is shown in Figure 6-8.

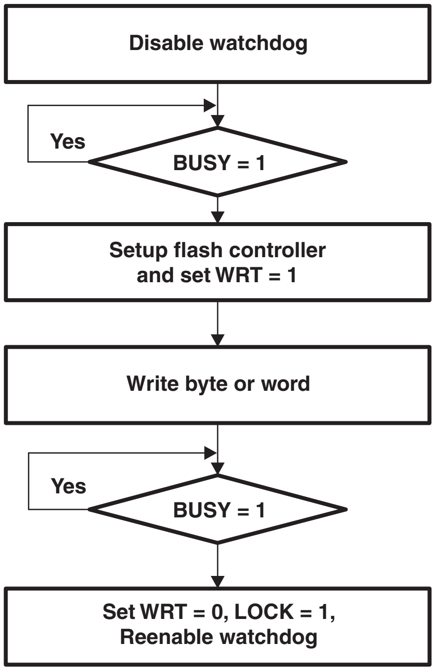

Disable watchdog
Yes
BUSY = 1
Setup flash controller and set WRT = 1
Write byte or word
Yes
BUSY = 1
Set WRT = 0, LOCK = 1, Reenable watchdog

Figure 6-8. Initiating a Byte/Word Write From RAM

```
; Byte/word write from RAM.
; Assumes 0x0FF1E 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,&FCTL3              ; Clear LOCK
    MOV   #FWKEY+WRT,&FCTL1          ; Enable write
    MOV   #0123h,&0FF1Eh             ; 0123h -> 0x0FF1E
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?
```

Long-Word Write

A long-word write operation can be initiated from within flash memory or from RAM. The BUSY bit is set to 1 after 32 bits are written to the flash controller and the programming cycle starts. When initiating from within flash memory, the CPU is held while the write completes. After the write completes, the CPU resumes code execution with the instruction following the write access. The long-word write timing is shown in Figure 6-6.

A long-word consists of four consecutive bytes aligned to at 32-bit address (only the lower two address bits are different). The bytes can be written in any order or any combination of bytes and words. If a byte or word is written more than once, the last data written to the four bytes are stored into the flash memory.

If a write to a flash address outside of the 32-bit address happens before all four bytes are available, the data written so far is discarded, and the latest byte/word written defines the new 32-bit aligned address.

When 32 bits are available, the write cycle is executed. When executing from RAM, the CPU continues to execute code. 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 long-word write mode, the internally-generated programming voltage is applied to a complete 128-byte block. The cumulative programming time, t_{CPT}, must not be exceeded for any block. Each byte or word write adds to the cumulative program time of a segment. If the maximum cumulative program time is reached or exceeded the segment must be erased. Further programming or using the data returns unpredictable results.

With each byte or word write, the amount of time the block is subjected to the programming voltage accumulates. If the cumulative programming time is reached or exceeded, the block must be erased before further programming or use. See the device-specific data sheet for specifications.

Initiating Long-Word Write From Flash

The flow to initiate a long-word write from flash is shown in Figure 6-9.

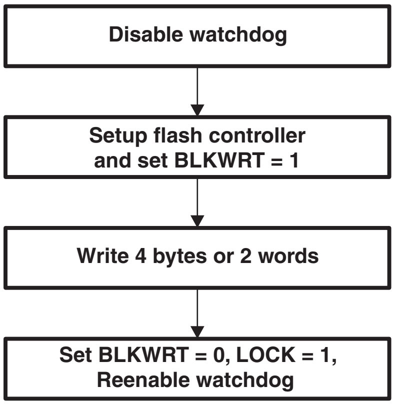

Disable watchdog
Setup flash controller and set BLKWRT = 1
Write 4 bytes or 2 words
Set BLKWRT = 0, LOCK = 1, Reenable watchdog

Figure 6-9. Initiating Long-Word Write From Flash

```
; Long-word write from flash.
; Assumes 0x0FF1C and 0x0FF1E is already erased
; Assumes ACCVIE = NMIIE = OFIE = 0.
   MOV   #WDTPW+WDTHOLD,&WDTCTL     ; Disable WDT
   MOV   #FWKEY,&FCTL3              ; Clear LOCK
   MOV   #FWKEY+BLKWRT,&FCTL1       ; Enable 2-word write
   MOV   #0123h,&0FF1Ch             ; 0123h -> 0x0FF1C
   MOV   #45676h,&0FF1Eh            ; 04567h -> 0x0FF1E
   MOV   #FWKEY,&FCTL1              ; Done. Clear BLKWRT
   MOV   #FWKEY+LOCK,&FCTL3         ; Set LOCK
   ...                              ; Re-enable WDT?
```

Initiating Long-Word Write From RAM

The flow to initiate a long-word write from RAM is shown in Figure 6-10.

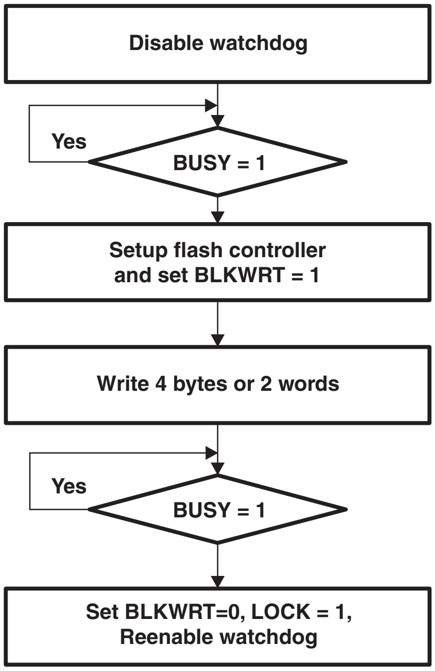

Disable watchdog
Yes
BUSY = 1
Setup flash controller and set BLKWRT = 1
Write 4 bytes or 2 words
Yes
BUSY = 1
Set BLKWRT=0, LOCK = 1, Reenable watchdog

Figure 6-10. Initiating Long-Word Write from RAM

```
; Two 16-bit word writes from RAM.
; Assumes 0x0FF1C and 0x0FF1E 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,&FCTL3              ; Clear LOCK
    MOV   #FWKEY+BLKWRT,&FCTL1       ; Enable write
    MOV   #0123h,&0FF1Ch             ; 0123h -> 0x0FF1C
    MOV   #4567h,&0FF1Eh             ; 4567h -> 0x0FF1E
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 128-byte row. The cumulative programming time t_{CPT} must not be exceeded for any row during a block write.

A block write cannot be initiated from within flash memory. The block write must be initiated from RAM. The BUSY bit remains set throughout the duration of the block write. The WAIT bit must be checked between writing four bytes, or two words to the block. When WAIT is set, then four bytes, or two 16-bit words of the block can be written. When writing successive blocks, the BLKWRT bit must be cleared after the current block is completed. 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 6-11 shows the block write timing.

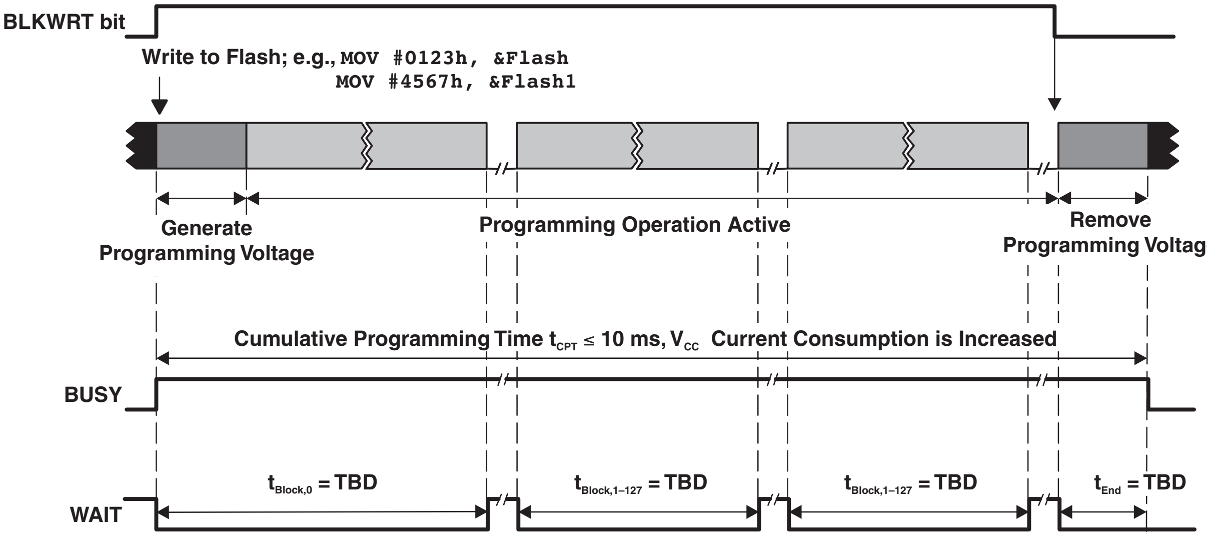

BLKWRT bit
Write to Flash; e.g., MOV #0123h, &Flash
MOV #4567h, &Flash1
Generate Programming Voltage
Programming Operation Active
Remove Programming Voltage
Cumulative Programming Time t_{CPT} ≤ 10 ms, V_{CC} Current Consumption is Increased
BUSY
WAIT
$t_{Block,0}$ = TBD
$t_{Block,1-127}$ = TBD
$t_{Block,1-127}$ = TBD
t_{End} = TBD

Figure 6-11. Block-Write Cycle Timing

Block Write Flow and Example

A block write flow is shown in Figure 6-12 and the following code example.

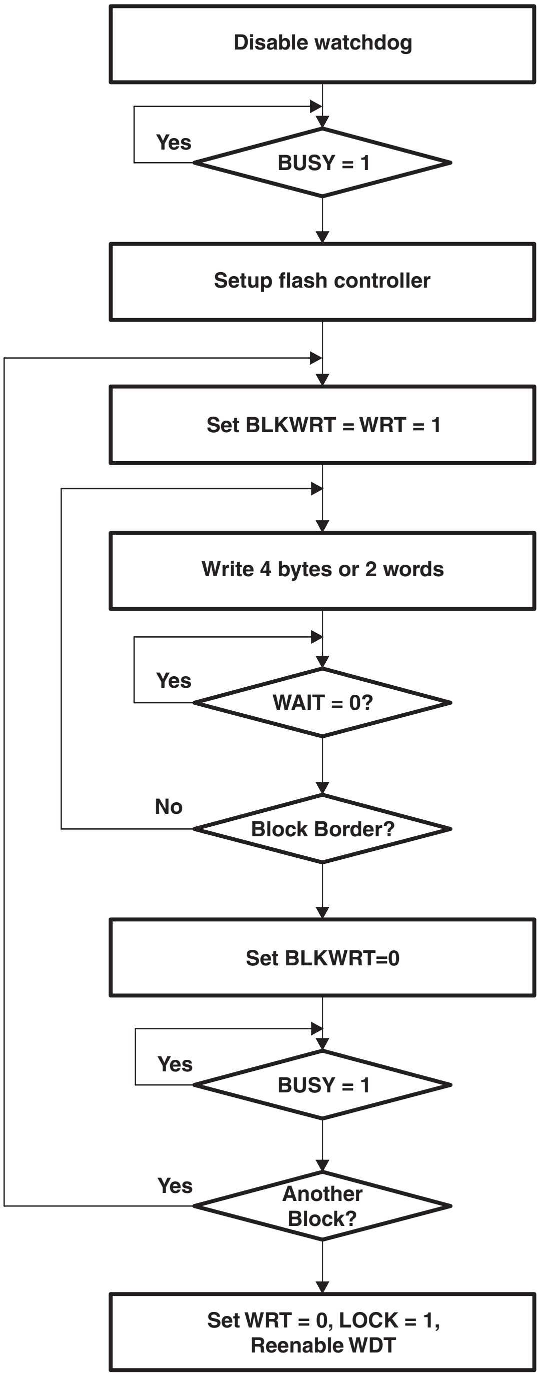


Figure 6-12. Block Write Flow

```
; Write one block starting at 0F000h.
; Must be executed from RAM, Assumes Flash is already erased.
; 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,&FCTL3               ; Clear LOCK
    MOV    #FWKEY+BLKWRT+WRT,&FCTL1    ; Enable block write
L2  MOV    Write_Value1,0(R6)          ; Write 1st location
    MOV    Write_Value2,2(R6)          ; Write 2nd word
L3  BIT    #WAIT,&FCTL3                ; Test WAIT
    JZ     L3                          ; Loop while WAIT=0
    INCD   R6                          ; Point to next words
    INCD   R6                          ; Point to next words
    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
```

6.3.3 Flash Memory Access During Write or Erase

When a write or an erase operation is initiated from RAM while BUSY = 1, the CPU may not write to any flash location. Otherwise, an access violation occurs, ACCVIFG is set, and the result is unpredictable.

When a write operation is initiated from within flash memory, the CPU continues code execution with the next instruction fetch after the write cycle completed (BUSY = 0).

The op-code 3FFFh 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 op-code and program execution resumes.

The flash access conditions while BUSY = 1 are listed in Table 6-3.

Table 6-3. Flash Access While the Flash is busy (BUSY = 1)

Flash Operation	Flash Access	WAIT	Result
Bank erase	Read	0	From the erased bank: ACCVIFG = 0. 03FFFh is the value read. From any other flash location: ACCVIFG = 0. Valid read.
	Write	0	ACCVIFG = 1. Write is ignored.
	Instruction fetch	0	From the erased bank: ACCVIFG = 0. CPU fetches 03FFFh. This is the JMP PC instruction. From any other flash location: ACCVIFG = 0. Valid instruction fetch.
Segment erase	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.
Word/byte write or long-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, block write is exited.
	Read	1	ACCVIFG = 0: 03FFFh is the value read.
	Write	1	ACCVIFG = 0, Valid write.
	Instruction fetch	1	ACCVIFG = 1, LOCK = 1, block write is exited.

Interrupts are automatically disabled during any flash operation.

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 reenabled.

6.3.4 Stopping 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 LOCK bit of FCTL3 is set. The result of the intended operation is unpredictable.

6.3.5 Checking Flash memory

The result of a programming cycle of the flash memory can be checked by calculating and storing a checksum (CRC) of parts and/or the complete flash memory content. The CRC module can be used for this purpose (see the device-specific data sheet). During the runtime of the system, the known checksums can be recalculated and compared with the expected values stored in the flash memory. The program checking the flash memory content is executed in RAM. To get an early indication of weak memory cells, reading the flash can be done in combination with the device-specific marginal read modes. The marginal read modes are controlled by the FCTL4.MRG0 and FCTL4.MRG1 register bits if available (device specific).

6.3.6 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 a 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/double-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 (this register is currently not implemented) when BUSY = 1 is an access violation.

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

6.3.7 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 reenabled after a flash write or erase, a set ACCVIFG flag generates 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.

6.3.8 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 VCC and RST/NMI.

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

Programming Flash Memory via 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 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 6-13. The user can choose to provide data to the MSP430 through any means available (UART, SPI, etc.). User-developed software can receive the data and program the flash memory. Since this type of solution is developed by the user, it can be completely customized to fit the application needs for programming, erasing, or updating the flash memory.

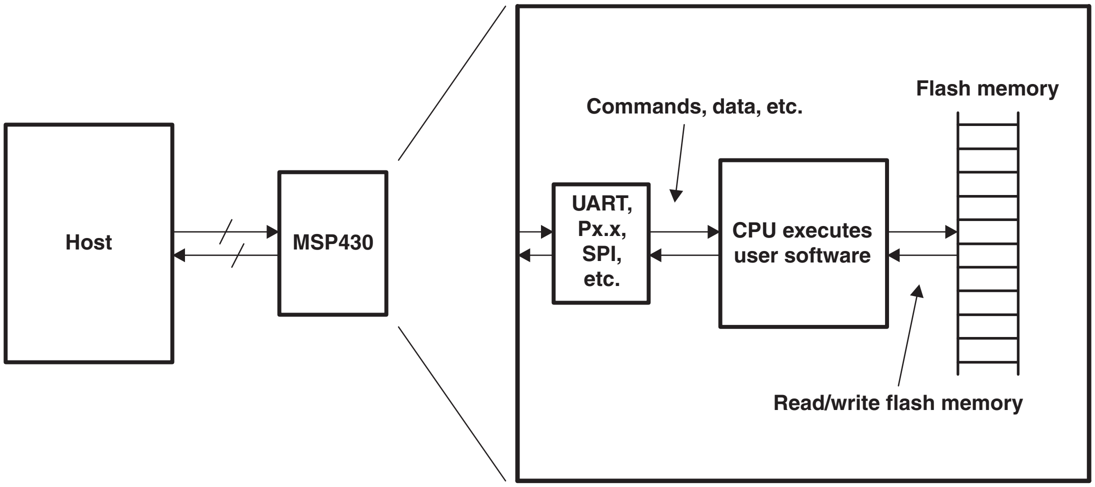


Figure 6-13. User-Developed Programming Solution

6.4 Flash Memory Registers

The flash memory registers are listed in Table 6-4. The base address can be found in the device-specific data sheet. The address offset is given in Table 6-4.

Table 6-4. Flash Controller Registers

Register	Short Form	Register Type	Address	Initial State
Flash memory control register 1	FCTL1	Read/write	0000h	9600h
Flash memory control register 3	FCTL3	Read/write	0004h	9658h
Flash memory control register 4	FCTL4	Read/write	0006h	9600h
Interrupt Enable 1	IE1	Read/write	000Ah	0000h
Interrupt Flag 1	IFG1	Read/write	000Ch	0000h

FCTL1, Flash Memory Control Register 1

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

7	6	5	4	3	2	1	0
BLKWRT	**WRT**	**SWRT**	**Reserved**	**Reserved**	**MERAS**	**ERASE**	**Reserved**
rw-0	rw-0	rw-0	r-0	r-0	rw-0	rw-0	r-0

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

BLKWRT Bit 7 See following table.

WRT Bit 6 See following table.

BLKWRT	WRT	Write Mode
0	1	Byte/word write
1	0	Long-word write
1	1	Long-word block write

SWRT Bit 5 Smart write. If this bit is set the program time is shortened. The programming quality has to be checked by marginal read modes.

Reserved Bits 4-3 Reserved. Must be written to 0. Always read 0.

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	Segment erase
1	0	Bank erase (of one bank)
1	1	Mass erase (Erase all flash memory banks)

Reserved Bit 0 Reserved. Always read 0.

FCTL3, Flash Memory Control Register 3

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

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

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

Reserved Bit 7 Reserved. Always read 0.

LOCKA Bit 6 Segment A lock. Write a 1 to this bit to change its state. Writing 0 has no effect.

0 Segment A, B, C, D are unlocked. and are erased during a mass erase.

1 Segment A of the information memory is write protected. Segment B, C, and D are protected from all erase.

EMEX Bit 5 Emergency exit. Setting this bit stops any erase or write operation. The LOCK bit is set.

0 No emergency exit

1 Emergency exit

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

0 Unlocked

1 Locked

WAIT Bit 3 Wait. Indicates the flash memory is being written to.

0 The flash memory is not ready for the next byte/word write.

1 The flash memory is ready for the next byte/word write.

ACCVIFG Bit 2 Access violation interrupt flag

0 No interrupt pending

1 Interrupt pending

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

0 FCTLx password was written correctly

1 FCTLx password was written incorrectly

BUSY Bit 0 Busy. This bit indicates if the flash is currently busy erasing or programming.

0 Not busy

1 Busy

FCTL4, Flash Memory Control Register 4

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
LOCKINFO	**Reserved**	**MRG1**	**MRG0**	**Reserved**			**VPE**
rw-0	r-0	rw-0	rw-0	r-0	r-0	r-0	rw-0

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

LOCKINFO Bit 7 Lock information memory. If set the information memory cannot be erased in segment erase mode and cannot be written to.

Reserved Bit 6 Reserved. Always read as 0.

MRG1 Bit 5 Marginal read 1 mode. This bit enables the marginal 1 read mode. The marginal read 1 bit is valid for reads from the flash memory only. During a fetch cycle the marginal mode is turned off automatically. If both MRG1 and MRG0 are set MRG1 is active and MRG0 is ignored.

0 Marginal 1 read mode is disabled.

1 Marginal 1 read mode is enabled.

MRG0 Bit 4 Marginal read 0 mode. This bit enables the marginal 0 read mode. The marginal read 1 bit is valid for reads from the flash memory only. During a fetch cycle the marginal mode is turned off automatically. If both MRG1 and MRG0 are set MRG1 is active and MRG0 is ignored.

0 Marginal 0 read mode is disabled.

1 Marginal 0 read mode is enabled.

Reserved Bit 3–1 Reserved. Always read as 0.

VPE Bit 0 Voltage changed during program error. This bit is set by hardware and can only be cleared by software. If DVCC changed significantly during programming, this bit is set to indicate an invalid result. The ACCVIFG bit is set if VPE is set.

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 the device-specific data sheet.

ACCVIE Bit 5 Flash memory access violation interrupt enable. This bit enables the ACCVIFG interrupt. Because other bits in IE1 may be used for other modules, it is recommended to set or clear this bit using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions.

0 Interrupt not enabled

1 Interrupt enabled

Chapter 7

SLAU208–June 2008

Digital I/O

This chapter describes the operation of the digital I/O ports. The digital I/O ports are implemented in all MSP430x5xx devices.

Topic **Page**

7.1 Digital I/O Introduction

MSP430x5xx devices may have up to 12 digital I/O ports implemented, P1 to P11 and PJ. Most ports have eight I/O pins, however some ports may contain less. See the device-specific data sheet for ports available. Each I/O pin is individually configurable for input or output direction, and each I/O line can be individually read or written to. All ports have individually configurable pullup or pulldown resistors, as well as, configurable drive strength.

Ports P1 and P2 always 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 P1IV, and all P2 I/O lines source a different, single interrupt vector P2IV. On some MSP430x5xx devices, additional ports with interrupt capability may be available. Please refer to the device specific datasheet for details.

Individual ports can be accessed as byte wide ports or can be combined into word wide ports and accessed via word formats. Port pairs P1/P2, P3/P4, P5/P6, P7/P8, etc. are associated with the names PA, PB, PC, PD, etc., respectively. When writing to port PA with word operations, all 16 bits are written to the port. Writing to the lower byte of the PA port using byte operations, the upper byte remains unchanged. Similarly, writing to the upper byte of the PA port using byte instructions leaves the lower byte unchanged. Similarly for other ports. Writing to a port that contains less than the maximum number of bits possible, the unused bits are a "do not care". All port registers are handled in this manner with this naming convention except for the interrupt vector registers, P1IV and P2IV. These are word accessible only, and PAIV does not exist.

Reading of the PA port using word operations causes all 16 bits to be transferred to the destination. Reading the lower or upper byte of the PA port (P1 or P2) and storing to memory using byte operations causes only the lower or upper byte to be transferred to the destination, respectively. Reading of the PA port and storing to a general purpose register using byte operations causes the byte transferred to be written to the least significant byte of the register. The upper significant byte of the destination register will be cleared automatically. Ports PB, PC, PD, and PE behave similarly. When reading from ports that contain less than the maximum bits possible, unused bits are read as zeros. Similarly, for Port PJ.

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 pullup or pulldown resistors

7.2 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.

7.2.1 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. These registers are read only.

- 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.

7.2.2 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.

- Bit = 0: The output is low
- Bit = 1: The output is high

If the pin is configured as I/O function, input direction and the pullup/pulldown resistor is enabled, the corresponding bit in the PxOUT register selects pullup or pulldown.

- Bit = 0: The pin is pulled down
- Bit = 1: The pin is pulled up

7.2.3 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: Port pin is switched to input direction
- Bit = 1: Port pin is switched to output direction

7.2.4 Pullup/Pulldown Resistor Enable Registers PxREN

Each bit in each PxREN register enables or disables the pullup/pulldown 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: Pullup/pulldown resistor disabled
- Bit = 1: Pullup/pulldown resistor enabled

Table 7-1 summarizes the usage of PxDIRx, PxRENx, and PxOUTx for proper I/O configuration.

Table 7-1. I/O Configuration

PxDIRx	PxRENx	PxOUTx	I/O Configuration
0	0	x	Input
0	1	0	Input with pulldown resistor
0	1	1	Input with pullup resistor
1	x	x	Output

7.2.5 Output Drive Strength Registers PxDS

Each bit in each PxDS register selects either full drive or reduced drive strength. Default is reduced drive strength.

- Bit = 0: Reduced drive strength
- Bit = 1: Full drive strength

Note: Drive Strength and EMI

All outputs default to reduced drive strength to reduce EMI. Using full drive strength can result in increased EMI.

7.2.6 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 PxSELx 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.

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

When any PxSEL 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.

7.2.7 P1 and P2 Interrupts

Each pin in ports P1 and P2 have interrupt capability, configured with the PxIFG, PxIE, and PxIES registers. All P1 interrupt flags are prioritized, with P1IFG.0 being the highest, and combined to source a single interrupt vector. The highest priority enabled interrupt generates a number in the P1IV register. This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled P1 interrupts do not affect the P1IV value. The same functionality exists for P2. The PxIV registers are word access only.

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. 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, PxDIR, or PxREN

Writing to P1OUT, P1DIR, P1REN, P2OUT, P2DIR, or P2REN can result in setting the corresponding P1IFG or P2IFG flags.

Any access, read or write, of the P1IV 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, assume that P1IFG.0 has the highest priority. If the P1IFG.0 and P1IFG.2 flags are set when the interrupt service routine accesses the P1IV register, P1IFG.0 is reset automatically. After the RETI instruction of the interrupt service routine is executed, the P1IFG.2 will generate another interrupt.

Port P2 interrupts behave similarly, and source a separate single interrupt vector and utilizes the P2IV register.

P1IV, P2IV Software Example

The following software example shows the recommended use of P1IV and the handling overhead. The P1IV value is added to the PC to automatically jump to the appropriate routine. The P2IV is similar.

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.

```
;Interrupt handler for P1IFGx                              Cycles
P1_HND   ...                 ; Interrupt latency             6
         ADD      &P1IV,PC   ; Add offset to Jump table      3
         RETI                ; Vector  0: No interrupt       5
         JMP      P1_0_HND   ; Vector  2: Port 1 bit 0       2
         JMP      P1_1_HND   ; Vector  4: Port 1 bit 1       2
         JMP      P1_2_HND   ; Vector  6: Port 1 bit 2       2
         JMP      P1_3_HND   ; Vector  8: Port 1 bit 3       2
         JMP      P1_4_HND   ; Vector 10: Port 1 bit 4       2
         JMP      P1_5_HND   ; Vector 12: Port 1 bit 5       2
         JMP      P1_6_HND   ; Vector 14: Port 1 bit 6       2
         JMP      P1_7_HND   ; Vector 16: Port 1 bit 7       2

P1_7_HND                     ; Vector 16: Port 1 bit 7
         ...                 ; Task starts here
         RETI                ; Back to main program          5

P1_6_HND                     ; Vector 14: Port 1 bit 6
         ...                 ; Task starts here
         RETI                ; Back to main program          5

P1_5_HND                     ; Vector 12: Port 1 bit 5
         ...                 ; Task starts here
         RETI                ; Back to main program          5

P1_4_HND                     ; Vector 10: Port 1 bit 4
         ...                 ; Task starts here
         RETI                ; Back to main program          5

P1_3_HND                     ; Vector 8: Port 1 bit 3
         ...                 ; Task starts here
         RETI                ; Back to main program          5

P1_2_HND                     ; Vector 6: Port 1 bit 2
         ...                 ; Task starts here
         RETI                ; Back to main program          5

P1_1_HND                     ; Vector 4: Port 1 bit 1
         ...                 ; Task starts here
         RETI                ; Back to main program          5
P1_0_HND                     ; Vector 2: Port 1 bit 0
         ...                 ; Task starts here
         RETI                ; Back to main program          5
```

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 → 1	0	May be set
0 → 1	1	Unchanged
1 → 0	0	Unchanged
1 → 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

7.2.8 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 pullup/pulldown 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 of unused pins.

Note: Configuring Port J and Shared JTAG pins:

It is important to remember in the application to take special precautions to ensure that the Port J is configured properly to prevent any floating input. Since Port PJ is shared with the JTAG function, floating inputs may not be noticed when in an emulation environment . Port J is initialized to high impedance inputs by default.

7.3 Digital I/O Registers

The digital I/O registers are listed in Table 7-2. The base addresses can be found in the device specific datasheet Each port grouping begins at its base address. The address offsets are given in Table 7-2.

Table 7-2. Digital I/O Registers

Port	Register	Short Form	Address Offset	Register Type	Initial State
P1	P1 Interrupt Vector	P1IV	0Eh	Read only	0000h
P2	P2 Interrupt Vector	P2IV	1Eh	Read only	0000h
P1	Input	P1IN	00h	Read only	
	Output	P1OUT	02h	Read/write	Unchanged
	Direction	P1DIR	04h	Read/write	00h
	Resistor Enable	P1REN	06h	Read/write	00h
	Output drive strength	P1DS	08h	Read/write	00h
	Port Select	P1SEL	0Ah	Read/write	00h
	Interrupt Edge Select	P1IES	18h	Read/write	Unchanged
	Interrupt Enable	P1IE	1Ah	Read/write	00h
	Interrupt Flag	P1IFG	1Ch	Read/write	00h
P2	Input	P2IN	01h	Read only	
	Output	P2OUT	03h	Read/write	Unchanged
	Direction	P2DIR	05h	Read/write	00h
	Resistor Enable	P2REN	07h	Read/write	00h
	Output drive strength	P2DS	09h	Read/write	00h
	Port Select	P2SEL	0Bh	Read/write	00h
	Interrupt Edge Select	P2IES	19h	Read/write	Unchanged
	Interrupt Enable	P2IE	1Bh	Read/write	00h
	Interrupt Flag	P2IFG	1Dh	Read/write	00h
P3	Input	P3IN	00h	Read only	
	Output	P3OUT	02h	Read/write	Unchanged
	Direction	P3DIR	04h	Read/write	00h
	Resistor Enable	P3REN	06h	Read/write	00h
	Output drive strength	P3DS	08h	Read/write	00h
	Port Select	P3SEL	0Ah	Read/write	00h
P4	Input	P4IN	01h	Read only	
	Output	P4OUT	03h	Read/write	Unchanged
	Direction	P4DIR	05h	Read/write	00h
	Resistor Enable	P4REN	07h	Read/write	00h
	Output drive strength	P4DS	09h	Read/write	00h
	Port Select	P4SEL	0Bh	Read/write	00h
P5	Input	P5IN	00h	Read only	
	Output	P5OUT	02h	Read/write	Unchanged
	Direction	P5DIR	04h	Read/write	00h
	Resistor Enable	P5REN	06h	Read/write	00h
	Output drive strength	P5DS	08h	Read/write	00h
	Port Select	P5SEL	0Ah	Read/write	00h

Table 7-2. Digital I/O Registers (continued)

Port	Register	Short Form	Address Offset	Register Type	Initial State
P6	Input	P6IN	01h	Read only	
	Output	P6OUT	03h	Read/write	Unchanged
	Direction	P6DIR	05h	Read/write	00h
	Resistor Enable	P6REN	07h	Read/write	00h
	Output drive strength	P6DS	09h	Read/write	00h
	Port Select	P6SEL	0Bh	Read/write	00h
P7	Input	P7IN	00h	Read only	
	Output	P7OUT	02h	Read/write	Unchanged
	Direction	P7DIR	04h	Read/write	00h
	Resistor Enable	P7REN	06h	Read/write	00h
	Output drive strength	P7DS	08h	Read/write	00h
	Port Select	P7SEL	0Ah	Read/write	00h
P8	Input	P8IN	01h	Read only	
	Output	P8OUT	03h	Read/write	Unchanged
	Direction	P8DIR	05h	Read/write	00h
	Resistor Enable	P8REN	07h	Read/write	00h
	Output drive strength	P8DS	09h	Read/write	00h
	Port Select	P8SEL	0Bh	Read/write	00h
P9	Input	P9IN	00h	Read only	
	Output	P9OUT	02h	Read/write	Unchanged
	Direction	P9DIR	04h	Read/write	00h
	Resistor Enable	P9REN	06h	Read/write	00h
	Output drive strength	P9DS	08h	Read/write	00h
	Port Select	P9SEL	0Ah	Read/write	00h
P10	Input	P10IN	01h	Read only	
	Output	P10OUT	03h	Read/write	Unchanged
	Direction	P10DIR	05h	Read/write	00h
	Resistor Enable	P10REN	07h	Read/write	00h
	Output drive strength	P10DS	09h	Read/write	00h
	Port Select	P10SEL	0Bh	Read/write	00h
P11	Input	P11IN	00h	Read only	
	Output	P11OUT	02h	Read/write	Unchanged
	Direction	P11DIR	04h	Read/write	00h
	Resistor Enable	P11REN	06h	Read/write	00h
	Output drive strength	P11DS	08h	Read/write	00h
	Port Select	P11SEL	0Ah	Read/write	00h
PJ	Input	PJIN	00h	Read only	
	Output	PJOUT	02h	Read/write	Unchanged
	Direction	PJDIR	04h	Read/write	00h
	Resistor Enable	PJREN	06h	Read/write	00h
	Output drive strength	PJDS	08h	Read/write	00h

P1IV, Port 1 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	**P1IVx**					**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

P1IVx Bits 15-0 Port 1 interrupt vector value

P1IVx Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	Port 1.0 interrupt	P1IFG.0	Highest
04h	Port 1.1 interrupt	P1IFG.1	
06h	Port 1.2 interrupt	P1IFG.2	
08h	Port 1.3 interrupt	P1IFG.3	
0Ah	Port 1.4 interrupt	P1IFG.4	
0Ch	Port 1.5 interrupt	P1IFG.5	
0Eh	Port 1.6 interrupt	P1IFG.6	
10h	Port 1.7 interrupt	P1IFG.7	Lowest

P2IV, Port 2 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	**P2IVx**					**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

P2IVx Bits 15-0 Port 2 interrupt vector value

P2IVx Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	Port 2.0 interrupt	P2IFG.0	Highest
04h	Port 2.1 interrupt	P2IFG.1	
06h	Port 2.2 interrupt	P2IFG.2	
08h	Port 2.3 interrupt	P2IFG.3	
0Ah	Port 2.4 interrupt	P2IFG.4	
0Ch	Port 2.5 interrupt	P2IFG.5	
0Eh	Port 2.6 interrupt	P2IFG.6	
10h	Port 2.7 interrupt	P2IFG.7	Lowest

P1IES Port 1 Interrupt Edge Select Register

7	6	5	4	3	2	1	0
P1IES							
rw	rw	rw	rw	rw	rw	rw	rw

P1IES Bits 7-0 Port 1 interrupt edge select

0 P1IFGx flag is set with a low-to-high transition

1 P1IFGx flag is set with a high-to-low transition

P1IE, Port 1 Interrupt Enable Register

7	6	5	4	3	2	1	0
P1IE							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

P1IE Bits 7-0 Port 1 interrupt enable

0 Corresponding port interrupt disabled

1 Corresponding port interrupt enabled

P1IFG, Port 1 Interrupt Flag Register

7	6	5	4	3	2	1	0
P1IFG							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

P1IFG Bits 7-0 Port 1 interrupt flag

0 No interrupt is pending

1 Interrupt is pending

P2IES Port 2 Interrupt Edge Select Register

7	6	5	4	3	2	1	0
P2IES							
rw	rw	rw	rw	rw	rw	rw	rw

P2IES Bits 7-0 Port 2 interrupt edge select

0 P2IFGx flag is set with a low-to-high transition

1 P2IFGx flag is set with a high-to-low transition

P2IE, Port 2 Interrupt Enable Register

7	6	5	4	3	2	1	0
P2IE							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

P2IE Bits 7-0 Port 2 interrupt enable

0 Corresponding port interrupt disabled

1 Corresponding port interrupt enabled

P2IFG, Port 2 Interrupt Flag Register

7	6	5	4	3	2	1	0
P2IFG							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

P2IFG Bits 7-0 Port 2 interrupt flag

0 No interrupt is pending

1 Interrupt is pending

PxIN, Port x Input Register

7	6	5	4	3	2	1	0
PxIN							
r	r	r	r	r	r	r	r

PxIN Bits 7-0 Port x input. Read only.

PxOUT, Port x Output Register

7	6	5	4	3	2	1	0
PxOUT							
rw	rw	rw	rw	rw	rw	rw	rw

PxOUT Bits 7-0 Port x output
When I/O configured to output mode:

0 The output is low

1 The output is high

When I/O configured to input mode and pullups/pulldowns enabled:

0 Pull-down selected

1 Pullup selected

PxDIR, Port x Direction Register

7	6	5	4	3	2	1	0
PxDIR							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

PxDIR Bits 7-0 Port x direction

0 Port configured as input

1 Port configured as output

PxREN, Port x Resistor Enable Register

7	6	5	4	3	2	1	0
PxREN							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

PxREN Bits 7-0 Port x pullup/pulldown resistor enable

0 Pullup/pulldown disabled

1 Pullup/pulldown enabled

PxDS, Port x Drive Strength Register

7	6	5	4	3	2	1	0
PxDS							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

PxDS Bits 7-0 Port x drive strength

0 Reduced output drive strength

1 Full output drive strength

Chapter 8

SLAU208–June 2008

RAM Controller

The RAM Controller (RAMCTL) allows control of the operation of the RAM.

Topic **Page**

8.1 RAMCTL Introduction

The RAMCTL provides access to the different power modes of the RAM. The RAMCTL allows the ability to reduce the leakage current while the CPU is off. The RAM can also be switched off. In retention mode the RAM content is saved while the RAM content is lost in off mode. The RAM is partitioned in sectors, typically of 4k-byte (sector) size. Please refer to the device specific datasheet for actual block allocation and size. Each sector is controlled by the RAM Controller RAM Sector Off control bit (RCRSyOFF) of the RAMCTL control register 0 (RCCTL0). The RCCTL0 register is password protected. Only if the correct password is written during a word write, the RCCTL0 register content can be modified. Byte write accesses or write accesses with a wrong password are ignored.

8.2 RAMCTL Operation

Active Mode

In active mode the RAM can be read and written at any time. If a RAM address of a sector needs to hold data the whole sector cannot be switched off.

Low-Power Modes

In all low-power modes, the CPU is switched off. As soon as the CPU is switched off, the RAM enters retention mode to reduce the leakage current.

RAM Off Mode

Each sector can be turned off independently of each other by setting the respective RCRSyOFF bit to 1. Reading from a switched off RAM sector returns 0 as data. All data previously stored into a switched off RAM sector is lost and cannot be read, even if the sector is turned on again.

Stack pointer

The program stack is located in RAM. Sectors holding the stack must not be turned off if an interrupt has to be executed or a low-power mode is entered.

8.3 RAMCTL Module Registers

The RAMCTL module register is listed in Table 8-1. The base address can be found in the device specific datasheet. The address offset is given in Table 8-1.

Table 8-1. RAMCTL Module Register

Register	Short Form	Register Type	Address	Initial State
RAMCTL control register 0	RCCTL0	Read/write	0000h	0000h

RCCTL0, RAM Controller Control Register 0

15	14	13	12	11	10	9	8
RCKEYx Always reads as 69h Must be written as 5Ah							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
Reserved				**RCRS3OFF**	**RCRS2OFF**	**RCRS1OFF**	**RCRS0OFF**
r-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0

RCKEY Bits 15-8 RAM Controller Key. Always read as 69h. Must be written as 5Ah, otherwise the RAM controller write is ignored.

Reserved Bits 7-4 Reserved. Always read as 0.

RCRSyOFF Bits 3-0 RAM Controller RAM Sector y Off. Setting the bit to 1 turns off the RAM sector y. All data of the RAM sector y is lost. See the device specific datasheet to find the address range and size of each RAM sector.

Chapter 9

SLAU208–June 2008

DMA Controller

The DMA controller module transfers data from one address to another without CPU intervention. This chapter describes the operation of the DMA controller that is available on all MSP430x5xx devices.

Topic Page

9.1 DMA Introduction

The direct memory access (DMA) controller transfers data from one address to another, without CPU intervention, across the entire address range. For example, the DMA controller can move data from the ADC12_A conversion memory to RAM.

Devices that contain a DMA controller may have up to eight DMA channels available. Therefore, depending on the number of DMA channels available, some features described in this chapter are not applicable to all devices.

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

The DMA controller features include:

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

The DMA controller block diagram is shown in Figure 9-1.

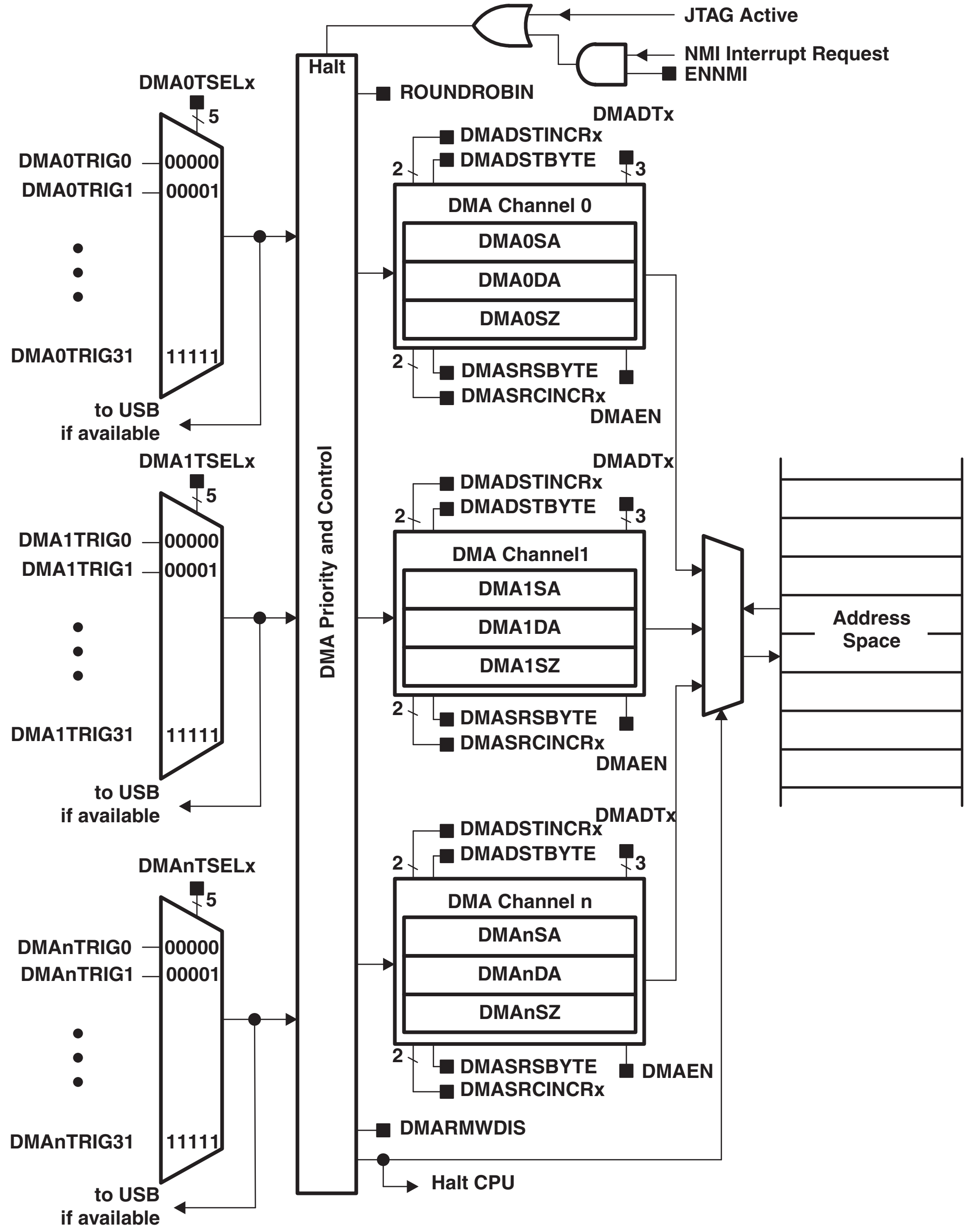


Figure 9-1. DMA Controller Block Diagram

9.2 DMA Operation

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

9.2.1 DMA Addressing Modes

The DMA controller has four addressing modes. The addressing mode for each DMA channel is independently configurable. For example, channel 0 may transfer between two fixed addresses, while channel 1 transfers between two blocks of addresses. The addressing modes are shown in Figure 9-2. The addressing modes are:

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

The addressing modes are configured with the DMASRCINCRx and DMADSTINCRx control bits. The DMASRCINCRx bits select if the source address is incremented, decremented, or unchanged after each transfer. The DMADSTINCRx bits select if the destination address is incremented, decremented, or unchanged after each transfer.

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

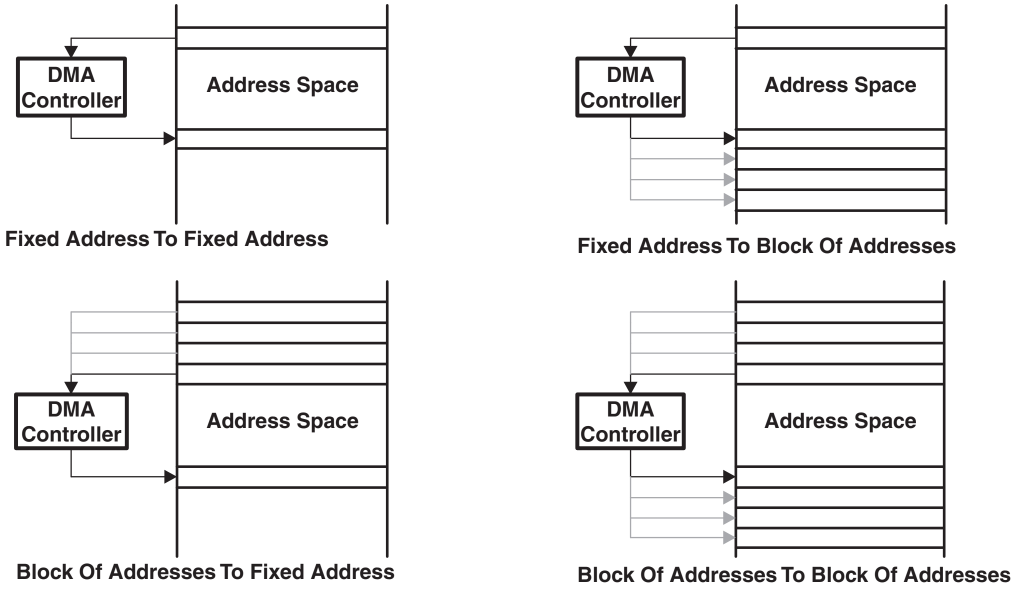

DMA Controller
Address Space
Fixed Address To Fixed Address
DMA Controller
Address Space
Fixed Address To Block Of Addresses
DMA Controller
Address Space
Block Of Addresses To Fixed Address
DMA Controller
Address Space
Block Of Addresses To Block Of Addresses

Figure 9-2. DMA Addressing Modes

9.2.2 DMA Transfer Modes

The DMA controller has six transfer modes selected by the DMADTx bits as listed in Table 9-1. Each channel is individually configurable for its transfer mode. For example, channel 0 may be configured in single transfer mode, while channel 1 is configured for burst-block transfer mode, and channel 2 operates in repeated block mode. The transfer mode is configured independently from the addressing mode. Any addressing mode can be used with any transfer mode.

Two types of data can be transferred selectable by the DMAxCTL DSTBYTE and SRCBYTE fields. The source and/or destination location can be either byte or word data. It is also possible to transfer byte to byte, word to word or any combination.

Table 9-1. DMA Transfer Modes

DMADTx	Transfer Mode	Description
000	Single transfer	Each transfer requires a trigger. DMAEN is automatically cleared when DMAxSZ transfers have been made.
001	Block transfer	A complete block is transferred with one trigger. DMAEN is automatically cleared at the end of the block transfer.
010, 011	Burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN is automatically cleared at the end of the burst-block transfer.
100	Repeated single transfer	Each transfer requires a trigger. DMAEN remains enabled.
101	Repeated block transfer	A complete block is transferred with one trigger. DMAEN remains enabled.
110, 111	Repeated burst-block transfer	CPU activity is interleaved with a block transfer. DMAEN remains enabled.

Single Transfer

In single transfer mode, each byte/word transfer requires a separate trigger. The single transfer state diagram is shown in Figure 9-3.

The DMAxSZ register is used to define the number of transfers to be made. The DMADSTINCRx and DMASRCINCRx bits select if the destination address and the source address are incremented or decremented after each transfer. If DMAxSZ = 0, no transfers occur.

The DMAxSA, DMAxDA, and DMAxSZ registers are copied into temporary registers. The temporary values of DMAxSA and DMAxDA are incremented or decremented after each transfer. The DMAxSZ register is decremented after each transfer. When the DMAxSZ register decrements to zero it is reloaded from its temporary register and the corresponding DMAIFG flag is set. When DMADTx = 0, the DMAEN bit is cleared automatically when DMAxSZ decrements to zero and must be set again for another transfer to occur.

In repeated single transfer mode, the DMA controller remains enabled with DMAEN = 1, and a transfer occurs every time a trigger occurs.

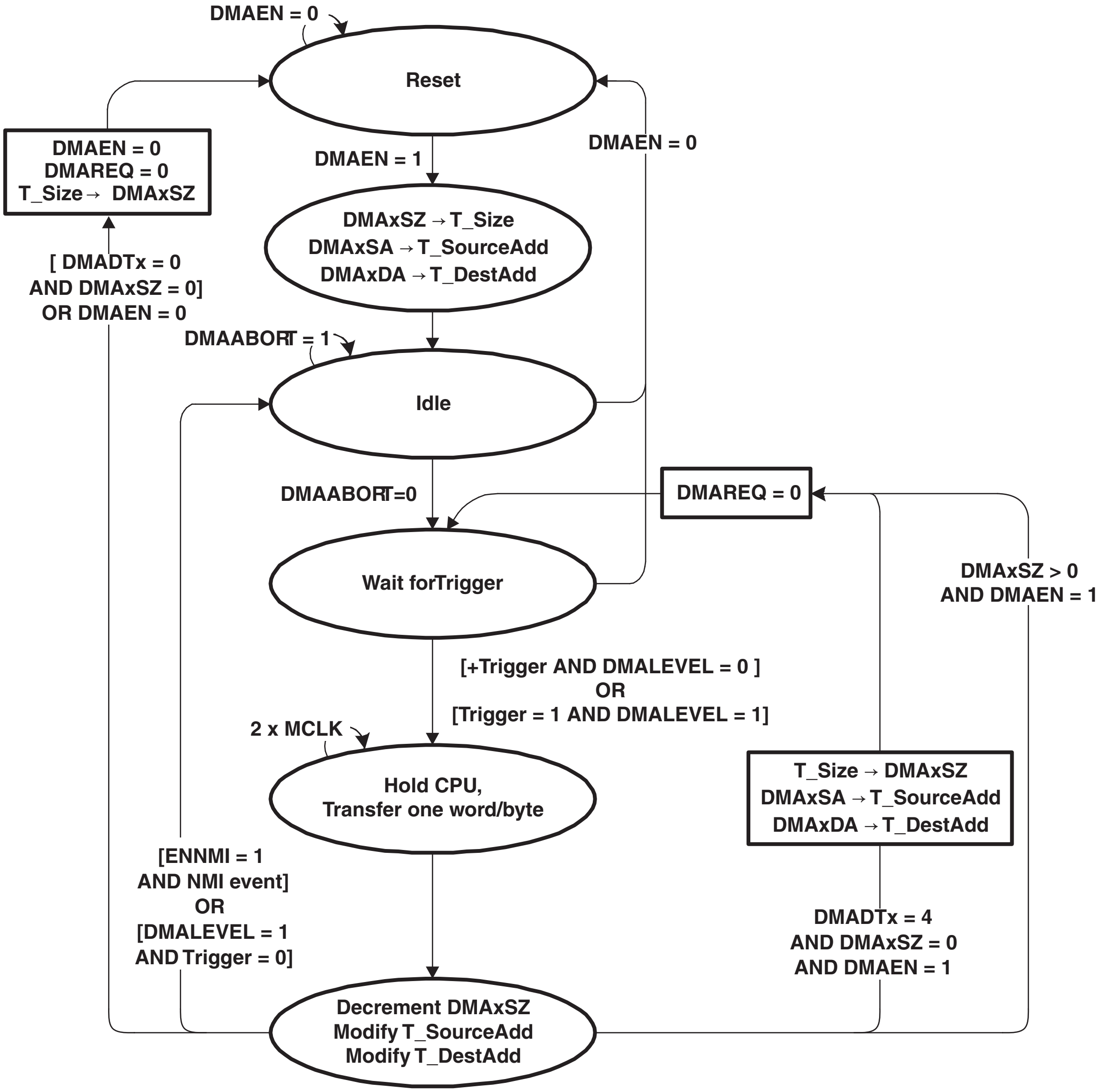


Figure 9-3. DMA Single Transfer State Diagram

Block Transfers

In block transfer mode, a transfer of a complete block of data occurs after one trigger. When DMADTx = 1, the DMAEN bit is cleared after the completion of the block transfer and must be set again before another block transfer can be triggered. After a block transfer has been triggered, further trigger signals occurring during the block transfer are ignored. The block transfer state diagram is shown in Figure 9-4.

The DMAxSZ register is used to define the size of the block and the DMADSTINCRx and DMASRCINCRx bits select if the destination address and the source address are incremented or decremented after each transfer of the block. If DMAxSZ = 0, no transfers occur.

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

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

In repeated block transfer mode, the DMAEN bit remains set after completion of the block transfer. The next trigger after the completion of a repeated block transfer triggers another block transfer.

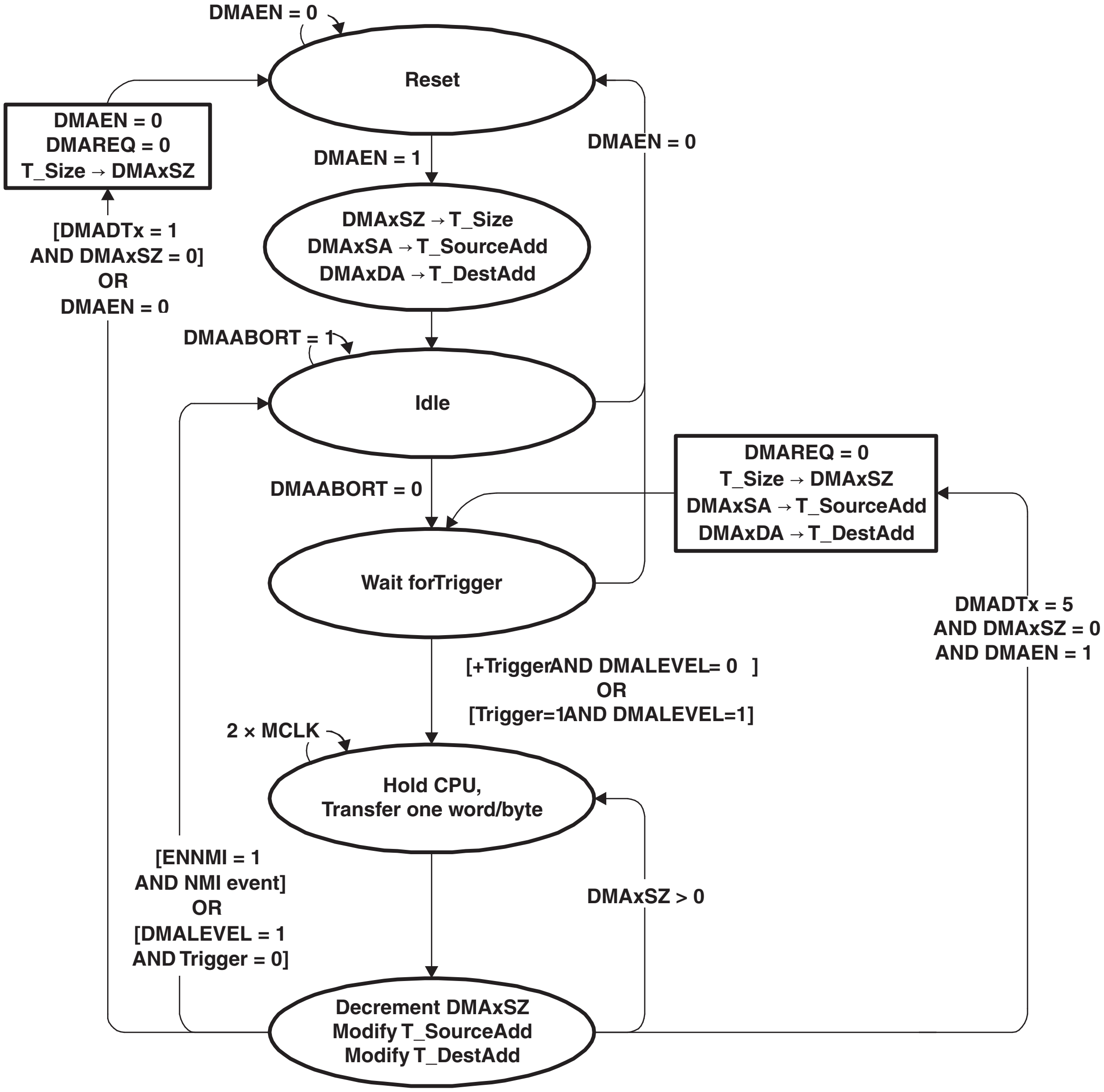


Figure 9-4. DMA Block Transfer State Diagram

9.2.2.1 Burst-Block Transfers

In burst-block mode, transfers are block transfers with CPU activity interleaved. The CPU executes 2 MCLK cycles after every four byte/word transfers of the block resulting in 20% CPU execution capacity. After the burst-block, CPU execution resumes at 100% capacity and the DMAEN bit is cleared. DMAEN must be set again before another burst-block transfer can be triggered. After a burst-block transfer has been triggered, further trigger signals occurring during the burst-block transfer are ignored. The burst-block transfer state diagram is shown in Figure 9-5.

The DMAxSZ register is used to define the size of the block and the DMADSTINCRx and DMASRCINCRx bits select if the destination address and the source address are incremented or decremented after each transfer of the block. If DMAxSZ = 0, no transfers occur.

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

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

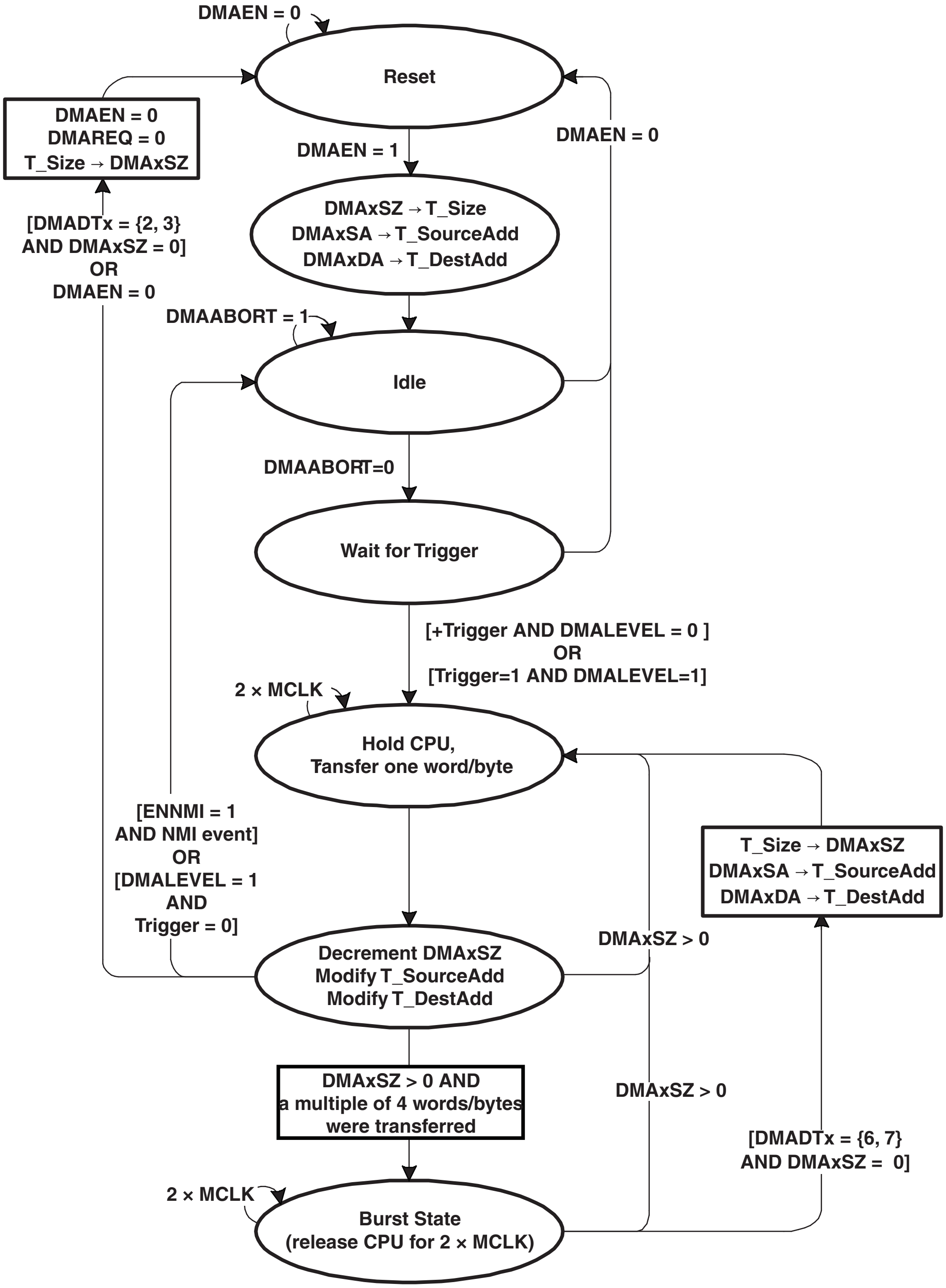


Figure 9-5. DMA Burst-Block Transfer State Diagram

9.2.3 Initiating DMA Transfers

Each DMA channel is independently configured for its trigger source with the DMAxTSELx.The DMAxTSELx bits should be modified only when the DMACTLx DMAEN bit is 0. Otherwise, unpredictable DMA triggers may occur. Table 9-2 describes the trigger operation for each type of module. Please refer to the specific device datasheet for the list of triggers available, along with their respective DMAxTSELx values.

When selecting the trigger, the trigger must not have already occurred, or the transfer will not take place.

Note: DMA Trigger Selection and USB

On devices that contain a USB module, the triggers selection from DMA channels 0, 1, or 2 can be used for the USB time stamp event selection. Please refer to the USB module description for further details.

Edge-Sensitive Triggers

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

Level-Sensitive Triggers

When DMALEVEL = 1, level-sensitive triggers are used. For proper operation, level-sensitive triggers can only be used when external trigger DMAE0 is selected as the trigger. DMA transfers are triggered as long as the trigger signal is high and the DMAEN bit remains set.

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

When DMALEVEL = 1, transfer modes selected when DMADTx = {0, 1, 2, 3} are recommended because the DMAEN bit is automatically reset after the configured transfer.

Halting Executing Instructions for DMA Transfers

The DMARMWDIS bit controls when the CPU is halted for DMA transfers. When DMARMWDIS = 0, the CPU is halted immediately and the transfer begins when a trigger is received. In this case, it is possible that CPU read-modify-write operations can be interrupted by a DMA transfer. When DMARMWDIS = 1, the CPU finishes the currently executing read-modify-write operation before the DMA controller halts the CPU and the transfer begins.See Table 9-2

Table 9-2. DMA Trigger Operation

Module	Operation
DMA	A transfer is triggered when the DMAREQ bit is set. The DMAREQ bit is automatically reset when the transfer starts. A transfer is triggered when the DMAxIFG flag is set. DMA0IFG triggers channel 1, DMA1IFG triggers channel 2, and DMA2IFG triggers channel 0. None of the DMAxIFG flags are automatically reset when the transfer starts. A transfer is triggered by the external trigger DMAE0.
Timer_A	A transfer is triggered when the TACCR0 CCIFG flag is set. The TACCR0 CCIFG flag is automatically reset when the transfer starts. If the TACCR0 CCIE bit is set, the TACCR0 CCIFG flag will not trigger a transfer. A transfer is triggered when the TACCR2 CCIFG flag is set. The TACCR2 CCIFG flag is automatically reset when the transfer starts. If the TACCR2 CCIE bit is set, the TACCR2 CCIFG flag will not trigger a transfer.
Timer_B	A transfer is triggered when the TBCCR0 CCIFG flag is set. The TBCCR0 CCIFG flag is automatically reset when the transfer starts. If the TBCCR0 CCIE bit is set, the TBCCR0 CCIFG flag will not trigger a transfer. A transfer is triggered when the TBCCR2 CCIFG flag is set. The TBCCR2 CCIFG flag is automatically reset when the transfer starts. If the TBCCR2 CCIE bit is set, the TBCCR2 CCIFG flag will not trigger a transfer.
USCI_Ax	A transfer is triggered when USCI_Ax receives new data. UCAxRXIFG is automatically reset when the transfer starts. If UCAxRXIE is set, the UCAxRXIFG will not trigger a transfer. A transfer is triggered when USCI_Ax is ready to transmit new data. UCAxTXIFG is automatically reset when the transfer starts. If UCAxTXIE is set, the UCAxTXIFG will not trigger a transfer.
USCI_Bx	A transfer is triggered when USCI_Bx receives new data. UCBxRXIFG is automatically reset when the transfer starts. If UCBxRXIE is set, the UCBxRXIFG will not trigger a transfer. A transfer is triggered when USCI_Bx is ready to transmit new data. UCBxTXIFG is automatically reset when the transfer starts. If UCBxTXIE is set, the UCBxTXIFG will not trigger a transfer.
DAC12_A	A transfer is triggered when the DAC12_xCTL0 DAC12IFG flag is set. The DAC12_xCTL0 DAC12IFG flag is automatically cleared when the transfer starts. If the DAC12_xCTL0 DAC12IE bit is set, the DAC12_xCTL0 DAC12IFG flag will not trigger a transfer.
ADC12_A	A transfer is triggered by an ADC12IFGx flag. When single-channel conversions are performed, the corresponding ADC12IFGx is the trigger. When sequences are used, the ADC12IFGx for the last conversion in the sequence is the trigger. A transfer is triggered when the conversion is completed and the ADC12IFGx is set. Setting the ADC12IFGx with software will not trigger a transfer. All ADC12IFGx flags are automatically reset when the associated ADC12MEMx register is accessed by the DMA controller.
MPY	A transfer is triggered when the hardware multiplier is ready for a new operand.
Reserved	No transfer is triggered.

9.2.4 *Stopping DMA Transfers*

There are two ways to stop DMA transfers in progress:

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

9.2.5 *DMA Channel Priorities*

The default DMA channel priorities are DMA0 through DMA7. If two or three triggers happen simultaneously or are pending, the channel with the highest priority completes its transfer (single, block or burst-block transfer) first, then the second priority channel, then the third priority channel. Transfers in progress are not halted if a higher priority channel is triggered. The higher priority channel waits until the transfer in progress completes before starting.

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

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

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

9.2.6 DMA Transfer Cycle Time

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

If the MCLK source is active, but the CPU is off, the DMA controller will use the MCLK source for each transfer, without re-enabling the CPU. If the MCLK source is off, the DMA controller will temporarily restart MCLK, sourced with DCOCLK, for the single transfer or complete block or burst-block transfer. The CPU remains off, and after the transfer completes, MCLK is turned off. The maximum DMA cycle time for all operating modes is shown in Table 9-3.

Table 9-3. Maximum Single-Transfer DMA Cycle Time

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

[1] The additional 5 s are needed to start the DCOCLK. It is the $t_{(LPMx)}$ parameter in the data sheet.

9.2.7 Using DMA With System Interrupts

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

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

9.2.8 DMA Controller Interrupts

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

All DMAIFG flags are prioritized, with DMA0IFG being the highest, and combined to source a single interrupt vector. The highest priority enabled interrupt generates a number in the DMAIV register. This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled DMA interrupts do not affect the DMAIV value.

Any access, read or write, of the DMAIV 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, assume that DMA0 has the highest priority. If the DMA0IFG and DMA2IFG flags are set when the interrupt service routine accesses the DMAIV register, DMA0IFG is reset automatically. After the RETI instruction of the interrupt service routine is executed, the DMA2IFG will generate another interrupt.

DMAIV Software Example

The following software example shows the recommended use of DMAIV and the handling overhead for a three channel DMA controller. The DMAIV 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.

```
;Interrupt handler for DMAxIFG                    Cycles

DMA_HND      ...              ; Interrupt latency          6
         ADD      &DMAIV,PC   ; Add offset to Jump table    3
         RETI                 ; Vector  0: No interrupt     5
         JMP      DMA0_HND    ; Vector  2: DMA channel 0    2
         JMP      DMA1_HND    ; Vector  4: DMA channel 1    2
         JMP      DMA2_HND    ; Vector  6: DMA channel 2    2
         JMP      DMA3_HND    ; Vector  8: DMA channel 3    2
         JMP      DMA4_HND    ; Vector 10: DMA channel 4    2
         JMP      DMA5_HND    ; Vector 12: DMA channel 5    2
         JMP      DMA6_HND    ; Vector 14: DMA channel 6    2
         JMP      DMA7_HND    ; Vector 16: DMA channel 7    2

DMA7_HND                      ; Vector 16: DMA channel 7
         ...                  ; Task starts here
         RETI                 ; Back to main program        5

DMA6_HND                      ; Vector 14: DMA channel 6
         ...                  ; Task starts here
         RETI                 ; Back to main program        5

DMA5_HND                      ; Vector 12: DMA channel 5
         ...                  ; Task starts here
         RETI                 ; Back to main program        5

DMA4_HND                      ; Vector 10: DMA channel 4
         ...                  ; Task starts here
         RETI                 ; Back to main program        5

DMA3_HND                      ; Vector 8: DMA channel 3
         ...                  ; Task starts here
         RETI                 ; Back to main program        5

DMA2_HND                      ; Vector 6: DMA channel 2
         ...                  ; Task starts here
         RETI                 ; Back to main program        5

DMA1_HND                      ; Vector 4: DMA channel 1
         ...                  ; Task starts here
         RETI                 ; Back to main program        5

DMA0_HND                      ; Vector 2: DMA channel 0
         ...                  ; Task starts here
         RETI                 ; Back to main program        5
```

9.2.9 Using the USCI_B I^{2}C Module with the DMA Controller

The USCI_B I^{2}C module provides two trigger sources for the DMA controller. The USCI_B I^{2}C module can trigger a transfer when new I^{2}C data is received and the when the transmit data is needed.

9.2.10 Using ADC12 with the DMA Controller

MSP430 devices with an integrated DMA controller can automatically move data from any ADC12MEMx register to another location. DMA transfers are done without CPU intervention and independently of any low-power modes. The DMA controller increases throughput of the ADC12 module, and enhances low-power applications allowing the CPU to remain off while data transfers occur.

DMA transfers can be triggered from any ADC12IFGx flag. When CONSEQx = {0,2} the ADC12IFGx flag for the ADC12MEMx used for the conversion can trigger a DMA transfer. When CONSEQx = {1,3}, the ADC12IFGx flag for the last ADC12MEMx in the sequence can trigger a DMA transfer. Any ADC12IFGx flag is automatically cleared when the DMA controller accesses the corresponding ADC12MEMx.

9.2.11 Using DAC12 With the DMA Controller

MSP430 devices with an integrated DMA controller can automatically move data to the DAC12_xDAT register. DMA transfers are done without CPU intervention and independently of any low-power modes. The DMA controller increases throughput to the DAC12 module, and enhances low-power applications allowing the CPU to remain off while data transfers occur.

Applications requiring periodic waveform generation can benefit from using the DMA controller with the DAC12. For example, an application that produces a sinusoidal waveform may store the sinusoid values in a table. The DMA controller can continuously and automatically transfer the values to the DAC12 at specific intervals creating the sinusoid with zero CPU execution. The DAC12_xCTL DAC12IFG flag is automatically cleared when the DMA controller accesses the DAC12_xDAT register.

9.3 DMA Registers

The DMA module registers are listed in Table 9-4. The base addresses can be found in the device specific datasheet. Each channel starts at its respective base address. The address offsets are listed in Table 9-4.

Table 9-4. DMA Registers

Register	Short Form	Register Type	Address Offset	Initial State
DMA control 0	DMACTL0	Read/write	00h	0000h
DMA control 1	DMACTL1	Read/write	02h	0000h
DMA control 2	DMACTL2	Read/write	04h	0000h
DMA control 3	DMACTL3	Read/write	06h	0000h
DMA control 4	DMACTL4	Read/write	08h	0000h
DMA interrupt vector	DMAIV	Read only	0Eh	0000h
DMA channel 0 control	DMA0CTL	Read/write	00h	0000h
DMA channel 0 source address	DMA0SA	Read/write	02h	Unchanged
DMA channel 0 destination address	DMA0DA	Read/write	06h	Unchanged
DMA channel 0 transfer size	DMA0SZ	Read/write	0Ah	Unchanged
DMA channel 1 control	DMA1CTL	Read/write	00h	0000h
DMA channel 1 source address	DMA1SA	Read/write	02h	Unchanged
DMA channel 1 destination address	DMA1DA	Read/write	06h	Unchanged
DMA channel 1 transfer size	DMA1SZ	Read/write	0Ah	Unchanged
DMA channel 2 control	DMA2CTL	Read/write	00h	0000h
DMA channel 2 source address	DMA2SA	Read/write	02h	Unchanged
DMA channel 2 destination address	DMA2DA	Read/write	06h	Unchanged
DMA-channel 2 transfer size	DMA2SZ	Read/write	0Ah	Unchanged
DMA channel 3 control	DMA3CTL	Read/write	00h	0000h
DMA channel 3 source address	DMA3SA	Read/write	02h	Unchanged
DMA channel 3 destination address	DMA3DA	Read/write	06h	Unchanged
DMA-channel 3 transfer size	DMA3SZ	Read/write	0Ah	Unchanged
DMA channel 4 control	DMA4CTL	Read/write	00h	0000h
DMA channel 4 source address	DMA4SA	Read/write	02h	Unchanged
DMA channel 4 destination address	DMA4DA	Read/write	06h	Unchanged
DMA-channel 4 transfer size	DMA4SZ	Read/write	0Ah	Unchanged
DMA channel 5 control	DMA5CTL	Read/write	00h	0000h
DMA channel 5 source address	DMA5SA	Read/write	02h	Unchanged
DMA channel 5 destination address	DMA5DA	Read/write	06h	Unchanged
DMA-channel 5 transfer size	DMA5SZ	Read/write	0Ah	Unchanged
DMA channel 6 control	DMA6CTL	Read/write	00h	0000h
DMA channel 6 source address	DMA6SA	Read/write	02h	Unchanged
DMA channel 6 destination address	DMA6DA	Read/write	06h	Unchanged
DMA-channel 6 transfer size	DMA6SZ	Read/write	0Ah	Unchanged
DMA channel 7 control	DMA7CTL	Read/write	00h	0000h
DMA channel 7 source address	DMA7SA	Read/write	02h	Unchanged
DMA channel 7 destination address	DMA7DA	Read/write	06h	Unchanged
DMA-channel 7 transfer size	DMA7SZ	Read/write	0Ah	Unchanged

DMACTL0, DMA Control Register 0

15	14	13	12	11	10	9	8
Reserved			**DMA1TSELx**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
Reserved			**DMA0TSELx**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Reserved Bits 15-13 Reserved. Read only. Always read as 0.

DMA1TSELx Bits 12-8 DMA trigger select. These bits select the DMA transfer trigger. Refer to the device specific datasheet for number of channels and trigger assignment.

00000	DMA1TRIG0
00001	DMA1TRIG1
00010	DMA1TRIG2
⋮	
11110	DMA1TRIG30
11111	DMA1TRIG31

Reserved Bits 7-5 Reserved. Read only. Always read as 0.

DMA0TSELx Bits 4-0 Same as DMA1TSELx

DMACTL1, DMA Control Register 1

15	14	13	12	11	10	9	8
Reserved			**DMA3TSELx**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
Reserved			**DMA2TSELx**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Reserved Bits 15-13 Reserved. Read only. Always read as 0.

DMA3TSELx Bits 12-8 DMA trigger select. These bits select the DMA transfer trigger. Refer to the device specific datasheet for number of channels and trigger assignment.

00000	DMA3TRIG0
00001	DMA3TRIG1
00010	DMA3TRIG2
⋮	
11110	DMA3TRIG30
11111	DMA3TRIG31

Reserved Bits 7-5 Reserved. Read only. Always read as 0.

DMA2TSELx Bits 4-0 Same as DMA3TSELx

DMACTL2, DMA Control Register 2

15	14	13	12	11	10	9	8
Reserved			**DMA5TSELx**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
Reserved			**DMA4TSELx**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Reserved Bits 15-13 Reserved. Read only. Always read as 0.

DMA5TSELx Bits 12-8 DMA trigger select. These bits select the DMA transfer trigger. Refer to the device specific datasheet for number of channels and trigger assignment.

00000	DMA5TRIG0
00001	DMA5TRIG1
00010	DMA5TRIG2
⋮	
11110	DMA5TRIG30
11111	DMA5TRIG31

Reserved Bits 7-5 Reserved. Read only. Always read as 0.

DMA4TSELx Bits 4-0 Same as DMA5TSELx

DMACTL3, DMA Control Register 3

15	14	13	12	11	10	9	8
Reserved			**DMA7TSELx**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
Reserved			**DMA6TSELx**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Reserved Bits 15-13 Reserved. Read only. Always read as 0.

DMA7TSELx Bits 12-8 DMA trigger select. These bits select the DMA transfer trigger. Refer to the device specific datasheet for number of channels and trigger assignment.

00000	DMA7TRIG0
00001	DMA7TRIG1
00010	DMA7TRIG2
⋮	
11110	DMA7TRIG30
11111	DMA7TRIG31

Reserved Bits 7-5 Reserved. Read only. Always read as 0.

DMA6TSELx Bits 4-0 Same as DMA7TSELx

DMACTL4, DMA Control Register 4

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

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**DMARMWDIS**	**ROUND ROBIN**	**ENNMI**
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

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

DMARMWDIS Bit 2 Read-Modify-Write Disable. This bit when set, inhibits any DMA transfers from occurring during CPU read-modify-write operations.

0 DMA transfers can occur during read-modify-write CPU operations

1 DMA transfers inhibited during read-modify-write CPU operations

ROUNDROBIN Bit 1 Round robin. This bit enables the round-robin DMA channel priorities.

0 DMA channel priority is DMA0 - DMA1 - DMA2 - ...... - DMA7

1 DMA channel priority changes with each transfer

ENNMI Bit 0 Enable NMI. This bit enables the interruption of a DMA transfer by an NMI interrupt. When an NMI interrupts a DMA transfer, the current transfer is completed normally, further transfers are stopped, and DMAABORT is set.

0 NMI interrupt does not interrupt DMA transfer

1 NMI interrupt interrupts a DMA transfer

DMAxCTL, DMA Channel x Control Register

15	14	13	12	11	10	9	8
Reserved	**DMADTx**			**DMADSTINCRx**		**DMASRCINCRx**	
r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

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

Reserved Bit 15 Reserved. Read only. Always read as 0.

DMADTx Bits 14-12 DMA transfer mode

000	Single transfer
001	Block transfer
010	Burst-block transfer
011	Burst-block transfer
100	Repeated single transfer
101	Repeated block transfer
110	Repeated burst-block transfer
111	Repeated burst-block transfer

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

00	Destination address is unchanged
01	Destination address is unchanged
10	Destination address is decremented
11	Destination address is incremented

DMASRCINCRx Bits 9-8 DMA source increment. This bit selects automatic incrementing or decrementing of the source address for each byte or word transfer. When DMASRCBYTE=1, the source address increments/decrements by one. When DMASRCBYTE=0, the source address increments/decrements by two. The DMAxSA is copied into a temporary register and the temporary register is incremented or decremented. DMAxSA is not incremented or decremented.

00	Source address is unchanged
01	Source address is unchanged
10	Source address is decremented
11	Source address is incremented

DMADSTBYTE Bit 7 DMA destination byte. This bit selects the destination as a byte or word.

0	Word
1	Byte

DMASRCBYTE Bit 6 DMA source byte. This bit selects the source as a byte or word.

0	Word
1	Byte

DMALEVEL Bit 5 DMA level. This bit selects between edge-sensitive and level-sensitive triggers.

0	Edge sensitive (rising edge)
1	Level sensitive (high level)

DMAEN Bit 4 DMA enable

0	Disabled
1	Enabled

DMAIFG	Bit 3	DMA interrupt flag	
		0	No interrupt pending
		1	Interrupt pending
DMAIE	Bit 2	DMA interrupt enable	
		0	Disabled
		1	Enabled
DMAABORT	Bit 1	DMA abort. This bit indicates if a DMA transfer was interrupt by an NMI.	
		0	DMA transfer not interrupted
		1	DMA transfer was interrupted by NMI
DMAREQ	Bit 0	DMA request. Software-controlled DMA start. DMAREQ is reset automatically.	
		0	No DMA start
		1	Start DMA

DMAxSA, DMA Source Address Register

31	30	29	28	27	26	25	24
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0

23	22	21	20	19	18	17	16
Reserved				**DMAxSAx**			
r0	r0	r0	r0	rw	rw	rw	rw

15	14	13	12	11	10	9	8
DMAxSAx							
rw	rw	rw	rw	rw	rw	rw	rw

7	6	5	4	3	2	1	0
DMAxSAx							
rw	rw	rw	rw	rw	rw	rw	rw

Reserved Bits 31-20 Reserved. Read only. Always read as 0.

DMAxSA Bits 15-0 DMA source address. The source address register points to the DMA source address for single transfers or the first source address for block transfers. The source address register remains unchanged during block and burst-block transfers. There are two words for the DMAxSA register. Bits 31-20 are reserved and always read as zero. Reading or writing bits 19-16 requires the use of extended instructions. When writing to DMAxSA with word instructions, bits 19-16 are cleared.

DMAxDA, DMA Destination Address Register

31	30	29	28	27	26	25	24
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0

23	22	21	20	19	18	17	16
Reserved				**DMAxDAx**			
r0	r0	r0	r0	rw	rw	rw	rw

15	14	13	12	11	10	9	8
DMAxDAx							
rw	rw	rw	rw	rw	rw	rw	rw

7	6	5	4	3	2	1	0
DMAxDAx							
rw	rw	rw	rw	rw	rw	rw	rw

Reserved Bits 31-20 Reserved. Read only. Always read as 0.

DMAxDAx Bits 15-0 DMA destination address. The destination address register points to the DMA destination address for single transfers or the first destination address for block transfers. The destination address register remains unchanged during block and burst-block transfers. There are two words for the DMAxDA register. Bits 31-20 are reserved and always read as zero. Reading or writing bits 19-16 requires the use of extended instructions. When writing to DMAxDA with word instructions, bits 19-16 are cleared.

DMAxSZ, DMA Size Address Register

15	14	13	12	11	10	9	8
DMAxSZx							
rw	rw	rw	rw	rw	rw	rw	rw

7	6	5	4	3	2	1	0
DMAxSZx							
rw	rw	rw	rw	rw	rw	rw	rw

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

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

DMAIV, DMA 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**	**DMAIVx**					**0**
r0	r0	r-(0)	r-(0)	r-(0)	r-(0)	r-(0)	r0

DMAIVx Bits 15-0 DMA interrupt vector value

DMAIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	DMA channel 0	DMA0IFG	Highest
04h	DMA channel 1	DMA1IFG	
06h	DMA channel 2	DMA2IFG	
08h	DMA channel 3	DMA3IFG	
0Ah	DMA channel 4	DMA4IFG	
0Ch	DMA channel 5	DMA5IFG	
0Eh	DMA channel 6	DMA6IFG	
10h	DMA channel 7	DMA7IFG	Lowest

Chapter 10

SLAU208–June 2008

32-Bit Hardware Multiplier (MPY32)

This chapter describes the 32-bit hardware multiplier (MPY32). The 32-bit hardware multiplier is implemented in all MSP430x5xx devices.

Topic **Page**

10.1 32-Bit Hardware Multiplier Introduction

The 32-bit hardware multiplier is a peripheral and is not part of the MSP430 CPU. This means its activities do not interfere with the CPU activities. The multiplier registers are peripheral registers that are loaded and read with CPU instructions.

The hardware multiplier supports:

- Unsigned multiply
- Signed multiply
- Unsigned multiply accumulate
- Signed multiply accumulate
- 8-bit, 16-bit, 24-bit, and 32-bit operands
- Saturation
- Fractional numbers
- 8-bit and 16-bit operation compatible with 16-bit hardware multiplier
- 8-bit and 24-bit multiplications without requiring a "sign extend" instruction

The 32-bit hardware multiplier block diagram is shown in Figure 10-1.

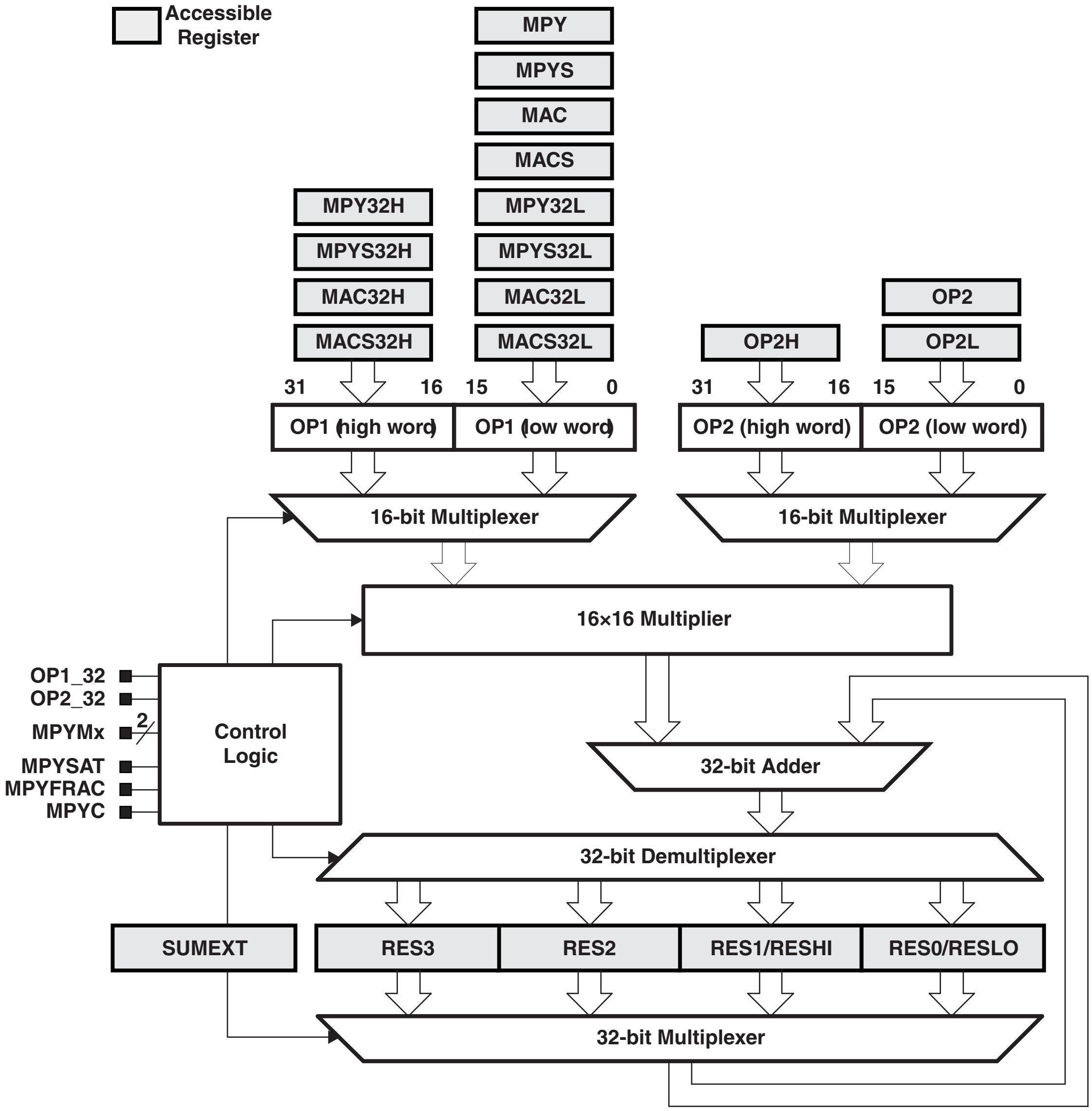


Figure 10-1. 32-Bit Hardware Multiplier Block Diagram

10.2 32-Bit Hardware Multiplier Operation

The hardware multiplier supports 8-bit, 16-bit, 24-bit, and 32-bit operands with unsigned multiply, signed multiply, unsigned multiply-accumulate, and signed multiply-accumulate operations. The size of the operands are defined by the address the operand is written to and if it is written as word or byte. The type of operation is selected by the address the first operand is written to.

The hardware multiplier has two 32-bit operand registers, operand one OP1 and operand two OP2, and a 64-bit result register accessible via registers RES0 to RES3. For compatibility with the 16×16 hardware multiplier the result of a 8-bit or 16-bit operation is accessible via RESLO, RESHI, and SUMEXT, as well. RESLO stores the low word of the 16×16-bit result, RESHI stores the high word of the result, and SUMEXT stores information about the result.

The result of a 8-bit or 16-bit operation is ready in three MCLK cycles and can be read with the next instruction after writing to OP2, except when using an indirect addressing mode to access the result. When using indirect addressing for the result, a `NOP` is required before the result is ready.

The result of a 24-bit or 32-bit operation can be read with successive instructions after writing OP2 or OP2H starting with RES0, except when using an indirect addressing mode to access the result. When using indirect addressing for the result, a `NOP` is required before the result is ready.

Table 10-1 summarizes when each word of the 64-bit result is available for the various combinations of operand sizes. With a 32-bit wide second operand, OP2L and OP2H need to be written. Depending on when the two 16-bit parts are written, the result availability may vary; thus, the table shows two entries, one for OP2L written and one for OP2H written. The worst case defines the actual result availability.

Table 10-1. Result Availability (MPYFRAC = 0, MPYSAT = 0)

Operation (OP1 × OP2)	Result ready in MCLK cycles					After
	RES0	RES1	RES2	RES3	MPYC Bit	
8/16 × 8/16	3	3	4	4	3	OP2 written
24/32 × 8/16	3	5	6	7	7	OP2 written
8/16 × 24/32	3	5	6	7	7	OP2L written
	N/A	3	4	4	4	OP2H written
24/32 × 24/32	3	8	10	11	11	OP2L written
	N/A	3	5	6	6	OP2H written

10.2.1 Operand Registers

Operand one OP1 has twelve registers, shown in Table 10-2, used to load data into the multiplier and also select the multiply mode. Writing the low-word of the first operand to a given address selects the type of multiply operation to be performed but does not start any operation. When writing a second word to a high-word register with suffix 32H the multiplier assumes a 32-bit wide OP1, otherwise 16-bits are assumed. The last address written prior to writing OP2 defines the width of the first operand. For example, if MPY32L is written first followed by MPY32H, all 32 bits are used and the data width of OP1 is set to 32 bits. If MPY32H is written first followed by MPY32L, the multiplication will ignore MPY32H and assume a 16-bit wide OP1 using the data written into MPY32L.

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

Table 10-2. OP1 Registers

OP1 Register Name	Operation
MPY	Unsigned multiply – operand bits 0 up to 15
MPYS	Signed multiply – operand bits 0 up to 15
MAC	Unsigned multiply accumulate –operand bits 0 up to 15
MACS	Signed multiply accumulate – operand bits 0 up to 15
MPY32L	Unsigned multiply – operand bits 0 up to 15
MPY32H	Unsigned multiply – operand bits 16 up to 31
MPYS32L	Signed multiply – operand bits 0 up to 15
MPYS32H	Signed multiply – operand bits 16 up to 31
MAC32L	Unsigned multiply accumulate – operand bits 0 up to 15
MAC32H	Unsigned multiply accumulate – operand bits 16 up to 31
MACS32L	Signed multiply accumulate – operand bits 0 up to 15
MACS32H	Signed multiply accumulate – operand bits 16 up to 31

Writing the second operand to the operand two register OP2 initiates the multiply operation. Writing OP2 starts the selected operation with a 16-bit wide second operand together with the values stored in OP1. Writing OP2L starts the selected operation with a 32-bit wide second operand and the multiplier expects a the high-word to be written to OP2H. Writing to OP2H without a preceding write to OP2L is ignored.

Table 10-3. OP2 Registers

OP2 Register Name	Operation
OP2	Start multiplication with 16-bit wide operand two (OP2) (operand bits 0 up to 15)
OP2L	Start multiplication with 32-bit wide operand two (OP2) (operand bits 0 up to 15)
OP2H	Continue multiplication with 32-bit wide operand two (OP2) (operand bits 16 up to 31)

For 8-bit or 24-bit operands the operand registers can be accessed with byte instructions. Accessing the multiplier with a byte instruction during a signed operation will automatically cause a sign extension of the byte within the multiplier module. For 24-bit operands only the high-word should be written as byte. If the 24-bit operands are sign-extended is defined by the register that is used to write the low-word to because this register defines if the operation is unsigned or signed.

The high-word of a 32-bit operand remains unchanged when changing the size of the operand to 16 bit either by modifying the operand size bits or by writing to the respective operand register. During the execution of the 16-bit operation the content of the high-word is ignored.

Note: Changing of First or Second Operand During Multiplication

By default changing OP1 or OP2 while the selected multiply operation is being calculated will render any results invalid that are not ready at the time the new operand(s) are changed. Writing OP2 or OP2L will abort any ongoing calculation and start a new operation. Results that are not ready at that time are invalid also for following MAC or MACS operations.

To avoid this behavior the MPYDLYWRTEN bit can be set to 1. Then all writes to any MPY32 registers are delayed with MPYDLY32=0 until the 64-bit result is ready or with MPYDLY32=1 until the 32-bit result is ready. For MAC and MACS operations always the complete 64-bit result should be ready.

See Table 10-1 for how many CPU cycles are needed until a certain result register is ready and valid for each of the different modes.

10.2.2 Result Registers

The multiplication result is always 64-bits wide. It is accessible via registers RES0 to RES3. Used with a signed operation MPYS or MACS the results are appropriately sign extended. If the result registers are loaded with initial values before a MACS operation the user software must take care that the written value is properly sign extended to 64 bits.

Note: Changing of Result Registers During Multiplication

The result registers must not be modified by the user software after writing the second operand into OP2 or OP2L until the initiated operation is completed.

In addition to RES0 to RES3, for compatibility with the 16×16 hardware multiplier the 32-bit result of a 8-bit or 16-bit operation is accessible via RESLO, RESHI, and SUMEXT. In this case the result low register RESLO holds the lower 16-bits of the calculation result and the result high register RESHI holds the upper 16-bits. RES0 and RES1 are identical to RESLO and RESHI, respectively, in usage and access of calculated results.

The sum extension registers SUMEXT contents depend on the multiply operation and are listed in Table 10-4. If all operands are 16 bits wide or less the 32-bit result is used to determine sign and carry. If one of the operands is larger than 16 bits the 64-bit result is used.

The MPYC bit reflects the multiplier's carry as listed in Table 10-4 and, thus, can be used as 33rd or 65th bit of the result, if fractional or saturation mode is not selected. With MAC or MACS operations, the MPYC bit reflects the carry of the 32-bit or 64-bit accumulation and is not taken into account for successive MAC and MACS operations as the 33rd or 65th bit.

Table 10-4. SUMEXT Contents and MPYC Contents

Mode	SUMEXT		MPYC	
MPY	SUMEXT is always 0000h.		MPYC is always 0.	
MPYS	SUMEXT contains the extended sign of the result.		MPYC contains the sign of the result.	
	00000h	Result was positive or zero	0	Result was positive or zero
	0FFFFh	Result was negative	1	Result was negative
MAC	SUMEXT contains the carry of the result.		MPYC contains the carry of the result.	
	0000h	No carry for result	0	No carry for result
	0001h	Result has a carry	1	Result has a carry
MACS	SUMEXT contains the extended sign of the result.		MPYC contains the carry of the result.	
	00000h	Result was positive or zero	0	No carry for result
	0FFFFh	Result was negative	1	Result has a carry

MACS Underflow and Overflow

The multiplier does not automatically detect underflow or overflow in MACS mode. For example working with 16-bit input data and 32-bit results, i.e. using just RESLO and RESHI, the available range for positive numbers is 0 to 07FFF FFFFh and for negative numbers is 0FFFF FFFFh to 08000 0000h. An underflow occurs when the sum of two negative numbers yields a result that is in the range for a positive number. An overflow occurs when the sum of two positive numbers yields a result that is in the range for a negative number.

The SUMEXT register contains the sign of the result in both cases described above, 0FFFFh for a 32-bit overflow and 0000h for a 32-bit underflow. The MPYC bit in MPY32CTL0 can be used to detect the overflow condition. If the carry is different than the sign reflected by the SUMEXT register an overflow or underflow occurred. User software must handle these conditions appropriately.

10.2.3 Software Examples

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

There is no sign extension necessary in software. Accessing the multiplier with a byte instruction during a signed operation will automatically cause a sign extension of the byte within the multiplier module.

```
; 32x32 Unsigned Multiply
    MOV     #01234h,&MPY32L   ; Load low  word of 1st operand
    MOV     #01234h,&MPY32H   ; Load high word of 1st operand
    MOV     #05678h,&OP2L     ; Load low  word of 2nd operand
    MOV     #05678h,&OP2H     ; Load high word of 2nd operand
;   ...                       ; Process results

; 16x16 Unsigned Multiply
    MOV     #01234h,&MPY      ; Load 1st operand
    MOV     #05678h,&OP2      ; Load 2nd operand
;   ...                       ; Process results

; 8x8 Unsigned Multiply. Absolute addressing.
    MOV.B   #012h,&MPY_B      ; Load 1st operand
    MOV.B   #034h,&OP2_B      ; Load 2nd operand
;   ...                       ; Process results

; 32x32 Signed Multiply
    MOV     #01234h,&MPYS32L  ; Load low  word of 1st operand
    MOV     #01234h,&MPYS32H  ; Load high word of 1st operand
    MOV     #05678h,&OP2L     ; Load low  word of 2nd operand
    MOV     #05678h,&OP2H     ; Load high word of 2nd operand
;   ...                       ; Process results

; 16x16 Signed Multiply
    MOV     #01234h,&MPYS     ; Load 1st operand
    MOV     #05678h,&OP2      ; Load 2nd operand
;   ...                       ; Process results

; 8x8 Signed Multiply. Absolute addressing.
    MOV.B   #012h,&MPYS_B     ; Load 1st operand
    MOV.B   #034h,&OP2_B      ; Load 2nd operand
;   ...                       ; Process results
```

10.2.4 Fractional Numbers

The 32-bit multiplier provides support for fixed-point signal processing. In fixed-point signal processing, fractional number are represented by using a fixed decimal point. To classify different ranges of decimal numbers, a Q-format is used. Different Q-formats represent different locations of the decimal point. Figure 10-2 shows the format of a signed Q15 number using 16 bits. Every bit after the decimal point has a resolution of 1/2, the most significant bit is used as the sign bit. The most negative number is 08000h and the maximum positive number is 07FFFh. This gives a range from –1.0 to 0.999969482 ≈ 1.0 for the signed Q15 format with 16 bits.

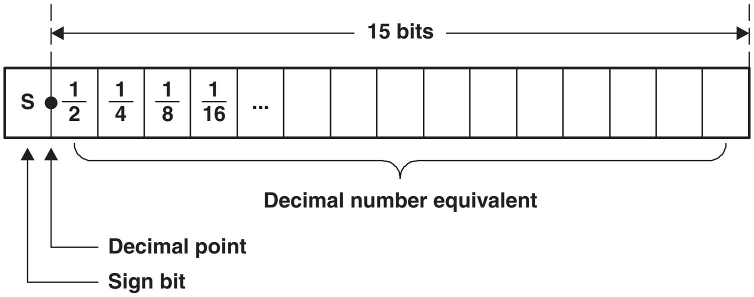


Figure 10-2. Q15 Format Representation

The range can be increased by shifting the decimal point to the right as shown in Figure 10-3. The signed Q14 format with 16 bits gives a range from –2.0 to 1.999938965 ≈ 2.0.

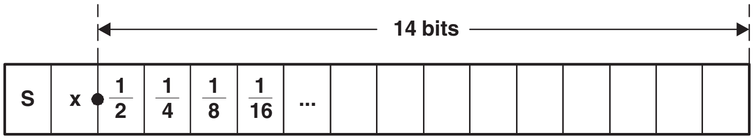


Figure 10-3. Q14 Format Representation

The benefit of using 16-bit signed Q15 or 32-bit signed Q31 numbers with multiplication is that the product of two number in the range from -1.0 to 1.0 is always in that same range.

Fractional Number Mode

Multiplying two fractional numbers using the default multiplication mode with MPYFRAC = 0 and MPYSAT = 0 gives a result with 2 sign bits. For example if two 16-bit Q15 numbers are multiplied a 32-bit result in Q30 format is obtained. To convert the result into Q15 format manually, the first 15 trailing bits and the extended sign bit must be removed. However, when the fractional mode of the multiplier is used, the redundant sign bit is automatically removed yielding a result in Q31 format for the multiplication of two 16-bit Q15 numbers. Reading the result register RES1 gives the result as 16-bit Q15 number. The 32-bit Q31 result of a multiplication of two 32-bit Q31 numbers is accessed by reading registers RES2 and RES3.

The fractional mode is enabled with MPYFRAC = 1 in register MPY32CTL0. The actual content of the result register(s) is not modified when MPYFRAC = 1. When the result is accessed using software, the value is left-shifted 1 bit resulting in the final Q formatted result. This allows user software to switch between reading both the shifted (fractional) and the un-shifted result. The fractional mode should only be enabled when required and disabled after use.

In fractional mode the SUMEXT register contains the sign extended bits 32 and 33 of the shifted result for 16×16-bit operations and bits 64 and 65 for 32×32-bit operations – not only bits 32 or 64, respectively.

The MPYC bit is not affected by the fractional mode. It always reads the carry of the non-fractional result.

```
; Example using
; Fractional 16x16 multiplication
   BIS      #MPYFRAC,&MPY32CTL0   ; Turn on fractional mode
   MOV      &FRACT1,&MPYS         ; Load 1st operand as Q15
   MOV      &FRACT2,&OP2          ; Load 2nd operand as Q15
   MOV      &RES1,&PROD           ; Save result as Q15
   BIC      #MPYFRAC,&MPY32CTL0   ; Back to normal mode
```

Table 10-5. Result Availability in Fractional Mode (MPYFRAC = 1, MPYSAT = 0)

Operation (OP1 × OP2)	Result ready in MCLK cycles					After
	RES0	RES1	RES2	RES3	MPYC Bit	
8/16 × 8/16	3	3	4	4	3	OP2 written
24/32 × 8/16	3	5	6	7	7	OP2 written
8/16 × 24/32	3	5	6	7	7	OP2L written
	N/A	3	4	4	4	OP2H written
24/32 × 24/32	3	8	10	11	11	OP2L written
	N/A	3	5	6	6	OP2H written

Saturation Mode

The multiplier prevents overflow and underflow of signed operations in saturation mode. The saturation mode is enabled with MPYSAT = 1 in register MPY32CTL0. If an overflow occurs the result is set to the most positive value available. If an underflow occurs the result is set to the most negative value available. This is useful to reduce mathematical artifacts in control systems on overflow and underflow conditions. The saturation mode should only be enabled when required and disabled after use.

The actual content of the result register(s) is not modified when MPYSAT = 1. When the result is accessed using software, the value is automatically adjusted providing the most positive or most negative result when an overflow or underflow has occurred. The adjusted result is also used for successive multiply-and-accumulate operations. This allows user software to switch between reading the saturated and the non-saturated result.

With 16×16 operations the saturation mode only applies to the least significant 32 bits, i.e. the result registers RES0 and RES1. Using the saturation mode in MAC or MACS operations that mix 16×16 operations with 32×32, 16×32 or 32×16 operations will lead to unpredictable results.

With 32×32, 16×32, and 32×16 operations the saturated result can only be calculated when RES3 is ready. In non-5xx devices, reading RES0 to RES2 prior to the complete result being ready will deliver the non-saturated results independent of the MPYSAT bit setting.

Enabling the saturation mode does not affect the content of the SUMEXT register nor the content of the MPYC bit.

```
; Example using
; Fractional 16x16 multiply accumulate with Saturation
   ; Turn on fractional and saturation mode:
   BIS      #MPYSAT+MPYFRAC,&MPY32CTL0
   MOV      &A1,&MPYS                      ; Load A1 for 1st term
   MOV      &K1,&OP2                       ; Load K1 to get A1*K1
   MOV      &A2,&MACS                      ; Load A2 for 2nd term
   MOV      &K2,&OP2                       ; Load K2 to get A2*K2
   MOV      &RES1,&PROD                    ; Save A1*K1+A2*K2 as result
   BIC      #MPYSAT+MPYFRAC,&MPY32CTL0     ; turn back to normal
```

Table 10-6. Result Availability in Saturation Mode (MPYSAT = 1)

Operation (OP1 × OP2)	Result ready in MCLK cycles					After
	RES0	RES1	RES2	RES3	MPYC Bit	
8/16 × 8/16	3	3	N/A	N/A	3	OP2 written
24/32 × 8/16	7	7	7	7	7	OP2 written
8/16 × 24/32	7	7	7	7	7	OP2L written
	4	4	4	4	4	OP2H written
24/32 × 24/32	11	11	11	11	11	OP2L written
	6	6	6	6	6	OP2H written

Figure 10-4 shows the flow for 32-bit saturation used for 16×16 bit multiplications and the flow for 64-bit saturation used in all other cases. Primarily, the saturated results depends on the carry bit MPYC and the most significant bit of the result. Secondly, if the fractional mode is enabled it depends also on the two most significant bits of the unshift result; i.e., the result that is read with fractional mode disabled.

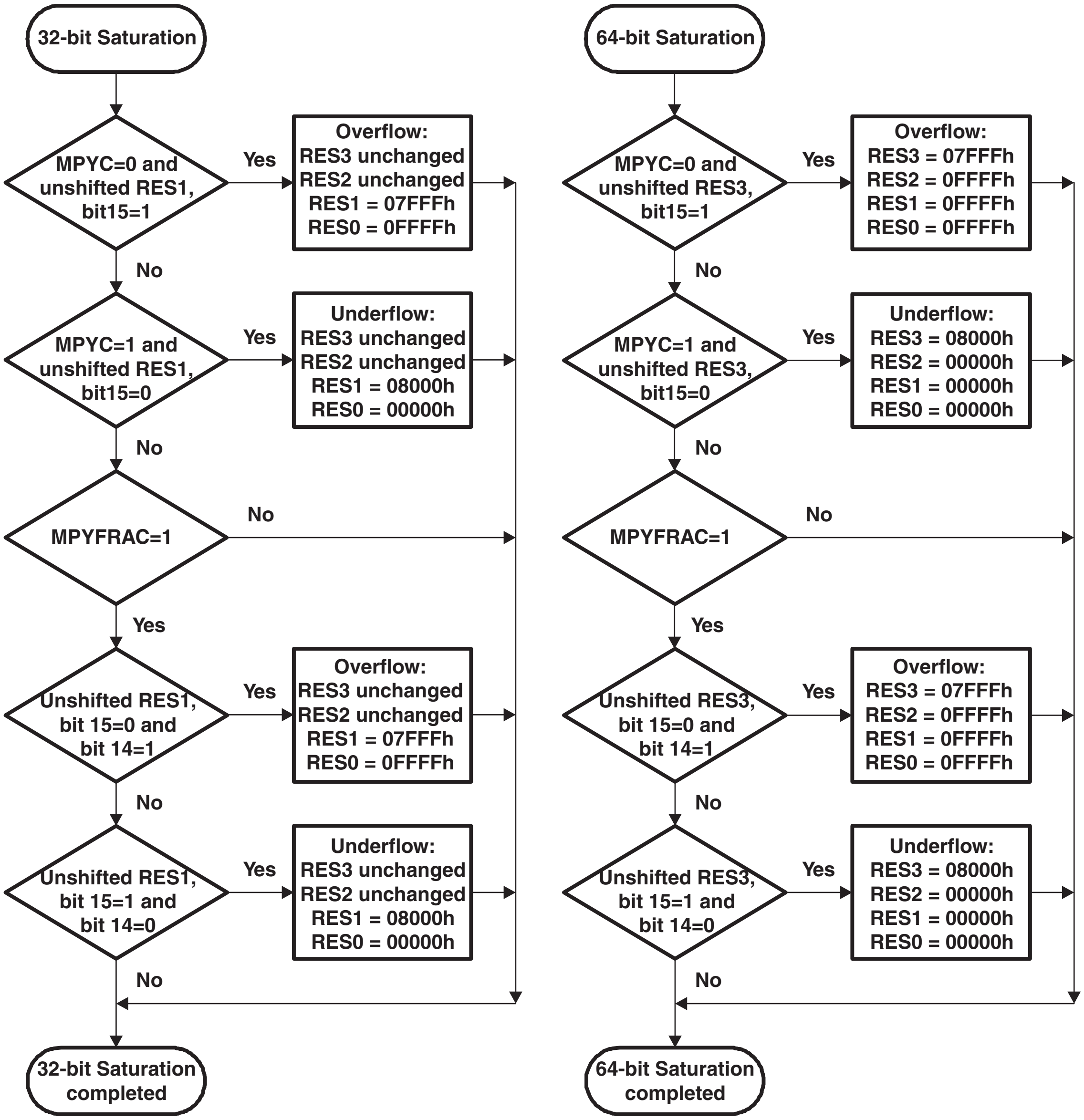

32-bit Saturation
MPYC=0 and unshifted RES1, bit15=1
Yes
Overflow: RES3 unchanged RES2 unchanged RES1 = 07FFFh RES0 = 0FFFFh
No
MPYC=1 and unshifted RES1, bit15=0
Yes
Underflow: RES3 unchanged RES2 unchanged RES1 = 08000h RES0 = 00000h
No
MPYFRAC=1
No
Yes
Unshifted RES1, bit 15=0 and bit 14=1
Yes
Overflow: RES3 unchanged RES2 unchanged RES1 = 07FFFh RES0 = 0FFFFh
No
Unshifted RES1, bit 15=1 and bit 14=0
Yes
Underflow: RES3 unchanged RES2 unchanged RES1 = 08000h RES0 = 00000h
No
32-bit Saturation completed
64-bit Saturation
MPYC=0 and unshifted RES3, bit15=1
Yes
Overflow: RES3 = 07FFFh RES2 = 0FFFFh RES1 = 0FFFFh RES0 = 0FFFFh
No
MPYC=1 and unshifted RES3, bit15=0
Yes
Underflow: RES3 = 08000h RES2 = 00000h RES1 = 00000h RES0 = 00000h
No
MPYFRAC=1
No
Yes
Unshifted RES3, bit 15=0 and bit 14=1
Yes
Overflow: RES3 = 07FFFh RES2 = 0FFFFh RES1 = 0FFFFh RES0 = 0FFFFh
No
Unshifted RES3, bit 15=1 and bit 14=0
Yes
Underflow: RES3 = 08000h RES2 = 00000h RES1 = 00000h RES0 = 00000h
No
64-bit Saturation completed

Figure 10-4. Saturation Flow Chart

Note: Saturation in Fractional Mode

In case of multiplying –1.0 × –1.0 in fractional mode, the result of +1.0 is out of range, thus, the saturated result gives the most positive result.

When using multiply-and-accumulate operations the accumulated values are saturated as if MPYFRAC=0 - only during read accesses to the result registers the values are saturated taking the fractional mode into account. This provides additional dynamic range during the calculation and only the end-result is then saturated if needed.

The following example illustrates a special case showing the saturation function in fractional mode. It also uses the 8-bit functionality of the MPY32 module.

```
; Turn on fractional and saturation mode,
; clear all other bits in MPY32CTL0:
MOV      #MPYSAT+MPYFRAC,&MPY32CTL0
;Pre-load result registers to demonstrate overflow
MOV      #0,&RES3          ;
MOV      #0,&RES2          ;
MOV      #07FFFh,&RES1     ;
MOV      #0FA60h,&RES0     ;
MOV.B    #050h,&MACS_B     ; 8-bit signed MAC operation
MOV.B    #012h,&OP2_B      ; Start 16x16 bit operation
MOV      &RES0,R6          ; R6 = 0FFFFh
MOV      &RES1,R7          ; R7 = 07FFFh
```

The result is saturated because already the result not converted into a fractional number shows an overflow. The multiplication of the two positive numbers 00050h and 00012h gives 005A0h. 005A0h added to 07FFF FA60h results in 8000 059Fh without MPYC being set. Since the MSB of the unmodified result RES1 is 1 and MPYC = 0, the result is saturated according to the saturation flow chart in Figure 10-4.

Note: Validity of Saturated Result

The saturated result is only valid if the registers RES0 to RES3, the size of operands 1 and 2 and MPYC are not modified.

If the saturation mode is used with a preloaded result, user software must ensure that MPYC in the MPY32CTL0 register is loaded with the sign bit of the written result otherwise the saturation mode will erroneously saturate the result.

10.2.5 Putting It All Together

Figure 10-5 shows the complete multiplication flow, depending on the various selectable modes for the MPY32 module.

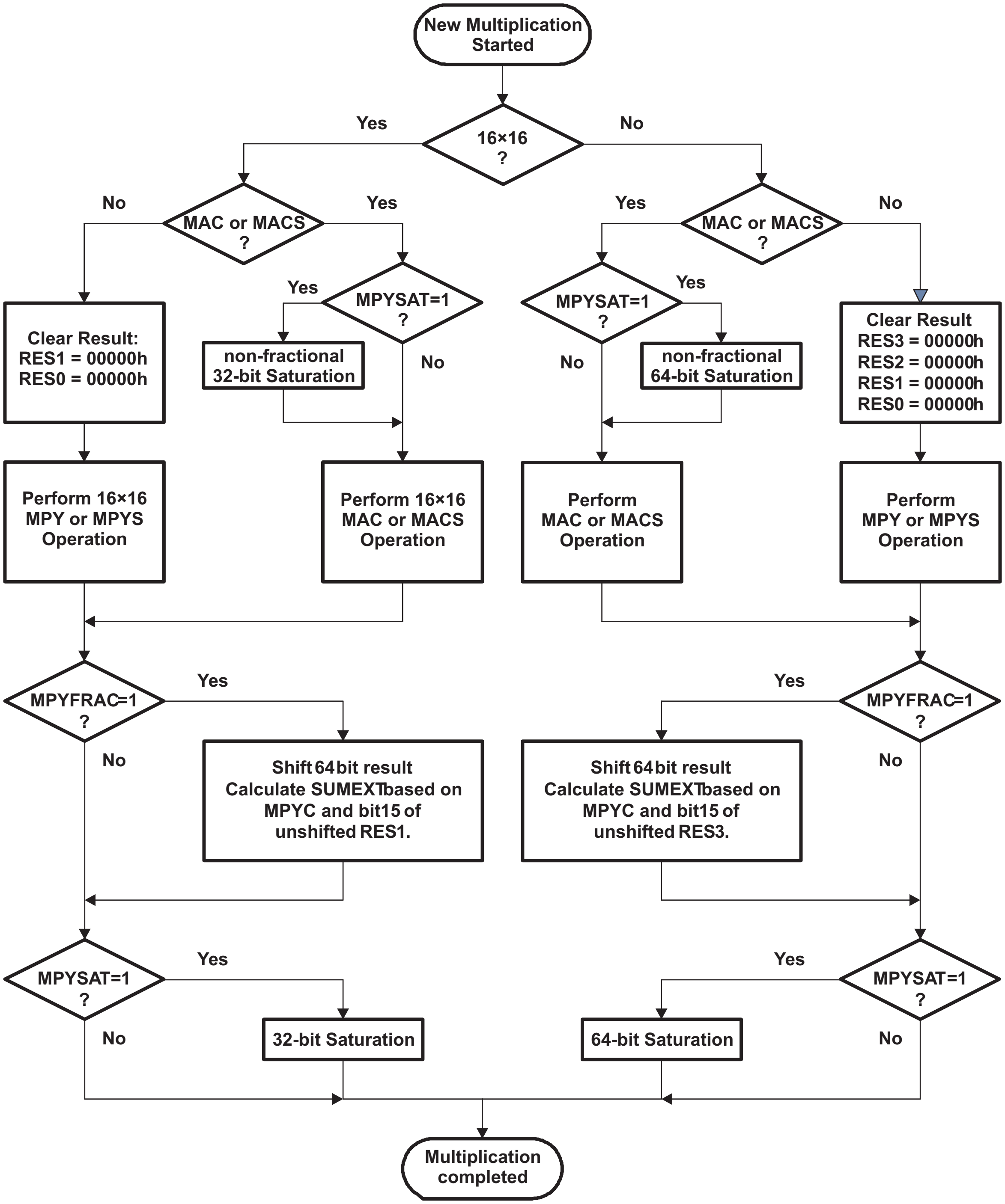

New Multiplication Started
16×16 ?
Yes
No
MAC or MACS ?
No
Yes
MAC or MACS ?
Yes
No
MPYSAT=1 ?
Yes
No
non-fractional 32-bit Saturation
MPYSAT=1 ?
Yes
No
non-fractional 64-bit Saturation
Clear Result: RES1 = 00000h RES0 = 00000h
Clear Result RES3 = 00000h RES2 = 00000h RES1 = 00000h RES0 = 00000h
Perform 16×16 MPY or MPYS Operation
Perform 16×16 MAC or MACS Operation
Perform MAC or MACS Operation
Perform MPY or MPYS Operation
MPYFRAC=1 ?
Yes
No
Shift 64bit result Calculate SUMEXT based on MPYC and bit15 of unshifted RES1.
Shift 64bit result Calculate SUMEXT based on MPYC and bit15 of unshifted RES3.
MPYFRAC=1 ?
Yes
No
MPYSAT=1 ?
Yes
No
32-bit Saturation
64-bit Saturation
MPYSAT=1 ?
Yes
No
Multiplication completed

Figure 10-5. Multiplication Flow Chart

Given the separation in processing of 16-bit operations (32-bit results) and 32-bit operations (64-bit results) by the module, it is important to understand the implications when using MAC/MACS operations and mixing 16-bit operands/results with 32-bit operands/results. User software must address these points during usage when mixing these operations. The following code snippet illustrates the issue.

```
; Mixing 32x24 multiplication with 16x16 MACS operation
   MOV      #MPYSAT,&MPY32CTL0   ; Saturation mode
   MOV      #052C5h,&MPY32L      ; Load low word of 1st operand
   MOV      #06153h,&MPY32H      ; Load high word of 1st operand
   MOV      #001ABh,&OP2L        ; Load low word of 2nd operand
   MOV.B    #023h,&OP2H_B        ; Load high word of 2nd operand
                                 ;... 5 NOPs required
   MOV      &RES0,R6             ; R6 = 00E97h
   MOV      &RES1,R7             ; R7 = 0A6EAh
   MOV      &RES2,R8             ; R8 = 04F06h
   MOV      &RES3,R9             ; R9 = 0000Dh
                                 ; Note that MPYC = 0!
   MOV      #0CCC3h,&MACS        ; Signed MAC operation
   MOV      #0FFB6h,&OP2         ; 16x16 bit operation
   MOV      &RESLO,R6            ; R6 = 0FFFFh
   MOV      &RESHI,R7            ; R7 = 07FFFh
```

The second operation gives a saturated result because the 32-bit value used for the 16×16 bit MACS operation was already saturated when the operation was started: the carry bit MPYC was 0 from the previous operation but the most significant bit in result register RES1 is set. As one can see in the flow chart the content of the result registers are saturated for multiply-and-accumulate operations after starting a new operation based on the previous results but depending on the size of the result (32-bit or 64-bit) of the newly initiated operation.

The saturation before the multiplication can cause issues if the MPYC bit is not properly set as the following code example illustrates.

```
   ;Pre-load result registers to demonstrate overflow
   MOV      #0,&RES3          ;
   MOV      #0,&RES2          ;
   MOV      #0,&RES1          ;
   MOV      #0,&RES0          ;
   ; Saturation mode and set MPYC:
   MOV      #MPYSAT+MPYC,&MPY32CTL0
   MOV.B    #082h,&MACS_B     ; 8-bit signed MAC operation
   MOV.B    #04Fh,&OP2_B      ; Start 16x16 bit operation
   MOV      &RES0,R6          ; R6 = 00000h
   MOV      &RES1,R7          ; R7 = 08000h
```

Even though the result registers were loaded with all zeros the final result is saturated. This is because the MPYC bit was set causing the result used for the multiply-and-accumulate to be saturated to 08000 0000h. Adding a negative number to it would again cause an underflow thus the final result is also saturated to 08000 0000h.

10.2.6 Indirect Addressing of Result Registers

When using indirect or indirect autoincrement addressing mode to access the result registers and the multiplier requires 3 cycles until result availability according to Table 1-1, at least one instruction is needed between loading the second operand and accessing the result registers:

```
; Access multiplier 16x16 results with indirect addressing
   MOV      #RES0,R5         ; RES0 address in R5 for indirect
   MOV      &OPER1,&MPY      ; Load 1st operand
   MOV      &OPER2,&OP2      ; Load 2nd operand
   NOP                       ; Need one cycle
   MOV      @R5+,&xxx        ; Move RES0
   MOV      @R5,&xxx         ; Move RES1
```

In case of a 32×16 multiplication there is also one instruction required between reading the first result register RES0 and the second result register RES1:

```
; Access multiplier 32x16 results with indirect addressing
   MOV   #RES0,R5          ; RES0 address in R5 for indirect
   MOV   &OPER1L,&MPY32L   ; Load low word of 1st operand
   MOV   &OPER1H,&MPY32H   ; Load high word of 1st operand
   MOV   &OPER2,&OP2       ; Load 2nd operand (16 bits)
   NOP                     ; Need one cycle
   MOV   @R5+,&xxx         ; Move RES0
   NOP                     ; Need one additional cycle
   MOV   @R5,&xxx          ; Move RES1
                           ; No additional cycles required!
   MOV   @R5,&xxx          ; Move RES2
```

10.2.7 Using Interrupts

If an interrupt occurs after writing OP1, but before writing OP2, and the multiplier is used in servicing that interrupt, the original multiplier mode selection is lost and the results are unpredictable. To avoid this, disable interrupts before using the hardware multiplier, do not use the multiplier in interrupt service routines, or use the save and restore functionality of the 32-bit multiplier.

```
; Disable interrupts before using the hardware multiplier
   DINT                 ; Disable interrupts
   NOP                  ; Required for DINT
   MOV    #xxh,&MPY     ; Load 1st operand
   MOV    #xxh,&OP2     ; Load 2nd operand
   EINT                 ; Interrupts may be enabled before
                        ; processing results if result
                        ; registers are stored and restored in
                        ; interrupt service routines
```

Save and Restore

If the multiplier is used in interrupt service routines its state can be saved and restored using the MPY32CTL0 register. The following code example shows how the complete multiplier status can be saved and restored to allow interruptible multiplications together with the usage of the multiplier in interrupt service routines. Since the state of the MPYSAT and MPYFRAC bits are unknown they should be cleared before the registers are saved as shown in the code example.

```
; Interrupt service routine using multiplier
MPY_USING_ISR
   PUSH   &MPY32CTL0     ; Save multiplier mode, etc.
   BIC    #MPYSAT+MPYFRAC,&MPY32CTL0
                         ; Clear MPYSAT+MPYFRAC
   PUSH   &RES3          ; Save result 3
   PUSH   &RES2          ; Save result 2
   PUSH   &RES1          ; Save result 1
   PUSH   &RES0          ; Save result 0
   PUSH   &MPY32H        ; Save operand 1, high word
   PUSH   &MPY32L        ; Save operand 1, low word
   PUSH   &OP2H          ; Save operand 2, high word
   PUSH   &OP2L          ; Save operand 2, low word
                         ;
   ...                   ; Main part of ISR
                         ; Using standard MPY routines
                         ;
   POP    &OP2L          ; Restore operand 2, low word
   POP    &OP2H          ; Restore operand 2, high word
                         ; Starts dummy multiplication but
                         ; result is overwritten by
                         ; following restore operations:
   POP    &MPY32L        ; Restore operand 1, low word
   POP    &MPY32H        ; Restore operand 1, high word
   POP    &RES0          ; Restore result 0
   POP    &RES1          ; Restore result 1
   POP    &RES2          ; Restore result 2
   POP    &RES3          ; Restore result 3
   POP    &MPY32CTL0     ; Restore multiplier mode, etc.
   reti                  ; End of interrupt service routine
```

10.2.8 Using DMA

In devices with a DMA controller the multiplier can trigger a transfer when the complete result is available. The DMA controller needs to start reading the result with MPY32RES0 successively up to MPY32RES3. Not all registers need to be read. The trigger timing is such that the DMA controller starts reading MPY32RES0 when its ready and that the MPY32RES3 can be read exactly in the clock cycle when it is available to allow fastest access via DMA. The signal into the DMA controller is 'Multiplier ready'. Please refer to the DMA user's guide chapter for details.

10.3 32-Bit Hardware Multiplier Registers

The 32-bit hardware multiplier registers are listed in Table 10-7.

Table 10-7. 32-Bit Hardware Multiplier Registers

Register	Short Form	Register Type	Address	Initial State
16-bit operand one – multiply	MPY	Read/write	0130h	Unchanged
8-bit operand one – multiply	MPY_B	Read/write	0130h	Unchanged
16-bit operand one – signed multiply	MPYS	Read/write	0132h	Unchanged
8-bit operand one – signed multiply	MPYS_B	Read/write	0132h	Unchanged
16-bit operand one – multiply accumulate	MAC	Read/write	0134h	Unchanged
8-bit operand one – multiply accumulate	MAC_B	Read/write	0134h	Unchanged
16-bit operand one – signed multiply accumulate	MACS	Read/write	0136h	Unchanged
8-bit operand one – signed multiply accumulate	MACS_B	Read/write	0136h	Unchanged
16-bit operand two	OP2	Read/write	0138h	Unchanged
8-bit operand two	OP2_B	Read/write	0138h	Unchanged
16x16-bit result low word	RESLO	Read/write	013Ah	Undefined
16x16-bit result high word	RESHI	Read/write	013Ch	Undefined
16x16-bit sum extension register	SUMEXT	Read	013Eh	Undefined
32-bit operand 1 – multiply – low word	MPY32L	Read/write	0140h	Unchanged
32-bit operand 1 – multiply – high word	MPY32H	Read/write	0142h	Unchanged
24-bit operand 1 – multiply – high byte	MPY32H_B	Read/write	0142h	Unchanged
32-bit operand 1 – signed multiply – low word	MPYS32L	Read/write	0144h	Unchanged
32-bit operand 1 – signed multiply – high word	MPYS32H	Read/write	0146h	Unchanged
24-bit operand 1 – signed multiply – high byte	MPYS32H_B	Read/write	0146h	Unchanged
32-bit operand 1 – multiply accumulate – low word	MAC32L	Read/write	0148h	Unchanged
32-bit operand 1 – multiply accumulate – high word	MAC32H	Read/write	014Ah	Unchanged
24-bit operand 1 – multiply accumulate – high byte	MAC32H_B	Read/write	014Ah	Unchanged
32-bit operand 1 – signed multiply accumulate – low word	MACS32L	Read/write	014Ch	Unchanged
32-bit operand 1 – signed multiply accumulate – high word	MACS32H	Read/write	014Eh	Unchanged
24-bit operand 1 – signed multiply accumulate – high byte	MACS32H_B	Read/write	014Eh	Unchanged
32-bit operand 2 – low word	OP2L	Read/write	0150h	Unchanged
32-bit operand 2 – high word	OP2H	Read/write	0152h	Unchanged
24-bit operand 2 – high byte	OP2H_B	Read/write	0152h	Unchanged
32x32-bit result 0 – least significant word	RES0	Read/write	0154h	Undefined
32x32-bit result 1	RES1	Read/write	0156h	Undefined
32x32-bit result 2	RES2	Read/write	0158h	Undefined
32x32-bit result 3 – most significant word	RES3	Read/write	015Ah	Undefined
MPY32 control register 0	MPY32CTL0	Read/write	015Ch	Undefined

The registers listed in Table 10-8 are treated equally.

Table 10-8. Alternative Registers

Register	Alternative 1	Alternative 2
16-bit operand one – multiply	MPY	MPY32L
8-bit operand one – multiply	MPY_B	MPY32L_B
16-bit operand one – signed multiply	MPYS	MPYS32L
8-bit operand one – signed multiply	MPYS_B	MPYS32L_B
16-bit operand one – multiply accumulate	MAC	MAC32L
8-bit operand one – multiply accumulate	MAC_B	MAC32L_B
16-bit operand one – signed multiply accumulate	MACS	MACS32L
8-bit operand one – signed multiply accumulate	MACS_B	MACS32L_B
16x16-bit result low word	RESLO	RES0
16x16-bit result high word	RESHI	RES1

MPY32CTL0, 32-Bit Multiplier Control Register 0

15	14	13	12	11	10	9	8
Reserved						**MPYDLY32**	**MPYDLY WRTEN**
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

7	6	5	4	3	2	1	0
MPYOP2_32	**MPYOP1_32**	**MPYMx**		**MPYSAT**	**MPYFRAC**	**Reserved**	**MPYC**
rw	rw	rw	rw	rw-0	rw-0	rw-0	rw

Reserved Bits 15-10 Reserved

MPYDLY32 Bit 9 Delayed write mode

0 Writes are delayed until 64-bit result (RES0 to RES3) is available.

1 Writes are delayed until 32-bit result (RES0 to RES1) is available.

MPYDLYWRTEN Bit 8 Delayed write enable

All writes to any MPY32 register are delayed until the 64-bit (MPYDLY32 = 0) or 32-bit (MPYDLY32 = 1) result is ready.

0 Writes are not delayed.

1 Writes are delayed.

MPYOP2_32 Bit 7 Multiplier bit width of operand 2

0 16 bits

1 32 bits

MPYOP1_32 Bit 6 Multiplier bit width of operand 1

0 16 bits

1 32 bits

MPYMx Bits 5-4 Multiplier mode

00 MPY – Multiply

01 MPYS – Signed multiply

10 MAC – Multiply accumulate

11 MACS – Signed multiply accumulate

MPYSAT Bit 3 Saturation mode

0 Saturation mode disabled

1 Saturation mode enabled

MPYFRAC Bit 2 Fractional mode

0 Fractional mode disabled

1 Fractional mode enabled

Reserved Bit 1 Reserved

MPYC Bit 0 Carry of the multiplier. It can be considered as 33rd or 65th bit of the result if fractional or saturation mode is not selected because the MPYC bit does not change when switching to saturation or fractional mode.

It is used to restore the SUMEXT content in MAC mode.

0 No carry for result

1 Result has a carry

Chapter 11
SLAU208–June 2008

CRC Module

The Cyclic Redundancy Check module provides a signature for a given data sequence.

Topic **Page**

11.1 CRC Module Introduction

The CRC module produces a signature for a given sequence of memory data bus values. The signature is generated through a feedback path from data bus bits 0, 4, 11, and 15. See also Figure 11-1. The CRC signature is based on the polynomial given in the CRC-CCITT-BR polynomial (see Equation 11-1) .

$$f(x) = x^{16} + x^{12} + x^{5} +1 \tag{11-1}$$

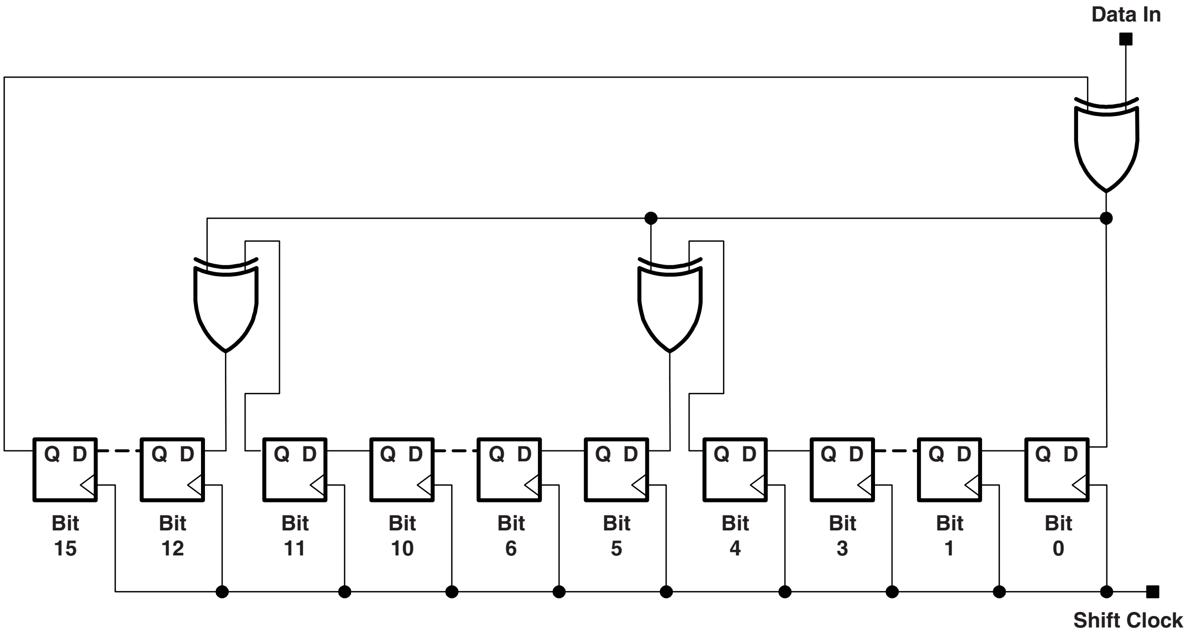


Figure 11-1. LFSR Implementation of the CRC-CCITT Standard, Bit 0 is the MSB of the result

Identical bus sequences result into identical signatures when the CRC is initialized with a fixed seed value, whereas different sequences of input data in general result in different signatures.

11.2 CRC Checksum Generation

The CRC generator is at first initialized by writing a 16-bit word (seed) to the CRC initialization and result register (CRCINIRES). Any data that should be included into the CRC calculation has to be written to the CRC data input register (CRCDI) in the same order as the CRC signature was calculated originally. The actual signature can be read from the initialization and result register (CRCINIRES) to compare the checksum with the expected checksum.

The signature generation (Check Sum) describes a method how the result of a signature operation can be calculated. The calculated signature is called Check Sum in the following text. This calculation is done by an external tool. The Check Sum is stored in the product's memory and is used to check the correctness of the result of the CRC operation.

11.2.1 CRC Implementation

To allow parallel processing of the CRC the linear-feedback-shift-register (LFSR) functionality is implemented with an XOR Tree. This implementation shows the identical behavior as the LFSR approach after 8-bits of data are shifted in when the LSB is 'shifted' in first. The generation of a signature calculation has to be started by writing a seed to the initialization and result register CRCINIRES to initialize the register. Software or hardware (e.g. DMA) can transfer data to the data in register (CRCDI) (e.g. from memory). The value in the data in register is then included into the signature and the result is available in the signature result register at the next read access (CRCINIRES). The signature can be generated using word or byte data. If a word is processed the lower byte at the even address is used at the first clock (MCLK) cycle. During the second clock cycle the higher byte is processed. Thus it takes two clock cycles to process word data while it takes only one clock (MCLK) cycle to process byte data. If the Check Sum itself (with reversed bit order) is included into the CRC operation (as data written to CRCDI) the result in CRCINIRES register must be zero.

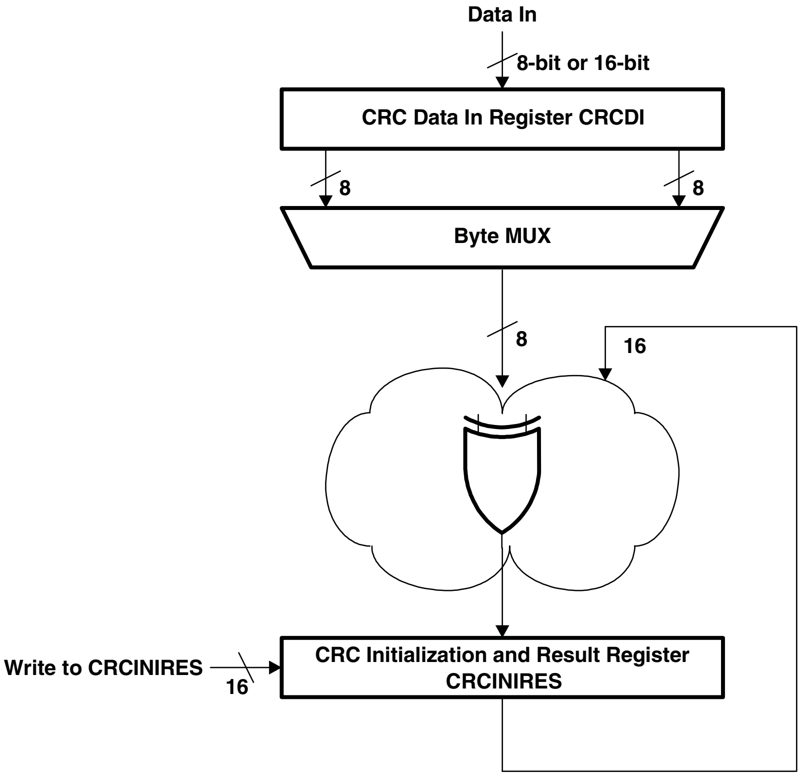

Data In
8-bit or 16-bit
CRC Data In Register CRCDI
8
8
Byte MUX
8
16
Write to CRCINIRES
16
CRC Initialization and Result Register
CRCINIRES

Figure 11-2. Implementation of the CRC-CCITT

11.2.2 Assembler Examples

General Assembler Example

An example demonstrates the operation of the on-chip CRC check:

```
   ...
   PUSH   R4                 ; Save registers
   PUSH   R5
   MOV    #StartAddress,R4   ; StartAddress < EndAddress
   MOV    #EndAddress,R5
   MOV    &INIT, &CRCINIRES  ; INIT to CRCINIRES
L1 MOV    @R4+,&CRCDI        ; Item to Data In register
   CMP    R5,R4              ; End address reached?
   JLO    L1                 ; No
   MOV    &Check_Sum,&CRCDI  ; Yes, Include checksum
   TST    &CRCINIRES         ; Result = 0?
   JNZ    CRC_ERROR          ; No, CRCRES <> 0: error
   ...                       ; Yes, CRCRES=0:
                             ; information ok.
   POP    R5                 ; Restore registers
   POP    R4
```

Reference Data Sequence

The details of the implemented CRC checking algorithm is shown by the data sequence below:

```
   ...
mov      #0FFFFh,&CRC16RES     ; initialize CRC16
mov.b    #00031h,&CRC16DI      ; "1"
mov.b    #00032h,&CRC16DI      ; "2"
mov.b    #00033h,&CRC16DI      ; "3"
mov.b    #00034h,&CRC16DI      ; "4"
mov.b    #00035h,&CRC16DI      ; "5"
mov.b    #00036h,&CRC16DI      ; "6"
mov.b    #00037h,&CRC16DI      ; "7"
mov.b    #00038h,CRC16DI       ; "8"
mov.b    #00039h,&CRC16DI      ; "9"

cmp      #089F6h,&CRC16RES     ; compare result
jeq      &Success              ; no error
br       &Error                ; to error handler
   ...
```

11.3 CRC Module Registers

The CRC module registers are listed in Table 11-1. The base address can be found in the device specific datasheet. The address offset is given in Table 11-1.

Table 11-1. CRC Module Registers

Register	Short Form	Register Type	Address	Initial State
CRC data in register	CRCDI	Read/write	0000h	0000h
CRC initialization and result register	CRCINIRES	Read/Write	0004h	FFFFh

CRCDI, Data In Register

15	14	13	12	11	10	9	8
CRCDI							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
CRCDI							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

CRCDI Bits 15-0 CRC data in. Data written to the CRCDI register will be included to the present signature in the CRCINIRES register according to the CRC-CCITT standard.

CRCINIRES, Initialization and Result Register

15	14	13	12	11	10	9	8
CRCINIRES							
rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1

7	6	5	4	3	2	1	0
CRCINIRES							
rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1	rw-1

CRCINIRES Bits 15-0 CRC initialization and result. This register holds the current CRC result (according to the CRC-CCITT standard). Writing to this register initializes the CRC calculation with the value written to it. The value just written can be read from CRCINIRES register.

Chapter 12
SLAU208–June 2008

Timer_A

Timer_A is a 16-bit timer/counter with multiple capture/compare registers. This chapter describes Timer_A is used on MSP430x5xx devices.

Topic **Page**

12.1 Timer_A Introduction

Timer_A is a 16-bit timer/counter with up to seven 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
- Up to seven 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 12-1.

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 will not take place.

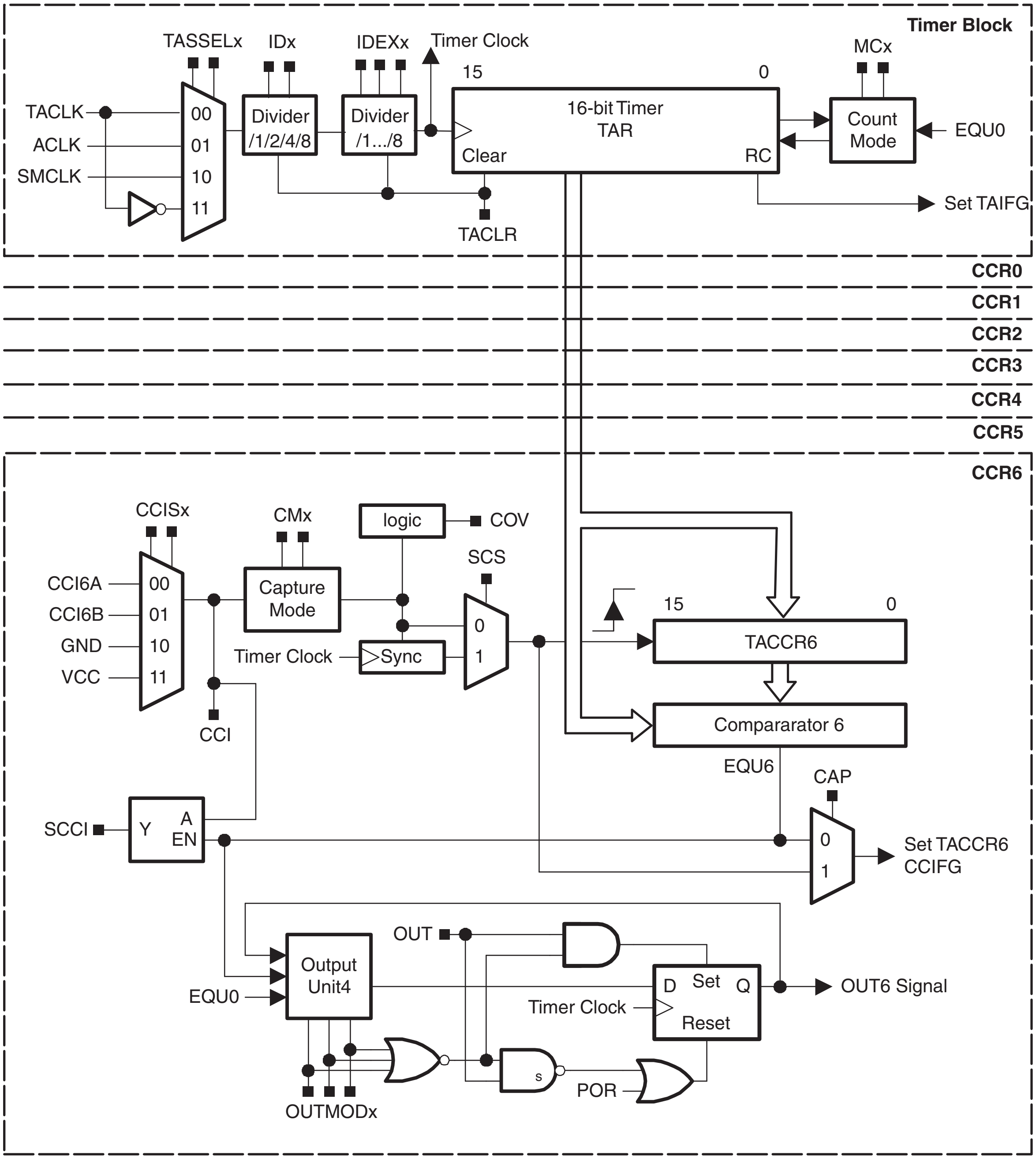


Figure 12-1. Timer_A Block Diagram

12.2 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.

12.2.1 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 TACLK is asynchronous to the CPU clock, any read from TAR should occur while the timer is not operating or the results may be unpredictable. 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 TACLK can be sourced from ACLK, SMCLK, or externally via TACLK. 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 selected clock source can be further divided by 2, 3, 4, 5, 6, 7, or 8 using the IDEXx bits.The TACLK dividers are reset when TACLR is set.

Note: Timer_A Dividers

Setting the TACLR bit will clear the contents of TAR, as well as, the dividers. When the TACLR bit is cleared, the Timer Clock will immediately begin clocking at the first rising edge of the Timer_A clock source selected with the TASSELx bits, and will continue clocking at the divider settings set by the IDx and IDEXx bits.

12.2.2 Starting the Timer

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

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

12.2.3 Timer Mode Control

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

Table 12-1. Timer Modes

MCx	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of 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 backdown 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 12-2. 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.

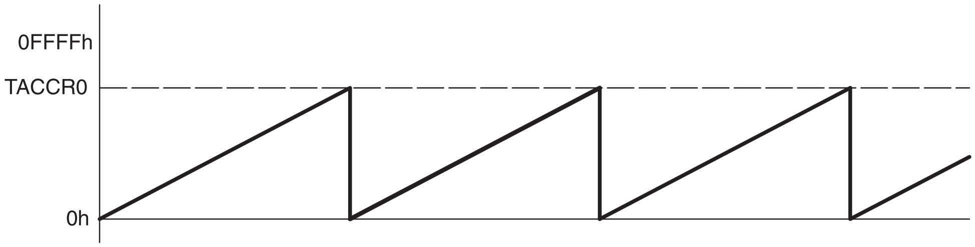


Figure 12-2. Up Mode

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 12-3 shows the flag set cycle.

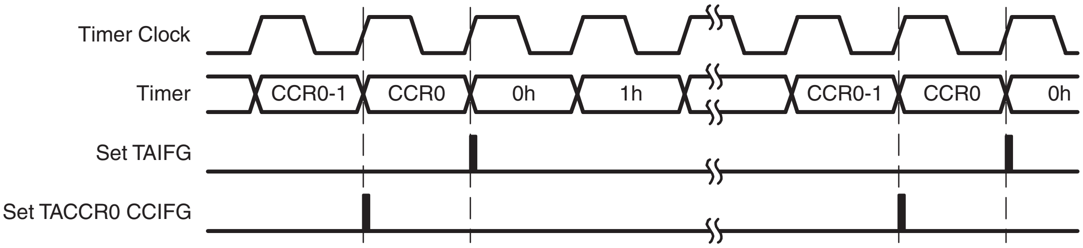


Figure 12-3. Up Mode Flag Setting

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 12-4. The capture/compare register TACCR0 works the same way as the other capture/compare registers.

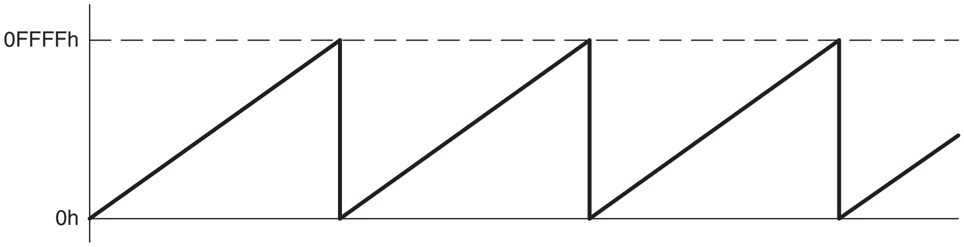


Figure 12-4. Continuous Mode

The TAIFG interrupt flag is set when the timer *counts* from 0FFFFh to zero. Figure 12-5 shows the flag set cycle.

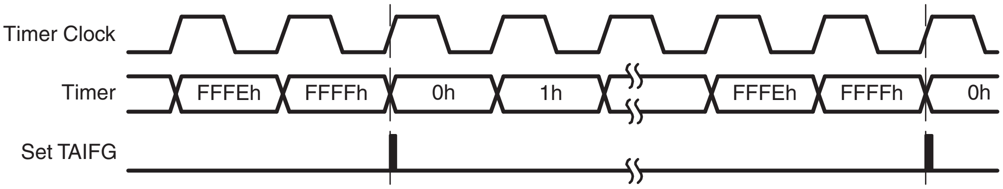


Figure 12-5. Continuous Mode Flag Setting

Use of the Continuous Mode

The continuous mode can be used to generate independent time intervals and output frequencies. Each time an interval is completed, an interrupt is generated. The next time interval is added to the TACCRx register in the interrupt service routine. Figure 12-6 shows two separate time intervals t_0 and t_1 being added to the capture/compare registers. In this usage, the time interval is controlled by hardware, not software, without impact from interrupt latency. Up to n (Timer_An), where n = 0 to 7, independent time intervals or output frequencies can be generated using capture/compare registers.

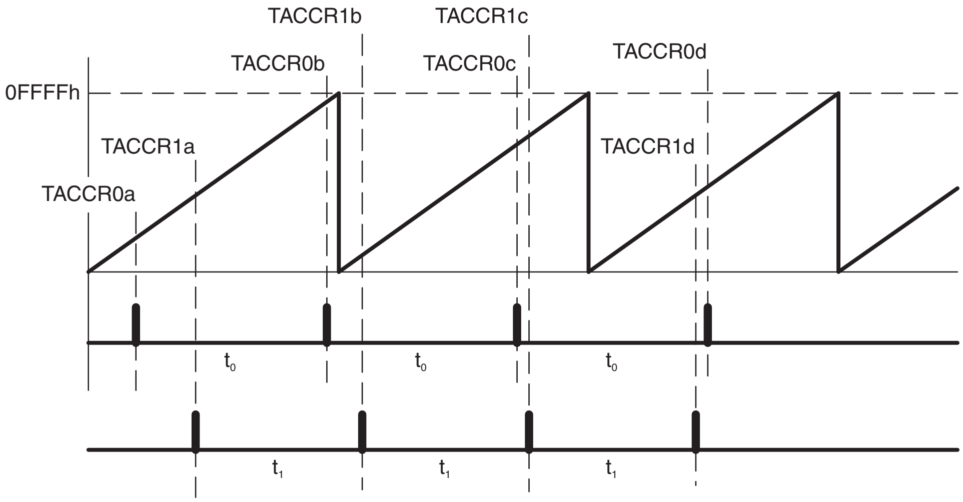


Figure 12-6. Continuous Mode Time Intervals

Time intervals can be produced with other modes as well, where 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 12-7. The period is twice the value in TACCR0.

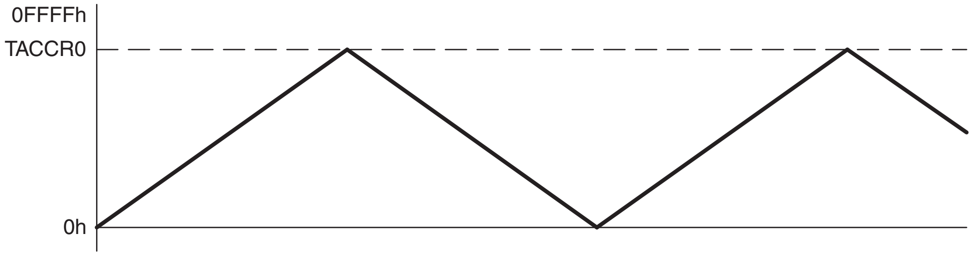


Figure 12-7. Up/Down Mode

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

In up/down mode, the TACCR0 CCIFG interrupt flag and the TAIFG interrupt flag are set only once during a period, separated by 1/2 the timer period. The TACCR0 CCIFG interrupt flag is set when the timer *counts* from TACCR0-1 to TACCR0, and TAIFG is set when the timer completes *counting* down from 0001h to 0000h. Figure 12-8 shows the flag set cycle.

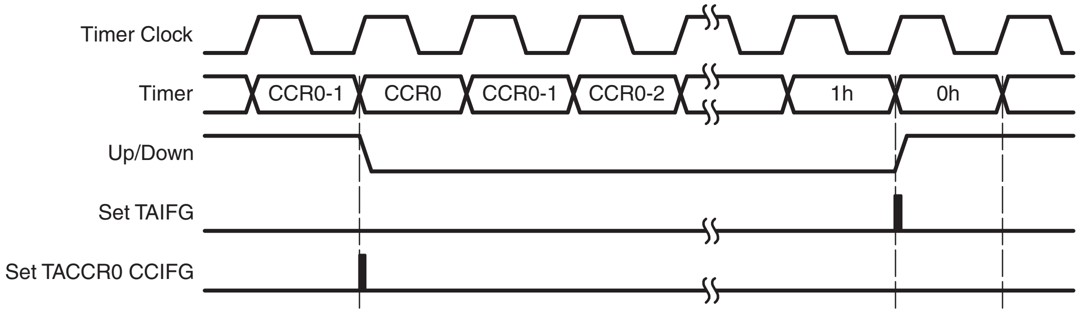


Figure 12-8. Up/Down Mode Flag Setting

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 12-9 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.

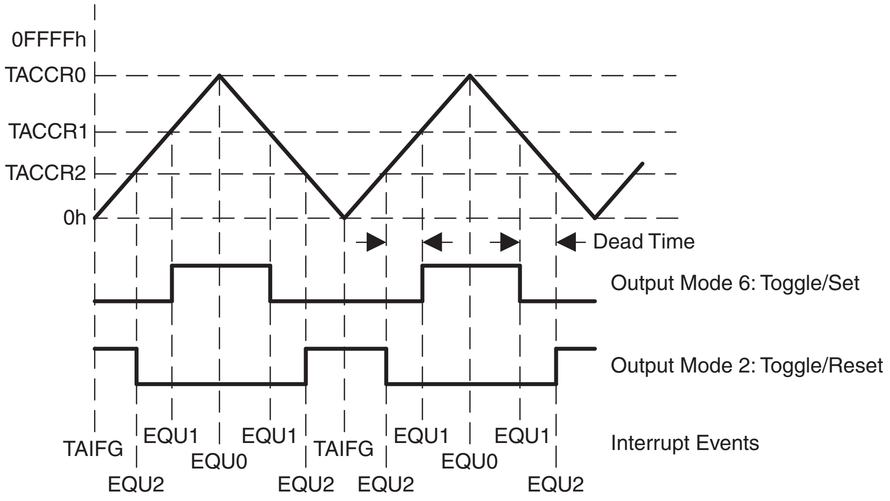


Figure 12-9. Output Unit in Up/Down Mode

12.2.4 Capture/Compare Blocks

Three or five 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. MSP430x5xx family devices may have different signals connected to CCIxA and CCIxB. Refer to the device-specific data sheet for the connections of these signals.

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

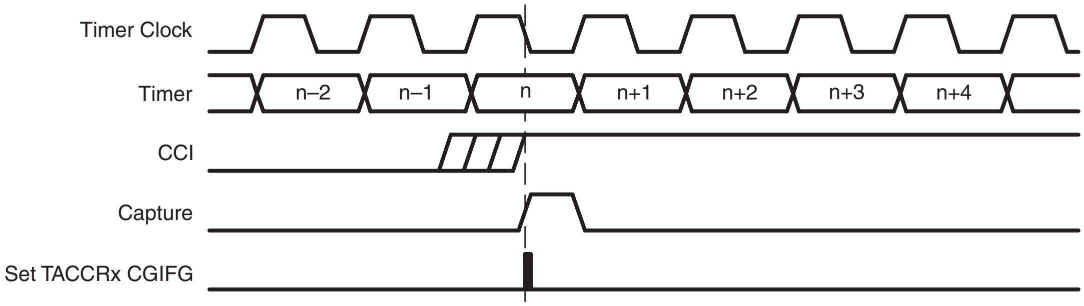


Figure 12-10. Capture Signal (SCS = 1)

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

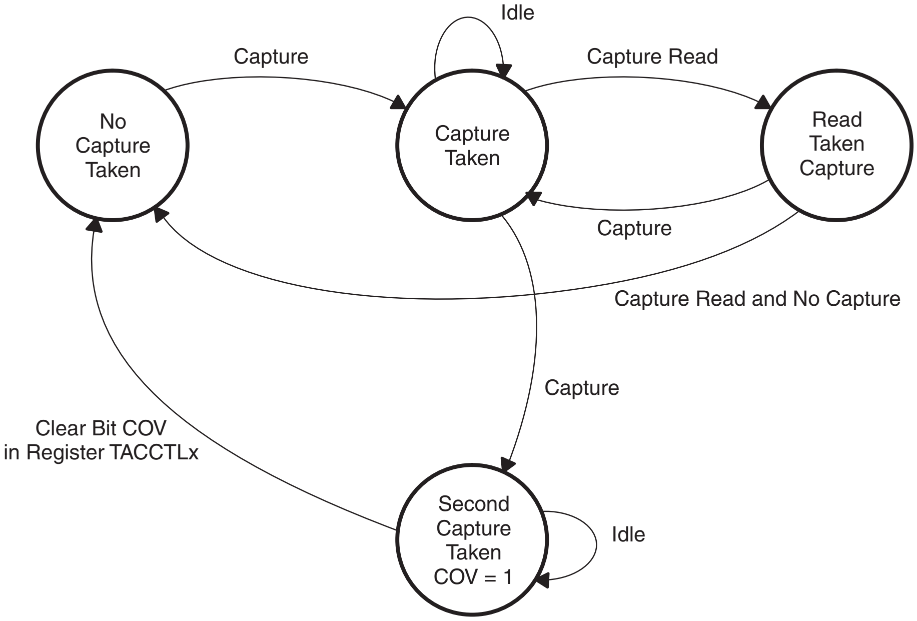

Idle
Capture
Capture Read
No Capture Taken
Capture Taken
Read Taken Capture
Capture
Capture Read and No Capture
Capture
Clear Bit COV in Register TACCTLx
Second Capture Taken COV = 1
Idle

Figure 12-11. Capture Cycle

12.2.4.0.1 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

12.2.5 Output Unit

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

Output Modes

The output modes are defined by the OUTMODx bits and are described in Table 12-2. 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 12-2. 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 12-12 using TACCR0 and TACCR1.

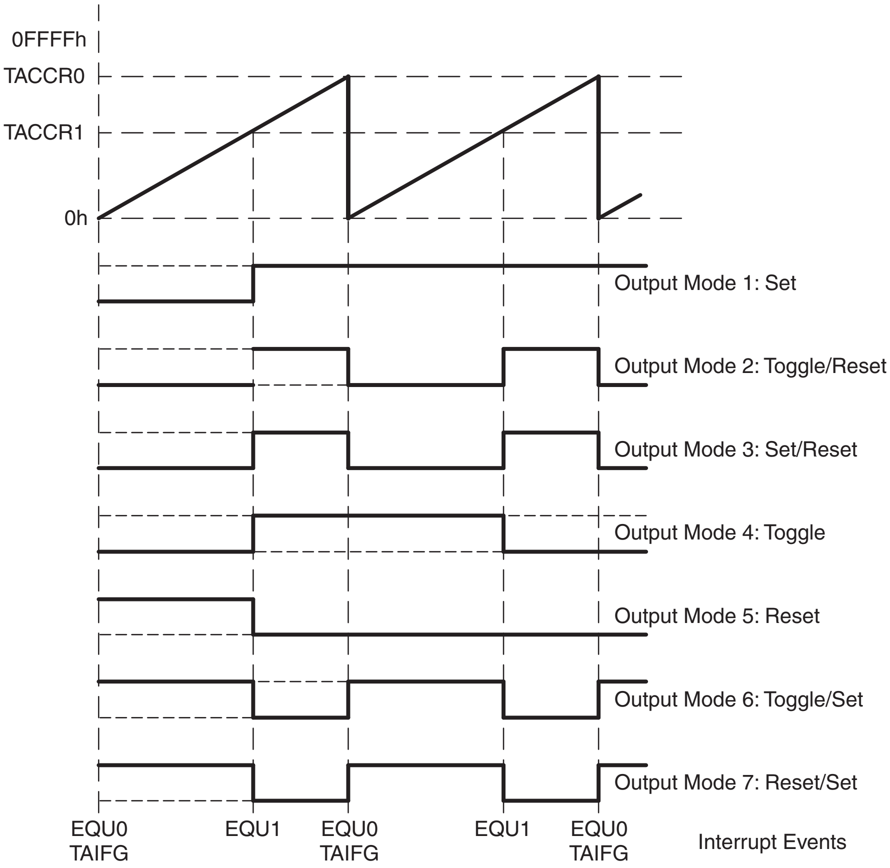


Figure 12-12. Output Example—Timer in Up Mode

Output Example—Timer in Continuous Mode

The OUTx signal is changed when the timer reaches the TACCRx and TACCR0 values, depending on the output mode. An example is shown in Figure 12-13 using TACCR0 and TACCR1.

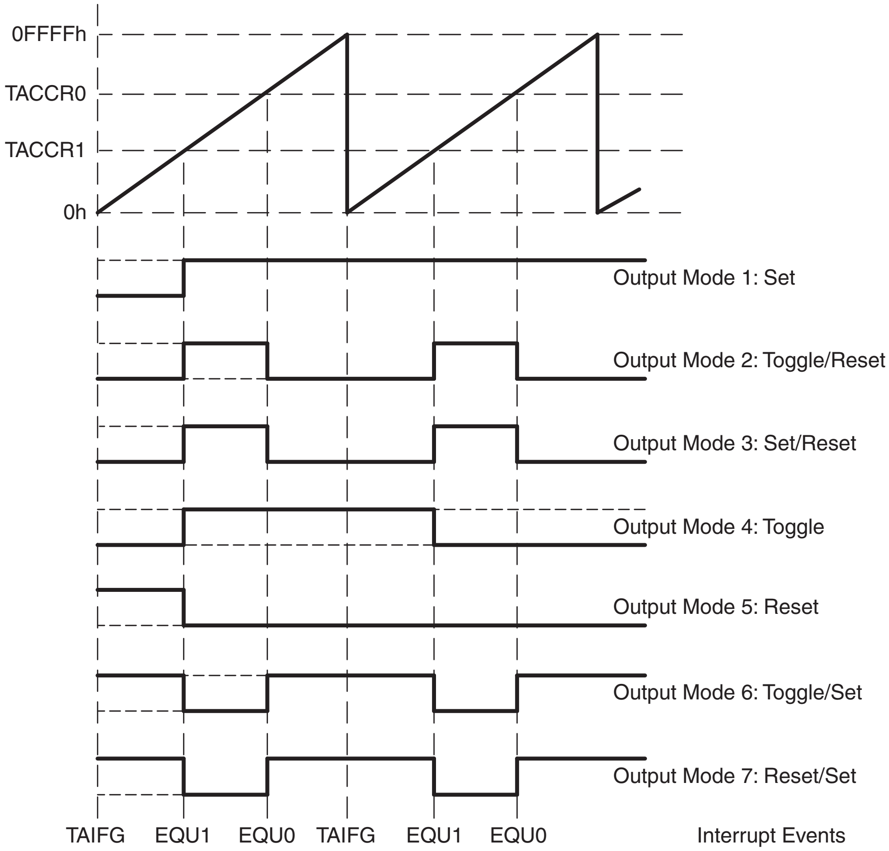


Figure 12-13. Output Example—Timer in Continuous Mode

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 12-14 using TACCR0 and TACCR2.

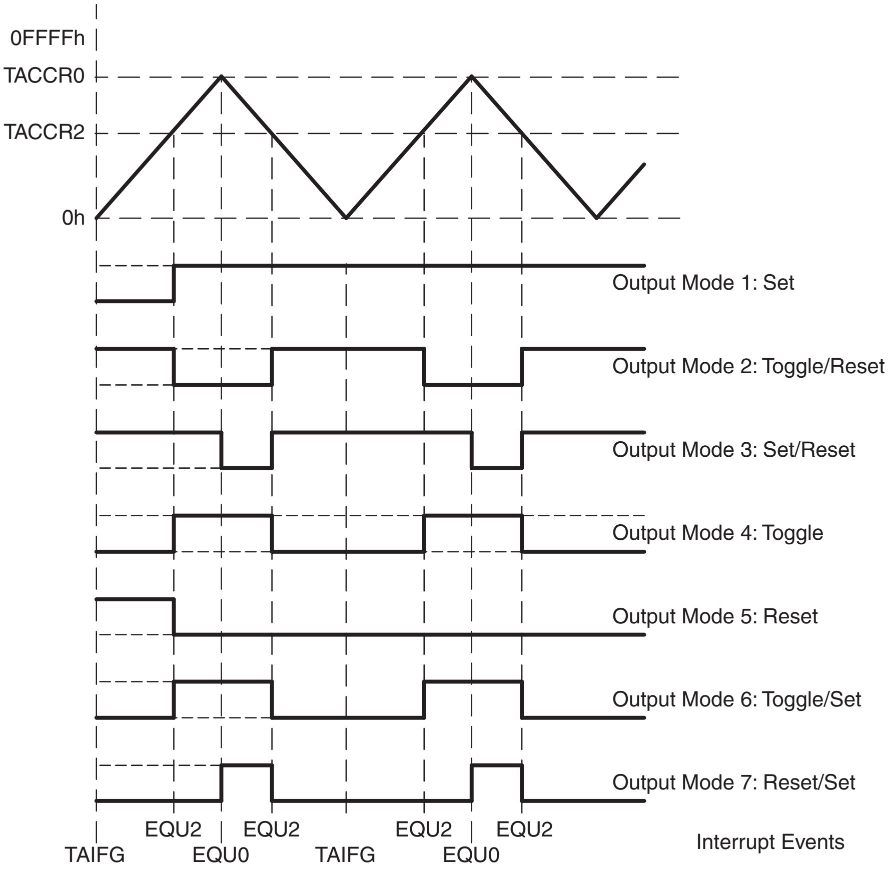


Figure 12-14. Output Example—Timer in Up/Down Mode

Note: Switching Between Output Modes

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

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

12.2.6 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 12-15. The TACCR0 CCIFG flag is automatically reset when the TACCR0 interrupt request is serviced.

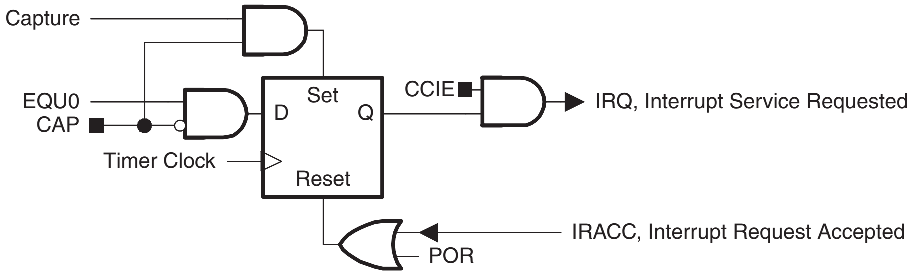

Capture
EQU0
CAP
Timer Clock
Set
D
Q
Reset
CCIE
IRQ, Interrupt Service Requested
IRACC, Interrupt Request Accepted
POR

Figure 12-15. 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 example assumes a Timer_A3 configuration.

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
         RETI                  ; Vector 10: Reserved         5
         RETI                  ; Vector 12: Reserved         5
```

```
TAIFG_HND                      ; Vector 14: 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
```

12.3 Timer_A Registers

The Timer_A registers are listed in Table 12-3 for Timer_A7, which is the largest configuration available. The base address can be found in the device specific data sheet. The address offsets are listed in Table 12-3.

Table 12-3. Timer_A7 Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Timer_A7 control	TACTL	Read/write	Word	00h	0000h
	TACTL_L	Read/write	Byte	00h	00h
	TACTL_H	Read/write	Byte	01h	00h
Timer_A7 capture/compare control 0	TACCTL0	Read/write	Word	02h	0000h
	TACCTL0_L	Read/write	Byte	02h	00h
	TACCTL0_H	Read/write	Byte	03h	00h
Timer_A7 capture/compare control 1	TACCTL1	Read/write	Word	04h	0000h
	TACCTL1_L	Read/write	Byte	04h	00h
	TACCTL1_H	Read/write	Byte	05h	00h
Timer_A7 capture/compare control 2	TACCTL2	Read/write	Word	06h	0000h
	TACCTL2_L	Read/write	Byte	06h	00h
	TACCTL2_H	Read/write	Byte	07h	00h
Timer_A7 capture/compare control 3	TACCTL3	Read/write	Word	08h	0000h
	TACCTL3_L	Read/write	Byte	08h	00h
	TACCTL3_H	Read/write	Byte	09h	00h
Timer_A7 capture/compare control 4	TACCTL4	Read/write	Word	0Ah	0000h
	TACCTL4_L	Read/write	Byte	0Ah	00h
	TACCTL4_H	Read/write	Byte	0Bh	00h
Timer_A7 capture/compare control 5	TACCTL5	Read/write	Word	0Ch	0000h
	TACCTL5_L	Read/write	Byte	0Ch	00h
	TACCTL5_H	Read/write	Byte	0Dh	00h
Timer_A7 capture/compare control 6	TACCTL6	Read/write	Word	0Eh	0000h
	TACCTL6_L	Read/write	Byte	0Eh	00h
	TACCTL6_H	Read/write	Byte	0Fh	00h
Timer_A7 counter	TAR	Read/write	Word	10h	0000h
	TAR_L	Read/write	Byte	10h	00h
	TAR_H	Read/write	Byte	11h	00h
Timer_A7 capture/compare 0	TACCR0	Read/write	Word	12h	0000h
	TACCR0_L	Read/write	Byte	12h	00h
	TACCR0_H	Read/write	Byte	13h	00h
Timer_A7 capture/compare 1	TACCR1	Read/write	Word	14h	0000h
	TACCR1_L	Read/write	Byte	14h	00h
	TACCR1_H	Read/write	Byte	15h	00h
Timer_A7 capture/compare 2	TACCR2	Read/write	Word	16h	0000h
	TACCR2_L	Read/write	Byte	16h	00h
	TACCR2_H	Read/write	Byte	17h	00h
Timer_A7 capture/compare 3	TACCR3	Read/write	Word	18h	0000h
	TACCR3_L	Read/write	Byte	18h	00h
	TACCR3_H	Read/write	Byte	19h	00h
Timer_A7 capture/compare 4	TACCR4	Read/write	Word	1Ah	0000h
	TACCR4_L	Read/write	Byte	1Ah	00h

Table 12-3. Timer_A7 Registers (continued)

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
	TACCR4_H	Read/write	Byte	1Bh	00h
Timer_A7 capture/compare 5	TACCR5	Read/write	Word	1Ch	0000h
	TACCR5_L	Read/write	Byte	1Ch	00h
	TACCR5_H	Read/write	Byte	1Dh	00h
Timer_A7 capture/compare 6	TACCR6	Read/write	Word	1Eh	0000h
	TACCR6_L	Read/write	Byte	1Eh	00h
	TACCR6_H	Read/write	Byte	1Fh	00h
Timer_A7 Interrupt Vector	TAIV	Read only	Word	2Eh	0000h
	TAIV_L	Read only	Byte	2Eh	00h
	TAIV_H	Read only	Byte	2Fh	00h
Timer_A7 Extension	TAEX0	Read/write	Word	20h	0000h
	TAEX0_L	Read/write	Byte	20h	00h
	TAEX0_H	Read/write	Byte	21h	00h

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

00 TACLK
01 ACLK
10 SMCLK
11 Inverted TACLK

IDx Bits 7-6 Input divider. These bits along with the IDEXx bits select the divider for the input clock.

00 /1
01 /2
10 /4
11 /8

MCx Bits 5-4 Mode control. Setting MCx = 00h when Timer_A is not in use conserves power.

00 Stop mode: the timer is halted
01 Up mode: the timer counts up to TACCR0
10 Continuous mode: the timer counts up to 0FFFFh
11 Up/down mode: the timer counts up to TACCR0 then down to 0000h

Unused Bit 3 Unused

TACLR Bit 2 Timer_A clear. Setting this bit resets TAR, the TACLK 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.

0 Interrupt disabled
1 Interrupt enabled

TAIFG Bit 0 Timer_A interrupt flag

0 No interrupt pending
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-(0)	r-(0)	rw-(0)

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

CMx Bit 15-14 Capture mode

00 No capture

01 Capture on rising edge

10 Capture on falling edge

11 Capture on both rising and falling edges

CCISx Bit 13-12 Capture/compare input select. These bits select the TACCRx input signal. See the device-specific data sheet for specific signal connections.

00 CCIxA

01 CCIxB

10 GND

11 V_{CC}

SCS Bit 11 Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.

0 Asynchronous capture

1 Synchronous capture

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

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

CAP Bit 8 Capture mode

0 Compare mode

1 Capture mode

OUTMODx Bits 7-5 Output mode. Modes 2, 3, 6, and 7 are not useful for TACCR0 because EQUx = EQU0.

000 OUT bit value

001 Set

010 Toggle/reset

011 Set/reset

100 Toggle

101 Reset

110 Toggle/set

111 Reset/set

CCIE Bit 4 Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag.

0 Interrupt disabled

1 Interrupt enabled

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

OUT Bit 2 Output. For output mode 0, this bit directly controls the state of the output.

0 Output low

1 Output high

COV Bit 1 Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software.

0 No capture overflow occurred

1 Capture overflow occurred

CCIFG Bit 0 Capture/compare interrupt flag

0	No interrupt pending
1	Interrupt pending

TAIV, Timer_A Interrupt Vector Register

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

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

TAIVx Bits 15-0 Timer_A interrupt vector value

TAIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	Capture/compare 1	TACCR1 CCIFG	Highest
04h	Capture/compare 2	TACCR2 CCIFG	
06h	Capture/compare 3	TACCR3 CCIFG	
08h	Capture/compare 4	TACCR4 CCIFG	
0Ah	Capture/compare 5	TACCR5 CCIFG	
0Ch	Capture/compare 6	TACCR6 CCIFG	
0Eh	Timer overflow	TAIFG	Lowest

TAEX0, Timer_A Expansion Register 0

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

7	6	5	4	3	2	1	0
Unused	**Unused**	**Unused**	**Unused**	**Unused**	**IDEX**		
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

Unused Bits 15-3 Unused. Read only. Always read as 0.

IDEX Bits 2-0 Input divider expansion. These bits along with the IDx bits select the divider for the input clock.

000	/1
001	/2
010	/3
011	/4
100	/5
101	/6
110	/7
111	/8

Chapter 13

SLAU208–June 2008

Timer_B

Timer_B is a 16-bit timer/counter with multiple capture/compare registers. This chapter describes Timer_B is used in MSP430x5xx devices.

Topic **Page**

13.1 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
- Up to 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 13-1.

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 will not take place.

13.1.1 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.

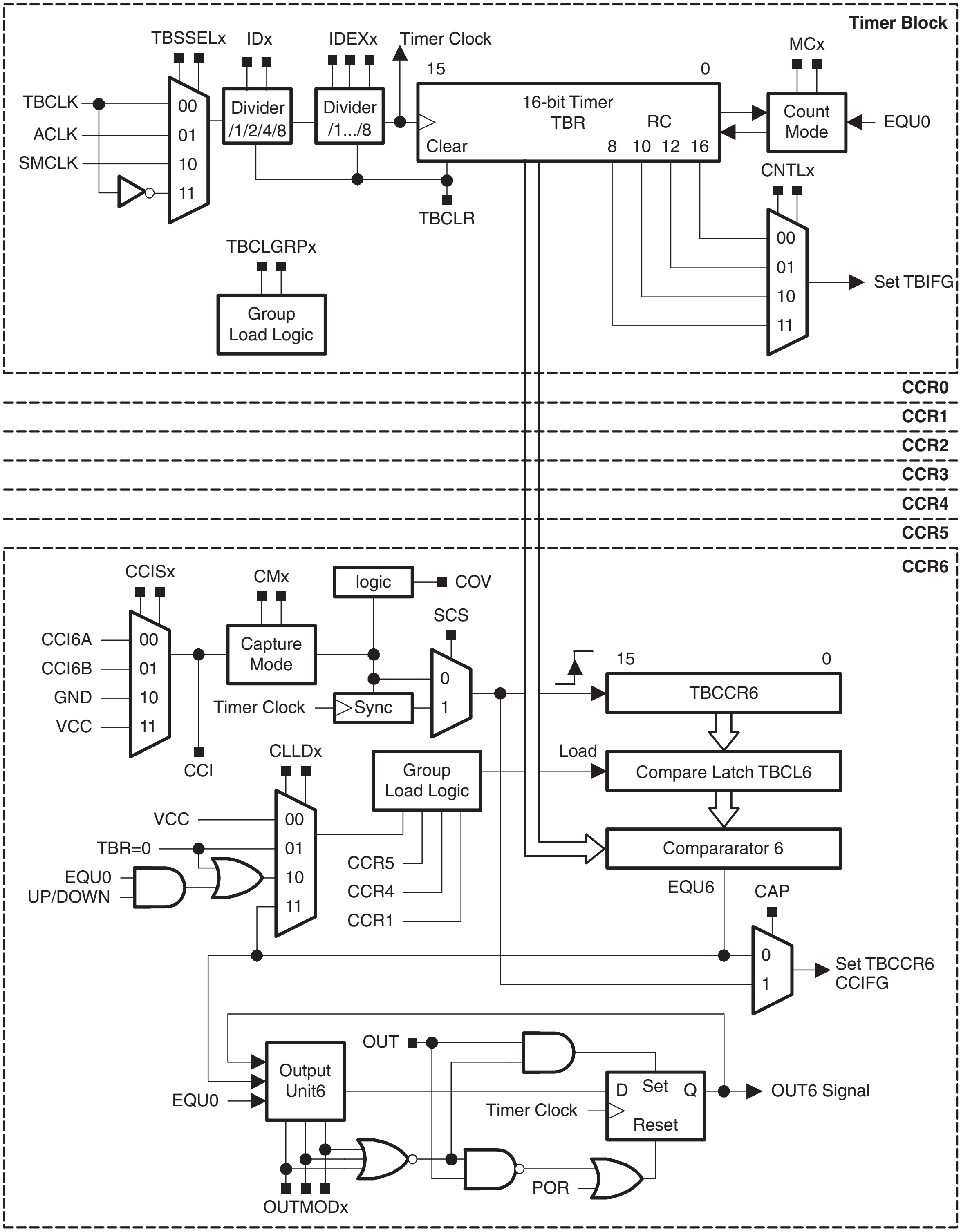


Figure 13-1. Timer_B Block Diagram

13.2 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.

13.2.1 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 TBCLK is asynchronous to the CPU clock, any read from TBR should occur while the timer is not operating or the results may be unpredictable. 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 TBCLK can be sourced from ACLK, SMCLK, or externally via TBCLK. 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 selected clock source can be further divided by 2, 3, 4, 5, 6, 7, or 8 using the IDEXx bits.The TBCLK dividers are reset when TBCLR is set.

Note: Timer_B Dividers

Setting the TBCLR bit will clear the contents of TBR, as well as, the dividers. When the TBCLR bit is cleared, the Timer Clock will immediately begin clocking at the first rising edge of the Timer_B clock source selected with the TBSSELx bits, and will continue clocking at the divider settings set by the IDx and IDEXx bits.

13.2.2 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.

13.2.3 Timer Mode Control

The timer has four modes of operation as described in Table 13-1: stop, up, continuous, and up/down. The operating mode is selected with the MCx bits.

Table 13-1. 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.

13.2.3.1 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 13-2. 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.

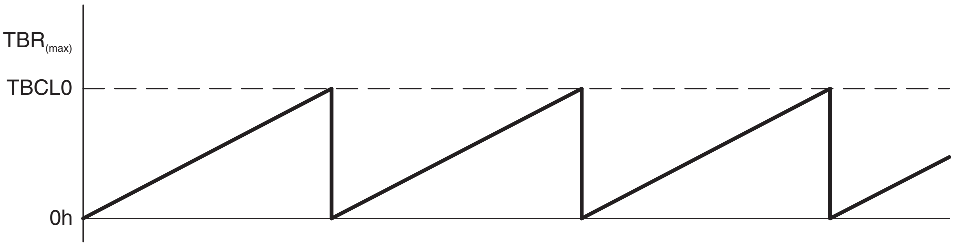


Figure 13-2. Up Mode

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 13-3 shows the flag set cycle.

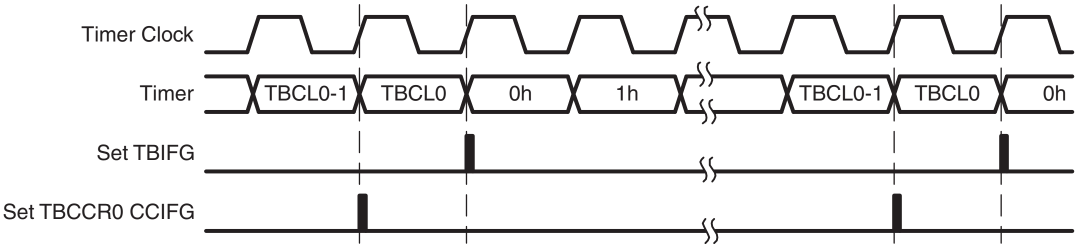


Figure 13-3. Up Mode Flag Setting

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 13-4. The compare latch TBCL0 works the same way as the other capture/compare registers.

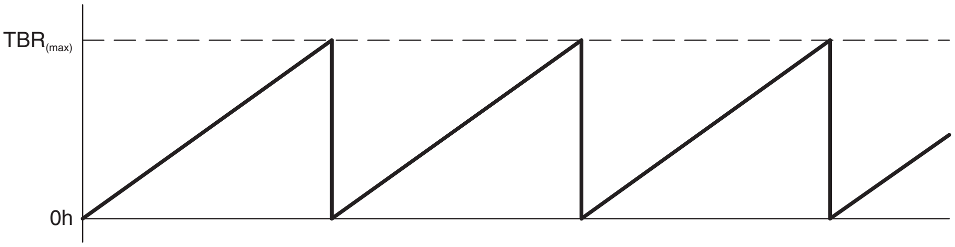


Figure 13-4. Continuous Mode

The TBIFG interrupt flag is set when the timer *counts* from $TBR_{(max)}$ to zero. Figure 13-5 shows the flag set cycle.

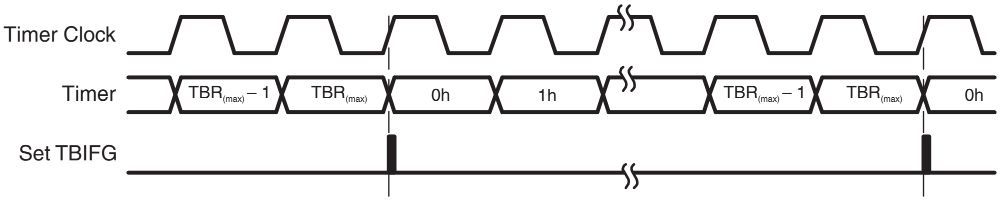


Figure 13-5. Continuous Mode Flag Setting

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 13-6 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 n (Timer_Bn), where n = 0 to 7, independent time intervals or output frequencies can be generated using capture/compare registers.

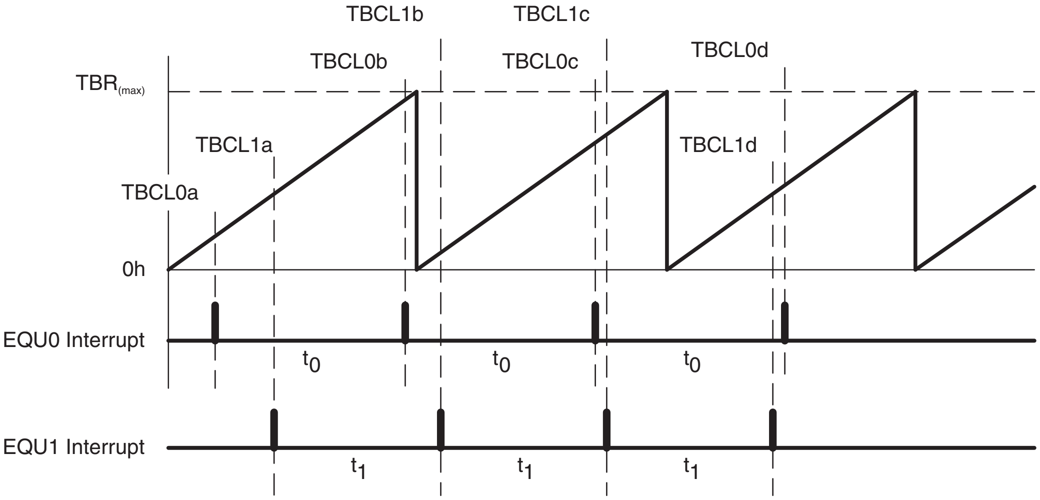


Figure 13-6. Continuous Mode Time Intervals

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 13-7. 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.

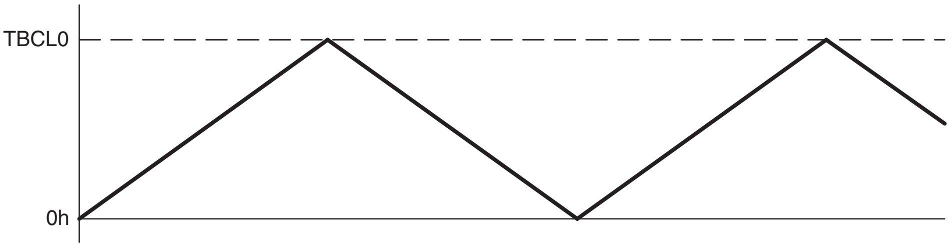


Figure 13-7. Up/Down Mode

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

In up/down mode, the TBCCR0 CCIFG interrupt flag and the TBIFG interrupt flag are set only once during the period, separated by 1/2 the timer period. The TBCCR0 CCIFG interrupt flag is set when the timer *counts* from TBCL0-1 to TBCL0, and TBIFG is set when the timer completes *counting* down from 0001h to 0000h. Figure 13-8 shows the flag set cycle.

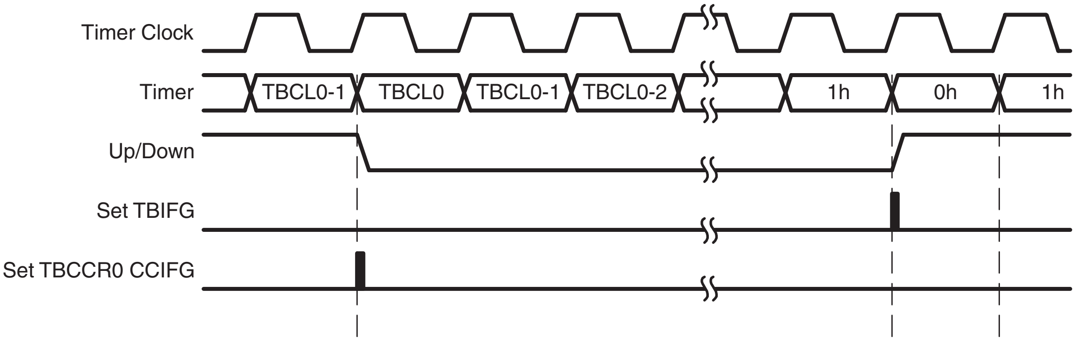


Figure 13-8. Up/Down Mode Flag Setting

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 13-9 the t_{dead} is:

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

With:

t_{dead} = Time during which both outputs need to be inactive
t_{timer} = Cycle time of the timer clock
TBCLx = Content of compare latch x

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

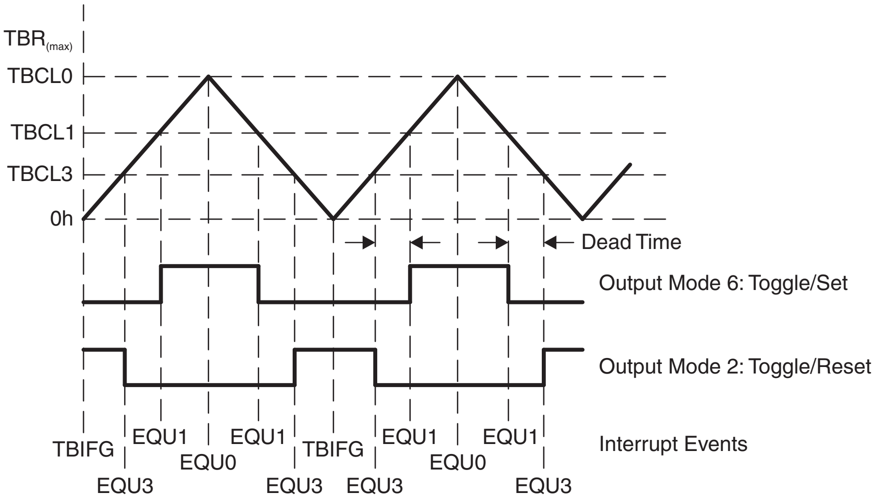


Figure 13-9. Output Unit in Up/Down Mode

13.2.4 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. MSP430x5xx family devices may have different signals connected to CCIxA and CCIxB. Refer to the device-specific data sheet 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 13-10.

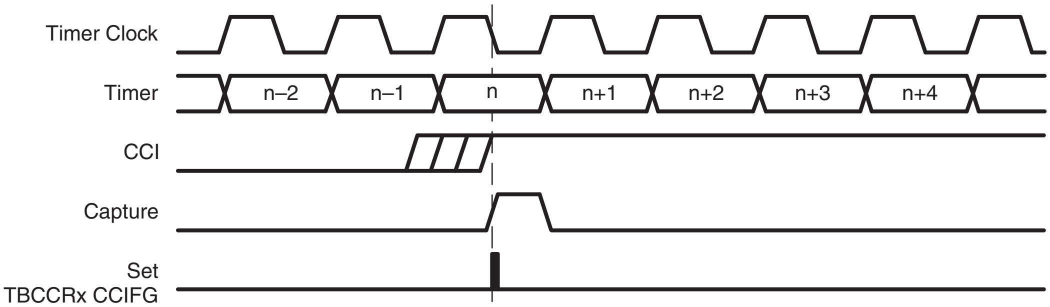


Figure 13-10. Capture Signal (SCS = 1)

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 13-11. COV must be reset with software.

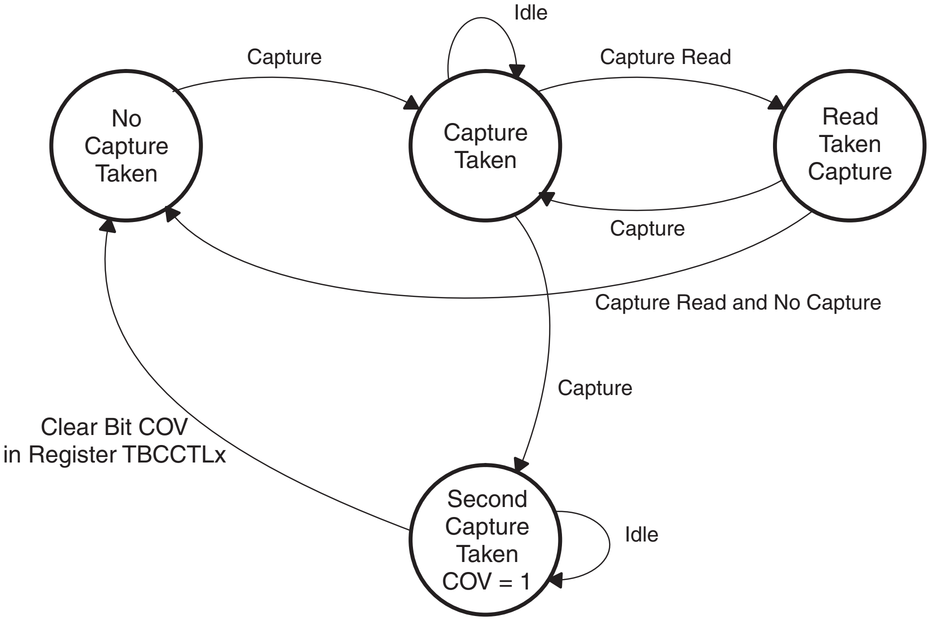


Figure 13-11. Capture Cycle

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 13-2.

Table 13-2. 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 13-3. 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 13-3. Compare Latch Operating Modes

TBCLGRPx	Grouping	Update Control
00	None	Individual
01	TBCL1+TBCL2TBCL3+TBCL4TBCL5+TBCL6	TBCCR1TBCCR3TBCCR5
10	TBCL1+TBCL2+TBCL3TBCL4+TBCL5+TBCL6	TBCCR1TBCCR4
11	TBCL0+TBCL1+TBCL2+ TBCL3+TBCL4+TBCL5+TBCL6	TBCCR1

13.2.5 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 (corresponding PSEL bit is set, and port configured as input), and when the pin is pulled high, all Timer_B outputs are in a high-impedance state.

13.2.5.1 Output Modes

The output modes are defined by the OUTMODx bits and are described in Table 13-4. 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 13-4. 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 13-12 using TBCL0 and TBCL1.

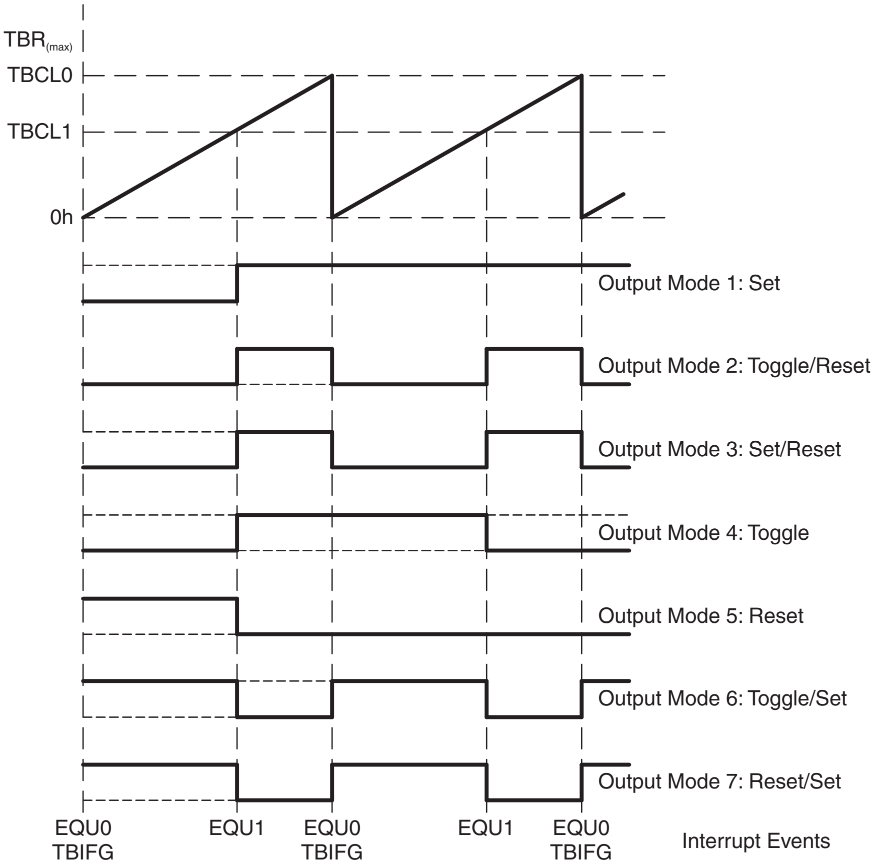


Figure 13-12. Output Example—Timer in Up Mode

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 13-13 using TBCL0 and TBCL1.

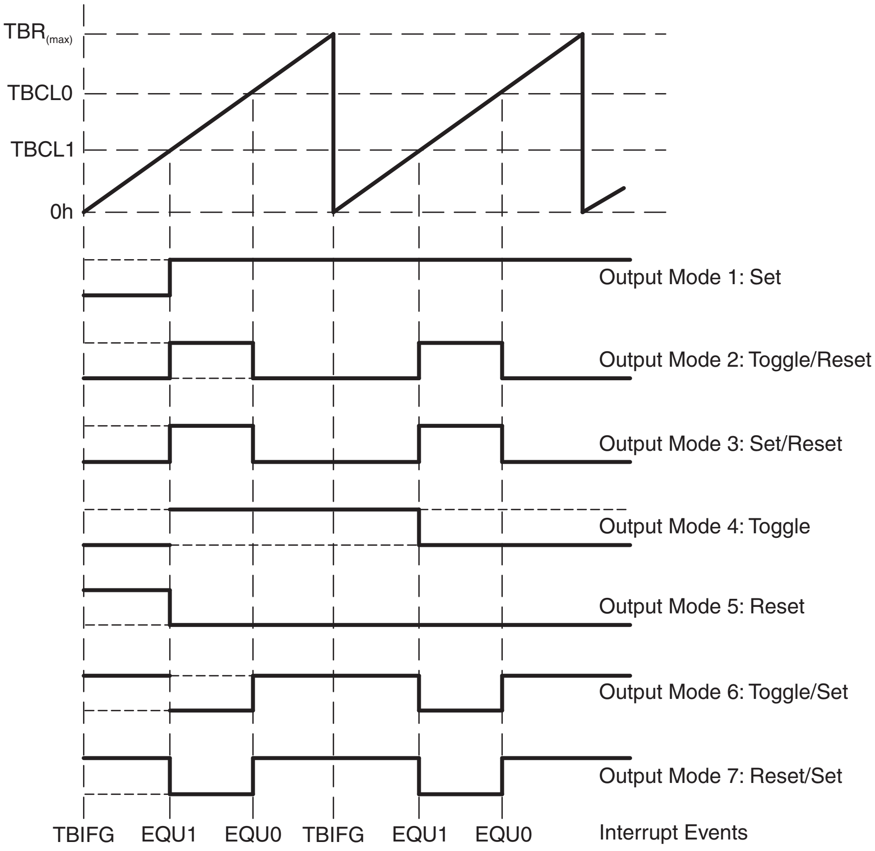


Figure 13-13. Output Example—Timer in Continuous Mode

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 13-14 using TBCL0 and TBCL3.

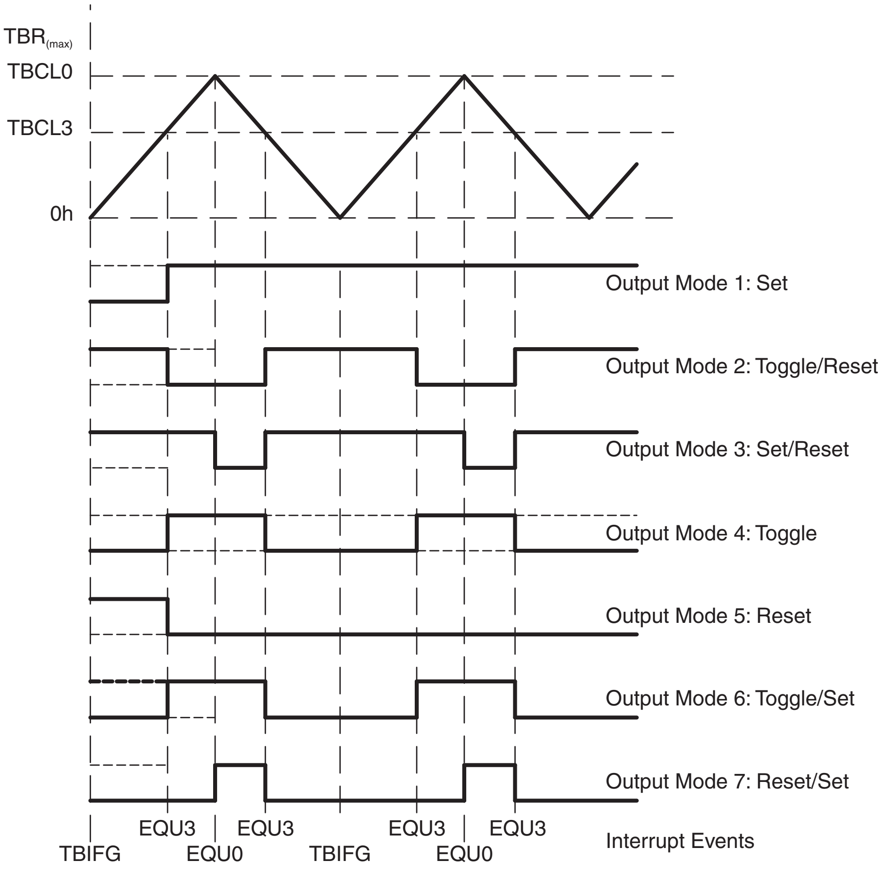


Figure 13-14. Output Example—Timer in Up/Down Mode

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
```

13.2.6 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 13-15. The TBCCR0 CCIFG flag is automatically reset when the TBCCR0 interrupt request is serviced.

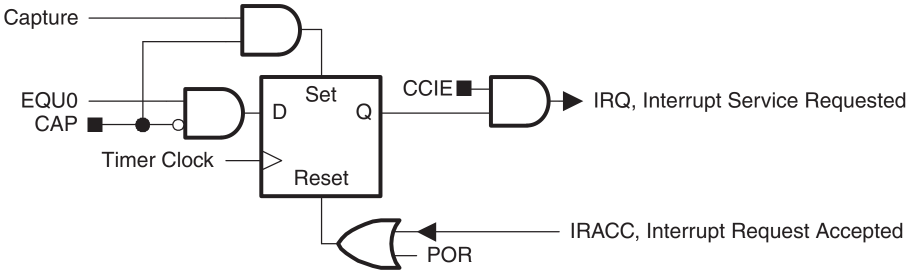

Capture
EQU0
CAP
Timer Clock
Set
D
Q
Reset
CCIE
IRQ, Interrupt Service Requested
IRACC, Interrupt Request Accepted
POR

Figure 13-15. 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: TBIFG 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
```

13.3 Timer_B Registers

The Timer_B registers are listed in Table 13-5. The base address can be found in the device specific data sheet. The address offset is listed in Table 13-5.

Table 13-5. Timer_B Registers

Register	Short Form	Register Type	Address Offset	Initial State
Timer_B control	TBCTL	Read/write	00h	0000h
Timer_B capture/compare control 0	TBCCTL0	Read/write	02h	0000h
Timer_B capture/compare control 1	TBCCTL1	Read/write	04h	0000h
Timer_B capture/compare control 2	TBCCTL2	Read/write	06h	0000h
Timer_B capture/compare control 3	TBCCTL3	Read/write	08h	0000h
Timer_B capture/compare control 4	TBCCTL4	Read/write	0Ah	0000h
Timer_B capture/compare control 5	TBCCTL5	Read/write	0Ch	0000h
Timer_B capture/compare control 6	TBCCTL6	Read/write	0Eh	0000h
Timer_B counter	TBR	Read/write	10h	0000h
Timer_B capture/compare 0	TBCCR0	Read/write	12h	0000h
Timer_B capture/compare 1	TBCCR1	Read/write	14h	0000h
Timer_B capture/compare 2	TBCCR2	Read/write	16h	0000h
Timer_B capture/compare 3	TBCCR3	Read/write	18h	0000h
Timer_B capture/compare 4	TBCCR4	Read/write	1Ah	0000h
Timer_B capture/compare 5	TBCCR5	Read/write	1Ch	0000h
Timer_B capture/compare 6	TBCCR6	Read/write	1Eh	0000h
Timer_B Interrupt Vector	TBIV	Read only	2Eh	0000h
Timer_B Extension	TBEX0	Read/write	20h	0000h

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 along with the IDEXx 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 TBCLK 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.

0 Interrupt disabled

1 Interrupt enabled

TBIFG Bit 0 Timer_B interrupt flag.

0 No interrupt pending

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)	rw-(0)	rw-(0)

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

CMx Bit 15-14 Capture mode

00 No capture

01 Capture on rising edge

10 Capture on falling edge

11 Capture on both rising and falling edges

CCISx Bit 13-12 Capture/compare input select. These bits select the TBCCRx input signal. See the device-specific data sheet for specific signal connections.

00 CCIxA

01 CCIxB

10 GND

11 V_{CC}

SCS Bit 11 Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.

0 Asynchronous capture

1 Synchronous capture

CLLDx Bit 10-9 Compare latch load. These bits select the compare latch load event.

00 TBCLx loads on write to TBCCRx

01 TBCLx loads when TBR *counts* to 0

10 TBCLx loads when TBR *counts* to 0 (up or continuous mode)
TBCLx loads when TBR *counts* to TBCL0 or to 0 (up/down mode)

11 TBCLx loads when TBR *counts* to TBCLx

CAP Bit 8 Capture mode

0 Compare mode

1 Capture mode

OUTMODx Bits 7-5 Output mode. Modes 2, 3, 6, and 7 are not useful for TBCL0 because EQUx = EQU0.

000 OUT bit value

001 Set

010 Toggle/reset

011 Set/reset

100 Toggle

101 Reset

110 Toggle/set

111 Reset/set

CCIE Bit 4 Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag.

0 Interrupt disabled

1 Interrupt enabled

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

OUT Bit 2 Output. 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

TBEX0, Timer_B Expansion Register 0

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

7	6	5	4	3	2	1	0
Unused	**Unused**	**Unused**	**Unused**	**Unused**	**IDEX**		
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

Unused Bits 15-3 Unused. Read only. Always read as 0.

IDEX Bits 2-0 Input divider expansion. These bits along with the IDx bits select the divider for the input clock.

000	/1
001	/2
010	/3
011	/4
100	/5
101	/6
110	/7
111	/8

Chapter 14

SLAU208–June 2008

Real-Time Clock (RTC_A)

The Real-Time Clock module provides clock counters with calendar mode, a flexible programmable alarm, and calibration. This chapter describes the Real-Time Clock (RTC_A) module. The RTC_A is implemented in the MSP430x5xx devices.

Topic **Page**

14.1 Real-Time Clock Introduction

The Real-Time Clock module provides a clock with calendar that can also be configured as a general purpose counter.

Real-Time Clock features include:

- Configurable for Real-Time Clock mode or general purpose counter
- Provides seconds, minutes, hours, day of week, day of month, month and year in calender mode.
- Interrupt capability.
- Selectable BCD or binary format in Real-Time Clock mode
- Programmable alarms in Real-Time Clock mode
- Calibration logic for time offset correction in Real-Time clock mode

The Real-Time Clock block diagram is shown in Figure 14-1.

Note: Real-Time Clock Initialization

Most Real-Time Clock module registers have no initial condition. These registers must be configured by user software before use.

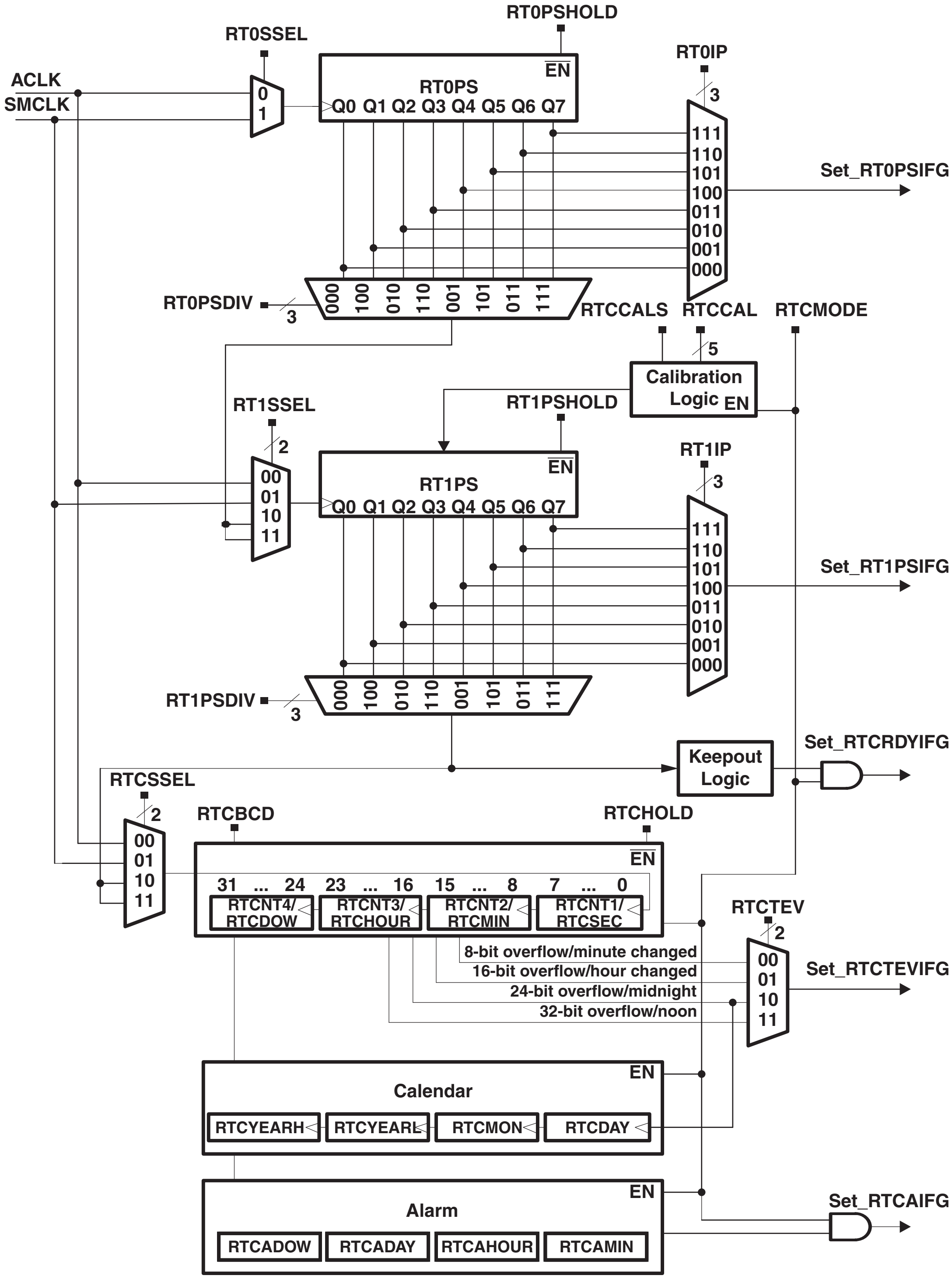


Figure 14-1. Real-Time Clock

14.2 Real-Time Clock Operation

The Real-Time Clock module can be configured as a real-time clock with calendar function or as a 32-bit general purpose counter with the RTCMODE bit

14.2.1 Counter Mode

Counter mode is selected when RTCMODE is reset. In this mode, a 32-bit counter is provided that is directly accessible by software. Switching from calendar mode to counter mode resets the count value (RTCNT1, RTCNT2, RTCNT3, RTCNT4), as well as, the prescale counters (RT0PS, RT1PS).

The clock to increment the counter can be sourced from ACLK, SMCLK, or prescaled versions of ACLK or SMCLK. Prescaled versions of ACLK or SMCLK are sourced from the prescale dividers , RT0PS and RT1PS. RT0PS and RT1PS output /2, /4, /8, 16, /32, /64, /128, /256 versions of ACLK and SMCLK, respectively. The output of RT0PS can be cascaded with RT1PS. The cascaded output can be used as a clock source input to the 32-bit counter.

Four individual 8-bit counters are cascaded to provide the 32-bit counter. This provides 8-bit, 16-bit, 24-bit, or 32-bit overflow intervals of the counter clock. The RTCTEV bits select the respective trigger event. An RTCTEV event can trigger an interrupt by setting the RTCTEVIE bit. Each counter RTCNT1 through RTCNT4 is individually accessible and may be written to.

RT0PS and RT1PS can be configured as two 8-bit counters or cascaded into a single 16-bit counter. RT0PS and RT1PS can be halted on an individual basis by setting their respective RT0PSHOLD and RT1PSHOLD bits. When RT0PS is cascaded with RT1PS, setting RT0PSHOLD will cause both RT0PS and RT1PS to be halted. The 32-bit counter can be halted several ways depending on the configuration. If the 32-bit counter is sourced directly from ACLK or SMCLK, it can be halted by setting RTCHOLD. If it is sourced from the output of RT1PS, it can be halted by setting RT1PSHOLD or RTCHOLD. Finally, if it is sourced from the cascaded outputs of RT0PS and RT1PS, it can be halted by setting RT0PSHOLD, RT1PSHOLD, or RTCHOLD.

Note: Accessing the RTCNTx registers

When the counter clock is asynchronous to theCPUclock, any read from any RTCNTx, RT0PS, or RT1PS registers should occur while the counter is not operating. Otherwise, the results may be unpredictable. Alternatively, the counter may be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Anywrite to anyRTCNTx, RT0PS, or RT1PS registers takes effect immediately.

14.2.2 Calendar Mode

Calendar mode is selected when RTCMODE is set. In calendar mode, the Real-Time Clock module provides seconds, minutes, hours, day of week, day of month, month, and year in selectable BCD or hexadecimal format. The calendar includes a leap year algorithm that considers all years evenly divisible by 4 as leap years. This algorithm is accurate from the year 1901 through 2099.

14.2.2.1 Real-Time Clock and Prescale Dividers

The prescale dividers, RT0PS and RT1PS are automatically configured to provide a one second clock interval for the Real-Time Clock. RT0PS is sourced from ACLK. ACLK must be set to 32768 Hz, nominal for proper Real-Time Clock calendar operation. RT1PS is cascaded with the output ACLK/256 of RT0PS. The Real-Time Clock is sourced with the /128 output of RT1PS, thereby providing the required one second interval. Switching from counter to calendar mode clears the seconds, minutes, hours, day-of-week, and year counts and sets day-of-month and month counts to 1. In addition, the RT0PS and RT1PS are cleared.

When RTCBCD = 1, BCD format is selected for the calendar registers. The format must be selected before the time is set. Changing the state of RTCBCD clears the seconds, minutes, hours, day-of-week, and year counts and sets day-of-month and month counts to 1. In addition, RT0PS and RT1PS are cleared.

In calendar mode, the RT0SSEL, RT1SSEL, RT0PSDIV, RT1PSDIV, RT0PSHOLD, RT1PSHOLD, and RTCSSEL bits are do not care. Setting RTCHOLD halts the real-time counters and prescale counters, RT0PS and RT1PS.

14.2.2.2 Real-Time Clock Alarm Function

The Real-Time Clock module provides for a flexible alarm system. There is a single, user programmable alarm that can be programmed based on the settings contained in the alarm registers for minutes, hours, day of week, and day of month. The user programmable alarm function is only available in calendar mode of operation.

Each alarm register contains an alarm enable bit, AE that can be used to enable the respective alarm register. By setting AE bits of the various alarm registers, a variety of alarm events can be generated.

For example, a user wishes to set an alarm every hour at 15 minutes past the hour i.e. 00:15:00, 01:15:00, 02:15:00, etc. This is possible by setting RTCAMIN to 15. By setting the AE bit of the RTCAMIN, and clearing all other AE bits of the alarm registers, the alarm will be enabled. When enabled, the AF will be set when the count transitions from 00:14:59 to 00:15:00, 01:14:59 to 01:15:00, 02:14:59 to 02:15:00, etc.

For example, a user wishes to set an alarm every day at 04:00:00. This is possible by setting RTCAHOUR to 4. By setting the AE bit of the RTCHOUR, and clearing all other AE bits of the alarm registers, the alarm will be enabled. When enabled, the AF will be set when the count transitions from 03:59:59 to 04:00:00.

For example, a user wishes to set an alarm for 06:30:00. RTCAHOUR would be set to 6 and RTCAMIN would be set to 30. By setting the AE bits of RTCAHOUR and RTCAMIN, the alarm will be enabled. Once enabled, the AF will be set when the the time count transitions from 06:29:59 to 06:30:00. In this case, the alarm event will occur every day at 06:30:00.

For example, a user wishes to set an alarm every Tuesday at 06:30:00. RTCADOW would be set to 2, RTCAHOUR would be set to 6 and RTCAMIN would be set to 30. By setting the AE bits of RTCADOW, RTCAHOUR and RTCAMIN, the alarm will be enabled. Once enabled, the AF will be set when the the time count transitions from 06:29:59 to 06:30:00 and the RTCDOW transitions from 1 to 2.

For example, a user wishes to set an alarm the fifth day of each month at 06:30:00. RTCADAY would be set to 5, RTCAHOUR would be set to 6 and RTCAMIN would be set to 30. By setting the AE bits of RTCADAY, RTCAHOUR and RTCAMIN, the alarm will be enabled. Once enabled, the AF will be set when the the time count transitions from 06:29:59 to 06:30:00 and the RTCDAY equals 5.

Note: Invalid Alarm Settings

Invalid alarm settings are not checked via hardware. It is the user responsibility that valid alarm settings are entered.

Note: Invalid Time and Date Values

Writing of invalid date and/or time information or data values outside the legal ranges specified in the RTCSEC, RTCMIN, RTCHOUR, RTCDAY, RTCDOW, RTCYEARH, RTCYEARL, RTCAMIN, RTCAHOUR, RTCADAY, and RTCADOW registers can result in unpredictable behavior.

Note: Setting the Alarm

In order to prevent potential erroneous alarm conditions from occurring, the alarms should be disabled be clearing the RTCAIE, RTCAIFG, and AE bits prior to writing new time values to the RTC time registers.

14.2.2.3 Reading or Writing Real-Time Clock Registers in Calendar Mode

Since the system clock may in fact be asynchronous to the Real-Time Clock clock source, special care must be used when accessing the Real-Time Clock registers.

In calendar mode, the real-time clock registers are updated once per second. In order to prevent reading any real-time clock register at the time of an update that could result in an invalid time being read, a keepout window is provided. The keepout window is centered approximately - 128/32768 seconds around the update transition. The read only RTCRDY bit is reset during the keepout window period and set outside the keepout the window period. Any read of the clock registers while RTCRDY is reset, is considered to be potentially invalid, and the time read should be ignored.

An easy way to safely read the real-time clock registers is to utilize the RTCRDYIFG interrupt flag. Setting RTCRDYIE enables the RTCRDYIFG interrupt. Once enabled, an interrupt will be generated based on the rising edge of the RTCRDY bit, causing the RTCRDYIFG to be set. At this point, the application has nearly a complete second to safely read any or all of the real-time clock registers. This synchronization process prevents reading the time value during transition. The RTCRDYIFG flag is reset automatically when the interrupt is serviced, or can be reset with software.

In counter mode, the RTCRDY bit remains reset. The RTCRDYIE is a do not care and the RTCRDYIFG remains reset.

Note: Reading or Writing Real-Time Clock Registers

When the counter clock is asynchronous to theCPUclock, any read from any RTCSEC, RTCMIN, RTCHOUR, RTCDOW, RTCDAY, RTCMON, RTCYEARL, RTCYEARH registers while the RTCRDY is resetmay result in invalid data being read. To safely read the counting registers, either polling of the RTCRDY bit or the synchronization procedure described above can be used. Alternatively, the counter register can be read multiple times while operating, and a majority vote taken in software to determine the correct reading. Reading theRT0PS andRT1PS can only be handled by reading the registers multiple times and a majority vote taken in software to determine the correct reading or by halting the counters.

Any write to any counting register takes effect immediately. However, the clock is stopped during the write. In addition, RT0PS and RT1PS registers are reset. This could result in losing up to one second during a write.Writing of data outside the legal ranges or invalid time stamp combinations results in unpredictable behavior.

14.2.3 Real-Time Clock Interrupts

The Real-Time Clock module has five interrupt sources available, each with independent enables and flags.

14.2.3.1 Real-Time Clock Interrupts in Calendar Mode

In calendar mode, five sources for interrupts are available, namely RT0PSIFG, RT1PSIFG, RTCRDYIFG, RTCTEVIFG, and RTCAIFG. These flags are prioritized and combined to source a single interrupt vector. The interrupt vector register RTCIV is used to determine which flag requested an interrupt.

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

Any access, read or write, of the RTCIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. In addition, all flags can be cleared via software.

The user programmable alarm event sources the real-time clock interrupt, RTCAIFG. Setting the RTCAIE enables the interrupt. In addition to the user programmable alarm, The Real-Time Clock Module provides for an interval alarm that sources real-time clock interrupt, RTCTEVIFG. The interval alarm can be selected to cause an alarm event when RTCMIN changed, RTCHOUR changed, every day at midnight (00:00:00), or every day at noon (12:00:00). The event is selectable with the RTCTEV bits Setting the RTCTEVIE bit enables the interrupt.

The RTCRDY bit sources the real-time clock interrupt, RTCRDYIFG and is useful in synchronizing the read of time registers with the system clock. Setting the RTCRDYIE bit enables the interrupt.

The RT0PSIFG can be used to generate interrupt intervals selectable by the RT0IP bits. In calendar mode, RT0PS is sourced with ACLK at 32768 Hz, so intervals of 16384 Hz, 8192 Hz, 4096 Hz, 2048 Hz, 1024 Hz, 512 Hz, 256 Hz, or 128 Hz are possible. Setting the RT0PSIE bit enables the interrupt.

The RT1PSIFG can be used to generate interrupt intervals selectable by the RT1IP bits. In calendar mode, RT1PS is sourced with the output of RT0PS, which is 128Hz (32768/256 Hz). Therefore, intervals of 64 Hz, 32 Hz, 16 Hz, 8 Hz, 4 Hz, 2 Hz, 1 Hz, or 0.5 Hz are possible. Setting the RT1PSIE bit enables the interrupt.

14.2.3.2 Real-Time Clock Interrupts in Counter Mode

In counter mode, a three interrupt sources are available, namely RT0PSIFG, RT1PSIFG, and RTCTEVIFG. The RTCAIFG and RTCRDYIFG are cleared. RTCRDYIE and RTCAIE are do not care.

The RT0PSIFG can be used to generate interrupt intervals selectable by the RT0IP bits. In counter mode, RT0PS is sourced with ACLK or SMCLK so divide ratios of /2, /4, /8, /16, /32, /64, /128, /256 of the respective clock source are possible. Setting the RT0PSIE bit enables the interrupt.

The RT1PSIFG can be used to generate interrupt intervals selectable by the RT1IP bits. In counter mode, RT1PS is sourced with ACLK, SMCLK, or the output of RT0PS so divide ratios of /2, /4, /8, /16, /32, /64, /128, /256 of the respective clock source are possible. Setting the RT1PSIE bit enables the interrupt.

The Real-Time Clock Module provides for an interval timer that sources real-time clock interrupt, RTCTEVIFG. The interval timer can be selected to cause an interrupt event when an 8-bit, 16-bit, 24-bit, or 32-bit overflow occurs within the 32-bit counter. The event is selectable with the RTCTEV bits Setting the RTCTEVIE bit enables the interrupt.

RTCIV Software Example

The following software example shows the recommended use of RTCIV and the handling overhead. The RTCIV 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.

```
; Interrupt handler for RTC interrupt flags.

RTC_HND                        ; Interrupt latency          6
         ADD   &RTCIV,PC       ; Add offset to Jump table    3
         RETI                  ; Vector 0: No interrupt      5
         JMP   RTCRDYIFG_HND   ; Vector 2: RTCRDYIFG         2
         JMP   RTCTEVIFG_HND   ; Vector 4: RTCTEVIFG         2
         JMP   RTCAIFG         ; Vector 6: RTCAIFG           5
         JMP   RT0PSIFG        ; Vector 8: RT0PSIFG          5
         JMP   RT1PSIFG        ; Vector A: RT1PSIFG          5
         RETI                  ; Vector C: Reserved          5

RTCRDYIFG_HND                  ; Vector 2: RTCRDYIFG Flag
          to                    ; Task starts here
         RETI                                                5

RTCTEVIFG_HND                  ; Vector 4: RTCTEVIFG
          to                    ; Task starts here
         RETI                  ; Back to main program        5

RTCAIFG_HND                    ; Vector 6: RTCAIFG
          to                    ; Task starts here

RT0PSIFG_HND                   ; Vector 8: RT0PSIFG
```

```
          to                   ; Task starts here

RT1PSIFG_HND                  ; Vector A: RT1PSIFG
          to                   ; Task starts here
```

14.2.4 Real-Time Clock Calibration

The Real-Time Clock module has calibration logic that allows for adjusting the crystal frequency in +4 ppm or –2 ppm steps allowing for higher time keeping accuracy from standard crystals.

The RTCCALx bits are used to adjust the frequency. When RTCCALS is set, each RTCCALx LSB will cause a +4 ppm adjustment. When RTCCALS is cleared, each RTCCALx LSB will cause a –2 ppm adjustment.

To calibrate the frequency, the RTCCLK output signal is available at a pin. The RTCCALF bits can be used to select the frequency rate of the output signal. During calibration, the RTCCLK can be measured. The result of this measurement can be applied to the RTCCALS and RTCCALx bits to effectively reduce the initial offset of the clock. For example, say the RTCCLK is output at a frequency of 512 Hz. The measured RTCCLK is 511.9658 Hz. This frequency error is approximately 67 ppm too low. In order to increase the frequency by 67 ppm, RTCCALS would be set, and RTCCALx would be set to 17 (67/4).

In counter mode (RTCMODE = 0), the calibration logic is disabled.

Note: Calibration Output Frequency

The 512-Hz and 256-Hz output frequencies observed at the RTCCLK pin are not effected by changes in the calibration settings. The 1-Hz output frequency is affected by changes in the calibration settings.

14.3 Real-Time Clock Registers

The Real-Time Clock module registers are listed in and Table 14-1. Some of the registers can be accessed word-wise as shown in Table 14-2. The base register for the Real-Time Clock module registers can be found in the device specific data sheet. The address offsets are given in Table 14-1 and Table 14-2.

Table 14-1. Real-Time Clock Registers

Register	Short Form	Register Type	Address Offset	Initial State
Real-Time Clock control register 0	RTCCTL0	Read/write	00h	00h
Real-Time Clock control register 1	RTCCTL1	Read/write	01h	40h
Real-Time Clock control register 2	RTCCTL2	Read/write	02h	00h
Real-Time Clock control register 3	RTCCTL3	Read/write	03h	00h
Real-Time Prescale Timer 0 control register	RTCPS0CTL	Read/write	08h	10h
Real-Time Prescale Timer 1 control register	RTCPS1CTL	Read/write	0Ah	10h
Real-Time Prescale Timer 0	RTCPS0	Read/write	0Ch	Unchanged
Real-Time Prescale Timer 1	RTCPS1	Read/write	0Dh	Unchanged
Real Time Clock Interrupt vector	RTCIV	Read	0Eh	00h
Real-Time Clock Second Real-Time Counter register 1	RTCSEC/RTCNT1	Read/write	10h	Unchanged
Real-Time Clock Minute Real-Time Counter register 2	RTCMIN/RTCNT2	Read/write	11h	Unchanged
Real-Time Clock Hour Real-Time Counter register 3	RTCHOUR/RTCNT3	Read/write	12h	Unchanged
Real-Time Clock Day of Week Real-Time Counter register 4	RTCDOW/RTCNT4	Read/write	13h	Unchanged
Real-Time Clock Day of Month	RTCDAY	Read/write	14h	Unchanged
Real-Time Clock Month	RTCMON	Read/write	15h	Unchanged
Real-Time Clock Year (Low Byte)	RTCYEARL	Read/write	16h	Unchanged
Real-Time Clock Year (High Byte)	RTCYEARH	Read/write	17h	Unchanged
Real-Time Clock Minute Alarm	RTCAMIN	Read/write	18h	Unchanged
Real-Time Clock Hour Alarm	RTCAHOUR	Read/write	19h	Unchanged
Real-Time Clock Day of Week Alarm	RTCADOW	Read/write	1Ah	Unchanged
Real-Time Clock Day of Month Alarm	RTCADAY	Read/write	1Bh	Unchanged

Table 14-2. Word Access to Registers in Counter Mode

Word Register	Short Form	High-Byte Register	Low-Byte Register	Address Offset
Real-Time control registers 0, 1	RTCCTL01	RTCCTL1	RTCCTL0	00h
Real-Time control registers 2, 3	RTCCTL23	RTCCTL3	RTCCTL2	02h
Real-Time Prescale Timer 0 control	RTCPS0CTL	RTCPS0CTLH	RTCPS0CTLL	08h
Real-Time Prescale Timer 1 control	RTCPS1CTL	RTCPS1CTLH	RTCPS1CTLL	0Ah
Real-Time Prescale Timer	RTCPS	RTCPS1	RTCPS0	0Ch
Real Time Clock Interrupt vector	RTCIV			0Eh
Real-Time Clock Time 0 Real-Time Counter registers 1, 2	RTCTIM0/RTCNT12	RTCMIN/ RTCNT2	RTCSEC/ RTCNT1	10h
Real-Timer Clock Time 1 Real-Time Counter registers 3, 4	RTCTIM1/RTCNT34	RTCDOW/ RTCNT4	RTCHOUR/ RTCNT3	12h
Real-Timer Clock Date	RTCDATE	RTCMON	RTCDAY	14h
Real-Timer Clock Year	RTCYEAR	RTCYEARH	RTCYEARL	16h
Real-Timer Clock Alarm min/hour	RTCAMINHR	RTCAHOUR	RTCAMIN	18h
Real-Timer Clock Alarm day of week/day	RTCADOWDAY	RTCADAY	RTCADOW	1Ah

RTCCTL0, Real-Time Clock Control Register 0

7	6	5	4	3	2	1	0
Reserved	**RTCTEVIE**	**RTCAIE**	**RTCRDYIE**	**Reserved**	**RTCTEVIFG**	**RTCAIFG**	**RTCRDYIFG**
r0	rw-0	rw-0	rw-0	r0	rw-(0)	rw-(0)	rw-(0)

Reserved Bit 7 Reserved. Always read as 0.

RTCTEVIE Bit 6 Real-time clock time event interrupt enable

0 Interrupt not enabled

1 Interrupt enabled

RTCAIE Bit 5 Real-time clock alarm interrupt enable. This bit remains cleared when in counter mode (RTCMODE = 0).

0 Interrupt not enabled

1 Interrupt enabled

RTCRDYIE Bit 4 Real-time clock alarm interrupt enable

0 Interrupt not enabled

1 Interrupt enabled

Reserved Bit 3 Reserved. Always read as 0.

RTCTEVIFG Bit 2 Real-time clock time event flag

0 No time event occurred.

1 Time event occurred.

RTCAIFG Bit 1 Real-time clock alarm flag. This bit remains cleared when in counter mode (RTCMODE = 0).

0 No time event occurred.

1 Time event occurred.

RTCRDYIFG Bit 0 Real-time clock alarm flag

0 RTC can not be read safely

1 RTC can be read safely

RTCCTL1, Real-Time Clock Control Register 1

7	6	5	4	3	2	1	0
RTCBCD	**RTCHOLD**	**RTCMODE**	**RTCRDY**	**RTCSSEL**		**RTCTEV**	
rw-(0)	rw-(1)	rw-(0)	r-(0)	rw-0	rw-0	rw-(0)	rw-(0)

RTCBCD Bit 7 Real-time clock BCD select. Selects BCD counting for real-time clock. Applies to calendar mode (RTCMODE = 1) only - setting will be ignored in counter mode. Changing this bit will clear seconds, minutes, hours, day of week, and year are to 0 and sets day of month and month to 1. The real-time clock registers need to be set by software afterwards.

0 Binary/hexadecimal code selected

1 BCD (Binary Coded Decimal) code selected

RTCHOLD Bit 6 Real-time clock hold

0 Real-Time Clock (32-bit counter or calendar mode) is operational

1 In counter mode (RTCMODE = 0) only the 32-bit counter is stopped. In calendar mode (RTCMODE = 1) the calendar is stopped as well as the Prescale counters, RT0PS and RT1PS. RT0PSHOLD and RT1PSHOLD are do not care.

RTCMODE Bit 5 Real-time clock mode

0 32-bit counter mode

1 Calendar modeSwitching between counter and calendar mode will reset the real-time clock/counter registers. Switching to calendar mode clears seconds, minutes, hours, day of week, and year are to 0 and sets day of month and month to 1. The real-time clock registers need to be set by software afterwards. The Basic Timer counters, BT0CNT and BT1CNT, are also cleared.

RTCRDY Bit 4 Real-time clock ready

0 RTC time values in transition (calendar mode only).

1 RTC time values safe for reading (calendar mode only)This bit indicates when the RTC time values are safe for reading (calendar mode only). In counter mode, RTCRDY signal remains cleared.

RTCSSEL Bits 3-2 Real-time clock source select. Selects clock input source to the RTC/32-bit counter. In Real-Time Clock calendar mode, these bits are do not care. The clock input is automatically set to the output of RT1PS.

00 ACLK

01 SMCLK

10 Output from RT1PS

11 Output from RT1PS

RTCTEV Bits 1-0 Real-time clock time event

RTC Mode	RTCTEVx	Interrupt Interval
Counter Mode (RTCMODE = 0)	00	8-bit overflow
	01	16-bit overflow
	10	24-bit overflow
	11	32-bit overflow
Calendar Mode (RTCMODE = 1)	00	Minute changed
	01	Hour changed
	10	Every day at midnight (00:00)
	11	Every day at noon (12:00)

RTCCTL2, Real-Time Clock Control Register 2

7	6	5	4	3	2	1	0
RTCCALS	**Reserved**	**RTCCAL**					
rw-(0)	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

RTCCALS Bit 7 Real-time clock calibration sign

0 Frequency adjusted down

1 Frequency adjusted up

Reserved Bit 6 Reserved. Always read as 0.

RTCCAL Bits 5-0 Real-time clock calibration bits
Each LSB represents approximately +4 ppm (RTCCALS = 1) or a -2 ppm (RTCCALS = 0) adjustment in frequency.

RTCCTL3, Real-Time Clock Control Register 3

7	6	5	4	3	2	1	0
Reserved						**RTCCALF**	
r0	r0	r0	r0	r0	r0	rw-0	rw-0

Reserved Bits 7-2 Reserved. Always read as 0.

RTCCALF Bits 1-0 Real-time clock calibration frequency
Selects frequency output to RTCCLK pin for calibration measurement. The corresponding port must be configured for the peripheral module function. The RTCCLK is not available in counter mode and remains low and the RTCCALF bits are do not care.

00	No frequency output to RTCCLK pin
01	512 Hz
10	256 Hz
11	1 Hz

RTCNT1, RTC Counter Register 1, Counter Mode

7	6	5	4	3	2	1	0
RTCNT1x							
rw	rw	rw	rw	rw	rw	rw	rw

RTCNT1x Bits 7-0 The RTCNT1 register is the count of RTCNT1

RTCNT2, RTC Counter Register 2, Counter Mode

7	6	5	4	3	2	1	0
RTCNT2x							
rw	rw	rw	rw	rw	rw	rw	rw

RTCNT2x Bits 7-0 The RTCNT2 register is the count of RTCNT2

RTCNT3, RTC Counter Register 3, Counter Mode

7	6	5	4	3	2	1	0
RTCNT3x							
rw	rw	rw	rw	rw	rw	rw	rw

RTCNT3x Bits 7-0 The RTCNT3 register is the count of RTCNT3

RTCNT4, RTC Counter Register 4, Counter Mode

7	6	5	4	3	2	1	0
RTCNT4x							
rw	rw	rw	rw	rw	rw	rw	rw

RTCNT4x	Bits 7-0	The RTCNT4 register is the count of RTCNT4

RTCSEC, RTC Seconds Register, Calendar Mode with Hexadecimal Format

7	6	5	4	3	2	1	0
0	**0**	**Seconds (0 to 59)**					
r-0	r-0	rw	rw	rw	rw	rw	rw

RTCSEC, RTC Seconds Register, Calendar Mode with BCD Format

7	6	5	4	3	2	1	0
0	**Seconds – high digit (0 to 5)**			**Seconds – low digit (0 to 9)**			
r-0	rw	rw	rw	rw	rw	rw	rw

RTCMIN, RTC Minutes Register, Calendar Mode with Hexadecimal Format

7	6	5	4	3	2	1	0
0	**0**	**Minutes (0 to 59)**					
r-0	r-0	rw	rw	rw	rw	rw	rw

RTCMIN, RTC Minutes Register, Calendar Mode with BCD Format

7	6	5	4	3	2	1	0
0	**Minutes – high digit (0 to 5)**			**Minutes – low digit (0 to 9)**			
r-0	rw	rw	rw	rw	rw	rw	rw

RTCHOUR, RTC Hours Register, Calendar Mode with Hexadecimal Format

7	6	5	4	3	2	1	0
0	**0**	**0**	**Hours (0 to 24)**				
r-0	r-0	r-0	rw	rw	rw	rw	rw

RTCHOUR, RTC Hours Register, Calendar Mode with BCD Format

7	6	5	4	3	2	1	0
0	**0**	**Hours – high digit (0 to 2)**		**Hours – low digit (0 to 9)**			
r-0	r-0	rw	rw	rw	rw	rw	rw

RTCDOW, RTC Day of Week Register, Calendar Mode

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**Day of week (0 to 6)**		
r-0	r-0	r-0	r-0	r-0	rw	rw	rw

RTCDAY, RTC Day of Month Register, Calendar Mode with Hexadecimal Format

7	6	5	4	3	2	1	0
0	**0**	**0**	**Day of month (1 to 28, 29, 30, 31)**				
r-0	r-0	r-0	rw	rw	rw	rw	rw

RTCDAY, RTC Day of Month Register, Calendar Mode with BCD Format

7	6	5	4	3	2	1	0
0	**0**	**Day of month – high digit (0 to 3)**		**Day of month – low digit (0 to 9)**			
r-0	r-0	rw	rw	rw	rw	rw	rw

RTCMON, RTC Month Register, Calendar Mode with Hexadecimal Format

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**Month (1 to 12)**			
r-0	r-0	r-0	r-0	rw	rw	rw	rw

RTCMON, RTC Month Register, Calendar Mode with BCD Format

7	6	5	4	3	2	1	0
0	**0**	**0**	**Month – high digit (0 to 3)**	**Month – low digit (0 to 9)**			
r-0	r-0	r-0	rw	rw	rw	rw	rw

RTCYEARL, RTC Year Low-Byte Register, Calendar Mode with Hexadecimal Format

7	6	5	4	3	2	1	0
Year – low byte of 0 to 4095							
rw	rw	rw	rw	rw	rw	rw	rw

RTCYEARL, RTC Year Low-Byte Register, Calendar Mode with BCD Format

7	6	5	4	3	2	1	0
Decade (0 to 9)				**Year – lowest digit (0 to 9)**			
rw	rw	rw	rw	rw	rw	rw	rw

RTCYEARH, RTC Year High-Byte Register, Calendar Mode with Hexadecimal Format

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**Year – high byte of 0 to 4095**			
r-0	r-0	r-0	r-0	rw	rw	rw	rw

RTCYEARH, RTC Year High-Byte Register, Calendar Mode with BCD Format

7	6	5	4	3	2	1	0
0	**Century – high digit (0 to 4)**			**Century – low digit (0 to 9)**			
r-0	rw	rw	rw	rw	rw	rw	rw

RTCAMIN, RTC Minutes Alarm Register, Calendar Mode with Hexadecimal Format

7	6	5	4	3	2	1	0
AE	**0**	**Minutes (0 to 59)**					
rw-0	r-0	rw	rw	rw	rw	rw	rw

RTCAMIN, RTC Minutes Alarm Register, Calendar Mode with BCD Format

7	6	5	4	3	2	1	0
AE	**Minutes – high digit (0 to 5)**			**Minutes – low digit (0 to 9)**			
rw-0	rw	rw	rw	rw	rw	rw	rw

RTCAHOUR, RTC Hours Alarm Register, Calendar Mode with Hexadecimal Format

7	6	5	4	3	2	1	0
AE	**0**	**0**	**Hours (0 to 24)**				
rw-0	r-0	r-0	rw	rw	rw	rw	rw

RTCAHOUR, RTC Hours Alarm Register, Calendar Mode with BCD Format

7	6	5	4	3	2	1	0
AE	**0**	**Hours – high digit (0 to 2)**		**Hours – low digit (0 to 9)**			
rw-0	r-0	rw	rw	rw	rw	rw	rw

RTCADOW, RTC Day of Week Alarm Register, Calendar Mode

7	6	5	4	3	2	1	0
AE	**0**	**0**	**0**	**0**	**Day of week (0 to 6)**		
rw-0	r-0	r-0	r-0	r-0	rw	rw	rw

RTCADAY, RTC Day of Month Alarm Register, Calendar Mode with Hexadecimal Format

7	6	5	4	3	2	1	0
AE	**0**	**0**	**Day of month (1 to 28, 29, 30, 31)**				
rw-0	r-0	r-0	rw	rw	rw	rw	rw

RTCADAY, RTC Day of Month Alarm Register, Calendar Mode with BCD Format

7	6	5	4	3	2	1	0
AE	**0**	**Day of month – high digit (0 to 3)**		**Day of month – low digit (0 to 9)**			
rw-0	r-0	rw	rw	rw	rw	rw	rw

RTCPS0CTL, Prescale Timer 0 Control Register

15	14	13	12	11	10	9	8
Reserved	**RT0SSEL**	**RT0PSDIV**			**Reserved**	**Reserved**	**RT0PSHOLD**
r0	rw-0	rw-0	rw-0	rw-0	r0	r0	rw-1

7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**RT0IP**			**RT0PSIE**	**RT0PSIFG**
r0	r0	r0	rw-0	rw-0	rw-0	rw-0	rw-(0)

Reserved Bits 15 Reserved. Always read as 0.

RT0SSEL Bits 14 Prescale Timer 0 clock source select. Selects clock input source to the RT0PS counter. In Real-Time Clock calendar mode, these bits are do not care. RT0PS clock input is automatically set to ACLK. RT1PS clock input is automatically set to the output of RT0PS.

0 ACLK

1 SMCLK

RT0PSDIV Bits 13-11 Prescale Timer 0 clock divide. These bits control the divide ratio of the RT0PS counter. In Real-Time Clock calendar mode, these bits are do not care for RT0PS and RT1PS. RT0PS clock output is automatically set to /256. RT1PS clock output is automatically set to /128.

000	/2
001	/4
010	/8
011	/16
100	/32
101	/64
110	/128
111	/256

Reserved Bits 10-9 Reserved. Always read as 0.

RT0PSHOLD Bit 8 Prescale Timer 0 Hold. In Real-Time Clock calendar mode, this bit is do not care. RT0PS is stopped via the RTCHOLD bit.

0 RT0PS is operational

1 RT0PS is held

Reserved Bits 7-5 Reserved. Always read as 0.

RT0IP Bits 4-2 Prescale Timer 0 interrupt interval

000	/2
001	/4
010	/8
011	/16
100	/32
101	/64
110	/128
111	/256

RT0IE Bit 1 Prescale Timer 0 interrupt enable

0 Interrupt not enabled

1 Interrupt enabled

RT0IFG Bit 0 Prescale Timer 0 interrupt flag

0 No time event occurred

1 Time event occurred

RTCPS1CTL, Prescale Timer 1 Control Register

15	14	13	12	11	10	9	8
RT1SSEL		**RT1PSDIV**			**Reserved**	**Reserved**	**RT1PSHOLD**
rw-0	rw-0	rw-0	rw-0	rw-0	r0	r0	rw-1

7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**RT1IP**			**RT1PSIE**	**RT1PSIFG**
r0	r0	r0	rw-0	rw-0	rw-0	rw-0	rw-(0)

RT1SSEL Bits 15-14 Prescale Timer 1 clock source select. Selects clock input source to the RT1PS counter. In Real-Time Clock calendar mode, these bits are do not care. RT1PS clock input is automatically set to the output of RT0PS.

00	ACLK
01	SMCLK
10	Output from RT0PS
11	Output from RT0PS

RT1PSDIV Bits 13-11 Prescale Timer 1 clock divide. These bits control the divide ratio of the RT0PS counter. In Real-Time Clock calendar mode, these bits are do not care for RT0PS and RT1PS. RT0PS clock output is automatically set to /256. RT1PS clock output is automatically set to /128.

000	/2
001	/4
010	/8
011	/16
100	/32
101	/64
110	/128
111	/256

Reserved Bits 10-9 Reserved. Always read as 0.

RT1PSHOLD Bit 8 Prescale Timer 1 hold. In Real-Time Clock calendar mode, this bit is do not care. RT1PS is stopped via the RTCHOLD bit.

0	RT1PS is operational
1	RT1PS is held

Reserved Bits 7-5 Reserved. Always read as 0.

RT1IP Bits 4-2 Prescale Timer 1 interrupt interval

000	/2
001	/4
010	/8
011	/16
100	/32
101	/64
110	/128
111	/256

RT1PSIE Bit 1 Prescale Timer 1 interrupt enable

0	Interrupt not enabled
1	Interrupt enabled

RT1PSIFG Bit 0 Prescale Timer 1 interrupt flag

0	No time event occurred
1	Time event occurred

RTCPS0, Prescale Timer 0 Counter Register

7	6	5	4	3	2	1	0
RT0PS							
rw	rw	rw	rw	rw	rw	rw	rw

RT0PS Bits 7-0 Prescale Timer 0 counter value

RTCPS1, Prescale Timer 1 Counter Register

7	6	5	4	3	2	1	0
RT1PS							
rw	rw	rw	rw	rw	rw	rw	rw

RT1PS Bits 7-0 Prescale Timer 1 counter value

RTCIV, RTC 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**		**RTCIVx**				0
r0	r0	r0	r-(0)	r-(0)	r-(0)	r-(0)	r0

RTCIVx Bits 15-0 RTC interrupt vector value

RTCIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	RTC ready	RTCRDYIFG	Highest
04h	RTC interval timer	RTCTEVIFG	
06h	RTC user alarm	RTCAIFG	
08h	RTC prescaler 0	RT0PSIFG	
0Ah	RTC prescaler 1	RT1PSIFG	
0Ch	Reserved		
0Eh	Reserved		
10h	Reserved		Lowest

Chapter 15

SLAU208–June 2008

Universal Serial Communication Interface, UART Mode

The 5xx 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**

15.1 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^{2}C mode
- SPI mode

15.2 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 15-1 shows the USCI_Ax when configured for UART mode.

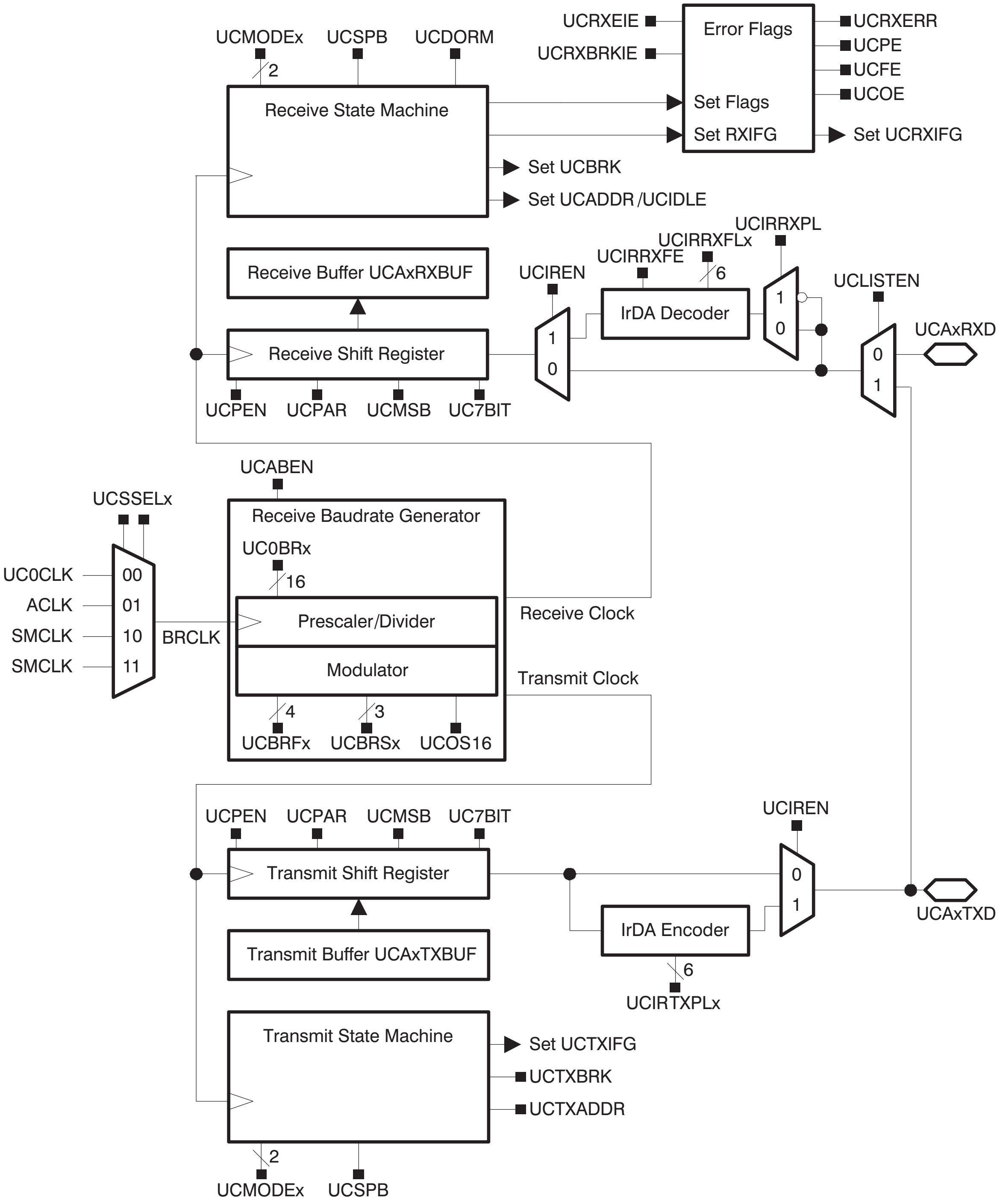


Figure 15-1. USCI_Ax Block Diagram: UART Mode (UCSYNC = 0)

15.3 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.

15.3.1 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 UCRXIE, UCTXIE, UCRXIFG, UCRXERR, UCBRK, UCPE, UCOE, UCFE, UCSTOE and UCBTOE bits and sets the UCTXIFG 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 UCRXIE and/or UCTXIE

15.3.2 Character Format

The UART character format, shown in Figure 15-2, 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.

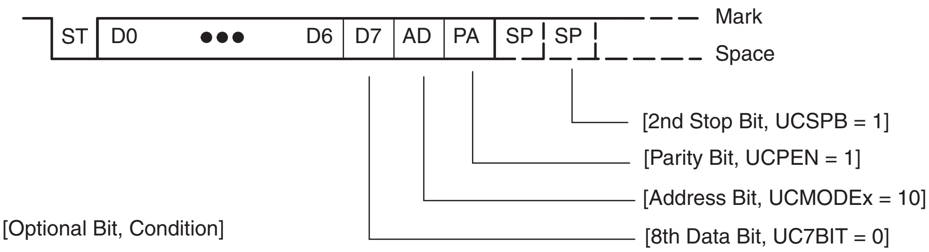

ST
D0
●●●
D6
D7
AD
PA
SP
SP
Mark
Space
[2nd Stop Bit, UCSPB = 1]
[Parity Bit, UCPEN = 1]
[Address Bit, UCMODEx = 10]
[8th Data Bit, UC7BIT = 0]
[Optional Bit, Condition]

Figure 15-2. Character Format

15.3.3 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 15-3. 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

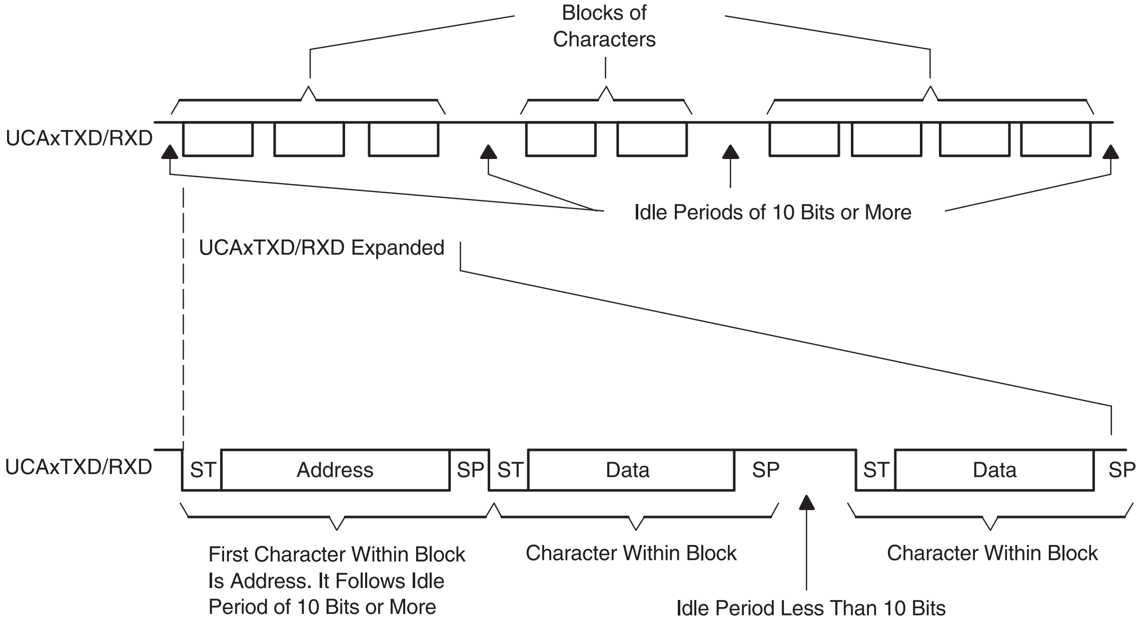

Blocks of Characters
UCAxTXD/RXD
Idle Periods of 10 Bits or More
UCAxTXD/RXD Expanded
UCAxTXD/RXD
ST
Address
SP
ST
Data
SP
ST
Data
SP
First Character Within Block Is Address. It Follows Idle Period of 10 Bits or More
Character Within Block
Character Within Block
Idle Period Less Than 10 Bits

Figure 15-3. 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, UCRXIFG is set, and any applicable error flag is set when UCRXEIE = 1. When UCRXEIE = 0 and an address character is received but has a framing error or parity error, the character is not transferred into UCAxRXBUF and UCRXIFG 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 (UCTXIFG = 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 (UCTXIFG = 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 15-4. 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, UCRXIFG 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 UCRXIFG 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 UCRXIFG. 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.

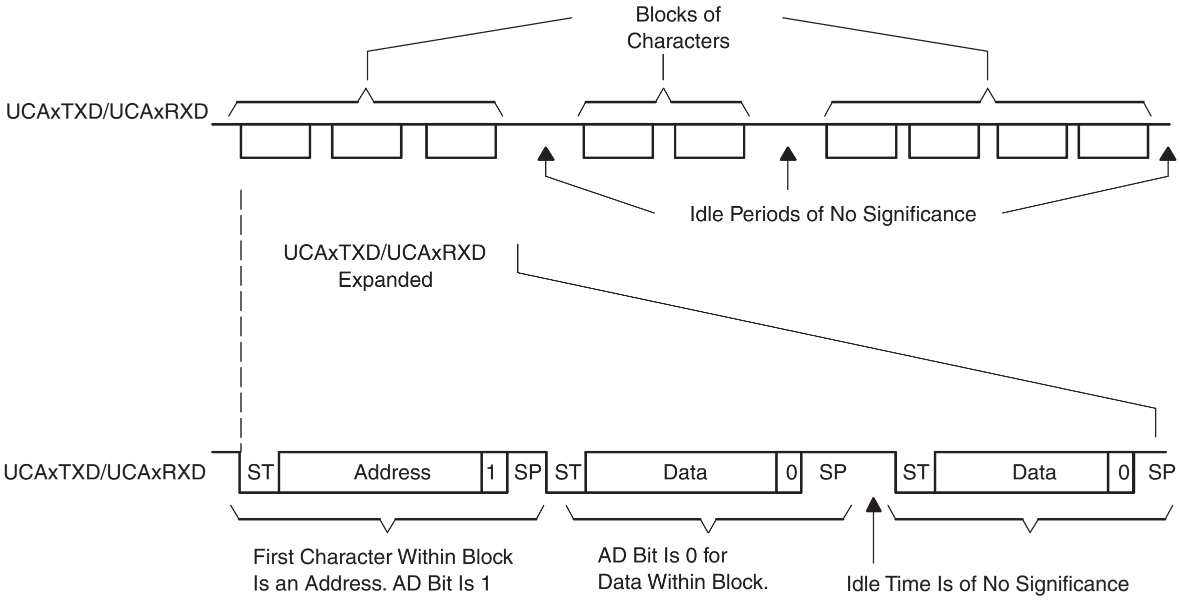

Blocks of Characters
UCAxTXD/UCAxRXD
Idle Periods of No Significance
UCAxTXD/UCAxRXD Expanded
UCAxTXD/UCAxRXD
ST
Address
1
SP
ST
Data
0
SP
ST
Data
0
SP
First Character Within Block Is an Address. AD Bit Is 1
AD Bit Is 0 for Data Within Block.
Idle Time Is of No Significance

Figure 15-4. 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 UCRXIFG 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 (UCTXIFG = 1). This generates a break with all bits low. UCTXBRK is automatically cleared when the start bit is generated.

15.3.4 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 21 bit times the break timeout error flag UCBTOE is set. The USCI can not transmit data while receiving the break/sync field. The synch field follows the break as shown in Figure 15-5.

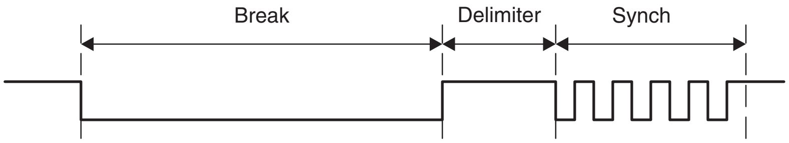


Figure 15-5. Auto Baud Rate Detection – Break/Synch Sequence

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 15-6. 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.

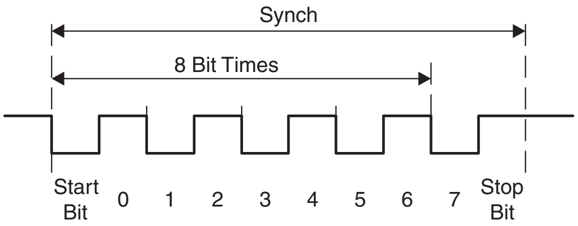


Figure 15-6. Auto Baud Rate Detection – Synch Field

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 UCRXIFG interrupt flag is set. Any applicable error flag is also set. If the UCBRKIE bit is set, reception of the break/synch sets the UCRXIFG. 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 UCRXIFG. If UCDORM is cleared during the reception of a character the receive interrupt flag will be set after the reception is complete.

The counter used to detect the baud rate is limited to 07FFFh (32767) counts. This means the minimum baud rate detectable is 488 Baud in oversampling mode and 30 Baud in low-frequency mode.

The automatic baud rate detection mode can be used in a full-duplex communication system with some restrictions. The USCI can not transmit data while receiving the break/sync field and if a 0h byte with framing error is received any data transmitted during this time gets corrupted. The latter case can be discovered by checking the received data and the UCFE bit.

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 (UCTXIFG = 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 (UCTXIFG = 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.

15.3.5 IrDA Encoding and Decoding

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

15.3.5.1 IrDA Encoding

The encoder sends a pulse for every zero bit in the transmit bit stream coming from the UART as shown in Figure 15-7. The pulse duration is defined by UCIRTXPLx bits specifying the number of half clock periods of the clock selected by UCIRTXCLK.

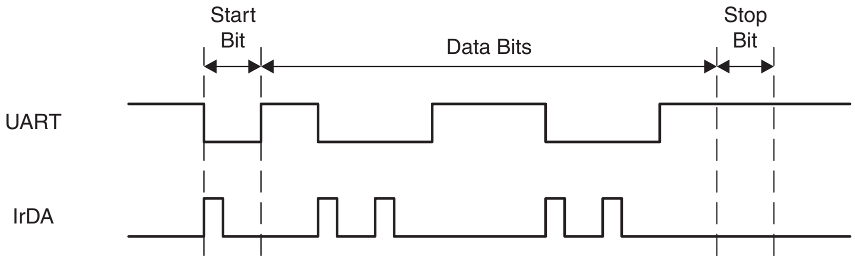


Figure 15-7. UART vs IrDA Data Format

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:

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

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:

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

where:

t_{PULSE} = Minimum receive pulse width

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

15.3.6 Automatic Error Detection

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

When a low period on UCAxRXD exceeds t_t 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 15-1.

Table 15-1. 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 UCRXIFG 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 any of the UCFE, UCPE, UCOE, UCBRK, or UCRXERR bit 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. To detect overflows reliably the following flow is recommended. After a character was received and UCAxRXIFG is set, first read UCAxSTAT to check the error flags including the overflow flag UCOE. Read UCAxRXBUF next. This will clear all error flags except UCOE if UCAxRXBUF was overwritten between the read access to UCAxSTAT and to UCAxRXBUF. So the UCOE flag should be checked after reading UCAxRXBUF to detect this condition. Note, in this case the UCRXERR flag is not set.

15.3.7 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_t (approximately 150 ns) will be ignored by the USCI and further action will be initiated as shown in Figure 15-8. See the device-specific datasheet for parameters.

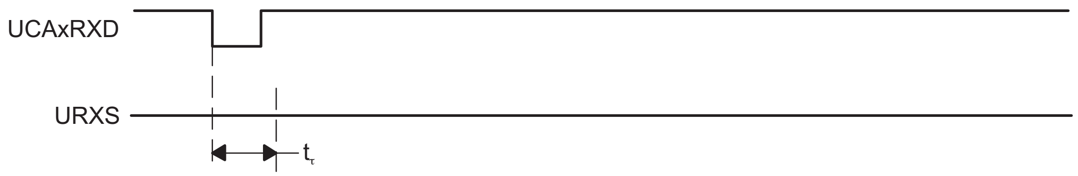


Figure 15-8. Glitch Suppression, USCI Receive Not Started

When a glitch is longer than $t_{t,}$ or a valid start bit occurs on UCAxRXD, the USCI receive operation is started and a majority vote is taken as shown in Figure 15-9. If the majority vote fails to detect a start bit the USCI halts character reception.

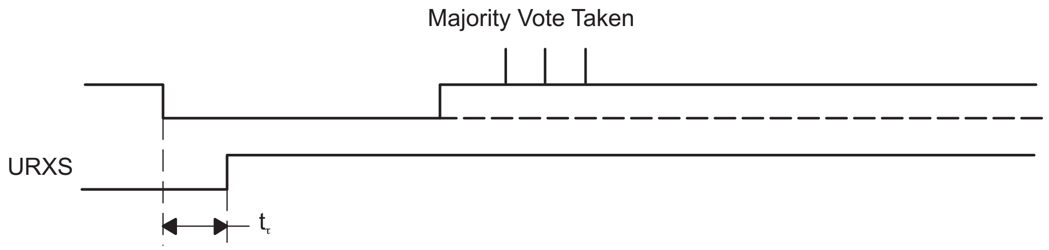


Figure 15-9. Glitch Suppression, USCI Activated

15.3.8 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. UCTXIFG 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.

15.3.9 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 15-10. 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.

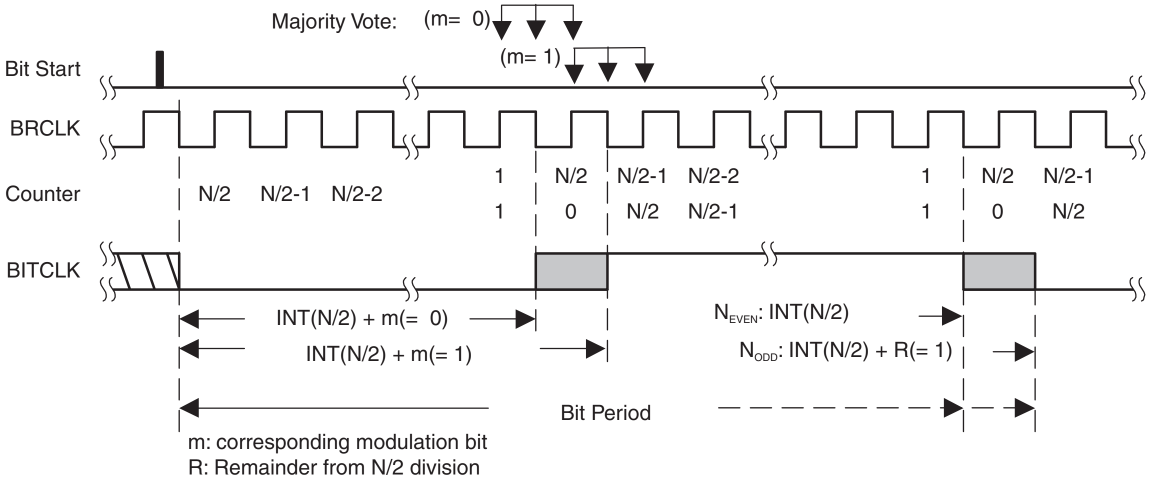


Figure 15-10. BITCLK Baud Rate Timing with UCOS16 = 0

Modulation is based on the UCBRSx setting as shown in Table 15-2. 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 15-2. 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 - in this case no modulation for the BITCLK16 is possible and thus the UCBRFx bits are ignored.

Modulation for BITCLK16 is based on the UCBRFx setting as shown in Table 15-3. 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 15-2 as previously described.

Table 15-3. 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

15.3.10 Setting a Baud Rate

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

$N = f_{BRCLK}/\text{Baudrate}$

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:

$\text{UCBRx} = \text{INT}(N)$

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

$\text{UCBRSx} = \text{round}(\ (N - \text{INT}(N)\ ) \times 8\ )$

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:

$\text{UCBRx} = \text{INT}(N/16)$

and the first stage modulator is set to:

$\text{UCBRFx} = \text{round}(\ (\ (N/16) - \text{INT}(N/16)\ ) \times 16\ )$

When greater accuracy is required, the UCBRSx modulator can also be implemented with values from 0 to 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 to 7 with the initial UCBRFx setting and with the UCBRFx setting incremented and decremented by one.

15.3.11 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:

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

where:

$m_{UCBRSx}[i]$ = Modulation of bit i from Table 15-2

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:

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

where:

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

$m_{UCBRSx}[i]$ = Modulation of bit i from Table 15-2

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 $t_{bit,ideal,TX}[i]$:

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

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

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

15.3.12 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 15-11 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 RCLKs, independent of the selected baud rate generation mode.

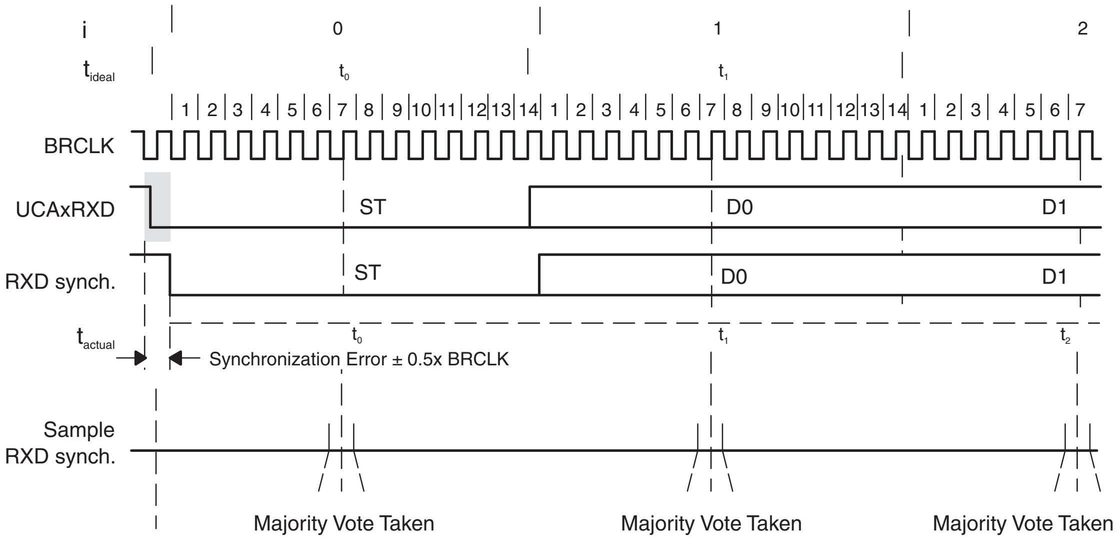


Figure 15-11. Receive Error

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

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

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:

$$t_{bit,RX}[i] = t_{SYNC} + \sum_{j=0}^{i-1} T_{bit,RX}[j] + \frac{1}{f_{BRCLK}}\left(INT(½UCBRx) + m_{UCBRSx}[i]\right)$$

where:

$$T_{bit,RX}[i] = (1/f_{BRCLK})(UCBRx + m_{UCBRSx}[i])$$

$m_{UCBRSx}[i]$ = Modulation of bit i from Table 15-2

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

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

where:

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

$\sum_{j=0}^{7 + m_{UCBRSx}[i]} m_{UCBRFx}[j]$ = Sum of ones from columns 0 to (7 + $m_{UCBRSx}[i]$) from the corresponding row in Table 15-3.

$m_{UCBRSx}[i]$ = Modulation of bit i from Table 15-2

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

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

15.3.13 Typical Baud Rates and Errors

Standard baud rate data for UCBRx, UCBRSx, and UCBRFx are listed in Table 15-4 and Table 15-5 for a 32,768-Hz crystal sourcing ACLK and typical SMCLK frequencies. Please ensure that the selected BRCLK frequency does not exceed the device specific maximum USCI input frequency. Please refer to the device-specific datasheet.

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 15-4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0

BRCLK Frequency (Hz)	Baud Rate (Baud)	UCBRx	UCBRSx	UCBRFx	Maximum TX Error (%)		Maximum 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
1,000,000	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	57600	17	3	0	-2.1	4.8	-6.8	5.8
1,000,000	115200	8	6	0	-7.8	6.4	-9.7	16.1
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	57600	18	1	0	-4.6	3.3	-6.8	6.6
1,048,576	115200	9	1	0	-1.1	10.7	-11.5	11.3
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	57600	69	4	0	-0.6	0.8	-1.8	1.1
4,000,000	115200	34	6	0	-2.1	0.6	-2.5	3.1
4,000,000	230400	17	3	0	-2.1	4.8	-6.8	5.8
4,194,304	9600	436	7	0	-0.3	0	-0.3	0.2
4,194,304	19200	218	4	0	-0.2	0.2	-0.3	0.6
4,194,304	57600	72	7	0	-1.1	0.6	-1.3	1.9
4,194,304	115200	36	3	0	-1.9	1.5	-2.7	3.4
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	57600	138	7	0	-0.7	0	-0.8	0.6
8,000,000	115200	69	4	0	-0.6	0.8	-1.8	1.1
8,000,000	230400	34	6	0	-2.1	0.6	-2.5	3.1
8,000,000	460800	17	3	0	-2.1	4.8	-6.8	5.8
8,388,608	9600	873	7	0	-0.1	0.06	-0.2	0,1
8,388,608	19200	436	7	0	-0.3	0	-0.3	0.2
8,388,608	57600	145	5	0	-0.5	0.3	-1.0	0.5
8,388,608	115200	72	7	0	-1.1	0.6	-1.3	1.9

Table 15-4. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 0 (continued)

BRCLK Frequency (Hz)	Baud Rate (Baud)	UCBRx	UCBRSx	UCBRFx	Maximum TX Error (%)		Maximum RX Error (%)	
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	57600	208	2	0	-0.5	0.2	-0.6	0.5
12,000,000	115200	104	1	0	-0.5	0.6	-0.9	1.2
12,000,000	230400	52	0	0	-1.8	0	-2.6	0.9
12,000,000	460800	26	0	0	-1.8	0	-3.6	1.8
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	57600	277	7	0	-0.3	0.3	-0.5	0.4
16,000,000	115200	138	7	0	-0.7	0	-0.8	0.6
16,000,000	230400	69	4	0	-0.6	0.8	-1.8	1.1
16,000,000	460800	34	6	0	-2.1	0.6	-2.5	3.1
16,777,216	9600	1747	5	0	-0.04	0.03	-0.08	0.05
16,777,216	19200	873	7	0	-0.09	0.06	-0.2	0.1
16,777,216	57600	291	2	0	-0.2	0.2	-0.5	0.2
16,777,216	115200	145	5	0	-0.5	0.3	-1.0	0.5
20,000,000	9600	2083	2	0	-0.05	0.02	-0.09	0.02
20,000,000	19200	1041	6	0	-0.06	0.06	-0.1	0.1
20,000,000	38400	520	7	0	-0.2	0.06	-0.2	0.2
20,000,000	57600	347	2	0	-0.06	0.2	-0.3	0.3
20,000,000	115200	173	5	0	-0.4	0.3	-0.8	0.5
20,000,000	230400	86	7	0	-1.0	0.6	-1.0	1.7
20,000,000	460800	43	3	0	-1.4	1.3	-3.3	1.8

Table 15-5. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 1

BRCLK Frequency (Hz)	Baud Rate (Baud)	UCBRx	UCBRSx	UCBRFx	Maximum TX Error (%)		Maximum RX Error (%)	
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
1,048,576	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
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	57600	4	5	3	-3.5	3.2	-1.8	6.4
4,000,000	115200	2	3	2	-2.1	4.8	-2.5	7.3
4,194,304	9600	27	0	5	0	0.2	0	0.5
4,194,304	19200	13	0	10	-2.3	0	-2.4	0.1
4,194,304	57600	4	4	7	-2.5	2.5	-1.3	5.1
4,194,304	115200	2	6	3	-3.9	2.0	-1.9	6.7
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

Table 15-5. Commonly Used Baud Rates, Settings, and Errors, UCOS16 = 1 (continued)

BRCLK Frequency (Hz)	Baud Rate (Baud)	UCBRx	UCBRSx	UCBRFx	Maximum TX Error (%)		Maximum RX Error (%)	
8,000,000	38400	13	0	0	-1.8	0	-1.9	0.2
8,000,000	57600	8	0	11	0	0.88	0	1.6
8,000,000	115200	4	5	3	-3.5	3.2	-1.8	6.4
8,000,000	230400	2	3	2	-2.1	4.8	-2.5	7.3
8,388,608	9600	54	0	10	0	0.2	-0.05	0.3
8,388,608	19200	27	0	5	0	0.2	0	0.5
8,388,608	57600	9	0	2	0	2.8	-0.2	3.0
8,388,608	115200	4	4	7	-2.5	2.5	-1.3	5.1
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	57600	13	0	0	-1.8	0	-1.9	0.2
12,000,000	115200	6	0	8	-1.8	0	-2.2	0.4
12,000,000	230400	3	0	4	-1.8	0	-2.6	0.9
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	57600	17	0	6	0	0.9	-0.1	1.0
16,000,000	115200	8	0	11	0	0.9	0	1.6
16,000,000	230400	4	5	3	-3.5	3.2	-1.8	6.4
16,000,000	460800	2	3	2	-2.1	4.8	-2.5	7.3
16,777,216	9600	109	0	4	0	0.2	-0.02	0.3
16,777,216	19200	54	0	10	0	0.2	-0.05	0.3
16,777,216	57600	18	0	3	-1.0	0	-1.0	0.3
16,777,216	115200	9	0	2	0	2.8	-0.2	3.0
20,000,000	9600	130	0	3	-0.2	0	-0.2	0.04
20,000,000	19200	65	0	2	0	0.4	-0.03	0.4
20,000,000	38400	32	0	9	0	0.4	0	0.5
20,000,000	57600	21	0	11	-0.7	0	-0.7	0.3
20,000,000	115200	10	0	14	0	2.5	-0.2	2.6
20,000,000	230400	5	0	7	0	2.5	0	3.5
20,000,000	460800	2	6	10	-3.2	1.8	-2.8	4.6

15.3.14 Using the USCI Module in UART Mode with Low Power Modes

The USCI module provides automatic clock activation for use with low-power modes. When the USCI clock source 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.

15.3.15 USCI Interrupts

The USCI has only one interrupt vector that is shared for transmission and for reception. USCI_Ax and USC_Bx do not share the same interrupt vector.

USCI Transmit Interrupt Operation

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

UCTXIFG is set after a PUC or when UCSWRST = 1. UCTXIE is reset after a PUC or when UCSWRST = 1.

USCI Receive Interrupt Operation

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

Additional interrupt control features include:

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

UCAxIV, Interrupt Vector Generator

The USCI interrupt flags are prioritized and combined to source a single interrupt vector. The interrupt vector register UCAxIV is used to determine which flag requested an interrupt. The highest priority enabled interrupt generates a number in the UCAxIV register that can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled interrupts do not affect the UCAxIV value.

Any access, read or write, of the UCAxIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.

UCAxIV Software Example

The following software example shows the recommended use of UCAxIV. The UCAxIV value is added to the PC to automatically jump to the appropriate routine. The following example is given for USCI_A0.

```
USCI_UART_ISR
         ADD    &UCA0IV, PC   ; Add offset to jump table
         RETI                 ; Vector 0: No interrupt
         JMP    RXIFG_ISR     ; Vector 2: RXIFG
TXIFG_ISR                     ; Vector 4: TXIFG
         ...                  ; Task starts here
         RETI                 ; Return
RXIFG_ISR                     ; Vector 2
         ...                  ; Task starts here
         RETI                 ; Return
```

15.4 USCI Registers: UART Mode

The USCI registers applicable in UART mode listed in Table 15-6. The word accessible registers are listed in Table 15-7.

Table 15-6. USCI_Ax Registers

Register	Short Form	Register Type	Address Offset	Initial State
USCI_Ax control register 0	UCAxCTL0	Byte - R/W	+01h	Reset with PUC
USCI_Ax control register 1	UCAxCTL1	Byte - R/W	+00h	001h with PUC
USCI_Ax Baud rate control register 0	UCAxBR0	Byte - R/W	+06h	Reset with PUC
USCI_Ax Baud rate control register 1	UCAxBR1	Byte - R/W	+07h	Reset with PUC
USCI_Ax modulation control register	UCAxMCTL	Byte - R/W	+08h	Reset with PUC
Reserved - reads zero		Byte - R only	+09h	000h
USCI_Ax status register	UCAxSTAT	Byte - R/W	+0Ah	Reset with PUC
Reserved - reads zero		Byte - R only	+0Bh	000h
USCI_Ax Receive buffer register	UCAxRXBUF	Byte - R/W	+0Ch	Reset with PUC
Reserved - reads zero		Byte - R only	+0Dh	000h
USCI_Ax Transmit buffer register	UCAxTXBUF	Byte - R/W	+0Eh	Reset with PUC
Reserved - reads zero		Byte - R only	+0Fh	000h
USCI_Ax Auto Baud control register	UCAxABCTL	Byte - R/W	+10h	Reset with PUC
Reserved - reads zero		Byte - R only	+11h	000h
USCI_Ax IrDA Transmit control register	UCAxIRTCTL	Byte - R/W	+12h	Reset with PUC
USCI_Ax IrDA Receive control register	UCAxIRRCTL	Byte - R/W	+13h	Reset with PUC
USCI_Ax interrupt enable register	UCAxIE	Byte - R/W	+1Ch	Reset with PUC
USCI_Ax interrupt flag register	UCAxIFG	Byte - R/W	+1Dh	Reset with PUC
USCI_Ax interrupt vector register	UCAxIV	Word - R	+1Eh	Reset with PUC

Table 15-7. Word Access to USCI_Ax Registers

Word Register	Short Form	High-Byte Register	Low-Byte Register	Address Offset
USCI_Ax control word register 0	UCAxCTLW0	UCAxCTL0	UCAxCTL1	+00h
USCI_Ax Baud rate control word register	UCAxBRW	UCAxBR1	UCAxBR0	+06h
USCI_Ax IrDA control register	UCAxIRCTL	UCAxIRRCTL	UCAxIRTCTL	+12h
USCI_Ax interrupt control register	UCAxICTL	UCAxIFG	UCAxIE	+1Ch

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

0 Parity disabled

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.

0 Odd parity

1 Even parity

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

UCSPB Bit 3 Stop bit select. Number of stop bits.

0 One stop bit

1 Two stop bits

UCMODEx Bits 2-1 USCI mode. The UCMODEx bits select the asynchronous mode when UCSYNC = 0.

00 UART mode

01 Idle-line multiprocessor mode

10 Address-bit multiprocessor mode

11 UART mode with automatic baud rate detection

UCSYNC Bit 0 Synchronous mode enable

0 Asynchronous mode

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.

00 UCLK

01 ACLK

10 SMCLK

11 SMCLK

UCRXEIE Bit 5 Receive erroneous-character interrupt-enable

0 Erroneous characters rejected and UCRXIFG is not set

1 Erroneous characters received will set UCRXIFG

UCBRKIE Bit 4 Receive break character interrupt-enable

0 Received break characters do not set UCRXIFG.

1 Received break characters set UCRXIFG.

UCDORM Bit 3 Dormant. Puts USCI into sleep mode.

0 Not dormant. All received characters will set UCRXIFG.

1 Dormant. Only characters that are preceded by an idle-line or with address bit set will set UCRXIFG. In UART mode with automatic baud rate detection only the combination of a break and synch field will set UCRXIFG.

UCTXADDR Bit 2 Transmit address. Next frame to be transmitted will be marked as address depending on the selected multiprocessor mode.

0 Next frame transmitted is data

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.

0 Next frame transmitted is not a break

1 Next frame transmitted is a break or a break/synch

UCSWRST Bit 0 Software reset enable

0 Disabled. USCI reset released for operation.

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 15-3 shows the modulation pattern.

UCBRSx Bits 3-1 Second modulation stage select. These bits determine the modulation pattern for BITCLK. Table 15-2 shows the modulation pattern.

UCOS16 Bit 0 Oversampling mode enabled

0 Disabled

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.

0 Disabled

1 Enabled. UCAxTXD is internally fed back to the receiver.

UCFE Bit 6 Framing error flag

0 No error

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.

0 No error

1 Overrun error occurred

UCPE Bit 4 Parity error flag. When UCPEN = 0, UCPE is read as 0.

0 No error

1 Character received with parity error

UCBRK Bit 3 Break detect flag

0 No break condition

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.

0 No receive errors detected

1 Receive error detected

UCADDR Bit 1 Address received in address-bit multiprocessor mode.

0 Received character is data

1 Received character is an address

UCIDLE Idle line detected in idle-line multiprocessor mode.

0 No idle line detected

1 Idle line detected

UCBUSY Bit 0 USCI busy. This bit indicates if a transmit or receive operation is in progress.

0 USCI inactive

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 UCRXIFG. 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 UCTXIFG. 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

00 1 bit time

01 2 bit times

10 3 bit times

11 4 bit times

UCSTOE Bit 3 Synch field time out error

0 No error

1 Length of synch field exceeded measurable time.

UCBTOE Bit 2 Break time out error

0 No error

1 Length of break field exceeded 22 bit times.

Reserved Bit 1 Reserved

UCABDEN Bit 0 Automatic baud rate detect enable

0 Baud rate detection disabled. Length of break and synch field is not measured.

1 Baud rate detection enabled. Length of break and synch field is measured and baud rate settings are changed accordingly.

UCAxIE, USCI_Ax Interrupt Enable Register

7	6	5	4	3	2	1	0
Reserved						**UCTXIE**	**UCRXIE**
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

Reserved Bits 7-2 Reserved

UCTXIE Bit 1 Transmit interrupt enable

0 Interrupt disabled

1 Interrupt enabled

UCRXIE Bit 0 Receive interrupt enable

0 Interrupt disabled

1 Interrupt enabled

UCAxIFG, USCI_Ax Interrupt Flag Register

7	6	5	4	3	2	1	0
Reserved						**UCTXIFG**	**UCRXIFG**
r-0	r-0	r-0	r-0	r-0	r-0	rw-1	rw-0

Reserved Bits 7-2 Reserved

UCTXIFG Bit 1 Transmit interrupt flag. UCTXIFG is set when UCAxTXBUF empty.

0 No interrupt pending

1 Interrupt pending

UCRXIFG Bit 0 Receive interrupt flag. UCRXIFG is set when UCAxRXBUF has received a complete character.

0 No interrupt pending

1 Interrupt pending

UCAxIV, USCI_Ax 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**	**0**	**UCIVx**		**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

UCIVx Bits 15-0 USCI interrupt vector value

UCAxIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No interrupt pending		
002h	Data received	UCRXIFG	Highest
004h	Transmit buffer empty	UCTXIFG	Lowest

Chapter 16

SLAU208–June 2008

Universal Serial Communication Interface, SPI Mode

The 5xx 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**

16.1 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^{2}C mode
- SPI mode

16.2 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 16-1 shows the USCI when configured for SPI mode.

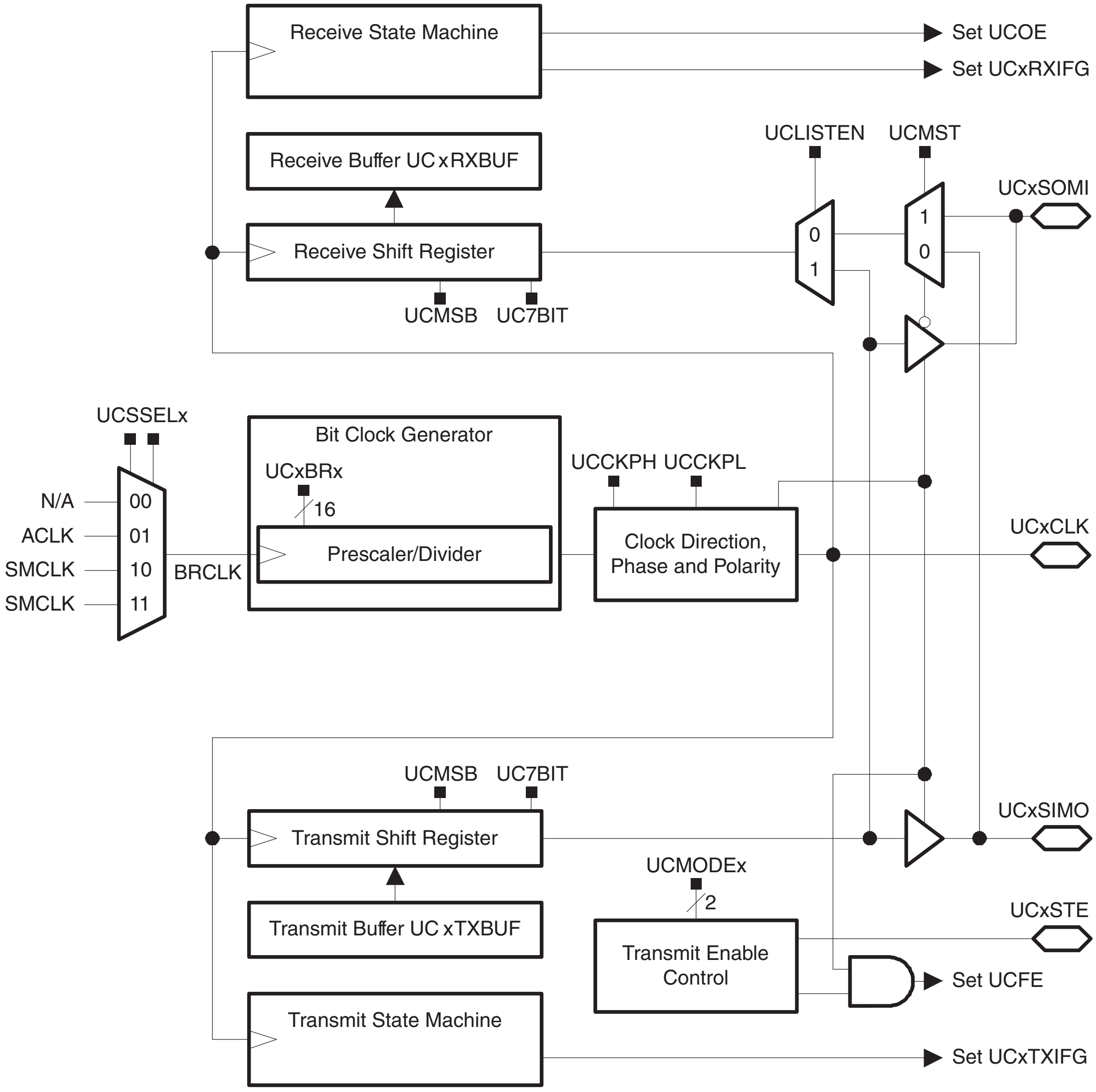


Figure 16-1. USCI Block Diagram: SPI Mode

16.3 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 16-1 describes the UCxSTE operation.

Table 16-1. 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

16.3.1 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 UCRXIE, UCTXIE, UCRXIFG, UCOE, and UCFE bits and sets the UCTXIFG 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 UCRXIE and/or UCTXIE.

16.3.2 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.

16.3.3 Master Mode

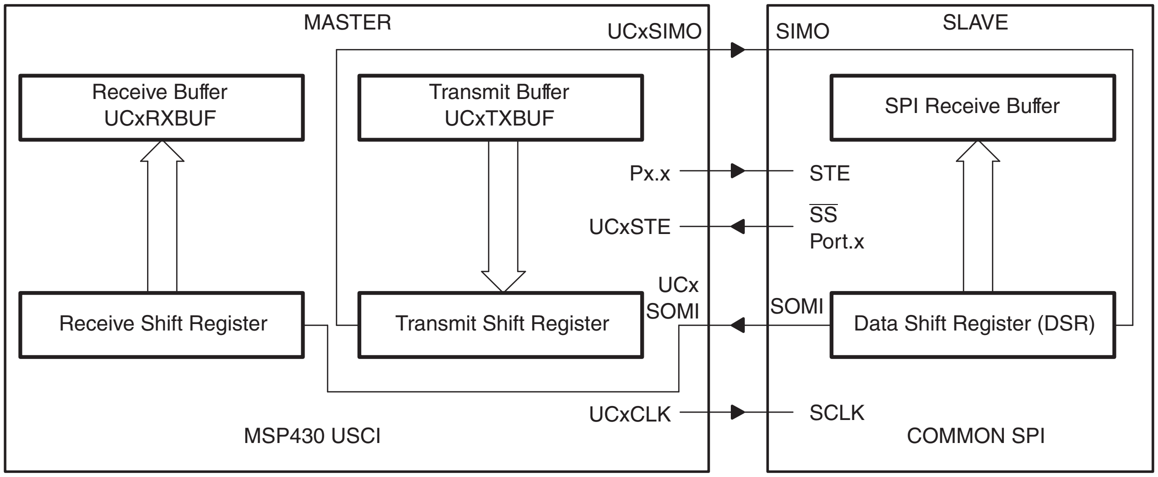


Figure 16-2. USCI Master and External Slave

Figure 16-2 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, UCRXIFG, is set, indicating the RX/TX operation is complete.

A set transmit interrupt flag, UCTXIFG, 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 16-1. 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.

16.3.4 Slave Mode

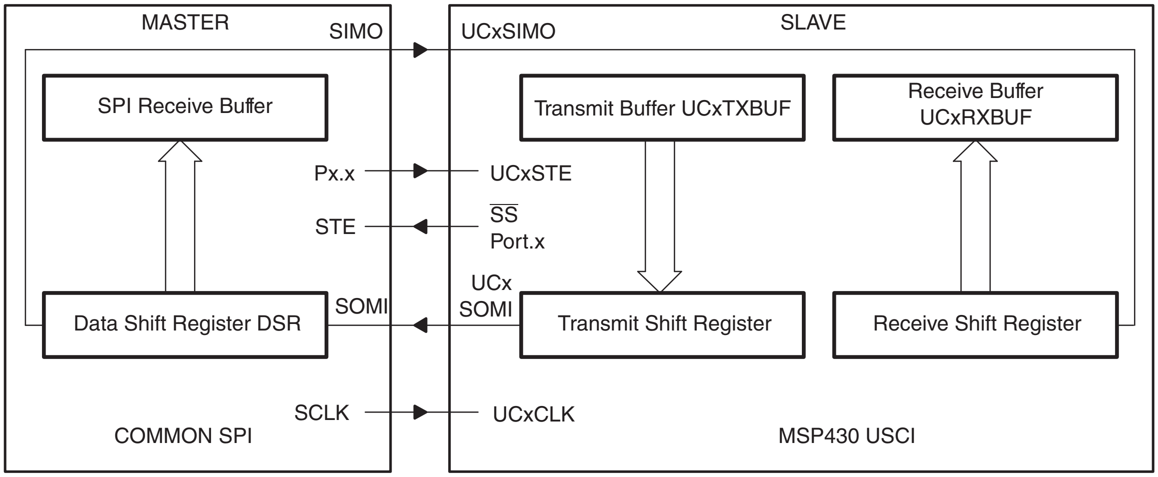


Figure 16-3. USCI Slave and External Master

Figure 16-3 shows the USCI as a slave in both 3-pin and 4-pin configurations. UCxCLK is used as the inputfor 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 UCRXIFG 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.

16.3.5 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.

16.3.6 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:

$f_{BitClock} = f_{BRCLK}/UCBRx$

16.3.6.1 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 16-4.

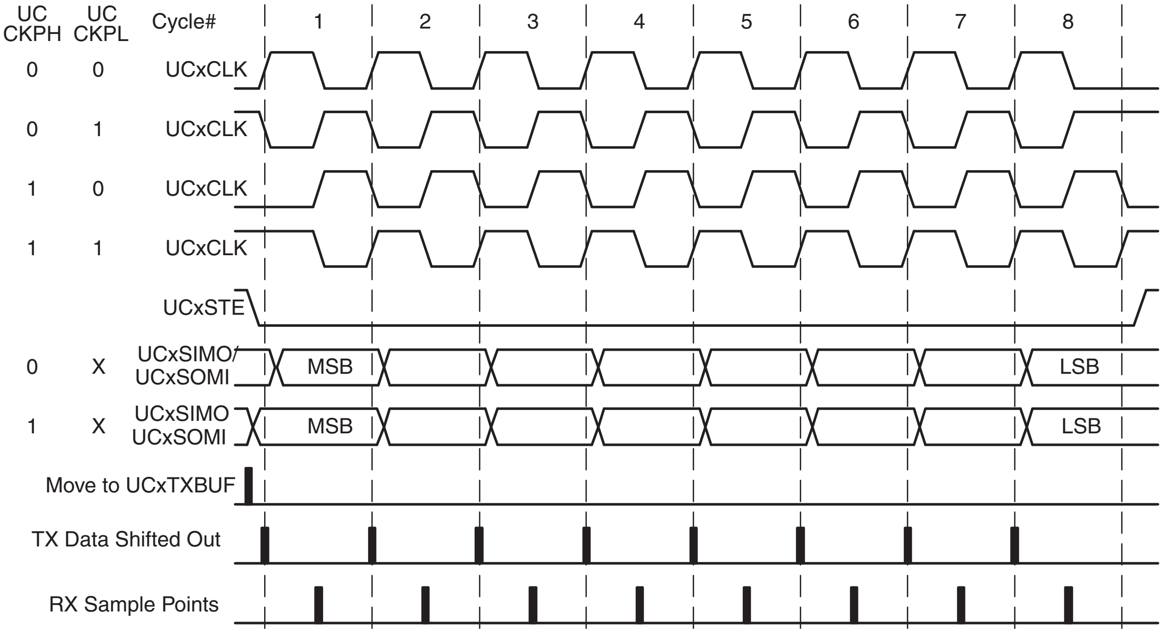


Figure 16-4. USCI SPI Timing with UCMSB = 1

16.3.7 Using the SPI Mode with Low Power Modes

The USCI module provides automatic clock activation for use with low-power modes. When the USCI clock source 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.

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.

16.3.8 SPI Interrupts

The USCI has only one interrupt vector that is shared for transmission and for reception. USCI_Ax and USC_Bx do not share the same interrupt vector.

SPI Transmit Interrupt Operation

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

Note: Writing to UCxTXBUF in SPI Mode

Data written to UCxTXBUFwhen UCTXIFG = 0 may result in erroneous data transmission.

SPI Receive Interrupt Operation

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

UCxIV, Interrupt Vector Generator

The USCI interrupt flags are prioritized and combined to source a single interrupt vector. The interrupt vector register UCxIV is used to determine which flag requested an interrupt. The highest priority enabled interrupt generates a number in the UCxIV register that can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled interrupts do not affect the UCxIV value.

Any access, read or write, of the UCxIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.

UCxIV Software Example

The following software example shows the recommended use of UCxIV. The UCxIV value is added to the PC to automatically jump to the appropriate routine. The following example is given for USCI_B0.

```
USCI_SPI_ISR
         ADD      &UCB0IV, PC  ; Add offset to jump table
         RETI                  ; Vector 0: No interrupt
         JMP      RXIFG_ISR    ; Vector 2: RXIFG
TXIFG_ISR                      ; Vector 4: TXIFG
         ...                   ; Task starts here
         RETI                  ; Return
RXIFG_ISR                      ; Vector 2
         ...                   ; Task starts here
         RETI                  ; Return
```

16.4 USCI Registers: SPI Mode

The USCI registers applicable in SPI mode listed in Table 16-2. The word accessible registers are listed in Table 16-3.

Table 16-2. USCI_xx Registers

Register	Short Form	Register Type	Address Offset	Initial State
USCI_Ax control register 0	UCAxCTL0	Byte - R/W	+01h	Reset with PUC
USCI_Bx control register 0	UCBxCTL0	Byte - R/W	+01h	001h with PUC
USCI_xx control register 1	UCxxCTL1	Byte - R/W	+00h	001h with PUC
USCI_xx Bit rate control register 0	UCxxBR0	Byte - R/W	+06h	Reset with PUC
USCI_xx Bit rate control register 1	UCxxBR1	Byte - R/W	+07h	Reset with PUC
USCI_Ax modulation control register	UCAxMCTL	Byte - R/W	+08h	Reset with PUC
USCI_xx status register	UCxxSTAT	Byte - R/W	+0Ah	Reset with PUC
Reserved - reads zero		Byte - R only	+0Bh	000h
USCI_xx Receive buffer register	UCxxRXBUF	Byte - R/W	+0Ch	Reset with PUC
Reserved - reads zero		Byte - R only	+0Dh	000h
USCI_xx Transmit buffer register	UCxxTXBUF	Byte - R/W	+0Eh	Reset with PUC
Reserved - reads zero		Byte - R only	+0Fh	000h
USCI_xx interrupt enable register	UCxxIE	Byte - R/W	+1Ch	Reset with PUC
USCI_xx interrupt flag register	UCxxIFG	Byte - R/W	+1Dh	002h with PUC
USCI_xx interrupt vector register	UCxxIV	Word - R	+1Eh	Reset with PUC

Table 16-3. Word Access to USCI_xx Registers

Word Register	Short Form	High-Byte Register	Low-Byte Register	Address Offset
USCI_xx control word register 0	UCxxCTLW0	UCxxCTL0	UCxxCTL1	+00h
USCI_xx bit rate control word register	UCxxBRW	UCxxBR1	UCxxBR0	+06h
USCI_xx interrupt control register	UCxxICTL	UCxxIFG	UCxxIE	+1Ch

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	rw-0[1] rw-1[2]

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^{2}C mode
UCSYNC	Bit 0	Synchronous mode enable
		0 Asynchronous mode
		1 Synchronous mode

[1] UCAxCTL0 (USCI_Ax)
[2] UCBxCTL0 (USCI_Bx)

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[1] r0[2]	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.
		00 NA
		01 ACLK
		10 SMCLK
		11 SMCLK
Unused	Bits 5-1	Unused
UCSWRST	Bit 0	Software reset enable
		0 Disabled. USCI reset released for operation.
		1 Enabled. USCI logic held in reset state.

[1] UCAxCTL1 (USCI_Ax)
[2] UCBxCTL1 (USCI_Bx)

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} forms the prescaler value.

UCAxMCTL, USCI_Ax Modulation Control Register

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

Bits 7-0 Write as 0.

UCAxSTAT, USCI_Ax Status Register
UCBxSTAT, USCI_Bx Status Register

7	6	5	4	3	2	1	0
UCLISTEN	**UCFE**	**UCOE**	**Unused**				**UCBUSY**
rw-0	rw-0	rw-0	rw-0[1] r0[2]	rw-0[1] r0[2]	rw-0[1] r0[2]	rw-0[1] r0[2]	r-0

UCLISTEN Bit 7 Listen enable. The UCLISTEN bit selects loopback mode.

0 Disabled

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.

0 No error

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.

0 No error

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.

0 USCI inactive

1 USCI transmitting or receiving

[1] UCAxSTAT (USCI_Ax)
[2] UCBxSTAT (USCI_Bx)

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 UCRXIFG. 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 UCTXIFG. The MSB of UCxTXBUF is not used for 7-bit data and is reset.

UCAxIE, USCI_Ax Interrupt Enable Register
UCBxIE, USCI_Bx Interrupt Enable Register

7	6	5	4	3	2	1	0
Reserved						**UCTXIE**	**UCRXIE**
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

Reserved Bits 7-2 Reserved

UCTXIE Bit 1 Transmit interrupt enable

0 Interrupt disabled

1 Interrupt enabled

UCRXIE Bit 0 Receive interrupt enable

0 Interrupt disabled

1 Interrupt enabled

UCAxIFG, USCI_Ax Interrupt Flag Register
UCBxIFG, USCI_Bx Interrupt Flag Register

7	6	5	4	3	2	1	0
Reserved						**UCTXIFG**	**UCRXIFG**
r-0	r-0	r-0	r-0	r-0	r-0	rw-1	rw-0

Reserved Bits 7-2 Reserved

UCTXIFG Bit 1 Transmit interrupt flag. UCTXIFG is set when UCxxTXBUF empty.

0 No interrupt pending

1 Interrupt pending

UCRXIFG Bit 0 Receive interrupt flag. UCRXIFG is set when UCxxRXBUF has received a complete character.

0 No interrupt pending

1 Interrupt pending

UCAxIV, USCI_Ax Interrupt Vector Register
UCBxIV, USCI_Bx 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**	**0**	**UCIVx**		**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

UCIVx Bits 15-0 USCI interrupt vector value

UCAxIV/ UCBxIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No interrupt pending	–	
002h	Data received	UCRXIFG	Highest
004h	Transmit buffer empty	UCTXIFG	Lowest

Chapter 17

SLAU208 – June 2008

Universal Serial Communication Interface, I2C Mode

The 5xx universal serial communication interface (USCI) supports multiple serial communication modes with one hardware module. This chapter discusses the operation of the I^{2}C mode.

Topic **Page**

17.1 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^{2}C mode
- SPI mode

17.2 USCI Introduction: I^2C 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
- J 7-bit and 10-bit device addressing modes
- J General call
- J START/RESTART/STOP
- J Multi-master transmitter/receiver mode
- J Slave receiver/transmitter mode
- J Standard mode up to 100 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 17-1 shows the USCI when configured in I^2C mode.

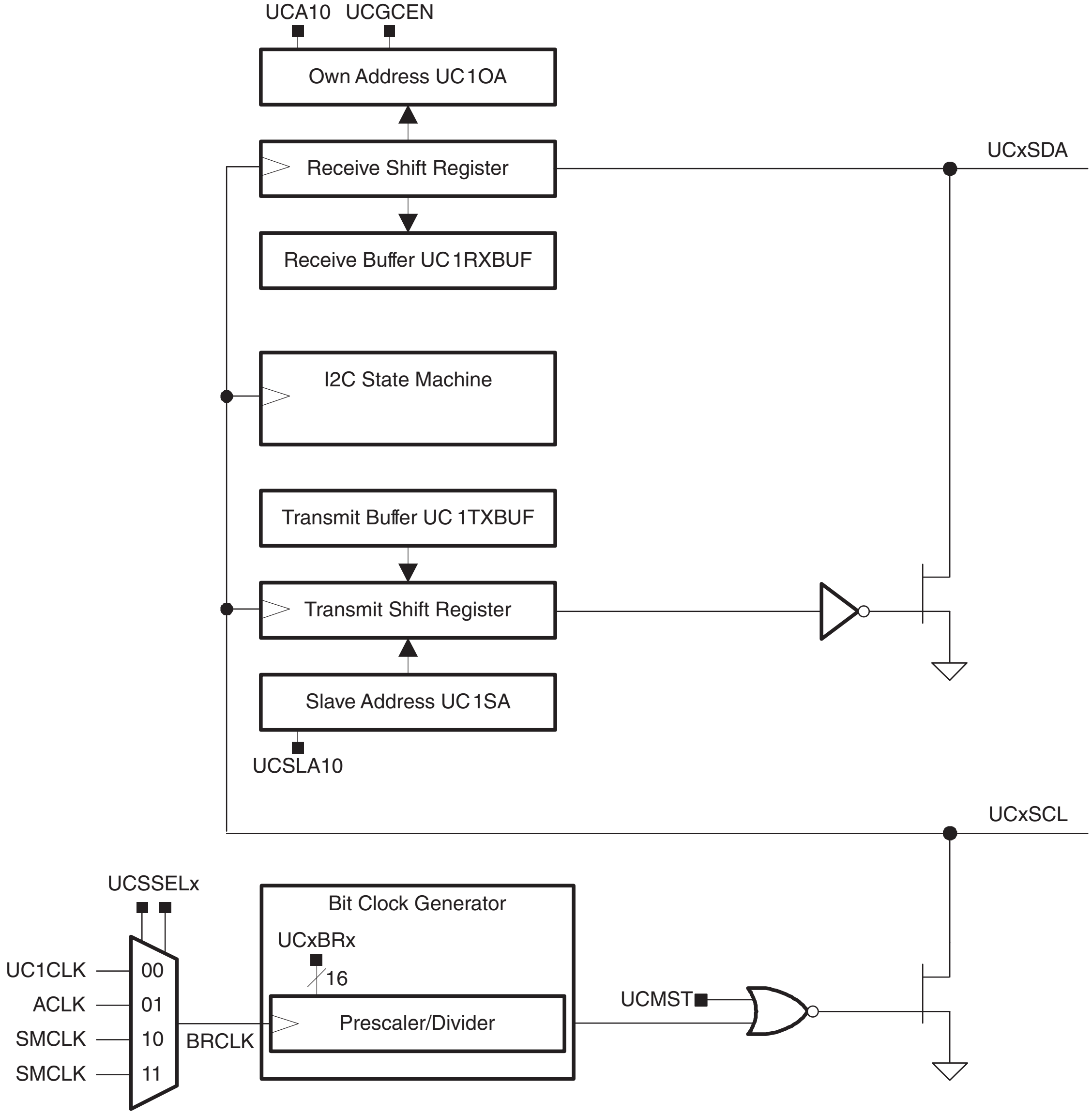


Figure 17-1. USCI Block Diagram: I^{2}C Mode

17.3 USCI Operation: I^{2}C Mode

The I^{2}C mode supports any slave or master I^{2}C-compatible device. Figure 17-2 shows an example of an I^{2}C bus. Each I^{2}C device is recognized by a unique address and can operate as either a transmitter or a receiver. A device connected to the I^{2}C 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.

I^{2}C 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.

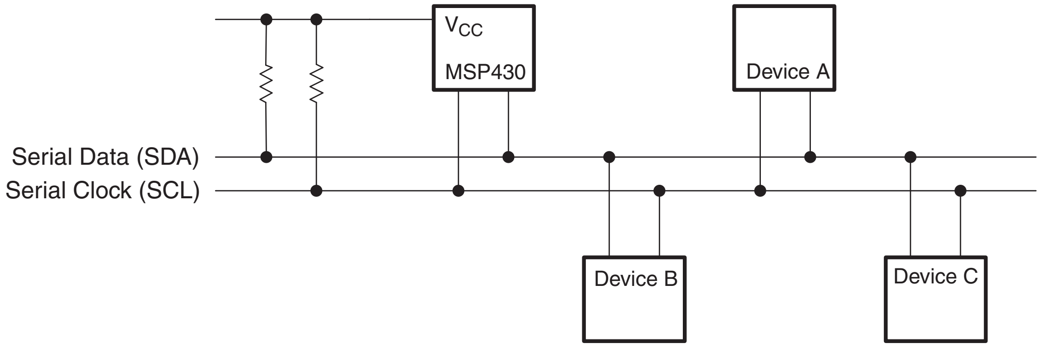


Figure 17-2. I^{2}C Bus Connection Diagram

Note: SDA and SCL Levels

The MSP430 SDA and SCL pins must not be pulled up above the MSP430 VCC level.

17.3.1 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^{2}C 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^{2}C mode has the following effects:

- I^{2}C communication stops
- SDA and SCL are high impedance
- UCBxI2CSTAT, bits 6-0 are cleared
- UCTXIE and UCRXIE are cleared
- UCTXIFG and UCRXIFG are cleared
- All other bits and register 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

17.3.2 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 17-3.

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.

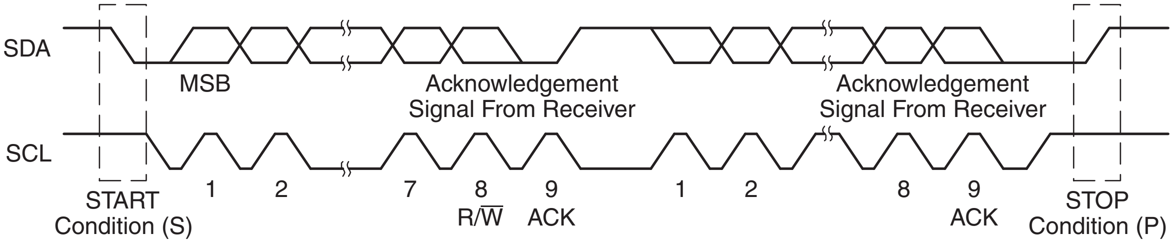


Figure 17-3. I2C Module Data Transfer

START and STOP conditions are generated by the master and are shown in Figure 17-3. 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 17-4. The high and low state of SDA can only change when SCL is low, otherwise START or STOP conditions will be generated.

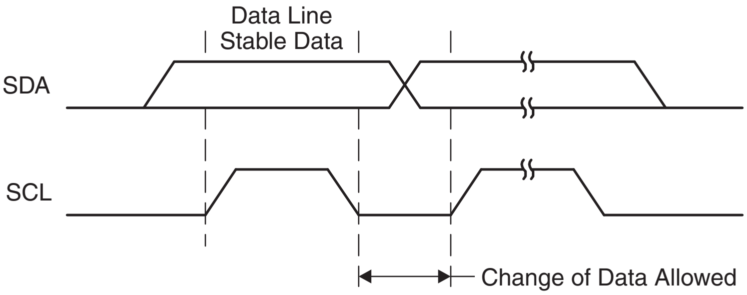


Figure 17-4. Bit Transfer on the I2C Bus

17.3.3 I²C Addressing Modes

The I²C mode supports 7-bit and 10-bit addressing modes.

7-Bit Addressing

In the 7-bit addressing format, shown in Figure 17-5, 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.

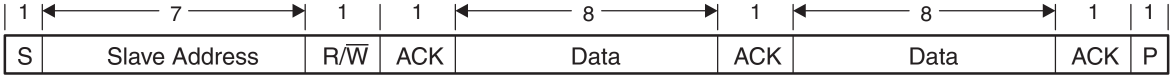


Figure 17-5. I²C Module 7-Bit Addressing Format

10-Bit Addressing

In the 10-bit addressing format, shown in Figure 17-6, 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.

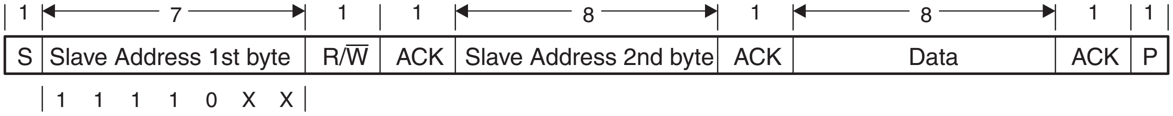


Figure 17-6. I²C Module 10-Bit Addressing Format

Repeated Start Conditions

The direction of data flow on SDA can be changed by the master, without first stopping a transfer, by issuing a repeated START condition. This is called a RESTART. After a RESTART is issued, the slave address is again sent out with the new data direction specified by the R/$\overline{W}$ bit. The RESTART condition is shown in Figure 17-7.

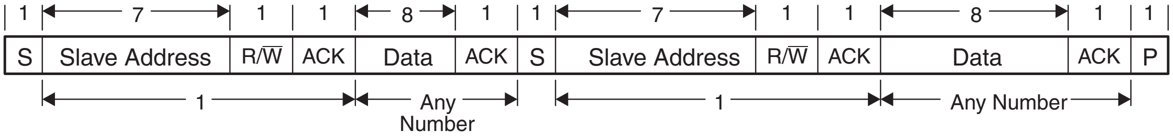


Figure 17-7. I²C Module Addressing Format with Repeated START Condition

17.3.4 I²C Module Operating Modes

In I²C 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 17-8 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.

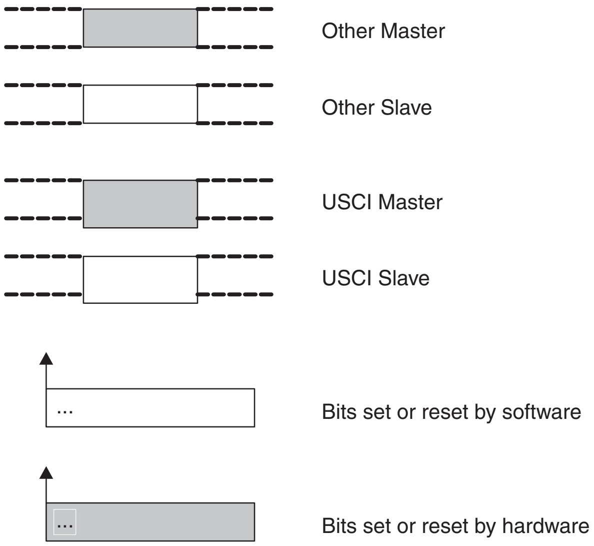


Figure 17-8. I²C Time Line Legend

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.

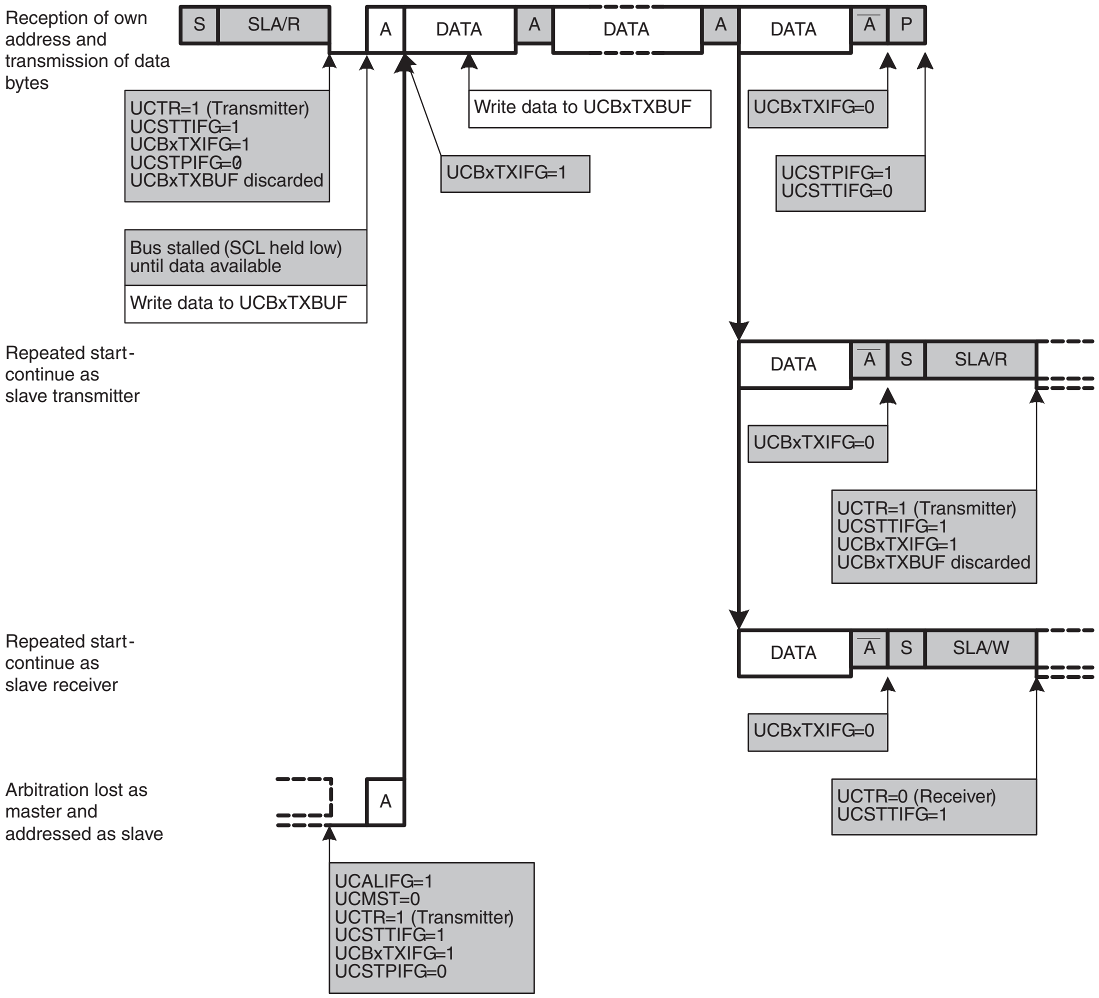
I²C Slave Transmitter Mode

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 UCTXIFG 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 UCTXIFG 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 17-9 illustrates the slave transmitter operation.



A   5xx: Replace UCBxTXIFG with UCTXIFG. Replace UCBxRXIFG with UCRXIFG.

Figure 17-9. 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 UCRXIFG is set. The USCI module automatically acknowledges the received data and can receive the next data byte.

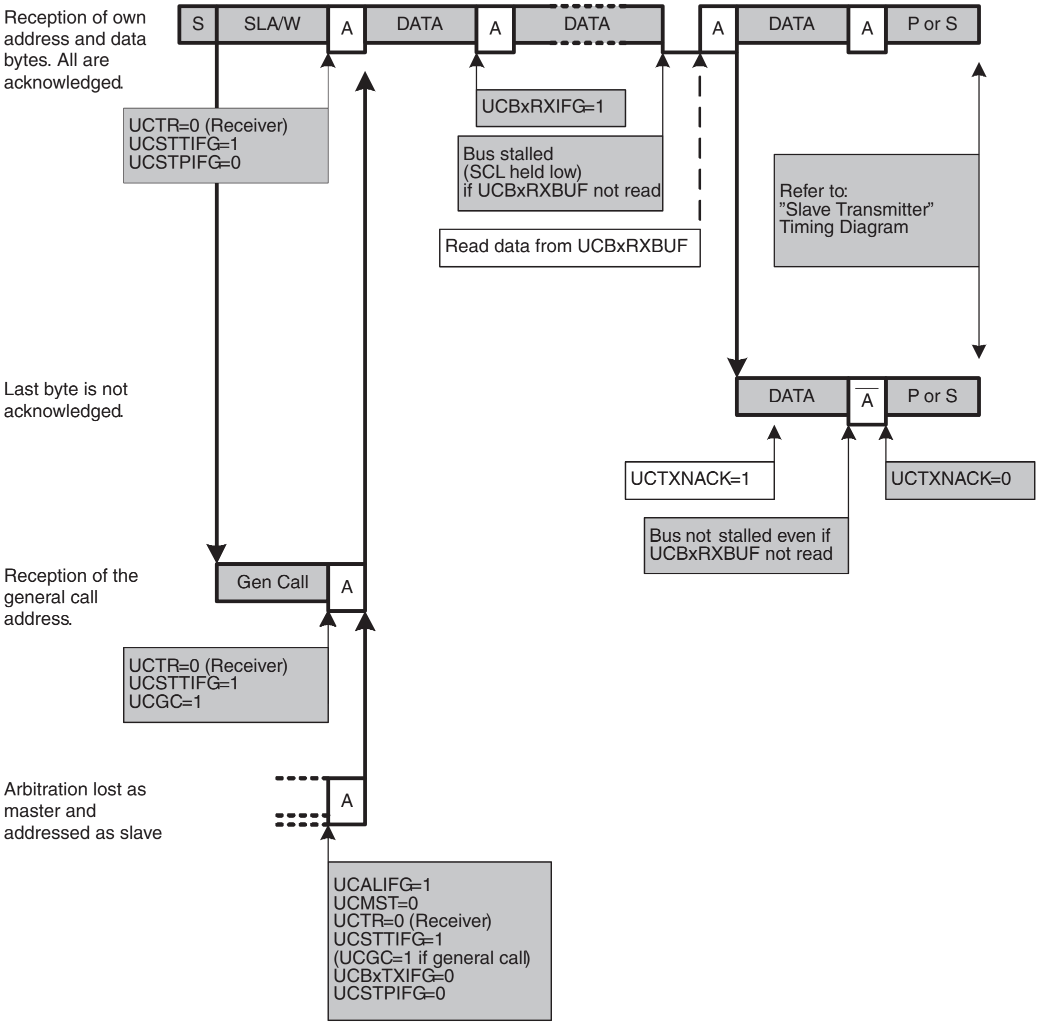
If the previous data wasn't not read from the receive buffer UCBxRXBUF at the end of a reception, the bus is stalled by holding SCL low. As soon as UCBxRXBUF is read the new data is transferred into UCBxRXBUF, an acknowledge is sent to the master, and the next data can be received.

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^{2}C state machine returns to its address reception state.

Figure 17-10 illustrates the the I^{2}C slave receiver operation.

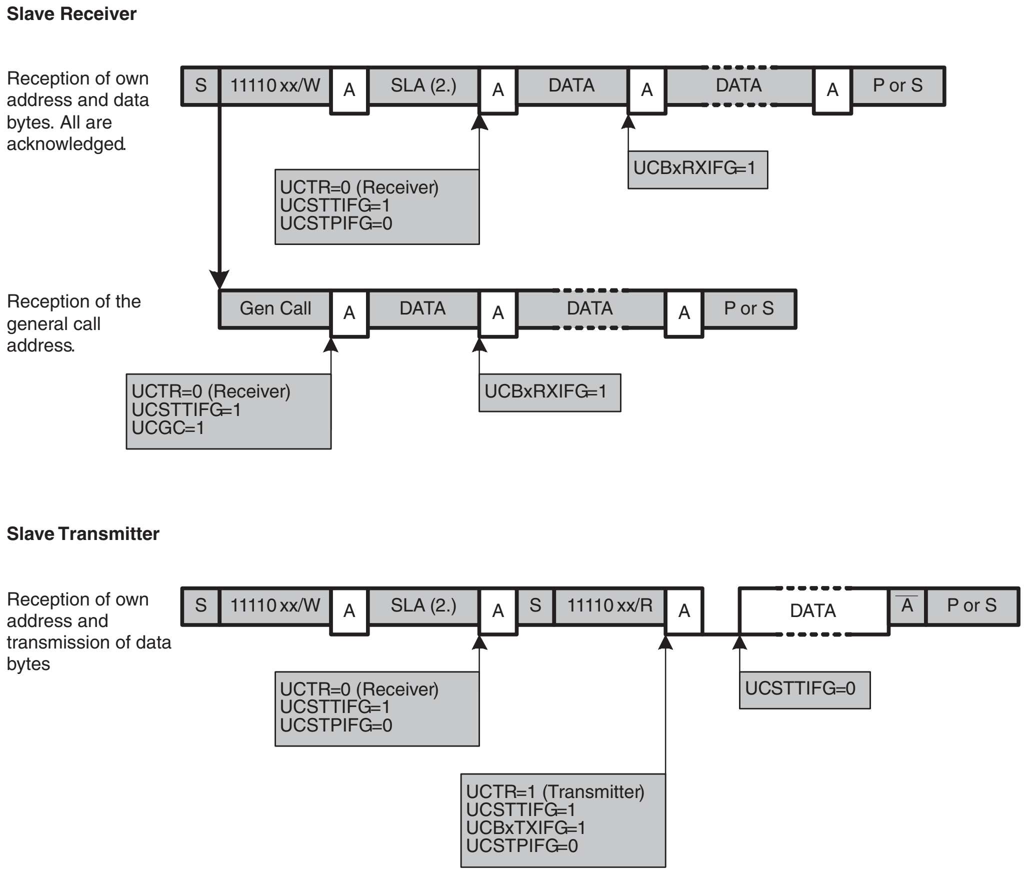

Reception of own address and data bytes. All are acknowledged.
S
SLA/W
A
DATA
A
DATA
A
DATA
A
P or S
UCTR=0 (Receiver)
UCSTTIFG=1
UCSTPIFG=0
UCBxRXIFG=1
Bus stalled (SCL held low) if UCBxRXBUF not read
Read data from UCBxRXBUF
Refer to: "Slave Transmitter" Timing Diagram
Last byte is not acknowledged.
DATA
A
P or S
UCTXNACK=1
UCTXNACK=0
Bus not stalled even if UCBxRXBUF not read
Reception of the general call address.
Gen Call
A
UCTR=0 (Receiver)
UCSTTIFG=1
UCGC=1
Arbitration lost as master and addressed as slave
A
UCALIFG=1
UCMST=0
UCTR=0 (Receiver)
UCSTTIFG=1
(UCGC=1 if general call)
UCBxTXIFG=0
UCSTPIFG=0

A 5xx: Replace UCBxTXIFG with UCTXIFG. Replace UCBxRXIFG with UCRXIFG.

Figure 17-10. I^{2}C Slave Receiver Mode

I^{2}C Slave 10-bit Addressing Mode

The 10-bit addressing mode is selected when UCA10 = 1 and is as shown in Figure 17-11. 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/$\overline{W}$ 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.



A 5xx: Replace UCBxTXIFG with UCTXIFG. Replace UCBxRXIFG with UCRXIFG.

Figure 17-11. I^{2}C Slave 10-bit Addressing Mode

Master Mode

The USCI module is configured as an I^{2}C master by selecting the I^{2}C 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^{2}C 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 UCTXIFG 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.

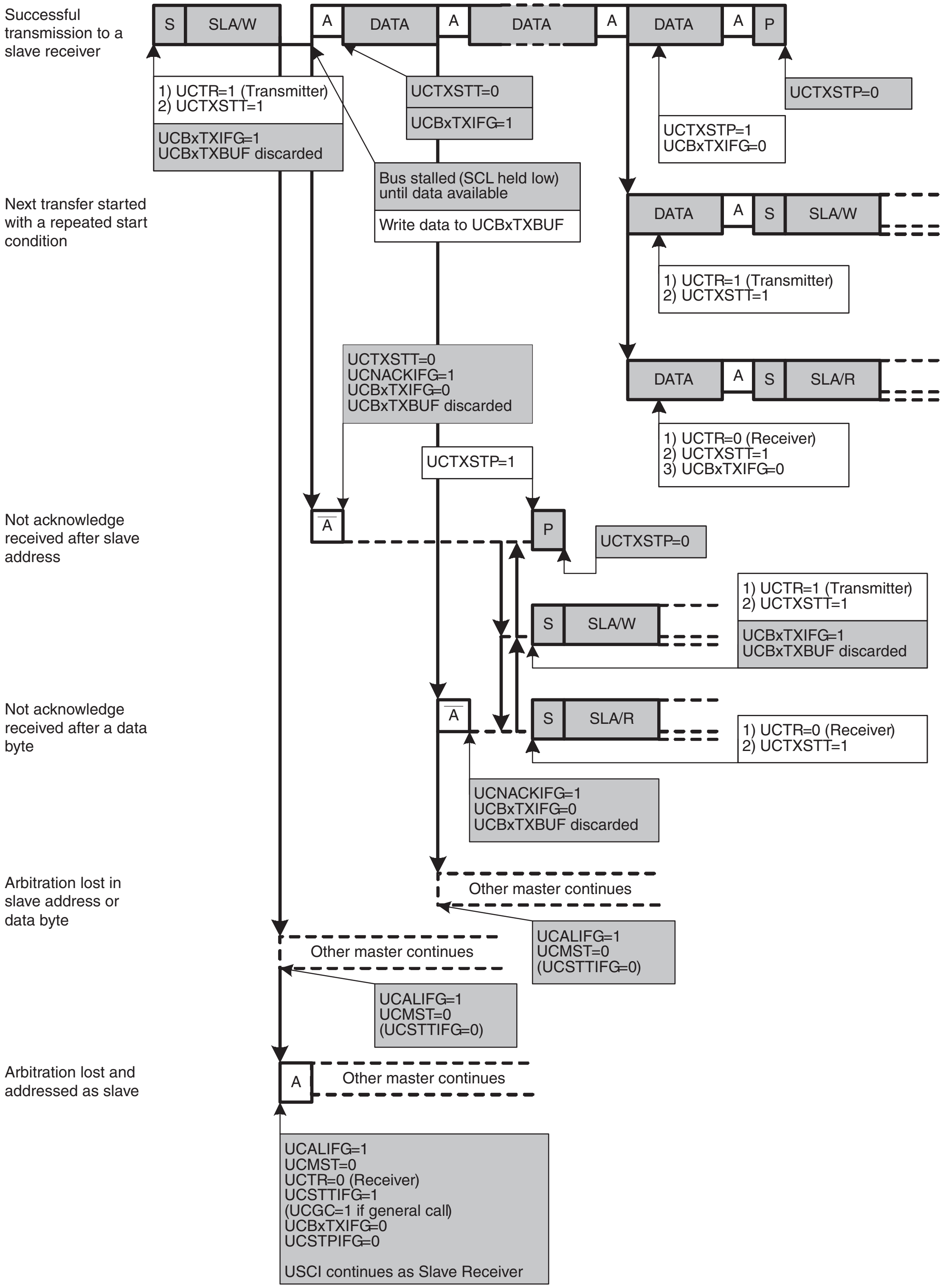
The data written into UCBxTXBUF is transmitted if arbitration is not lost during transmission of the slave address. UCTXIFG is set again as soon as the data is transferred from the buffer into the shift register. If there is no data loaded to UCBxTXBUF before the acknowledge cycle, the bus is held during 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, UCTXIFG 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. Any set UCTXSTT is discarded, too. To trigger a repeated start, UCTXSTT needs to be set again.

Figure 17-12 illustrates the I^{2}C master transmitter operation.



A 5xx: Replace UCBxTXIFG with UCTXIFG. Replace UCBxRXIFG with UCRXIFG.

Figure 17-12. I^2C Master Transmitter Mode

I^{2}C 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 UCRXIFG 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:

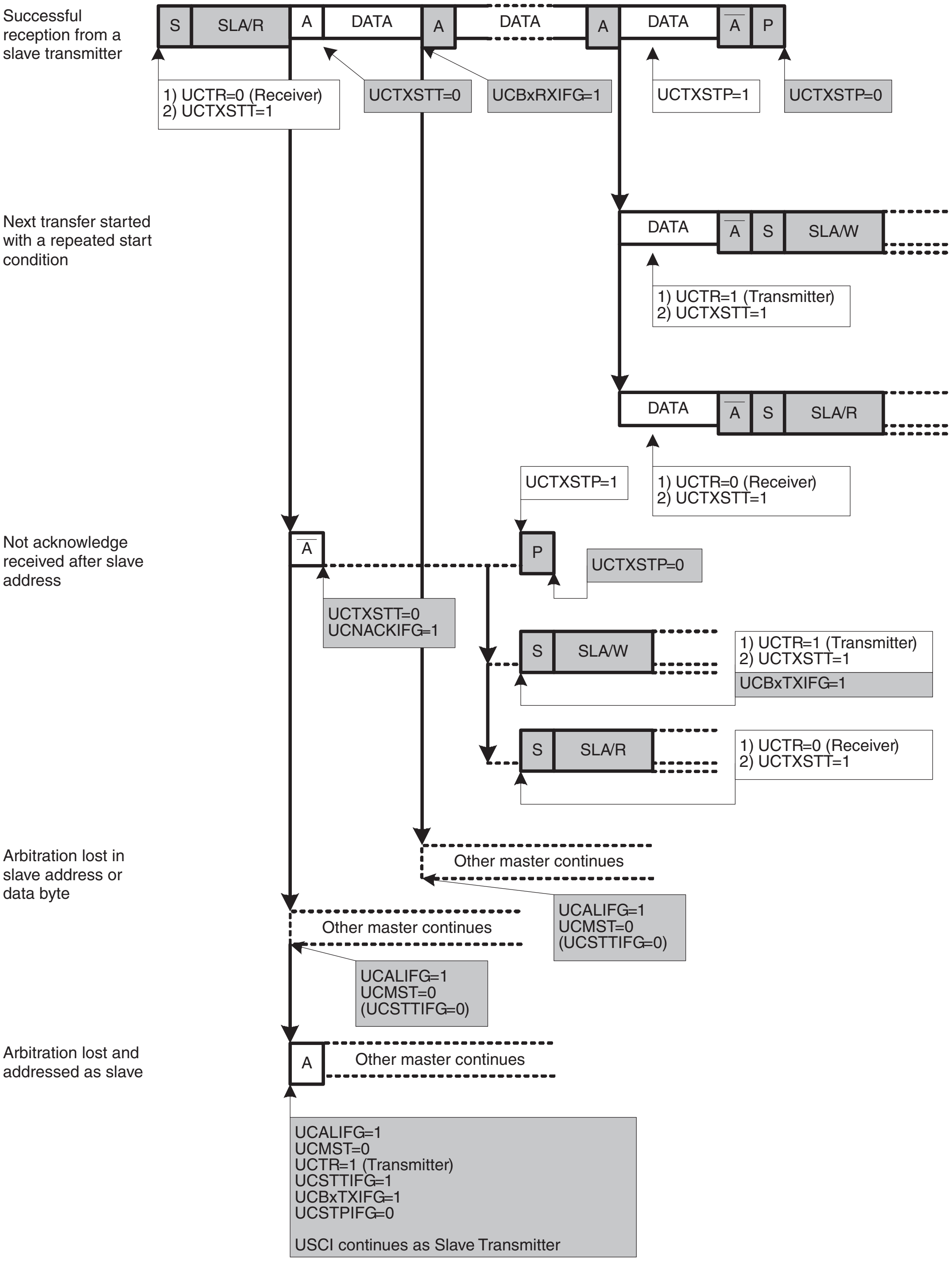
```
           BIS.B   #UCTXSTT, &UCBOCTL1  ;Transmit START cond.
POLL_STT   BIT.B   #UCTXSTT, &UCBOCTL1  ;Poll UCTXSTT bit
           JC      POLL_STT             ;When cleared,
           BIS.B   #UCTXSTP, &UCB0CTL1  ;transmit STOP cond.
```

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 17-13 illustrates the I^{2}C master receiver operation.

Note: Consecutive Master Transactions Without Repeated Start

When performing multiple consecutive I^{2}C master transactions without the repeated start feature, the current transaction must be completed before the next one is initiated. This can be done by ensuring that the transmit stop condition flag UCTXSTP is cleared before the next I^{2}C transaction is initiated with setting UCTXSTT = 1. Otherwise, the current transaction might be affected.

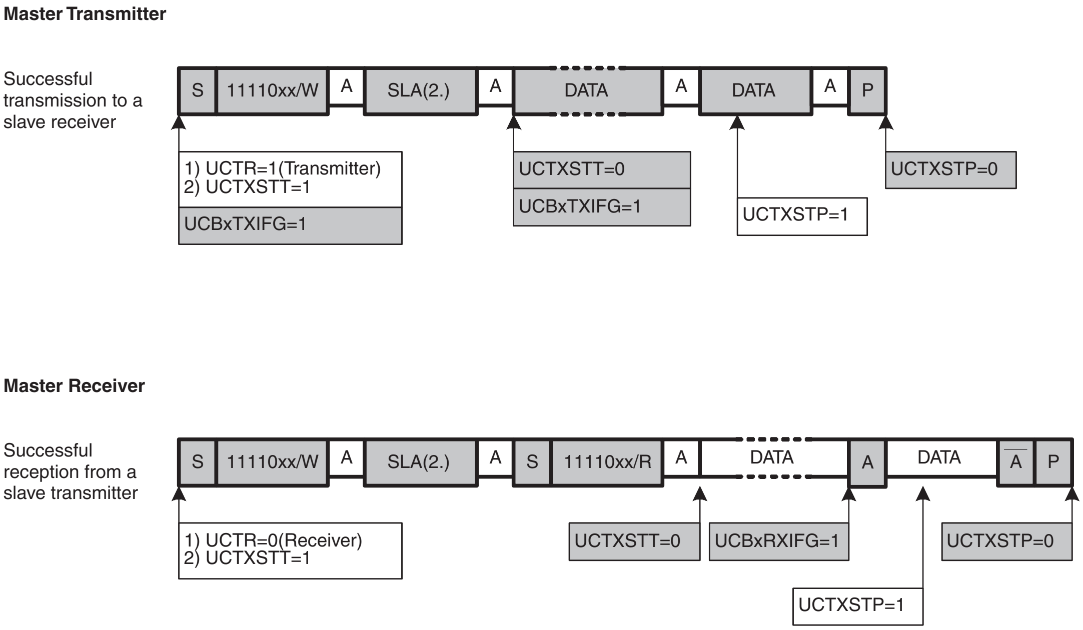


A 5xx: Replace UCBxTXIFG with UCTXIFG. Replace UCBxRXIFG with UCRXIFG.

Figure 17-13. I^{2}C Master Receiver Mode

I^{2}C Master 10-bit Addressing Mode

The 10-bit addressing mode is selected when UCSLA10 = 1 and is shown in Figure 17-14.



A 5xx: Replace UCBxTXIFG with UCTXIFG. Replace UCBxRXIFG with UCRXIFG.

Figure 17-14. I^{2}C 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 17-15 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.

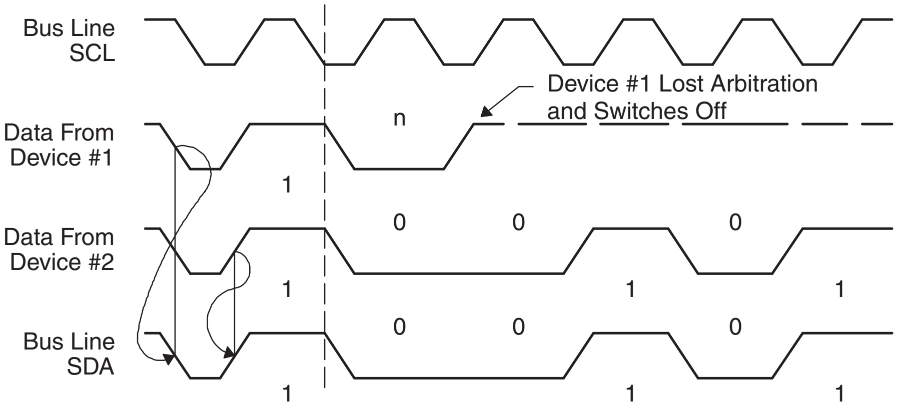


Figure 17-15. Arbitration Procedure Between Two Master Transmitters

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

17.3.5 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:

$f_{BitClock} = f_{BRCLK}/UCBRx$

The minimum high and low periods of the generated SCL are:

$t_{LOW,MIN} = t_{HIGH,MIN} = (UCBRx/2)/f_{BRCLK}$ when UCBRx is even

$t_{LOW,MIN} = t_{HIGH,MIN} = (UCBRx – 1/2)/f_{BRCLK}$ 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 17-16 illustrates the clock synchronization. This allows a slow slave to slow down a fast master.

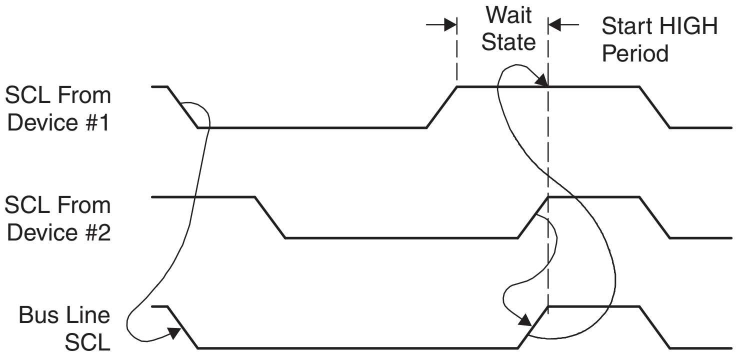


Figure 17-16. Synchronization of Two I²C Clock Generators During Arbitration

Clock Stretching

The USCI module supports clock stretching and also makes use of this feature as described in the operation mode sections.

The UCSCLLOW bit can be used to observe if another device pulls SCL low while the USCI module already released SCL due to the following conditions:

- USCI is acting as master and a connected slave drives SCL low.
- USCI is acting as master and another master drives SCL low during arbitration.

The UCSCLLOW bit is also active if the USCI holds SCL low because it is waiting as transmitter for data being written into UCBxTXBUF or as receiver for the data being read from UCBxRXBUF.

The UCSCLLOW bit might get set for a short time with each rising SCL edge because the logic observes the external SCL and compares it to the internally generated SCL.

17.3.6 Using the USCI Module in I^2C Mode with Low Power Modes

The USCI module provides automatic clock activation for use with low-power modes. When the USCI clock source 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.

In I^2C 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 I^2C 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.

17.3.7 USCI Interrupts in I^2C Mode

The USCI has only one interrupt vector that is shared for transmission, for reception, and for the state change. USCI_Ax and USC_Bx do not share the same interrupt vector.

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 UCTXIFG and UCRXIFG flags on devices with a DMA controller.

I^2C Transmit Interrupt Operation

The UCTXIFG interrupt flag is set by the transmitter to indicate that UCBxTXBUF is ready to accept another character. An interrupt request is generated if UCTXIE and GIE are also set. UCTXIFG is automatically reset if a character is written to UCBxTXBUF or if a NACK is received. UCTXIFG is set when UCSWRST = 1 and the I^2C mode is selected. UCTXIE is reset after a PUC or when UCSWRST = 1.

I^2C Receive Interrupt Operation

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

I^2C State Change Interrupt Operation

Table 17-1 describes the I^2C state change interrupt flags.

Table 17-1. I^2C 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 I^2C 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 I^2C 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 I^2C 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.

UCBxIV, Interrupt Vector Generator

The USCI interrupt flags are prioritized and combined to source a single interrupt vector. The interrupt vector register UCBxIV is used to determine which flag requested an interrupt. The highest priority enabled interrupt generates a number in the UCBxIV register that can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled interrupts do not affect the UCBxIV value.

Any access, read or write, of the UCBxIV register automatically resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.

UCBxIV Software Example

The following software example shows the recommended use of UCBxIV. The UCBxIV value is added to the PC to automatically jump to the appropriate routine. The example is given for USCI_B0.

```
USCI_I2C_ISR
         ADD      &UCB0IV, PC  ; Add offset to jump table
         RETI                  ; Vector 0: No interrupt
         JMP      ALIFG_ISR    ; Vector 2: ALIFG
         JMP      NACKIFG_ISR  ; Vector 4: NACKIFG
         JMP      STTIFG_ISR   ; Vector 6: STTIFG
         JMP      STPIFG_ISR   ; Vector 8: STPIFG
         JMP      RXIFG_ISR    ; Vector 10: RXIFG
TXIFG_ISR                      ; Vector 12
         ...                   ; Task starts here
         RETI                  ; Return
ALIFG_ISR                      ; Vector 2
         ...                   ; Task starts here
         RETI                  ; Return
NACKIFG_ISR                    ; Vector 4
         ...                   ; Task starts here
         RETI                  ; Return
STTIFG_ISR                     ; Vector 6
         ...                   ; Task starts here
         RETI                  ; Return
STPIFG_ISR                     ; Vector 8
         ...                   ; Task starts here
         RETI                  ; Return
RXIFG_ISR                      ; Vector 10
         ...                   ; Task starts here
         RETI                  ; Return
```

17.4 USCI Registers: I^{2}C Mode

The USCI registers applicable in I^{2}C mode listed in Table 17-2. The word accessible registers are listed in Table 17-3.

Table 17-2. USCI_Bx Registers

Register	Short Form	Register Type	Address Offset	Initial State
USCI_Bx control register 0	UCBxCTL0	Byte - R/W	+01h	001h with PUC
USCI_Bx control register 1	UCBxCTL1	Byte - R/W	+00h	001h with PUC
USCI_Bx Bit rate control register 0	UCBxBR0	Byte - R/W	+06h	Reset with PUC
USCI_Bx Bit rate control register 1	UCBxBR1	Byte - R/W	+07h	Reset with PUC
USCI_Bx status register	UCBxSTAT	Byte - R/W	+0Ah	Reset with PUC
Reserved - reads zero		Byte - R only	+0Bh	000h
USCI_Bx Receive buffer register	UCBxRXBUF	Byte - R/W	+0Ch	Reset with PUC
Reserved - reads zero		Byte - R only	+0Dh	000h
USCI_Bx Transmit buffer register	UCBxTXBUF	Byte - R/W	+0Eh	Reset with PUC
Reserved - reads zero		Byte - R only	+0Fh	000h
USCI_Bx I2C Own Address register	UCBxI2COA	Word - R/W	+10h	Reset with PUC
USCI_Bx I2C Slave Address register	UCBxI2CSA	Word - R/W	+12h	Reset with PUC
USCI_Bx interrupt enable register	UCBxIE	Byte - R/W	+1Ch	Reset with PUC
USCI_Bx interrupt flag register	UCBxIFG	Byte - R/W	+1Dh	002h with PUC
USCI_Bx interrupt vector register	UCBxIV	Word - R	+1Eh	Reset with PUC

Table 17-3. Word Access to USCI_Bx Registers

Word Register	Short Form	High-Byte Register	Low-Byte Register	Address Offset
USCI_Bx control word register 0	UCBxCTLW0	UCBxCTL0	UCBxCTL1	+00h
USCI_Bx bit rate control word register	UCBxBRW	UCBxBR1	UCBxBR0	+06h
USCI_Bx interrupt control register	UCBxICTL	UCBxIFG	UCBxIE	+1Ch

UCBxCTL0, USCI_Bx Control Register 0

7	6	5	4	3	2	1	0
UCA10	**UCSLA10**	**UCMM**	**Unused**	**UCMST**	**UCMODEx=11**		**UCSYNC=1**
R/W-0							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-1

UCA10 Bit 7 Own addressing mode select

0 Own address is a 7-bit address

1 Own address is a 10-bit address

UCSLA10 Bit 6 Slave addressing mode select

0 Address slave with 7-bit address

1 Address slave with 10-bit address

UCMM Bit 5 Multi-master environment select

0 Single master environment. There is no other master in the system. The address compare unit is disabled.

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.

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 (master/slave enabled if STE = 1)

10 4-pin SPI (master/slave enabled if STE = 0)

11 I^{2}C mode

UCSYNC Bit 0 Synchronous mode enable

0 Asynchronous mode

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	r0	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.

00 UCLKI

01 ACLK

10 SMCLK

11 SMCLK

Unused Bit 5 Unused

UCTR Bit 4 Transmitter/Receiver

0 Receiver

1 Transmitter

UCTXNACK Bit 3 Transmit a NACK. UCTXNACK is automatically cleared after a NACK is transmitted.

0 Acknowledge normally

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.

0 No STOP generated

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.Ignored in slave mode.

0 Do not generate START condition

1 Generate START condition

UCSWRST Bit 0 Software reset enable

0 Disabled. USCI reset released for operation.

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} forms the prescaler value.

UCBxSTAT, USCI_Bx Status Register

7	6	5	4	3	2	1	0
Unused	**UCSCLLOW**	**UCGC**	**UCBBUSY**	**Unused**			
rw-0	r-0	rw-0	r-0	r0	r0	r0	r0

Unused Bit 7 Unused

UCSCLLOW Bit 6 SCL low

0 SCL is not held low

1 SCL is held low

UCGC Bit 5 General call address received. UCGC is automatically cleared when a START condition is received.

0 No general call address received

1 General call address received

UCBBUSY Bit 4 Bus busy

0 Bus inactive

1 Bus busy

Unused Bits 3-0 Unused

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 UCRXIFG.

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 UCTXIFG.

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

0 Do not respond to a general call

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 and 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.

UCBxIE, USCI_Bx I^2C Interrupt Enable Register

7	6	5	4	3	2	1	0
Reserved		**UCNACKIE**	**UCALIE**	**UCSTPIE**	**UCSTTIE**	**UCTXIE**	**UCRXIE**
r-0	r-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Reserved Bits 7-6 Reserved

UCNACKIE Bit 5 Not-acknowledge interrupt enable

0 Interrupt disabled

1 Interrupt enabled

UCALIE Bit 4 Arbitration lost interrupt enable

0 Interrupt disabled

1 Interrupt enabled

UCSTPIE Bit 3 Stop condition interrupt enable

0 Interrupt disabled

1 Interrupt enabled

UCSTTIE Bit 2 Start condition interrupt enable

0 Interrupt disabled

1 Interrupt enabled

UCTXIE Bit 1 Transmit interrupt enable

0 Interrupt disabled

1 Interrupt enabled

UCRXIE Bit 0 Receive interrupt enable

0 Interrupt disabled

1 Interrupt enabled

UCBxIFG, USCI_Bx I^2C Interrupt Flag Register

7	6	5	4	3	2	1	0
Reserved		**UCNACKIFG**	**UCALIFG**	**UCSTPIFG**	**UCSTTIFG**	**UCTXIFG**	**UCRXIFG**
r-0	r-0	rw-0	rw-0	rw-0	rw-0	rw-1	rw-0

Reserved Bits 7-6 Reserved

UCNACKIFG Bit 5 Not-acknowledge received interrupt flag. UCNACKIFG is automatically cleared when a START condition is received.

0 No interrupt pending

1 Interrupt pending

UCALIFG Bit 4 Arbitration lost interrupt flag

0 No interrupt pending

1 Interrupt pending

UCSTPIFG Bit 3 Stop condition interrupt flag. UCSTPIFG is automatically cleared when a START condition is received.

0 No interrupt pending

1 Interrupt pending

UCSTTIFG Bit 2 Start condition interrupt flag. UCSTTIFG is automatically cleared if a STOP condition is received.

0 No interrupt pending

1 Interrupt pending

UCTXIFG Bit 1 USCI transmit interrupt flag. UCTXIFG is set when UCBxTXBUF is empty.

0 No interrupt pending

1 Interrupt pending

UCRXIFG Bit 0 USCI receive interrupt flag. UCRXIFG is set when UCBxRXBUF has received a complete character.

0 No interrupt pending

1 Interrupt pending

UCBxIV, USCI_Bx 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**	**UCIVx**			**0**
r0	r0	r0	r0	r-0	r-0	r-0	r0

UCIVx Bits 15-0 USCI interrupt vector value

UCBxIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No interrupt pending	–	
002h	Arbitration lost	UCALIFG	Highest
004h	Not acknowledgement	UCNACKIFG	
006h	Start condition received	UCSTTIFG	
008h	Stop condition received	UCSTPIFG	
00Ah	Data received	UCRXIFG	
00Ch	Transmit buffer empty	UCTXIFG	Lowest

Chapter 18

SLAU208–June 2008

ADC12_A

The ADC12_A module is a high-performance 12-bit analog-to-digital converter (ADC). This chapter describes the ADC12_A of the MSP430 5xx devices.

Topic **Page**

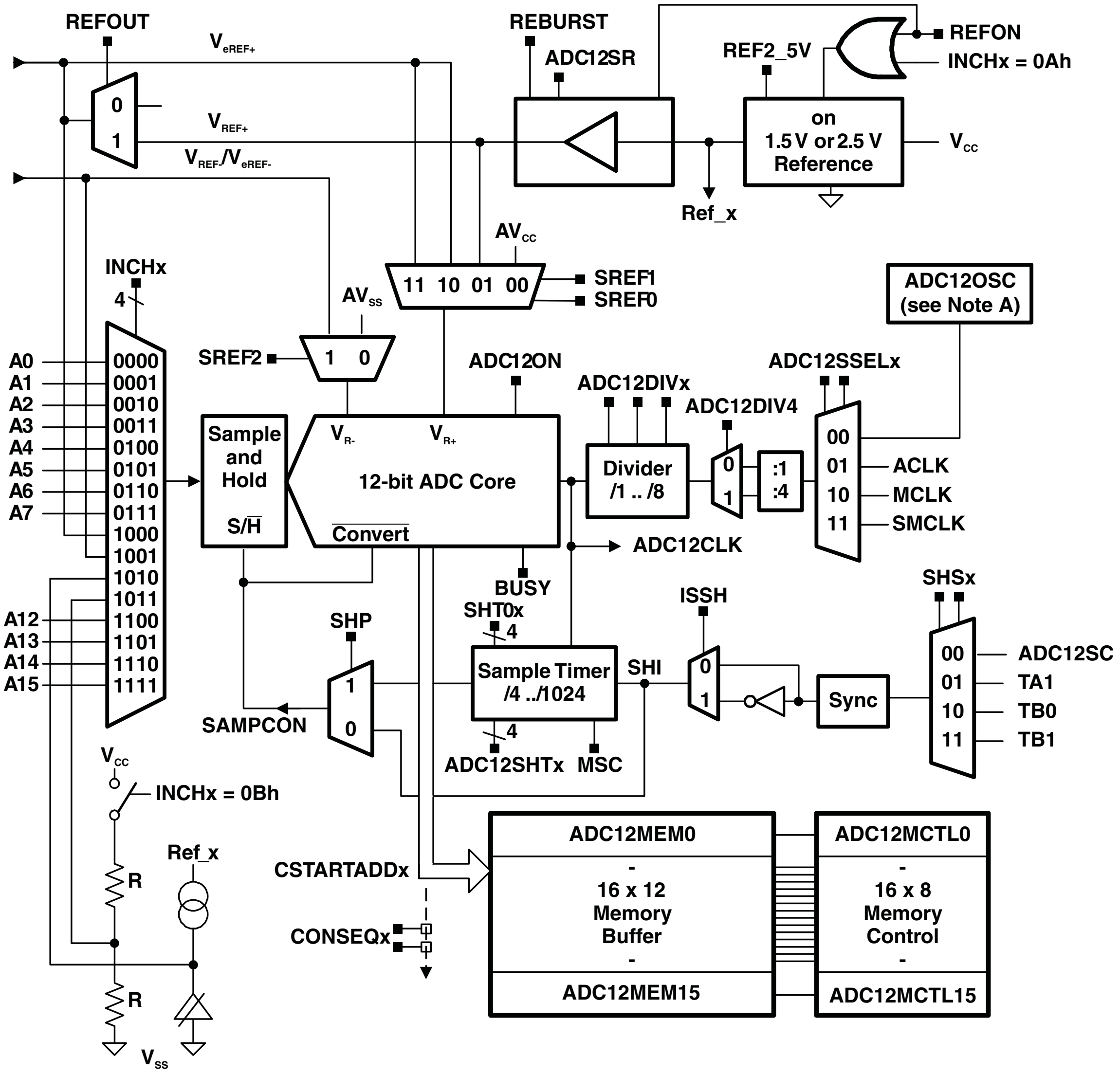
18.1 ADC12_A Introduction

The ADC12_A module supports fast, 12-bit analog-to-digital conversions. The module implements a 12-bit SAR core, sample select control, reference generator (MSP430F54xx only in other devices separate REF module) and a 16 word conversion-and-control buffer. The conversion-and-control buffer allows up to 16 independent ADC samples to be converted and stored without any CPU intervention.

ADC12_A features include:

- Greater than 200-ksps maximum conversion rate
- Monotonic 12-bit converter with no missing codes
- Sample-and-hold with programmable sampling periods controlled by software or timers.
- Conversion initiation by software, Timer_A, or Timer_B
- Software selectable on-chip reference voltage generation (MSP430F54xx: 1.5 V or 2.5 V, other devices: 1.5 V, 2.0 V or 2.5 V)
- Software selectable internal or external reference
- Twelve individually configurable external input channels
- Conversion channels for internal temperature sensor, AV_{CC}, and external references
- Independent channel-selectable reference sources for both positive and negative references
- Selectable conversion clock source
- Single-channel, repeat-single-channel, sequence, and repeat-sequence conversion modes
- ADC core and reference voltage can be powered down separately (MSP430F54xx only. Other devices see REF module specification for details)
- Interrupt vector register for fast decoding of 18 ADC interrupts
- 16 conversion-result storage registers

The block diagram of ADC12_A is shown in Figure 18-1. The reference generation is in the MSP430F54xx devices located in the ADC12_A module. In other devices the reference generator is located in the reference module. See also the device specific datasheet.



A The MODOSC is part of the UCS. See the UCS chapter for more information.

Figure 18-1. ADC12_A Block Diagram

18.2 ADC12_A Operation

The ADC12_A module is configured with user software. The setup and operation of the ADC12_A is discussed in the following sections.

18.2.1 12-Bit ADC Core

The ADC core converts an analog input to its 12-bit digital representation and stores the result in conversion memory. The core uses two programmable/selectable voltage levels (V_{R+} and V_{R-}) to define the upper and lower limits of the conversion. The digital output (N_{ADC}) is full scale (0FFFh) when the input signal is equal to or higher than V_{R+}, and zero when the input signal is equal to or lower than V_{R-}. The input channel and the reference voltage levels (V_{R+} and V_{R-}) are defined in the conversion-control memory. The conversion formula for the ADC result N_{ADC} is:

$$N_{ADC} = 4095 \times \frac{Vin - V_{R-}}{V_{R+} - V_{R-}}$$

The ADC12_A core is configured by two control registers, ADC12CTL0 and ADC12CTL1. The core is enabled with the ADC12ON bit. The ADC12_A can be turned off when not in use to save power. With few exceptions the ADC12_A control bits can only be modified when ADC12ENC = 0. ADC12ENC must be set to 1 before any conversion can take place.

Conversion Clock Selection

The ADC12CLK is used both as the conversion clock and to generate the sampling period when the pulse sampling mode is selected. The ADC12_A source clock is selected using the pre-divider controlled by the ADC12DIV4 bit and the divider using the ADC12SSELx bits. The input clock can be divided from 1-32 using both the ADC12DIVx bits and the ADC12DIV4 bit. Possible ADC12CLK sources are SMCLK, MCLK, ACLK, and the MODOSC.

The ADC12OSC, generated internally, is in the 5-MHz range, but varies with individual devices, supply voltage, and temperature. See the device-specific datasheet for the ADC12OSC specification.

The user must ensure that the clock chosen for ADC12CLK remains active until the end of a conversion. If the clock is removed during a conversion, the operation will not complete and any result will be invalid.

18.2.2 ADC12_A Inputs and Multiplexer

The twelve external and four internal analog signals are selected as the channel for conversion by the analog input multiplexer. The input multiplexer is a break-before-make type to reduce input-to-input noise injection resulting from channel switching as shown in Figure 18-2. The input multiplexer is also a T-switch to minimize the coupling between channels. Channels that are not selected are isolated from the A/D and the intermediate node is connected to analog ground (AV_{SS}) so that the stray capacitance is grounded to help eliminate crosstalk.

The ADC12_A uses the charge redistribution method. When the inputs are internally switched, the switching action may cause transients on the input signal. These transients decay and settle before causing errant conversion.

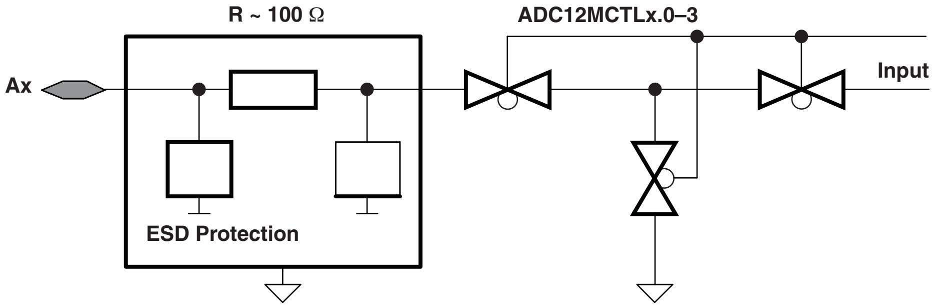


Figure 18-2. Analog Multiplexer

Analog Port Selection

The ADC12_A inputs are multiplexed with digital port pins. When analog signals are applied to digital 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 digital pat of the port pin eliminates the parasitic current flow and therefore reduces overall current consumption. The PySELx bits provide the ability to disable the port pin input and output buffers.

```
; Py.0 and Py.1 configured for analog input
      BIS.B    #3h,&PySEL   ; Py.1 and Py.0 ADC12_A function
```

18.2.3 Voltage Reference Generator

The ADC12_A module of the MSP430F54xx contains a built-in voltage reference with two selectable voltage levels, 1.5 V and 2.5 V. Either of these reference voltages may be used internally and externally on pin V_{REF+}.

The ADC12_A modules of other devices have a separate reference module which supplies three selectable voltage levels, 1.5V, 2.0V and 2.5V to the ADC12_A. Either of these voltages may be used internally and externally on pin V_{REF+}.

Setting ADC12REFON = 1 enables the reference voltage of the ADC12_A module. When ADC12REF2_5V = 1, the internal reference is 2.5 V; when ADC12REF2_5V = 0, the reference is 1.5 V . The reference can be turned off to save power when not in use. Devices with the REF module can use the control bits located in the ADC12_A module or the control registers located in the REF module to control the reference voltage supplied to the ADC. Per default the register settings of the REF module define the reference voltage settings. The control bit REFMSTR in the REF module is used to hand over control to the ADC12_A reference control register settings. If the register bit REFMSTR is set to 1 (default) then the REF module registers control the reference settings. If REFMSTR is set to 0 then the ADC12_A reference setting define the reference voltage of the ADC12_A module.

External references may be supplied for V_{R+} and V_{R-} through pins V_{REF+}/Ve_{REF+} and V_{REF-}/Ve_{REF-} respectively.

External storage capacitors are only requied if REFOUT = 1 and the reference voltage is made available at the pins.

Internal Reference Low-Power Features

The ADC12_A 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 ADC12REFON = 1, both are enabled and if ADC12REFON = 0 both are disabled. The total settling time when ADC12REFON gets set is ≤ 30 µs.

When ADC12REFON = 1 and REFBURST = 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.

The REFBURST bit controls the operation of the reference buffer. When REFBURST = 1, the buffer is automatically disabled when the ADC12_A is not actively converting, and automatically re-enabled when needed. When REFBURST = 0, the buffer will be on continuously this allows the reference voltage to be present outside the device continuously if REFOUT = 1.

The internal reference buffer also has selectable speed vs. power settings. When the maximum conversion rate is below 50 ksps, setting ADC12SR = 1 reduces the current consumption of the buffer approximately 50%.

18.2.4 Auto Power-Down

The ADC12_A is designed for low power applications. When the ADC12_A is not actively converting, the core is automatically disabled and automatically re-enabled when needed The MODOSC is also automatically enabled when needed and disabled when not needed.

18.2.5 Sample and Conversion Timing

An analog-to-digital conversion is initiated with a rising edge of the sample input signal SHI. The source for SHI is selected with the SHSx bits and includes the following:

- The ADC12SC bit
- The Timer_A Output Unit 1
- The Timer_B Output Unit 0
- The Timer_B Output Unit 1

The polarity of the SHI signal source can be inverted with the ADC12ISSH bit. The SAMPCON signal controls the sample period and start of conversion. When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the analog-to-digital conversion, which requires 13 ADC12CLK cycles in 12-bit resolution mode. Two different sample-timing methods are defined by control bit ADC12SHP, extended sample mode and pulse mode.

Extended Sample Mode

The extended sample mode is selected when ADC12SHP = 0. The SHI signal directly controls SAMPCON and defines the length of the sample period $t_{sample.}$ When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the conversion after synchronization with ADC12CLK (see Figure 18-3).

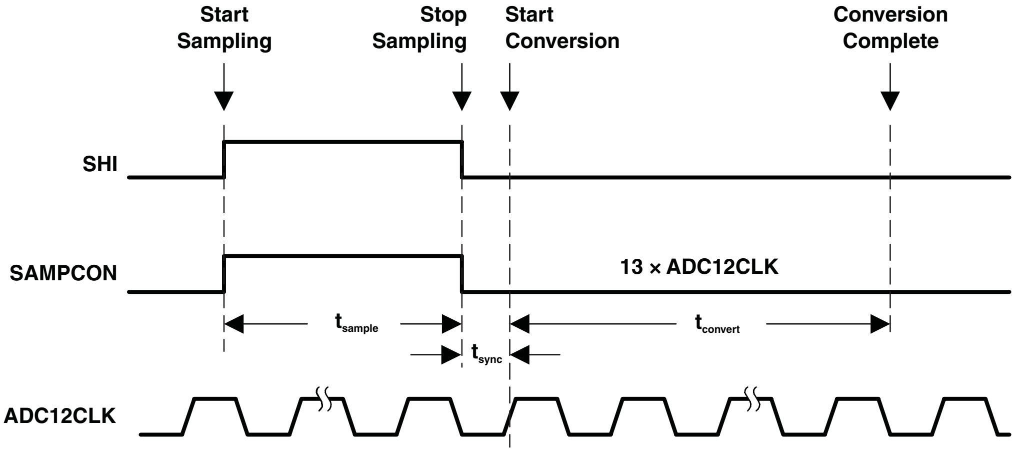


Figure 18-3. Extended Sample Mode

Pulse Sample Mode

The pulse sample mode is selected when ADC12SHP = 1. The SHI signal is used to trigger the sampling timer. The ADC12SHT0x and ADC12SHT1x bits in ADC12CTL0 control the interval of the sampling timer that defines the SAMPCON sample period $t_{sample.}$ The sampling timer keeps SAMPCON high after synchronization with AD12CLK for a programmed interval t_{sample}. The total sampling time is t_{sample} plus t_{sync} (see Figure 18-4).

The ADC12SHTx bits select the sampling time in 4× multiples of ADC12CLK. ADC12SHT0x selects the sampling time for ADC12MCTL0 to 7, and ADC12SHT1x selects the sampling time for ADC12MCTL8 to 15.

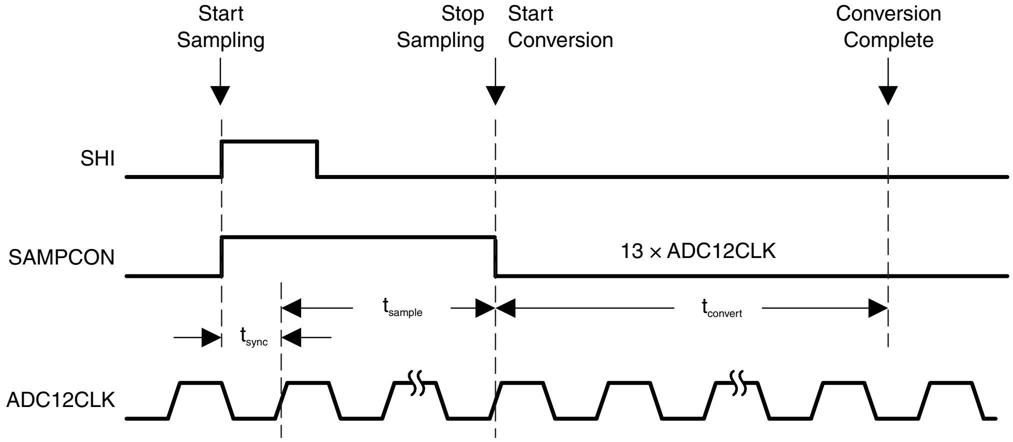


Figure 18-4. Pulse Sample Mode

Sample Timing Considerations

When SAMPCON = 0 all Ax inputs are high impedance. When SAMPCON = 1, the selected Ax input can be modeled as an RC low-pass filter during the sampling time t_{sample}, as shown below in Figure 18-5. An internal MUX-on input resistance R_I (maximum 2 kΩ) in series with capacitor C_I (40 pF maximum) is seen by the source. The capacitor C_I voltage V_C must be charged to within 1/2 LSB of the source voltage V_S for an accurate 12-bit conversion.

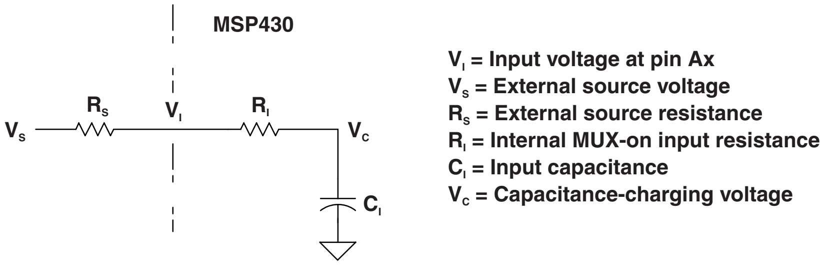

MSP430
R_S
V_I
R_I
V_S
V_C
C_I
V_I = Input voltage at pin Ax
V_S = External source voltage
R_S = External source resistance
R_I = Internal MUX-on input resistance
C_I = Input capacitance
V_C = Capacitance-charging voltage

Figure 18-5. Analog Input Equivalent Circuit

The resistance of the source R_S and R_I affect t_{sample}. The following equation can be used to calculate the minimum sampling time t_{sample} for a 12-bit conversion:

$$t_{sample} > (R_S + R_I) \times \ln(2^{13}) \times C_I + 800\text{ns}$$

Substituting the values for R_I and C_I given above, the equation becomes:

$$t_{sample} > (R_S + 2\text{k}\Omega) \times 9.011 \times 40\text{pF} + 800\text{ns}$$

For example, if R_S is 10 kΩ, t_{sample} must be greater than 5.13 µs.

18.2.6 Conversion Memory

There are 16 ADC12MEMx conversion memory registers to store conversion results. Each ADC12MEMx is configured with an associated ADC12MCTLx control register. The SREFx bits define the voltage reference and the INCHx bits select the input channel. The ADC12EOS bit defines the end of sequence when a sequential conversion mode is used. A sequence rolls over from ADC12MEM15 to ADC12MEM0 when the ADC12EOS bit in ADC12MCTL15 is not set.

The CSTARTADDx bits define the first ADC12MCTLx used for any conversion. If the conversion mode is single-channel or repeat-single-channel the CSTARTADDx points to the single ADC12MCTLx to be used.

If the conversion mode selected is either sequence-of-channels or repeat-sequence-of-channels, CSTARTADDx points to the first ADC12MCTLx location to be used in a sequence. A pointer, not visible to software, is incremented automatically to the next ADC12MCTLx in a sequence when each conversion completes. The sequence continues until an ADC12EOS bit in ADC12MCTLx is processed - this is the last control byte processed.

When conversion results are written to a selected ADC12MEMx, the corresponding flag in the ADC12IFGx register is set.

18.2.7 ADC12_A Conversion Modes

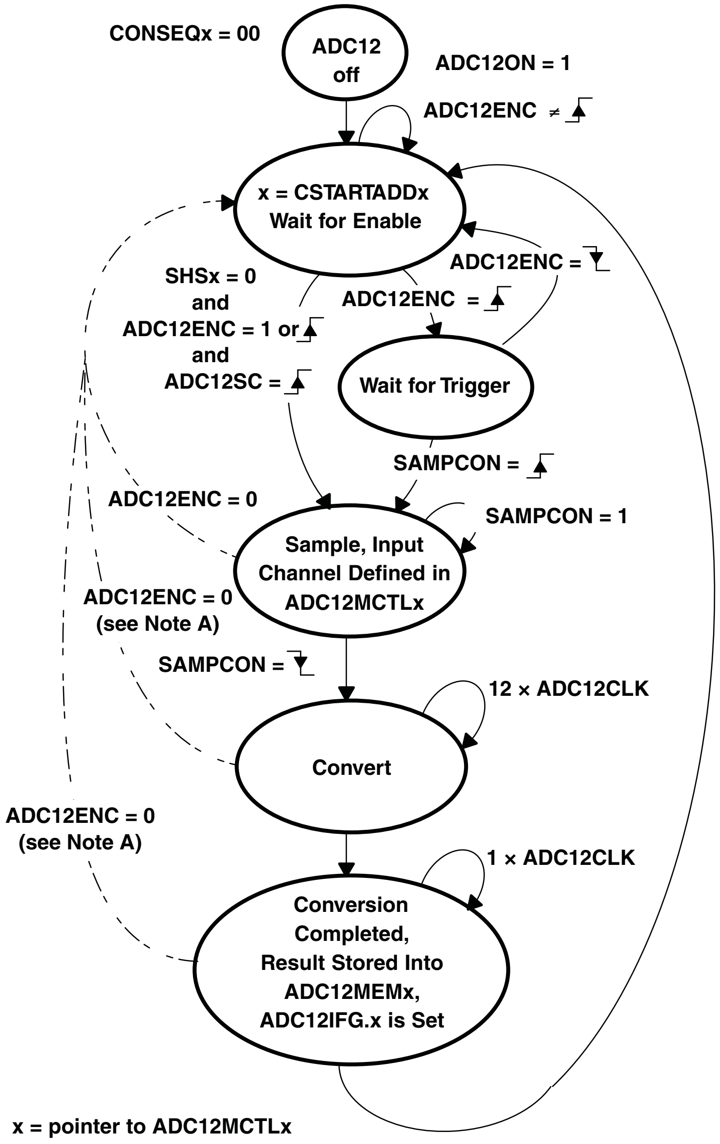
The ADC12_A has four operating modes selected by the CONSEQx bits as discussed in Table 18-1.

Table 18-1. Conversion Mode Summary

ADC12CONSEQx	Mode	Operation
00	Single channel single-conversion	A single channel is converted once.
01	Sequence-of-channels	A sequence of channels is converted once.
10	Repeat-single-channel	A single channel is converted repeatedly.
11	Repeat-sequence-of-channels	A sequence of channels is converted repeatedly.

Single-Channel Single-Conversion Mode

A single channel is sampled and converted once. The ADC result is written to the ADC12MEMx defined by the CSTARTADDx bits. Figure 18-6 shows the flow of the Single-Channel, Single-Conversion mode. When ADC12SC triggers a conversion, successive conversions can be triggered by the ADC12SC bit. When any other trigger source is used, ADC12ENC must be toggled between each conversion.

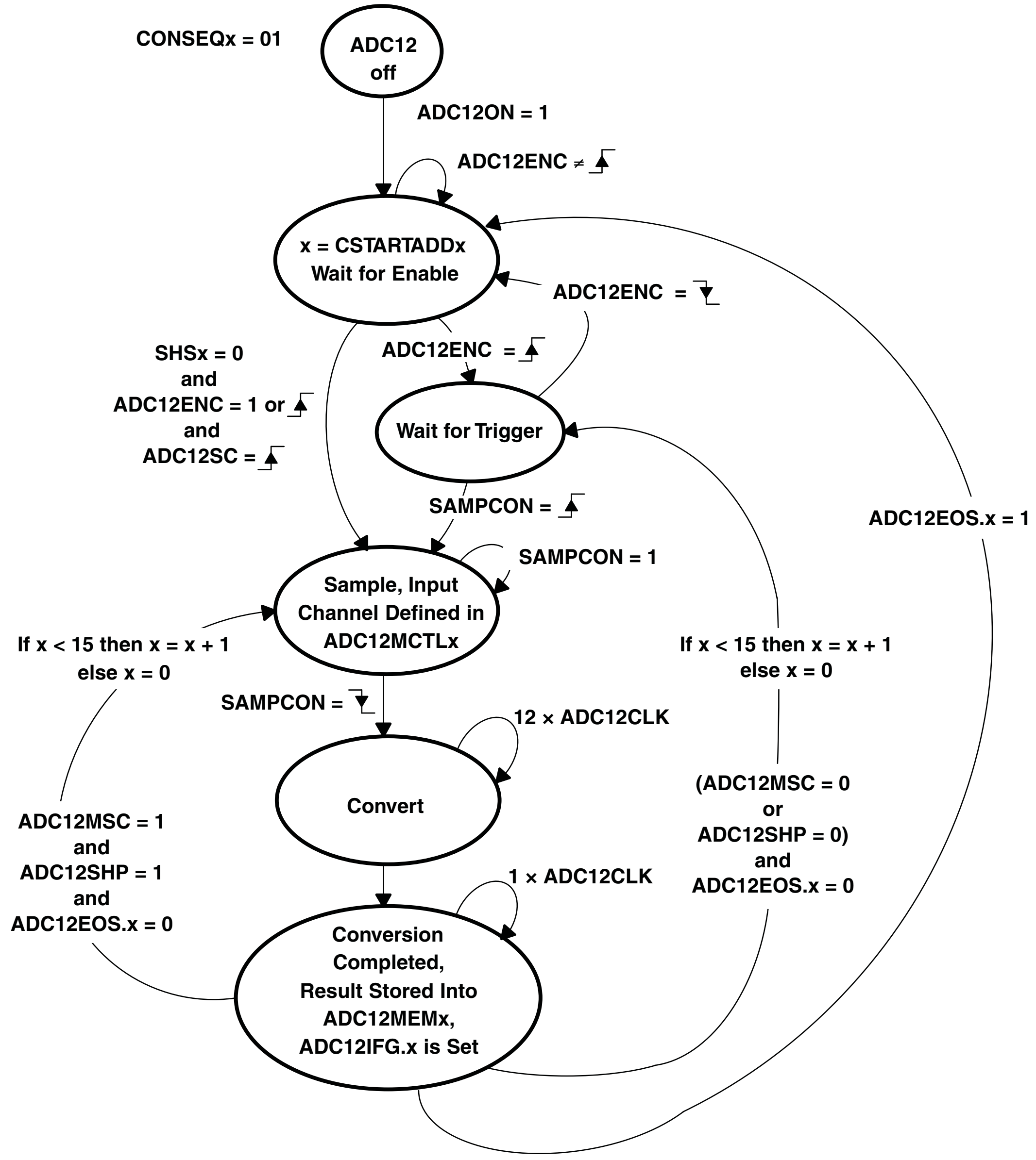

CONSEQx = 00
ADC12 off
ADC12ON = 1
ADC12ENC ≠
x = CSTARTADDx Wait for Enable
ADC12ENC =
SHSx = 0 and ADC12ENC = 1 or and ADC12SC =
ADC12ENC =
Wait for Trigger
SAMPCON =
ADC12ENC = 0
SAMPCON = 1
Sample, Input Channel Defined in ADC12MCTLx
ADC12ENC = 0 (see Note A)
SAMPCON =
12 × ADC12CLK
Convert
ADC12ENC = 0 (see Note A)
1 × ADC12CLK
Conversion Completed, Result Stored Into ADC12MEMx, ADC12IFG.x is Set
x = pointer to ADC12MCTLx

A Conversion result is unpredictable.

Figure 18-6. Single-Channel, Single-Conversion Mode

Sequence-of-Channels Mode

A sequence of channels is sampled and converted once. The ADC results are written to the conversion memories starting with the ADCMEMx defined by the CSTARTADDx bits. The sequence stops after the measurement of the channel with a set ADC12EOS bit. Figure 18-7 shows the sequence-of-channels mode. When ADC12SC triggers a sequence, successive sequences can be triggered by the ADC12SC bit. When any other trigger source is used, ADC12ENC must be toggled between each sequence.


CONSEQx = 01
ADC12 off
ADC12ON = 1
ADC12ENC ≠ ↑
x = CSTARTADDx Wait for Enable
ADC12ENC = ↓
SHSx = 0 and ADC12ENC = 1 or ↑ and ADC12SC = ↑
ADC12ENC = ↑
Wait for Trigger
SAMPCON = ↑
ADC12EOS.x = 1
SAMPCON = 1
Sample, Input Channel Defined in ADC12MCTLx
If x < 15 then x = x + 1 else x = 0
If x < 15 then x = x + 1 else x = 0
SAMPCON = ↓
12 × ADC12CLK
Convert
ADC12MSC = 1 and ADC12SHP = 1 and ADC12EOS.x = 0
(ADC12MSC = 0 or ADC12SHP = 0) and ADC12EOS.x = 0
1 × ADC12CLK
Conversion Completed, Result Stored Into ADC12MEMx, ADC12IFG.x is Set

x = pointer to ADC12MCTLx

Figure 18-7. Sequence-of-Channels Mode

Repeat-Single-Channel Mode

A single channel is sampled and converted continuously. The ADC results are written to the ADC12MEMx defined by the CSTARTADDx bits. It is necessary to read the result after the completed conversion because only one ADC12MEMx memory is used and is overwritten by the next conversion. Figure 18-8 shows repeat-single-channel mode

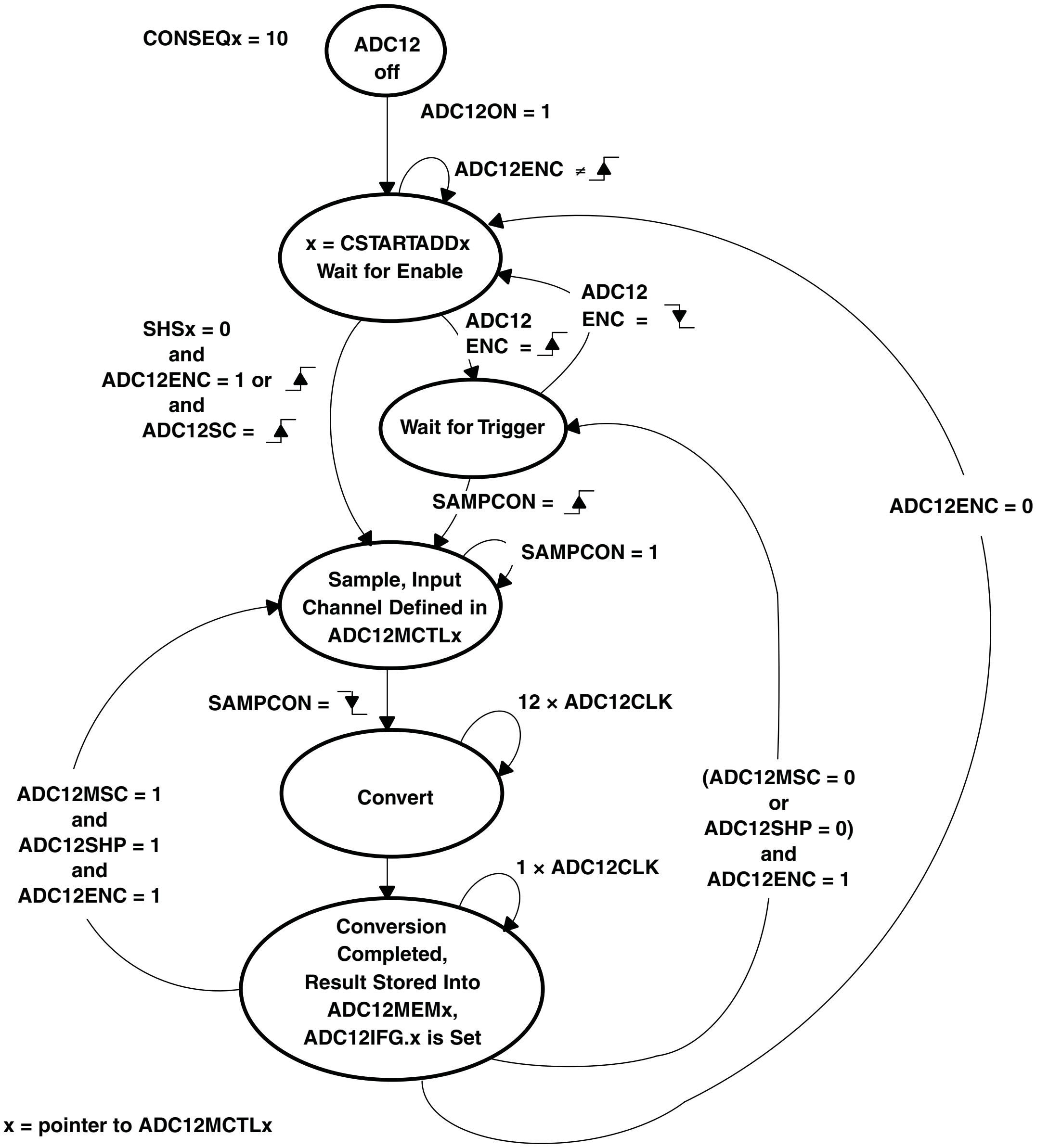

CONSEQx = 10
ADC12 off
ADC12ON = 1
ADC12ENC ≠
x = CSTARTADDx Wait for Enable
ADC12 ENC =
ADC12 ENC =
SHSx = 0 and ADC12ENC = 1 or and ADC12SC =
Wait for Trigger
SAMPCON =
ADC12ENC = 0
SAMPCON = 1
Sample, Input Channel Defined in ADC12MCTLx
SAMPCON =
12 × ADC12CLK
ADC12MSC = 1 and ADC12SHP = 1 and ADC12ENC = 1
Convert
(ADC12MSC = 0 or ADC12SHP = 0) and ADC12ENC = 1
1 × ADC12CLK
Conversion Completed, Result Stored Into ADC12MEMx, ADC12IFG.x is Set
x = pointer to ADC12MCTLx

Figure 18-8. Repeat-Single-Channel Mode

Repeat-Sequence-of-Channels Mode

A sequence of channels is sampled and converted repeatedly. The ADC results are written to the conversion memories starting with the ADC12MEMx defined by the CSTARTADDx bits. The sequence ends after the measurement of the channel with a set ADC12EOS bit and the next trigger signal re-starts the sequence. Figure 18-9 shows the repeat-sequence-of-channels mode.

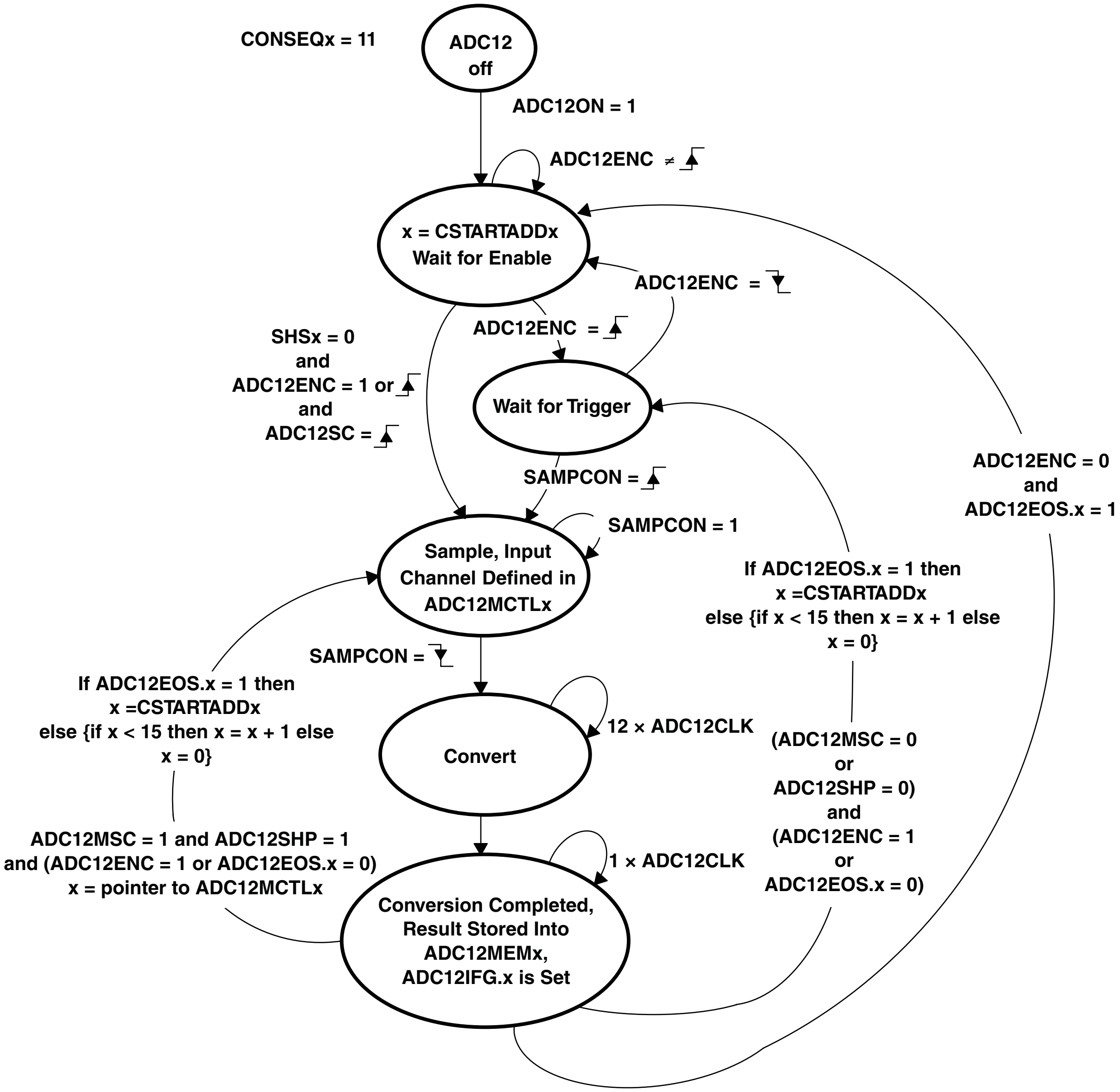

CONSEQx = 11
ADC12 off
ADC12ON = 1
ADC12ENC ≠
x = CSTARTADDx Wait for Enable
ADC12ENC =
ADC12ENC =
SHSx = 0 and ADC12ENC = 1 or and ADC12SC =
Wait for Trigger
SAMPCON =
ADC12ENC = 0 and ADC12EOS.x = 1
SAMPCON = 1
Sample, Input Channel Defined in ADC12MCTLx
If ADC12EOS.x = 1 then x =CSTARTADDx else {if x < 15 then x = x + 1 else x = 0}
SAMPCON =
If ADC12EOS.x = 1 then x =CSTARTADDx else {if x < 15 then x = x + 1 else x = 0}
Convert
12 × ADC12CLK
(ADC12MSC = 0 or ADC12SHP = 0) and (ADC12ENC = 1 or ADC12EOS.x = 0)
ADC12MSC = 1 and ADC12SHP = 1 and (ADC12ENC = 1 or ADC12EOS.x = 0) x = pointer to ADC12MCTLx
1 × ADC12CLK
Conversion Completed, Result Stored Into ADC12MEMx, ADC12IFG.x is Set

Figure 18-9. Repeat-Sequence-of-Channels Mode

Using the Multiple Sample and Convert (ADC12MSC) Bit

To configure the converter to perform successive conversions automatically and as quickly as possible, a multiple sample and convert function is available. When ADC12MSC = 1, CONSEQx > 0, and the sample timer is used, the first rising edge of the SHI signal triggers the first conversion. Successive conversions are triggered automatically as soon as the prior conversion is completed. Additional rising edges on SHI are ignored until the sequence is completed in the single-sequence mode or until the ADC12ENC bit is toggled in repeat-single-channel, or repeated-sequence modes. The function of the ADC12ENC bit is unchanged when using the ADC12MSC bit.

Stopping Conversions

Stopping ADC12_A activity depends on the mode of operation. The recommended ways to stop an active conversion or conversion sequence are:

- Resetting ADC12ENC in single-channel single-conversion mode stops a conversion immediately and the results are unpredictable. For correct results, poll the busy bit until reset before clearing ADC12ENC.
- Resetting ADC12ENC during repeat-single-channel operation stops the converter at the end of the current conversion.
- Resetting ADC12ENC 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 ADC12ENC bit. Conversion data are unreliable.

Note: No ADC12EOS Bit Set For Sequence

If no ADC12EOS bit is set and a sequence mode is selected, resetting the ADC12ENC bit does not stop the sequence. To stop the sequence, first select a single-channel mode and then reset ADC12ENC.

18.2.8 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 temperature sensor is in the ADC12_A in the MSP430F54xx devices while it is part of the REF module in other devices.

The typical temperature sensor transfer function is shown in Figure 18-10. When using the temperature sensor, the sample period must be greater than 30 = s. The temperature sensor offset error can be large, and may need to be calibrated for most applications. See device-specific datasheet for parameters.

Selecting the temperature sensor automatically turns on the on-chip reference generator as a voltage source for the temperature sensor. However, it does not enable the V_{REF+} output or affect the reference selections for the conversion. The reference choices for converting the temperature sensor are the same as with any other channel.

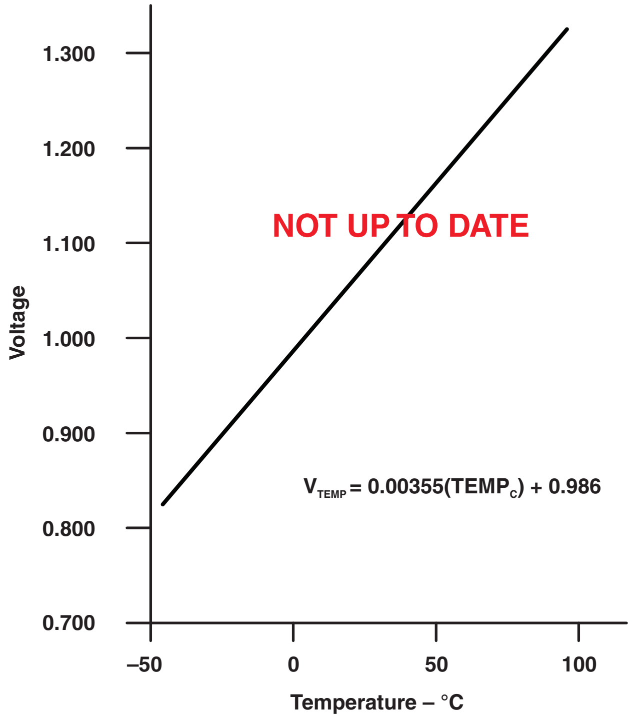


Figure 18-10. Typical Temperature Sensor Transfer Function

18.2.9 ADC12_A Grounding and Noise Considerations

As with any high-resolution ADC, appropriate printed-circuit-board layout and grounding techniques should be followed to eliminate ground loops, unwanted parasitic effects, and noise.

Ground loops are formed when return current from the A/D flows through paths that are common with other analog or digital circuitry. If care is not taken, this current can generate small, unwanted offset voltages that can add to or subtract from the reference or input voltages of the A/D converter. The connections shown in Figure 18-11 help avoid this.

In addition to grounding, ripple and noise spikes on the power supply lines due to digital switching or switching power supplies can corrupt the conversion result. A noise-free design using separate analog and digital ground planes with a single-point connection is recommend to achieve high accuracy.

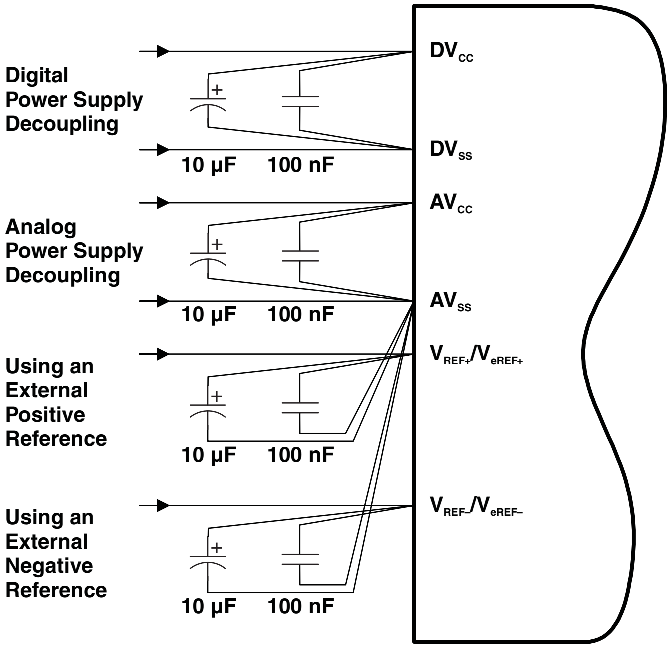

Digital Power Supply Decoupling
10 µF
100 nF
DV$_{CC}$
DV$_{SS}$
Analog Power Supply Decoupling
10 µF
100 nF
AV$_{CC}$
AV$_{SS}$
Using an External Positive Reference
10 µF
100 nF
V$_{REF+}$/V$_{eREF+}$
Using an External Negative Reference
10 µF
100 nF
V$_{REF-}$/V$_{eREF-}$

Figure 18-11. ADC12_A Grounding and Noise Considerations

18.2.10 ADC12_A Interrupts

The ADC12_A has 18 interrupt sources:

- ADC12IFG0-ADC12IFG15
- ADC12OV, ADC12MEMx overflow
- ADC12TOV, ADC12_A conversion time overflow

The ADC12IFGx bits are set when their corresponding ADC12MEMx memory register is loaded with a conversion result. An interrupt request is generated if the corresponding ADC12IEx bit and the GIE bit are set. The ADC12OV condition occurs when a conversion result is written to any ADC12MEMx before its previous conversion result was read. The ADC12TOV condition is generated when another sample-and-conversion is requested before the current conversion is completed. The DMA is triggered after the conversion in single channel conversion mode or after the completion of a sequence of channel conversions in sequence of channels conversion mode.

ADC12IV, Interrupt Vector Generator

All ADC12_A interrupt sources are prioritized and combined to source a single interrupt vector. The interrupt vector register ADC12IV is used to determine which enabled ADC12_A interrupt source requested an interrupt.

The highest priority enabled ADC12_A interrupt generates a number in the ADC12IV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled ADC12_A interrupts do not affect the ADC12IV value.

Any access, read or write, of the ADC12IV register automatically resets the ADC12OV condition or the ADC12TOV condition if either was the highest pending interrupt. Neither interrupt condition has an accessible interrupt flag. The ADC12IFGx flags are not reset by an ADC12IV access. ADC12IFGx bits are reset automatically by accessing their associated ADC12MEMx register or may be reset with software.

If another interrupt is pending after servicing of an interrupt, another interrupt is generated. For example, if the ADC12OV and ADC12IFG3 interrupts are pending when the interrupt service routine accesses the ADC12IV register, the ADC12OV interrupt condition is reset automatically. After the RETI instruction of the interrupt service routine is executed, the ADC12IFG3 generates another interrupt.

ADC12_A Interrupt Handling Software Example

The following software example shows the recommended use of ADC12IV and the handling overhead. The ADC12IV value is added to the PC to automatically jump to the appropriate routine.

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

- ADC12IFG0–ADC12IFG14, ADC12TOV, and ADC12OV: 16 cycles
- ADC12IFG15: 14 cycles

The interrupt handler for ADC12IFG15 shows a way to check immediately if a higher prioritized interrupt occurred during the processing of ADC12IFG15. This saves nine cycles if another ADC12_A interrupt is pending.

```
; Interrupt handler for ADC12.
INT_ADC12                ; Enter Interrupt Service Routine
   ADD      &ADC12IV,PC  ; Add offset to PC
   RETI                  ; Vector 0: No interrupt
   JMP      ADOV         ; Vector 2: ADC overflow
   JMP      ADTOV        ; Vector 4: ADC timing overflow
   JMP      ADM0         ; Vector 6: ADC12IFG0
      ...                ; Vectors 8-32
   JMP      ADM14        ; Vector 34: ADC12IFG14
;
; Handler for ADC12IFG15 starts here. No JMP required.
;
ADM15     MOV   &ADC12MEM15,xxx     ; Move result, flag is reset
          ...                       ; Other instruction needed?
          JMP   INT_ADC12           ; Check other int pending
;
; ADC12IFG14-ADC12IFG1 handlers go here
;
ADM0      MOV   &ADC12MEM0,xxx      ; Move result, flag is reset
          ...                       ; Other instruction needed?
RETI                                ; Return
;
ADTOV     ...                       ; Handle Conv. time overflow
          RETI                      ; Return
;
ADOV      ...                       ; Handle ADCMEMx overflow
          RETI                      ; Return
```

18.3 ADC12_A Registers

The ADC12_A registers are listed in Table 18-2. The base address of the ADC12_A can be found in the devices specific datasheet. The address offset of each ADC12_A register is given in Table 18-2.

Table 18-2. ADC12_A Registers

Register	Short Form	Register Type	Address	Initial State
ADC12 control register 0	ADC12CTL0	Read/write	00h	Reset with POR
ADC12 control register 1	ADC12CTL1	Read/write	02h	Reset with POR
ADC12 control register 2	ADC12CTL2	Read/write	04h	Reset with POR
ADC12 interrupt flag register	ADC12IFG	Read/write	0Ah	Reset with POR
ADC12 interrupt enable register	ADC12IE	Read/write	0Ch	Reset with POR
ADC12 interrupt vector word	ADC12IV	Read	0Eh	Reset with POR
ADC12 memory 0	ADC12MEM0	Read/write	20h	Reset with POR
ADC12 memory 1	ADC12MEM1	Read/write	22h	Reset with POR
ADC12 memory 2	ADC12MEM2	Read/write	24h	Reset with POR
ADC12 memory 3	ADC12MEM3	Read/write	26h	Reset with POR
ADC12 memory 4	ADC12MEM4	Read/write	28h	Reset with POR
ADC12 memory 5	ADC12MEM5	Read/write	2Ah	Reset with POR
ADC12 memory 6	ADC12MEM6	Read/write	2Ch	Reset with POR
ADC12 memory 7	ADC12MEM7	Read/write	2Eh	Reset with POR
ADC12 memory 8	ADC12MEM8	Read/write	30h	Reset with POR
ADC12 memory 9	ADC12MEM9	Read/write	32h	Reset with POR
ADC12 memory 10	ADC12MEM10	Read/write	34h	Reset with POR
ADC12 memory 11	ADC12MEM11	Read/write	36h	Reset with POR
ADC12 memory 12	ADC12MEM12	Read/write	38h	Reset with POR
ADC12 memory 13	ADC12MEM13	Read/write	3Ah	Reset with POR
ADC12 memory 14	ADC12MEM14	Read/write	3Ch	Reset with POR
ADC12 memory 15	ADC12MEM15	Read/write	3Eh	Reset with POR
ADC12 memory control 0	ADC12MCTL0	Read/write	10h	Reset with POR
ADC12 memory control 1	ADC12MCTL1	Read/write	11h	Reset with POR
ADC12 memory control 2	ADC12MCTL2	Read/write	12h	Reset with POR
ADC12 memory control 3	ADC12MCTL3	Read/write	13h	Reset with POR
ADC12 memory control 4	ADC12MCTL4	Read/write	14h	Reset with POR
ADC12 memory control 5	ADC12MCTL5	Read/write	15h	Reset with POR
ADC12 memory control 6	ADC12MCTL6	Read/write	16h	Reset with POR
ADC12 memory control 7	ADC12MCTL7	Read/write	17h	Reset with POR
ADC12 memory control 8	ADC12MCTL8	Read/write	18h	Reset with POR
ADC12 memory control 9	ADC12MCTL9	Read/write	19h	Reset with POR
ADC12 memory control 10	ADC12MCTL10	Read/write	1Ah	Reset with POR
ADC12 memory control 11	ADC12MCTL11	Read/write	1Bh	Reset with POR
ADC12 memory control 12	ADC12MCTL12	Read/write	1Ch	Reset with POR
ADC12 memory control 13	ADC12MCTL13	Read/write	1Dh	Reset with POR
ADC12 memory control 14	ADC12MCTL14	Read/write	1Eh	Reset with POR
ADC12 memory control 15	ADC12MCTL15	Read/write	1Fh	Reset with POR

ADC12CTL0, ADC12_A Control Register 0

15	14	13	12	11	10	9	8
ADC12SHT1x				**ADC12SHT0x**			
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
ADC12MSC	**ADC12 REF2_5V**	**ADC12 REFON**	**ADC120N**	**ADC12OVIE**	**ADC12TOVIE**	**ADC12ENC**	**ADC12SC**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

▨ Modifiable only when ADC12ENC = 0

ADC12SHT1x Bits 15-12 ADC12_A sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM8 to ADC12MEM15.

ADC12SHT0x Bits 11-8 ADC12_A sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM0 to ADC12MEM7.

ADC12SHTx Bits	ADC12CLK Cycles
0000	4
0001	8
0010	16
0011	32
0100	64
0101	96
0110	128
0111	192
1000	256
1001	384
1010	512
1011	768
1100	1024
1101	1024
1110	1024
1111	1024

ADC12MSC Bit 7 ADC12_A multiple sample and conversion. Valid only for sequence or repeated modes.

0 The sampling timer requires a rising edge of the SHI signal to trigger each sample-and-convert.

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.

ADC12REF2_5V Bit 6 ADC12_A reference generator voltage. ADC12REFON must also be set.

0 1.5 V

1 2.5 V

ADC12REFON Bit 5 ADC12_A reference generator on. In devices with the REF module this bit is only valid if the REFMSTR bit of the REF module is set to 0. In the F54xx device the REF module is not available.

0 Reference off

1 Reference on

ADC120N Bit 4 ADC12_A on

0 ADC12_A off

1 ADC12_A on

ADC12OVIE Bit 3 ADC12MEMx overflow-interrupt enable. The GIE bit must also be set to enable the interrupt.

0 Overflow interrupt disabled

1 Overflow interrupt enabled

ADC12TOVIE	Bit 2	ADC12_A conversion-time-overflow interrupt enable. The GIE bit must also be set to enable the interrupt. 0 Conversion time overflow interrupt disabled 1 Conversion time overflow interrupt enabled
ADC12ENC	Bit 1	ADC12_A enable conversion 0 ADC12_A disabled 1 ADC12_A enabled
ADC12SC	Bit 0	ADC12_A start conversion. Software-controlled sample-and-conversion start. ADC12SC and ADC12ENC may be set together with one instruction. ADC12SC is reset automatically. 0 No sample-and-conversion-start 1 Start sample-and-conversion

ADC12CTL1, ADC12_A Control Register 1

15	14	13	12	11	10	9	8
ADC12CSTARTADDx				**ADC12SHSx**		**ADC12SHP**	**ADC12ISSH**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
ADC12DIVx			**ADC12SSELx**		**ADC12CONSEQx**		**ADC12BUSY**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)

▨ Modifiable only when ADC12ENC = 0

ADC12 CSTARTADDx	Bits 15-12	ADC12_A conversion start address. These bits select which ADC12_A conversion-memory register is used for a single conversion or for the first conversion in a sequence. The value of CSTARTADDx is 0 to 0Fh, corresponding to ADC12MEM0 to ADC12MEM15.	
ADC12SHSx	Bits 11-10	ADC12_A sample-and-hold source select	
		00	ADC12SC bit
		01	Timer_A.OUT1
		10	Timer_B.OUT0
		11	Timer_B.OUT1
ADC12SHP	Bit 9	ADC12_A sample-and-hold pulse-mode select. This bit selects the source of the sampling signal (SAMPCON) to be either the output of the sampling timer or the sample-input signal directly.	
		0	SAMPCON signal is sourced from the sample-input signal.
		1	SAMPCON signal is sourced from the sampling timer.
ADC12ISSH	Bit 8	ADC12_A invert signal sample-and-hold	
		0	The sample-input signal is not inverted.
		1	The sample-input signal is inverted.
ADC12DIVx	Bits 7-5	ADC12_A clock divider	
		000	/1
		001	/2
		010	/3
		011	/4
		100	/5
		101	/6
		110	/7
		111	/8
ADC12SSELx	Bits 4-3	ADC12_A clock source select	
		00	MODCLK
		01	ACLK
		10	MCLK
		11	SMCLK
ADC12CONSEQx	Bits 2-1	ADC12_A Conversion sequence mode select	
		00	Single-channel, single-conversion
		01	Sequence-of-channels
		10	Repeat-single-channel
		11	Repeat-sequence-of-channels
ADC12BUSY	Bit 0	ADC12_A busy. This bit indicates an active sample or conversion operation.	
		0	No operation is active.
		1	A sequence, sample, or conversion is active**.**

ADC12CTL2, ADC12_A Control Register 2

15	14	13	12	11	10	9	8
Reserved							ADC12PDIV
r-0	r-0	r-0	r-0	r-0	r-0	r-0	rw-0

7	6	5	4	3	2	1	0
ADC12TCOFF	**Reserved**	**ADC12RES**		**ADC12DF**	**ADC12SR**	**ADC12 REFOUT**	**ADC12 REFBURST**
rw-(0)	r-0	rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

▨ Modifiable only when ADC12ENC = 0

Reserved Bits 15-9 Reserved. Read back as 0.

ADC12PDIV Bit 8 ADC12_A pre-divider. This bit pre-divides the selected ADC12_A clock source.

0 Pre-divide by 1

1 Pre-divide by 4

ADC12TCOFF Bit 7 ADC12_A temperature sensor off. If the bit is set the temperature sensor turned off. This is used to save power.

Reserved Bit 6 Reserved. Read back as 0.

ADC12RES Bits 5-4 ADC12_A resolution. This bit defines the conversion result resolution.

00 8-bit (9 clock cycle conversion time)

01 10-bit (11 clock cycle conversion time)

10 12-bit (13 clock cycle conversion time)

11 Reserved

ADC12DF Bit 3 ADC12_A data read-back format. Data is always stored in the binary unsigned format.

0 Binary unsigned. Theoretically the analog input voltage – V_{REF} results in 0000h, the analog input voltage + V_{REF} results in 0FFFh.

1 Signed binary (2's complement), left aligned. Theoretically the analog input voltage – V_{REF} results in 8000h, the analog input voltage + V_{REF} results in 7FF0h.

ADC12SR Bit 2 ADC12_A sampling rate. This bit selects the reference buffer drive capability for the maximum sampling rate. Setting ADC12SR reduces the current consumption of the reference buffer.

0 Reference buffer supports up to ~200 ksps

1 Reference buffer supports up to ~50 ksps

ADC12REFOUT Bit 1 Reference output

0 Reference output off

1 Reference output on

ADC12REFBURST Bit 0 Reference burst. ADC12REFOUT must also be set.

0 Reference buffer on continuously

1 Reference buffer on only during sample-and-conversion

ADC12MEMx, ADC12_A Conversion Memory Registers

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**Conversion Results**			
r0	r0	r0	r0	rw	rw	rw	rw

7	6	5	4	3	2	1	0
Conversion Results							
rw	rw	rw	rw	rw	rw	rw	rw

Conversion Results Bits 15-0 The 12-bit conversion results are right-justified. Bit 11 is the MSB. Bits 15-12 are 0 in 12-bit mode, bits 15-10 are 0 in 10-bit mode and bits 15-8 are 0 in 8-bit mode. Writing to the conversion memory registers will corrupt the results. This data format is used if ADC12DF = 0.

ADC12MEMx, ADC12_A Conversion-Memory Register, 2's Complement Format

15	14	13	12	11	10	9	8
Conversion Results							
rw	rw	rw	rw	rw	rw	rw	rw

7	6	5	4	3	2	1	0
Conversion Results				**0**	**0**	**0**	**0**
rw	rw	rw	rw	r0	r0	r0	r0

Conversion Results Bits 15-0 The 12-bit conversion results are left-justified, 2's complement format. Bit 15 is the MSB. Bits 3-0 are 0 in 12-bit mode, bits 5-0 are 0 in 10-bit mode and bits 7-0 are 0 in 8-bit mode. This data format is used if ADC12DF = 1. The data is stored in the right justified format and is converted to the left-justified 2's complement during read-back.

ADC12MCTLx, ADC12_A Conversion Memory Control Registers

7	6	5	4	3	2	1	0
ADC12EOS	**ADC12SREFx**			**ADC12INCHx**			
rw	rw	rw	rw	rw	rw	rw	rw

▨ Modifiable only when ADC12ENC = 0

ADC12EOS Bit 7 End of sequence. Indicates the last conversion in a sequence.

0	Not end of sequence
1	End of sequence

ADC12SREFx Bits 6-4 Select reference

000	V_{R+} = AV_{CC} and V_{R-} = AV_{SS}
001	V_{R+} = V_{REF+} and V_{R-} = AV_{SS}
010	V_{R+} = Ve_{REF+} and V_{R-} = AV_{SS}
011	V_{R+} = Ve_{REF+} and V_{R-} = AV_{SS}
100	V_{R+} = AV_{CC} and V_{R-} = V_{REF-}/ Ve_{REF-}
101	V_{R+} = V_{REF+} and V_{R-} = V_{REF-}/ Ve_{REF-}
110	V_{R+} = Ve_{REF+} and V_{R-} = V_{REF-}/ Ve_{REF-}
111	V_{R+} = Ve_{REF+} and V_{R-} = V_{REF-}/ Ve_{REF-}

ADC12INCHx Bits 3-0 Input channel select

0000	A0
0001	A1
0010	A2
0011	A3
0100	A4
0101	A5
0110	A6
0111	A7
1000	Ve_{REF+}
1001	V_{REF-}/Ve_{REF-}
1010	Temperature diode
1011	(AV_{CC} – AV_{SS}) / 2
1100	A12
1101	A13
1110	A14
1111	A15

ADC12IE, ADC12_A Interrupt Enable Register

15	14	13	12	11	10	9	8
ADC12IE15	**ADC12IE14**	**ADC12IE13**	**ADC12IE12**	**ADC12IE11**	**ADC12IE10**	**ADC12IFG9**	**ADC12IE8**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
ADC12IE7	**ADC12IE6**	**ADC12IE5**	**ADC12IE4**	**ADC12IE3**	**ADC12IE2**	**ADC12IE1**	**ADC12IE0**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC12IEx Bits 15-0 Interrupt enable. These bits enable or disable the interrupt request for the ADC12IFGx bits.

0 Interrupt disabled

1 Interrupt enabled

ADC12IFG, ADC12_A Interrupt Flag Register

15	14	13	12	11	10	9	8
ADC12IFG15	**ADC12IFG14**	**ADC12IFG13**	**ADC12IFG12**	**ADC12IFG11**	**ADC12IFG10**	**ADC12IFG9**	**ADC12IFG8**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
ADC12IFG7	**ADC12IFG6**	**ADC12IFG5**	**ADC12IFG4**	**ADC12IFG3**	**ADC12IFG2**	**ADC12IFG1**	**ADC12IFG0**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC12IFGx Bits 15-0 ADC12MEMx Interrupt flag. These bits are set when corresponding ADC12MEMx is loaded with a conversion result. The ADC12IFGx bits are reset if the corresponding ADC12MEMx is accessed, or may be reset with software.

0 No interrupt pending

1 Interrupt pending

ADC12IV, ADC12_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**	**ADC12IVx**					**0**
r0	r0	r-(0)	r-(0)	r-(0)	r-(0)	r-(0)	r0

ADC12IVx Bits 15-0 ADC12_A interrupt vector value.

ADC12IV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No interrupt pending	–	
002h	ADC12MEMx overflow	–	Highest
004h	Conversion time overflow	–	
006h	ADC12MEM0 interrupt flag	ADC12IFG0	
008h	ADC12MEM1 interrupt flag	ADC12IFG1	
00Ah	ADC12MEM2 interrupt flag	ADC12IFG2	
00Ch	ADC12MEM3 interrupt flag	ADC12IFG3	
00Eh	ADC12MEM4 interrupt flag	ADC12IFG4	
010h	ADC12MEM5 interrupt flag	ADC12IFG5	
012h	ADC12MEM6 interrupt flag	ADC12IFG6	
014h	ADC12MEM7 interrupt flag	ADC12IFG7	
016h	ADC12MEM8 interrupt flag	ADC12IFG8	
018h	ADC12MEM9 interrupt flag	ADC12IFG9	
01Ah	ADC12MEM10 interrupt flag	ADC12IFG10	
01Ch	ADC12MEM11 interrupt flag	ADC12IFG11	
01Eh	ADC12MEM12 interrupt flag	ADC12IFG12	
020h	ADC12MEM13 interrupt flag	ADC12IFG13	
022h	ADC12MEM14 interrupt flag	ADC12IFG14	
024h	ADC12MEM15 interrupt flag	ADC12IFG15	Lowest

Chapter 19

SLAU208–June 2008

Embedded Emulation Module (EEM)

This chapter describes the Embedded Emulation Module (EEM) that is implemented in all MSP430 flash devices.

Topic **Page**

19.1 EEM Introduction

Every MSP430 flash-based microcontroller implements an embedded emulation module (EEM). It is accessed and controlled through either 4-wire JTAG mode or Spy-Bi-Wire mode. Each implementation is device dependent and is described in Section 19.3 EEM Configurations and the device data sheet.

In general, the following features are available:

- Nonintrusive code execution with real-time breakpoint control
- Single step, step into, and step over functionality
- Full support of all low-power modes
- Support for all system frequencies, for all clock sources
- Up to eight (device dependent) hardware triggers/breakpoints on memory address bus (MAB) or memory data bus (MDB)
- Up to two (device dependent) hardware triggers/breakpoints on CPU register write accesses
- MAB, MDB, and CPU register access triggers can be combined to form up to ten (device dependent) complex triggers/breakpoints
- Up to two (device dependent) cycle counters
- Trigger sequencing (device dependent)
- Storage of internal bus and control signals using an integrated trace buffer (device dependent)
- Clock control for timers, communication peripherals, and other modules on a global device level or on a per-module basis during an emulation stop

Figure 19-1 shows a simplified block diagram of the largest currently available 5xx EEM implementation.

For more details on how the features of the EEM can be used together with the IAR Embedded Workbench™ debugger see the application report *Advanced Debugging Using the Enhanced Emulation Module* (SLAA263) at www.msp430.com. Code Composer Essentials (CCE) and most other debuggers supporting MSP430 have the same or a similar feature set. For details see the user's guide of the applicable debugger.

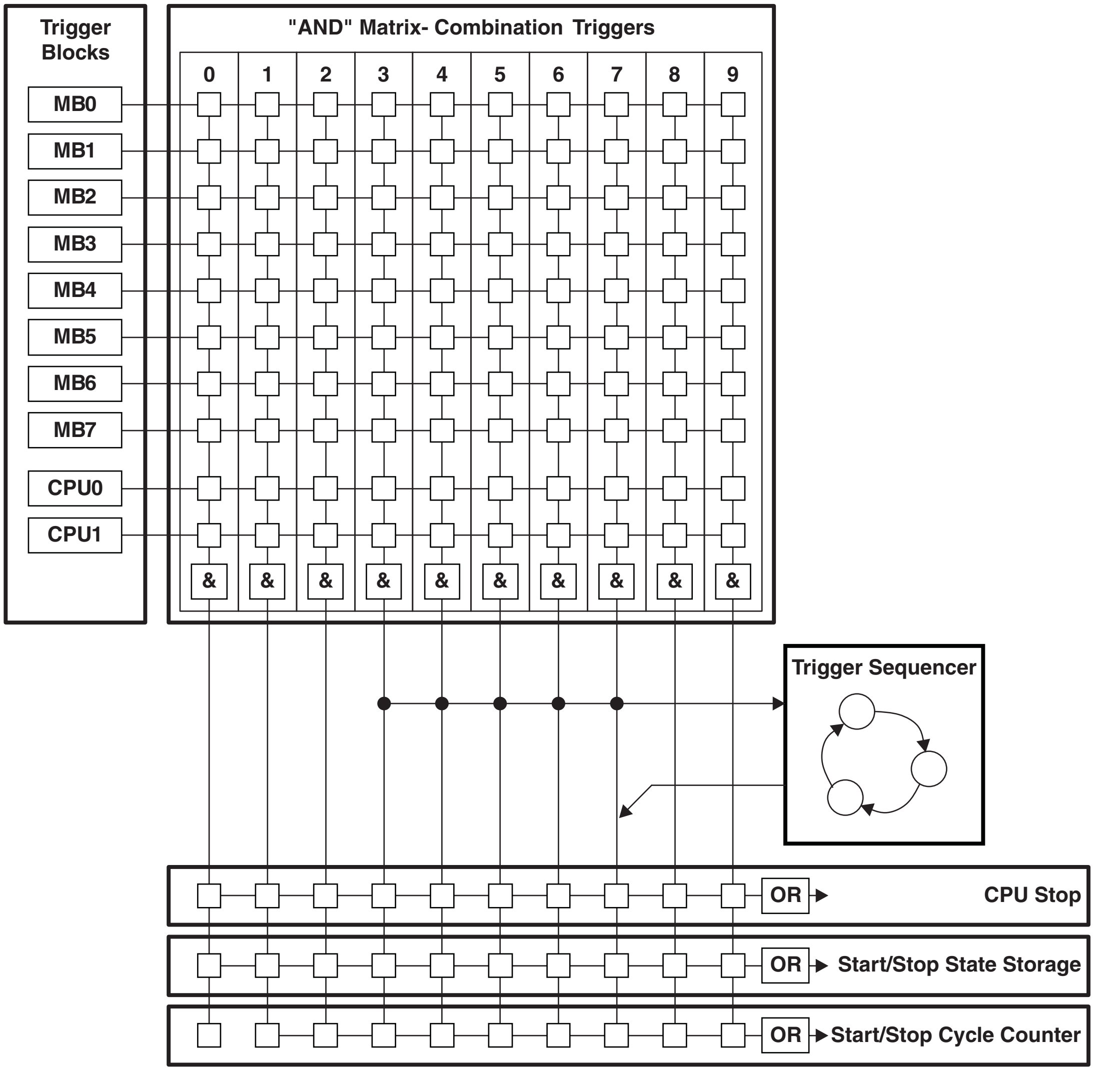


Figure 19-1. Large Implementation of the Embedded Emulation Module (EEM)

19.2 EEM Building Blocks

19.2.1 Triggers

The event control in the EEM of the MSP430 system consists of triggers, which are internal signals indicating that a certain event has happened. These triggers may be used as simple breakpoints, but it is also possible to combine two or more triggers to allow detection of complex events and trigger various reactions besides stopping the CPU.

In general, the triggers can be used to control the following functional blocks of the EEM:

- Breakpoints (CPU stop)
- State storage
- Sequencer
- Cycle counter

There are two different types of triggers: the memory trigger and the CPU register write trigger.

Each memory trigger block can be independently selected to compare either the MAB or the MDB with a given value. Depending on the implemented EEM the comparison can be =, ≠, ≥, or ≤. The comparison can also be limited to certain bits with the use of a mask. The mask is either bit-wise or byte-wise, depending upon the device. In addition to selecting the bus and the comparison, the condition under which the trigger is active can be selected. The conditions include read access, write access, DMA access, and instruction fetch.

Each CPU register write trigger block can be independently selected to compare what is written into a selected register with a given value. The observed register can be selected for each trigger independently. The comparison can be =, ≠, ≥, or ≤. The comparison can also be limited to certain bits with the use of a bit mask.

Both types of triggers can be combined to form more complex triggers. For example, a complex trigger can signal when a particular value is written into a user-specified address.

19.2.2 Trigger Sequencer

The trigger sequencer allows the definition of a certain sequence of trigger signals before an event is accepted for a break or state storage event. Within the trigger sequencer, it is possible to use the following features:

- Four states (State 0 to State 3)
- Two transitions per state to any other state
- Reset trigger that resets the sequencer to State 0.

The trigger sequencer always starts at State 0 and must execute to State 3 to generate an action. If State 1 or State 2 are not required, they can be bypassed.

19.2.3 State Storage (Internal Trace Buffer)

The state storage function uses a built-in buffer to store MAB, MDB, and CPU control signal information (i.e., read, write, or instruction fetch) in a nonintrusive manner. The built-in buffer can hold up to eight entries. The flexible configuration allows the user to record the information of interest very efficiently.

19.2.4 Cycle Counter

The cycle counter provides one or two 40-bit counters to measure the cycles used by the CPU to execute certain tasks. On some devices, the cycle counter operation can be controlled using triggers. This allows, for example, conditional profiling, such as profiling a specific section of code.

19.2.5 Clock Control

The EEM provides device dependent flexible clock control. This is useful in applications where a running clock is needed for peripherals after the CPU is stopped (e.g., to allow a UART module to complete its transfer of a character or to allow a timer to continue generating a PWM signal).

The clock control is flexible and supports both modules that need a running clock and modules that must be stopped when the CPU is stopped due to a breakpoint.

19.3 EEM Configurations

Table 19-1 gives an overview of the EEM configurations in the MSP430 5xx family. The implemented configuration is device dependent (see the device-specific data sheet for details).

Table 19-1. 5xx EEM Configurations

Feature	XS	S	M	L
Memory bus triggers	2 (=, ≠ only)	3	5	8
Memory bus trigger mask for	1) Low byte 2) High byte 3) Four upper addr bits	1) Low byte 2) High byte 3) Four upper addr bits	1) Low byte 2) High byte 3) Four upper addr bits	All 16 or 20 bits
CPU register write triggers	0	1	1	2
Combination triggers	2	4	6	10
Sequencer	No	No	Yes	Yes
State storage	No	No	No	Yes
Cycle counter	1	1	1	2 (including triggered start/stop)

In general the following features can be found on any 5xx device:

- At least two MAB/MDB triggers supporting:
  - Distinction between CPU, DMA, read, and write accesses
  - =, ≠, ≥, or ≤ comparison (in XS, only =, ≠)
- At least two trigger combination registers
- Hardware breakpoints using the CPU stop reaction
- At least one 40-bit cycle counter
- Enhanced clock control with individual control of module clocks

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 TI 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. Information of third parties may be subject to additional restrictions.

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.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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
Clocks and Timers	www.ti.com/clocks	Digital Control	www.ti.com/digitalcontrol
Interface	interface.ti.com	Medical	www.ti.com/medical
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
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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