PRELIMINARY INFORMATION

MSP430FR5xx Family

User's Guide

Literature Number: SLAU272
May 2011

Contents

List of Figures

List of Tables

Read This First

About This Manual

This manual describes the modules and peripherals of the MSP430x5xx/MSP430x6xx 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
z16	16 bit address space

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

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

The system control module (SYS) is available on all devices. The basic features of SYS are:

- Brownout reset/power on reset (BOR/POR) handling
- Power up clear (PUC) handling
- (Non)maskable interrupt (SNMI/UNMI) event source selection and management
- Providing an user data-exchange mechanism via the JTAG mailbox (JMB)
- Bootstrap loader (BSL) entry mechanism
- Configuration management (device descriptors)
- Providing interrupt vector generators for reset and NMIs

Topic **Page**

1.1 System Control Module (SYS) Introduction

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

1.2 System Reset and Initialization

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

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

- Powering up the device
- A low signal on the $\overline{\text{RST}}$/NMI pin when configured in the reset mode
- A wakeup event from LPMx.5 (that is, LPM3.5 or LPM4.5) modes
- A software BOR event (see the PMM and SVS chapter for details)

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

- A BOR signal
- A SVS_H low condition when enabled (see the PMM and SVS chapter for details)
- A SVS_L low condition when enabled (see the PMM and SVS chapter for details)
- A software POR event (see the PMM and SVS chapter for details)

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

- A POR signal
- Watchdog timer expiration when watchdog mode only (see the WDT_A chapter for details)
- Watchdog timer password violation (see the WDT_A chapter for details)
- A FRAM memory password violation (see the FRAM Memory Controller chapter for details)
- Power Management Module password violation (see the PMM and SVS chapter for details)
- Memory Protection Unit password violation (see the MPU chapter for details)
- Memory segment violation (see the MPU chapter for details)
- Clock System password violation (see the CS chapter for details)
- Fetch from peripheral area

NOTE: The number and type of resets available may vary from device to device. See the device-specific data sheet for all reset sources available.

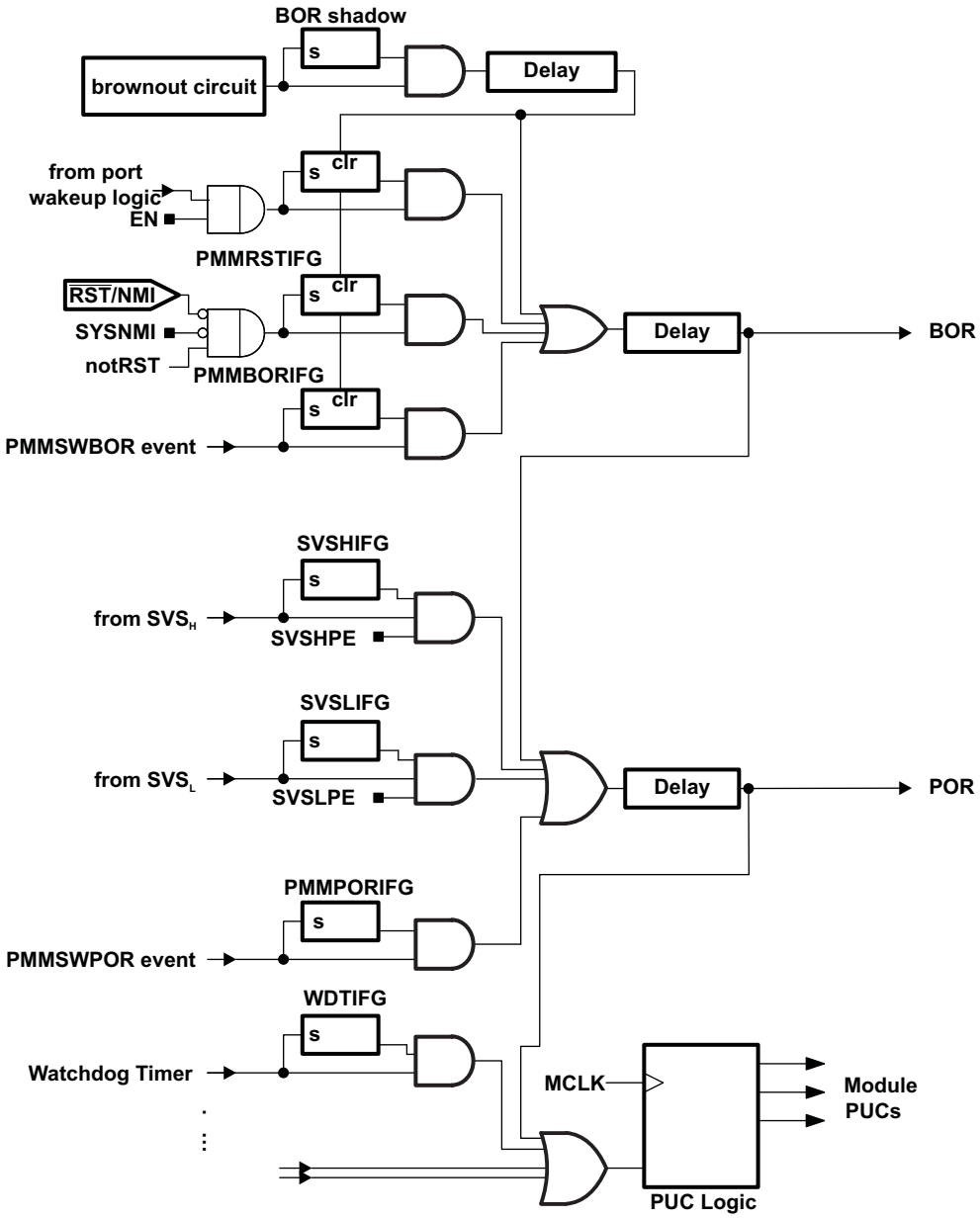


Figure 1-1. BOR/POR/PUC Reset Circuit

1.2.1 Device Initial Conditions After System Reset

After a BOR, the initial device conditions are:

- The $\overline{\text{RST}}$/NMI pin is configured in the reset mode. See Section 1.7 for details on configuring the $\overline{\text{RST}}$/NMI pin.
- I/O pins are switched to input mode as described in the Digital I/O chapter.
- Other peripheral modules and registers are initialized as described in their respective chapters in this manual.
- Status register (SR) is reset.
- The watchdog timer powers up active in watchdog mode.
- Program counter (PC) is loaded with the boot code address and boot code execution begins at that address. See Section 1.9 for more information regarding the boot code. Upon completion of the boot code, the PC is loaded with the address contained at the SYSRSTIV reset location (0FFFEh).

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

- Initialize the stack pointer (SP), typically to the top of RAM when available, otherwise FRAM location.
- Initialize the watchdog to the requirements of the application.
- Configure peripheral modules to the requirements of the application.

NOTE: A device that is unprogrammed or blank is defined as having its reset vector value, residing at memory address FFFEh, equal to FFFFh. Upon system reset of a blank device, the device will enter operating mode LPM4 automatically. SeeSection 1.4 for information on operating modes andSection 1.3.6 for details on interrupt vectors.

1.3 Interrupts

The interrupt priorities are fixed and defined by the arrangement of the modules in the connection chain as shown in Figure 1-2. 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
- Maskable

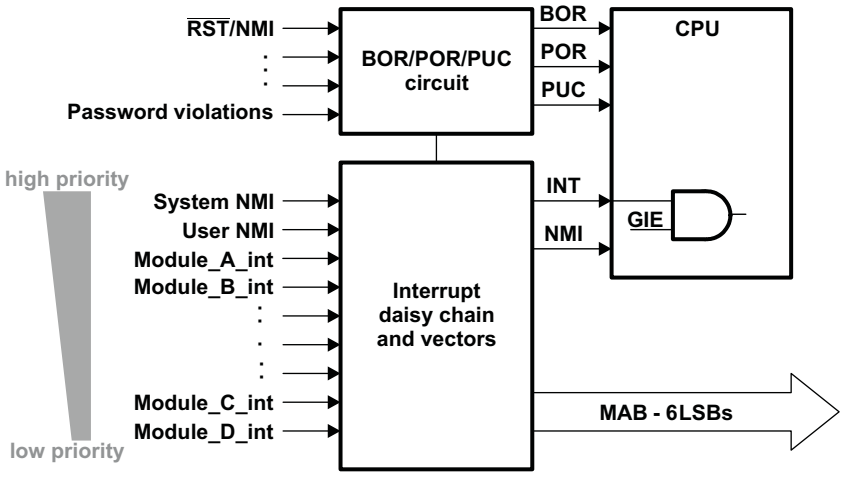


Figure 1-2. Interrupt Priority

NOTE: The types of interrupt sources available and their respective priorities can change from device to device. See the device-specific data sheet for all interrupt sources and their priorities.

1.3.1 (Non)Maskable Interrupts (NMIs)

In general, NMIs are not masked by the general interrupt enable (GIE) bit. The family supports two levels of NMIs — system NMI (SNMI) and user NMI (UNMI). The NMI sources are enabled by individual interrupt enable bits. When an 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 NMI vector as shown in Section 1.3.6. To allow software backward compatibility to users of earlier MSP430 families, the software may, but does not need to, reenable NMI sources. The block diagram for NMI sources is shown in Section 1.3.

A UNMI interrupt can be generated by following sources:

- An edge on the $\overline{\text{RST}}$/NMI pin when configured in NMI mode
- An oscillator fault occurs

A SNMI interrupt can be generated by following sources:

- Single bit, double bit FRAM errors (see the *FRAM Memory Controller* chapter for details)
- Vacant memory access
- JTAG mailbox (JMB) event

NOTE: The number and types of NMI sources may vary from device to device. See the device-specific data sheet for all NMI sources available.

1.3.2 SNMI Timing

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

1.3.3 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 in this manual.

1.3.4 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 (NMI) to be requested.

1.3.4.1 Interrupt Acceptance

The interrupt latency is six 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.

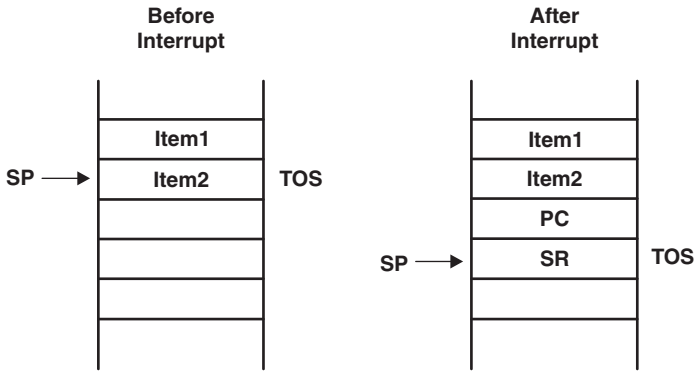


Figure 1-3. Interrupt Processing

1.3.4.2 Return From Interrupt

The interrupt handling routine terminates with the instruction:

```
RETI //return from an interrupt service routine
```

The return from the interrupt takes five 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.

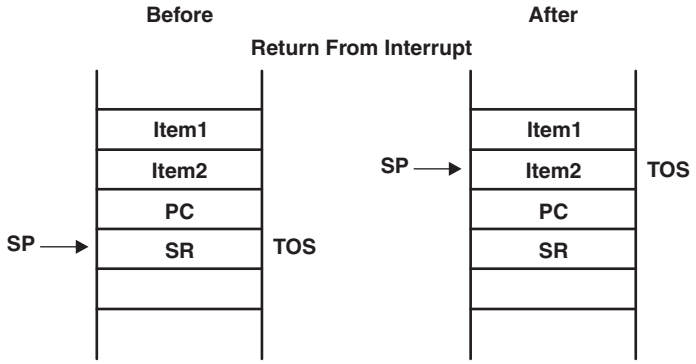


Figure 1-4. Return From Interrupt

1.3.5 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 interrupts the routine, regardless of the interrupt priorities.

1.3.6 Interrupt Vectors

The interrupt vectors are located in the address range 0FFFFh to 0FF80h, for a maximum of 64 interrupt sources. A vector is programmed by the user and points to the start location of the corresponding interrupt service routine. Table 1-1 is an example of the interrupt vectors available. See the device-specific data sheet for the complete interrupt vector list.

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

Interrupt Source	Interrupt Flag	System Interrupt	Word Address	Priority
Reset: power up, external reset watchdog, FRAM password	... WDTIFG FRCTLPW	... Reset	... 0FFFEh	... Highest
System NMI: JTAG Mailbox	JMBINIFG, JMBOUTIFG	(Non)maskable	0FFFCh	…
User NMI: NMI oscillator fault	... NMIIFG OFIFG	... (Non)maskable (Non)maskable	... 0FFFAh	... …
Device specific			0FFF8h	…
...			...	...
Watchdog timer	WDTIFG	Maskable	...	...
...			...	...
Device specific			…	…
Reserved		Maskable	…	Lowest

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

1.3.6.1 Alternate Interrupt Vectors

On devices that contain RAM, It is possible to use the RAM as an alternate location for the interrupt vector locations. Setting the SYSRIVECT bit in SYSCTL causes the interrupt vectors to be remapped to the top of RAM. Once set, any interrupt vectors to the alternate locations now residing in RAM. Because SYSRIVECT is automatically cleared on a BOR, it is critical that the reset vector at location 0FFFEh still be available and handled properly in firmware.

1.3.7 SYS Interrupt Vector Generators

SYS collects all system NMI (SNMI) sources, user NMI (UNMI) sources, and BOR/POR/PUC (reset) 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 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. Reading 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. Writing to the SYSRSTIV, SYSSNIV, SYSUNIV register automatically resets all pending interrupt flags of the group.

1.3.7.1 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 software approach can be used. The following is an example for a generic 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      DBD_ISR        ; Vector 2: DBDIFG
          JMP      ACCTIM_ISR     ; Vector 4: ACCTIMIFG
          JMP      RSVD1_ISR      ; Vector 6: Reserved for future usage.
          JMP      RSVD2_ISR      ; Vector 8: Reserved for future usage.
          JMP      RSVD3_ISR      ; Vector 10: Reserved for future usage.
          JMP      RSVD4_ISR      ; Vector 12: Reserved for future usage.
          JMP   ACCV_ISR    ; Vector 14: ACCVIFG
          JMP   VMA_ISR     ; Vector 16: VMAIFG
          JMP   JMBI_ISR    ; Vector 18: JMBINIFG
          JMP   JMBO_ISR    ; Vector 20: JMBOUTIFG
          JMP   SBD_ISR     ; Vector 22: SBDIFG

DBD_ISR:                          ; Vector 2: ACCVIFG
         ...                      ; Task_2 starts here
         RETI                     ; Return
ACCTIM_ISR:                       ; Vector 4
         ...                      ; Task_4 starts here
         RETI                     ; Return
RSVD1_ISR:                        ; Vector 6
         ...                   ; Task_6 starts here
         RETI                     ; Return
RSVD2_ISR:                        ; Vector 8
         ...                      ; Task_8 starts here
         RETI                     ; Return
RSVD3_ISR:                        ; Vector 10
         ...                      ; Task_10 starts here
         RETI                     ; Return
RSVD4_ISR:                        ; Vector 12
         ...                      ; Task_12 starts here
         RETI                     ; Return
ACCV_ISR:                         ; Vector 14
         ...                      ; Task_14 starts here
         RETI                     ; Return
VMA_ISR:                             ; Vector 16
        ...                       ; Task_16 starts here
          RETI                    ; Return
JMBI_ISR:                         ; Vector 18
        ...                       ; Task_18 starts here
JMBO_ISR:                         ; Vector 20
        ...                       ; Task_20 starts here
        RETI                      ; Return
SBD_ISR:                          ; Vector 22
        ...                       ; Task_22 starts here
        RETI                      ; Return
```

1.4 Operating Modes

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

The operating modes take into account three different needs:

- Ultra-low 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 SR. The advantage of including the CPUOFF, OSCOFF, SCG0, and SCG1 mode-control bits in the SR 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. 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. Peripherals may also be disabled with their individual control register settings. All I/O port pins and RAM/registers are unchanged. Wakeup from LPM0 through LPM4 is possible through all enabled interrupts.

When LPMx.5 (LPM3.5 or LPM4.5) is entered, the voltage regulator of the Power Management Module (PMM) is disabled. All RAM and register contents are lost. Although the I/O register contents are lost, the I/O pin states are locked upon LPMx.5 entry. See the *Digital I/O* chapter for further details. Wakeup from LPM4.5 is possible via a power sequence, a $\overline{\text{RST}}$ event, or from specific I/O. Wakeup from LPM3.5 is possible via a power sequence, a $\overline{\text{RST}}$ event, RTC event, or from specific I/O.

NOTE: The TEST/SBWTCK pin is used for interfacing to the development tools via Spy-Bi-Wire and JTAG. When the TEST/SBWTCK pin is high, wakeup times from LPM2, LPM3, and LPM4 may be different compared to when TEST/SBWTCK is low. Pay careful attention to the real-time behavior when exiting from LPM2, LPM3, and LPM4 with the device connected to a development tool (for example, MSP-FET430UIF). See the *Power Management Module* chapter for further details.

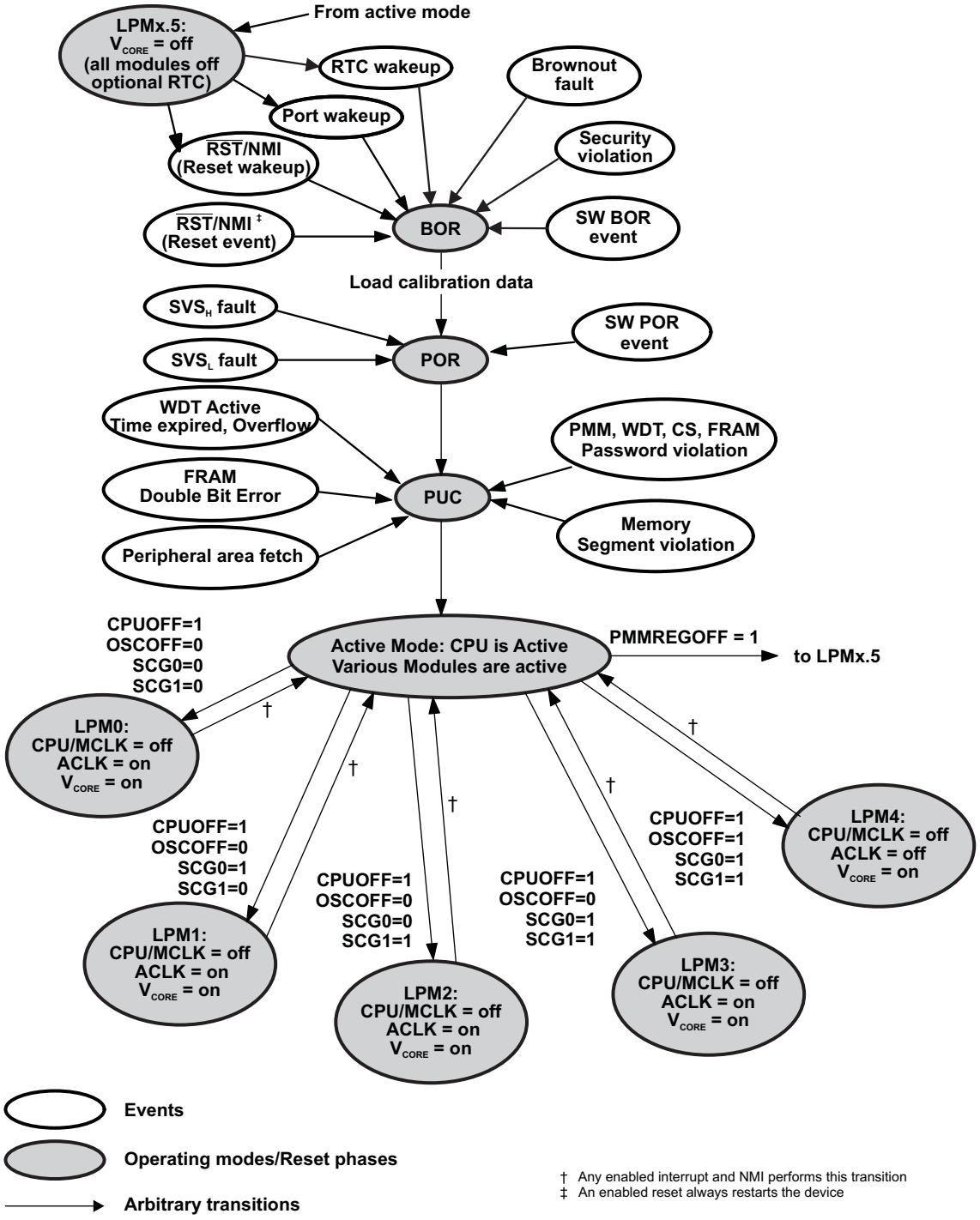


Figure 1-5. Operation Modes

SCG1 (1)	SCG0	OSCOFF (1)	CPUOFF (1)	Mode	CPU and Clocks Status (2)
0	0	0	0	Active	CPU, MCLK are active. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK, MCLK, or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0).
0	0	0	1	LPM0	CPU, MCLK are disabled. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0).
0	1	0	1	LPM1	CPU, MCLK are disabled. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0).
1	0	0	1	LPM2	CPU, MCLK are disabled. ACLK is active. SMCLK is disabled. DCO is enabled if sources ACLK.
1	1	0	1	LPM3	CPU, MCLK are disabled. ACLK is active. SMCLK is disabled. DCO is enabled if sources ACLK.
1	1	1	1	LPM4	CPU and all clocks are disabled.
1	1	1	1	LPM3.5	When PMMREGOFF = 1, regulator is disabled. No memory retention. In this mode, RTC operation is possible when configured properly. See the *RTC* module for further details.
1	1	1	1	LPM4.5	When PMMREGOFF = 1, regulator is disabled. No memory retention. In this mode, all clock sources are disabled i.e. no RTC operation is possible.

(1) This bit is automatically reset when exiting low power modes. See Section 1.4.1 for details.

(2) The low power modes and hence the system clocks can be affected by the clock request system. See the *Clock System* chapter for details.

1.4.1 Entering and Exiting Low-Power Modes LPM0 Through LPM4

An enabled interrupt event wakes the device from low-power operating modes LPM0 through LPM4. The program flow for 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
```

1.4.2 Entering and Exiting Low-Power Modes LPMx.5

LPMx.5 entry and exit is handled differently than the other low power modes. LPMx.5, when used properly, gives the lowest power consumption available on a device. To achieve this, entry to LPMx.5 disables the LDO of the PMM module, removing the supply voltage from the core of the device. Because the supply voltage is removed from the core, all register contents, as well as, SRAM contents are lost. Exit from LPMx.5 causes a BOR event, which forces a complete reset of the system. Therefore, it is the application's responsibility to properly reconfigure the device upon exit from LPMx.5.

The wakeup time from LPMx.5 is significantly longer than the wakeup time from the other power modes (see the device-specific data sheet). This is primarily due to the facts that after exit from LPMx.5, time is required for the core voltage supply to be regenerated, as well as, boot code execution to complete before the application code can begin. Therefore, the usage of LPMx.5 is restricted to very low duty cycle events.

There are two LPMx.5 power modes, LPM3.5 and LPM4.5. LPM4.5 allows for the lowest power consumption available. No clock sources are active during LPM4.5. LPM3.5 is similar to LPM4.5, but has the additional capability of having a RTC mode available. In addition to the wakeup events possible in LPM4.5, RTC wakeup events are also possible in LPM3.5.

The program flow for entering LPMx.5 is:

- Configure I/O appropriately. See the *Digital I/O* chapter for complete details on configuring I/O for LPMx.5.
  - Set all ports to general purpose I/O. Configure each port to ensure no floating inputs based on the application requirements.
  - If wakeup from I/O is desired, configure input ports with interrupt capability appropriately.
- If LPM3.5, is available, and desired, enable RTC operation. In addition, configure any RTC interrupts, if desired for LPM3.5 wakeup event. See the *RTC* chapter for complete details.
- Enter LPMx.5. The following code example shows how to enter LPMx.5 mode. See the *Power Management Module and Supply Voltage Supervisor* chapter for further details.

```
; Enter LPMx.5 Example
   MOV.B #PMMPW_H, &PMMCTL0_H                ; Open PMM registers for write
   BIS.B #PMMREGOFF, &PMMCTL0_L              ;
   BIS   #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR     ; Enter LPMx.5 when PMMREGOFF is set.
```

Exit from LPMx.5 is possible with a $\overline{\text{RST}}$ event, a power on cycle, or via specific I/O. Any exit from LPMx.5 causes a BOR. Program execution continues at the location stored in the system reset vector location 0FFFEh after execution of the boot code. The PMMLPM5IFG bit inside the PMM module is set, indicating that the device was in LPMx.5 prior to the wakeup event. Additionally, SYSRSTIV = 08h, which can be used to generate an efficient reset handler routine. During LPMx.5, all I/O pin conditions are automatically locked to the current state. Upon exit from LPMx.5, the I/O pin conditions remain locked until the application unlocks them. See the *Digital I/O* chapter for complete details. If LPM3.5 was in effect, RTC operation continues uninterrupted upon wakeup. The program flow for exiting LPMx.5 is:

- Enter system reset service routine
  - Reconfigure system as required for the application.
  - Reconfigure I/O as required for the application.
  - Unlock system by clearing LOCKLPM5 bit in PM5CTL0.

1.5 Principles for Low-Power Applications

Often, the most important factor for reducing power consumption is using the device 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.

If the application has low duty cycle, slow response time events, maximizing time in LPMx.5 can further reduce power consumption significantly.

1.6 Connection of Unused Pins

The correct termination of all unused pins is listed in Table 1-2.

Table 1-2. Connection of Unused Pins(1)

Pin	Potential	Comment
AV_{CC}	DV_{CC}	
AV_{SS}	DV_{SS}	
Px.0 to Px.7	Open	Switched to port function, output direction (PxDIR.n = 1)
$\overline{RST}$/NMI	DV_{CC} or V_{CC}	47-kΩ pullup or internal pullup selected with 10-nF (2.2 nF(2)) pulldown(2)
PJ.0/TDO PJ.1/TDI PJ.2/TMS PJ.3/TCK	Open	The JTAG pins are shared with general purpose I/O function (PJ.x). If not being used, these should be switched to port function, output direction. When used as JTAG pins, these pins should remain open.
TEST	Open	This pin always has an internal pulldown enabled.

(1) Any unused pin with a secondary function that is shared with general purpose I/O should follow the Px.0 to Px.7 unused pin connection guidelines.

(2) The pulldown capacitor should not exceed 2.2 nF when using devices with Spy-Bi-Wire interface in Spy-Bi-Wire mode or in 4-wire JTAG mode with TI tools like FET interfaces or GANG programmers.

1.7 Reset pin ($\overline{RST}$/NMI) Configuration

The reset pin can be configured as a reset function (default) or as an NMI function via the Special Function Register (SFR), SFRRPCR. Setting SYSNMI causes the $\overline{RST}$/NMI pin to be configured as an external NMI source. The external NMI is edge sensitive and its edge is selectable by SYSNMIIES. Setting the NMIIE enables the interrupt of the external NMI. Upon an external NMI event, the NMIIFG is set.

The $\overline{RST}$/NMI pin can have either a pullup or pulldown present or not. SYSRSTUP selects either pullup or pulldown, and SYSRSTRE causes the pullup or pulldown to be enabled or not. If the $\overline{RST}$/NMI pin is unused, it is required to have either the internal pullup selected and enabled or an external resistor connected to the $\overline{RST}$/NMI pin as shown in Table 1-2

1.8 Configuring JTAG pins

The JTAG pins are shared with general purpose I/O pins. There are several ways that the JTAG pins can be selected for four wire JTAG mode via software. Normally, upon a BOR, SYSJTAGPIN is cleared. With SYSJTAGPIN cleared, the JTAG are configured as general purpose I/O. See the *Digital I/O* chapter for details on controlling the JTAG pins as general purpose I/O. If SYSJTAG = 1, the JTAG pins are configured to four wire JTAG mode and remain in this mode until another BOR condition occurs. Therefore, SYSJTAGPIN is a write only once function. Clearing it by software is not possible, and the device does not change from four wire JTAG mode to general purpose I/O.

1.9 Boot Code

The boot code is always executed after a BOR. The boot code loads factory stored calibration values of the oscillator and reference voltages. In addition, it checks for a BSL entry sequence.

1.10 Bootstrap Loader (BSL)

The BSL is software that is executed after start-up when a certain BSL entry condition is applied. The BSL enables the user to communicate with the embedded memory in the microcontroller during the prototyping phase, final production, and in service. All memory mapped resources, the programmable memory, the data memory (RAM), and the peripherals, can be modified by the BSL as required.

A basic BSL program is provided by TI and resides in ROM. This supports the commonly used UART protocol with RS232 interfacing, allowing flexible use of both hardware and software. To use the BSL, a specific BSL entry sequence must be applied to specific device pins. The correct entry sequence causes SYSBSLIND to be set. 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 applying the standard reset sequence. Access to the device memory via the BSL is protected against misuse by a user-defined password. For more details, see the *MSP430 Memory Programming User's Guide* (SLAU265) at www.ti.com/msp430.

1.11 Vacant Memory Space

Vacant memory is non-existent memory space. Accesses to vacant memory space generate a system (non)maskable interrupt (SNMI) when enabled (VMAIE = 1). 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 result in a PUC. After the boot code is executed, it behaves like vacant memory space and also causes an NMI on access.

1.12 JTAG Lock Mechanism via the Electronic Fuse

A device can be protected from unauthorized access by disabling the JTAG and SBW interface. This is achieved by programming the electronic fuse. Programming the electronic fuse, completely disables the debug and access capabilities associated with the JTAG and SpyBiWire interface and is not reversible. The JTAG is locked by programming a certain signature into the devices' FRAM memory at dedicated addresses. The JTAG security lock key resides at the end of the bootstrap loader (BSL) memory at addresses 17FCh through 17FFh. Anything other than 0h or FFFFFFFFh programmed to these addresses locks the JTAG interface irreversibly.

All of the 5xx MSP430 devices come with a preprogrammed BSL (TI-BSL) code which by default protects itself from unintended erase and write access. This is done by setting SYSBSLPE in the SYSBSLC register. Because the JTAG security lock key resides in the BSL memory address range, appropriate action must be taken to unprotect the BSL memory area before programming the protection key. For more details on the electronic fuse see the *MSP430 Memory Programming User's Guide* (SLAU265) at www.ti.com/msp430.

Some JTAG commands are still possible after the device is secured, including the BYPASS command (see IEEE1149-2001 Standard) and the JMB_EXCHANGE command which allows access to the JTAG Mailbox System (see Section 1.13 for details).

NOTE: If a device has its JTAG protected, Texas Instruments cannot access the device for a customer return. Access is only possible if a BSL is provided with its corresponding key or an unlock mechanism is provided by the customer.

1.13 JTAG Mailbox (JMB) System

The SYS module provides the capability to exchange user data via the regular JTAG test/debug interface. The idea behind the JMB 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 JMB are:

- Providing entry password for device lock/unlock protection
- Run-time data exchange (RTDX)

1.13.1 JMB Configuration

The JMB supports two transfer modes - 16-bit and 32-bit. Setting JMBMODE enables 32-bit transfer mode. Clearing JMBMODE enables 16-bit transfer mode.

1.13.2 JMBOUT0 and JMBOUT1 Outgoing Mailbox

Two 16-bit registers are available for outgoing messages to the JTAG port. JMBOUT0 is only used when using 16-bit transfer mode (JMBMODE = 0). JMBOUT1 is used in addition to JMBOUT0 when using 32-bit transfer mode (JMBMODE = 1). When the application wishes to send a message to the JTAG port, it writes data to JMBOUT0 for 16-bit mode, or JMBOUT0 and JMBOUT1 for 32-bit mode.

JMBOUT0FG and JMBOUT1FG are read only flags that indicate the status of JMBOUT0 and JMBOUT1, respectively. When JMBOUT0FG is set, JMBOUT0 has been read by the JTAG port and is ready to receive new data. When JMBOUT0FG is reset, the JMBOUT0 is not ready to receive new data. JMBOUT1FG behaves similarly.

1.13.3 JMBIN0 and JMBIN1 Incoming Mailbox

Two 16-bit registers are available for incoming messages from the JTAG port. Only JMBIN0 is used when in 16-bit transfer mode (JMBMODE = 0). JMBIN1 is used in addition to JMBIN0 when using 32-bit transfer mode (JMBMODE = 1). When the JTAG port wishes to send a message to the application, it writes data to JMBIN0 for 16-bit mode, or JMBIN0 and JMBIN1 for 32-bit mode.

JMBIN0FG and JMBIN1FG are flags that indicate the status of JMBIN0 and JMBIN1, respectively. When JMBIN0FG is set, JMBIN0 has data that is available for reading. When JMBIN0FG is reset, no new data is available in JMBIN0. JMBIN1FG behaves similarly.

JMBIN0FG and JMBIN1FG can be configured to clear automatically by clearing JMBCLR0OFF and JMBCLR1OFF, respectively. Otherwise, these flags must be cleared by software.

1.13.4 JMB NMI Usage

The JMB handshake mechanism can be configured to use interrupts to avoid unnecessary polling if desired. In 16-bit mode, JMBOUTIFG is set when JMBOUT0 has been read by the JTAG port and is ready to receive data. In 32-bit mode, JMBOUTIFG is set when both JMBOUT0 and JMBOUT1 has been read by the JTAG port and are ready to receive data. If JMBOUTIE is set, these events cause a system NMI. In 16-bit mode, JMBOUTIFG is cleared automatically when data is written to JMBOUT0. In 32-bit mode, JMBOUTIFG Is cleared automatically when data is written to both JMBOUT0 and JMBOUT1. In addition, the JMBOUTIFG can be cleared when reading SYSSNIV. Clearing JMBOUTIE disables the NMI interrupt.

In 16-bit mode, JMBINIFG is set when JMBIN0 is available for reading. In 32-bit mode, JMBINIFG is set when both JMBIN0 and JMBIN1 are available for reading. If JMBOUTIE is set, these events cause a system NMI. In 16-bit mode, JMBINIFG is cleared automatically when JMBIN0 is read. In 32-bit mode, JMBINIFG Is cleared automatically when both JMBIN0 and JMBIN1 are read. In addition, the JMBINIFG can be cleared when reading SYSSNIV. Clearing JMBINIE disables the NMI interrupt.

1.14 Device Descriptor Table

Each device provides a data structure in memory that allows an unambiguous identification of the device. The validity of the device descriptor can be verified by cyclic redundancy check (CRC).Figure 1-6 shows the logical order and structure of the device descriptor table. The complete device descriptor table and its contents can be found in the device specific data sheet.

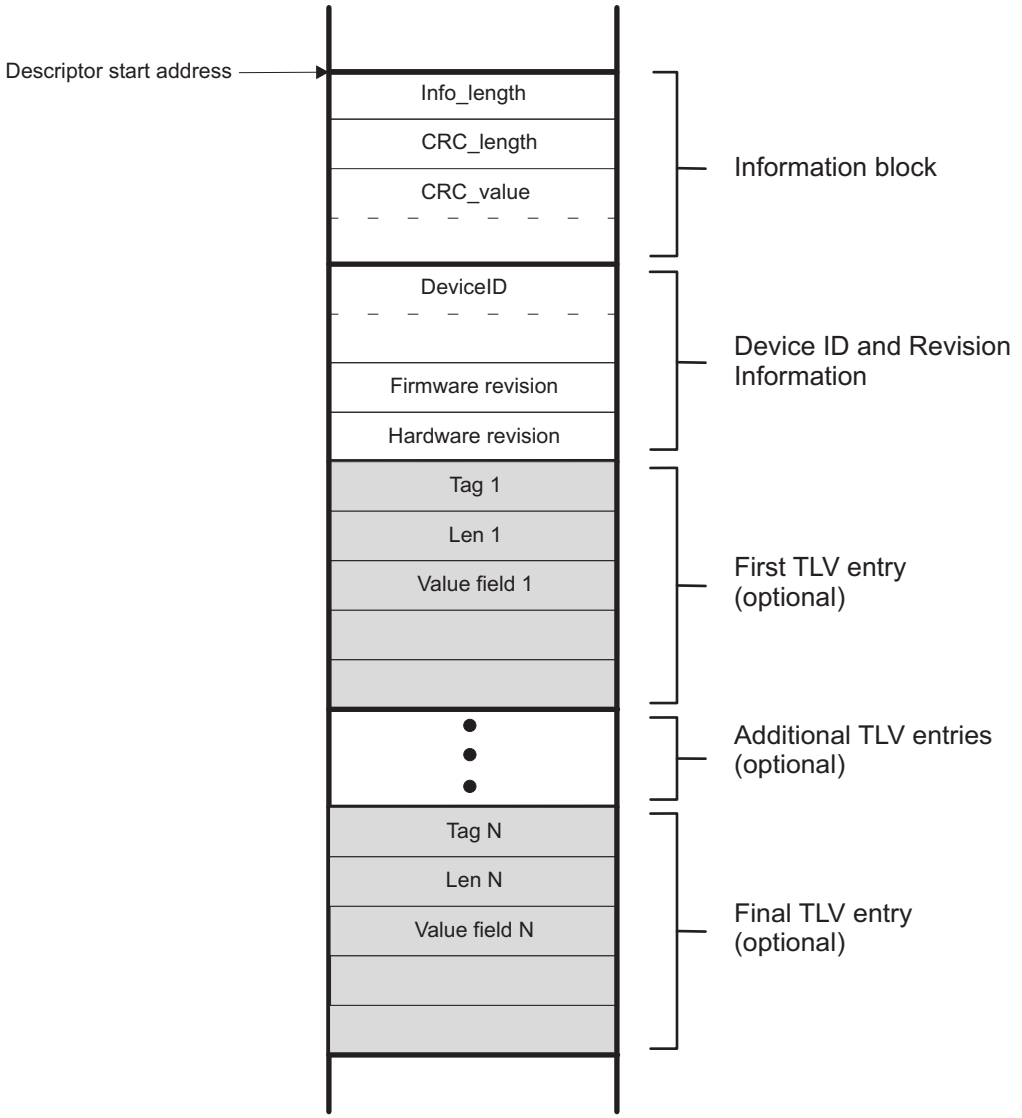


Figure 1-6. Devices Descriptor Table

1.14.1 Identifying 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 information block and a TLV tag-length-value (TLV) structure containing the various descriptors. 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. The information block, shown in Figure 1-6 contains the device ID, die revisions, firmware revisions, and other manufacturer and tool related information.

The length of the descriptors represented by Info_length is computed as follows:

Length = 2^{Info_length} in 32-bit words (1)

For example, if Info_length = 5, then the length of the descriptors equals 128 bytes.

1.14.2 TLV Descriptors

The TLV descriptors follow the information block. Because the information block is always a fixed length, the start location of the TLV descriptors is fixed for a given device family. For the MSP430FR57xx family, this location is 01A08h. See the device-specific data sheet for the complete TLV structure and what descriptors are available.

The TLV descriptors are unique to their respective TLV block and are always followed by the descriptor descriptor block length.

Each TLV descriptor contains a tag field which identifies the descriptor type. Table 1-3 shows the currently supported tags.

Table 1-3. Tag Values

Short Name	Value	Description
LDTAG	01h	Legacy descriptor (1xx, 2xx, 4xx families)
PDTAG	02h	Peripheral discovery descriptor
Reserved	03h	Reserved for future use
Reserved	04h	Reserved for future use
BLANK	05h	Blank descriptor
Reserved	06h	Reserved for future use
Reserved	07h	Reserved for future use
Reserved	08h	Unique Die Record
Reserved	09h - 0Fh	Reserved for future use
Reserved	10h	Reserved
ADC12CAL	11h	ADC12 calibration
REFCAL	12h	REF calibration
ADC10CAL	13h	ADC10 calibration
Reserved	14h	Reserved for future use
Reserved	16h - FDh	Reserved for future use
TAGEXT	FEh	Tag extender

Each tag field is unique to its respective descriptor and is always followed by a length field. The length field is one byte if the tag value is 01h through 0FDh and represents the length of the descriptor in bytes. If the tag value equals 0FEh (TAGEXT), the next byte extends the tag values, and the following two bytes represent the length of the descriptor in bytes. In this way, a user can search through the TLV descriptor table for a particular tag value, using a routine similar to below written in pseudo code:

```
// Identify the descriptor ID (d_ID_value) for the TLV descriptor of interest:
descriptor_address = TLV_START address;

while ( value at descriptor_address != d_ID_value && descriptor_address != TLV_TAGEND &&
descriptor_address < TLV_END)
{
  // Point to next descriptor
  descriptor_address = descriptor_address + (length of the current TLV block) + 2;
}

if (value at descriptor_address == d_ID_value) {
  // Appropriate TLV descriptor has been found!
  Return length of descriptor & descriptor_address as the location of the TLV descriptor
} else {
  // No TLV descriptor found with a matching d_ID_value
  Return a failing condition
}
```

1.14.3 Calibration Values

The TLV structure contains calibration values that can be used to improve the measurement capability of various functions. The calibration values available on a given device are shown in the TLV structure of the device-specific data sheet.

1.14.3.1 REF Calibration

The calibration data for the REF module consists of three words, one word for each reference voltage available (1.5 V, 2.0 V, and 2.5 V). The reference voltages are measured at room temperature. The measured values are normalized by 1.5 V, 2.0 V, or 2.5 V before being stored into the TLV structure, as shown in Equation 2:

$$CAL\_ADC\_15VREF\_FACTOR = \frac{V_{REF+}}{1.5V} \times 2^{15}$$

$$CAL\_ADC\_20VREF\_FACTOR = \frac{V_{REF+}}{2.0V} \times 2^{15}$$

$$CAL\_ADC\_25VREF\_FACTOR = \frac{V_{REF+}}{2.5V} \times 2^{15} \quad (2)$$

In this way, a conversion result is corrected by multiplying it with the CAL_15VREF_FACTOR (or CAL_20VREF_FACTOR, CAL_25VREF_FACTOR) and dividing the result by 2^{15}as shown below for each of the respective reference voltages:

$$ADC(corrected) = ADC(raw) \times CAL\_ADC15VREF\_FACTOR \times \frac{1}{2^{15}}$$

$$ADC(corrected) = ADC(raw) \times CAL\_ADC20VREF\_FACTOR \times \frac{1}{2^{15}}$$

$$ADC(corrected) = ADC(raw) \times CAL\_ADC25VREF\_FACTOR \times \frac{1}{2^{15}} \quad (3)$$

In the following example, the integrated 1.5V reference voltage is used during a conversion.

- Conversion result: 0x0100 = 256 decimal
- Reference voltage calibration factor (CAL_15VREF_FACTOR) : 0x7BBB

The following steps show how the ADC conversion result can be corrected:

- Multiply the conversion result by 2 (this step simplifies the final division): 0x0100 x 0x0002 = 0x0200
- Multiply the result by CAL_15VREF_FACTOR: 0x200 x 0x7FEE = 0x00F7_7600
- Divide the result by 2^{16}: 0x00F7_7600 / 0x0001_0000 = 0x0000_00F7 = 247 decimal

1.14.3.2 ADC Offset and Gain Calibration

The offset of the ADC is determined and stored as a twos-complement number in the TLV structure. The offset error correction is done by adding the CAL_ADC_OFFSET to the conversion result.

$$ADC(offset\_corrected) = ADC(raw) + CAL\_ADC\_OFFSET \quad (4)$$

The gain of the ADC12 is calculated by the following equation:

$$CAL\_ADC\_GAIN\_FACTOR = \frac{1}{GAIN} \times 2^{15} \quad (5)$$

The conversion result is gain corrected by multiplying it with the CAL_ADC_GAIN_FACTOR and dividing the result by 2^{15}:

$$ADC(gain\_corrected) = ADC(raw) \times CAL\_ADC\_GAIN\_FACTOR \times \frac{1}{2^{15}} \quad (6)$$

If both gain and offset are corrected, the gain correction is done first:

$$ADC(gain\_corrected) = ADC(raw) \times CAL\_ADC\_GAIN\_FACTOR \times \frac{1}{2^{15}}$$

$$ADC(final) = ADC(gain\_corrected) + CAL\_ADC\_OFFSET \quad (7)$$

1.14.3.3 Temperature Sensor Calibration

The temperature sensor is calibrated using the internal voltage references. Each reference voltage (1.5 V, 2.0 V, or 2.5 V) contains a measured value for two temperatures, 30°C ± 3°C and 85°C ± 3°C and are stored in the TLV structure. The characteristic equation of the temperature sensor voltage, in millivolts is:

$$V_{SENSE} = TC_{SENSOR} \times Temp + V_{SENSOR} \quad (8)$$

The temperature coefficient, TC_{SENSOR}in mV/°C, represents the slope of the equation. V_{SENSOR}, in mV, represents the y-intercept of the equation. Temp, in °C, is the temperature of interest.

The temperature (Temp, °C) can be computed as follows for each of the reference voltages used in the ADC measurement:

$$Temp = \left(ADC(raw) - CAL\_ADC\_15T30\right) \times \left(\frac{85-30}{CAL\_ADC\_15T85 - CAL\_ADC\_15T30}\right) + 30$$

$$Temp = \left(ADC(raw) - CAL\_ADC\_20T30\right) \times \left(\frac{85-30}{CAL\_ADC\_20T85 - CAL\_ADC\_20T30}\right) + 30$$

$$Temp = \left(ADC(raw) - CAL\_ADC\_25T30\right) \times \left(\frac{85-30}{CAL\_ADC\_25T85 - CAL\_ADC\_25T30}\right) + 30 \quad (9)$$

1.15 Special Function Registers (SFRs)

The SFRs are listed in Table 1-5. The base address for the SFRs is listed in Table 1-4. Many of the bits inside the SFRs are described in other chapters throughout the user's guide. These bits are marked with a note and a reference. See the specific chapter of the respective module for details.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 1-4. SFR Base Address

Module	Base Address
SFR	00100h

Table 1-5. Special Function Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Interrupt Enable	SFRIE1	Read/write	Word	00h	0000h
	SFRIE1_L (IE1)	Read/write	Byte	00h	00h
	SFRIE1_H (IE2)	Read/write	Byte	01h	00h
Interrupt Flag	SFRIFG1	Read/write	Word	02h	0082h
	SFRIFG1_L (IFG1)	Read/write	Byte	02h	82h
	SFRIFG1_H (IFG2)	Read/write	Byte	03h	00h
Reset Pin Control	SFRRPCR	Read/write	Word	04h	0000h
	SFRRPCR_L	Read/write	Byte	04h	00h
	SFRRPCR_H	Read/write	Byte	05h	00h

Interrupt Enable Register (SFRIE1)

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**(1)	**NMIIE**	**VMAIE**	**Reserved**	**OFIE**(2)	**WDTIE**(3)
rw-0	rw-0	rw-0	rw-0	rw-0	r0	rw-0	rw-0

Reserved	Bits 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	FRAM 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

(1) See the FRAM Memory Controller chapter for details.
(2) Refer to the *Unified Clock System* chapter for details.
(3) Refer to the *Watchdog Timer* chapter for details.

Interrupt Flag Register (SFRIFG1)

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
JMBOUTIFG	**JMBINIFG**	**Reserved**	**NMIIFG**	**VMAIFG**	**Reserved**	**OFIFG**(1)	**WDTIFG** (2)
rw-(1)	rw-(0)	r0	rw-0	rw-0	r0	rw-(1)	rw-0

Reserved Bits 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 with a new message to the JTAG module by the CPU. When in 32-bit mode (JMBMODE = 1), this bit is cleared automatically when both JMBO0 and JMBO1 have been written with new messages to the JTAG module 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 the JTAG module and is ready for a new message from the CPU. In 32-bit mode (JMBMODE = 1) , JMBO0 and JMBO1 have been received by the JTAG module and are ready for new messages from the CPU.

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 the JTAG module. In 32-bit mode (JMBMODE = 1) when JMBI0 and JMBI1 have been written by the JTAG module.

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 set or 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

(1) Refer to the *Clock System* chapter for details.
(2) Refer to the *Watchdog Timer* chapter for details.

Reset Pin Control Register (SFRRPCR)

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-1	rw-1	rw-0	rw-0

Reserved	Bits 15-4	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 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

1.16 SYS Configuration Registers

The SYS configuration registers are listed in Table 1-6 and the base address is listed in Table 1-6. 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 configuration registers.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 1-6. SYS Base Address

Module	Base Address
SYS	00180h

Table 1-7. SYS Configuration Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
System Control	SYSCTL	Read/write	Word	00h	0000h
	SYSCTL_L	Read/write	Byte	00h	00h
	SYSCTL_H	Read/write	Byte	01h	00h
JTAG Mailbox Control	SYSJMBC	Read/write	Word	06h	0000h
	SYSJMBC_L	Read/write	Byte	06h	00h
	SYSJMBC_H	Read/write	Byte	07h	00h
JTAG Mailbox Input 0	SYSJMBI0	Read/write	Word	08h	0000h
	SYSJMBI0_L	Read/write	Byte	08h	00h
	SYSJMBI0_H	Read/write	Byte	09h	00h
JTAG Mailbox Input 1	SYSJMBI1	Read/write	Word	0Ah	0000h
	SYSJMBI1_L	Read/write	Byte	0Ah	00h
	SYSJMBI1_H	Read/write	Byte	0Bh	00h
JTAG Mailbox Output 0	SYSJMBO0	Read/write	Word	0Ch	0000h
	SYSJMBO0_L	Read/write	Byte	0Ch	00h
	SYSJMBO0_H	Read/write	Byte	0Dh	00h
JTAG Mailbox Output 1	SYSJMBO1	Read/write	Word	0Eh	0000h
	SYSJMBO1_L	Read/write	Byte	0Eh	00h
	SYSJMBO1_H	Read/write	Byte	0Fh	00h
User NMI Vector Generator	SYSUNIV	Read	Word	1Ah	0000h
System NMI Vector Generator	SYSSNIV	Read	Word	1Ch	0000h
Reset Vector Generator	SYSRSTIV	Read	Word	1Eh	0002h

SYS Control Register (SYSCTL)

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
Reserved	**Reserved**	**SYSJTAGPIN**	**SYSBSLIND**	**Reserved**	**SYSPMMPE**	**Reserved**	**SYSRIVECT**
r0	r0	rw-[0]	r-0	r0	rw-[0]	r0	rw-[0]

Reserved	Bits 15-6	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 remains 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	BSL entry indication. This bit indicates a BSL entry sequence detected on the Spy-Bi-Wire pins. 0 No BSL entry sequence detected 1 BSL entry sequence detected
Reserved	Bit 3	Reserved. Reads back 0.
SYSPMMPE	Bit 2	PMM access protect. This controls the accessibility of the PMM control registers. Once set to 1, it only can be cleared by a BOR. 0 Access from anywhere in memory 1 Access only from the 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 FRAM FFFFh 1 Interrupt vectors generated with end address TOP of RAM, when RAM available.

JTAG Mailbox Control Register (SYSJMBC)

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
JMBCLR1OFF	**JMBCLR0OFF**	**Reserved**	**JMBM0DE**	**JMBOUT1FG**	**JMBOUT0FG**	**JMBIN1FG**	**JMBIN0FG**
rw-(0)	rw-(0)	r0	rw-0	r-(1)	r-(1)	rw-(0)	rw-(0)

Reserved Bits 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 defines 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.

JTAG Mailbox Input 0 Register (SYSJMBI0)
JTAG Mailbox Input 1 Register (SYSJMBI1)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	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 Bits 15-8 JTAG mailbox incoming message high byte

MSGLO Bits 7-0 JTAG mailbox incoming message low byte

JTAG Mailbox Output 0 Register (SYSJMBO0)
JTAG Mailbox Output 1 Register (SYSJMBO1)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	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	Bits 15-8	JTAG mailbox outgoing message high byte
MSGLO	Bits 7-0	JTAG mailbox outgoing message low byte

User NMI Vector Register (SYSUNIV)

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**	**SYSUNVEC**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

SYSUNIV Bits 15-0 User NMI vector. Generates a value that can be used as address offset for fast interrupt service routine handling. Writing to this register clears all pending user NMI 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 device in use.

System NMI Vector Register (SYSSNIV)

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 Bits 15-0 System NMI vector. Generates a value that can be used as address offset for fast interrupt service routine handling. Writing to this register clears all pending system NMI flags.

Value	Interrupt Type
0000h	No interrupt pending
0002h	FRAM double bit error (highest priority)
0004h	Access time error interrupt pending
0006h	Reserved for future extensions
0008h	Reserved for future extensions
000Ah	Reserved for future extensions
000Ch	Reserved for future extensions
000Eh	Access violation
0010h	VMAIFG interrupt pending
0012h	JMBINIFG interrupt pending
0014h	JMBOUTIFG interrupt pending
0016h	FRAM single bit error
0018h - 01Eh	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.

Reset Interrupt Vector Register (SYSRSTIV)

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**	**SYSRSTVEC**					**0**
r0	r0	r-0	r-0	r-0	r-0	r-1	r0

SYSRSTIV Bits 15-0 Reset interrupt vector. Generates a value that can be used as address offset for fast interrupt service routine handling to identify the last cause of a 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	$\overline{\text{RST}}$/NMI (BOR)
0006h	PMMSWBOR (BOR)
0008h	Wakeup from LPMx.5 (BOR)
000Ah	Security violation (BOR)
000Ch	SVSL (POR)
000Eh	SVSH (POR)
0010h	Reserved
0012h	Reserved
0014h	PMMSWPOR (POR)
0016h	WDT time out (PUC)
0018h	WDT password violation (PUC)
001Ah	FRAM password violation (PUC)
001Ch	FRAM double bit error (PUC)
001Eh	PERF peripheral/configuration area fetch (PUC)
0020h	PMM password violation (PUC)
0022h	MPU password violation (PUC)
0024h	CS password violation (PUC)
0026h	Information memory segment violation (PUC)
0028h	Segment 1 memory violation (PUC)
002Ah	Segment 2 memory violation (PUC)
002Ch	Segment 3 memory violation (PUC)
002Eh	FRAM busy (PUC)
0030h-003Eh	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.

Power Management Module and Supply Voltage Supervisor

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

Topic **Page**

2.1 Power Management Module (PMM) Introduction

PMM features include:

- Wide supply voltage (DV_{CC}) range: 2.0 V to 3.6 V
- Generation of voltage for the device core (V_{CORE})
- Supply voltage supervisor (SVS) for DV_{CC} and V_{CORE}
- Brownout reset (BOR)
- Software accessible power-fail indicators
- I/O protection during power-fail condition

The PMM manages all functions related to the power supply and its supervision for the device. Its primary functions are first to generate a supply voltage for the core logic, and second, provide several mechanisms for the supervision of both the voltage applied to the device (DV_{CC}) and the voltage generated for the core (V_{CORE}).

The PMM uses an integrated low-dropout voltage regulator (LDO) to produce a secondary core voltage (V_{CORE}) from the primary one applied to the device (DV_{CC}). In general, V_{CORE} supplies the CPU, memories, and the digital modules, while DV_{CC} supplies the I/Os and analog modules. The V_{CORE} output is maintained using a dedicated voltage reference. The input or primary side of the regulator is referred to in this chapter as its high side. The output or secondary side is referred to in this chapter as its low side.

The block diagram of the PMM is shown in Figure 2-1.

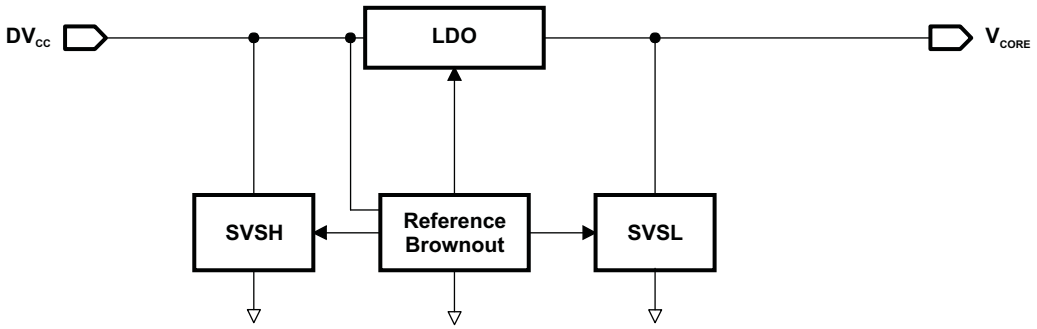


Figure 2-1. PMM Block Diagram

2.2 PMM Operation

2.2.1 V_{CORE} and the Regulator

DV_{CC} can be powered from a wide input voltage range, but the core logic of the device must be kept at a voltage lower than what this range allows. For this reason, a regulator has been integrated into the PMM. The regulator derives the necessary core voltage (V_{CORE}) from DV_{CC}.

The regulator supports two different load settings to optimize power. The high-performance mode is active when:

- The CPU is in active, LPM0, LPM1 or LPM2 modes
- A clock source greater than 100 kHz is used to drive any module
- An interrupt or DMA transfer is executed
- JTAG is active

Otherwise, the low-power mode is used. The hardware controls the load settings automatically, according to the criteria above.

2.2.2 Supply Voltage Supervisor

The high-side supervisor (SVSH) and the low-side supervisor(SVSL) oversee DV_{CC} and V_{CORE}, respectively. The high-side supervisor (SVSH) is always active in all power modes. It can be disabled only in LPM4.5 with SVSHE = 0. By default the low-side supervisor (SVSL) is enabled in active mode, LPM0, LPM1, and LPM2. It can be disabled in LPM1 and LPM2 with SVSLE = 0. The SVSL is always disabled in LPM3, LPM3.5, LPM4, and LPM4.5.

2.2.2.1 SVS Thresholds

As Figure 2-2 shows, there is hysteresis built into the supervision thresholds, such that the thresholds in force depend on whether the voltage rail is going up or down.

The behavior of the SVS according to these thresholds is best portrayed graphically. Figure 2-2 shows how the supervisors respond to various supply failure conditions.

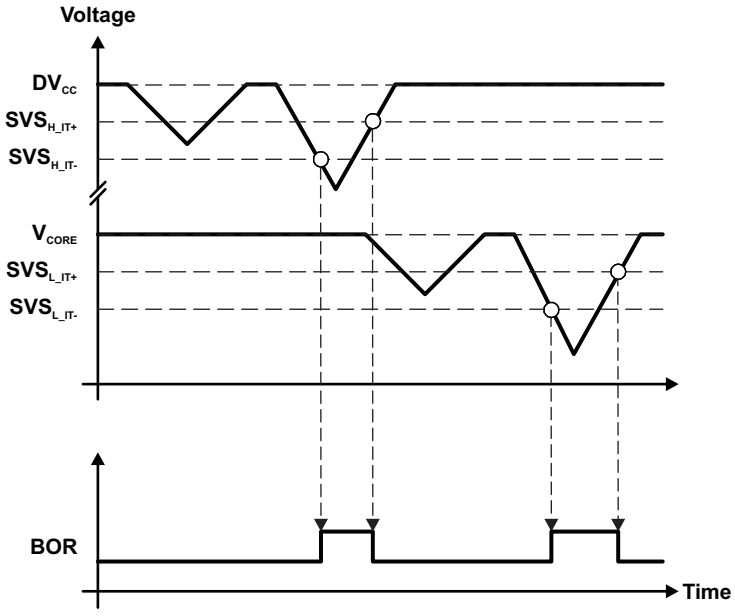


Figure 2-2. High-Side and Low-Side Voltage Failure and Resulting PMM Actions

2.2.3 Supply Voltage Supervisor - Power-Up

When the device is powering up, the SVSH and SVSL functions are enabled by default. Initially, DV_{CC} is low, and therefore the PMM holds the device in BOR reset. Once both the SVSH and SVSL levels are met, the reset is released. Figure 2-3 shows this process.

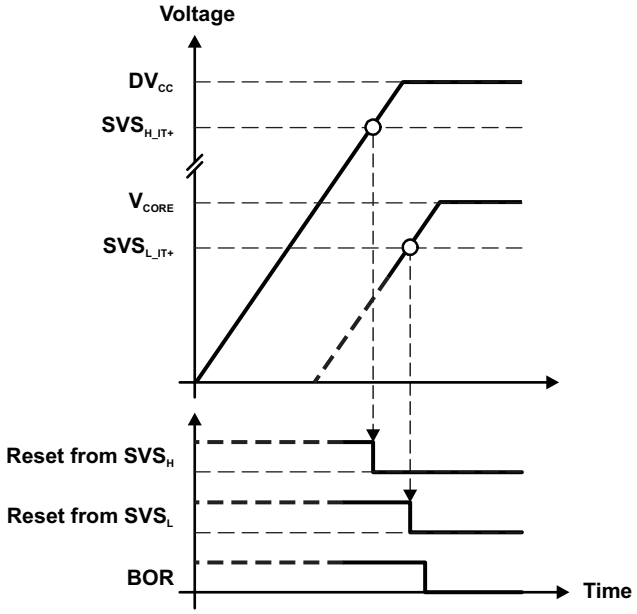


Figure 2-3. PMM Action at Device Power-Up

After this point, both voltage domains are supervised while the respective modules are enabled.

2.2.4 LPM3.5, LPM4.5

LPM3.5 and LPM4.5 are additional low-power modes in which the regulator of the PMM is completely disabled, providing additional power savings. Because there is no power supplied to V_{CORE} during LPMx.5, the CPU and all digital modules including RAM are unpowered. This essentially disables the entire device and, as a result, the contents of the registers and RAM are lost. Any essential values should be stored to FRAM prior to entering LPMx.5. See the SYS module for complete descriptions and usages of LPMx.5.

2.2.5 Brownout Reset (BOR)

The primary function of the brownout reset (BOR) circuit occurs when the device is powering up. It is functional very early in the power-up ramp, generating a BOR that initializes the system.

2.2.6 $\overline{RST}$/NMI

The external $\overline{RST}$/NMI terminal is pulled low on a BOR reset condition. The $\overline{RST}$/NMI can be used as reset source for the rest of the application.

2.2.7 PMM Interrupts

Interrupt flags generated by the PMM are routed to the system NMI interrupt vector generator register, SYSSNIV. When the PMM causes a reset, a value is generated in the system reset interrupt vector generator register, SYSRSTIV, corresponding to the source of the reset. These registers are defined within the SYS module. More information on the relationship between the PMM and SYS modules is available in the SYS chapter.

2.2.8 Port I/O Control

The PMM provides a means of ensuring that I/O pins cannot behave in uncontrolled fashion during an undervoltage event. During these times, outputs are disabled, both normal drive and the weak pullup/pulldown function. If the CPU is functioning normally, and then an undervoltage event occurs, any pin configured as an input has its PxIN register value locked in at the point the event occurs, until voltage is restored. During the undervoltage event, external voltage changes on the pin are not registered internally. This helps prevent erratic behavior from occurring.

2.3 PMM Registers

The PMM registers are listed in Table 2-1. The base address of the PMM module can be found in the device-specific data sheet. The address offset of each PMM register is given in Table 2-1. The password defined in the PMMCTL0 register controls access to all PMM registers - except PM5CTL0. PM5CTL0 can be accessed without a password. Once the correct password is written, the write access is enabled (this includes byte access to the PMMCTL0 lower byte). 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.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 2-1. PMM Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
PMM control register 0	PMMCTL0	Read/write	Word	00h	9660h
	PMMCTL0_L	Read/write	Byte	00h	60h
	PMMCTL0_H	Read/write	Byte	01h	96h
PMM interrupt flag register	PMMIFG	Read/write	Word	0Ah	0000h
	PMMIFG_L	Read/write	Byte	0Ah	00h
	PMMIFG_H	Read/write	Byte	0Bh	00h
Power mode 5 control register 0	PM5CTL0	Read/write	Word	10h	0000h
	PM5CTL0_L	Read/write	Byte	10h	00h
	PM5CTL0_H	Read/write	Byte	11h	00h

Power Management Module Control Register 0 (PMMCTL0)

15	14	13	12	11	10	9	8
PMMPW, Read as 96h, Must be written as A5h							
rw-1	rw-0	rw-0	rw-1	rw-0	rw-1	rw-1	rw-0
7	6	5	4	3	2	1	0
Reserved	**SVSHE**	**SVSLE**	**PMMREGOFF**	**PMMSWPOR**	**PMMSWBOR**	**Reserved**	**Reserved**
r0	rw-[1]	rw-[1]	rw-[0]	rw-(0)	rw-[0]	r0	rw-{0}

PMMPW	Bits 15-8	PMM password. Always read as 096h. Must be written with 0A5h to unlock the PMM registers.
Reserved	Bit 7	Reserved. Always read 0.
SVSHE	Bit 6	High-side SVS enable. 0 High-side SVS (SVSH) is disabled in LPM4.5. SVSH is always enabled in active mode and LPM0/1/2/3/4 and LPM3.5. 1 SVSH is always enabled.
SVSLE	Bit 5	Low-side SVS enable. 0 Low-side SVS (SVSL) is disabled in low power modes. SVSL is always enabled in active mode and LPM0. 1 SVSL is enabled in LPM0/1/2. SVSL is always enabled in AM and always disabled in LPM3/4 and LPM3.5/4.5.
PMMREGOFF	Bit 4	Regulator off 0 Regulator remains on when going into LPM3/4 1 Regulator is turned off when going to LPM3/4. System enters LPM3.5 or LPM4.5, respectively.
PMMSWPOR	Bit 3	Software POR. Setting this bit to 1 triggers a POR. This bit is self clearing.
PMMSWBOR	Bit 2	Software brownout reset. Setting this bit to 1 triggers a BOR. This bit is self clearing.
Reserved	Bit 1	Reserved. Always read 0.
Reserved	Bit 0	Reserved for future usage. Must be written with 0.

Power Management Module Interrupt Flag Register (PMMIFG)

15	14	13	12	11	10	9	8
PMMLPM5IFG	**Reserved**	**SVSHIFG**	**SVSLIFG**	**Reserved**	**PMMPORIFG**	**PMMRSTIFG**	**PMMBORIFG**
rw-{0}(1)	r0	rw-{0}(1)	rw-{0}(1)	r0	rw-[0](1)	rw-{0}(1)	rw-{0}(1)
7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**
r0	r0	r0	r0	r0	r0	r0	r0

PMMLPM5IFG Bit 15 LPMx.5 flag. This bit is only set if the system was in LPMx.5 before. The bit is cleared by software or by reading the reset vector word. A power failure on the DV_{CC} domain triggered by the high-side SVS (if enabled) or the brownout clears the bit.

0 Reset not due to wake-up from LPMx.5

1 Reset due to wake-up from LPMx.5

Reserved Bit 14 Reserved. Always read 0.

SVSHIFG Bit 13 High-side SVS interrupt flag. This interrupt flag is only set if the SVSH is the reset source i.e. DVCC dropped below the high-side SVS levels but remained above the brownout levels. The bit is cleared by software or by reading the reset vector word.

0 Reset not due to SVSH

1 Reset due to SVSH

SVSLIFG Bit 12 Low-side SVS interrupt flag. This interrupt flag is only set if the SVSL is the reset source i.e. the core voltage dropped below the low-side SVS levels but DVCC remained above the SVSH levels. The bit is cleared by software or by reading the reset vector word.

0 Reset not due to SVSL

1 Reset due to SVSL

Reserved Bit 11 Reserved. Always read 0.

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

0 Reset not due to SWPOR

1 Reset due to SWPOR

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

0 Reset not due to reset pin

1 Reset due to reset pin

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

0 Reset not due to SWBOR

1 Reset due to SWBOR

Reserved Bits 7-0 Reserved. Always read 0.

(1) This bit indicates a specific reset condition. Refer to bit description concerning reset conditions.

Power Mode 5 Control Register 0 (PM5CTL0)

15	14	13	12	11	10	9	8
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**	**Reserved**	**Reserved**	**Reserved**	**LOCKLPM5**
r0	r0	r0	r0	r0	r0	r0	rw-{0}(1)

Reserved Bits 15-1 Reserved. Always read as zero.

LOCKLPM5 Bit 0 Lock I/O pin and other LPMx.5 relevant (e.g. RTC) configurations upon entry/exit to/from LPMx.5. Once power is applied to the device, this bit, once set, can only be cleared by the user or via another power cycle.

0 LPMx.5 configuration is not locked and defaults to its reset condition.

1 LPMx.5 configuration remains locked. Pin state is held during LPMx.5 entry and exit.

(1) This bit is reset by a power cycle i.e. SVSH (if enabled) or brownout triggered a reset.

Clock System (CS)

This chapter describes the operation of the clock system, which is implemented in all devices.

Topic **Page**

3.1 Clock System Introduction

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

The 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 24 MHz range. When optional XT2 is present (see below), the XT1 high-frequency mode may or may not be available, depending on the device configuration. See the device-specific data sheet for supported functions.
- VLOCLK: Internal very-low-power low-frequency oscillator with 10-kHz typical frequency
- DCOCLK: Internal digitally controlled oscillator (DCO) with three selectable fixed frequencies
- XT2CLK: Optional high-frequency oscillator that can be used with standard crystals, resonators, or external clock sources in the 4 MHz to 24 MHz range. See device-specific data sheet for availability.

Four system clock signals are available from the clock module:

- ACLK: Auxiliary clock. The ACLK is software selectable as XT1CLK, VLOCLK, DCOCLK, and when available, XT2CLK. ACLK can be divided by 1, 2, 4, 8, 16, or 32. ACLK is software selectable by individual peripheral modules.
- MCLK: Master clock. MCLK is software selectable as XT1CLK, VLOCLK, DCOCLK, and when available, XT2CLK. MCLK can be divided by 1, 2, 4, 8, 16, or 32. MCLK is used by the CPU and system.
- SMCLK: Subsystem master clock. SMCLK is software selectable as XT1CLK, VLOCLK, DCOCLK, and when available, XT2CLK. SMCLK is software selectable by individual peripheral modules.
- MODCLK: Module clock. MODCLK is used by various peripheral modules and is sourced by MODOSC.

The block diagram of the clock system module is shown in Figure 3-1.

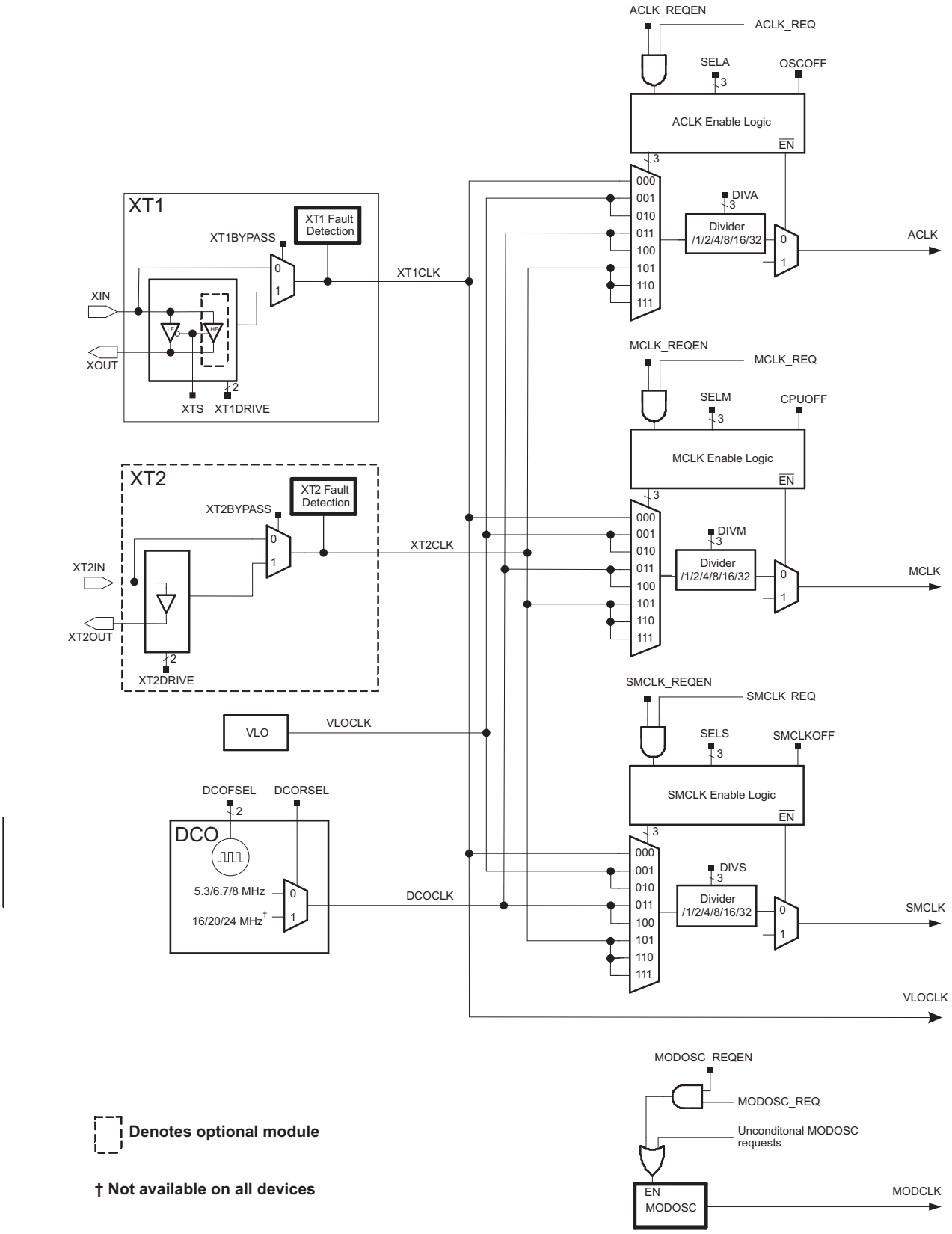


Figure 3-1. Clock System Block Diagram

3.2 Clock System Operation

After PUC, the CS module default configuration is:

- XT1 in low frequency (LF) mode (XTS = 0) is selected as the oscillator source for XT1CLK. XT1CLK is selected for ACLK (SELA = {0}).
- DCOCLK is selected for MCLK and SMCLK (SELM = SELS = {3}) and each are divided by 8 (DIVM = DIVS = {3}).
- XIN and XOUT pins are set to general-purpose I/Os and XT1 remains disabled until the I/O ports are configured for XT1 operation.
- When XT2 is available, XT2IN and XT2OUT pins are set to general-purpose I/Os and XT2 is disabled.

As previously stated, XT1 is selected by default, but XT1 is disabled. The crystal pins (XIN, XOUT) are shared with general-purpose I/Os. To enable XT1, the PSEL bits associated with the crystal pins must be set. When a 32768-Hz crystal is used for XT1CLK, the fault control logic immediately causes ACLK to be sourced by the VLOCLK, because XT1 is not stable immediately (see Section 3.2.7).

Status register control bits (SCG0, SCG1, OSCOFF, and CPUOFF) configure the device operating modes and enable or disable portions of the clock system module (see the *System Resets, Interrupts, and Operating Modes* chapter). Registers CSCTL0 through CSCTL6 configure the CS module.

The CS module can be configured or reconfigured by software at any time during program execution. The CS control registers are password protected to prevent inadvertent access.

3.2.1 CS Module Features for Low-Power Applications

Conflicting requirements typically exist in battery-powered applications:

- Low clock frequency for energy conservation and time keeping
- High clock frequency for fast 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 CS module addresses these 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, DCOCLK, or XT2CLK), giving complete flexibility in the system clock configuration. A flexible clock distribution and divider system is provided to fine-tune the individual clock requirements.

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

The internal VLO provides a typical frequency of 10 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 can be used to source ACLK, MCLK, or SMCLK (SELA = {1} or SELM = {1} or SELS = {1}).

3.2.3 XT1 Oscillator

The XT1 oscillator supports ultra-low-current consumption using a 32768-Hz watch crystal in low-frequency (LF) mode (XTS = 0). The watch crystal connects to XIN and XOUT and requires external capacitors on both terminals. These capacitors should be sized according to the crystal or resonator specifications.

On devices that do not include the optional XT2 oscillator (see Section 3.2.4), the XT1 oscillator also supports high-speed crystals or resonators when in high-frequency (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.

In XT1 LF or HF modes, 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 = 1. When used with an external signal, the external frequency must meet the data sheet 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 remains 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 causes the XIN and XOUT ports to be configured for XT1 operation. If XT1BYPASS is also set, XT1 is configured for bypass mode of operation, and the oscillator associated with XT1 is powered down. In bypass mode of operation, XIN can accept an external clock input signal and XOUT is configured as a general-purpose I/O. The PSEL bit associated with XOUT is a don't care.

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

XT1 is enabled under any of the following conditions:

- XT1 is a source for ACLK (SELA = {0}) and in active mode (AM) through LPM3 (OSCOFF = 0)
- XT1 is a source for MCLK (SELM = {0}) and in active mode (AM) (CPUOFF = 0)
- XT1 is a source for SMCLK (SELS = {0}) and in active mode (AM) through LPM1 (SMCLKOFF = 0)
- XT1OFF = 0. XT1 enabled in active mode (AM) through LPM4.

3.2.4 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. Devices that support XT2 may or may not support XT1 in HF mode; see the device-specific data sheet for availability.

XT2 may be used with external clock signals on the XT2IN pin by setting XT2BYPASS = 1. When used with an external signal, the external frequency must meet the data-sheet 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 remains 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 causes the XT2IN and XT2OUT ports to be configured for XT2 operation. If XT2BYPASS is also set, XT2 is configured for bypass mode of operation, and the oscillator associated with XT2 is powered down. In bypass mode of operation, XT2IN can accept an external clock input signal and XT2OUT is configured as a general-purpose I/O. The PSEL bit associated with XT2OUT is a don't care.

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

XT2 is enabled under any of the following conditions:

- XT2 is a source for ACLK (SELA = {5,6,7}) and in active mode (AM) through LPM3 (OSCOFF = 0)
- XT2 is a source for MCLK (SELM = {5,6,7}) and in active mode (AM) (CPUOFF = 0)
- XT2 is a source for SMCLK (SELS = {5,6,7}) and in active mode (AM) through LPM1 (SMCLKOFF = 0)
- XT2OFF = 0. XT2 enabled in active mode (AM) through LPM4.

3.2.5 Digitally Controlled Oscillator (DCO)

The DCO is an integrated digitally controlled oscillator. The DCO has three frequency settings determined by the DCOFSEL bits. Each frequency is trimmed at the factory. The DCO can be used as a source for ACLK, MCLK, or SMCLK. See the device-specific data sheet for DCO characteristics.

The DCO frequency can be changed at any time, but care should be taken to ensure no other system clock frequency constraints are exceeded with the new frequency selection. Any change in the DCOFSEL or DCORSEL bits causes the DCOCLK to be held for four clock cycles before releasing the new value into the system. This allows for the DCO to settle properly.

3.2.6 Operation From Low-Power Modes, Requested by Peripheral Modules

A peripheral module requests its clock sources automatically from the CS module if required for its proper operation, regardless of the current power mode of operation, as shown in Figure 3-2.

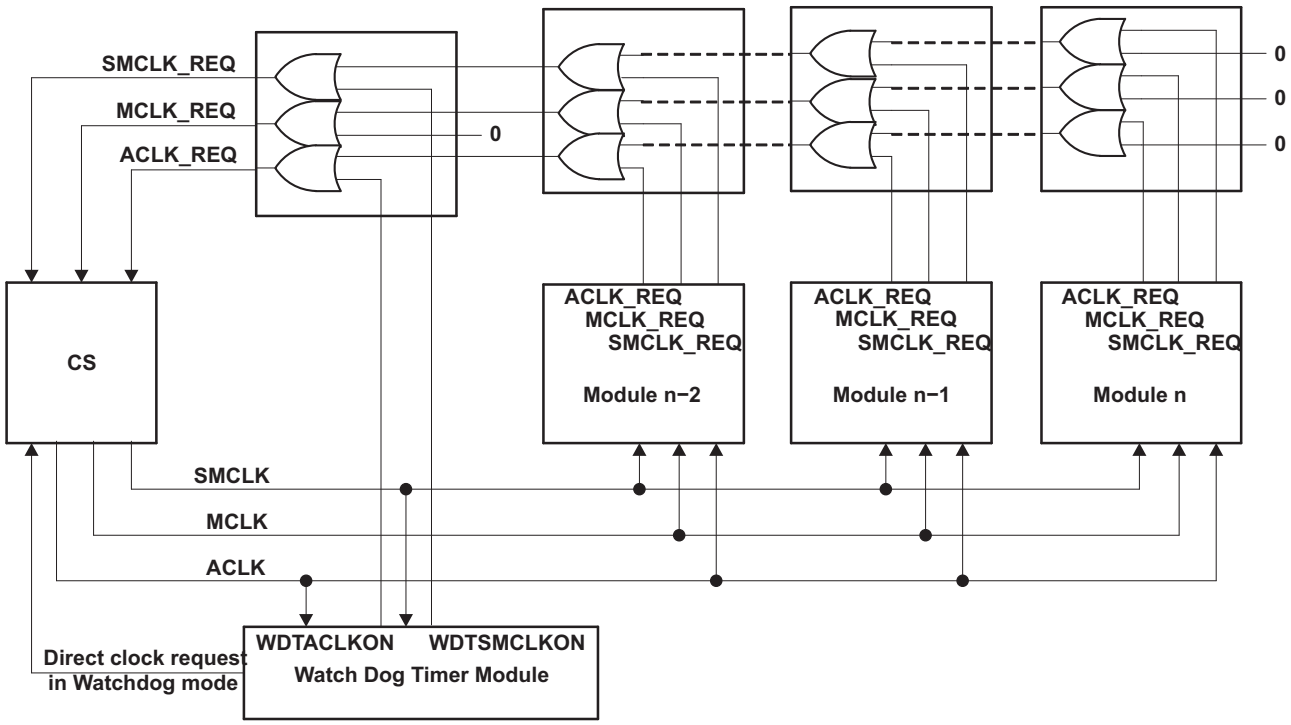


Figure 3-2. Module Request Clock System

A peripheral module asserts one of three possible clock request signals based on its control bits: ACLK_REQ, MCLK_REQ, or SMCLK_REQ. These request signals are based on the configuration and clock selection of the respective module. For example, if a timer selects ACLK as its clock source and the timer is enabled, the timer generates an ACLK_REQ signal to the CS system. The CS, in turn, enables ACLK regardless of the power mode settings.

Any clock request from a peripheral module causes 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 ACLK that is currently disabled by the OSCOFF bit (OSCOFF = 1). The module can request ACLK by generating an ACLK_REQ. This causes the OSCOFF bit to have no effect, thereby allowing ACLK to be available to the requesting peripheral module. The OSCOFF bit remains at its current setting (OSCOFF = 1).

If the requested source is not active, the software NMI handler must manage the required actions. For the previous example, if ACLK was sourced by XT1, and XT1 was not enabled, an oscillator fault condition occurs and the software must handle the event. The watchdog, due to its security requirement, actively selects the VLOCLK source if the originally selected clock source is not available.

Due to the clock request feature, care must be taken in the application when entering low-power modes to save power. Although the device enters the selected low-power mode, a clock request causes more current consumption than the specified values in the data sheet. By default, the clock request feature is enabled. The feature can be disabled for each system clock by clearing ACLKREQEN, MCLKREQEN, or SMCLKREQEN for the respective clocks. This does not disable fail-safe clock requests; for example, those of the watchdog timer or the clock system itself.

The function of the ACLKREQEN, MCLKREQEN, and SMCLKREQEN bits are dependent upon which power mode is selected; that is, they do not have an effect across all power modes. For example, ACLKREQEN is used to enable/disable ACLK requests. It is only effective in LPM4, because in all other modes (AM, LPM0, LPM1, LPM2, LPM3) ACLK is always active. SMCLKREQEN is used to enable/disable SMCLK requests. When SMCLKOFF = 0 and in AM, LPM0, or LPM1, it is a don't care because SMCLK is always on in these cases. For SMCLKOFF = 0 and in LPM2, LPM3, and LPM4, SMCLKREQEN can be used to enable/disable SMCLK requests, because in these modes, SMCLK is normally off. When SMCLKOFF = 1, SMCLKREQEN can be used to enable/disable SMCLK requests, because under this condition SMCLK is normally off in all power modes. This is summarized in Table 3-1.

Table 3-1. System Clocks verus Power Modes and Clock Requests

Mode	System Clocks							
	MCLK		ACLK		SMCLK			
					SMCLKOFF = 0		SMCLKOFF = 1	
	MCLKREQEN = 0 and clock requested	MCLKREQEN = 1 and clock requested	ACLKREQEN = 0 and clock requested	ACLKREQEN = 1 and clock requested	SMCLKREQEN = 0 and clock requested	SMCLKREQEN = 1 and clock requested	SMCLKREQEN = 0 and clock requested	SMCLKREQEN = 1 and clock requested
AM	Active	Active	Active	Active	Active	Active	Disabled	Active
LPM0	Disabled	Active	Active	Active	Active	Active	Disabled	Active
LPM1	Disabled	Active	Active	Active	Active	Active	Disabled	Active
LPM2	Disabled	Active	Active	Active	Disabled	Active	Disabled	Active
LPM3	Disabled	Active	Active	Active	Disabled	Active	Disabled	Active
LPM4	Disabled	Active	Disabled	Active	Disabled	Active	Disabled	Active
LPM3.5	Disabled	Disabled	Disabled[1]	Disabled	Disabled	Disabled	Disabled	Disabled
LPM4.5	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled

(1) LFXTCLK is available directly as the clock source to the RTC module.

3.2.7 CS Module Fail-Safe Operation

The CS module incorporates an oscillator-fault fail-safe feature. This feature detects an oscillator fault for XT1 and XT2 as shown in Figure 3-3. The available fault conditions are:

- Low-frequency oscillator fault (XT1OFFG) for XT1 in LF mode
- High-frequency oscillator fault (XT1OFFG) for XT1 in HF mode
- High-frequency oscillator fault (XT2OFFG) for XT2
- External clock signal faults for all bypass modes; that is, XT1BYPASS = 1 or XT2BYPASS = 1

The crystal oscillator fault bits XT1OFFG and XT2OFFG are set if the corresponding crystal oscillator is turned on and not operating properly. Once set, the fault bits remain set until reset in software, regardless if the fault condition no longer exists. If the user clears the fault bits and the fault condition still exists, the fault bits are automatically set, otherwise they remain cleared.

The OFIFG oscillator-fault interrupt flag is set and latched at POR or when any oscillator fault (XT1OFFG or XT2OFFG) is detected. When OFIFG is set and OFIE is set, the OFIFG requests a user NMI. When the interrupt is granted, the OFIE is not reset automatically as it is 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 XT1 in LF mode is sourcing any system clock (ACLK, MCLK, or SMCLK), and a fault is detected, the system clock is automatically switched to the VLO for its clock source (VLOCLK). Similarly, if XT1 in HF mode is sourcing any system clock and a fault is detected, the system clock is automatically switched to MODOSC for its clock source (MODCLK). When XT2 (if available) is sourcing any system clock and a fault is detected, the system clock is automatically switched to MODOSC for its clock source (MODCLK). The fail-safe logic does not change the respective SELA, SELM, and SELS bit settings. The fail-safe mechanism behaves the same regardless if in normal or bypass modes.

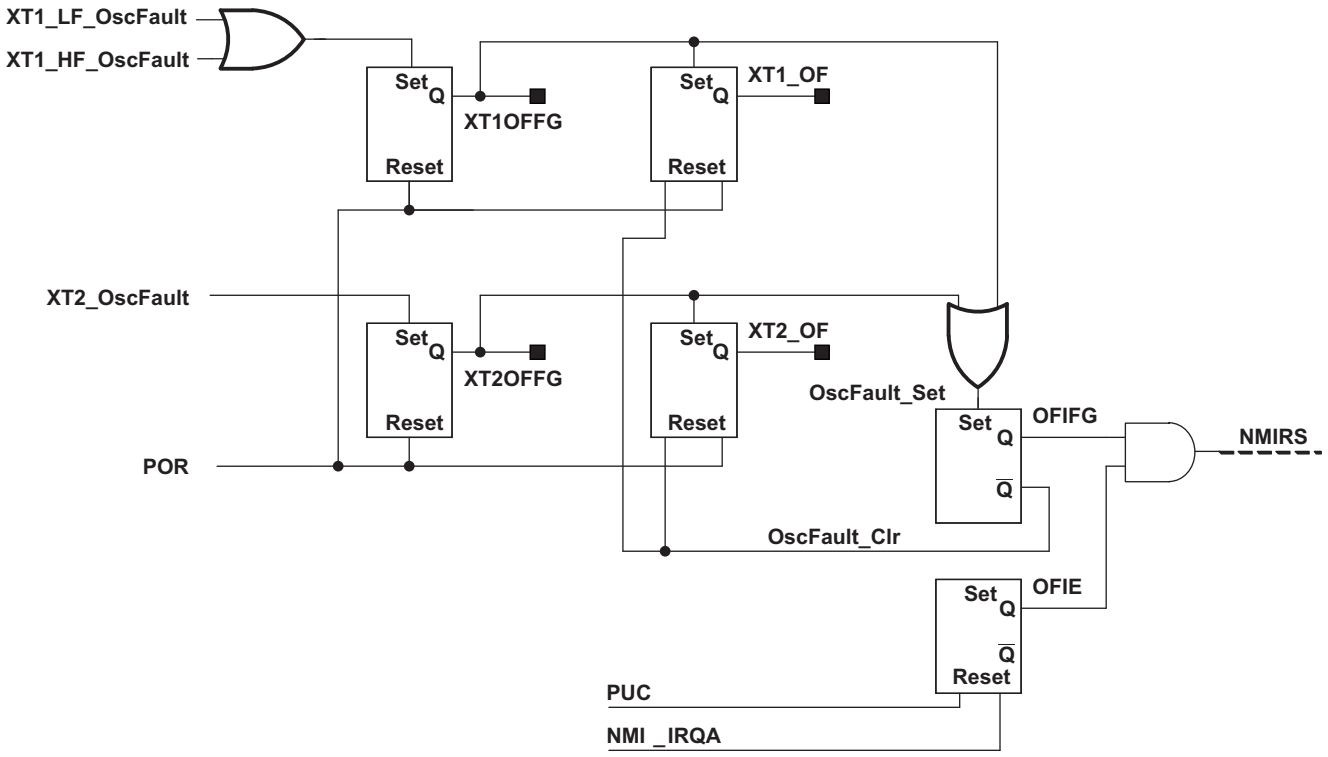


Figure 3-3. Oscillator Fault Logic

NOTE: Fault conditions

XT1_LF_OscFault: This signal is set after the XT1 (LF mode) oscillator has stopped operation and is cleared after operation resumes. The fault condition causes XT1OFFG to be set and remain set. If the user clears XT1OFFG and the fault condition still exists, XT1OFFG remains set.

XT1_HF_OscFault: This signal is set after the XT1 (HF mode) oscillator has stopped operation and is cleared after operation resumes. The fault condition causes XT1OFFG to be set and remain set. If the user clears XT1OFFG and the fault condition still exists, XT1OFFG remains set.

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

NOTE: Fault logic

Note that as long as a fault condition still exists, the OFIFG remains 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 switches back to the original user settings prior to the fault condition.

NOTE: The XT1 startup includes a counter that ensures that 4096 valid clock cycles have passed before XT1_LF_OscFault and XT1_HF_OscFault signals are cleared. A valid cycle is any cycle that meets the frequency requirement ($f_{Fault,LF}$ or $f_{Fault,HF}$) as outlined in the device-specific data sheet. Any crystal fault restarts the counter. It is recommended that the counter always be enabled; however, the counter can be disabled by clearing ENSTFCNT1. Similarly, XT2 startup includes a counter. It can be disabled by clearing ENSTFCNT2. The disabling of the counters is valid for bypass and normal modes of operation.

3.2.8 Synchronization of Clock Signals

When switching ACLK, MCLK, or SMCLK from one clock source to the another, the switch is synchronized to avoid critical race conditions as shown in Figure 3-4:

- 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.

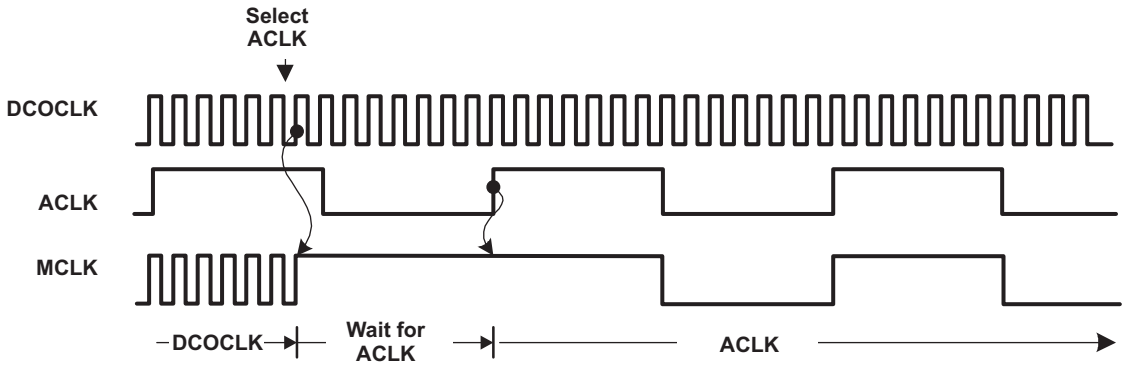


Figure 3-4. Switch MCLK from DCOCLK to XT1CLK

3.3 Module Oscillator (MODOSC)

The CS module also supports an internal oscillator, MODOSC, that is used by the power management module and, optionally, by other modules in the system. It is also used as a fail-safe clock source as described in Section 3.2.7. 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. MODOSC is enabled based on unconditional and conditional requests. Setting MODOSCREQEN enables conditional requests. Unconditional requests are always enabled. It is not necessary to set MODOSCREQEN for modules that use unconditional requests; for example, PMM, ADC, and fail-safe.

The ADC10_A may optionally use MODOSC as a clock source for its conversion clock. The user chooses the ADC10OSC as the conversion clock source. During a conversion, the ADC10_A module issues an unconditional request for the ADC10OSC clock source. Upon doing so, the MODOSC source is enabled, if not already enabled from other modules' previous requests.

3.4 CS Module Registers

The CS module registers are listed in Table 3-2. The base address can be found in the device-specific data sheet. The address offset is listed in Table 3-2. The password defined in CSCTL0 controls access to the CS registers. Once the correct password is written in word mode, write access to the CS registers is enabled. Write access is disabled by writing an incorrect password in byte mode to the CSCTL0 upper byte.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 3-2. Clock System Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Clock System Control 0	CSCTL0	Read/write	Word	00h	9600h
	CSCTL0_L	Read/write	Byte	00h	00h
	CSCTL0_H	Read/write	Byte	01h	96h
Clock System Control 1	CSCTL1	Read/write	Word	02h	0007h
	CSCTL1_L	Read/write	Byte	02h	07h
	CSCTL1_H	Read/write	Byte	03h	00h
Clock System Control 2	CSCTL2	Read/write	Word	04h	0033h
	CSCTL2_L	Read/write	Byte	04h	33h
	CSCTL2_H	Read/write	Byte	05h	00h
Clock System Control 3	CSCTL3	Read/write	Word	06h	0033h
	CSCTL3_L	Read/write	Byte	06h	33h
	CSCTL3_H	Read/write	Byte	07h	00h
Clock System Control 4	CSCTL4	Read/write	Word	08h	C1C1h
	CSCTL4_L	Read/write	Byte	08h	C1h
	CSCTL4_H	Read/write	Byte	09h	C1h
Clock System Control 5	CSCTL5	Read/write	Word	0Ah	0C01h
	CSCTL5_L	Read/write	Byte	0Ah	01h
	CSCTL5_H	Read/write	Byte	0Bh	0Ch
Clock System Control 6	CSCTL6	Read/write	Word	0Ch	0007h
	CSCTL6_L	Read/write	Byte	0Ch	07h
	CSCTL6_H	Read/write	Byte	0Dh	00h

Clock System Control 0 Register (CSCTL0)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
CSKEY, Read as 96h, Must be written as A5h							
rw-1	rw-0	rw-0	rw-1	rw-0	rw-1	rw-1	rw-0
7	6	5	4	3	2	1	0
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0

CSKEY	Bits 15-8	CSKEY password. Must always be written with A5h or PUC is generated. Always reads as 96h. Once corrected password is written, all CS registers are available for writing.
Reserved	Bits 7-0	Reserved. Reads back as 0.

Clock System Control 1 Register (CSCTL1)

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
DCORSEL	**Reserved**				**DCOFSEL**		**Reserved**
rw-[0]	r0	r0	r0	r0	rw-[1]	rw-[1]	r1

Reserved	Bits 15-8	Reserved. Always read as zero.
DCORSEL	Bit 7	DCO range select. For high speed devices, this bit can be written by the user. For low speed devices, it is always reset.
Reserved	Bits 6-4	Reserved. Always read as zero.
DCOFSEL	Bits 2-1	DCO frequency select. For some devices, DCORSEL = 1 setting is not available. See Table 3-3 for settings.
Reserved	Bit 0	Reserved. Always read as one.

Table 3-3. DCO Frequency Selection

DCOFSEL	Nominal DCO frequency, MHz	
	DCORSEL = 0	DCORSEL = 1
00, 10	5.33	16
01	6.67	20
11	8	24

Clock System Control 2 Register (CSCTL2)

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-0	rw-1	rw-1	r0	rw-0	rw-1	rw-1

Reserved	Bits 15-11	Reserved. Reads back as 0.
SELA	Bits 10-8	Selects the ACLK source 000 XT1CLK 001 VLOCLK 010 Reserved for future use. Defaults to VLOCLK. 011 DCOCLK 100 Reserved for future use. Defaults to DCOCLK. 101 XT2CLK when available, otherwise DCOCLK 110 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLK. 111 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLK.
Reserved	Bit 7	Reserved. Reads back as 0.
SELS	Bits 6-4	Selects the SMCLK source 000 XT1CLK 001 VLOCLK 010 Reserved for future use. Defaults to VLOCLK. 011 DCOCLK 100 Reserved for future use. Defaults to DCOCLK. 101 XT2CLK when available, otherwise DCOCLK 110 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLK. 111 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLK.
Reserved	Bit 3	Reserved. Reads back as 0.
SELM	Bits 2-0	Selects the MCLK source 000 XT1CLK 001 VLOCLK 010 Reserved for future use. Defaults to VLOCLK. 011 DCOCLK 100 Reserved for future use. Defaults to DCOCLK. 101 XT2CLK when available, otherwise DCOCLK 110 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLK. 111 Reserved for future use. Defaults to XT2CLK when available, otherwise DCOCLK.

Clock System Control 3 Register (CSCTL3)

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

Reserved	Bits 15-11	Reserved. Reads back as 0.
DIVA	Bits 10-8	ACLK source divider. Divides the frequency of the ACLK clock source.
	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	Bit 7	Reserved. Reads back as 0.
DIVS	Bits 6-4	SMCLK source divider
	000	$f_{SMCLK}/1$
	001	$f_{SMCLK}/2$
	010	$f_{SMCLK}/4$
	011	$f_{SMCLK}/8$
	100	$f_{SMCLK}/16$
	101	$f_{SMCLK}/32$
	110	Reserved for future use. Defaults to $f_{SMCLK}/32$.
	111	Reserved for future use. Defaults to $f_{SMCLK}/32$.
Reserved	Bit 3	Reserved. Reads back as 0.
DIVM	Bits 2-0	MCLK source divider
	000	$f_{MCLK}/1$
	001	$f_{MCLK}/2$
	010	$f_{MCLK}/4$
	011	$f_{MCLK}/8$
	100	$f_{MCLK}/16$
	101	$f_{MCLK}/32$
	110	Reserved for future use. Defaults to $f_{MCLK}/32$.
	111	Reserved for future use. Defaults to $f_{MCLK}/32$.

Clock System Control 4 Register (CSCTL4)

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**	**Reserved**		**SMCLKOFF**	**XT1OFF**
rw-1	rw-1	rw-0	rw-0	r0	r0	rw-0	rw-1

XT2DRIVE Bits 15-14 The XT2 oscillator current can be adjusted to its drive needs.

- 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 Bit 13 Reserved. Reads back as 0.

XT2BYPASS Bit 12 XT2 bypass select

- 0 XT2 sourced internally
- 1 XT2 sourced externally from pin

Reserved Bits 11-9 Reserved. Reads back as 0.

XT2OFF Bit 8 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

XT1DRIVE Bits 7-6 The XT1 oscillator current can be adjusted to its drive needs.

- 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 Bit 5 XT1 mode select

- 0 Low-frequency mode
- 1 High-frequency mode

XT1BYPASS Bit 4 XT1 bypass select

- 0 XT1 sourced internally
- 1 XT1 sourced externally from pin

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

SMCLKOFF Bit 1 SMCLK off. This bit turns off the SMCLK.

- 0 SMCLK on
- 1 SMCLK off

XT1OFF 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

Clock System Control 5 Register (CSCTL5)

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved							
r0	r0	0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
ENSTFCNT2	ENSTFCNT1	Reserved				XT2OFFG[1]	XT1OFFG
rw-(1)	rw-(1)	r0	r0	r0	r0	rw-(0)	rw-(1)

Reserved	Bits 15-8	Reserved. Reads back as 0.
ENSTFCNT2	Bit 7	Enable start counter for XT2 when available. 0 Startup fault counter disabled. Counter is cleared. 1 Startup fault counter enabled
ENSTFCNT1	Bit 6	Enable start counter for XT1. 0 Startup fault counter disabled. Counter is cleared. 1 Startup fault counter enabled
Reserved	Bits 5-2	Reserved. Reads back as 0.
XT2OFFG	Bit 1	XT2 oscillator fault flag. If this bit is set, the OFIFG flag is also set. XT2OFFG is set if a XT2 fault condition exists. XT2OFFG can be cleared via software. If the XT2 fault condition still remains, XT2OFFG is set. 0 No fault condition occurred after the last reset. 1 XT2 fault. An XT2 fault occurred after the last reset.
XT1OFFG	Bit 0	XT1 oscillator fault flag (LF mode). If this bit is set, the OFIFG flag is also set. XT1OFFG is set if a XT1 fault condition exists. XT1OFFG can be cleared via software. If the XT1 fault condition still remains, XT1OFFG is set. 0 No fault condition occurred after the last reset. 1 XT1 fault (LF mode or HF mode). A XT1 fault occurred after the last reset.

(1) On devices without XT2, this flag is read only zero.

Clock System Control 6 Register (CSCTL6)

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				MODCLKREQEN	SMCLKREQEN	MCLKREQEN	ACLKREQEN
r0	r0	r0	r0	rw-(0)	rw-(1)	rw-(1)	rw-(1)

Reserved	Bits 15-4	Reserved. Reads back as 0.
MODCLKREQEN	Bit 3	MODOSC clock request enable. Setting this enables conditional module requests for MODCLK. 0 MODCLK conditional requests are disabled. 1 MODCLK conditional requests are enabled.
SMCLKREQEN	Bit 2	SMCLK clock request enable. Setting this enables conditional module requests for SMCLK 0 SMCLK conditional requests are disabled. 1 SMCLK conditional requests are enabled.
MCLKREQEN	Bit 1	MCLK clock request enable. Setting this enables conditional module requests for MCLK 0 MCLK conditional requests are disabled. 1 MCLK conditional requests are enabled.
ACLKREQEN	Bit 0	ACLK clock request enable. Setting this enables conditional module requests for ACLK 0 ACLK conditional requests are disabled. 1 ACLK conditional requests are enabled.

CPUX

This chapter describes the extended MSP430X 16-bit RISC CPU (CPUX) with 1-MB memory access, its addressing modes, and instruction set.

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

Topic **Page**

4.1 MSP430X CPU (CPUX) 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 backward compatible with the MSP430 CPU.

The MSP430X CPU features include:

- RISC architecture
- Orthogonal architecture
- Full register access including program counter (PC), status register (SR), and stack pointer (SP)
- 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 4-1.

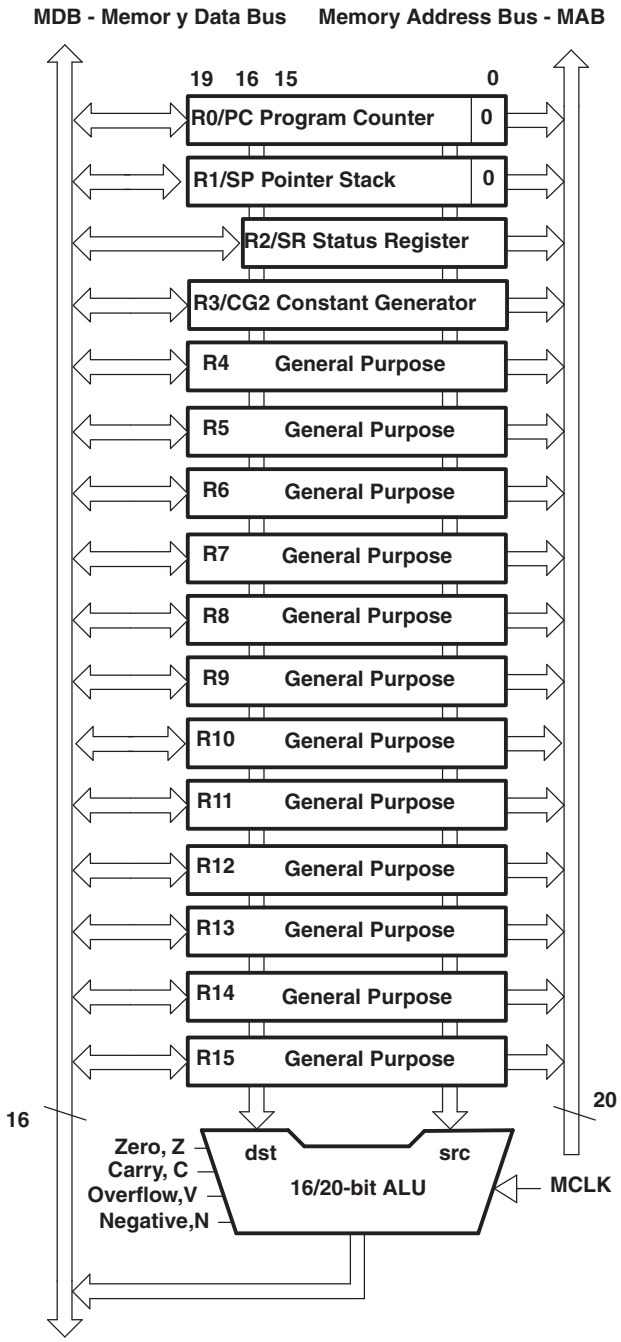


Figure 4-1. MSP430X CPU Block Diagram

4.2 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 (PC) and the status register (SR) are pushed onto the stack as shown in Figure 4-2. 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.

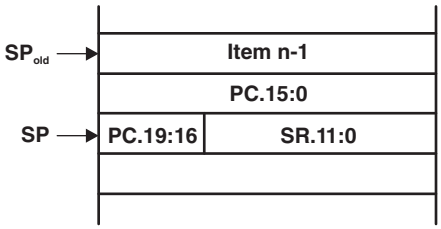


Figure 4-2. PC Storage on the Stack for Interrupts

4.3 CPU Registers

The CPU incorporates 16 registers (R0 through R15). Registers R0, R1, R2, and R3 have dedicated functions. Registers R4 through R15 are working registers for general use.

4.3.1 Program Counter (PC)

The 20-bit PC (PC/R0) points to the next instruction to be executed. Each instruction uses an even number of bytes (2, 4, 6, or 8 bytes), and the PC is incremented accordingly. Instruction accesses are performed on word boundaries, and the PC is aligned to even addresses. Figure 4-3 shows the PC.

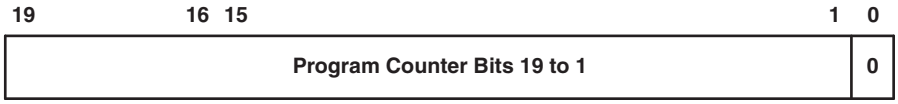


Figure 4-3. Program Counter

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` clears the upper four bits of the PC, because it is a .W instruction.

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

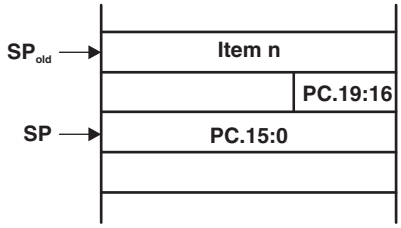


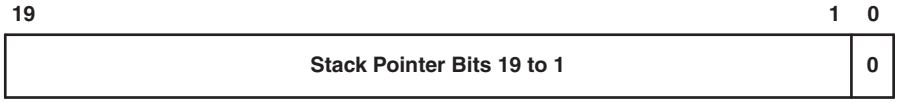
Figure 4-4. PC Storage on the Stack for CALLA

The RETA instruction restores bits 19:0 of the PC and adds 4 to the stack pointer (SP). The RET instruction restores bits 15:0 to the PC and adds 2 to the SP.

4.3.2 Stack Pointer (SP)

The 20-bit SP (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 4-5 shows the SP. The SP is initialized into RAM by the user, and is always aligned to even addresses.

Figure 4-6 shows the stack usage. Figure 4-7 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 4-5. Stack Pointer

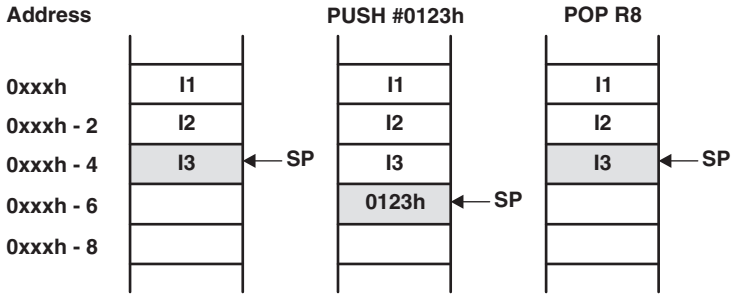


Figure 4-6. Stack Usage

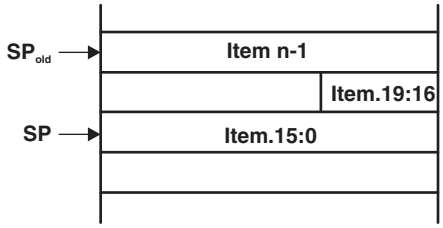


Figure 4-7. PUSHX.A Format on the Stack

The special cases of using the SP as an argument to the PUSH and POP instructions are described and shown in Figure 4-8.

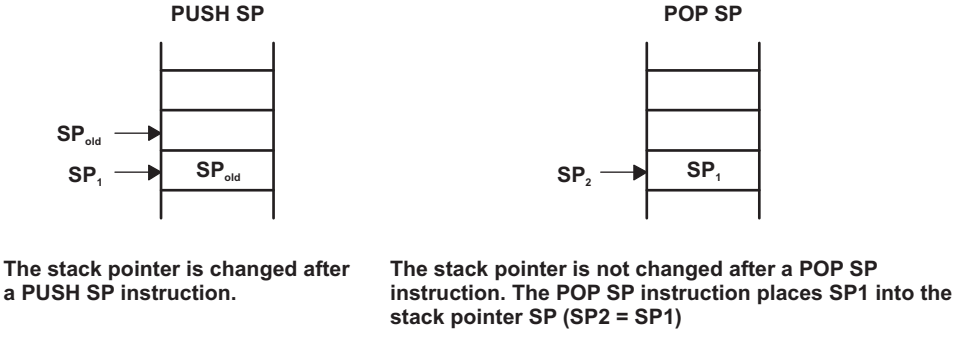


Figure 4-8. PUSH SP, POP SP Sequence

4.3.3 Status Register (SR)

The 16-bit SR (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 4-9 shows the SR bits. Do not write 20-bit values to the SR. Unpredictable operation can result.

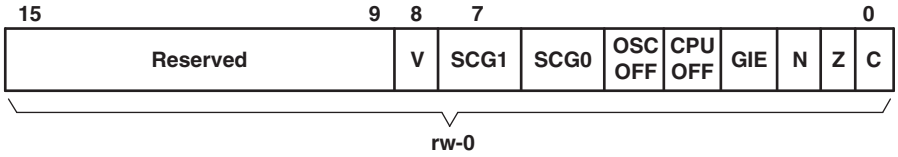


Figure 4-9. SR Bits

Table 4-1 describes the SR bits.

Table 4-1. SR Bit Description

Bit	Description	
Reserved	Reserved	
V	Overflow. 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 may be to enable/disable functions in the clock system depending on the device family; for example, DCO bias enable/disable	
SCG0	System clock generator 0. This bit may be used to enable/disable functions in the clock system depending on the device family; for example, FLL disable/enable	
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. This bit is set when the result of an operation is negative and cleared when the result is positive.	
Z	Zero. This bit is set when the result of an operation is 0 and cleared when the result is not 0.	
C	Carry. This bit is set when the result of an operation produced a carry and cleared when no carry occurred.	

NOTE: Bit manipulations of the SR should be done via the following instructions: *MOV*, *BIS*, and *BIC*.

4.3.4 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 4-2.

Table 4-2. 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.

4.3.4.1 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
```

4.3.5 General-Purpose Registers (R4 –R15)

The 12 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 most significant bits (MSBs) if a register is the destination of a byte or word instruction.

Figure 4-10 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.

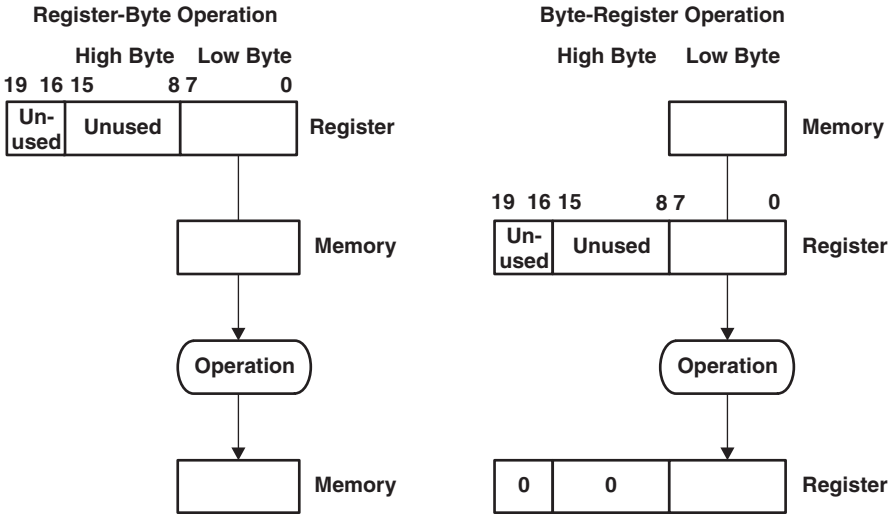


Figure 4-10. Register-Byte/Byte-Register Operation

Figure 4-11 and Figure 4-12 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.

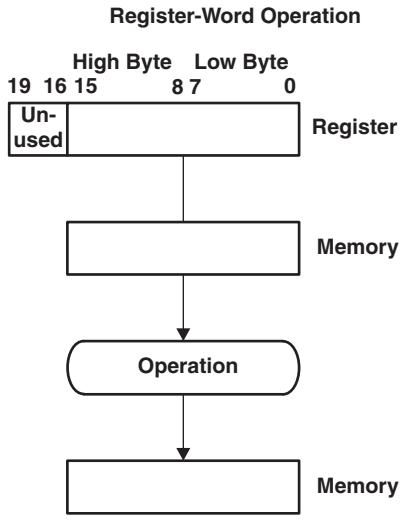


Figure 4-11. Register-Word Operation

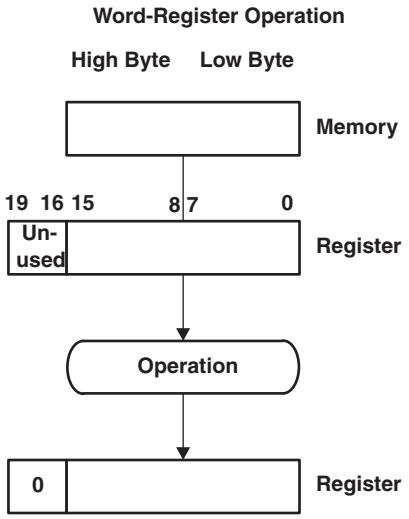


Figure 4-12. Word-Register Operation

Figure 4-13 and Figure 4-14 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.

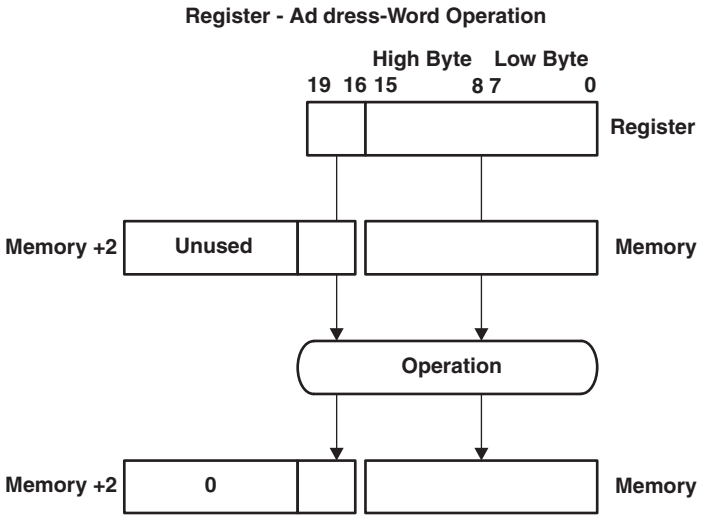


Figure 4-13. Register – Address-Word Operation

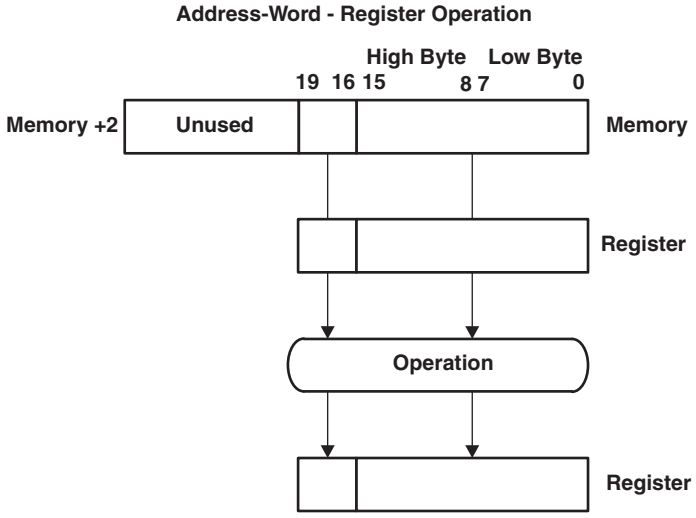


Figure 4-14. Address-Word – Register Operation

4.4 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 4-3). The MSP430 and MSP430X instructions are usable throughout the entire 1-MB memory range.

Table 4-3. Source/Destination Addressing

As/Ad	Addressing Mode	Syntax	Description
00/0	Register	Rn	Register contents are operand.
01/1	Indexed	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	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	&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	@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	#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 and have no special meaning.

4.4.1 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 eight least significant bits (LSBs) of the source register Rsrc and writes the result to the eight 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.

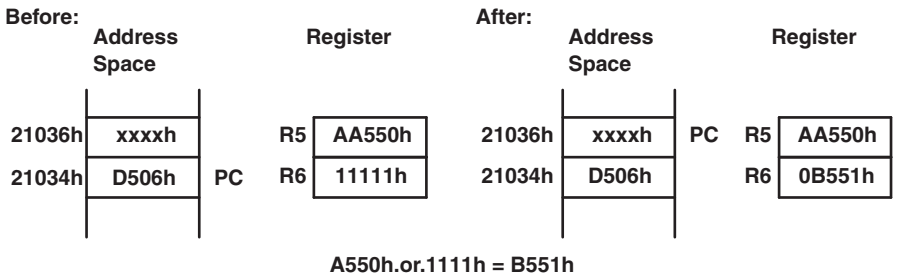
Address-word operation: Address-word operation reads the 20 bits of the source register Rsrc and writes the result to the 20 bits of the destination register Rdst. The register Rsrc is not modified

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.



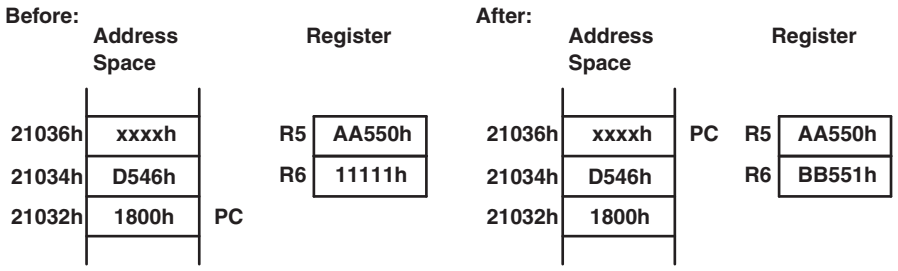
A550h.or.1111h = B551h

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

4.4.2 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

4.4.2.1 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 4-15.

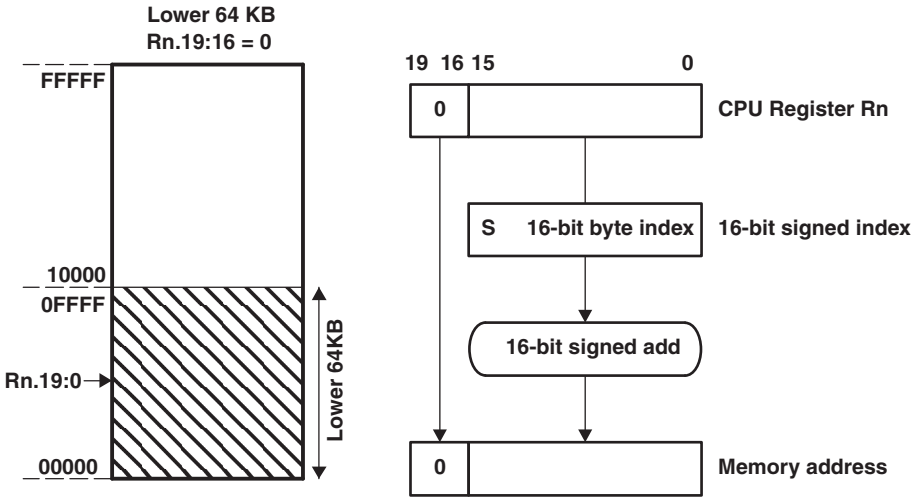
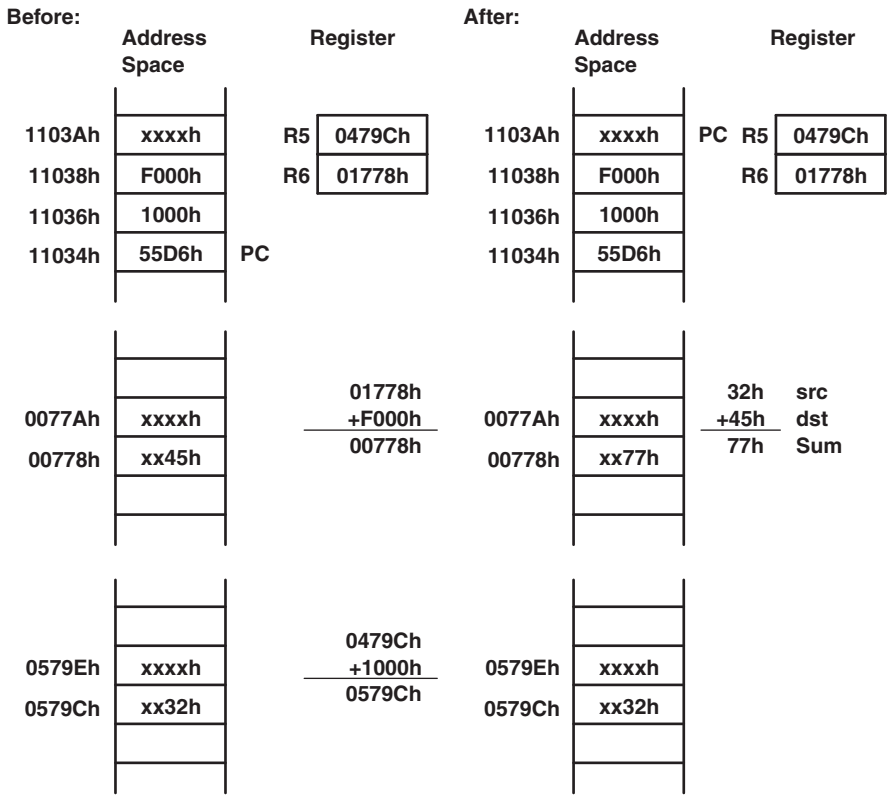


Figure 4-15. Indexed Mode in Lower 64 KB

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.



4.4.2.2 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 4-16 and Figure 4-17).

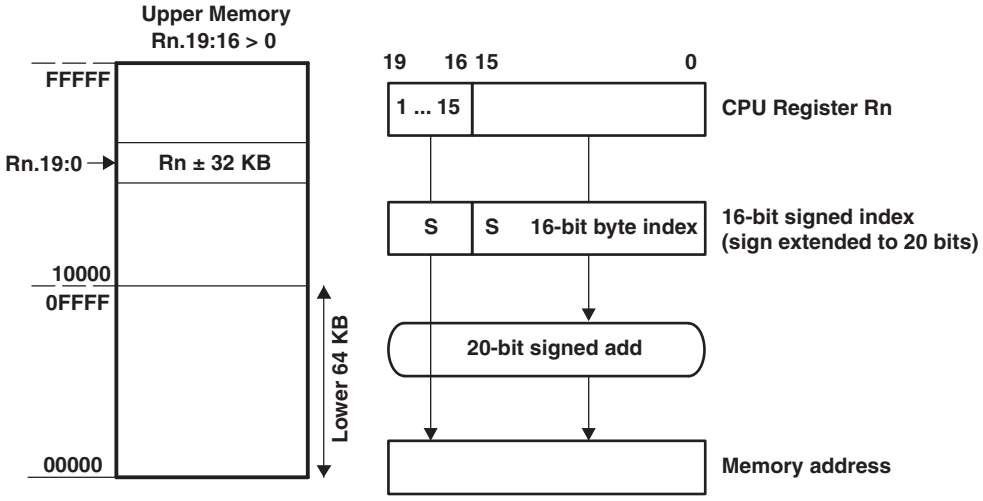


Figure 4-16. Indexed Mode in Upper Memory

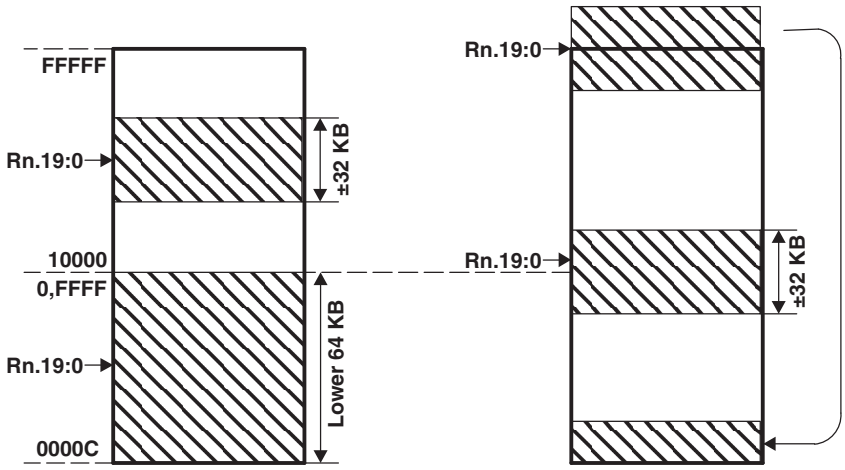


Figure 4-17. Overflow and Underflow for Indexed Mode

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.

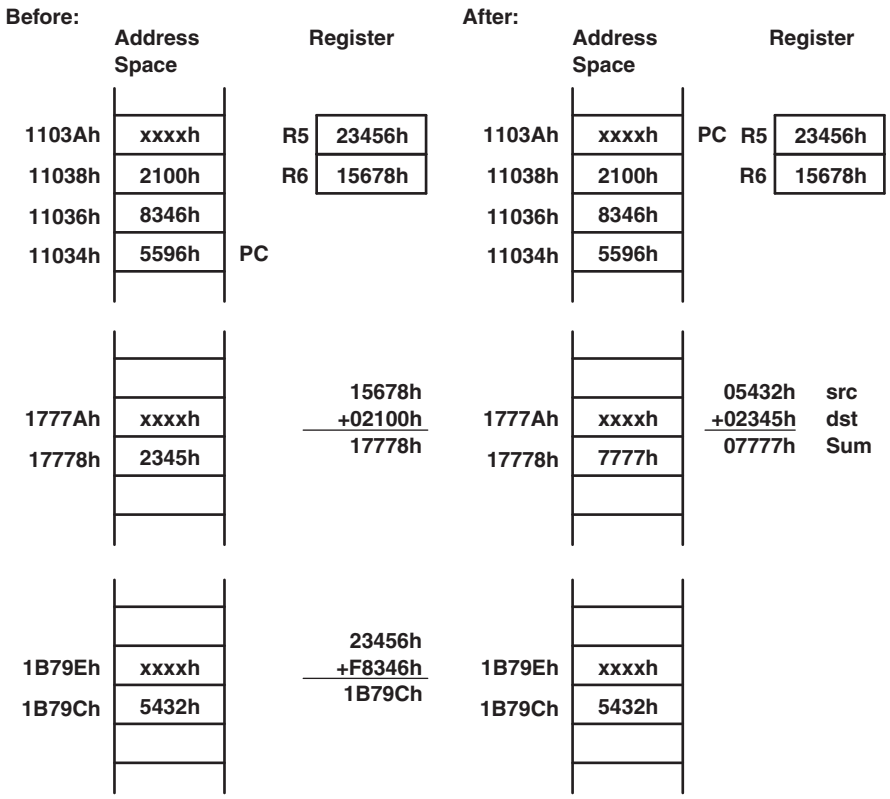


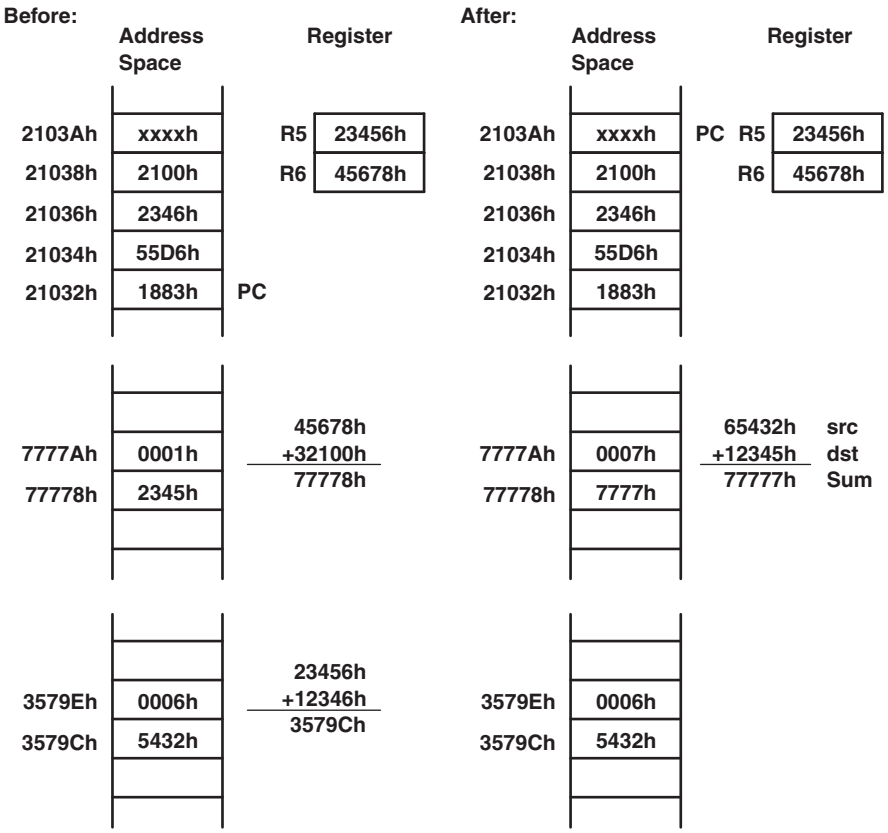
Figure 4-18. Example for Indexed Mode

4.4.2.3 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 4 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.



4.4.3 Symbolic Mode

The Symbolic mode calculates the address of the operand by adding the signed index to the PC. 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

4.4.3.1 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 4-19.

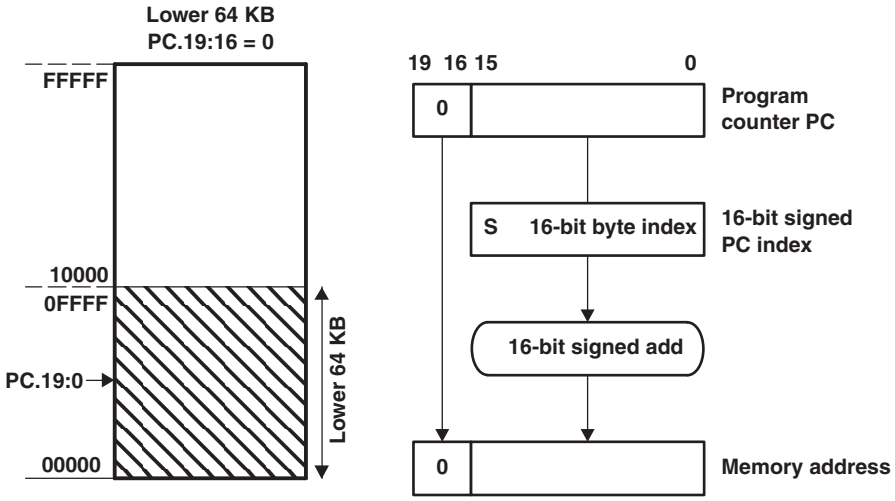
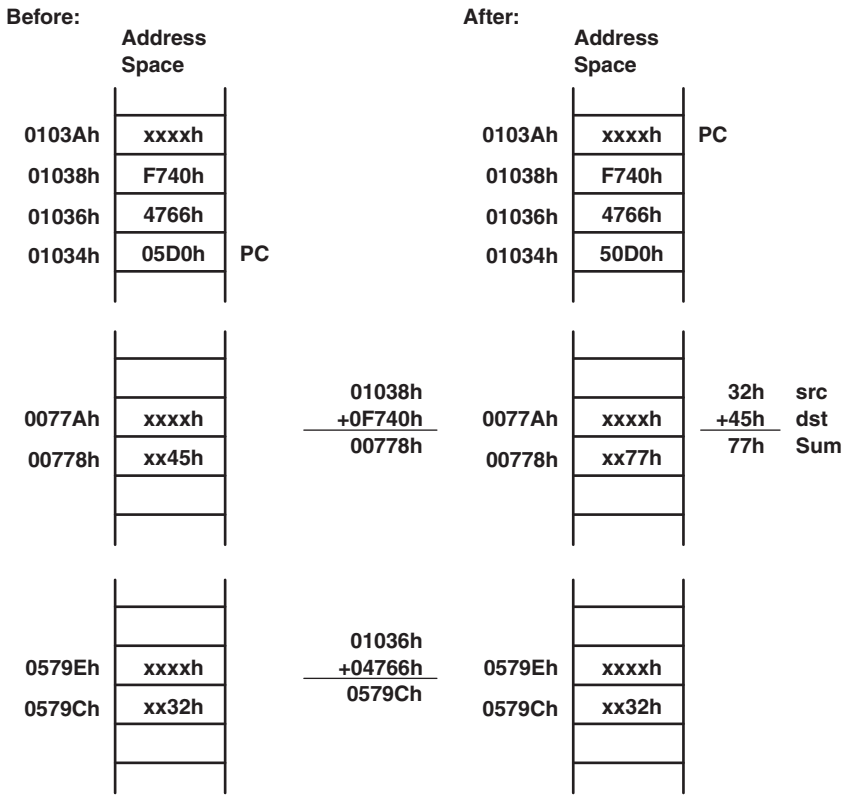


Figure 4-19. Symbolic Mode Running in Lower 64 KB

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 0579Ch, 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.



4.4.3.2 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 4-20 and Figure 4-21.

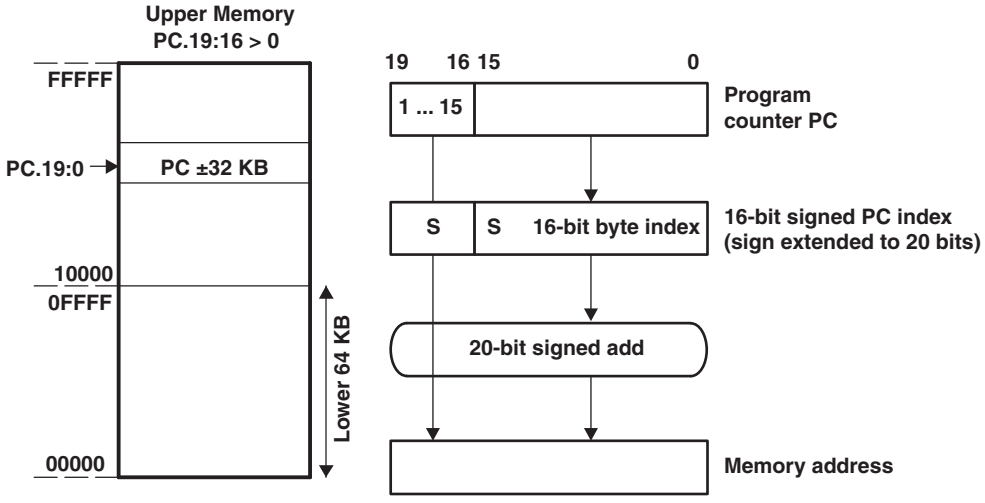


Figure 4-20. Symbolic Mode Running in Upper Memory

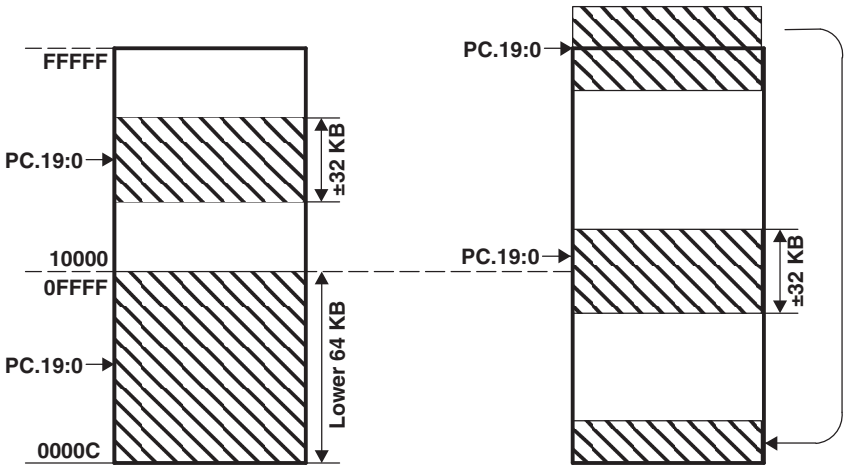
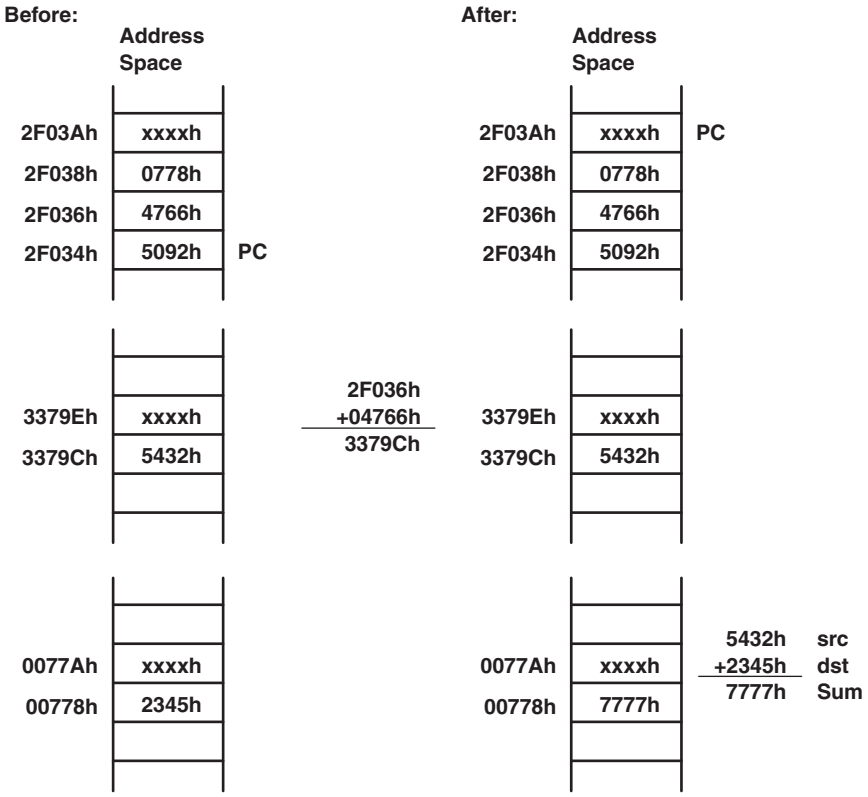


Figure 4-21. Overflow and Underflow for Symbolic Mode

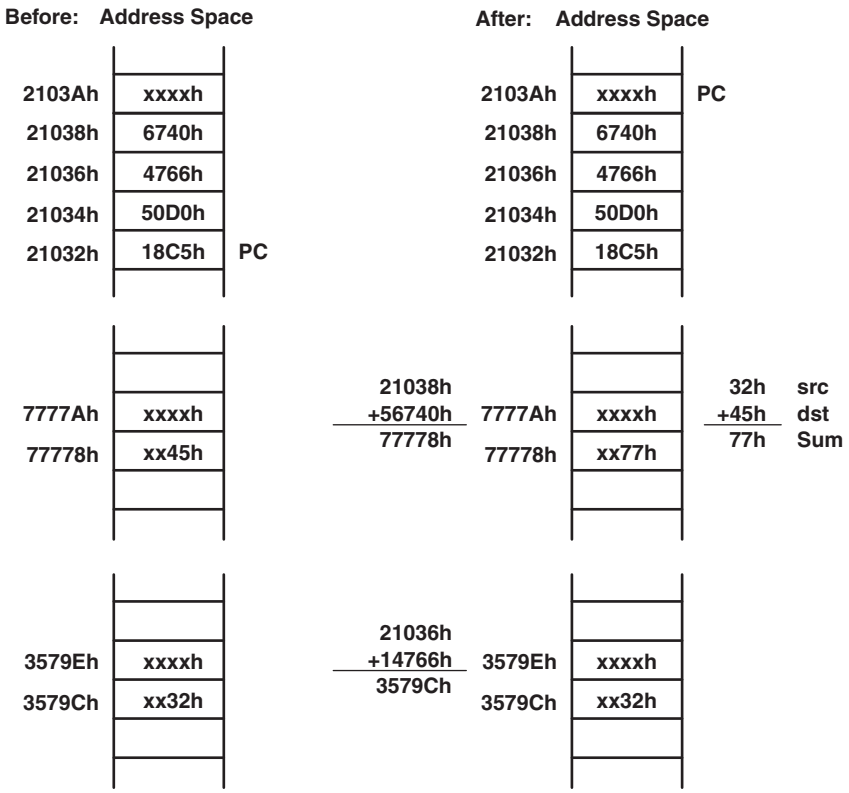
Length:	Two or three words
Operation:	The sign-extended 16-bit index in the next word after the instruction is added to the 20 bits of the PC. This delivers a 20-bit address, which points to an address in the range 0 to FFFFFh. The operand is the content of the addressed memory location.
Comment:	Valid 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 2F034h.
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



4.4.3.3 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 4 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.



4.4.4 Absolute Mode

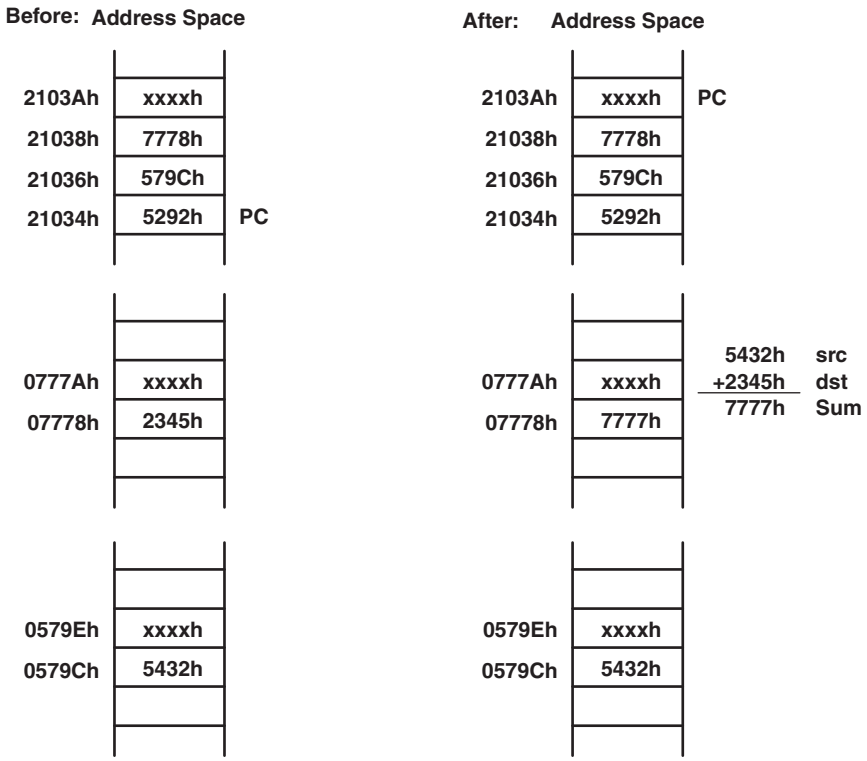
The Absolute mode uses the contents of the word following the instruction as the address of the operand. The Absolute mode has two addressing possibilities:

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

4.4.4.1 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.

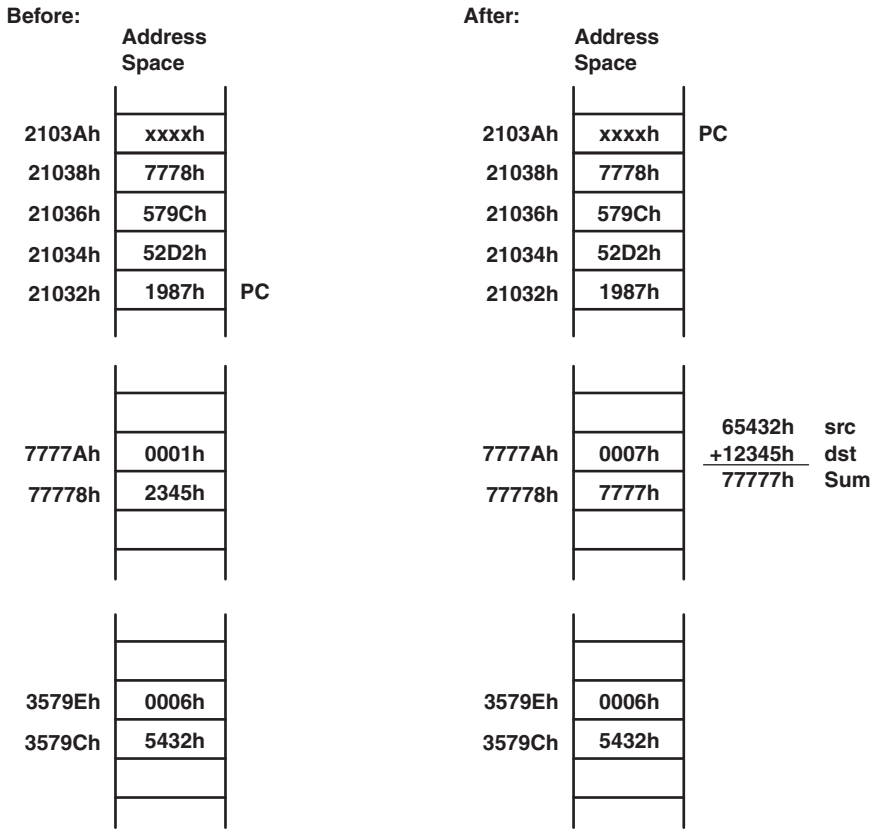
Length:	Two or three words
Operation:	The operand is the content of the addressed memory location.
Comment:	Valid for source and destination. The assembler calculates the index from 0 and inserts it.
Example:	`ADD.W &EDE,&TONI ;`
	This instruction adds the 16-bit data contained in the absolute source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



4.4.4.2 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 4 MSBs of the index are contained in the extension word, and the 16 LSBs are contained in the word following the instruction.

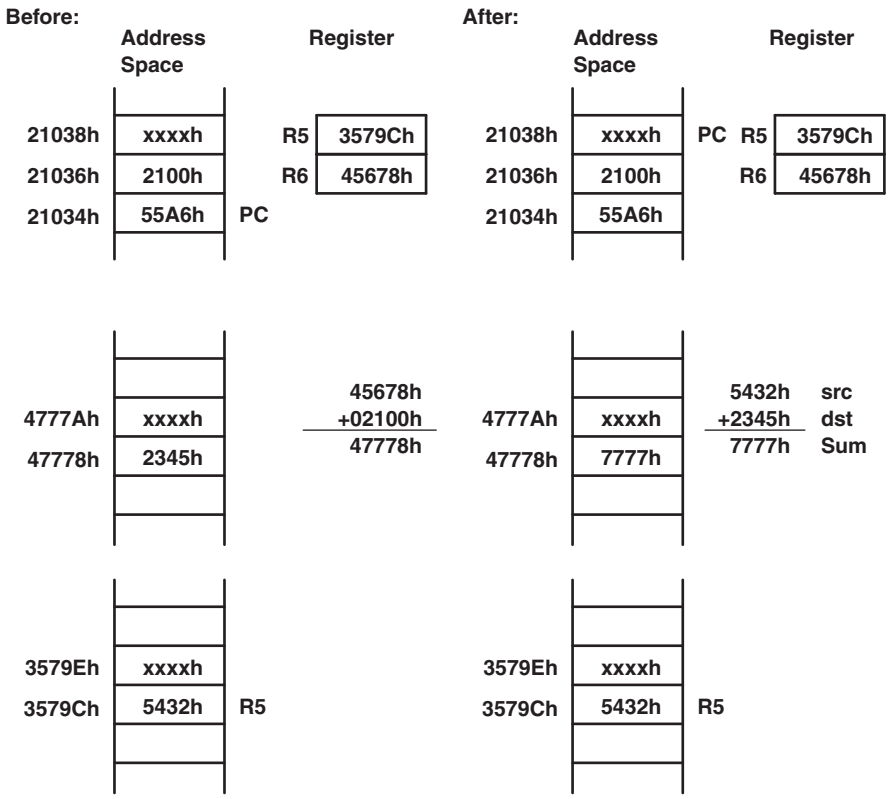
Length:	Three or four words
Operation:	The operand is the content of the addressed memory location.
Comment:	Valid for source and destination. The assembler calculates the index from 0 and inserts it.
Example:	`ADDX.A &EDE,&TONI ;`
	This instruction adds the 20-bit data contained in the absolute source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



4.4.5 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.

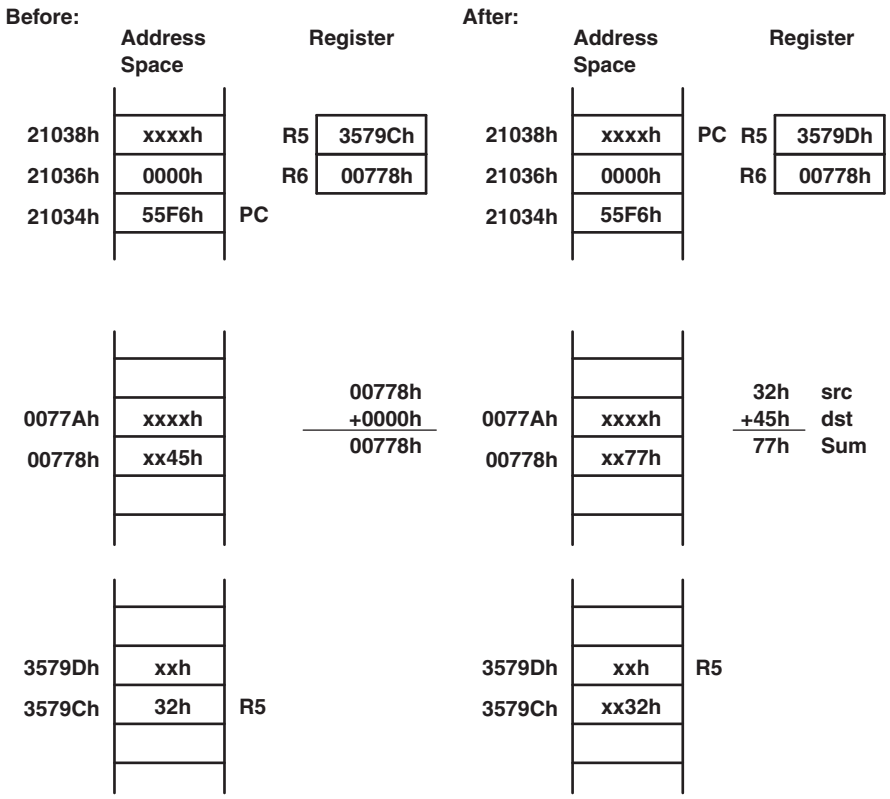
Length:	One, two, or three words
Operation:	The operand is the content the addressed memory location. The source register Rsrc is not modified.
Comment:	Valid only for the source operand. The substitute for the destination operand is 0(Rdst).
Example:	`ADDX.W @R5,2100h(R6)`
	This instruction adds the 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



4.4.6 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 3579Ch for this example.
Destination:	Byte pointed to by R6 + 0h, which results in address 0778h for this example



4.4.7 Immediate Mode

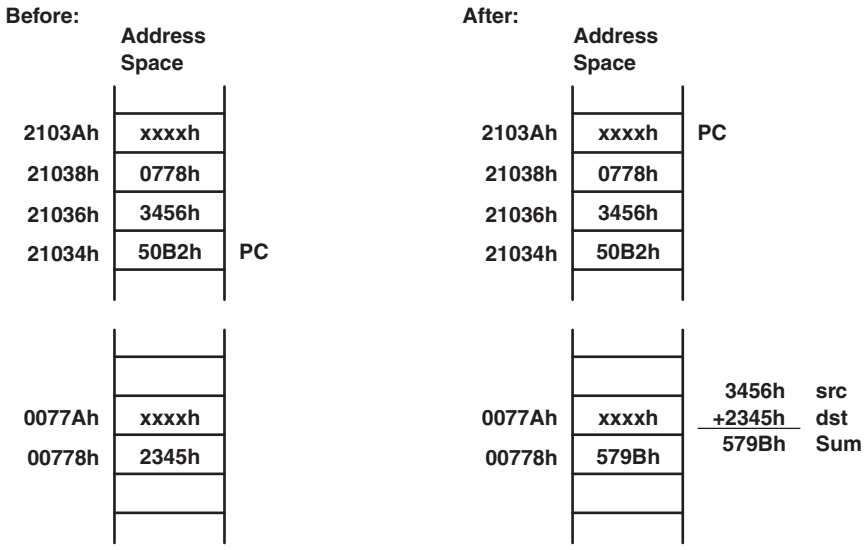
The Immediate mode allows accessing constants as operands by including the constant in the memory location following the instruction. The PC is used with the Indirect Autoincrement mode. The PC points to the immediate value contained in the next word. After the fetching of the immediate operand, the 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

4.4.7.1 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.

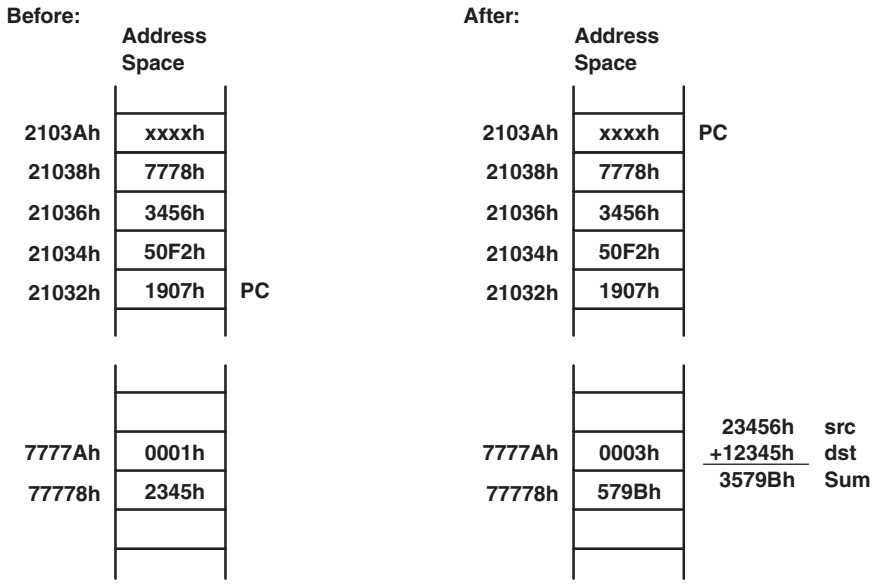
Length:	Two or three words. One word less if a constant of the constant generator can be used for the immediate operand.
Operation:	The 16-bit immediate source operand is used together with the 16-bit destination operand.
Comment:	Valid only for the source operand
Example:	`ADD #3456h,&TONI`
	This instruction 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



4.4.7.2 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



4.5 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.

4.5.1 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.

4.5.1.1 MSP430 Double-Operand (Format I) Instructions

Figure 4-22 shows the format of the MSP430 double-operand instructions. Source and destination words are appended for the Indexed, Symbolic, Absolute, and Immediate modes. Table 4-4 lists the 12 MSP430 double-operand instructions.

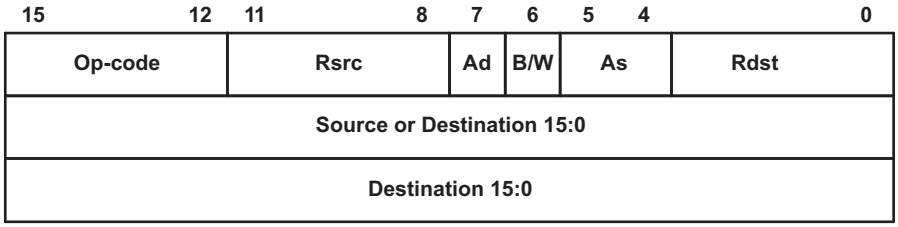


Figure 4-22. MSP430 Double-Operand Instruction Format

Table 4-4. 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	*	*	*	*
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

(1) * = Status bit is affected.
– = Status bit is not affected.
0 = Status bit is cleared.
1 = Status bit is set.

4.5.1.2 MSP430 Single-Operand (Format II) Instructions

Figure 4-23 shows the format for MSP430 single-operand instructions, except RETI. The destination word is appended for the Indexed, Symbolic, Absolute, and Immediate modes. Table 4-5 lists the seven single-operand instructions.

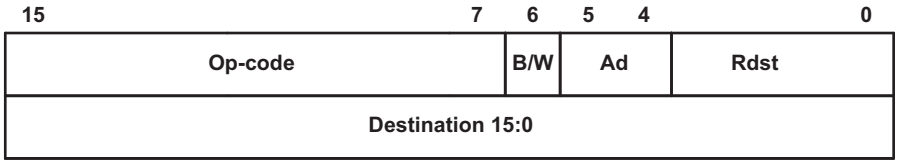


Figure 4-23. MSP430 Single-Operand Instructions

Table 4-5. MSP430 Single-Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits (1)			
			V	N	Z	C
RRC(.B)	dst	C → MSB →.....LSB → C	0	*	*	*
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) * = Status bit is affected.
– = Status bit is not affected.
0 = Status bit is cleared.
1 = Status bit is set.

4.5.1.3 Jump Instructions

Figure 4-24 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 PC. This allows jumps in a range of –511 to +512 words relative to the PC in the full 20-bit address space. Jumps do not affect the status bits. Table 4-6 lists and describes the eight jump instructions.

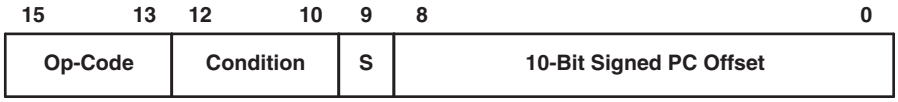


Figure 4-24. Format of Conditional Jump Instructions

Table 4-6. 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

4.5.1.4 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 4-7.

Table 4-7. 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	*	*	*	*

(1) * = Status bit is affected.
– = Status bit is not affected.
0 = Status bit is cleared.
1 = Status bit is set.

Table 4-7. Emulated Instructions (continued)

Instruction	Explanation	Emulation	Status Bits (1)			
			V	N	Z	C
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	*	*	*	*
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

4.5.1.5 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.

4.5.1.5.1 Instruction Cycles and Length for Interrupt, Reset, and Subroutines

Table 4-8 lists the length and the CPU cycles for reset, interrupts, and subroutines.

Table 4-8. 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	–

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

Table 4-9 lists the length and the CPU cycles for all addressing modes of the MSP430 single-operand instructions.

Table 4-9. 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

4.5.1.5.3 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.

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

Table 4-10 lists the length and CPU cycles for all addressing modes of the MSP430 Format I instructions.

Table 4-10. 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.

4.5.2 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 words:

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

4.5.2.1 Register Mode Extension Word

The register mode extension word is shown in Figure 4-25 and described in Table 4-11. An example is shown in Figure 4-27.

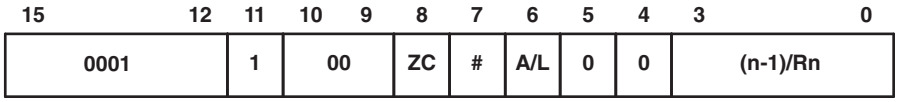


Figure 4-25. Extension Word for Register Modes

Table 4-11. 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
	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 is defined by the result of the final operation after instruction execution.
#	Repetition
	0 The number of instruction repetitions is set by extension word bits 3:0.
	1 The number of instruction repetitions is defined by the value of the four LSBs of Rn. See description for bits 3:0.
A/L	Data length extension. 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.

4.5.2.2 Non-Register Mode Extension Word

The extension word for non-register modes is shown in Figure 4-26 and described in Table 4-12. An example is shown in Figure 4-28.

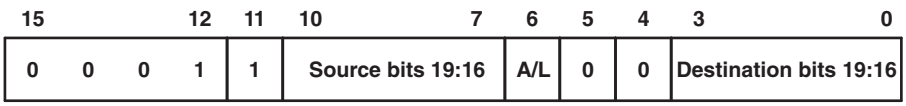


Figure 4-26. Extension Word for Non-Register Modes

Table 4-12. Description of 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.
A/L	Data length extension. 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
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.

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

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

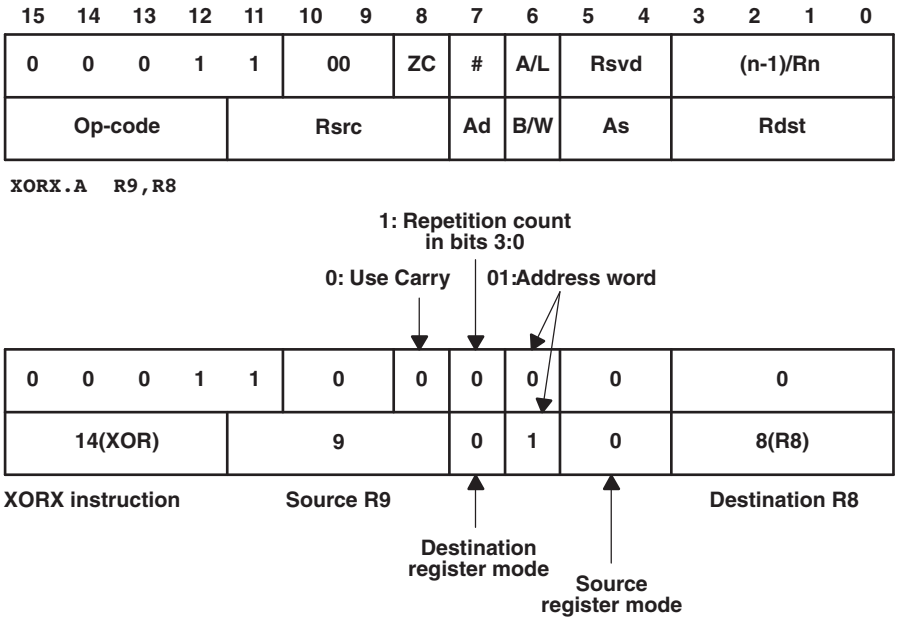


Figure 4-27. Example for Extended Register/Register Instruction

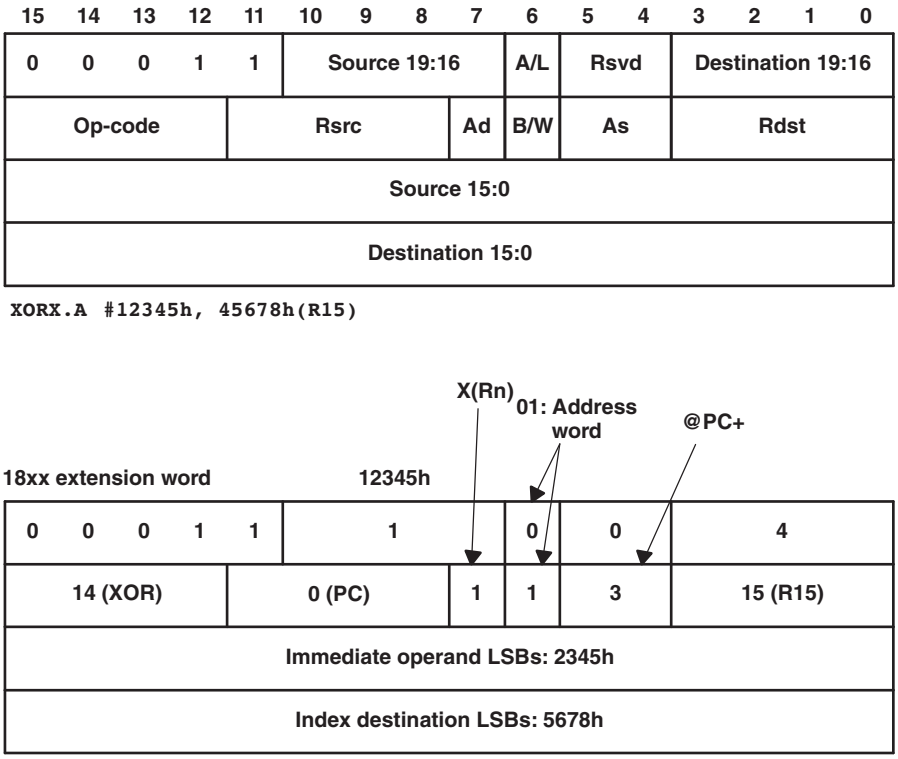


Figure 4-28. Example for Extended Immediate/Indexed Instruction

4.5.2.3 Extended Double-Operand (Format I) Instructions

All 12 double-operand instructions have extended versions as listed in Table 4-13.

Table 4-13. 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) * = Status bit is affected.
– = Status bit is not affected.
0 = Status bit is cleared.
1 = Status bit is set.

The four possible addressing combinations for the extension word for Format I instructions are shown in Figure 4-29.

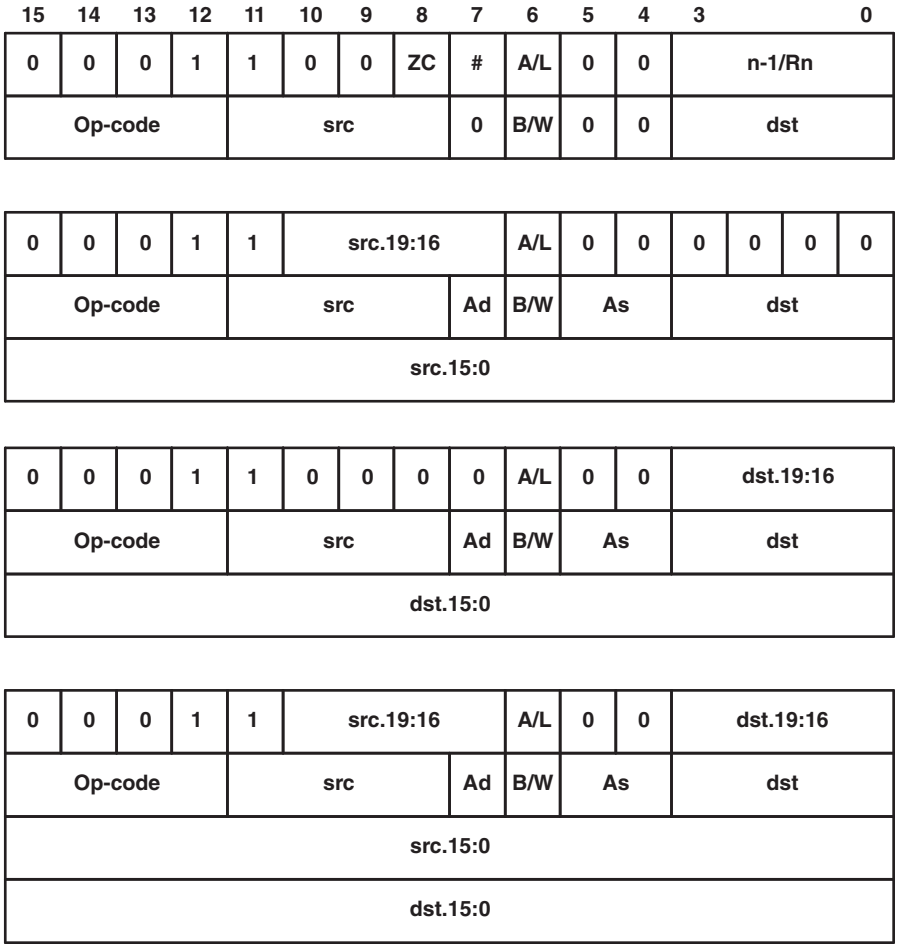


Figure 4-29. Extended Format I Instruction Formats

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 4-30.

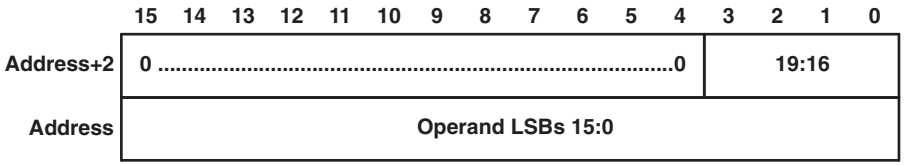


Figure 4-30. 20-Bit Addresses in Memory

4.5.2.4 Extended Single-Operand (Format II) Instructions

Extended MSP430X Format II instructions are listed in Table 4-14.

Table 4-14. 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	0	*	*	*
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	0	*	*	*
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	0	*	*	Z
RRUX(.B,.A)	Rdst	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) * = Status bit is affected.
– = Status bit is not affected.
0 = Status bit is cleared.
1 = Status bit is set.

The three possible addressing mode combinations for Format II instructions are shown in Figure 4-31.

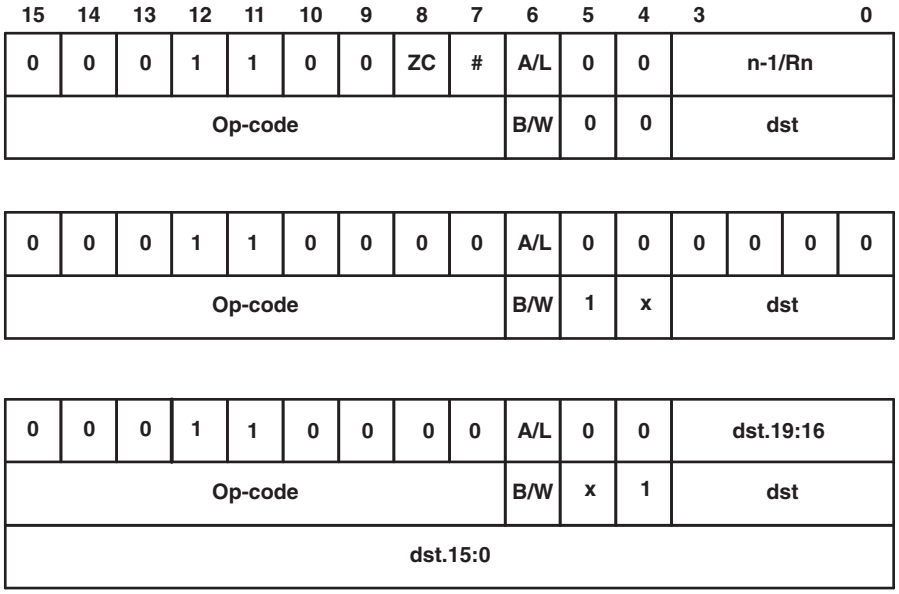


Figure 4-31. Extended Format II Instruction Format

4.5.2.4.1 Extended Format II Instruction Format Exceptions

Exceptions for the Format II instruction formats are shown in Figure 4-32 through Figure 4-35.

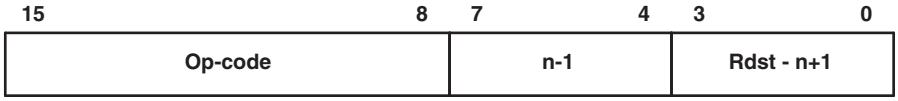


Figure 4-32. PUSHM/POPM Instruction Format

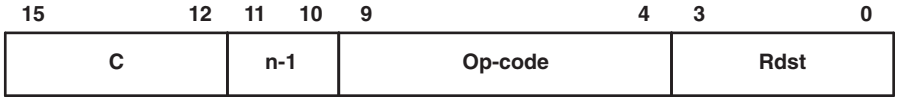


Figure 4-33. RRCM, RRAM, RRUM, and RLAM Instruction Format

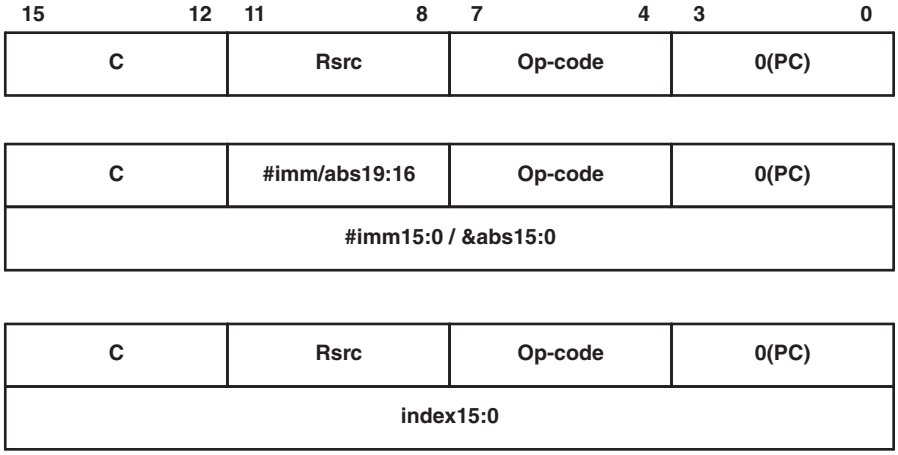


Figure 4-34. BRA Instruction Format

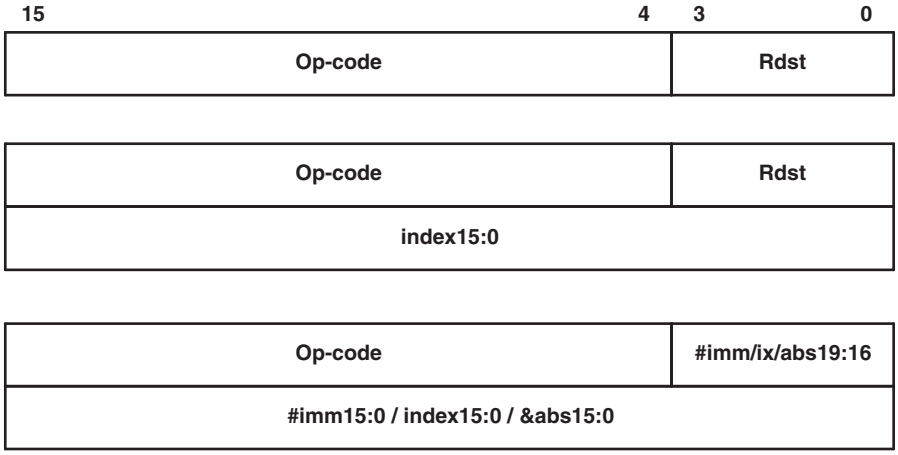


Figure 4-35. CALLA Instruction Format

4.5.2.5 Extended Emulated Instructions

The extended instructions together with the constant generator form the extended emulated instructions. Table 4-15 lists the emulated instructions.

Table 4-15. 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

4.5.2.6 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 4-16. 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 4-16. 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	Rsrc,Rdst	Compare source to destination register	*	*	*	*
	#imm20,Rdst					
SUBA	Rsrc,Rdst	Subtract source from destination register	*	*	*	*
	#imm20,Rdst					

(1) * = Status bit is affected.
– = Status bit is not affected.
0 = Status bit is cleared.
1 = Status bit is set.

4.5.2.7 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.

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

Table 4-17 lists the length and the CPU cycles for all addressing modes of the MSP430X extended single-operand instructions.

Table 4-17. 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

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

Table 4-18 lists the length and CPU cycles for all addressing modes of the MSP430X extended Format I instructions.

Table 4-18. 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	3	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.

4.5.2.7.3 MSP430X Address Instruction Cycles and Lengths

Table 4-19 lists the length and the CPU cycles for all addressing modes of the MSP430X address instructions.

Table 4-19. 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

4.6 Instruction Set Description

Table 4-20 shows all available instructions:

Table 4-20. 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 1Cxx	Extension word for Format I and Format II instructions															
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															

4.6.1 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

4.6.2 MSP430 Instructions

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

4.6.2.1 ADC

*** 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
```

4.6.2.2 ADD

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
```

4.6.2.3 ADDC

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
```

4.6.2.4 AND

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
```

4.6.2.5 BIC

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
```

4.6.2.6 BIS

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
```

4.6.2.7 BIT

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
```

4.6.2.8 BR, BRANCH

*** 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 (for example #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 (for example table with address
                 ; starting at X). X can be an address or a label
                 ; Core instruction MOV X(R5),PC
                 ; Indirect, indirect R5 + X
```

4.6.2.9 CALL

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
```

4.6.2.10 CLR

*** 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
```

4.6.2.11 CLRC

*** 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
```

4.6.2.12 CLRN

*** 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 SR 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
```

4.6.2.13 CLRZ

*** 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 SR 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); for example, 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)
```

4.6.2.14 CMP

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 1s 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
```

4.6.2.15 DADC

*** 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
```

4.6.2.16 DADD

*** 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 = 0

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

4.6.2.17 DEC

*** 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 4-36.

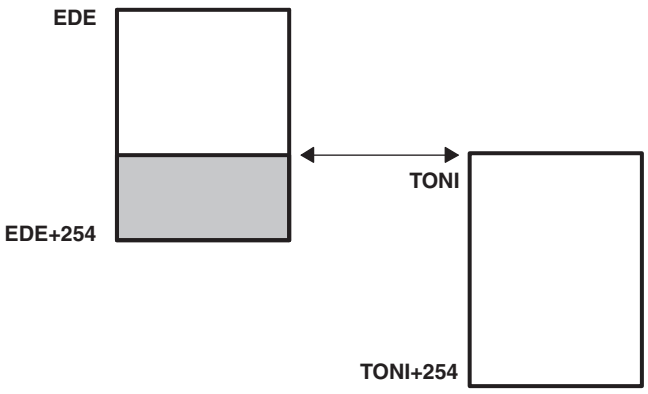


Figure 4-36. Decrement Overlap

4.6.2.18 DECD

*** 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
```

4.6.2.19 DINT

*** 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 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 SR 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.

4.6.2.20 EINT

*** 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 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 SR 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.

4.6.2.21 INC

*** 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
```

4.6.2.22 INCD

*** 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
```

4.6.2.23 INV

*** 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 (2s 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
```

4.6.2.24 JC, JHS

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 SR 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 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
```

4.6.2.25 JEQ, JZ

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 SR 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 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.
```

4.6.2.26 JGE

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 SR 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 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 nonimplemented 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
```

4.6.2.27 JL

JL Jump if less (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 SR 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 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 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
```

4.6.2.28 JMP

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 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 PC.
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 64 K.

```
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
```

4.6.2.29 JN

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 SR 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 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.
```

4.6.2.30 JNC, JLO

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 SR 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 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
```

4.6.2.31 JNZ, JNE

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 SR 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 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.
```

4.6.2.32 MOV

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 64 K.

```
        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
```

4.6.2.33 NOP

*** 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.

4.6.2.34 POP

*** 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 SP (TOS) is moved to the destination. The SP is incremented by two afterwards.

Status Bits Status bits are not affected.

Example The contents of R7 and the SR 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 SR 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: System stack pointer

The system SP is always incremented by two, independent of the byte suffix.

4.6.2.35 PUSH

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 SP 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
```

4.6.2.36 RET

*** 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 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 64 K 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
```

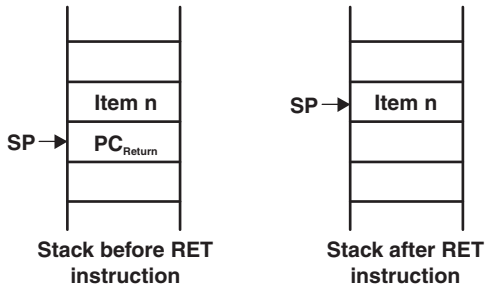


Figure 4-37. Stack After a RET Instruction

4.6.2.37 RETI

RETI	Return from interrupt
Syntax	`RETI`
Operation	@SP → SR.15:0 Restore saved SR with PC.19:16 SP + 2 → SP @SP → PC.15:0 Restore saved PC.15:0 SP + 2 → SP Housekeeping
Description	The SR is restored to the value at the beginning of the interrupt service routine. This includes the four MSBs of the PC.19:16. The SP 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 PC 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 SP 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
```

4.6.2.38 RLA

*** 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 4-38. 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.

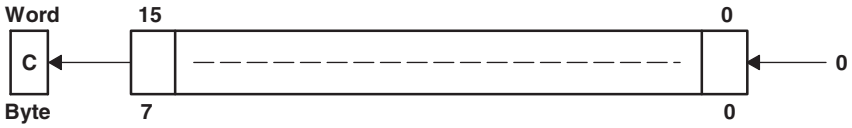


Figure 4-38. Destination Operand—Arithmetic Shift Left

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
```

4.6.2.39 RLC

*** 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 4-39. The carry bit (C) is shifted into the LSB, and the MSB is shifted into the carry bit (C).

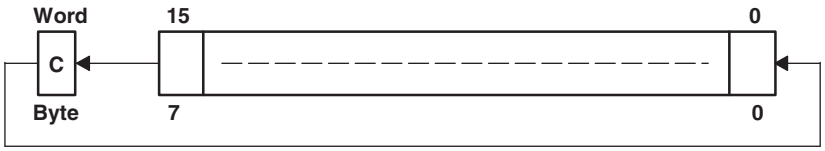


Figure 4-39. Destination Operand—Carry Left Shift

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`

4.6.2.40 RRA

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 4-40. 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
```

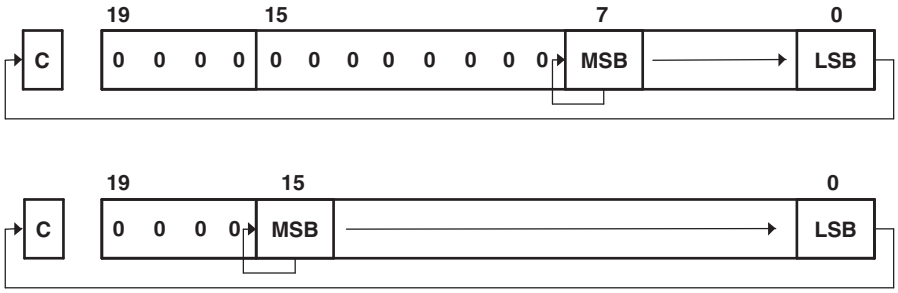


Figure 4-40. Rotate Right Arithmetically RRA.B and RRA.W

4.6.2.41 RRC

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 4-41. 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
```

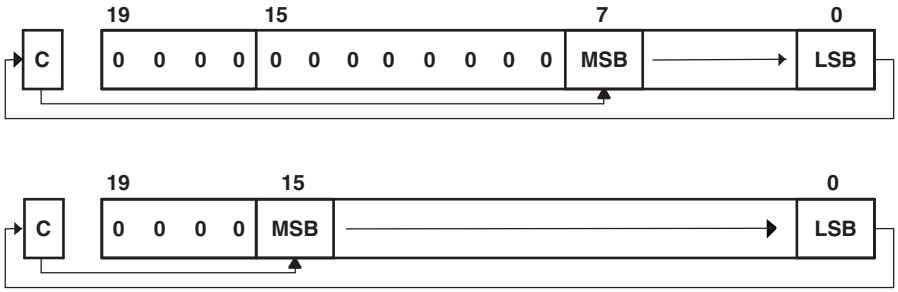


Figure 4-41. Rotate Right Through Carry RRC.B and RRC.W

4.6.2.42 SBC

*** SBC[.W]** Subtract borrow (.NOT. carry) from destination

*** SBC.B** Subtract 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

4.6.2.43 SETC

*** 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
```

4.6.2.44 SETN

*** SETN**	Set negative bit
Syntax	`SETN`
Operation	1 → N
Emulation	`BIS #4,SR`
Description	The negative bit (N) is set.
Status Bits	N: Set Z: Not affected C: Not affected V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.

4.6.2.45 SETZ

*** 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.

4.6.2.46 SUB

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 1s 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 ± 32K. The address R12 points to is in full memory range.

```
SUB.B   CNT,0(R12)      ; Subtract CNT from @R12
```

4.6.2.47 SUBC

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 1s 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
```

4.6.2.48 SWPB

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
```

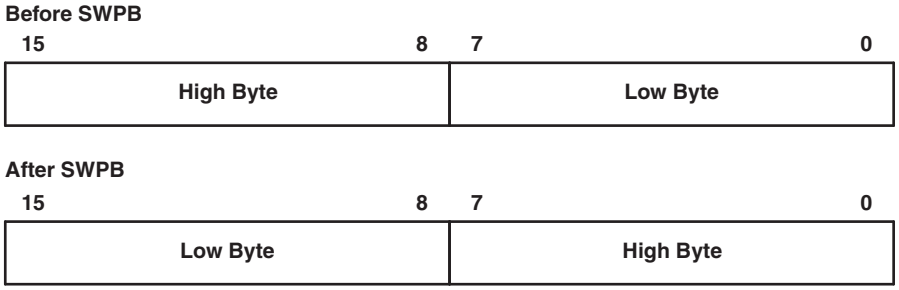


Figure 4-42. Swap Bytes in Memory

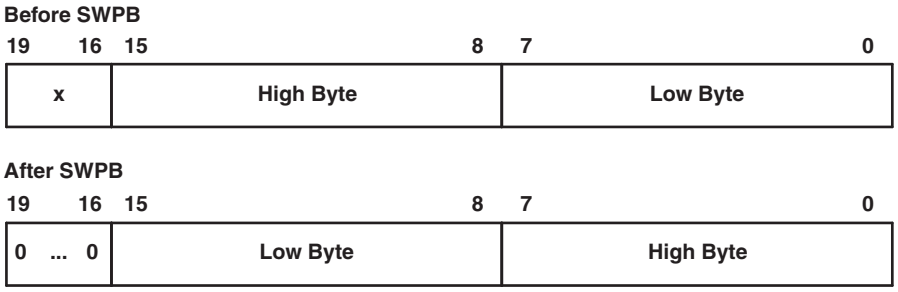


Figure 4-43. Swap Bytes in a Register

4.6.2.49 SXT

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
```

4.6.2.50 TST

*** 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
```

4.6.2.51 XOR

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
```

4.6.3 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.

4.6.3.1 ADCX

*** 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
```

4.6.3.2 ADDX

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
```

4.6.3.3 ADDCX

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
```

4.6.3.4 ANDX

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
```

4.6.3.5 BICX

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
```

4.6.3.6 BISX

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
```

4.6.3.7 BITX

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
```

4.6.3.8 CLRX

*** 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
```

4.6.3.9 CMPX

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 1s 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
```

4.6.3.10 DADCX

*** 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
```

4.6.3.11 DADDX

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
```

4.6.3.12 DECX

*** 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 one.

```
DECX.A   TONI     ; Decrement TONI
```

4.6.3.13 DECDX

*** 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 two.

```
DECDX.A   TONI     ; Decrement TONI
```

4.6.3.14 INCX

*** 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 one.

```
INCX.A   TONI     ; Increment TONI (20-bits)
```

4.6.3.15 INCDX

*** 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
```

4.6.3.16 INVX

*** 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 (2s 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
```

4.6.3.17 MOVX

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
```

4.6.3.18 POPM

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 SP is incremented by four for each register restored from stack. The 20-bit values from stack (two words per register) are restored to the registers.	
	POPM.W: Restore the 16-bit register values from stack to the specified CPU registers. The 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 SP is incremented by (n × 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 SP is incremented by (n × 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
```

4.6.3.19 PUSHM

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 ≤ n ≤ 16
	`PUSHM.W #n,Rdst` or `PUSHM #n,Rdst`	1 ≤ n ≤ 16
Operation	PUSHM.A: Save the 20-bit CPU register values on the stack. The 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 SP 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 SP 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 SP 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
```

4.6.3.20 POPX

*** 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 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 SP is incremented by two or four. Note: the SP 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
```

4.6.3.21 PUSHX

PUSHX.A Save single address-word to the stack

PUSHX.[W] Save single word to the stack

PUSHX.B Save single byte to 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 SP is decremented by two (byte and word operands) or by four (address-word operand) before the write operation.

Description The SP 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.

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
```

4.6.3.22 RLAM

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 4-44. 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
```

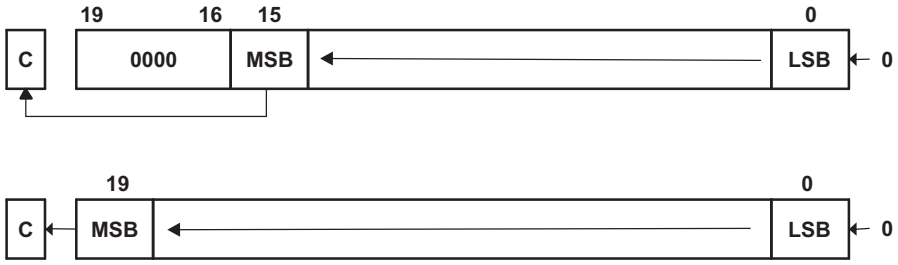


Figure 4-44. Rotate Left Arithmetically—RLAM[.W] and RLAM.A

4.6.3.23 RLAX

*** 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 4-45. 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)
```

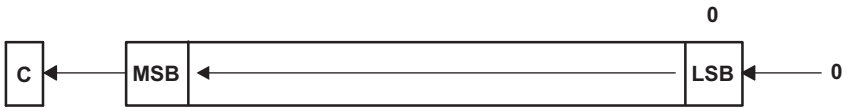


Figure 4-45. Destination Operand-Arithmetic Shift Left

4.6.3.24 RLCX

*** 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 4-46. 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)
```

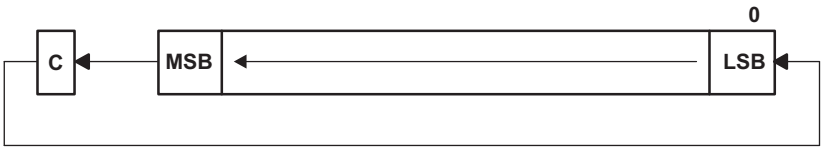


Figure 4-46. Destination Operand-Carry Left Shift

4.6.3.25 RRAM

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 4-47. 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
```

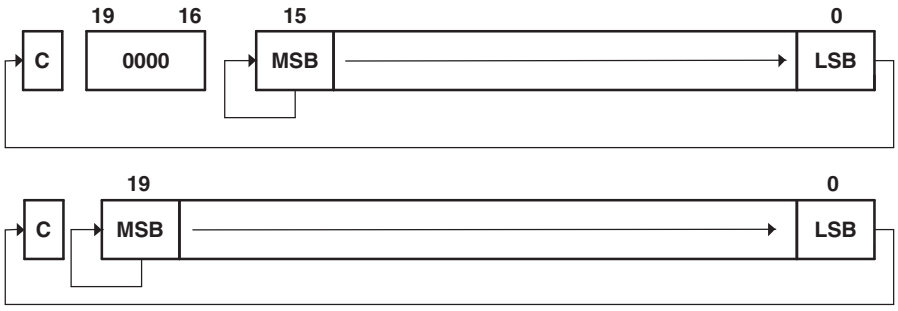


Figure 4-47. Rotate Right Arithmetically RRAM[.W] and RRAM.A

4.6.3.26 RRAX

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 4-48. 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 4-49. 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
```

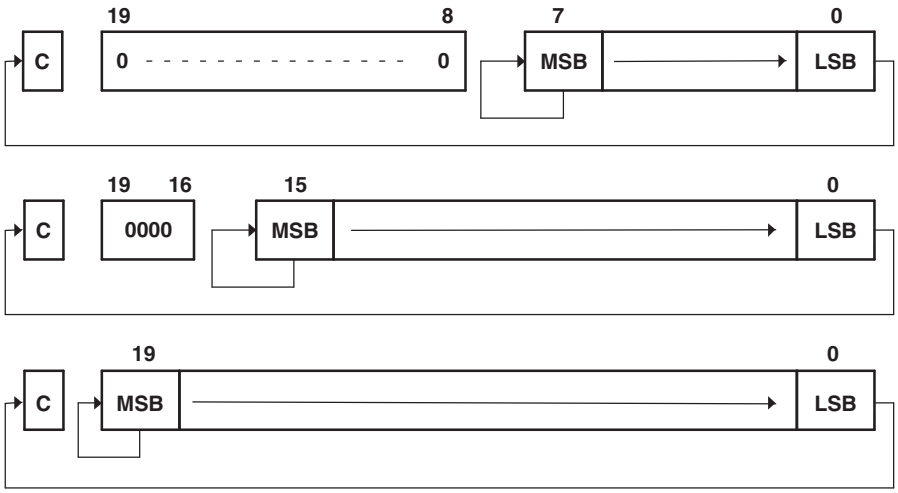


Figure 4-48. Rotate Right Arithmetically RRAX(.B,.A) – Register Mode

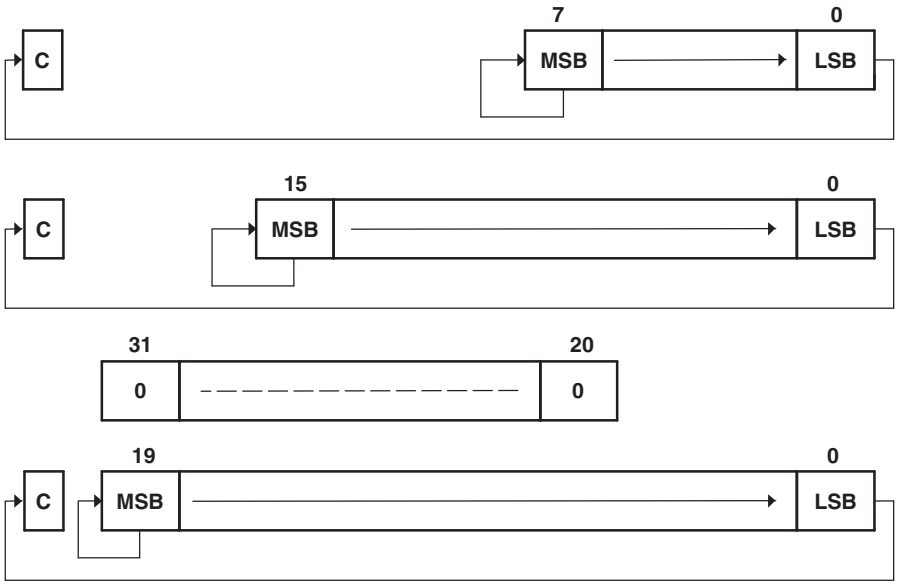


Figure 4-49. Rotate Right Arithmetically RRAX(.B,.A) – Non-Register Mode

4.6.3.27 RRCM

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 4-50. 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
```

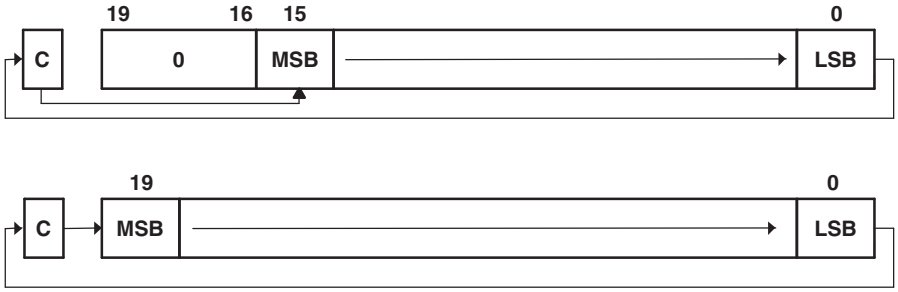


Figure 4-50. Rotate Right Through Carry RRCM[.W] and RRCM.A

4.6.3.28 RRCX

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 4-51. 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 4-52. 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 12 positions.

```
RPT      #12
RRCX.W   R6       ; R6 = R6 » 12. R6.19:16 = 0
```

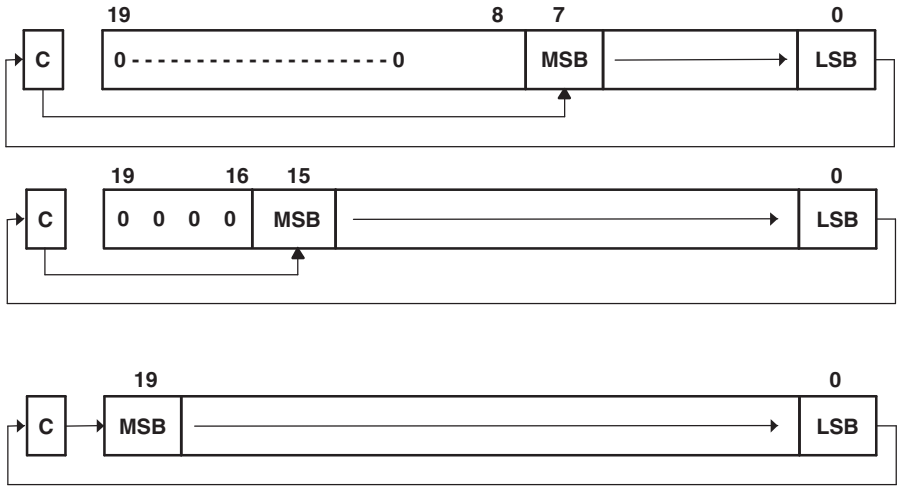


Figure 4-51. Rotate Right Through Carry RRCX(.B,.A) – Register Mode

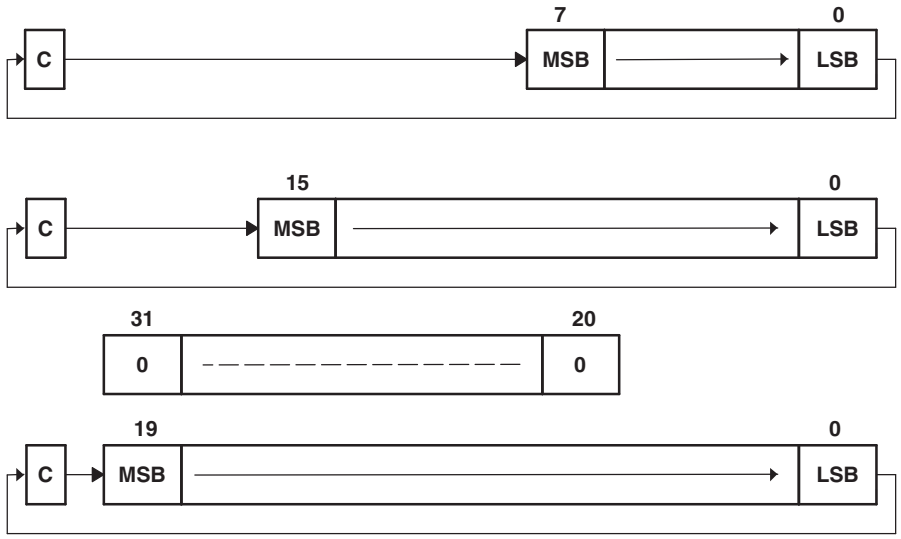


Figure 4-52. Rotate Right Through Carry RRCX(.B,.A) – Non-Register Mode

4.6.3.29 RRUM

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 4-53. 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
```

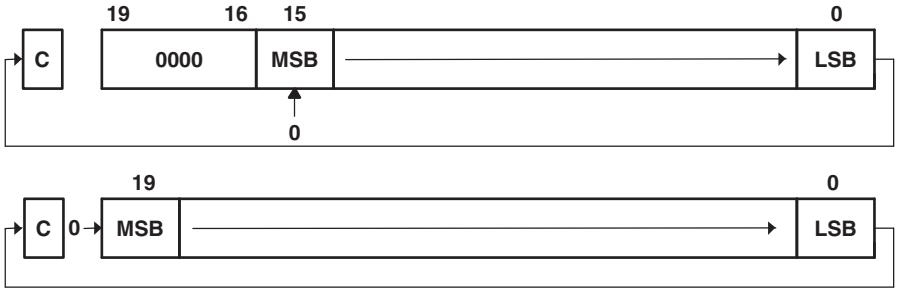


Figure 4-53. Rotate Right Unsigned RRUM[.W] and RRUM.A

4.6.3.30 RRUX

RRUX.A Shift right unsigned the 20-bit CPU register content

RRUX.[W] Shift right unsigned the 16-bit CPU register content

RRUX.B Shift right unsigned the 8-bit CPU register content

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 4-54. 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 12 positions.

```
RPT     #12
RRUX.W  R6      ; R6 = R6 » 12. R6.19:16 = 0
```

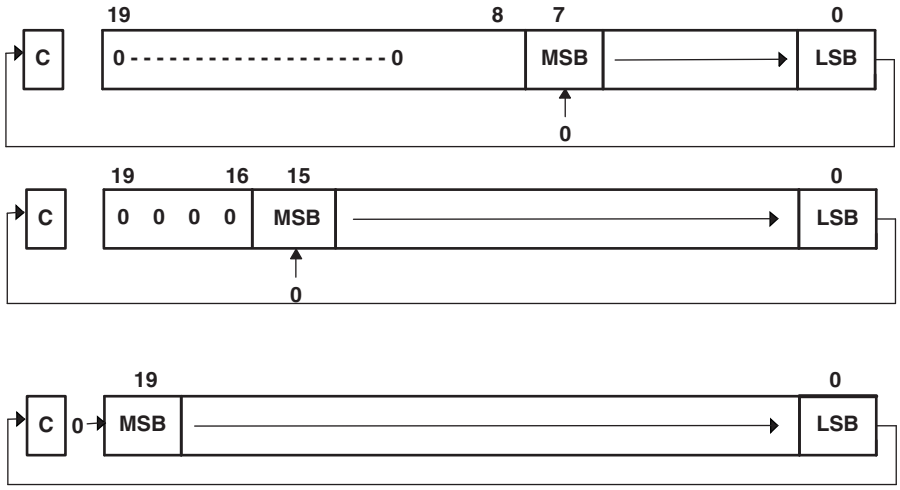


Figure 4-54. Rotate Right Unsigned RRUX(.B,.A) – Register Mode

4.6.3.31 SBCX

*** SBCX.A** Subtract borrow (.NOT. carry) from destination address-word

*** SBCX.[W]** Subtract borrow (.NOT. carry) from destination word

*** SBCX.B** Subtract 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

4.6.3.32 SUBX

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 1s 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 two. 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
```

4.6.3.33 SUBCX

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 1s 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
```

4.6.3.34 SWPBX

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
```

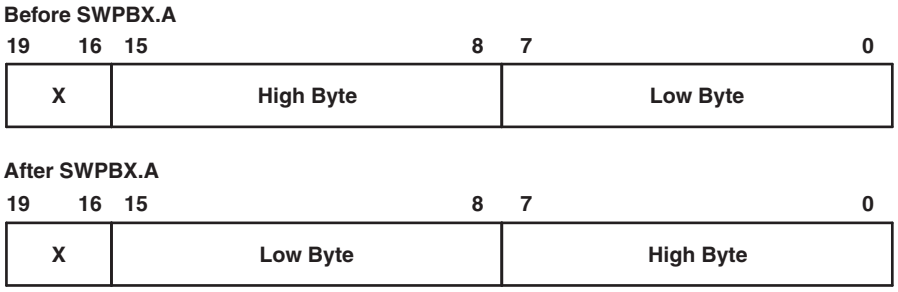


Figure 4-55. Swap Bytes SWPBX.A Register Mode

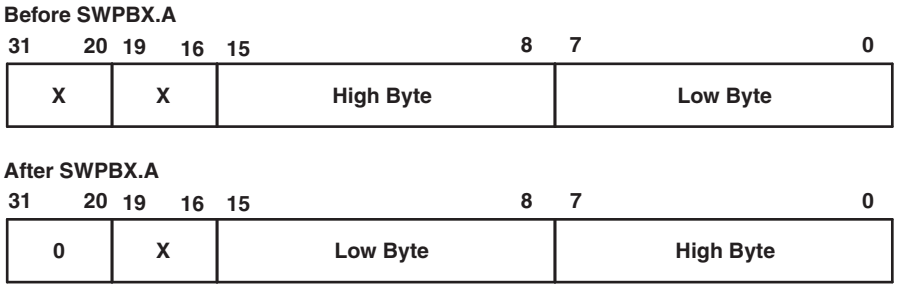


Figure 4-56. Swap Bytes SWPBX.A In Memory

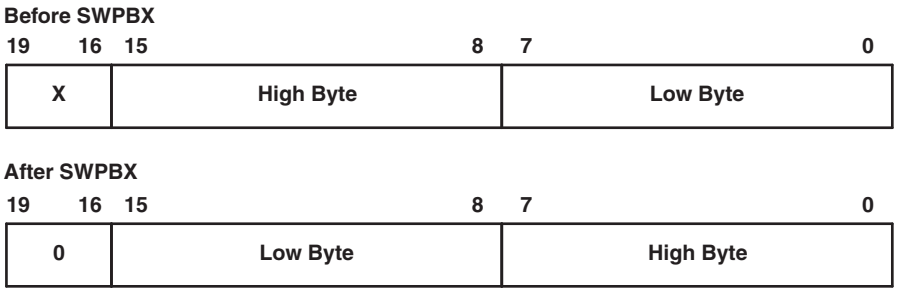


Figure 4-57. Swap Bytes SWPBX[.W] Register Mode

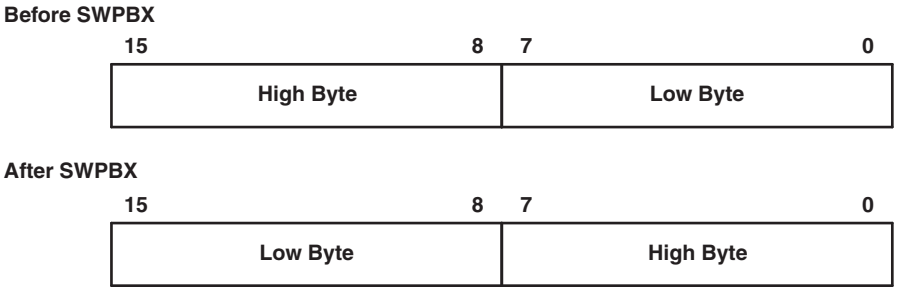


Figure 4-58. Swap Bytes SWPBX[.W] In Memory

4.6.3.35 SXTX

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
```

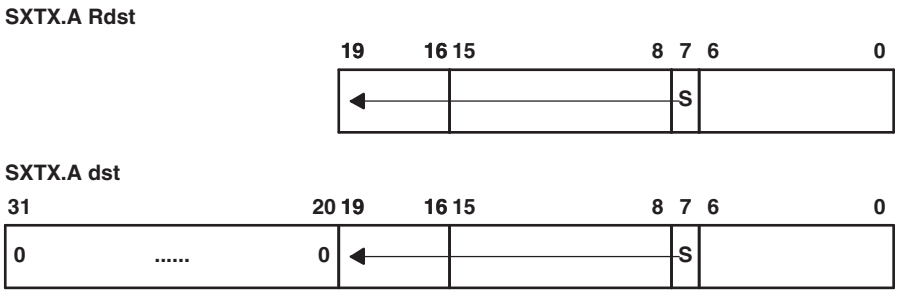


Figure 4-59. Sign Extend SXTX.A

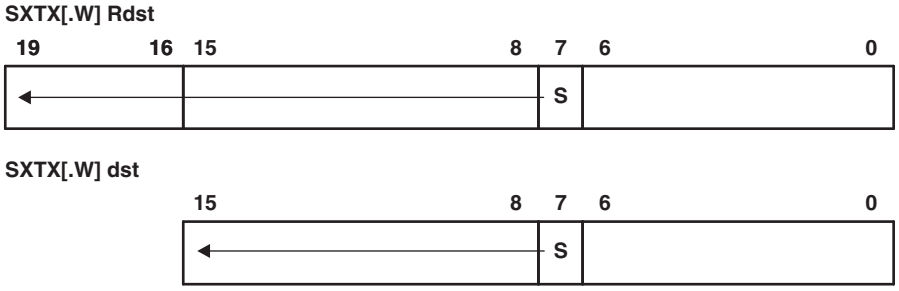


Figure 4-60. Sign Extend SXTX[.W]

4.6.3.36 TSTX

*** 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
```

4.6.3.37 XORX

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.
```

4.6.4 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.

4.6.4.1 ADDA

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
```

4.6.4.2 BRA

*** 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) (for example, 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
```

4.6.4.3 CALLA

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); for example, 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)
```

4.6.4.4 CLRA

*** 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
```

4.6.4.5 CMPA

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 1s 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
```

4.6.4.6 DECDA

*** 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
```

4.6.4.7 INCDA

*** 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 two.

```
INCDA   R5      ; Increment R5 by two
```

4.6.4.8 MOVA

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
```

4.6.4.9 RETA

*** 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 PC. The program continues at the address following the subroutine call. The SR 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)
```

4.6.4.10 SUBA

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 1s 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
```

4.6.4.11 TSTA

*** 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
```

FRAM Controller (FRCTL)

This chapter describes the operation of the FRAM memory controller.

Topic **Page**

5.1 FRAM Introduction

FRAM memory is a non-volatile memory that reads and writes like standard SRAM. The MSP430 FRAM memory features include:

- Byte or word write access
- Automatic and programmable wait state control with independent wait state settings for access and cycle times
- Error Correction Code with bit error correction capabilities, extended bit error detection and flag indicators
- Cache for fast read and endurance improvement

Figure 5-1 shows the block diagram of the FRAM Controller.

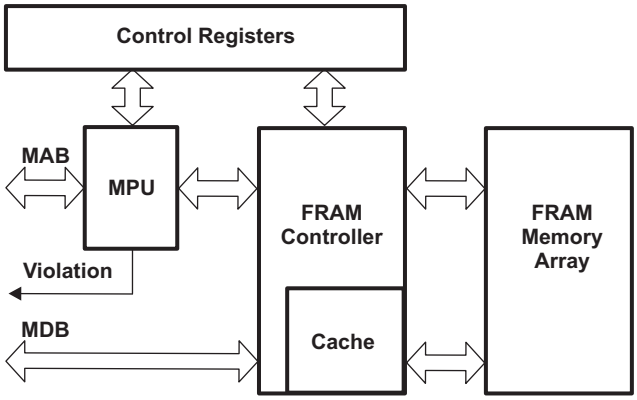


Figure 5-1. FRAM Controller Block Diagram

5.2 FRAM Organization

The FRAM memory can be arranged into segments by the Memory Protection Unit (MPU). See the *Memory Protection Unit* chapter for details. The address space is linear with the exception of the User Information Memory and the Device Descriptor Information (TLV).

5.3 FRCTL Module Operation

The FRAM module can be read in a similar fashion to SRAM and needs no special requirements. Similarly, any writes to unprotected segments can be written in the same fashion as SRAM. All writes to user protected segments are handled as described in the *Memory Protection Unit* chapter.

An FRAM read always requires a write back to the same memory location with the same information read. This write back is part of the FRAM module itself and requires no user interaction. These write backs are different from the normal write access from application code.

The FRAM module has built-in Error Correction Code logic (ECC) that is capable of correcting bit errors and detecting cumulated bit errors. Two flags are available to indicate the presence of an error. The CBDIFG is set when a correctable bit error has been detected and corrected. If CBDIE is also set, a System NMI event (SYSNMI) occurs. The UBDIFG is set when a cumulated bit error which is not correctable has been detected. If UBDIE is also set, a System NMI event (SYSNMI) occurs. Upon correctable or uncorrectable bit errors, the program vectors to the SYSSNIV if the NMI is enabled. If desired, a System Reset event (SYSRST) can be generated by setting the UBDRSTEN bit. If an uncorrectable error is detected, a PUC is initiated and the program vectors to the SYSRSTIV.

5.4 Programming FRAM Memory Devices

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

- Program via JTAG or the Spy-Bi-Wire interface
- Program via the BSL
- Program via a custom solution

5.4.1 Programming FRAM Memory via JTAG or Spy-Bi-Wire

Devices can be programmed via the JTAG port or the Spy-Bi-Wire port. The JTAG interface requires access to TDI, TDO, TMS, TCK, TEST, ground, and optionally VCC and $\overline{\text{RST}}$/NMI. Spy-Bi-Wire interface requires access to TEST, $\overline{\text{RST}}$/NMI, ground and, optionally, VCC.

5.4.2 Programming FRAM Memory via Bootstrap Loader (BSL)

Every device contains a BSL stored in ROM. The BSL enables users to read or program the FRAM memory or RAM using a UART serial interface. Access to the FRAM memory via the BSL is protected by a 256-bit user-defined password. For more details, see the *MSP430 Programming via the Bootstrap Loader User's Guide* (SLAU319).

5.4.3 Programming FRAM Memory via Custom Solution

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

5.5 Wait State Control

The system clock for the CPU or DMA may exceed the FRAM access and cycle time requirements. For these scenarios, a wait state generator mechanism is implemented. There are two modes to control the wait state generation - automatic and manual. When required, the system clock, CPU or DMA, is held until the FRAM access and cycle time constraints are met.

5.5.1 Manual Wait State Control

The complete FRAM cycle time is defined by two timings - access time and pre-charge time which can be defined separately. The cycle time is assumed to be the sum of the access and pre-charge times. If automatic wait state control is disabled (NAUTO=0) and if the clock is set higher than the maximum FRAM access frequency, NACCESS[2:0] and NPRECHG[2:0] have to be set properly to permit correct FRAM access.

The NACCESS bits can be used to define an integer number of CPU cycles required for access time described in the data sheet. The PRECHG bits can be used to define an integer number of CPU cycles required for pre-charge time described in the data sheet. For example, if the memory requires 1 entire cycle for access and 1 entire cycle for pre-charge time, NACCESS[2:0] and NPRECHG[2:0] bits have to be set to 1 (b001) each. For some devices, the values for NACCESS[2:0] and NPRECHG[2:0] are limited to a upper boundary.

By having independent access and pre-charge wait state control, the performance of the overall system can be optimized for a variety of slow memory types. The sum of NACCESS and NPRECHG should be set to equal the overall FRAM cycle time requirement. See the device-specific data sheet for required FRAM timings.

5.5.2 Automatic Wait State Control

The Automatic Mode is the default mode and after a boot the NAUTO bit is set to 1. The wait state is controlled by an internal FRAM state machine and the CPU is held when an access is executed. Manual settings in the NACCESS and NPRECHG have no influence when the NAUTO bit is set. The wait state is automatically adapted if an FRAM Cache hit as explained in Section 5.5.3 occurs.

5.5.3 Wait State and Cache Hit

The FRAM controller contains a cache with two cache sets. Each of these cache sets contents two lines which are pre-loaded with 4 words (64 bits) during one access cycle. An intelligent logic selects one of the cache lines to pre-load FRAM data and preserve recent accessed data in the other cache. If one of the 4 words stored in one of the cache lines is requested (a cache hit), no FRAM access occurs but a cache request. Upon a cache request, no wait state is needed and the data is accessed with full system speed. However if none of the words available in the cache are requested (a cache miss), the wait state controls the CPU to ensure proper FRAM access.

5.5.4 Safe Access

The Safe Access is implemented to ensure correct FRAM access in Manual Wait State Mode.

Safe Access is active when the user configures the NACCESS[2:0] and NPRECHG[2:0] bits to a value that does not meet the required FRAM timing for the given clock setting. In this case, the Safe Access logic ensures the correct timing for the access. The Access Time Error flag (ACCTEIFG) is set. A System NMI (SYSNMI) occurs when ACCTEIE is set.

5.6 FRAM ECC

The FRAM supports bit error correction and uncorrectable bit error detection. The UBDIFG FRAM uncorrectable bit error flag is set if an uncorrectable bit error has been detected in the FRAM memory error detection logic. The CBDIFG FRAM correctable bit error flag is set if a correctable bit error has been detected and corrected. UBDRSTEN enable a Power Up Clear (PUC) reset if an uncorrectable bit error is detected, UBDIEN enables a NMI event if an uncorrectable bit error is detected. CBDIEN enables a NMI event if a correctable bit error is detected and corrected.

5.7 FRCTL Module Registers

The FRCTL registers are listed inTable 5-1 . The base address of the FRCTL module can be found in the device-specific data sheet. The address offset of each FRCTL register is given inTable 5-1 . The password defined in the FRCTLCTL register controls access to all FRCTL 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 FRCTLCTL upper byte. Word accesses to FRCTLCTL with a wrong password triggers a PUC. A write access to a register other than FRCTLCTL while write access is not enabled causes a PUC.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 5-1. FRAM Controller Register

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
FRAM Controller Control 0	FRCTLCTL0	Read/write	Word	00h	9608h
	FRCTLCTL0_L	Read/Write	Byte	00h	08h
	FRCTLCTL0_H	Read/Write	Byte	01h	96h
General Control 0	GCCTL0	Read/write	Word	04h	0000h
	GCCTL0_L	Read/Write	Byte	04h	00h
	GCCTL0_H	Read/Write	Byte	05h	00h

Table 5-1. FRAM Controller Register (continued)

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
General Control 1	GCCTL1	Read/write	Word	06h	0000h
	GCCTL1_L	Read/Write	Byte	06h	00h
	GCCTL1_H	Read/Write	Byte	07h	00h

FRAM Controller Control Register 0 (FRCTLCTL0)

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

7	6	5	4	3	2	1	0
Reserved	**NACCESS[2]**	**NACCESS[1]**	**NACCESS[0]**	**NAUTO**	**NPRECHG[2]**	**NPRECHG[1]**	**NPRECHG[0]**
r-0	rw-[0]	rw-[0]	rw-[0]	rw-[1]	rw-[0]	rw-[0]	rw-[0]

FRCTLPW Bits 15–8 FRCTLPW Password. Always read as 096h. Must be written as 0A5h or a PUC is generated on word write. After a correct password is written and MPU register access is enabled, a wrong password write in byte mode disables the access, and no PUC is generated.

Reserved Bit 7 Reserved. Always read 0.

NACCESS Bits 6-4 Wait state generator access time control. Each wait state adds a N integer multiple increase of the IFCLK period where N = 0 through 7. N = 0 implies no wait states.

NAUTO Bit 3 Disables the wait state generator and manual settings rather controls wait state with internal FRAM state machine

0 Manual Mode

1 Auto Mode

NPRECHG Bits 2-0 Wait state generator pre-charge time control. Each wait state adds a N integer multiple increase of the IFCLK period where N = 0 through 7. N = 0 implies no wait states.

General Control Register 0 (GCCTL0)

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
UBDRSTEN	**UBDIE**	**CBDIE**	**ACCVIE**	**ACCTEIE**	**Reserved**	**Reserved**	**BUSY**
rw-[0]	rw-[0]	rw-[0]	rw-[0]	rw-[0]	r-0	r-0	r-0

Reserved	Bits 15-8	Reserved. Always read 0.
UBDRSTEN	Bit 7	Enable Power Up Clear (PUC) reset if FRAM uncorrectable bit error detected.
		0 PUC not initiated on uncorrectable bit detection flag.
		1 PUC initiated on uncorrectable bit detection flag. Generates vector in SYSRSTIV.
UBDIEN	Bit 6	Enable NMI event if uncorrectable bit error detected.
		0 Uncorrectable bit detection interrupt disabled.
		1 Uncorrectable bit detection interrupt enabled. Generates vector in SYSSNIV.
CBDIEN	Bit 5	Enable NMI event if correctable bit error detected.
		0 Correctable bit detection interrupt disabled.
		1 Correctable bit detection interrupt enabled. Generates vector in SYSSNIV.
ACCVIE	Bit4	Enable NMI event if Access Violation occurs
		0 Access violation interrupt disabled
		1 Access violation interrupt enabled
ACCTEIE	Bit 3	Enable NMI event if Access time error occurs.
		0 Access violation interrupt disabled
		1 Access violation interrupt enabled
Reserved	Bits 2-1	Reserved. Always read 0.
Busy	Bit 0	Busy. This bit indicated if the FRAM is currently busy programming
		0 Not Busy
		1 Busy

General Control Register (GCCTL1)

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	**Reserved**	**Reserved**	**Reserved**	**ACCTEIFG**	**UBDIFG**	**CBDIFG**	**ACCVIFG**
r-0	r-0	r-0	r-0	rw-[0]	rw-[0]	rw-[0]	rw-[0]

Reserved Bits 15-4 Reserved. Always read 0.

ACCTEIFG Bit 3 Access time error flag. This interrupt flag is set if a wrong setting for NPRECHG and NACCESS is set and fram access time is not hold. This bit is cleared by software or by reading the system NMI vector word SYSSNIV if it is the highest pending interrupt flag. This bit is write 0 only and write 1 has no effect

0 No interrupt pending

1 Interrupt pending. Can be cleared by user or by reading SYSSNIV

UBDIFG Bit 2 FRAM uncorrectable bit error flag. This interrupt flag is set if an uncorrectable bit error has been detected in the FRAM memory error detection logic. This bit is cleared by software or by reading the system NMI vector word SYSSNIV if it is the highest pending interrupt flag. This bit is write 0 only, and write 1 has no effect.

0 No interrupt pending

1 Interrupt pending. Can be cleared by user or by reading SYSSNIV

CBDIFG Bit 1 FRAM correctable bit error flag. This interrupt flag is set if a correctable bit error has been detected and corrected in the FRAM memory error detection logic. This bit is cleared by software or by reading the system NMI vector word SYSSNIV if it is the highest pending interrupt flag. This bit is write 0 only and write 1 has no effect.

0 No interrupt pending

1 Interrupt pending. Can be cleared by user or by reading SYSSNIV

ACCVIFG Bit 0 Access Violation Interrupt Flag. Is set if an access violation is triggered. This bit is cleared by software or by reading the system NMI vector word SYSSNIV if it is the highest pending interrupt flag. This bit is write 0 only and write 1 has no effect.

0 No interrupt pending

1 Interrupt pending. Can be cleared by user or by reading SYSSNIV

Memory Protection Unit (MPU)

This chapter describes the operation of the Memory Protection Unit.

Topic **Page**

6.1 Memory Protection Unit (MPU) Introduction

The MPU protects against accidental writes to designated read-only memory segments or execution of code from a constant memory segment memory. Clearing the MPUENA bit disables the MPU, making the complete memory accessible for read, write, and execute operations. After a BOR, the complete memory is accessible without restrictions for read, write, and execute operations.

MPU features include:

- Main memory can be configured up to three segments of variable size
- Access rights for each segment can be set independently
- Information memory can have its access rights set independently
- All MPU registers are protected from access by password

NOTE: After BOR, no segmentation exists, and the main memory and information memory are accessible by read, write, and execute operations.

An overview of the MPU is shown in Figure 6-1.

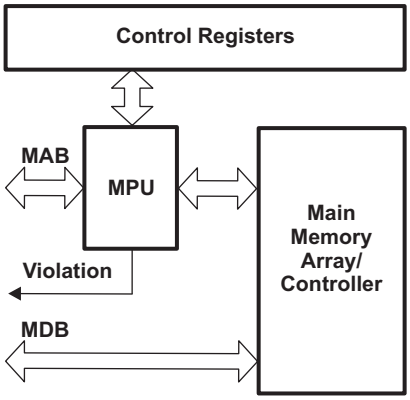


Figure 6-1. Memory Protection Unit Overview

6.2 MPU Segments

6.2.1 Main Memory Segments

The MPU offers the option to logically divide the main memory into three segments. The size of each segment is defined by appropriately setting the borders between adjacent segments. To configure three segments, a lower (B1) and higher (B2) border needs to be programmed by control register bits MPUSB1[4:0] and MPUSB2[4:0] of the MPUSEG register, respectively. Each segment consists of pages. The smallest size of a segment is a page, and therefore sets the granularity of a segment. A page size is restricted to 1/32 of the implemented memory size. For example, a device with a main memory size of 16KB would result in a page size of 512B.

The beginning of segment 1 is the lowest available address for the main memory as defined in the device-specific data sheet. The setting of the lower border (B1) defines the end of segment 1 and the beginning of segment 2. Similarly, the end of segment 2 and beginning of segment 3 is defined by the higher border (B2). Lastly, the end of segment 3 is given by the highest main memory address as defined in the device-specific data sheet. The segmentation of the main memory is shown in Figure 6-2.

The address bus (MAB) is analyzed by the MPU along with the current border settings to determine which segment of memory is selected. If the address is lower than B1 and B2, segment 1 is selected. For address values between B1 and B2, segment 2 is selected. For address values larger than B1 and B2, segment 3 is selected. Setting B1 equal to B2 results in the memory being partitioned in only two segments.

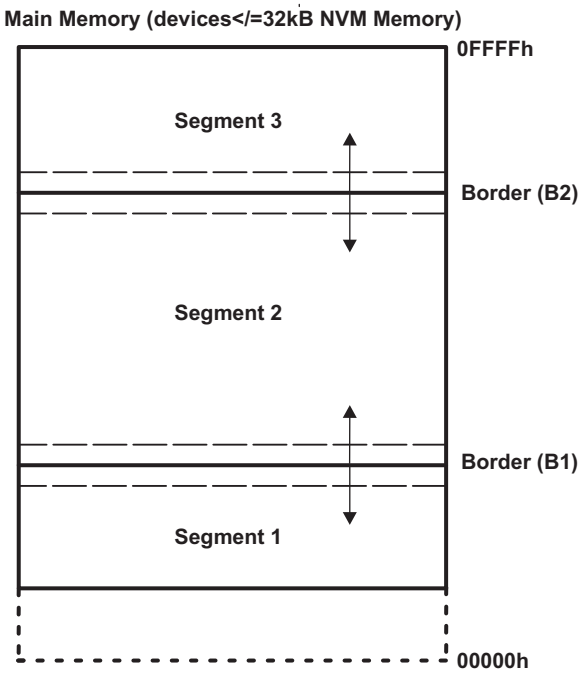


Figure 6-2. Segmentation of Main Memory

6.2.2 Segment Border Setting

Section 6.2.1 describes the procedure of setting borders for segmentation of the main memory. This section describes how the values in MPUSB1[4:0] and MPUSB2[4:0] bits need to be set to achieve the desired borders for different memory sizes. The bits of the MUSBx[4:0] bits represent the five most significant bits of the border address that can be selected. Therefore, the granularity of the border settings and the minimum segment size is 512 bytes in a 16KB device, 256 bytes in a 8KB device, and 128 bytes in a 4KB device.

The main memory always consists of 32 pages, page 0 through page 31. The page size changes based on the size of the available main memory on a device. For example, a 16KB device has a page size of 512B (16KB / 32), an 8KB device has a page size of 256B (8KB / 32) pages, and a 4KB device has a page size of 128B (4KB / 32). The border segments, B1 and B2, can be set to align on any of these 32 pages. The MUSBx[4:0] bits are used to select the appropriate page for the respective borders.

The start address for each page can be computed as follows:

$Page\_Start_n$ = *Maximum Memory Address - Memory Size x (n - 31) / 32 + 1, where n = 0 to 31*

The end address for each page can be computed as follows:

$Page\_End_n$ = *Maximum Memory Address + Memory Size x (n - 31) / 32, where n = 0 to 31*

Table 6-1 shows the results of these calculations for a 16KB, 8KB, and 4KB main memory devices.

Table 6-1. Page Addresses for 16KB, 8KB, and 4KB Main Memory

Page	MUSBx[4:0]	16KB Main Memory		8KB Main Memory		4KB Main Memory	
		Page_Start Address	Page_End Address	Page_Start Address	Page_End Address	Page_Start Address	Page_End Address
0	00h	C000h	C1FFh	E000h	E0FFh	F000h	F07Fh
1	01h	C200h	C3FFh	E100h	E1FFh	F080h	F0FFh
2	02h	C400h	C5FFh	E200h	E2FFh	F100h	F17Fh
3	03h	C600h	C7FFh	E300h	E3FFh	F180h	F1FFh
4	04h	C800h	C9FFh	E400h	E4FFh	F200h	F27Fh
5	05h	CA00h	CBFFh	E500h	E5FFh	F280h	F2FFh
6	06h	CC00h	CDFFh	E600h	E6FFh	F300h	F37Fh
7	07h	CE00h	CFFFh	E700h	E7FFh	F380h	F3FFh
8	08h	D000h	D1FFh	E800h	E8FFh	F400h	F47Fh
9	09h	D200h	D3FFh	E900h	E9FFh	F480h	F4FFh
10	0Ah	D400h	D5FFh	EA00h	EAFFh	F500h	F57Fh
11	0Bh	D600h	D7FFh	EB00h	EBFFh	F580h	F5FFh
12	0Ch	D800h	D9FFh	EC00h	ECFFh	F600h	F67Fh
13	0Dh	DA00h	DBFFh	ED00h	EDFFh	F680h	F6FFh
14	0Eh	DC00h	DDFFh	EE00h	EEFFh	F700h	F77Fh
15	0Fh	DE00h	DFFFh	EF00h	EFFFh	F780h	F7FFh
16	10h	E000h	E1FFh	F000h	F0FFh	F800h	F87Fh
17	11h	E200h	E3FFh	F100h	F1FFh	F880h	F8FFh
18	12h	E400h	E5FFh	F200h	F2FFh	F900h	F97Fh
19	13h	E600h	E7FFh	F300h	F3FFh	F980h	F9FFh
20	14h	E800h	E9FFh	F400h	F4FFh	FA00h	FA7Fh
21	15h	EA00h	EBFFh	F500h	F5FFh	FA80h	FAFFh
22	16h	EC00h	EDFFh	F600h	F6FFh	FB00h	FB7Fh
23	17h	EE00h	EFFFh	F700h	F7FFh	FB80h	FBFFh
24	18h	F000h	F1FFh	F800h	F8FFh	FC00h	FC7Fh
25	19h	F200h	F3FFh	F900h	F9FFh	FC80h	FCFFh
26	1Ah	F400h	F5FFh	FA00h	FAFFh	FD00h	FD7Fh
27	1Bh	F600h	F7FFh	FB00h	FBFFh	FD80h	FDFFh
28	1Ch	F800h	F9FFh	FC00h	FCFFh	FE00h	FE7Fh
29	1Dh	FA00h	FBFFh	FD00h	FDFFh	FE80h	FEFFh
30	1Eh	FC00h	FDFFh	FE00h	FEFFh	FF00h	FF7Fh
31	1Fh	FE00h	FFFFh	FF00h	FFFFh	FF80h	FFFFh

NOTE: Some devices may show a main memory size of less than a power of two. For example, 15.5KB of main memory, as opposed to 16KB. For the page address calculations above, the main memory size should be rounded up to the next power of two, in this case, 16KB. For the 16KB example, page 0 and page 1 settings behave identically.

The following example shows two borders being set on a 16KB device:

- B1 resides at the start of segment 2. If the user wishes to set segment 2 to start at location D800h, this would require MUSB1[4:0] = 0Bh.
- B2 resides at the start of segment 3. If the user wishes to set segment 3 to start at location EE00h, this would require setting MUSB2[4:0] = 17h.
- With these settings, the segment ranges are as follows:
  - Segment 1 resides at C000h through D7FFh.
  - Segment 2 resides at D800h through EDFFh.
  - Segment 3 resides at EE00h through FFFFh.

6.2.3 Information Memory

The information memory is a fixed partition of memory which is 256 bytes in size. The information memory can be used for application specific information (for example, IDs or version numbers), or it can be used for executable code. It is located at address 01800h to 018FFh and is also addressable from 01900h to 019FFh.

6.3 MPU Access Management Settings

Each segment described in Section 6.2.2 and Section 6.2.3 can have read, write, and execute access rights set independently.

The MPUSAM register allows setting the access rights for the four segments (information memory segment, three main memory segments) . MPUSEGxRE enables read access for segment x, MPUSEGxWE enables write access for segment x, and MPUSEGxXE enables code execution from segment x. JTAG/DMA accesses are treated as read or write data accesses and evaluate the corresponding access bits.

Table 6-2 shows the different settings of MPUSEGxXE, MPUSEGxWE, and MPUSEGxRE. Not all settings lead to a different memory protection. For example, as shown, if the execution bit MPUSEGxXE is set to 1, read access is automatically allowed independent of the setting of MPUSEGxRE. Also setting the MPUSEGxWE bit to 1 enables the read option.

NOTE: Combinations that are not shown in Table 6-2 should be avoided, because they may be used in future versions of the MPU.

Table 6-2. Segment Access Rights

MPUSEGxXE	MPUSEGxWE	MPUSEGxRE	Execute Rights	Write Rights	Read Rights
0	0	0	no	no	no
0	0	1	no	no	yes
0	1	1	no	yes	yes
1	0	1	yes	no	yes
1	1	1	yes	yes	yes

NOTE: Prefetching of the CPU can trigger a violation. When a segment contains code that is executed by the CPU, the CPU pipeline prefetches the next two higher words beyond the current Program Counter (PC), and this prefetch is treated as a read/fetch from the MPU perspective. This prefetching also occurs if a "jump" instruction is initiated from the actual address of the PC. A consequence of this can be that a "jump" is the last word in a segment that is open for code execution, but the next higher segment has only read access rights. This causes an access rights violation on executing the "jump". To avoid this, code for execution must stop two words below the highest word of a segment.

6.4 MPU Violations

6.4.1 Interrupt Table and Reset Vector

The interrupt vector table and the reset vector are located at addresses 0FF80h to 0FFFFh. It is possible to define a segment that includes this address space with restricted access rights. If an interrupt or a reset occurs, and this segment is read protected, the MPU automatically allows access to the interrupt vector memory space. In this scenario, only the interrupt vector table is accessible. Access to the interrupt routine itself is not automatically enabled.

NOTE: Only the interrupt table and the reset vector are opened on an interrupt or reset occurrence. If the application protects the segment from execution rights that contains the interrupt routine itself, a violation occurs.

6.4.2 Violation Handling

The handling of access rights violations can be selected for each segment with the MPUSEGxVS bit in the MPUSAM register.

By default (MPUSEGxVS = 0), any access right violation causes the respective violation flag to be set. Setting MPUSEGxVS = 1 causes a PUC to occur upon violation. In either case, the illegal instruction on a protected memory segment is not executed.

Upon an access rights violation, the data bus content (MDB) is driven with 03FFFh until the next valid data is available.

6.5 MPU Lock

The MPU registers can be protected from write access by setting the MPULOCK bit. Write access is not possible on all MPU registers until a BOR occurs. MPULOCK cannot be cleared manually.

6.6 MPU Registers

The MPU registers are listed in Table 6-3. The base address of the MPU module can be found in the device-specific data sheet. The address offset of each MPU register is given in Table 6-3.

The password defined in the MPUCTL0 register controls access to all MPU registers. Once the correct password is written, write access is enabled. Write access is disabled by writing a wrong password in byte mode to the MPUCTL0 upper byte. Word accesses to MPUCTL0 with a wrong password triggers a PUC. A write access to a register other than MPUCTL0 while write access is not enabled causes a PUC.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 6-3. Memory Protection Unit Register

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Memory Protection Unit Control 0	MPUCTL0	Read/write	Word	00h	9600h
	MPUCTL0_L	Read/Write	Byte	00h	00h
	MPUCTL0_H	Read/Write	Byte	01h	96h
Memory Protection Unit Control 1	MPUCTL1	Read/write	Word	02h	0000h
	MPUCTL1_L	Read/Write	Byte	02h	00h
	MPUCTL1_H	Read/Write	Byte	03h	00h
Memory Protection Unit Segmentation Register	MPUSEG	Read/write	Word	04h	0000h
	MPUSEG_L	Read/Write	Byte	04h	00h
	MPUSEG_H	Read/Write	Byte	05h	00h
Memory Protection Unit Segmentation Access Management Register	MPUSAM	Read/write	Word	06h	7777h
	MPUSAM_L	Read/Write	Byte	06h	77h
	MPUSAM_H	Read/Write	Byte	07h	77h
Memory Protection Unit Interrupt Vector Register	MPUIV	Read/write	Word	08h	0000h
	MPUIV_L	Read/Write	Byte	08h	00h
	MPUIV_H	Read/Write	Byte	09h	00h

Memory Protection Unit Control 0 (MPUCTL0)

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

7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**MPULOCK**	**MPUENA**
r-0	r-0	r-0	rw-[0]	r-0	r-0	rw[0]	rw-[0]

MPUPW	Bits 15–8	MPU password. Always read as 096h. Must be written with 0A5h or a PUC is generated on word write. After a correct password is written, all MPU registers are accessible. An incorrect password written in byte mode disables MPU register access and no PUC is generated.
Reserved	Bit 7-5	Reserved. Always read 0.
Reserved	BIt4	Reserved for future usage. Must always be written with 0.
Reserved	Bit 3-2	Reserved. Always read 0.
MPULOCK	Bit 1	MPU lock. If this bit is set, access to all MPU registers except MPUCTL1 are locked and are read only until a BOR occurs. BOR automatically sets this bit to 0. 0 Open 1 Locked
MPUENA	Bit 0	MPU enable. This bit enables the MPU operation. This bit can be set any time with word write and the correct password, if MPULOCK is not already set 0 Disabled 1 Enabled

Memory Protection Unit Control 1 (MPUCTL1)

15	14	13	12	11	10	9	8
Reserved	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**Reserved**	**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	**Reserved**	**Reserved**	**Reserved**	**MPUSEGIIFG**	**MPUSEG3IFG**	**MPUSEG2IFG**	**MPUSEG1IFG**
r-0	r-0	r-0	r-0	rw-[0]	rw-[0]	rw-[0]	rw-[0]

Reserved	Bit 15-4	Reserved. Always read as 0.
MPUSEGIIFG	Bit 3	User information memory violation interrupt flag. This bit is set if an access violation in user information memory is detected. This bit is cleared by software or by reading the reset vector word SYSRSTIV if it is the highest pending interrupt flag. This bit is write 0 only. Write 1 has no effect. 0 No interrupt pending 1 Interrupt pending
MPUSEG3IFG	Bit 2	Main memory segment 3 violation interrupt flag. This bit is set if an access violation in main memory segment 3 is detected. This bit is cleared by software or by reading the reset vector word SYSRSTIV if it is the highest pending interrupt flag. This bit is write 0 only. Write 1 has no effect. 0 No interrupt pending 1 Interrupt pending
MPUSEG2IFG	Bit 1	Main memory segment 2 violation interrupt flag. This bit is set if an access violation in main memory segment 2 is detected. This bit is cleared by software or by reading the reset vector word SYSRSTIV if it is the highest pending interrupt flag. This bit is write 0 only. Write 1 has no effect. 0 No interrupt pending 1 Interrupt pending
MPUSEG1IFG	Bit 0	Main memory segment 1 violation interrupt flag. This bit is set if an access violation in main memory segment 1 is detected. This bit is cleared by software or by reading the reset vector word SYSRSTIV if it is the highest pending interrupt flag. This bit is write 0 only. Write 1 has no effect. 0 No interrupt pending 1 Interrupt pending

Memory Protection Unit Segmentation Register (MPUSEG)

15	14	13	12	11	10	9	8
Reserved	**Reserved**	**Reserved**	**MPUSB2[4]**	**MPUSB2[3]**	**MPUSB2[2]**	**MPUSB2[1]**	**MPUSB2[0]**
r-0	r-0	r-0	rw-[0]	rw-[0]	rw-[0]	rw-[0]	rw-[0]
7	6	5	4	3	2	1	0
Reserved	**Reserved**	**Reserved**	**MPUSB1[4]**	**MPUSB1[3]**	**MPUSB1[2]**	**MPUSB1[1]**	**MPUSB1[0]**
r-0	r-0	r-0	rw-[0]	rw-[0]	rw-[0]	rw-[0]	rw-[0]

Reserved	Bit 15-13	Reserved. Always read as 0.
MPUSB2[4:0]	Bit 12-8	MPU segment border 2. After BOR these bits are automatically set to 0 and only segment 3 is active.
Reserved	Bit 7-5	Reserved. Always read as 0.
MPUSB1[4:0]	Bit 12	MPU segment border 1. After BOR these bits are automatically set to 0 and only segment 3 is active.

Memory Protection Unit Segmentation Access Management Register (MPUSAM)

15	14	13	12	11	10	9	8
MPUSEGIVS	**MPUSEGIXE**	**MPUSEGIWE**	**MPUSEGIRE**	**MPUSEG3VS**	**MPUSEG3XE**	**MPUSEG3WE**	**MPUSEG3RE**
rw-[0]	rw-[1]	rw-[1]	rw-[1]	rw-[0]	rw-[1]	rw-[1]	rw-[1]
7	6	5	4	3	2	1	0
MPUSEG2VS	**MPUSEG2XE**	**MPUSEG2WE**	**MPUSEG2RE**	**MPUSEG1VS**	**]MPUSEG1XE**	**MPUSEG1XE**	**MPUSEG1RE**
rw-[0]	rw-[1]	rw-[1]	rw-[1]	rw-[0]	rw-[1]	rw-[1]	rw-[1]

MPUSEGIVS Bit 15 MPU user information memory segment violation select. If set, a PUC must be executed on illegal access to user information memory.

0 Violation in user information memory asserts the MPUSEGIIFG bit.

1 Violation in user information memory asserts the MPUSEGIIFG bit and a PUC is executed.

MPUSEGIXE Bit 14 MPU user information memory segment execute enable. if set, this bit enables execution in user information memory.

0 Execution in user information memory causes a violation

1 Execution in user information memory is allowed

MPUSEGIWE Bit 13 MPU user information memory segment write enable. If set, this bit enables write access of user information memory.

0 Writes to user information memory cause a violation

1 Writes to user information memory are allowed

MPUSEGIRE Bit 12 MPU user information memory segment read enable. If set, this bit enables read access of user information memory.

0 Reads of user information memory causes a violation if MPUSEGIWE = MPUSEGIXE = 0

1 Reads of user information memory is allowed

MPUSEG3VS Bit 11 MPU main memory segment 3 violation select. If set, a PUC must be executed on illegal access to main memory segment 3.

0 Violation in main memory segment 3 asserts the MPUSEG3IFG bit.

1 Violation in main memory segment 3 asserts the MPUSEG3IFG bit and a PUC is executed.

MPUSEG3XE Bit 10 MPU main memory segment 3 execute enable. If set this bit enables execution in main memory segment 3.

0 Execution in main memory segment 3 causes a violation

1 Execution in main memory segment 3 is allowed

MPUSEG3WE Bit 9 MPU main memory segment 3 write enable. If set this bit enables write access of main memory segment 3.

0 Writes to main memory segment 3 cause a violation

1 Writes to main memory segment 3 are allowed

MPUSEG3RE Bit 8 MPU main memory segment 3 read enable. If set this bit enables read access of main memory segment 3.

0 Reads of main memory segment 3 cause a violation if MPUSEG3WE = MPUSEG3XE = 0

1 Reads of main memory segment 3 are allowed

MPUSEG2VS Bit 7 MPU main memory segment 2 violation Select. If set, a PUC must be executed on illegal access to main memory segment 2.

0 Violation in main memory segment 2 asserts the MPUSEG2IFG bit.

1 Violation in main memory segment 2 asserts the MPUSEG2IFG bit and a PUC is executed.

MPUSEG2XE Bit 6 MPU main memory segment 2 execute enable. If set this bit enables execution in main memory segment 2.

0 Execution in main memory segment 2 causes a violation

1 Execution in main memory segment 2 is allowed

MPUSEG2WE Bit 5 MPU main memory segment 2 write enable. If set this bit enables write access of main memory segment 2.

0 Writes to main memory segment 2 cause a violation

1 Writes to main memory segment 2 are allowed

MPUSEG2RE Bit 4 MPU main memory segment 2 read enable. If set this bit enables read access of main memory segment 2.

0 Reads of main memory segment 2 cause a violation if MPUSEG2WE = MPUSEG2XE = 0

1 Reads of main memory segment 3 are allowed

(continued)

MPUSEG1VS	Bit 3	MPU main memory segment 1 violation select. If set, a PUC must be executed on illegal access to main memory segment 1. 0 Violation in main memory segment 1 asserts the MPUSEG1IFG bit. 1 Violation in main memory segment 1 asserts the MPUSEG1IFG bit and a PUC is executed.
MPUSEG1XE	Bit 2	MPU main memory segment 1 execute enable. If set this bit enables execution in main memory segment 1. 0 Execution in main memory segment 1 causes a violation 1 Execution in main memory segment 1 is allowed
MPUSEG1WE	Bit 1	MPU main memory segment 1 write enable. If set this bit enables write access of main memory segment 1. 0 Writes to main memory segment 1 cause a violation 1 Writes to main memory segment 1 are allowed
MPUSEG1RE	Bit 0	MPU main memory segment 1 read enable. If set this bit enables read access of main memory segment 1. 0 Reads of main memory segment 1 cause a violation if MPUSEG1WE = MPUSEG1XE = 0 1 Reads of main memory segment 1 are allowed

DMA Controller

The direct memory access (DMA) controller module transfers data from one address to another, without CPU intervention. This chapter describes the operation of the DMA controller.

Topic **Page**

7.1 Direct Memory Access (DMA) Introduction

The 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 ADC 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. See the device-specific data sheet for number of channels supported.

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.

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 7-1.

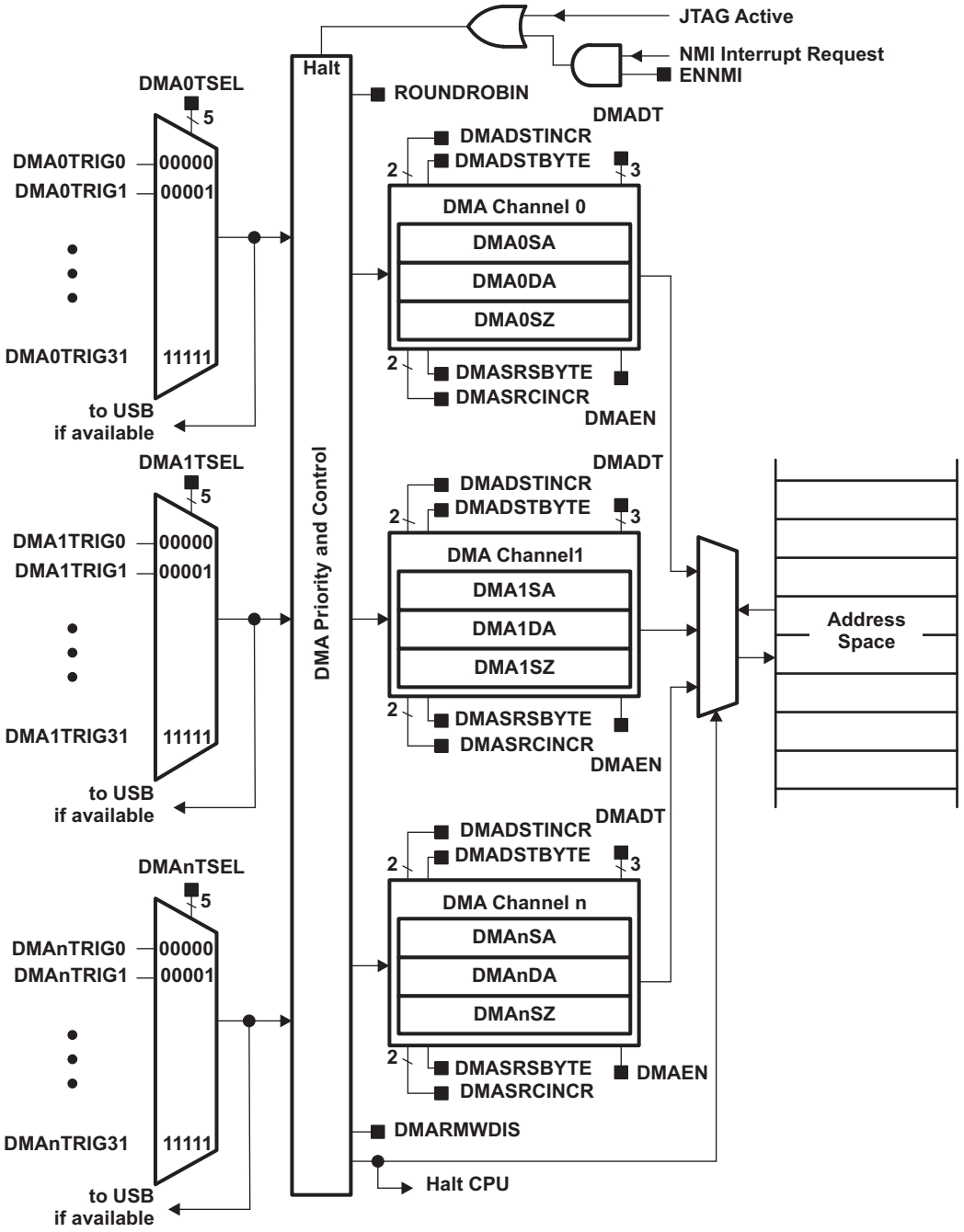


Figure 7-1. DMA Controller Block Diagram

7.2 DMA Operation

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

7.2.1 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 7-2. 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 DMASRCINCR and DMADSTINCR control bits. The DMASRCINCR bits select if the source address is incremented, decremented, or unchanged after each transfer. The DMADSTINCR 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.

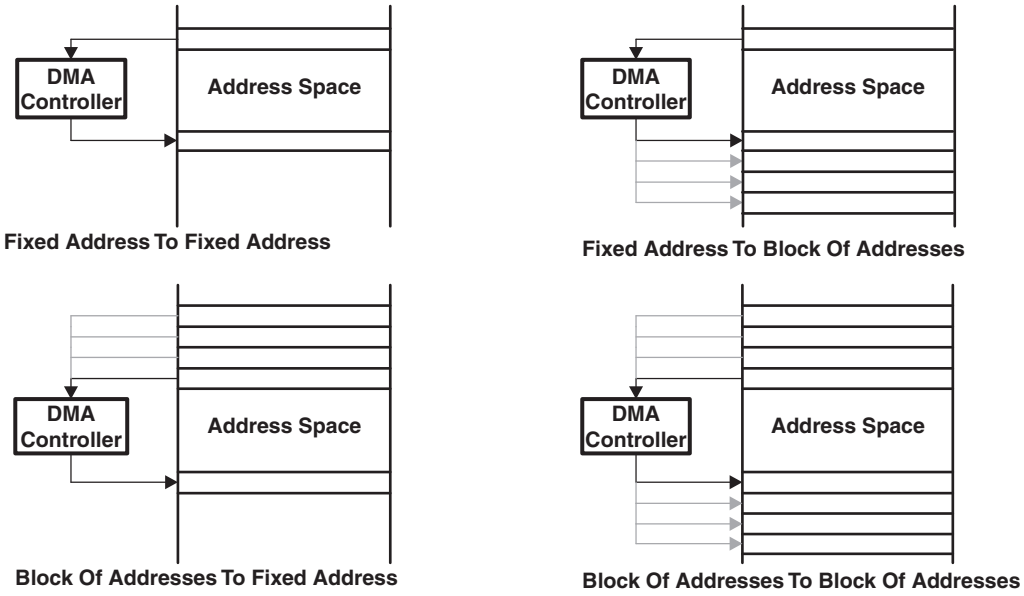


Figure 7-2. DMA Addressing Modes

7.2.2 DMA Transfer Modes

The DMA controller has six transfer modes selected by the DMADT bits as listed in Table 7-1. 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 7-1. DMA Transfer Modes

DMADT	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.

7.2.2.1 Single Transfer

In single transfer mode, each byte/word transfer requires a separate trigger. The single transfer state diagram is shown in Figure 7-3.

The DMAxSZ register is used to define the number of transfers to be made. The DMADSTINCR and DMASRCINCR 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 DMADT = {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.

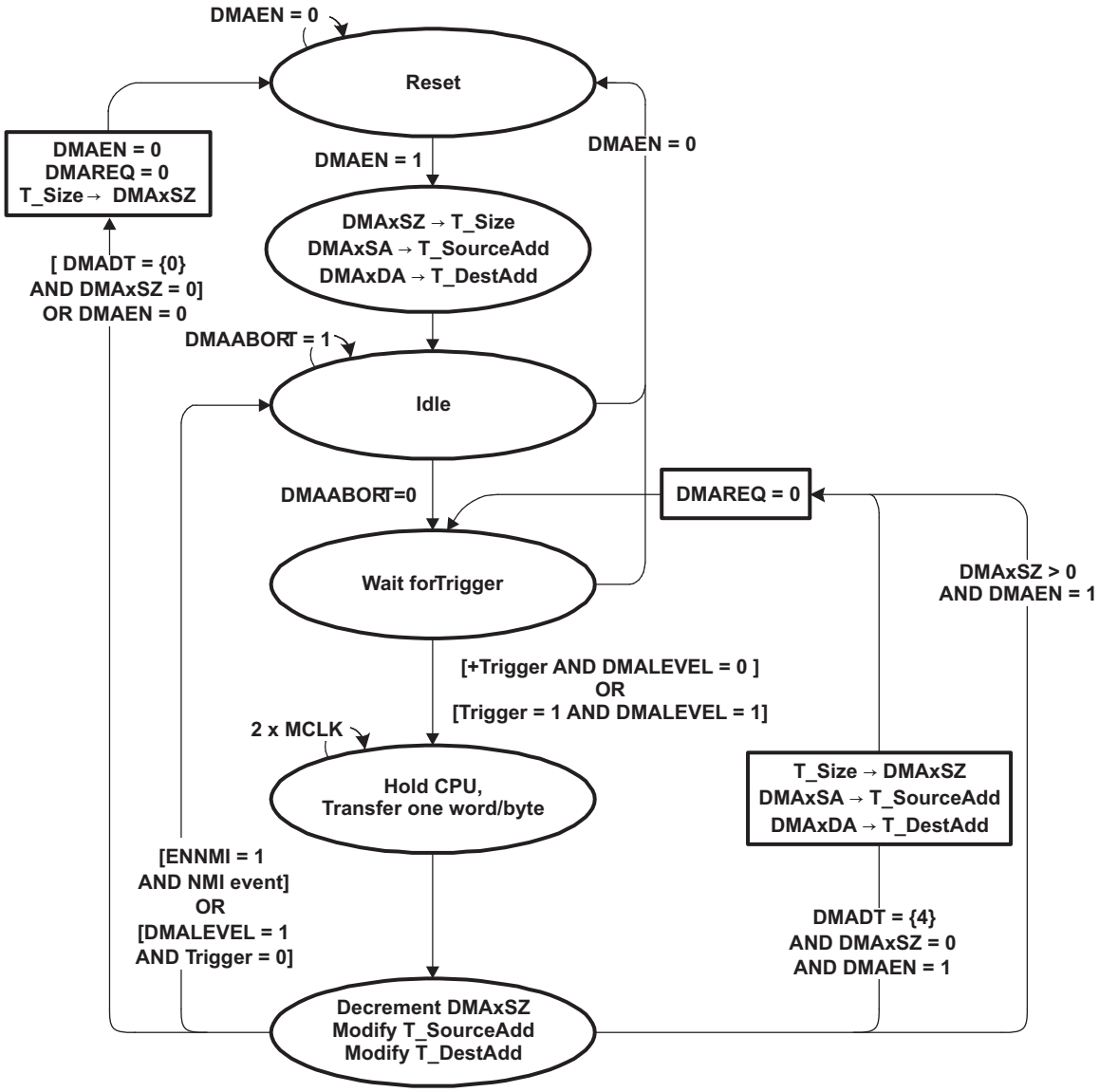


Figure 7-3. DMA Single Transfer State Diagram

7.2.2.2 Block Transfer

In block transfer mode, a transfer of a complete block of data occurs after one trigger. When DMADT = {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 7-4.

The DMAxSZ register is used to define the size of the block, and the DMADSTINCR and DMASRCINCR 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 × MCLK × 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.

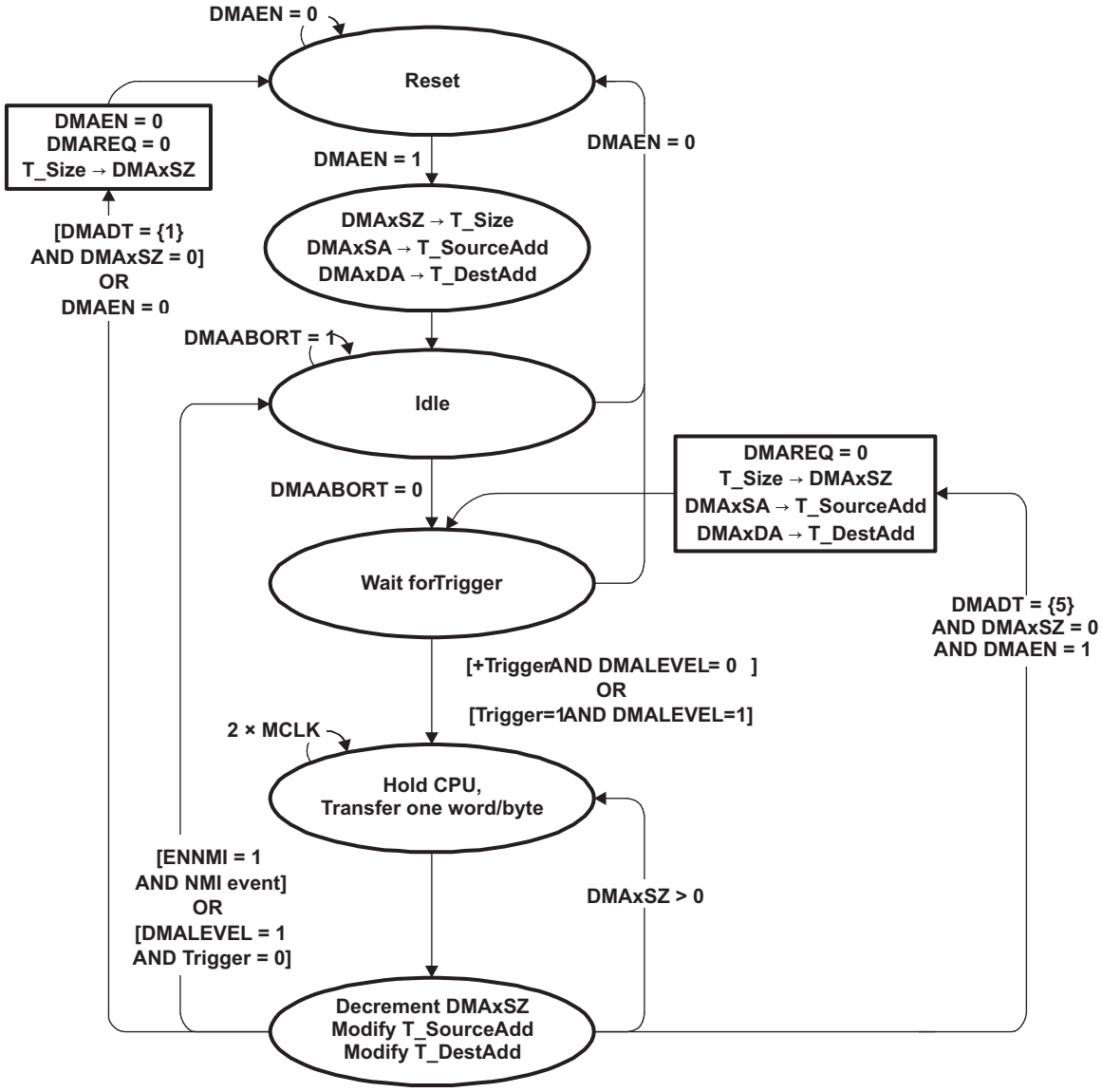


Figure 7-4. DMA Block Transfer State Diagram

7.2.2.3 Burst-Block Transfer

In burst-block mode, transfers are block transfers with CPU activity interleaved. The CPU executes two 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 7-5.

The DMAxSZ register is used to define the size of the block, and the DMADSTINCR and DMASRCINCR 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 (non)maskable interrupt (NMI) when ENNMI is set. In repeated burst-block mode the CPU executes at 20% capacity continuously until the repeated burst-block transfer is stopped.

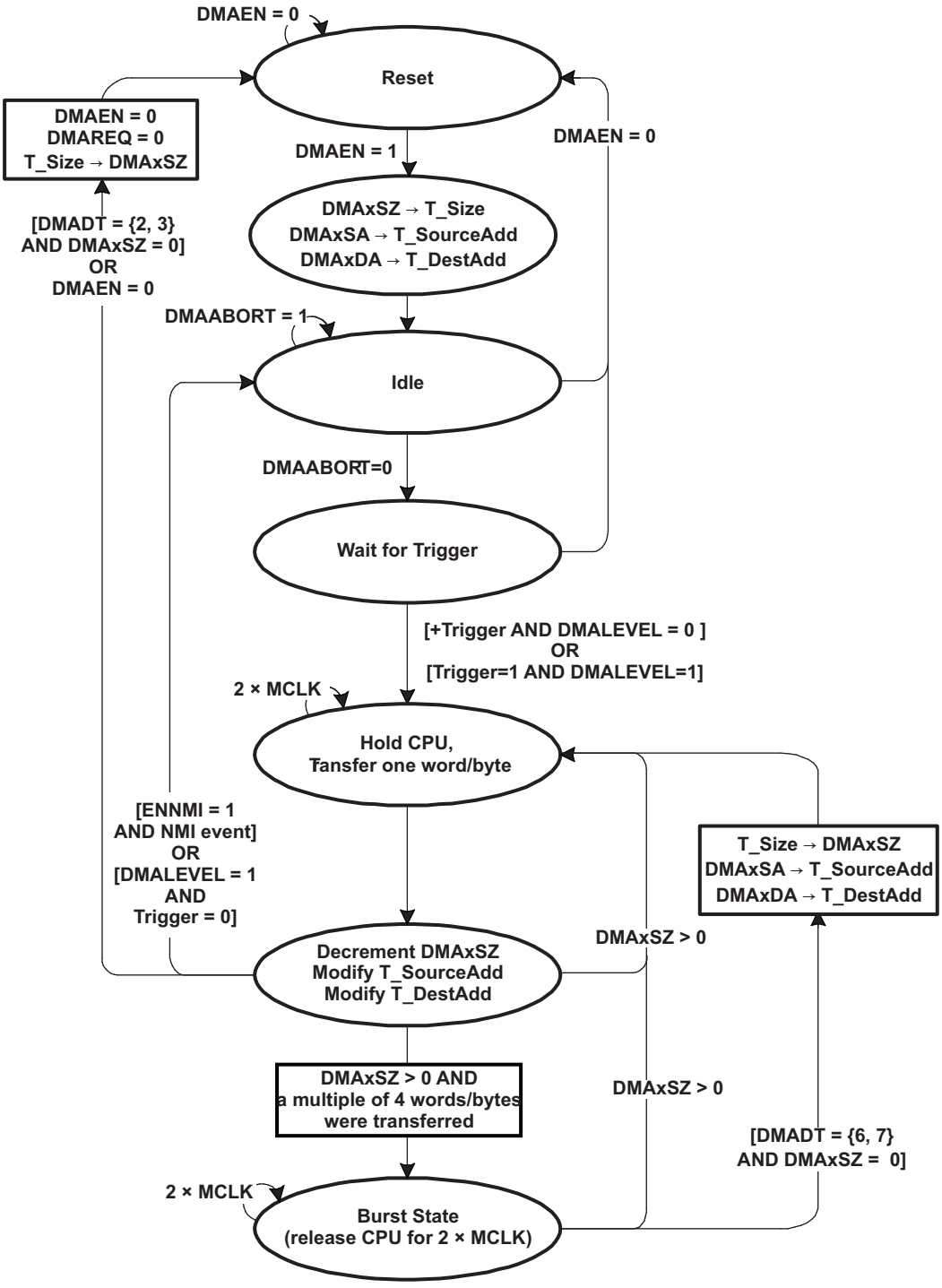


Figure 7-5. DMA Burst-Block Transfer State Diagram

7.2.3 Initiating DMA Transfers

Each DMA channel is independently configured for its trigger source with the DMAxTSEL. The DMAxTSEL bits should be modified only when the DMACTLx DMAEN bit is 0. Otherwise, unpredictable DMA triggers may occur.Table 7-2 describes the trigger operation for each type of module. See the device-specific data sheet for the list of triggers available, along with their respective DMAxTSEL values.

When selecting the trigger, the trigger must not have already occurred, or the transfer does 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 (see the USB module description for further details).

7.2.3.1 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.

7.2.3.2 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 DMADT = {0, 1, 2, 3} are recommended because the DMAEN bit is automatically reset after the configured transfer.

7.2.4 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 7-2).

Table 7-2. 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 TAxCCR0 CCIFG flag is set. The TAxCCR0 CCIFG flag is automatically reset when the transfer starts. If the TAxCCR0 CCIE bit is set, the TAxCCR0 CCIFG flag dies not trigger a transfer. A transfer is triggered when the TAxCCR2 CCIFG flag is set. The TAxCCR2 CCIFG flag is automatically reset when the transfer starts. If the TAxCCR2 CCIE bit is set, the TAxCCR2 CCIFG flag does not trigger a transfer.
Timer_B	A transfer is triggered when the TBxCCR0 CCIFG flag is set. The TBxCCR0 CCIFG flag is automatically reset when the transfer starts. If the TBxCCR0 CCIE bit is set, the TBxCCR0 CCIFG flag does not trigger a transfer. A transfer is triggered when the TBxCCR2 CCIFG flag is set. The TBxCCR2 CCIFG flag is automatically reset when the transfer starts. If the TBxCCR2 CCIE bit is set, the TBxCCR2 CCIFG flag does not trigger a transfer.
eUSCI_Ax	A transfer is triggered when eUSCI_Ax receives new data. UCAxRXIFG is automatically reset when the transfer starts. If UCAxRXIE is set, the UCAxRXIFG does not trigger a transfer. A transfer is triggered when eUSCI_Ax is ready to transmit new data. UCAxTXIFG is automatically reset when the transfer starts. If UCAxTXIE is set, the UCAxTXIFG does not trigger a transfer.
eUSCI_Bx	A transfer is triggered when eUSCI_Bx receives new data. UCBxRXIFG is automatically reset when the transfer starts. If UCBxRXIE is set, the UCBxRXIFG does not trigger a transfer. A transfer is triggered when eUSCI_Bx is ready to transmit new data. UCBxTXIFG is automatically reset when the transfer starts. If UCBxTXIE is set, the UCBxTXIFG does not trigger a transfer.
ADC10_B	A transfer is triggered by an ADC10IFG0 flag. A transfer is triggered when the conversion is completed and the ADC10IFG0 is set. Setting the ADC10IFG0 with software does not trigger a transfer. The ADC10IFG0 flag is automatically reset when the ADC10MEM0 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.

7.2.5 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, if the ENNMI bit is set in register DMACTL1.
- A burst-block transfer may be stopped by clearing the DMAEN bit.

7.2.6 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. When the ROUNDROBIN bit is cleared, the channel priority returns to the default priority.

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

7.2.7 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 uses the MCLK source for each transfer, without reenabling the CPU. If the MCLK source is off, the DMA controller temporarily restarts 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 7-3.

Table 7-3. 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 µs are needed to start the DCOCLK. It is the $t_{(LPMx)}$ parameter in the data sheet.

7.2.8 Using DMA With System Interrupts

DMA transfers are not interruptible by system interrupts. System interrupts remain pending until the completion of the transfer. NMIs 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.

7.2.9 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 (PC) 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 generates another interrupt.

7.2.9.1 DMAIV Software Example

The following software example shows the recommended use of DMAIV and the handling overhead for an eight 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
```

7.2.10 Using the eUSCI_B I²C Module With the DMA Controller

The eUSCI_B I²C module provides two trigger sources for the DMA controller. The eUSCI_B I²C module can trigger a transfer when new I²C data is received and the when the transmit data is needed.

7.2.11 Using ADC10 With the DMA Controller

MSP430 devices with an integrated DMA controller can automatically move data from the ADC10MEM0 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 ADC10 module, and enhances low-power applications allowing the CPU to remain off while data transfers occur. A transfer is triggered when the conversion is completed and the ADC10IFG0 is set. Setting the ADC10IFG0 with software does not trigger a transfer. The ADC10IFG0 flag is automatically reset when the ADC10MEM0 register is accessed by the DMA controller.

7.3 DMA Registers

The DMA module registers are listed in Table 7-4. The base addresses can be found in the device-specific data sheet. Each channel starts at its respective base address. The address offsets are listed in Table 7-4.

Table 7-4. DMA Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
DMA Control 0	DMACTL0	Read/write	Word	00h	0000h
DMA Control 1	DMACTL1	Read/write	Word	02h	0000h
DMA Control 2	DMACTL2	Read/write	Word	04h	0000h
DMA Control 3	DMACTL3	Read/write	Word	06h	0000h
DMA Control 4	DMACTL4	Read/write	Word	08h	0000h
DMA Interrupt Vector	DMAIV	Read only	Word	0Eh	0000h
DMA Channel 0 Control	DMA0CTL	Read/write	Word	00h	0000h
DMA Channel 0 Source Address	DMA0SA	Read/write	Word, double word	02h	undefined
DMA Channel 0 Destination Address	DMA0DA	Read/write	Word, double word	06h	undefined
DMA Channel 0 Transfer Size	DMA0SZ	Read/write	Word	0Ah	undefined
DMA Channel 1 Control	DMA1CTL	Read/write	Word	00h	0000h
DMA Channel 1 Source Address	DMA1SA	Read/write	Word, double word	02h	undefined
DMA Channel 1 Destination Address	DMA1DA	Read/write	Word, double word	06h	undefined
DMA Channel 1 Transfer Size	DMA1SZ	Read/write	Word	0Ah	undefined
DMA Channel 2 Control	DMA2CTL	Read/write	Word	00h	0000h
DMA Channel 2 Source Address	DMA2SA	Read/write	Word, double word	02h	undefined
DMA Channel 2 Destination Address	DMA2DA	Read/write	Word, double word	06h	undefined
DMA Channel 2 Transfer Size	DMA2SZ	Read/write	Word	0Ah	undefined
DMA Channel 3 Control	DMA3CTL	Read/write	Word	00h	0000h
DMA Channel 3 Source Address	DMA3SA	Read/write	Word, double word	02h	undefined
DMA Channel 3 Destination Address	DMA3DA	Read/write	Word, double word	06h	undefined
DMA Channel 3 Transfer Size	DMA3SZ	Read/write	Word	0Ah	undefined
DMA Channel 4 Control	DMA4CTL	Read/write	Word	00h	0000h
DMA Channel 4 Source Address	DMA4SA	Read/write	Word, double word	02h	undefined
DMA Channel 4 Destination Address	DMA4DA	Read/write	Word, double word	06h	undefined
DMA Channel 4 Transfer Size	DMA4SZ	Read/write	Word	0Ah	undefined
DMA Channel 5 Control	DMA5CTL	Read/write	Word	00h	0000h
DMA Channel 5 Source Address	DMA5SA	Read/write	Word, double word	02h	undefined
DMA Channel 5 Destination Address	DMA5DA	Read/write	Word, double word	06h	undefined
DMA Channel 5 Transfer Size	DMA5SZ	Read/write	Word	0Ah	undefined
DMA Channel 6 Control	DMA6CTL	Read/write	Word	00h	0000h
DMA Channel 6 Source Address	DMA6SA	Read/write	Word, double word	02h	undefined
DMA Channel 6 Destination Address	DMA6DA	Read/write	Word, double word	06h	undefined
DMA Channel 6 Transfer Size	DMA6SZ	Read/write	Word	0Ah	undefined

Table 7-4. DMA Registers (continued)

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
DMA Channel 7 Control	DMA7CTL	Read/write	Word	00h	0000h
DMA Channel 7 Source Address	DMA7SA	Read/write	Word, double word	02h	undefined
DMA Channel 7 Destination Address	DMA7DA	Read/write	Word, double word	06h	undefined
DMA Channel 7 Transfer Size	DMA7SZ	Read/write	Word	0Ah	undefined

DMA Control 0 Register (DMACTL0)

15	14	13	12	11	10	9	8
Reserved			**DMA1TSEL**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved			**DMA0TSEL**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Reserved Bits 15-13 Reserved. Read only. Always read as 0.

DMA1TSEL Bits 12-8 DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet 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.

DMA0TSEL Bits 4-0 Same as DMA1TSEL

DMA Control 1 Register (DMACTL1)

15	14	13	12	11	10	9	8
Reserved			**DMA3TSEL**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved			**DMA2TSEL**				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Reserved Bits 15-13 Reserved. Read only. Always read as 0.

DMA3TSEL Bits 12-8 DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet 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.

DMA2TSEL Bits 4-0 Same as DMA3TSEL

DMA Control 2 Register (DMACTL2)

15	14	13	12	11	10	9	8
Reserved			DMA5TSEL				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved			DMA4TSEL				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Reserved Bits 15-13 Reserved. Read only. Always read as 0.

DMA5TSEL Bits 12-8 DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet 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.

DMA4TSEL Bits 4-0 Same as DMA5TSEL

DMA Control 3 Register (DMACTL3)

15	14	13	12	11	10	9	8
Reserved			DMA7TSEL				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved			DMA6TSEL				
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Reserved Bits 15-13 Reserved. Read only. Always read as 0.

DMA7TSEL Bits 12-8 DMA trigger select. These bits select the DMA transfer trigger. See the device-specific data sheet 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.

DMA6TSEL Bits 4-0 Same as DMA7TSEL

DMA Control 4 Register (DMACTL4)

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. When set, this bit 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. When an NMI interrupts a DMA transfer, the current transfer is completed normally, further transfers are stopped and DMAABORT is set.

0 NMI does not interrupt DMA transfer.

1 NMI interrupts a DMA transfer.

DMA Channel x Control Register (DMAxCTL)

15	14	13	12	11	10	9	8
Reserved	**DMADT**			**DMADSTINCR**		**DMASRCINCR**	
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.

DMADT 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

DMADSTINCR 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.

DMASRCINCR 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

DMA Channel x Control Register (DMAxCTL) (continued)

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 interrupted by NMI
DMAREQ	Bit 0	DMA request. Software-controlled DMA start. DMAREQ is reset automatically.
		0 No DMA start
		1 Start DMA

DMA Source Address Register (DMAxSA)

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

DMA Destination Address Register (DMAxDA)

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

Reserved	Bits 31-20	Reserved. Read only. Always read as 0.
DMAxDA	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.

DMA Size Address Register (DMAxSZ)

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

DMAxSZ 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.

DMA Interrupt Vector Register (DMAIV)

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	DMAIV					0
r0	r0	r-(0)	r-(0)	r-(0)	r-(0)	r-(0)	r0

DMAIV 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

Digital I/O

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

Topic **Page**

8.1 Digital I/O Introduction

The digital I/O features include:

- Independently programmable individual I/Os
- Any combination of input or output
- Individually configurable P1 and P2 interrupts. Some devices may include additional port interrupts.
- Independent input and output data registers
- Individually configurable pullup or pulldown resistors

Devices within the family may have up to twelve digital I/O ports implemented (P1 to P11 and PJ). Most ports contain eight I/O lines; however, some ports may contain less (see the device-specific data sheet for ports available). Each I/O line is individually configurable for input or output direction, and each can be individually read or written. Each I/O line is individually configurable for pullup or pulldown resistors.

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 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 devices, additional ports with interrupt capability may be available (see the device-specific data sheet for details) and contain their own respective interrupt vectors.

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. All port registers are handled in this manner with this naming convention except for the interrupt vector registers, P1IV and P2IV; i.e. PAIV does not exist.

When writing to port PA with word operations, all 16 bits are written to the port. When 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. When writing to a port that contains less than the maximum number of bits possible, the unused bits are a "don't care". Ports PB, PC, PD, PE, and PF behave similarly.

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 is cleared automatically. Ports PB, PC, PD, PE, and PF behave similarly. When reading from ports that contain less than the maximum bits possible, unused bits are read as zeros (similarly for port PJ).

8.2 Digital I/O Operation

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

8.2.1 Input Registers 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: Input is low
- Bit = 1: 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.

8.2.2 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: Output is low
- Bit = 1: Output is high

If the pin is configured as I/O function, input direction and the pullup/pulldown resistor are enabled; the corresponding bit in the PxOUT register selects pullup or pulldown.

- Bit = 0: Pin is pulled down
- Bit = 1: Pin is pulled up

8.2.3 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

8.2.4 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 contains a pullup or pulldown.

- Bit = 0: Pullup/pulldown resistor disabled
- Bit = 1: Pullup/pulldown resistor enabled

Table 8-1 summarizes the usage of PxDIR, PxREN, and PxOUT for proper I/O configuration.

Table 8-1. I/O Configuration

PxDIR	PxREN	PxOUT	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

8.2.5 Function Select Registers PxSEL0, PxSEL1

Port pins are often multiplexed with other peripheral module functions. See the device-specific data sheet to determine pin functions. Each port pin uses two bits to select the pin function – I/O port or one of the three possible peripheral module function. Table 8-2 shows how to select the various module functions. See the device-specific data sheet to determine pin functions. Each PxSEL bit is used to select the pin function – I/O port or peripheral module function.

Table 8-2. I/O Function Selection

PxSEL1	PxSEL0	I/O Function
0	0	general purpose I/O is selected
0	1	primary module function is selected
1	0	secondary module function is selected
1	1	ternary module function is selected

Setting the PxSEL1 or PxSEL0 bits to a module function does not automatically set the pin direction. Other peripheral module functions may require the PxDIR bits to be configured according to the direction needed for the module function. See the pin schematics in the device-specific datasheet.

When a port pin is selected as an input to peripheral modules, the input signal to those peripheral modules is a latched representation of the signal at the device pin. While PxSEL1 and PxSEL0 is other than “00”, the internal input signal follows the signal at the pin for all connected modules. However, if PxSEL1 and PxSEL0 = "00”, the input to the peripherals maintain the value of the input signal at the device pin before the PxSEL1 and PxSEL0 bits were reset.

Since the PxSEL1 and PxSEL0 bits do not reside in contiguous addresses, changing both bits at once is not possible at the same time. For example, we wish to change P1.0 from general purpose I/O to the ternary module function residing on P1.0. Initially, P1SEL1 = 00h and P1SEL0 = 00h. To change the function, it would be necessary to write both P1SEL1 = 01h and P1SEL0 = 01h. This is not possible without first passing through an intermediate configuration. This may not be desirable from an application standpoint. The PxSELC complement register can be used to handle such situations. The PxSELC register always reads 0. Each set bit of the PxSELC register will complement the corresponding respective bit of the PxSEL1 and PxSEL0 registers. In the previous example, with P1SEL1 = 00h and P1SEL0 = 00h initially, writing P1SELC = 01h will cause P1SEL1 = 01h and P1SEL0 = 01h to be written simultaneously.

NOTE: P1 and P2 interrupts are disabled when PxSEL1 = 1 or PxSEL0 = 1

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

8.2.6 P1 and P2 Interrupts, Port 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. Some devices may contain additional port interrupts besides P1 and P2. Please see the device specific data sheet to determine which port interrupts are available.

Each PxIFG 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 PxIFG 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 PxIFG 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 PxIFG 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 generates another interrupt.

Port P2 interrupts behave similarly, and source a separate single interrupt vector and utilize 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 P1                                  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: Respective PxIFG flag is set with a low-to-high transition
- Bit = 1: Respective PxIFG flag is set with a high-to-low transition

NOTE: Writing to PxIES

Writing to P1IES or P2IES for each corresponding I/O can result in setting the corresponding interrupt flags.

PxIES	PxIN	PxIFG
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

8.2.7 Configuring Unused Port Pins

Unused I/O pins should be configured as I/O function, output direction, and left unconnected on the PC board, to prevent a floating input and reduce power consumption. The value of the PxOUT bit is don't care, because 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 the *System Resets, Interrupts, and Operating Modes, System Control Module (SYS))* chapter for termination of unused pins.

NOTE: Configuring port J and shared JTAG pins:

Application should ensure that port PJ is configured properly to prevent a floating input. Because 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.

8.3 I/O Configuration and LPMx.5 Low-Power Modes

NOTE: The LPMx.5 low power modes may not be available on all devices. The LPM4.5 power mode allows for lowest power consumption and no clocks are available. The LPM3.5 power mode allows for RTC mode operation at the lowest power consumption available. Please refer to the *SYS* chapter for details, as well as, the device specific datasheet for LPMx.5 low power modes that are available. With respect to the digital I/O, this section is applicable for both LPM3.5 and LPM4.5.

The regulator of the Power Management Module (PMM) is disabled upon entering LPMx.5 (LPM3.5 or LPM4.5), which causes all I/O register configurations to be lost. Because the I/O register configurations are lost, the configuration of I/O pins must be handled differently to ensure that all pins in the application behave in a controlled manner upon entering and exiting LPMx.5. Properly setting the I/O pins is critical to achieving the lowest possible power consumption in LPMx.5, as well as preventing any possible uncontrolled input or output I/O state in the application. The application has complete control of the I/O pin conditions preventing the possibility of unwanted spurious activity upon entry and exit from LPMx.5. The detailed flow for entering and exiting LPMx.5 with respect to the I/O operation is as follows:

1. Set all I/Os to general purpose I/Os and configure as needed. Each I/O can be set to input high impedance, input with pulldown, input with pullup, output high (low or high drive strength), or output low (low or high drive strength). It is critical that no inputs are left floating in the application, otherwise excess current may be drawn in LPMx.5. Configuring the I/O in this manner ensures that each pin is in a safe condition prior to entering LPMx.5. Optionally, configure input interrupt pins for wake-up from LPMx.5. To wake the device from LPMx.5, a general-purpose I/O port must contain an input port with interrupt capability. Not all devices include wakeup from LPMx.5 via I/O, and not all inputs with interrupt capability offer wakeup from LPMx.5. See the device-specific data sheet for availability. To configure a port to wake up the device, it should be configured properly prior to entering LPMx.5. Each port should be configured as general-purpose input. Pulldowns or pullups can be applied if required. Setting the PxIES bit of the corresponding register determines the edge transition that wakes the device. Lastly, the PxIE for the port must be enabled, as well as the general interrupt enable.
2. Enter LPMx.5 with LPMx.5 entry sequence, enable general interrupts for wake-up:

```
MOV.B #PMMPW_H, &PMMCTL0_H                ; Open PMM registers for write
BIS.B #PMMREGOFF, &PMMCTL0_L              ;
BIS   #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR     ; Enter LPMx.5 when PMMREGOFF is set
```

3. Upon entry into LPMx.5, LOCKLPM5 residing in PM5CTL0 of the PMM module, is set automatically. The I/O pin states are held and locked based on the settings prior to LPMx.5 entry. Please note that only the pin conditions are retained. All other port configuration register settings such as PxDIR, PxREN, PxOUT, PxDS, PxIES, and PxIE contents are lost.
4. A LPMx.5 wakeup event e.g. an edge on a configured wakeup input pin, will start the BOR entry sequence together with the regulator. All peripheral registers are set to their default conditions. Upon exit from LPMx.5, the I/O pins remain locked while LOCKLPM5 remains set. Keeping the I/O pins locked ensures that all pin conditions remain stable upon entering the active mode regardless of the default I/O register settings.
5. Once in active mode, the I/O configuration and I/O interrupt configuration that was not retained during LPMx.5 should be restored to the values prior to entering LPMx.5. It is recommended to reconfigure the PxIES and PxIE to their previous settings to prevent a false port interrupt from occurring. The LOCKLPM5 bit can then be cleared, which releases the I/O pin conditions and I/O interrupt configuration. Any changes to the port configuration registers while LOCKLPM5 is set, have no effect on the I/O pins.
6. After enabling the I/O interrupts, the I/O interrupt that caused the wakeup can be serviced indicated by the PxIFG flags. These flags can be used directly, or the corresponding PxIV register may be used. Please note that the PxIFG flag cannot be cleared until the LOCKLPM5 bit has been cleared.
7. To re-enter LPMx.5, the LOCKLPM5 bit must be cleared prior to re-entry, otherwise LPMx.5 will not be entered.

NOTE: It is possible that multiple events occurred on various ports. In these cases, multiple PxIFG flags will be set and it cannot be determined which port has caused the I/O wakeup.

8.4 Digital I/O Registers

The digital I/O registers are listed in Table 8-3. The base addresses can be found in the device-specific data sheet. Each port grouping begins at its base address. The address offsets are given in Table 8-3.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 8-3. Digital I/O Registers

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port 1	Interrupt Vector	P1IV	Read only	Word	0Eh	0000h
		P1IV_L	Read only	Byte	0Eh	00h
		P1IV_H	Read only	Byte	0Fh	00h
Port 2	Interrupt Vector	P2IV	Read only	Word	1Eh	0000h
		P2IV_L	Read only	Byte	1Eh	00h
		P2IV_H	Read only	Byte	1Fh	00h
Port 3	Interrupt Vector	P3IV	Read only	Word	2Eh	0000h
		P3IV_L	Read only	Byte	2Eh	00h
		P3IV_H	Read only	Byte	2Fh	00h
Port 4	Interrupt Vector	P4IV	Read only	Word	3Eh	0000h
		P4IV_L	Read only	Byte	3Eh	00h
		P4IV_H	Read only	Byte	3Fh	00h
Port 1	Input	P1IN or PAIN_L	Read only	Byte	00h	
	Output	P1OUT or PAOUT_L	Read/write	Byte	02h	undefined
	Direction	P1DIR or PADIR_L	Read/write	Byte	04h	00h
	Resistor Enable	P1REN or PAREN_L	Read/write	Byte	06h	00h
	Port Select 0	P1SEL0 or PASEL0_L	Read/write	Byte	0Ah	00h
	Port Select 1	P1SEL1 or PASEL1_L	Read/write	Byte	0Ch	00h
	Port complement selection	P1SELC or PASELC_L	Read/write	Byte	10h	00h
	Interrupt Edge Select	P1IES or PAIES_L	Read/write	Byte	18h	undefined
	Interrupt Enable	P1IE or PAIE_L	Read/write	Byte	1Ah	00h
	Interrupt Flag	P1IFG or PAIFG_L	Read/write	Byte	1Ch	00h

Table 8-3. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port 2	Input	P2IN or PAIN_H	Read only	Byte	01h	
	Output	P2OUT or PAOUT_H	Read/write	Byte	03h	undefined
	Direction	P2DIR or PADIR_H	Read/write	Byte	05h	00h
	Resistor Enable	P2REN or PAREN_H	Read/write	Byte	07h	00h
	Port Select 0	P2SEL0 or PASEL0_H	Read/write	Byte	0Bh	00h
	Port Select 1	P2SEL1 or PASEL1_H	Read/write	Byte	0Dh	00h
	Port complement selection	P2SELC or PASELC_L	Read/write	Byte	11h	00h
	Interrupt Edge Select	P2IES or PAIES_H	Read/write	Byte	19h	undefined
	Interrupt Enable	P2IE or PAIE_H	Read/write	Byte	1Bh	00h
	Interrupt Flag	P2IFG or PAIFG_H	Read/write	Byte	1Dh	00h
Port 3	Input	P3IN or PBIN_L	Read only	Byte	00h	
	Output	P3OUT or PBOUT_L	Read/write	Byte	02h	undefined
	Direction	P3DIR or PBDIR_L	Read/write	Byte	04h	00h
	Resistor Enable	P3REN or PBREN_L	Read/write	Byte	06h	00h
	Port Select 0	P3SEL0 or PBSEL0_L	Read/write	Byte	0Ah	00h
	Port Select 1	P3SEL1 or PBSEL1_L	Read/write	Byte	0Ch	00h
	Port complement selection	P3SELC or PBSELC_L	Read/write	Byte	10h	00h
	Interrupt Edge Select	P3IES or PBIES_L	Read/write	Byte	18h	undefined
	Interrupt Enable	P3IE or PBIE_L	Read/write	Byte	1Ah	00h
	Interrupt Flag	P3IFG or PBIFG_L	Read/write	Byte	1Ch	00h

Table 8-3. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port 4	Input	P4IN or PBIN_H	Read only	Byte	01h	
	Output	P4OUT or PBOUT_H	Read/write	Byte	03h	undefined
	Direction	P4DIR or PBDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	P4REN or PBREN_H	Read/write	Byte	07h	00h
	Port Select 0	P4SEL0 or PBSEL0_H	Read/write	Byte	0Bh	00h
	Port Select 1	P4SEL1 or PBSEL1_H	Read/write	Byte	0Dh	00h
	Port complement selection	P4SELC or PBSELC_L	Read/write	Byte	11h	00h
	Interrupt Edge Select	P4IES or PBIES_H	Read/write	Byte	19h	undefined
	Interrupt Enable	P4IE or PBIE_H	Read/write	Byte	1Bh	00h
	Interrupt Flag	P4IFG or PBIFG_H	Read/write	Byte	1Dh	00h
Port 5	Input	P5IN or PCIN_L	Read only	Byte	00h	
	Output	P5OUT or PCOUT_L	Read/write	Byte	02h	undefined
	Direction	P5DIR or PCDIR_L	Read/write	Byte	04h	00h
	Resistor Enable	P5REN or PCREN_L	Read/write	Byte	06h	00h
	Port Select 0	P5SEL0 or PCSEL0_L	Read/write	Byte	0Ah	00h
	Port Select 1	P5SEL1 or PCSEL1_L	Read/write	Byte	0Ch	00h
	Port complement selection	P5SELC or PCSELC_L	Read/write	Byte	10h	00h
	Interrupt Edge Select	P5IES or PCIES_L	Read/write	Byte	18h	undefined
	Interrupt Enable	P5IE or PCIE_L	Read/write	Byte	1Ah	00h
	Interrupt Flag	P5IFG or PCIFG_L	Read/write	Byte	1Ch	00h

Table 8-3. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port 6	Input	P6IN or PCIN_H	Read only	Byte	01h	
	Output	P6OUT or PCOUT_H	Read/write	Byte	03h	undefined
	Direction	P6DIR or PCDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	P6REN or PCREN_H	Read/write	Byte	07h	00h
	Port Select 0	P6SEL0 or PCSEL0_H	Read/write	Byte	0Bh	00h
	Port Select 1	P6SEL1 or PCSEL1_H	Read/write	Byte	0Dh	00h
	Port complement selection	P6SELC or PCSELC_L	Read/write	Byte	11h	00h
	Interrupt Edge Select	P6IES or PCIES_H	Read/write	Byte	19h	undefined
	Interrupt Enable	P6IE or PCIE_H	Read/write	Byte	1Bh	00h
	Interrupt Flag	P6IFG or PCIFG_H	Read/write	Byte	1Dh	00h
Port 7	Input	P7IN or PDIN_L	Read only	Byte	00h	
	Output	P7OUT or PDOUT_L	Read/write	Byte	02h	undefined
	Direction	P7DIR or PDDIR_L	Read/write	Byte	04h	00h
	Resistor Enable	P7REN or PDREN_L	Read/write	Byte	06h	00h
	Port Select 0	P7SEL0 or PDSEL0_L	Read/write	Byte	0Ah	00h
	Port Select 1	P7SEL1 or PDSEL1_L	Read/write	Byte	0Ch	00h
	Port complement selection	P7SELC or PDSELC_L	Read/write	Byte	10h	00h
	Interrupt Edge Select	P7IES or PDIES_L	Read/write	Byte	18h	undefined
	Interrupt Enable	P7IE or PDIE_L	Read/write	Byte	1Ah	00h
	Interrupt Flag	P7IFG or PDIFG_L	Read/write	Byte	1Ch	00h

Table 8-3. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port 8	Input	P8IN or PDIN_H	Read only	Byte	01h	
	Output	P8OUT or PDOUT_H	Read/write	Byte	03h	undefined
	Direction	P8DIR or PDDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	P8REN or PDREN_H	Read/write	Byte	07h	00h
	Port Select 0	P8SEL0 or PDSEL0_H	Read/write	Byte	0Bh	00h
	Port Select 1	P8SEL1 or PDSEL1_H	Read/write	Byte	0Dh	00h
	Port complement selection	P8SELC or PDSELC_L	Read/write	Byte	11h	00h
	Interrupt Edge Select	P8IES or PDIES_H	Read/write	Byte	19h	undefined
	Interrupt Enable	P8IE or PDIE_H	Read/write	Byte	1Bh	00h
	Interrupt Flag	P8IFG or PDIFG_H	Read/write	Byte	1Dh	00h
Port 9	Input	P9IN or PEIN_L	Read only	Byte	00h	
	Output	P9OUT or PEOUT_L	Read/write	Byte	02h	undefined
	Direction	P9DIR or PEDIR_L	Read/write	Byte	04h	00h
	Resistor Enable	P9REN or PEREN_L	Read/write	Byte	06h	00h
	Port Select 0	P9SEL0 or PESEL0_L	Read/write	Byte	0Ah	00h
	Port Select 1	P9SEL1 or PESEL1_L	Read/write	Byte	0Ch	00h
	Port complement selection	P9SELC or PESELC_L	Read/write	Byte	10h	00h
	Interrupt Edge Select	P9IES or PEIES_L	Read/write	Byte	18h	undefined
	Interrupt Enable	P9IE or PEIE_L	Read/write	Byte	1Ah	00h
	Interrupt Flag	P9IFG or PEIFG_L	Read/write	Byte	1Ch	00h

Table 8-3. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port 10	Input	P10IN or PEIN_H	Read only	Byte	01h	
	Output	P10OUT or PEOUT_H	Read/write	Byte	03h	undefined
	Direction	P10DIR or PEDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	P10REN or PEREN_H	Read/write	Byte	07h	00h
	Port Select 0	P10SEL0 or PESEL0_H	Read/write	Byte	0Bh	00h
	Port Select 1	P10SEL1 or PESEL1_H	Read/write	Byte	0Dh	00h
	Port complement selection	P10SELC or PESELC_L	Read/write	Byte	11h	00h
	Interrupt Edge Select	P10IES or PEIES_H	Read/write	Byte	19h	undefined
	Interrupt Enable	P10IE or PEIE_H	Read/write	Byte	1Bh	00h
	Interrupt Flag	P10IFG or PEIFG_H	Read/write	Byte	1Dh	00h
Port 11	Input	P11IN or PFIN_L	Read only	Byte	00h	
	Output	P11OUT or PFOUT_L	Read/write	Byte	02h	undefined
	Direction	P11DIR or PFDIR_L	Read/write	Byte	04h	00h
	Resistor Enable	P11REN or PFREN_L	Read/write	Byte	06h	00h
	Port Select 0	P11SEL0 or PFSEL0_L	Read/write	Byte	0Ah	00h
	Port Select 1	P11SEL1 or PFSEL1_L	Read/write	Byte	0Ch	00h
	Port complement selection	P11SELC or PFSELC_L	Read/write	Byte	10h	00h
	Interrupt Edge Select	P11IES or PFIES_L	Read/write	Byte	18h	undefined
	Interrupt Enable	P11IE or PFIE_L	Read/write	Byte	1Ah	00h
	Interrupt Flag	P11IFG or PFIFG_L	Read/write	Byte	1Ch	00h

Table 8-3. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port A	Input	PAIN	Read only	Word	00h	
		PAIN_L	Read only	Byte	00h	
		PAIN_H	Read only	Byte	01h	
	Output	PAOUT	Read/write	Word	02h	undefined
		PAOUT_L	Read/write	Byte	02h	undefined
		PAOUT_H	Read/write	Byte	03h	undefined
	Direction	PADIR	Read/write	Word	04h	0000h
		PADIR_L	Read/write	Byte	04h	00h
		PADIR_H	Read/write	Byte	05h	00h
	Resistor Enable	PAREN	Read/write	Word	06h	0000h
		PAREN_L	Read/write	Byte	06h	00h
		PAREN_H	Read/write	Byte	07h	00h
	Port Select 0	PASEL0	Read/write	Word	0Ah	0000h
		PASEL0_L	Read/write	Byte	0Ah	00h
		PASEL0_H	Read/write	Byte	0Bh	00h
	Port Select 1	PASEL1	Read/write	Word	0Ah	0000h
		PASEL1_L	Read/write	Byte	0Ah	00h
		PASEL1_H	Read/write	Byte	0Bh	00h
	Port Complement Select	PASELC	Read/write	Word	10h	0000h
		PASELC_L	Read/write	Byte	10h	00h
		PASELC_H	Read/write	Byte	11h	00h
	Interrupt Edge Select	PAIES	Read/write	Word	18h	undefined
		PAIES_L	Read/write	Byte	18h	undefined
		PAIES_H	Read/write	Byte	19h	undefined
	Interrupt Enable	PAIE	Read/write	Word	1Ah	0000h
		PAIE_L	Read/write	Byte	1Ah	00h
		PAIE_H	Read/write	Byte	1Bh	00h
	Interrupt Flag	PAIFG	Read/write	Word	1Ch	0000h
		PAIFG_L	Read/write	Byte	1Ch	00h
		PAIFG_H	Read/write	Byte	1Dh	00h

Table 8-3. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port B	Input	PBIN	Read only	Word	00h	
		PBIN_L	Read only	Byte	00h	
		PBIN_H	Read only	Byte	01h	
	Output	PBOUT	Read/write	Word	02h	undefined
		PBOUT_L	Read/write	Byte	02h	undefined
		PBOUT_H	Read/write	Byte	03h	undefined
	Direction	PBDIR	Read/write	Word	04h	0000h
		PBDIR_L	Read/write	Byte	04h	00h
		PBDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	PBREN	Read/write	Word	06h	0000h
		PBREN_L	Read/write	Byte	06h	00h
		PBREN_H	Read/write	Byte	07h	00h
	Port Select 0	PBSEL0	Read/write	Word	0Ah	0000h
		PBSEL0_L	Read/write	Byte	0Ah	00h
		PBSEL0_H	Read/write	Byte	0Bh	00h
	Port Select 1	PBSEL1	Read/write	Word	0Ah	0000h
		PBSEL1_L	Read/write	Byte	0Ah	00h
		PBSEL1_H	Read/write	Byte	0Bh	00h
	Port Complement Select	PBSELC	Read/write	Word	10h	0000h
		PBSELC_L	Read/write	Byte	10h	00h
		PBSELC_H	Read/write	Byte	11h	00h
	Interrupt Edge Select	PBIES	Read/write	Word	18h	undefined
		PBIES_L	Read/write	Byte	18h	undefined
		PBIES_H	Read/write	Byte	19h	undefined
	Interrupt Enable	PBIE	Read/write	Word	1Ah	0000h
		PBIE_L	Read/write	Byte	1Ah	00h
		PBIE_H	Read/write	Byte	1Bh	00h
	Interrupt Flag	PBIFG	Read/write	Word	1Ch	0000h
		PBIFG_L	Read/write	Byte	1Ch	00h
		PBIFG_H	Read/write	Byte	1Dh	00h

Table 8-3. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port C	Input	PCIN	Read only	Word	00h	
		PCIN_L	Read only	Byte	00h	
		PCIN_H	Read only	Byte	01h	
	Output	PCOUT	Read/write	Word	02h	undefined
		PCOUT_L	Read/write	Byte	02h	undefined
		PCOUT_H	Read/write	Byte	03h	undefined
	Direction	PCDIR	Read/write	Word	04h	0000h
		PCDIR_L	Read/write	Byte	04h	00h
		PCDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	PCREN	Read/write	Word	06h	0000h
		PCREN_L	Read/write	Byte	06h	00h
		PCREN_H	Read/write	Byte	07h	00h
	Port Select 0	PCSEL0	Read/write	Word	0Ah	0000h
		PCSEL0_L	Read/write	Byte	0Ah	00h
		PCSEL0_H	Read/write	Byte	0Bh	00h
	Port Select 1	PCSEL1	Read/write	Word	0Ah	0000h
		PCSEL1_L	Read/write	Byte	0Ah	00h
		PCSEL1_H	Read/write	Byte	0Bh	00h
	Port Complement Select	PCSELC	Read/write	Word	10h	0000h
		PCSELC_L	Read/write	Byte	10h	00h
		PCSELC_H	Read/write	Byte	11h	00h
	Interrupt Edge Select	PCIES	Read/write	Word	18h	undefined
		PCIES_L	Read/write	Byte	18h	undefined
		PCIES_H	Read/write	Byte	19h	undefined
	Interrupt Enable	PCIE	Read/write	Word	1Ah	0000h
		PCIE_L	Read/write	Byte	1Ah	00h
		PCIE_H	Read/write	Byte	1Bh	00h
	Interrupt Flag	PCIFG	Read/write	Word	1Ch	0000h
		PCIFG_L	Read/write	Byte	1Ch	00h
		PCIFG_H	Read/write	Byte	1Dh	00h

Table 8-3. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port D	Input	PDIN	Read only	Word	00h	
		PDIN_L	Read only	Byte	00h	
		PDIN_H	Read only	Byte	01h	
	Output	PDOUT	Read/write	Word	02h	undefined
		PDOUT_L	Read/write	Byte	02h	undefined
		PDOUT_H	Read/write	Byte	03h	undefined
	Direction	PDDIR	Read/write	Word	04h	0000h
		PDDIR_L	Read/write	Byte	04h	00h
		PDDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	PDREN	Read/write	Word	06h	0000h
		PDREN_L	Read/write	Byte	06h	00h
		PDREN_H	Read/write	Byte	07h	00h
	Port Select 0	PDSEL0	Read/write	Word	0Ah	0000h
		PDSEL0_L	Read/write	Byte	0Ah	00h
		PDSEL0_H	Read/write	Byte	0Bh	00h
	Port Select 1	PDSEL1	Read/write	Word	0Ah	0000h
		PDSEL1_L	Read/write	Byte	0Ah	00h
		PDSEL1_H	Read/write	Byte	0Bh	00h
	Port Complement Select	PDSELC	Read/write	Word	10h	0000h
		PDSELC_L	Read/write	Byte	10h	00h
		PDSELC_H	Read/write	Byte	11h	00h
	Interrupt Edge Select	PDIES	Read/write	Word	18h	undefined
		PDIES_L	Read/write	Byte	18h	undefined
		PDIES_H	Read/write	Byte	19h	undefined
	Interrupt Enable	PDIE	Read/write	Word	1Ah	0000h
		PDIE_L	Read/write	Byte	1Ah	00h
		PDIE_H	Read/write	Byte	1Bh	00h
	Interrupt Flag	PDIFG	Read/write	Word	1Ch	0000h
		PDIFG_L	Read/write	Byte	1Ch	00h
		PDIFG_H	Read/write	Byte	1Dh	00h

Table 8-3. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port E	Input	PEIN	Read only	Word	00h	
		PEIN_L	Read only	Byte	00h	
		PEIN_H	Read only	Byte	01h	
	Output	PEOUT	Read/write	Word	02h	undefined
		PEOUT_L	Read/write	Byte	02h	undefined
		PEOUT_H	Read/write	Byte	03h	undefined
	Direction	PEDIR	Read/write	Word	04h	0000h
		PEDIR_L	Read/write	Byte	04h	00h
		PEDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	PEREN	Read/write	Word	06h	0000h
		PEREN_L	Read/write	Byte	06h	00h
		PEREN_H	Read/write	Byte	07h	00h
	Port Select 0	PESEL0	Read/write	Word	0Ah	0000h
		PESEL0_L	Read/write	Byte	0Ah	00h
		PESEL0_H	Read/write	Byte	0Bh	00h
	Port Select 1	PESEL1	Read/write	Word	0Ah	0000h
		PESEL1_L	Read/write	Byte	0Ah	00h
		PESEL1_H	Read/write	Byte	0Bh	00h
	Port Complement Select	PESELC	Read/write	Word	10h	0000h
		PESELC_L	Read/write	Byte	10h	00h
		PESELC_H	Read/write	Byte	11h	00h
	Interrupt Edge Select	PEIES	Read/write	Word	18h	undefined
		PEIES_L	Read/write	Byte	18h	undefined
		PEIES_H	Read/write	Byte	19h	undefined
	Interrupt Enable	PEIE	Read/write	Word	1Ah	0000h
		PEIE_L	Read/write	Byte	1Ah	00h
		PEIE_H	Read/write	Byte	1Bh	00h
	Interrupt Flag	PEIFG	Read/write	Word	1Ch	0000h
		PEIFG_L	Read/write	Byte	1Ch	00h
		PEIFG_H	Read/write	Byte	1Dh	00h

Table 8-3. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port F	Input	PFIN	Read only	Word	00h	
		PFIN_L	Read only	Byte	00h	
		PFIN_H	Read only	Byte	01h	
	Output	PFOUT	Read/write	Word	02h	undefined
		PFOUT_L	Read/write	Byte	02h	undefined
		PFOUT_H	Read/write	Byte	03h	undefined
	Direction	PFDIR	Read/write	Word	04h	0000h
		PFDIR_L	Read/write	Byte	04h	00h
		PFDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	PFREN	Read/write	Word	06h	0000h
		PFREN_L	Read/write	Byte	06h	00h
		PFREN_H	Read/write	Byte	07h	00h
	Port Select 0	PFSEL0	Read/write	Word	0Ah	0000h
		PFSEL0_L	Read/write	Byte	0Ah	00h
		PFSEL0_H	Read/write	Byte	0Bh	00h
	Port Select 1	PFSEL1	Read/write	Word	0Ah	0000h
		PFSEL1_L	Read/write	Byte	0Ah	00h
		PFSEL1_H	Read/write	Byte	0Bh	00h
	Port Complement Select	PFSELC	Read/write	Word	10h	0000h
		PFSELC_L	Read/write	Byte	10h	00h
		PFSELC_H	Read/write	Byte	11h	00h
	Interrupt Edge Select	PFIES	Read/write	Word	18h	undefined
		PFIES_L	Read/write	Byte	18h	undefined
		PFIES_H	Read/write	Byte	19h	undefined
	Interrupt Enable	PFIE	Read/write	Word	1Ah	0000h
		PFIE_L	Read/write	Byte	1Ah	00h
		PFIE_H	Read/write	Byte	1Bh	00h
	Interrupt Flag	PFIFG	Read/write	Word	1Ch	0000h
		PFIFG_L	Read/write	Byte	1Ch	00h
		PFIFG_H	Read/write	Byte	1Dh	00h

Table 8-3. Digital I/O Registers (continued)

Port	Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Port J	Input	PJIN	Read only	Word	00h	
		PJIN_L	Read only	Byte	00h	
		PJIN_H	Read only	Byte	01h	
	Output	PJOUT	Read/write	Word	02h	undefined
		PJOUT_L	Read/write	Byte	02h	undefined
		PJOUT_H	Read/write	Byte	03h	undefined
	Direction	PJDIR	Read/write	Word	04h	0000h
		PJDIR_L	Read/write	Byte	04h	00h
		PJDIR_H	Read/write	Byte	05h	00h
	Resistor Enable	PJREN	Read/write	Word	06h	0000h
		PJREN_L	Read/write	Byte	06h	00h
		PJREN_H	Read/write	Byte	07h	00h
	Port Select 0	PJSEL0	Read/write	Word	0Ah	0000h
		PJSEL0_L	Read/write	Byte	0Ah	00h
		PJSEL0_H	Read/write	Byte	0Bh	00h
	Port Select 1	PJSEL1	Read/write	Word	0Ah	0000h
		PJSEL1_L	Read/write	Byte	0Ah	00h
		PJSEL1_H	Read/write	Byte	0Bh	00h

Port x Interrupt Vector Register (PxIV)

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	**PxIV**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

PxIV Bits 15-0 Port 1 interrupt vector value

PxIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	Port x.0 interrupt	PxIFG.0	Highest
04h	Port x.1 interrupt	PxIFG.1	
06h	Port x.2 interrupt	PxIFG.2	
08h	Port x.3 interrupt	PxIFG.3	
0Ah	Port x.4 interrupt	PxIFG.4	
0Ch	Port x.5 interrupt	PxIFG.5	
0Eh	Port x.6 interrupt	PxIFG.6	
10h	Port x.7 interrupt	PxIFG.7	Lowest

Port x Interrupt Edge Select Register (PxIES)

7	6	5	4	3	2	1	0
PxIES							
rw	rw	rw	rw	rw	rw	rw	rw

PxIES Bits 7-0 Port x interrupt edge select

0 PxIFG flag is set with a low-to-high transition.

1 PxIFG flag is set with a high-to-low transition.

Port x Interrupt Enable Register (PxIE)

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

PxIE Bits 7-0 Port x interrupt enable

0 Corresponding port interrupt disabled

1 Corresponding port interrupt enabled

Port x Interrupt Flag Register (PxIFG)

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

PxIFG Bits 7-0 Port x interrupt flag

0 No interrupt is pending.

1 Interrupt is pending.

Port x Input Register (PxIN)

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.

Port x Output Register (PxOUT)

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 Output is low.
1 Output is high.
When I/O configured to input mode and pullups/pulldowns enabled:
0 pulldown selected
1 pullup selected

Port x Direction Register (PxDIR)

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

Port x Pullup/Pulldown Resistor Enable Registers (PxREN)

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. When respective port is configured as input, setting this bit will enable the pullup or pulldown. Please refer to Table 8-1
0 Pullup or pulldown disabled.
1 Pullup or pulldown enabled.

Port x Function Selection Registers (PxSEL1, PxSEL0)

7	6	5	4	3	2	1	0
PxSEL1, 0							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

PxSEL Bits 7-0 Port function selection
0 0 general purpose I/O is selected
0 1 primary module function is selected
1 0 secondary module function is selected
1 1 ternary module function is selected

CRC Module

The cyclic redundancy check (CRC) module provides a signature for a given data sequence. This chapter describes the operation and use of the CRC module.

Topic **Page**

9.1 Cyclic Redundancy Check (CRC) Module Introduction

The CRC module produces a signature for a given sequence of data values. The signature is generated through a feedback path from data bits 0, 4, 11, and 15 (see Figure 9-1). The CRC signature is based on the polynomial given in the CRC-CCITT-BR polynomial (see Equation 10) .

$$f(x) = x^{16} + x^{12} + x^5 + 1 \quad (10)$$

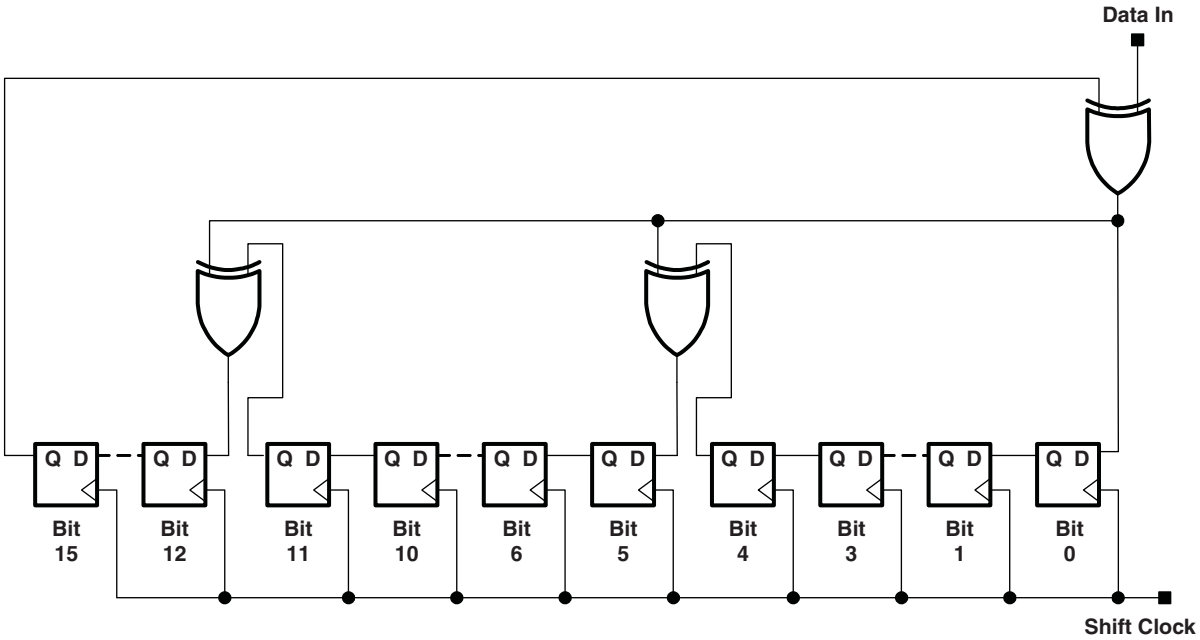


Figure 9-1. LFSR Implementation of CRC-CCITT Standard, Bit 0 is the MSB of the Result

Identical input data sequences result in identical signatures when the CRC is initialized with a fixed seed value, whereas different sequences of input data, in general, result in different signatures.

9.2 CRC Checksum Generation

The CRC generator is first initialized by writing a 16-bit word (seed) to the CRC Initialization and Result (CRCINIRES) register. Any data that should be included into the CRC calculation must be written to the CRC Data Input (CRCDI or CRCDIRB) register in the same order that the original CRC signature was calculated. The actual signature can be read from the CRCINIRES register to compare the computed checksum with the expected checksum.

Signature generation describes a method of how the result of a signature operation can be calculated. The calculated signature, which is computed by an external tool, is called checksum in the following text. The checksum is stored in the product's memory and is used to check the correctness of the CRC operation result.

9.2.1 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 CRCINIRES register to initialize the register. Software or hardware (for example, the DMA) can transfer data to the CRCDI or CRCDIRB register (for example, from memory). The value in CRCDI or CRCDIRB is then included into the signature, and the result is available in the signature result registers at the next read access (CRCINIRES and CRCRESR). The signature can be generated using word or byte data.

If a word data 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.

Data bytes written to CRCDIRB in word mode or the data byte in byte mode are bit-wise reversed before the CRC engine adds them to the signature. The bits among each byte are reversed. Data bytes written to CRCDI in word mode or the data byte in byte mode are not bit reversed before use by the CRC engine.

If the Check Sum itself (with reversed bit order) is included into the CRC operation (as data written to CRCDI or CRCDIRB), the result in the CRCINIRES and CRCRESR registers must be zero.

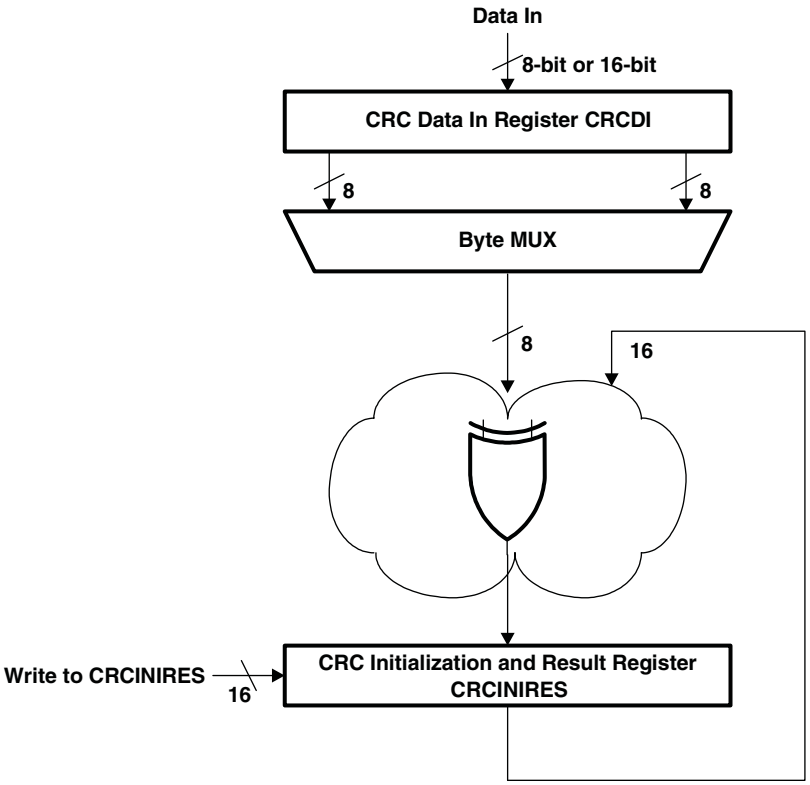


Figure 9-2. Implementation of CRC-CCITT using the CRCDI and CRCINIRES registers

9.2.2 Assembler Examples

Example 9-1 demonstrates the operation of the on-chip CRC.

Example 9-1. General Assembler Example

```
   ...
   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
```

The details of the implemented CRC algorithm are shown by the data sequences in Example 9-2 using word or byte accesses and the CRC data-in as well as the CRC data-in reverse byte registers.

Example 9-2. Reference Data Sequence

```
   ...
mov     #0FFFFh,&CRCINIRES  ; initialize CRC
mov.b   #00031h,&CRCDI_L    ; "1"
mov.b   #00032h,&CRCDI_L    ; "2"
mov.b   #00033h,&CRCDI_L    ; "3"
mov.b   #00034h,&CRCDI_L    ; "4"
mov.b   #00035h,&CRCDI_L    ; "5"
mov.b   #00036h,&CRCDI_L    ; "6"
mov.b   #00037h,&CRCDI_L    ; "7"
mov.b   #00038h,&CRCDI_L    ; "8"
mov.b   #00039h,&CRCDI_L    ; "9"

cmp     #089F6h,&CRCINIRES  ; compare result
                            ; CRCRESR contains 06F91h
jeq     &Success            ; no error
br      &Error              ; to error handler
mov     #0FFFFh,&CRCINIRES  ; initialize CRC
mov.w   #03231h,&CRCDI      ; "1" & "2"
mov.w   #03433h,&CRCDI      ; "3" & "4"
mov.w   #03635h,&CRCDI      ; "5" & "6"
mov.w   #03837h,&CRCDI      ; "7" & "8"
mov.b   #039h,  &CRCDI_L    ; "9"

cmp     #089F6h,&CRCINIRES  ; compare result
                               ; CRCRESR contains 06F91h
jeq     &Success            ; no error
br      &Error              ; to error handler
   ...
mov     #0FFFFh,&CRCINIRES  ; initialize CRC
mov.b   #00031h,&CRCDIRB_L  ; "1"
mov.b   #00032h,&CRCDIRB_L  ; "2"
mov.b   #00033h,&CRCDIRB_L  ; "3"
mov.b   #00034h,&CRCDIRB_L  ; "4"
mov.b   #00035h,&CRCDIRB_L  ; "5"
mov.b   #00036h,&CRCDIRB_L  ; "6"
mov.b   #00037h,&CRCDIRB_L  ; "7"
mov.b   #00038h,&CRCDIRB_L  ; "8"
mov.b   #00039h,&CRCDIRB_L  ; "9"

cmp     #029B1h,&CRCINIRES  ; compare result
                            ; CRCRESR contains 08D94h
jeq     &Success            ; no error
br      &Error              ; to error handler
...
mov     #0FFFFh,&CRCINIRES  ; initialize CRC
mov.w   #03231h,&CRCDIRB    ; "1" & "2"
mov.w   #03433h,&CRCDIRB    ; "3" & "4"
mov.w   #03635h,&CRCDIRB    ; "5" & "6"
mov.w   #03837h,&CRCDIRB    ; "7" & "8"
mov.b   #039h,  &CRCDIRB_L  ; "9"

cmp     #029B1h,&CRCINIRES  ; compare result
                            ; CRCRESR contains 08D94h
jeq     &Success            ; no error
br      &Error              ; to error handler
```

9.3 CRC Module Registers

The CRC module registers are listed in Table 9-1. The base address can be found in the device-specific data sheet. The address offset is given in Table 9-1.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 9-1. CRC Module Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
CRC Data In	CRCDI	Read/write	Word	0000h	0000h
	CRCDI_L	Read/write	Byte	0000h	00h
	CRCDI_H	Read/write	Byte	0001h	00h
CRC Data In Reverse Byte	CRCDIRB	Read/write	Word	0002h	0000h
	CRCDIRB_L	Read/write	Byte	0002h	00h
	CRCDIRB_H	Read/write	Byte	0003h	00h
CRC Initialization and Result	CRCINIRES	Read/write	Word	0004h	FFFFh
	CRCINIRES_L	Read/write	Byte	0004h	FFh
	CRCINIRES_H	Read/write	Byte	0005h	FFh
CRC Result Reverse	CRCRESR	Read only	Word	0006h	FFFFh
	CRCRESR_L	Read/write	Byte	0006h	FFh
	CRCRESR_H	Read/write	Byte	0007h	FFh

CRC Data In Register (CRCDI)

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 is included to the present signature in the CRCINIRES register according to the CRC-CCITT standard.

CRC Data In Reverse Register (CRCDIRB)

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

CRCDIRB	Bits 15-0	CRC data in reverse byte. Data written to the CRCDIRB register is included to the present signature in the CRCINIRES and CRCRESR registers according to the CRC-CCITT standard. Reading the register returns the register CRCDI content.

CRC Initialization and Result Register (CRCINIRES)

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.

CRC Reverse Result Register (CRCRESR)

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

CRCRESR	Bits 15-0	CRC reverse result. This register holds the current CRC result (according to the CRC-CCITT standard). The order of bits is reverse (for example, CRCINIRES[15] = CRCRESR[0]) to the order of bits in the CRCINIRES register (see example code).

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 devices.

Topic **Page**

10.1 WDT_A 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
- Password-protected access to Watchdog Timer Control (WDTCTL) register
- Selectable clock source
- Can be stopped to conserve power
- Clock fail-safe feature

The watchdog timer block diagram is shown in Figure 10-1.

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 set up or halt the WDT_A prior to the expiration of the initial reset interval.

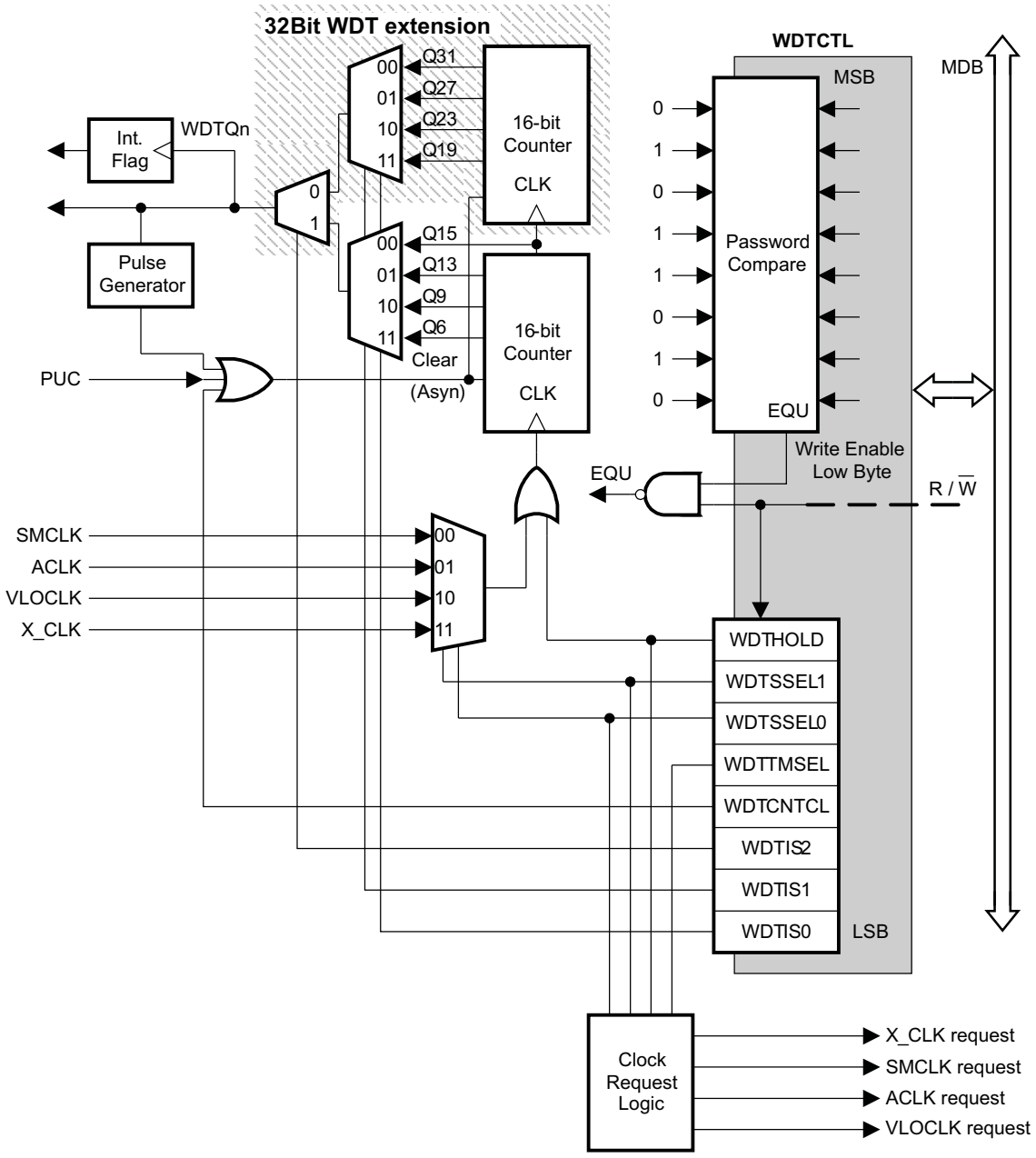


Figure 10-1. Watchdog Timer Block Diagram

10.2 WDT_A Operation

The watchdog timer 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 password 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 result in the value of the low byte. Writing byte wide to upper or lower parts of WDTCTL results in a PUC.

10.2.1 Watchdog Timer Counter (WDTCNT)

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

10.2.2 Watchdog Mode

After a PUC condition, the WDT module is configured in the watchdog mode with an initial ~32-ms reset interval using the SMCLK. The user must setup, halt, or clear the watchdog timer prior to the expiration of the initial reset interval or another PUC is generated. When the watchdog timer 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 watchdog timer to its default condition.

10.2.3 Interval Timer Mode

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

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

NOTE: Modifying the watchdog timer

The watchdog timer interval should be changed together with WDTCNTCL = 1 in a single instruction to avoid an unexpected immediate PUC or interrupt. The watchdog timer should be halted before changing the clock source to avoid a possible incorrect interval.

10.2.4 Watchdog Timer Interrupts

The watchdog timer uses two bits in the SFRs for interrupt control:

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

When using the watchdog timer 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, the watchdog timer initiated the reset condition, either by timing out or by a password violation. If WDTIFG is cleared, the reset was caused by a different source.

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

10.2.5 Clock Fail-Safe Feature

The WDT_A provides a fail-safe clocking feature, ensuring 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 the WDT_A clock source, VLOCLK is automatically selected as the 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.

10.2.6 Operation in Low-Power Modes

The devices have several low-power modes. Different clock signals are available in different low-power modes. The requirements of the application and the type of clocking that is 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.

Any write operation to WDTCTL must be a word operation with 05Ah (WDTPW) in the upper byte (see Example 10-1).

Example 10-1. Writes to WDTCTL

```
; Periodically clear an active watchdog
MOV #WDTPW+WDTIS2+WDTIS1+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
```

10.3 WDT_A Registers

The watchdog timer module registers are listed in Table 10-1. The base address for the watchdog timer module registers and special function registers (SFRs) can be found in the device-specific data sheets. The address offset is given in Table 10-1.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 10-1. Watchdog Timer Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Watchdog Timer Control	WDTCTL	Read/write	Word	0Ch	6904h
	WDTCTL_L	Read/write	Byte	0Ch	04h
	WDTCTL_H	Read/write	Byte	0Dh	69h

Watchdog Timer Control Register (WDTCTL)

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	**WDTSSEL**		**WDTTMSEL**	**WDTCNTCL**	**WDTIS**		
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 is 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 Bits 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

WDTCNTCL Bit 3 Watchdog timer counter clear. Setting WDTCNTCL = 1 clears the count value to 0000h. WDTCNTCL is automatically reset.

- 0 No action
- 1 WDTCNT = 0000h

WDTIS Bits 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)

Timer_A

Timer_A is a 16-bit timer/counter with multiple capture/compare registers. There can be multiple Timer_A modules on a given device (see the device-specific data sheet). This chapter describes the operation and use of the Timer_A module.

Topic **Page**

11.1 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 pulse width modulation (PWM) capability
- Asynchronous input and output latching
- Interrupt vector register for fast decoding of all Timer_A interrupts

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

NOTE: Use of the word *count*

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, an associated action does not take place.

NOTE: Nomenclature

There may be multiple instantiations of Timer_A on a given device. The prefix TAx is used, where x is a greater than equal to zero indicating the Timer_A instantiation. For devices with one instantiation, x = 0. The suffix n, where n = 0 to 6, represents the specific capture/compare registers associated with the Timer_A instantiation.

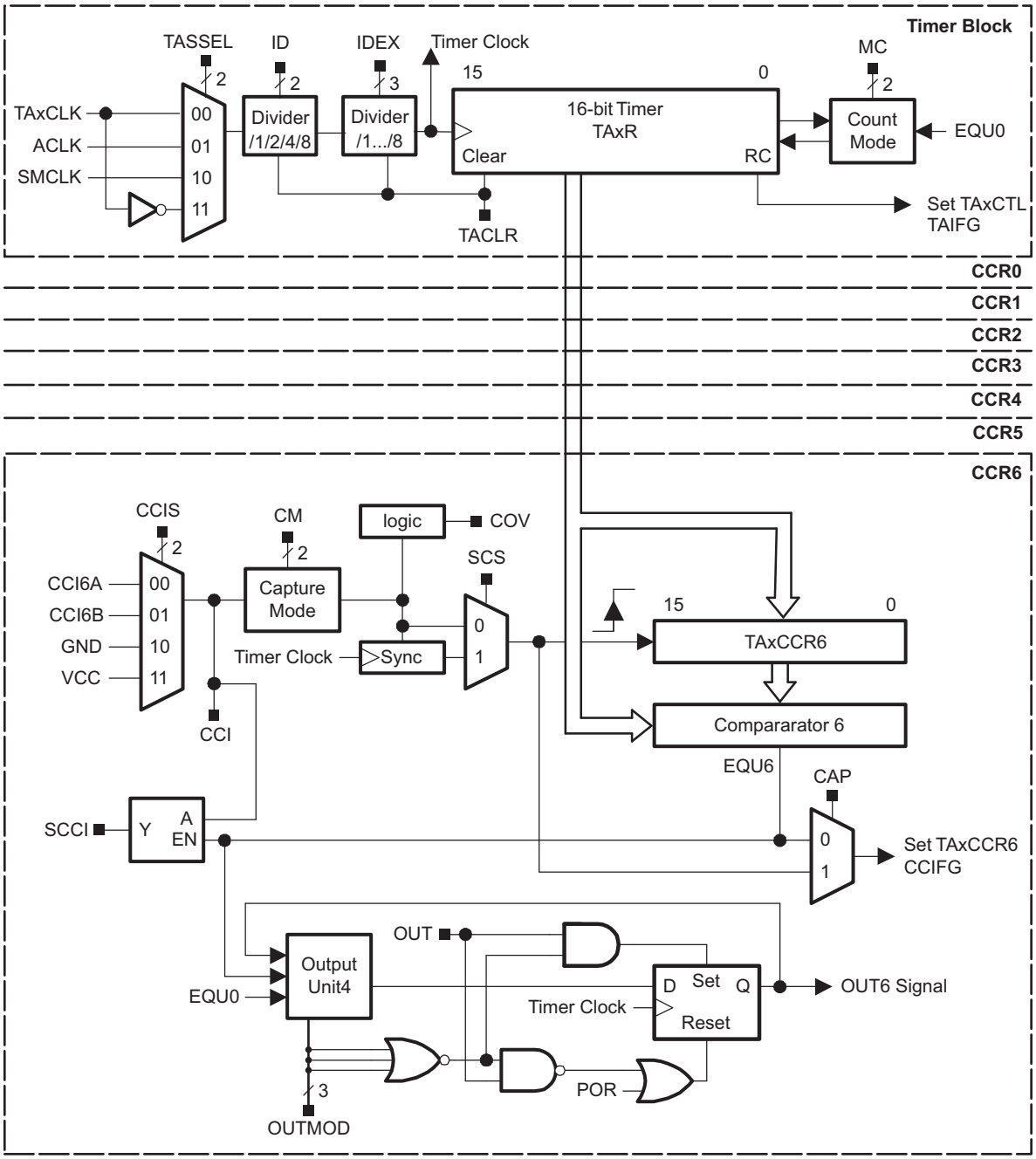


Figure 11-1. Timer_A Block Diagram

11.2 Timer_A Operation

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

11.2.1 16-Bit Timer Counter

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

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

NOTE: Modifying Timer_A registers

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

When the timer clock is asynchronous to the CPU clock, any read from TAxR 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 TAxR takes effect immediately.

11.2.1.1 Clock Source Select and Divider

The timer clock can be sourced from ACLK, SMCLK, or externally via TAxCLK. The clock source is selected with the TASSEL bits. The selected clock source may be passed directly to the timer or divided by 2, 4, or 8, using the ID bits. The selected clock source can be further divided by 2, 3, 4, 5, 6, 7, or 8 using the TAIDEX bits. The timer clock dividers are reset when TACLR is set.

NOTE: Timer_A dividers

Setting the TACLR bit clears the contents of TAxR and the clock dividers. The clock dividers are implemented as down counters. Therefore, when the TACLR bit is cleared, the timer clock immediately begins clocking at the first rising edge of the Timer_A clock source selected with the TASSEL bits and continues clocking at the divider settings set by the ID and TAIDEX bits.

11.2.2 Starting the Timer

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

- The timer counts when MC > { 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 TAxCCR0. The timer may then be restarted by writing a nonzero value to TAxCCR0. In this scenario, the timer starts incrementing in the up direction from zero.

11.2.3 Timer Mode Control

The timer has four modes of operation: stop, up, continuous, and up/down (see Table 11-1). The operating mode is selected with the MC bits.

Table 11-1. Timer Modes

MCx	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh.
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero.

11.2.3.1 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 TAxCCR0, which defines the period (see Figure 11-2). The number of timer counts in the period is TAxCCR0 + 1. When the timer value equals TAxCCR0, the timer restarts counting from zero. If up mode is selected when the timer value is greater than TAxCCR0, the timer immediately restarts counting from zero.

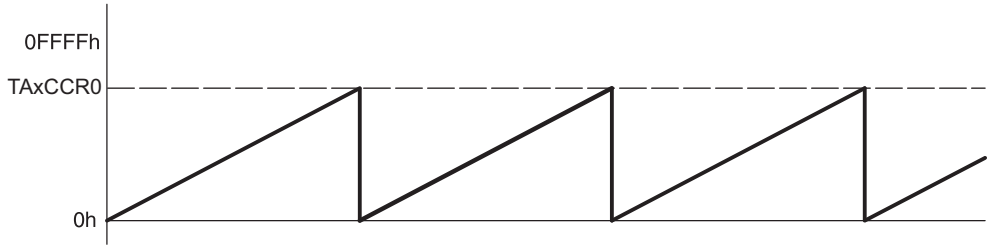


Figure 11-2. Up Mode

The TAxCCR0 CCIFG interrupt flag is set when the timer *counts* to the TAxCCR0 value. The TAIFG interrupt flag is set when the timer *counts* from TAxCCR0 to zero. Figure 11-3 shows the flag set cycle.

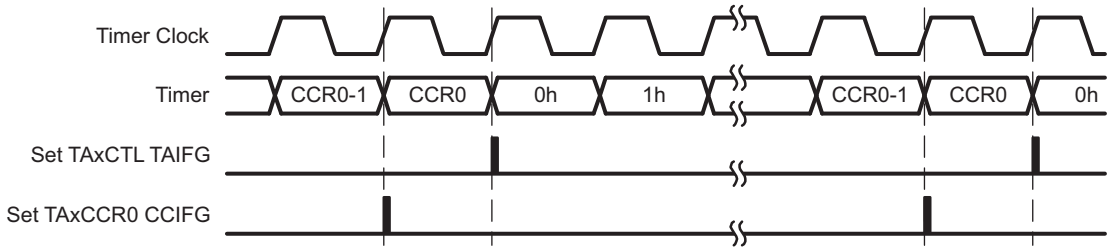


Figure 11-3. Up Mode Flag Setting

11.2.3.1.1 Changing Period Register TAxCCR0

When changing TAxCCR0 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.

11.2.3.2 Continuous Mode

In the continuous mode, the timer repeatedly counts up to 0FFFFh and restarts from zero as shown in Figure 11-4. The capture/compare register TAxCCR0 works the same way as the other capture/compare registers.

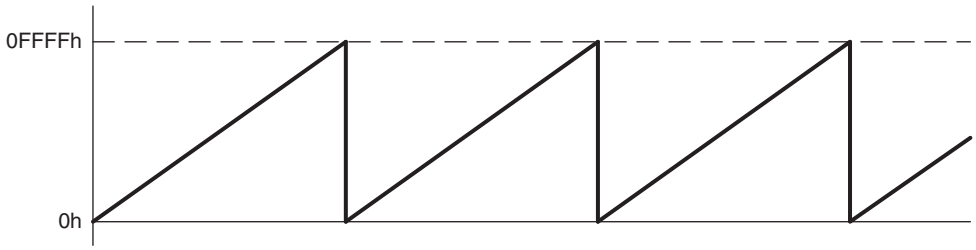


Figure 11-4. Continuous Mode

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

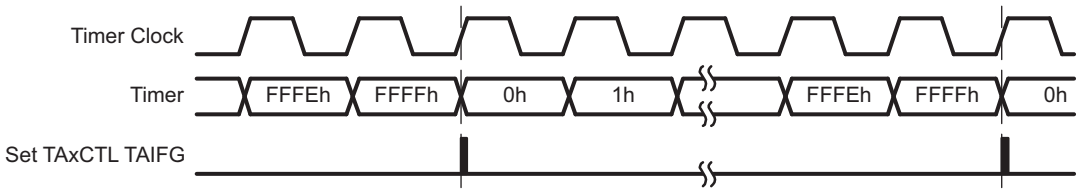


Figure 11-5. Continuous Mode Flag Setting

11.2.3.3 Use of 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 TAxCCRn register in the interrupt service routine. Figure 11-6 shows two separate time intervals, t_0 and t_1, being added to the capture/compare registers. In this usage, the time interval is controlled by hardware, not software, without impact from interrupt latency. Up to n (where n = 0 to 6), independent time intervals or output frequencies can be generated using capture/compare registers.

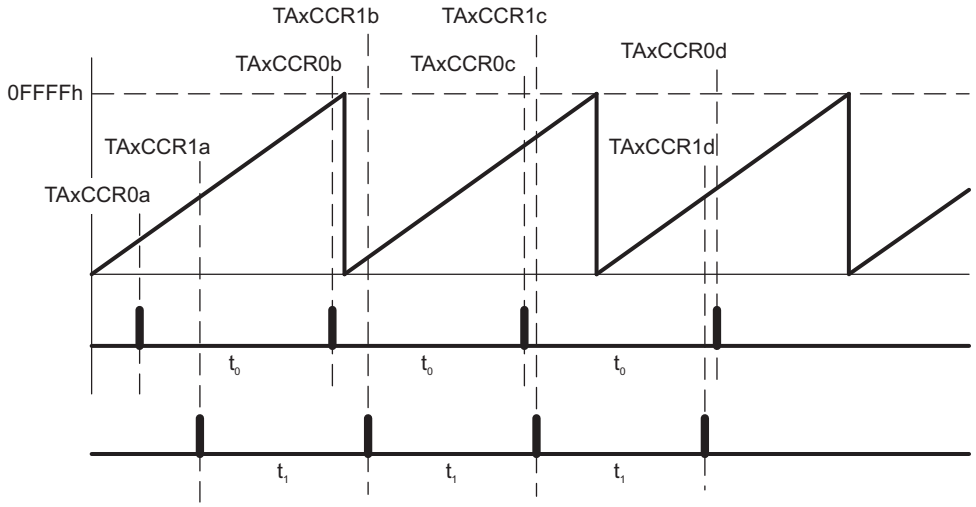


Figure 11-6. Continuous Mode Time Intervals

Time intervals can be produced with other modes as well, where TAxCCR0 is used as the period register. Their handling is more complex since the sum of the old TAxCCRn data and the new period can be higher than the TAxCCR0 value. When the previous TAxCCRn value plus t_x is greater than the TAxCCR0 data, the TAxCCR0 value must be subtracted to obtain the correct time interval.

11.2.3.4 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 TAxCCR0 and back down to zero (see Figure 11-7). The period is twice the value in TAxCCR0.

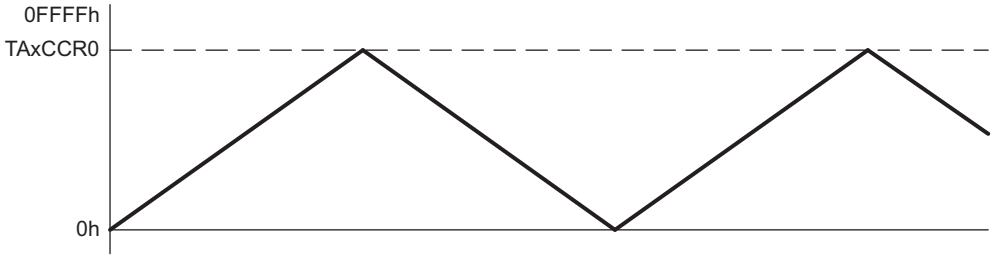


Figure 11-7. Up/Down Mode

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

In up/down mode, the TAxCCR0 CCIFG interrupt flag and the TAIFG interrupt flag are set only once during a period, separated by one-half the timer period. The TAxCCR0 CCIFG interrupt flag is set when the timer *counts* from TAxCCR0-1 to TAxCCR0, and TAIFG is set when the timer completes *counting* down from 0001h to 0000h. Figure 11-8 shows the flag set cycle.

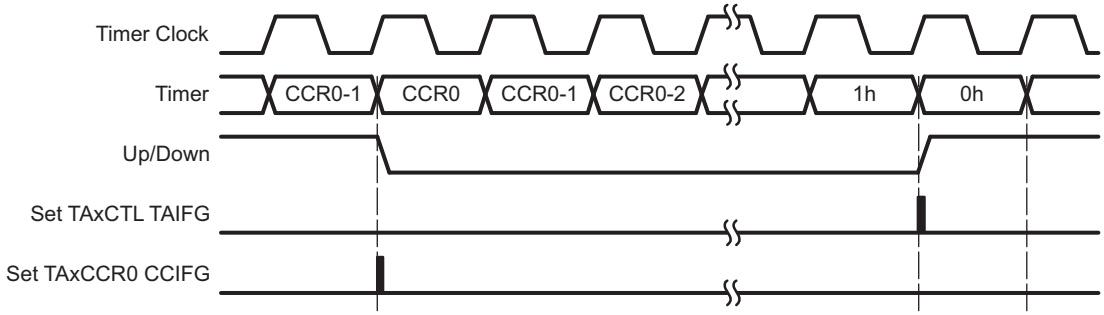


Figure 11-8. Up/Down Mode Flag Setting

11.2.3.4.1 Changing Period Register TAxCCR0

When changing TAxCCR0 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.

11.2.3.5 Use of Up/Down Mode

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

$t_{dead} = t_{timer} \times (TAxCCR1 – TAxCCR2)$

Where:

t_{dead} = Time during which both outputs need to be inactive
t_{timer} = Cycle time of the timer clock
TAxCCRn = Content of capture/compare register n

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

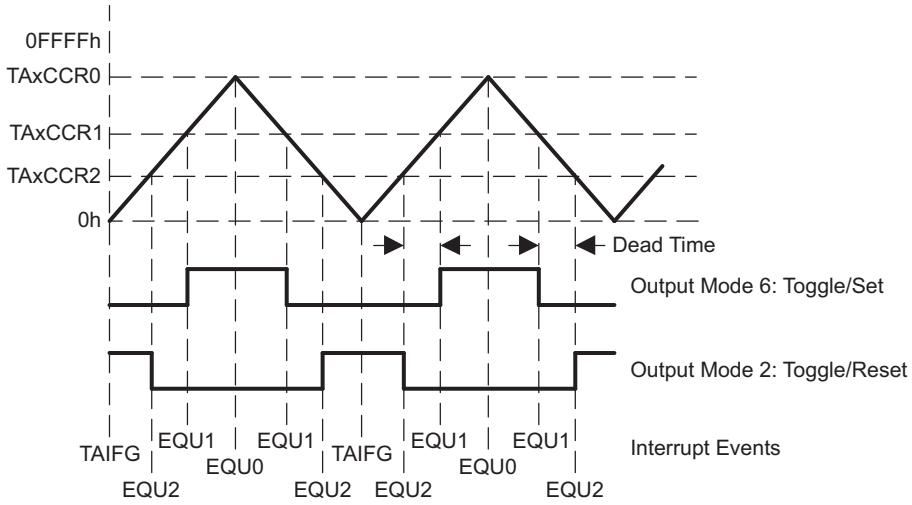


Figure 11-9. Output Unit in Up/Down Mode

11.2.4 Capture/Compare Blocks

Up to seven identical capture/compare blocks, TAxCCRn (where n = 0 to 7), are present in Timer_A. Any of the blocks may be used to capture the timer data or to generate time intervals.

11.2.4.1 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 CCIS bits. The CM 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 TAxCCRn register.
- The interrupt flag CCIFG is set.

The input signal level can be read at any time via the CCI bit. Devices may have different signals connected to CCIxA and CCIxB. See 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 synchronizes the capture with the next timer clock. Setting the SCS bit to synchronize the capture signal with the timer clock is recommended (see Figure 11-10).

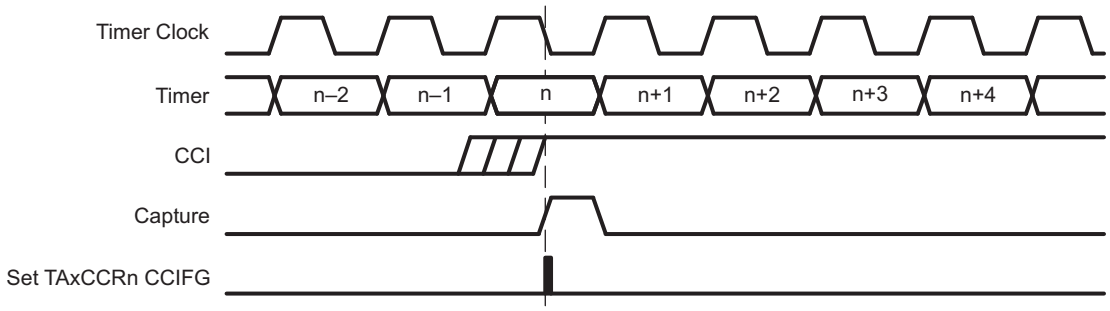


Figure 11-10. Capture Signal (SCS = 1)

NOTE: Changing Capture Inputs

Changing capture inputs while in capture mode may cause unintended capture events. To avoid this scenario, capture inputs should only be changed when capture mode is disabled (CM = {0} or CAP = 0).

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

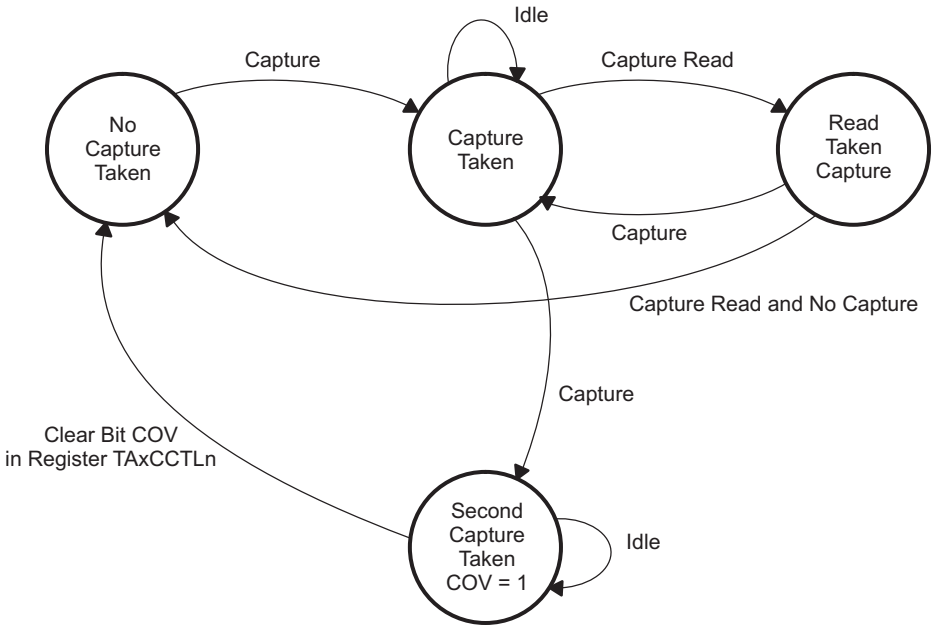


Figure 11-11. Capture Cycle

11.2.4.1.1 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,&TA0CCTL1  ; Setup TA0CCTL1, synch. capture mode
                                    ; Event trigger on both edges of capture input.
XOR  #CCIS0,&TA0CCTL1               ; TA0CCR1 = TA0R
```

NOTE: Capture Initiated by Software

In general, changing capture inputs while in capture mode may cause unintended capture events. For this scenario, switching the capture input between VCC and GND, disabling the capture mode is not required.

11.2.4.2 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 TAxR *counts* to the value in a TAxCCRn, where n represents the specific capture/compare register.

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

11.2.5 Output Unit

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

11.2.5.1 Output Modes

The output modes are defined by the OUTMOD bits and are described in Table 11-2. The OUTn 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 EQUn = EQU0.

Table 11-2. Output Modes

OUTMODx	Mode	Description
000	Output	The output signal OUTn is defined by the OUT bit. The OUTn signal updates immediately when OUT is updated.
001	Set	The output is set when the timer *counts* to the TAxCCRn 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 TAxCCRn value. It is reset when the timer *counts* to the TAxCCR0 value.
011	Set/Reset	The output is set when the timer *counts* to the TAxCCRn value. It is reset when the timer *counts* to the TAxCCR0 value.
100	Toggle	The output is toggled when the timer *counts* to the TAxCCRn value. The output period is double the timer period.
101	Reset	The output is reset when the timer *counts* to the TAxCCRn 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 TAxCCRn value. It is set when the timer *counts* to the TAxCCR0 value.
111	Reset/Set	The output is reset when the timer *counts* to the TAxCCRn value. It is set when the timer *counts* to the TAxCCR0 value.

11.2.5.1.1 Output Example—Timer in Up Mode

The OUTn signal is changed when the timer *counts* up to the TAxCCRn value and rolls from TAxCCR0 to zero, depending on the output mode. An example is shown in Figure 11-12 using TAxCCR0 and TAxCCR1.

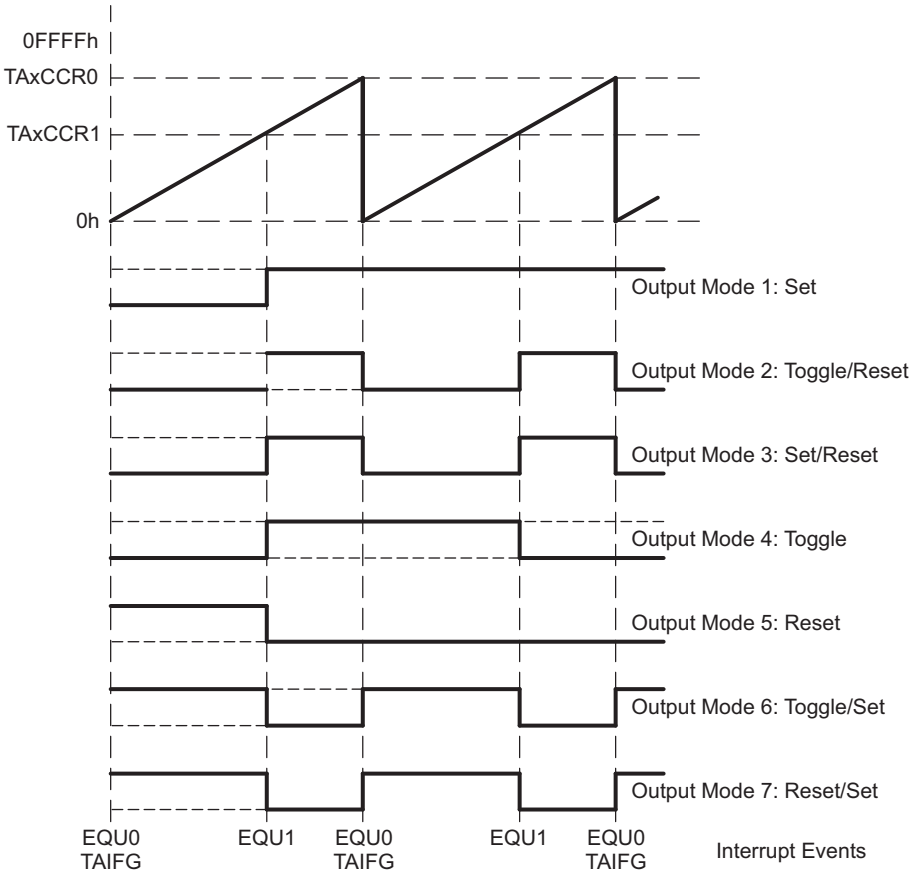


Figure 11-12. Output Example – Timer in Up Mode

11.2.5.1.2 Output Example – Timer in Continuous Mode

The OUTn signal is changed when the timer reaches the TAxCCRn and TAxCCR0 values, depending on the output mode. An example is shown in Figure 11-13 using TAxCCR0 and TAxCCR1.

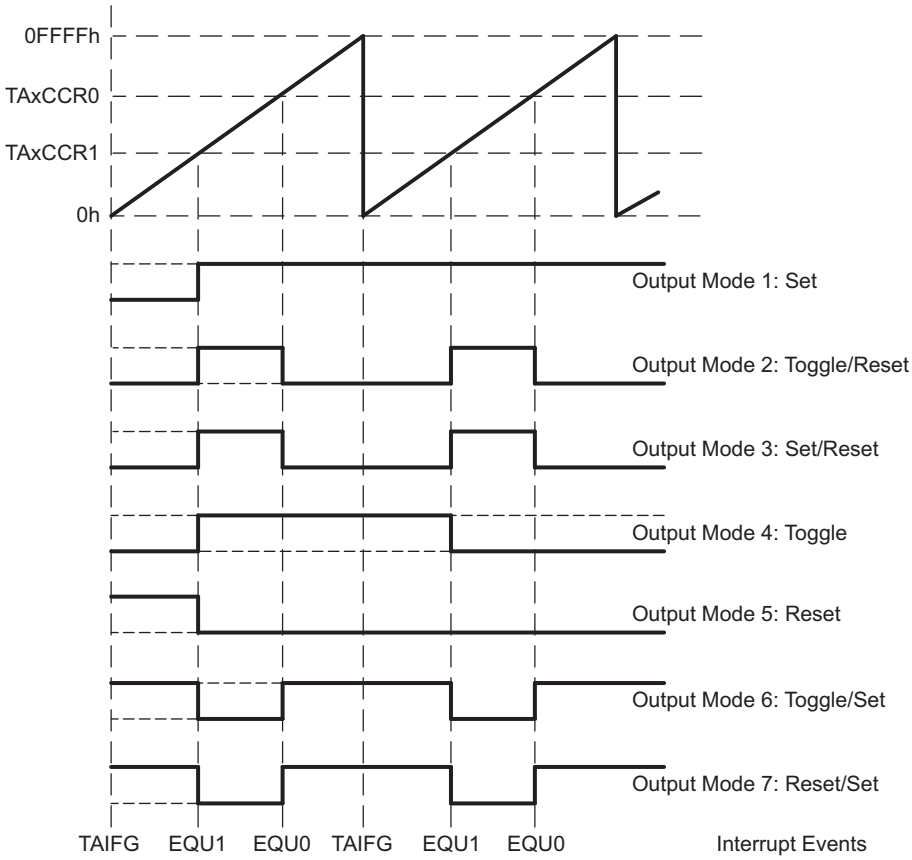


Figure 11-13. Output Example – Timer in Continuous Mode

11.2.5.1.3 Output Example – Timer in Up/Down Mode

The OUTn signal changes when the timer equals TAxCCRn in either count direction and when the timer equals TAxCCR0, depending on the output mode. An example is shown in Figure 11-14 using TAxCCR0 and TAxCCR2.

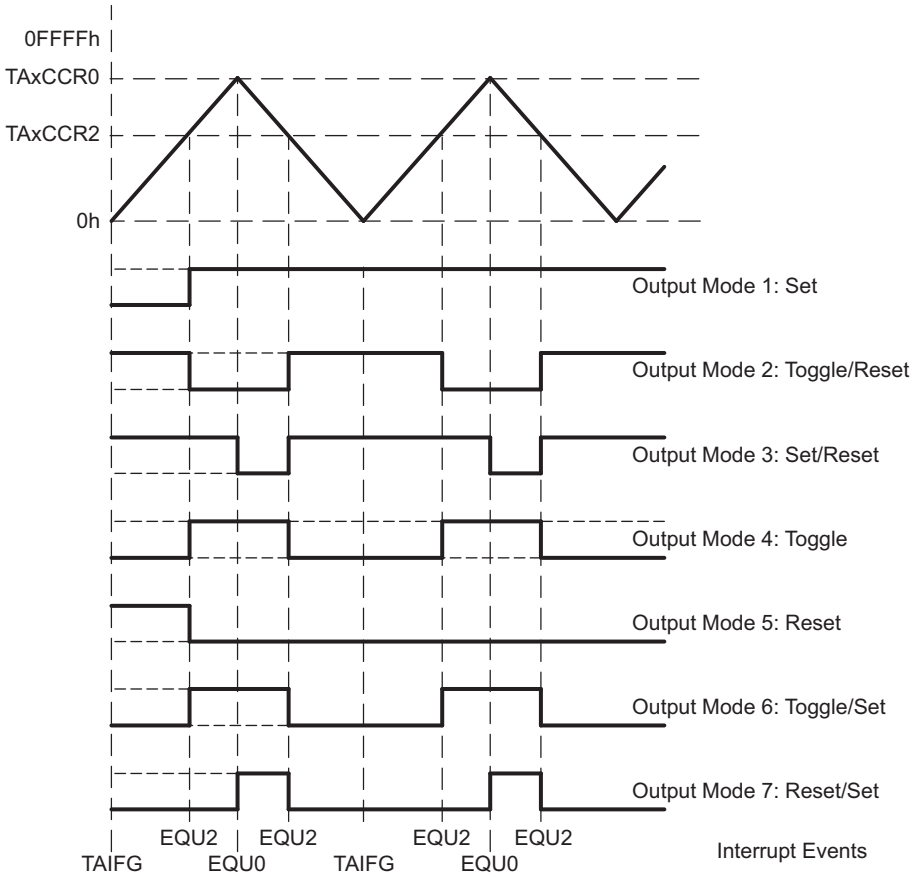


Figure 11-14. Output Example – Timer in Up/Down Mode

NOTE: Switching between output modes

When switching between output modes, one of the OUTMOD 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,&TA0CCTL1      ; Set output mode=7
BIC   #OUTMOD,&TA0CCTL1       ; Clear unwanted bits
```

11.2.6 Timer_A Interrupts

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

- TAxCCR0 interrupt vector for TAxCCR0 CCIFG
- TAxIV 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 TAxCCRn register. In compare mode, any CCIFG flag is set if TAxR *counts* to the associated TAxCCRn 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.

11.2.6.1 TAxCCR0 Interrupt

The TAxCCR0 CCIFG flag has the highest Timer_A interrupt priority and has a dedicated interrupt vector as shown in Figure 11-15. The TAxCCR0 CCIFG flag is automatically reset when the TAxCCR0 interrupt request is serviced.

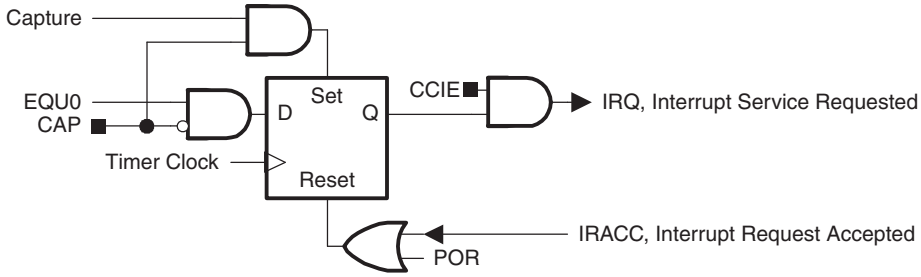


Figure 11-15. Capture/Compare TAxCCR0 Interrupt Flag

11.2.6.2 TAxIV, Interrupt Vector Generator

The TAxCCRy CCIFG flags and TAIFG flags are prioritized and combined to source a single interrupt vector. The interrupt vector register TAxIV is used to determine which flag requested an interrupt.

The highest-priority enabled interrupt generates a number in the TAxIV 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 TAxIV value.

Any access, read or write, of the TAxIV 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 TAxCCR1 and TAxCCR2 CCIFG flags are set when the interrupt service routine accesses the TAxIV register, TAxCCR1 CCIFG is reset automatically. After the RETI instruction of the interrupt service routine is executed, the TAxCCR2 CCIFG flag generates another interrupt.

11.2.6.2.1 TAxIV Software Example

The following software example shows the recommended use of TAxIV and the handling overhead. The TAxIV value is added to the PC to automatically jump to the appropriate routine. The example assumes a single instantiation of the largest timer configuration available.

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 TA0CCR0: 11 cycles
- Capture/compare blocks TA0CCR1, TA0CCR2, TA0CCR3, TA0CCR4, TA0CCR5, TA0CCR6: 16 cycles
- Timer overflow TA0IFG: 14 cycles

```
; Interrupt handler for TA0CCR0 CCIFG.                    Cycles
CCIFG_0_HND
;        ...          ; Start of handler Interrupt latency    6
         RETI                                                 5

; Interrupt handler for TA0IFG, TA0CCR1 through TA0CCR6 CCIFG.

TA0_HND      ...                ; Interrupt latency           6
         ADD      &TA0IV,PC     ; Add offset to Jump table    3
         RETI                   ; Vector  0: No interrupt     5
         JMP      CCIFG_1_HND   ; Vector  2: TA0CCR1          2
         JMP      CCIFG_2_HND   ; Vector  4: TA0CCR2          2
         JMP      CCIFG_3_HND   ; Vector  6: TA0CCR3          2
         JMP      CCIFG_4_HND   ; Vector  8: TA0CCR4          2
         JMP      CCIFG_5_HND   ; Vector 10: TA0CCR5          2
         JMP      CCIFG_6_HND   ; Vector 12: TA0CCR6          2

TA0IFG_HND                     ; Vector 14: TA0IFG Flag
         ...                   ; Task starts here
         RETI                                                 5

CCIFG_6_HND                    ; Vector 12: TA0CCR6
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_5_HND                    ; Vector 10: TA0CCR5
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_4_HND                    ; Vector 8: TA0CCR4
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_3_HND                    ; Vector 6: TA0CCR3
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_2_HND                    ; Vector 4: TA0CCR2
         ...                   ; Task starts here
         RETI                  ; Back to main program        5

CCIFG_1_HND                    ; Vector 2: TA0CCR1
         ...                   ; Task starts here
         RETI                  ; Back to main program        5
```

11.3 Timer_A Registers

Timer_A registers are listed in Table 11-3 for the largest configuration available. The base address can be found in the device-specific data sheet. The address offsets are listed in Table 11-3.

Table 11-3. Timer_A Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Timer_A Control	TAxCTL	Read/write	Word	00h	0000h
Timer_A Capture/Compare Control 0	TAxCCTL0	Read/write	Word	02h	0000h
Timer_A Capture/Compare Control 1	TAxCCTL1	Read/write	Word	04h	0000h
Timer_A Capture/Compare Control 2	TAxCCTL2	Read/write	Word	06h	0000h
Timer_A Capture/Compare Control 3	TAxCCTL3	Read/write	Word	08h	0000h
Timer_A Capture/Compare Control 4	TAxCCTL4	Read/write	Word	0Ah	0000h
Timer_A Capture/Compare Control 5	TAxCCTL5	Read/write	Word	0Ch	0000h
Timer_A Capture/Compare Control 6	TAxCCTL6	Read/write	Word	0Eh	0000h
Timer_A Counter	TAxR	Read/write	Word	10h	0000h
Timer_A Capture/Compare 0	TAxCCR0	Read/write	Word	12h	0000h
Timer_A Capture/Compare 1	TAxCCR1	Read/write	Word	14h	0000h
Timer_A Capture/Compare 2	TAxCCR2	Read/write	Word	16h	0000h
Timer_A Capture/Compare 3	TAxCCR3	Read/write	Word	18h	0000h
Timer_A Capture/Compare 4	TAxCCR4	Read/write	Word	1Ah	0000h
Timer_A Capture/Compare 5	TAxCCR5	Read/write	Word	1Ch	0000h
Timer_A Capture/Compare 6	TAxCCR6	Read/write	Word	1Eh	0000h
Timer_A Interrupt Vector	TAxIV	Read only	Word	2Eh	0000h
Timer_A Expansion 0	TAxEX0	Read/write	Word	20h	0000h

Timer_A Control Register (TAxCTL)

15	14	13	12	11	10	9	8
Unused						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID		MC		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
TASSEL	Bits 9-8	Timer_A clock source select 00 TAxCLK 01 ACLK 10 SMCLK 11 Inverted TAxCLK
ID	Bits 7-6	Input divider. These bits along with the TAIDEX bits select the divider for the input clock. 00 /1 01 /2 10 /4 11 /8
MC	Bits 5-4	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. 00 Stop mode: Timer is halted 01 Up mode: Timer counts up to TAxCCR0 10 Continuous mode: Timer counts up to 0FFFFh 11 Up/down mode: Timer counts up to TAxCCR0 then down to 0000h
Unused	Bit 3	Unused
TACLR	Bit 2	Timer_A clear. Setting this bit resets TAxR, the timer clock divider, and the count direction. The TACLR bit is automatically reset and is always read as zero.
TAIE	Bit 1	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0 Interrupt disabled 1 Interrupt enabled
TAIFG	Bit 0	Timer_A interrupt flag 0 No interrupt pending 1 Interrupt pending

Timer_A Counter Register (TAxR)

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

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

Capture/Compare Control Register (TAxCCTLn)

15	14	13	12	11	10	9	8
CM		**CCIS**		**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
OUTMOD			**CCIE**	**CCI**	**OUT**	**COV**	**CCIFG**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

CM Bits 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

CCIS Bits 13-12 Capture/compare input select. These bits select the TAxCCRn 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

OUTMOD Bits 7-5 Output mode. Modes 2, 3, 6, and 7 are not useful for TAxCCR0 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

Timer_A Interrupt Vector Register (TAxIV)

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**	**TAIV**			**0**
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

TAIV Bits 15-0 Timer_A interrupt vector value

TAIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	Capture/compare 1	TAxCCR1 CCIFG	Highest
04h	Capture/compare 2	TAxCCR2 CCIFG	
06h	Capture/compare 3	TAxCCR3 CCIFG	
08h	Capture/compare 4	TAxCCR4 CCIFG	
0Ah	Capture/compare 5	TAxCCR5 CCIFG	
0Ch	Capture/compare 6	TAxCCR6 CCIFG	
0Eh	Timer overflow	TAxCTL TAIFG	Lowest

Timer_A Expansion 0 Register (TAxEX0)

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**	**TAIDEX**		
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

Unused Bits 15-3 Unused. Read only. Always read as 0.

TAIDEX Bits 2-0 Input divider expansion. These bits along with the ID bits select the divider for the input clock.

000 /1
001 /2
010 /3
011 /4
100 /5
101 /6
110 /7
111 /8

Timer_B

Timer_B is a 16-bit timer/counter with multiple capture/compare registers. There can be multiple Timer_B modules on a given device (see the device-specific data sheet). This chapter describes the operation and use of the Timer_B module.

Topic **Page**

12.1 Timer_B Introduction

Timer_B is a 16-bit timer/counter with up to 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 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, an associated action does not take place.

NOTE: Nomenclature

There may be multiple instantiations of Timer_B on a given device. The prefix TBx is used, where x is a greater than equal to zero indicating the Timer_B instantiation. For devices with one instantiation, x = 0. The suffix n, where n = 0 to 6, represents the specific capture/compare registers associated with the Timer_B instantiation.

12.1.1 Similarities and Differences From Timer_A

Timer_B is identical to Timer_A with the following exceptions:

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

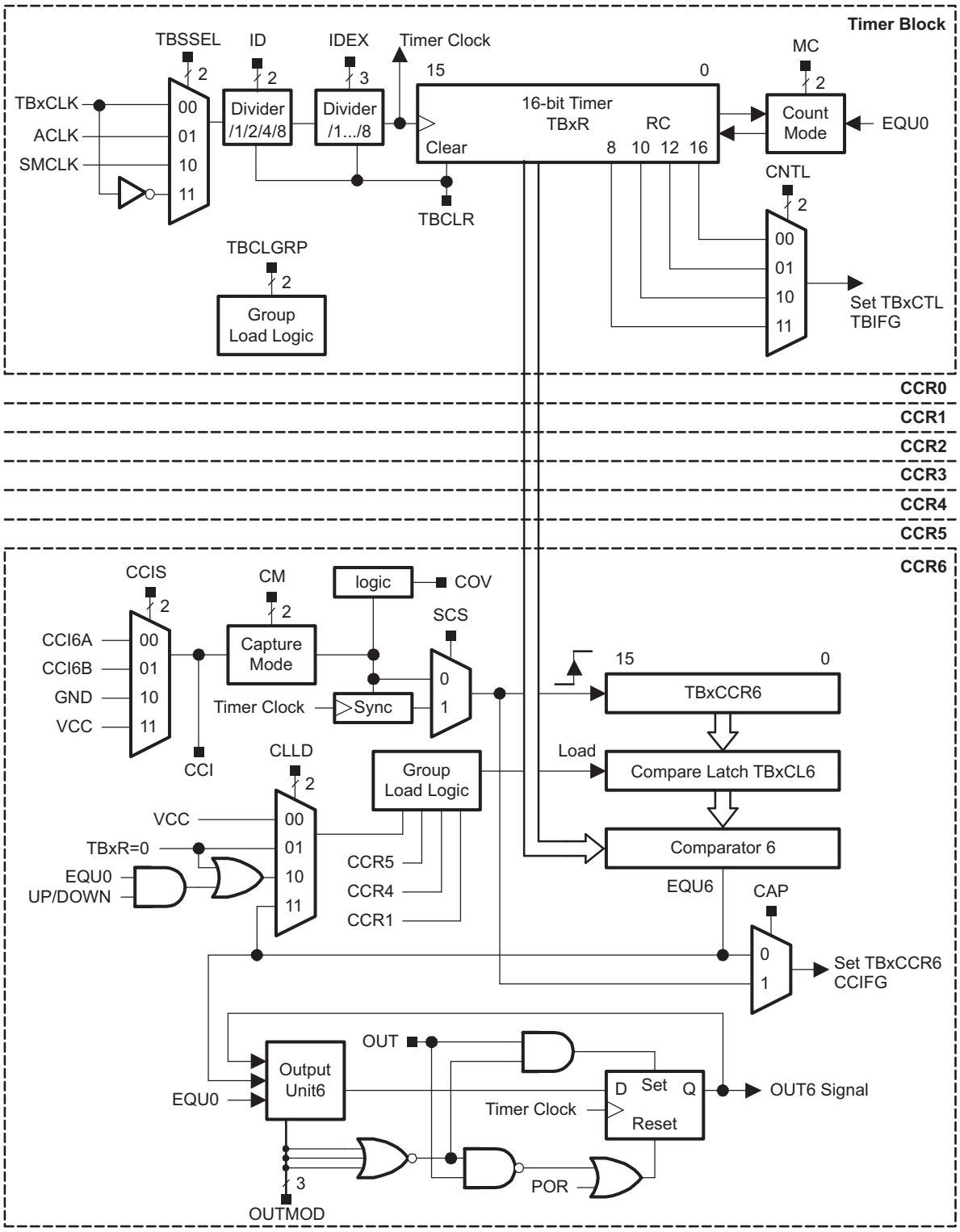


Figure 12-1. Timer_B Block Diagram

12.2 Timer_B Operation

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

12.2.1 16-Bit Timer Counter

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

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

NOTE: Modifying Timer_B registers

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

When the timer clock is asynchronous to the CPU clock, any read from TBxR 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 TBxR takes effect immediately.

12.2.1.1 TBxR Length

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

12.2.1.2 Clock Source Select and Divider

The timer clock can be sourced from ACLK, SMCLK, or externally via TBxCLK. The clock source is selected with the TBSSEL bits. The selected clock source may be passed directly to the timer or divided by 2,4, or 8, using the ID bits. The selected clock source can be further divided by 2, 3, 4, 5, 6, 7, or 8 using the TBIDEX bits. The timer clock dividers are reset when TBCLR is set.

NOTE: Timer_B dividers

Setting the TBCLR bit clears the contents of TBxR and the clock dividers. The clock dividers are implemented as down counters. Therefore, when the TBCLR bit is cleared, the timer clock immediately begins clocking at the first rising edge of the Timer_B clock source selected with the TBSSEL bits and continues clocking at the divider settings set by the ID and TBIDEX bits.

12.2.2 Starting the Timer

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

- The timer counts when MC > { 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 TBxCL0. The timer may then be restarted by loading a nonzero value to TBxCL0. 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: stop, up, continuous, and up/down (see Table 12-1). The operating mode is selected with the MC bits.

Table 12-1. Timer Modes

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of compare register TBxCL0.
10	Continuous	The timer repeatedly counts from zero to the value selected by the CNTL bits.
11	Up/down	The timer repeatedly counts from zero up to the value of TBxCL0 and then back down to zero.

12.2.3.1 Up Mode

The up mode is used if the timer period must be different from $TBxR_{(max)}$ counts. The timer repeatedly counts up to the value of compare latch TBxCL0, which defines the period (see Figure 12-2). The number of timer counts in the period is TBxCL0 + 1. When the timer value equals TBxCL0, the timer restarts counting from zero. If up mode is selected when the timer value is greater than TBxCL0, the timer immediately restarts counting from zero.

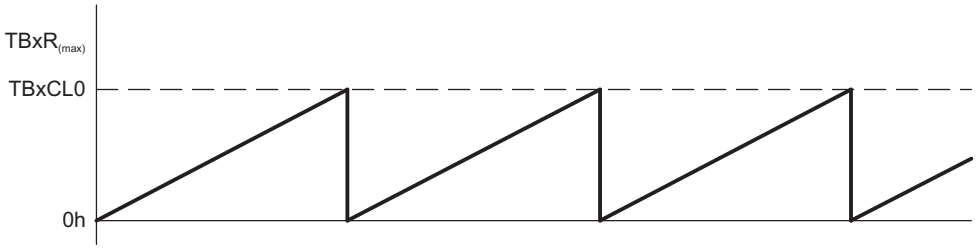


Figure 12-2. Up Mode

The TBxCCR0 CCIFG interrupt flag is set when the timer *counts* to the TBxCL0 value. The TBIFG interrupt flag is set when the timer *counts* from TBxCL0 to zero. Figure 12-3 shows the flag set cycle.

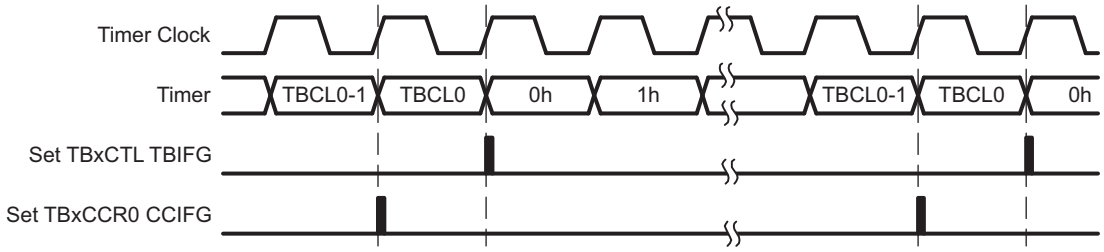


Figure 12-3. Up Mode Flag Setting

12.2.3.1.1 Changing Period Register TBxCL0

When changing TBxCL0 while the timer is running and when the TBxCL0 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.

12.2.3.2 Continuous Mode

In continuous mode, the timer repeatedly counts up to $TBxR_{(max)}$ and restarts from zero (see Figure 12-4). The compare latch TBxCL0 works the same way as the other capture/compare registers.

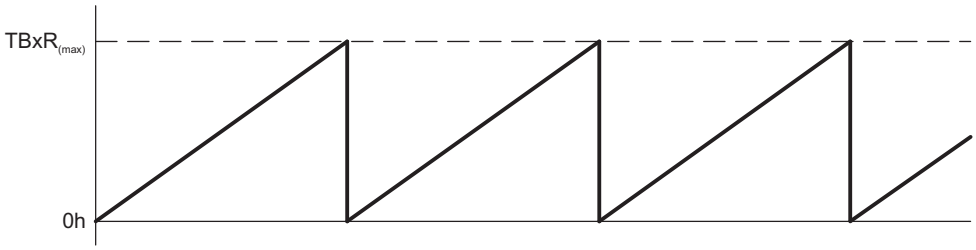


Figure 12-4. Continuous Mode

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

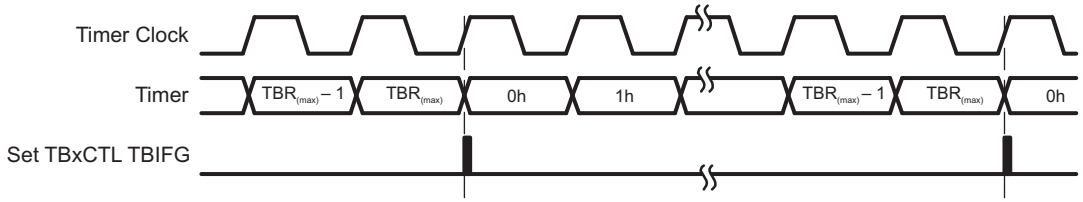


Figure 12-5. Continuous Mode Flag Setting

12.2.3.3 Use of 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 TBxCLn latch in the interrupt service routine. Figure 12-6 shows two separate time intervals, t_0 and t_1, being added to the capture/compare registers. The time interval is controlled by hardware, not software, without impact from interrupt latency. Up to n (where n = 0 to 7), independent time intervals or output frequencies can be generated using capture/compare registers.

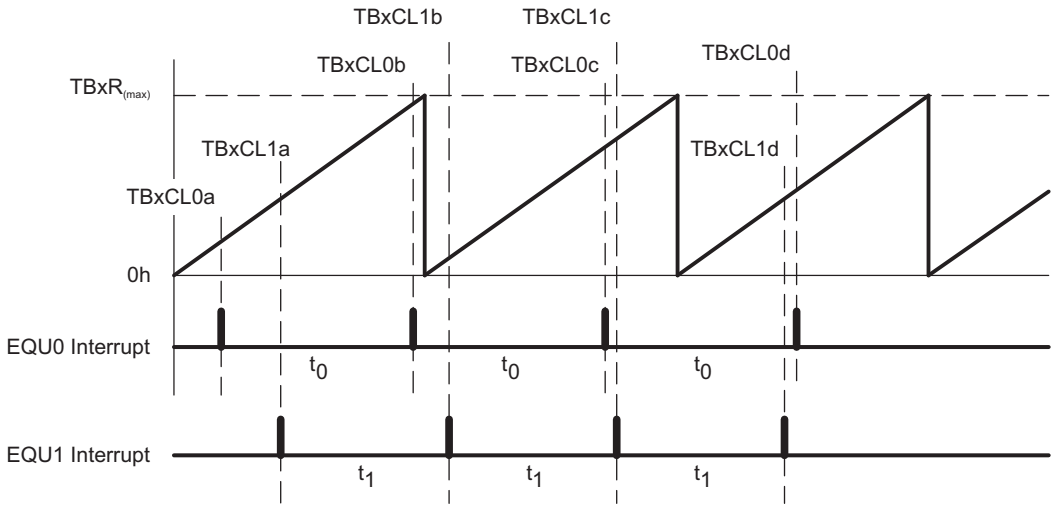


Figure 12-6. Continuous Mode Time Intervals

Time intervals can be produced with other modes as well, where TBxCL0 is used as the period register. Their handling is more complex, because the sum of the old TBxCLn data and the new period can be higher than the TBxCL0 value. When the sum of the previous TBxCLn value plus t_x is greater than the TBxCL0 data, the old TBxCL0 value must be subtracted to obtain the correct time interval.

12.2.3.4 Up/Down Mode

The up/down mode is used if the timer period must be different from $TBxR_{(max)}$ counts and if symmetrical pulse generation is needed. The timer repeatedly counts up to the value of compare latch TBxCL0, and back down to zero (see Figure 12-7). The period is twice the value in TBxCL0.

NOTE: TBxCL0 > $TBxR_{(max)}$

If TBxCL0 > $TBxR_{(max)}$, the counter operates as if it were configured for continuous mode. It does not count down from $TBxR_{(max)}$ to zero.

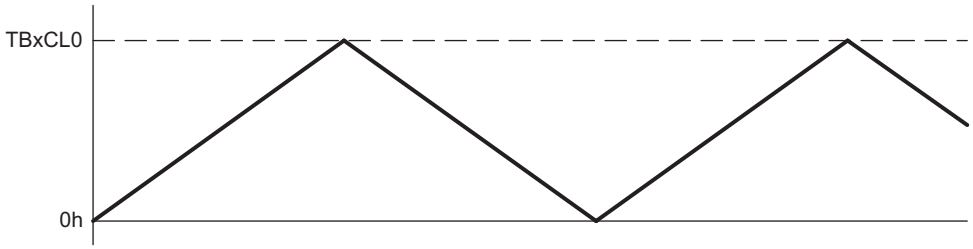


Figure 12-7. Up/Down Mode

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

In up/down mode, the TBxCCR0 CCIFG interrupt flag and the TBIFG interrupt flag are set only once during the period, separated by one-half the timer period. The TBxCCR0 CCIFG interrupt flag is set when the timer *counts* from TBxCL0-1 to TBxCL0, and TBIFG is set when the timer completes *counting* down from 0001h to 0000h. Figure 12-8 shows the flag set cycle.

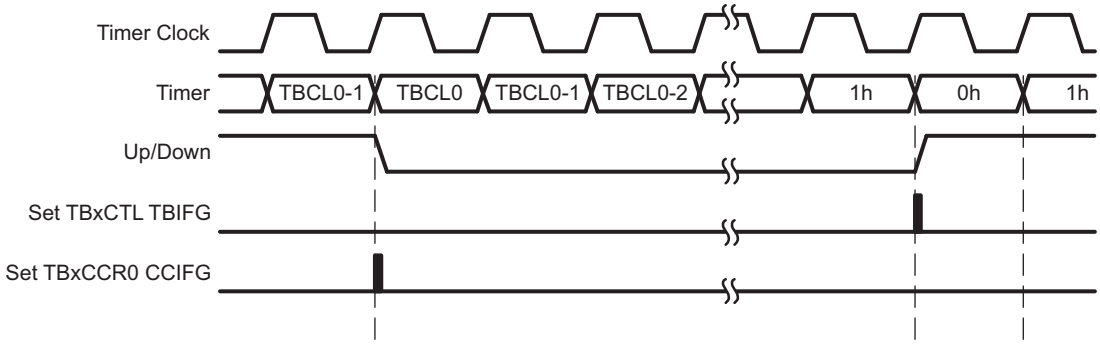


Figure 12-8. Up/Down Mode Flag Setting

12.2.3.4.1 Changing the Value of Period Register TBxCL0

When changing TBxCL0 while the timer is running and counting in the down direction, and when the TBxCL0 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 TBxCL0, 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 TBxCL0 is loaded, the timer begins counting down. However, one additional count may occur before the counter begins counting down.

12.2.3.5 Use of Up/Down Mode

The up/down mode supports applications that require dead times between output signals (see Section 12.2.5). 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:

$t_{dead} = t_{timer} \times (TBxCL1 - TBxCL3)$

Where:

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

t_{timer} = Cycle time of the timer clock

TBxCLn = Content of compare latch n

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

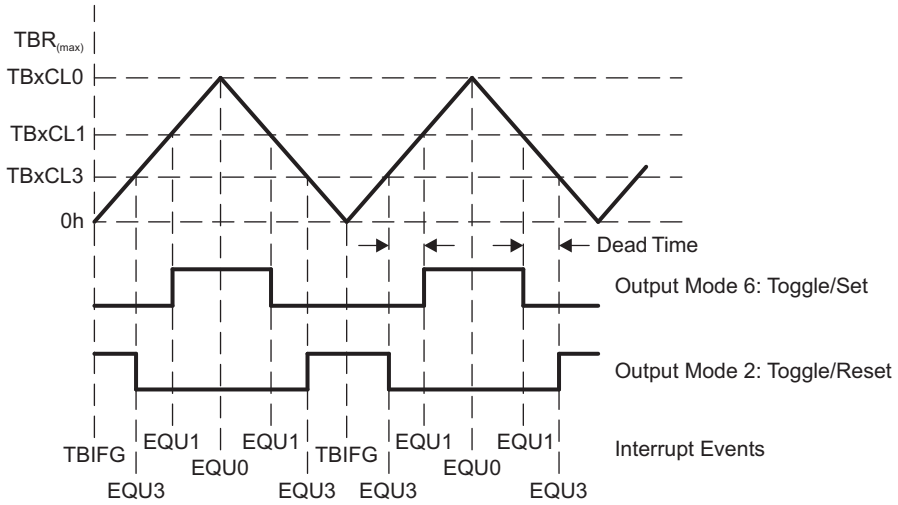


Figure 12-9. Output Unit in Up/Down Mode

12.2.4 Capture/Compare Blocks

Up to seven identical capture/compare blocks, TBxCCRn (where n = 0 to 6), are present in Timer_B. Any of the blocks may be used to capture the timer data or to generate time intervals.

12.2.4.1 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 CCIS bits. The CM 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 TBxCCRn register.
- The interrupt flag CCIFG is set.

The input signal level can be read at any time via the CCI bit. Devices may have different signals connected to CCIxA and CCIxB. See 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 synchronizes the capture with the next timer clock. Setting the SCS bit to synchronize the capture signal with the timer clock is recommended (see Figure 12-10).

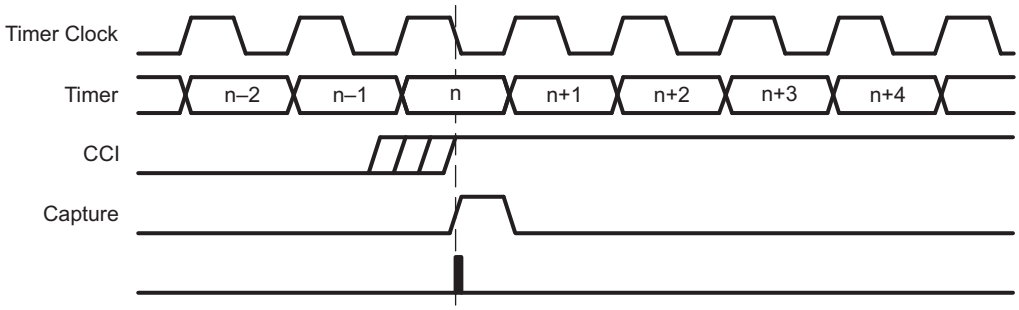


Figure 12-10. Capture Signal (SCS = 1)

NOTE: Changing Capture Inputs

Changing capture inputs while in capture mode may cause unintended capture events. To avoid this scenario, capture inputs should only be changed when capture mode is disabled (CM = {0} or CAP = 0).

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 (see Figure 12-11). COV must be reset with software.

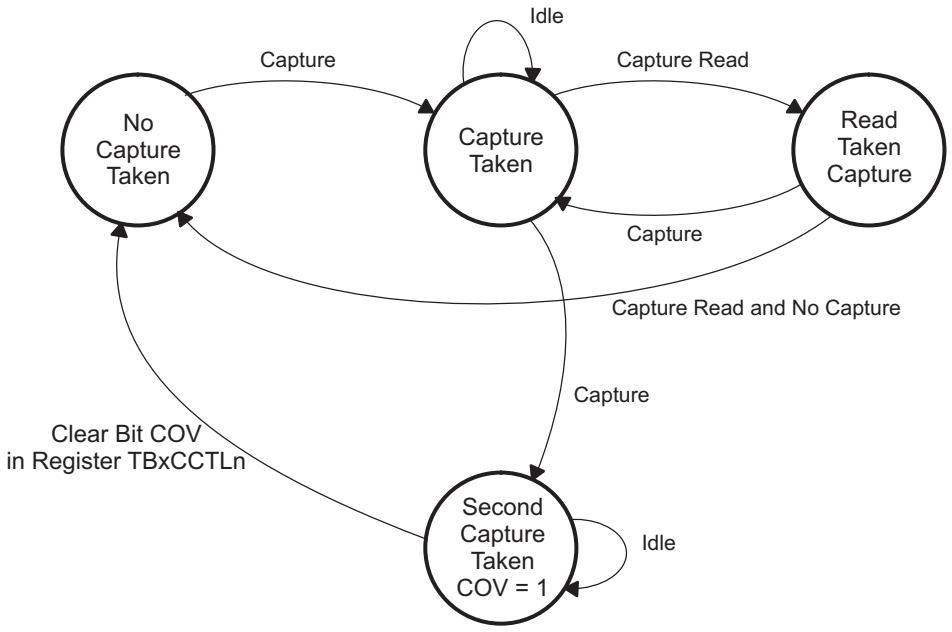


Figure 12-11. Capture Cycle

12.2.4.1.1 Capture Initiated by Software

Captures can be initiated by software. The CM 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,&TB0CCTL1   ; Setup TB0CCTL1
XOR     #CCIS0,&TB0CCTL1                ; TB0CCR1 = TB0R
```

NOTE: Capture Initiated by Software

In general, changing capture inputs while in capture mode may cause unintended capture events. For this scenario, switching the capture input between VCC and GND, disabling the capture mode is not required.

12.2.4.2 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 TBxR *counts* to the value in a TBxCLn, where n represents the specific capture/compare latch:

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

12.2.4.2.1 Compare Latch TBxCLn

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

Table 12-2. TBxCLn Load Events

CLLD	Description
00	New data is transferred from TBxCCRn to TBxCLn immediately when TBxCCRn is written to.
01	New data is transferred from TBxCCRn to TBxCLn when TBxR *counts* to 0.
10	New data is transferred from TBxCCRn to TBxCLn when TBxR *counts* to 0 for up and continuous modes. New data is transferred to from TBxCCRn to TBxCLn when TBxR *counts* to the old TBxCL0 value or to 0 for up/down mode.
11	New data is transferred from TBxCCRn to TBxCLn when TBxR *counts* to the old TBxCLn value.

12.2.4.2.2 Grouping Compare Latches

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

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

Table 12-3. Compare Latch Operating Modes

TBCLGRPx	Grouping	Update Control
00	None	Individual
01	TBxCL1+TBxCL2TBxCL3+TBxCL4+TBxCL5+TBxCL6	TBxCCR1 TBxCCR3 TBxCCR5
10	TBxCL1+TBxCL2+TBxCL3TBxCL4+TBxCL5+TBxCL6	TBxCCR1 TBxCCR4
11	TBxCL0+TBxCL1+TBxCL2+TBxCL3+TBxCL4+TBxCL5+TBxCL6	TBxCCR1

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 EQUn 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.

12.2.5.1 Output Modes

The output modes are defined by the OUTMOD bits and are described in Table 12-4. The OUTn 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 EQUn = EQU0.

Table 12-4. Output Modes

OUTMOD	Mode	Description
000	Output	The output signal OUTn is defined by the OUT bit. The OUTn signal updates immediately when OUT is updated.
001	Set	The output is set when the timer *counts* to the TBxCLn 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 TBxCLn value. It is reset when the timer *counts* to the TBxCL0 value.
011	Set/Reset	The output is set when the timer *counts* to the TBxCLn value. It is reset when the timer *counts* to the TBxCL0 value.
100	Toggle	The output is toggled when the timer *counts* to the TBxCLn value. The output period is double the timer period.
101	Reset	The output is reset when the timer *counts* to the TBxCLn 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 TBxCLn value. It is set when the timer *counts* to the TBxCL0 value.
111	Reset/Set	The output is reset when the timer *counts* to the TBxCLn value. It is set when the timer *counts* to the TBxCL0 value.

12.2.5.1.1 Output Example – Timer in Up Mode

The OUTn signal is changed when the timer *counts* up to the TBxCLn value, and rolls from TBxCL0 to zero, depending on the output mode. An example is shown in Figure 12-12 using TBxCL0 and TBxCL1.

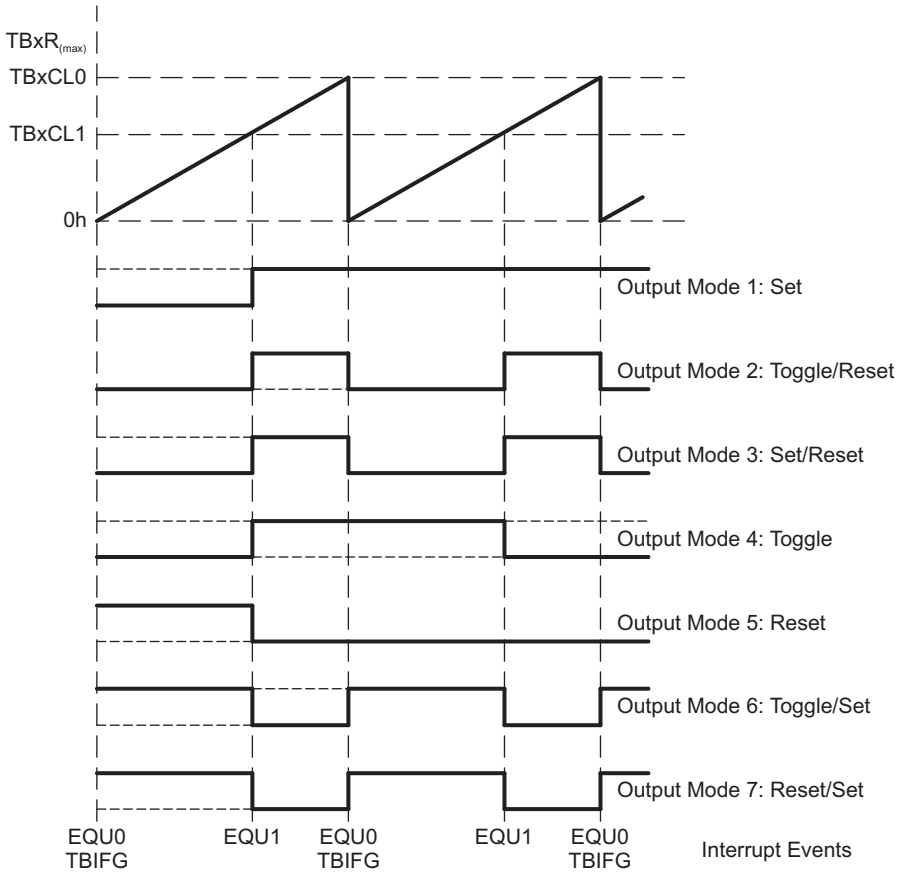


Figure 12-12. Output Example – Timer in Up Mode

12.2.5.1.2 Output Example – Timer in Continuous Mode

The OUTn signal is changed when the timer reaches the TBxCLn and TBxCL0 values, depending on the output mode. An example is shown in Figure 12-13 using TBxCL0 and TBxCL1.

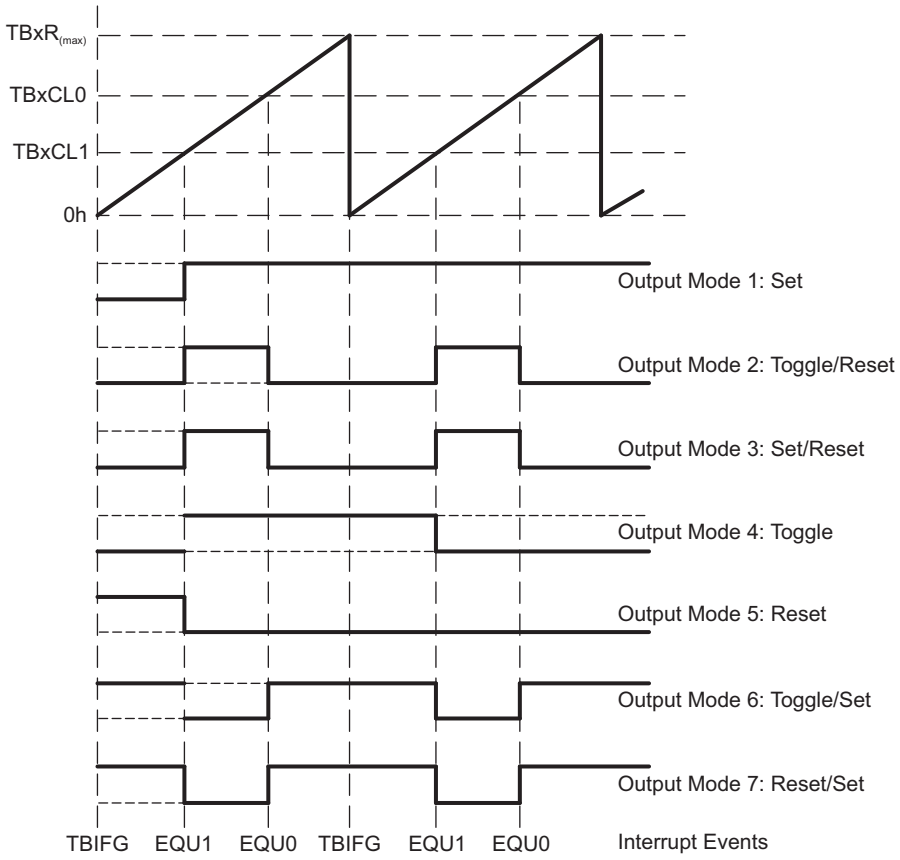


Figure 12-13. Output Example – Timer in Continuous Mode

12.2.5.1.3 Output Example – Timer in Up/Down Mode

The OUTn signal changes when the timer equals TBxCLn in either count direction and when the timer equals TBxCL0, depending on the output mode. An example is shown in Figure 12-14 using TBxCL0 and TBxCL3.

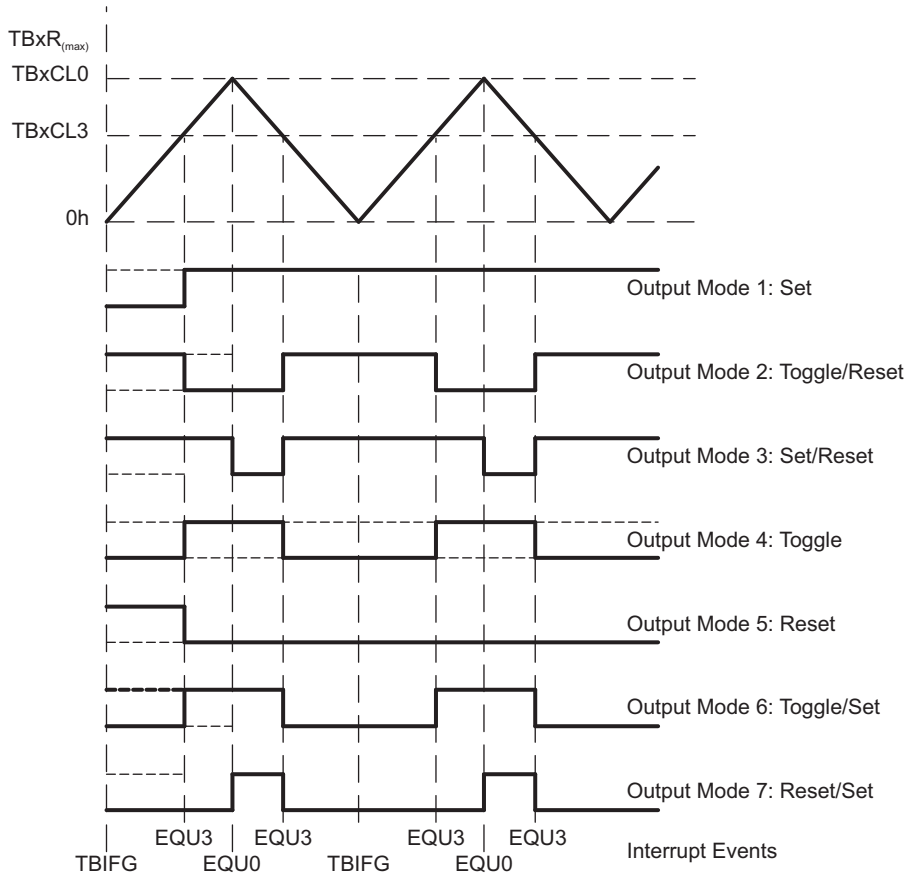


Figure 12-14. Output Example – Timer in Up/Down Mode

NOTE: Switching between output modes

When switching between output modes, one of the OUTMOD 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  #OUTMOD,&TBCCTLx    ; Clear unwanted bits
```

12.2.6 Timer_B Interrupts

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

- TBxCCR0 interrupt vector for TBxCCR0 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 TBxCCRn register. In compare mode, any CCIFG flag is set when TBxR *counts* to the associated TBxCLn 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.

12.2.6.1 TBxCCR0 Interrupt Vector

The TBxCCR0 CCIFG flag has the highest Timer_B interrupt priority and has a dedicated interrupt vector (see Figure 12-15). The TBxCCR0 CCIFG flag is automatically reset when the TBxCCR0 interrupt request is serviced.

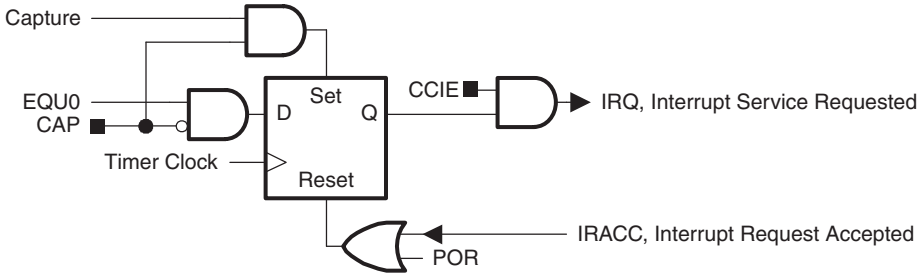


Figure 12-15. Capture/Compare TBxCCR0 Interrupt Flag

12.2.6.2 TBxIV, Interrupt Vector Generator

The TBIFG flag and TBxCCRn CCIFG flags (excluding TBxCCR0 CCIFG) are prioritized and combined to source a single interrupt vector. The interrupt vector register TBxIV is used to determine which flag requested an interrupt.

The highest-priority enabled interrupt (excluding TBxCCR0 CCIFG) generates a number in the TBxIV 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 TBxIV value.

Any access, read or write, of the TBxIV 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 TBxCCR1 and TBxCCR2 CCIFG flags are set when the interrupt service routine accesses the TBxIV register, TBxCCR1 CCIFG is reset automatically. After the RETI instruction of the interrupt service routine is executed, the TBxCCR2 CCIFG flag generates another interrupt.

12.2.6.3 TBxIV, Interrupt Handler Examples

The following software example shows the recommended use of TBxIV and the handling overhead. The TBxIV value is added to the PC to automatically jump to the appropriate routine. The example assumes a single instantiation of the largest timer configuration available.

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 TBxIV for Timer_B3.

```
; Interrupt handler for TB0CCR0 CCIFG.                    Cycles
CCIFG_0_HND
;        ...         ; Start of handler Interrupt latency    6
         RETI                                                5

; Interrupt handler for TB0IFG, TB0CCR1 through TB0CCR6 CCIFG.

TB0_HND      ...               ; Interrupt latency           6
         ADD      &TB0IV,PC    ; Add offset to Jump table    3
         RETI                  ; Vector  0: No interrupt     5
         JMP      CCIFG_1_HND  ; Vector  2: TB0CCR1          2
         JMP      CCIFG_2_HND  ; Vector  4: TB0CCR2          2
         JMP      CCIFG_3_HND  ; Vector  6: TB0CCR3          2
         JMP      CCIFG_4_HND  ; Vector  8: TB0CCR4          2
         JMP      CCIFG_5_HND  ; Vector 10: TB0CCR5          2
         JMP      CCIFG_6_HND  ; Vector 12: TB0CCR6          2

TB0IFG_HND                    ; Vector 14: TB0IFG Flag
         ...                  ; Task starts here
         RETI                                                5

CCIFG_6_HND                   ; Vector 12: TB0CCR6
         ...                  ; Task starts here
         RETI                 ; Back to main program        5

CCIFG_5_HND                   ; Vector 10: TB0CCR5
         ...                  ; Task starts here
         RETI                 ; Back to main program        5

CCIFG_4_HND                   ; Vector 8: TB0CCR4
         ...                  ; Task starts here
         RETI                 ; Back to main program        5

CCIFG_3_HND                   ; Vector 6: TB0CCR3
         ...                  ; Task starts here
         RETI                 ; Back to main program        5

CCIFG_2_HND                   ; Vector 4: TB0CCR2
         ...                  ; Task starts here
         RETI                 ; Back to main program        5

CCIFG_1_HND                   ; Vector 2: TB0CCR1
         ...                  ; Task starts here
         RETI                 ; Back to main program        5
```

12.3 Timer_B Registers

The Timer_B registers are listed in Table 12-5. The base address can be found in the device-specific data sheet. The address offset is listed in Table 12-5.

Table 12-5. Timer_B Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Timer_B Control	TBxCTL	Read/write	Word	00h	0000h
Timer_B Capture/Compare Control 0	TBxCCTL0	Read/write	Word	02h	0000h
Timer_B Capture/Compare Control 1	TBxCCTL1	Read/write	Word	04h	0000h
Timer_B Capture/Compare Control 2	TBxCCTL2	Read/write	Word	06h	0000h
Timer_B Capture/Compare Control 3	TBxCCTL3	Read/write	Word	08h	0000h
Timer_B Capture/Compare Control 4	TBxCCTL4	Read/write	Word	0Ah	0000h
Timer_B Capture/Compare Control 5	TBxCCTL5	Read/write	Word	0Ch	0000h
Timer_B Capture/Compare Control 6	TBxCCTL6	Read/write	Word	0Eh	0000h
Timer_B Counter	TBxR	Read/write	Word	10h	0000h
Timer_B Capture/Compare 0	TBxCCR0	Read/write	Word	12h	0000h
Timer_B Capture/Compare 1	TBxCCR1	Read/write	Word	14h	0000h
Timer_B Capture/Compare 2	TBxCCR2	Read/write	Word	16h	0000h
Timer_B Capture/Compare 3	TBxCCR3	Read/write	Word	18h	0000h
Timer_B Capture/Compare 4	TBxCCR4	Read/write	Word	1Ah	0000h
Timer_B Capture/Compare 5	TBxCCR5	Read/write	Word	1Ch	0000h
Timer_B Capture/Compare 6	TBxCCR6	Read/write	Word	1Eh	0000h
Timer_B Interrupt Vector	TBxIV	Read only	Word	2Eh	0000h
Timer_B Expansion 0	TBxEX0	Read/write	Word	20h	0000h

Timer_B Control Register (TBxCTL)

15	14	13	12	11	10	9	8
Unused	**TBCLGRPx**		**CNTL**		**Unused**	**TBSSEL**	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID		**MC**		**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 Bits 14-13 TBxCLn group

- 00 Each TBxCLn latch loads independently.
- 01 TBxCL1+TBxCL2 (TBxCCR1 CLLD bits control the update)
TBxCL3+TBxCL4 (TBxCCR3 CLLD bits control the update)
TBxCL5+TBxCL6 (TBxCCR5 CLLD bits control the update)
TBxCL0 independent
- 10 TBxCL1+TBxCL2+TBxCL3 (TBxCCR1 CLLD bits control the update)
TBxCL4+TBxCL5+TBxCL6 (TBxCCR4 CLLD bits control the update)
TBxCL0 independent
- 11 TBxCL0+TBxCL1+TBxCL2+TBxCL3+TBxCL4+TBxCL5+TBxCL6 (TBxCCR1 CLLD bits control the update)

CNTL Bits 12-11 Counter length

- 00 16-bit, $TBxR_{(max)}$ = 0FFFFh
- 01 12-bit, $TBxR_{(max)}$ = 0FFFh
- 10 10-bit, $TBxR_{(max)}$ = 03FFh
- 11 8-bit, $TBxR_{(max)}$ = 0FFh

Unused Bit 10 Unused

TBSSEL Bits 9-8 Timer_B clock source select

- 00 TBxCLK
- 01 ACLK
- 10 SMCLK
- 11 Inverted TBxCLK

ID Bits 7-6 Input divider. These bits, along with the TBIDEX bits, select the divider for the input clock.

- 00 /1
- 01 /2
- 10 /4
- 11 /8

MC Bits 5-4 Mode control. Setting MC = 00h when Timer_B is not in use conserves power.

- 00 Stop mode: Timer is halted
- 01 Up mode: Timer counts up to TBxCL0
- 10 Continuous mode: Timer counts up to the value set by CNTL
- 11 Up/down mode: Timer counts up to TBxCL0 and down to 0000h

Unused Bit 3 Unused

TBCLR Bit 2 Timer_B clear. Setting this bit resets TBxR, the timer clock divider, and the count direction. The TBCLR bit is automatically reset and is always read as zero.

TBIE Bit 1 Timer_B interrupt enable. This bit enables the TBIFG interrupt request.

- 0 Interrupt disabled
- 1 Interrupt enabled

TBIFG Bit 0 Timer_B interrupt flag

- 0 No interrupt pending
- 1 Interrupt pending

Timer_B Counter Register (TBxR)

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

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

Capture/Compare Control Register (TBxCCTLn)

15	14	13	12	11	10	9	8
CM		**CCIS**		**SCS**	**CLLD**		**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
OUTMOD			**CCIE**	**CCI**	**OUT**	**COV**	**CCIFG**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

CM Bits 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

CCIS Bits 13-12 Capture/compare input select. These bits select the TBxCCRn 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

CLLD Bits 10-9 Compare latch load. These bits select the compare latch load event.
- 00 TBxCLn loads on write to TBxCCRn
- 01 TBxCLn loads when TBxR *counts* to 0
- 10 TBxCLn loads when TBxR *counts* to 0 (up or continuous mode)
TBxCLn loads when TBxR *counts* to TBxCL0 or to 0 (up/down mode)
- 11 TBxCLn loads when TBxR *counts* to TBxCLn

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

OUTMOD Bits 7-5 Output mode. Modes 2, 3, 6, and 7 are not useful for TBxCL0 because EQUn = 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

Timer_B Interrupt Vector Register (TBxIV)

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**	**TBIV**			**0**
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

TBIV Bits 15-0 Timer_B interrupt vector value

TBIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	Capture/compare 1	TBxCCR1 CCIFG	Highest
04h	Capture/compare 2	TBxCCR2 CCIFG	
06h	Capture/compare 3	TBxCCR3 CCIFG	
08h	Capture/compare 4	TBxCCR4 CCIFG	
0Ah	Capture/compare 5	TBxCCR5 CCIFG	
0Ch	Capture/compare 6	TBxCCR6 CCIFG	
0Eh	Timer overflow	TBxCTL TBIFG	Lowest

Timer_B Expansion Register 0 (TBxEX0)

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**	**TBIDEX**		
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

Unused Bits 15-3 Unused. Read only. Always read as 0.

TBIDEX Bits 2-0 Input divider expansion. These bits along with the ID bits select the divider for the input clock.

000 /1
001 /2
010 /3
011 /4
100 /5
101 /6
110 /7
111 /8

Real-Time Clock B (RTC_B)

The real-time clock RTC_B module provides clock counters with calendar mode, a flexible programmable alarm, and calibration. The RTC_B also support operation in LPMx.5 and device-dependent operation from a backup supply. See the device-specific data sheet for the supported features. This chapter describes the RTC_B module.

Topic **Page**

13.1 Real-Time Clock RTC_B Introduction

The RTC_B module provides configurable clock counters.

RTC_B features include:

- Real-time clock and calendar mode providing seconds, minutes, hours, day of week, day of month, month, and year (including leap year correction)
- Interrupt capability
- Selectable BCD or binary format
- Programmable alarms
- Calibration logic for time offset correction
- Operation in LPMx.5
- Operation from backup supply with programmable charger for backup supply (device-dependent) (see the *Battery Backup System* chapter.

The RTC_B block diagram for devices supporting LPMx.5 is shown in Figure 13-1.

NOTE: Real-time clock initialization

Most RTC_B module registers have no initial condition. These registers must be configured by user software before use.

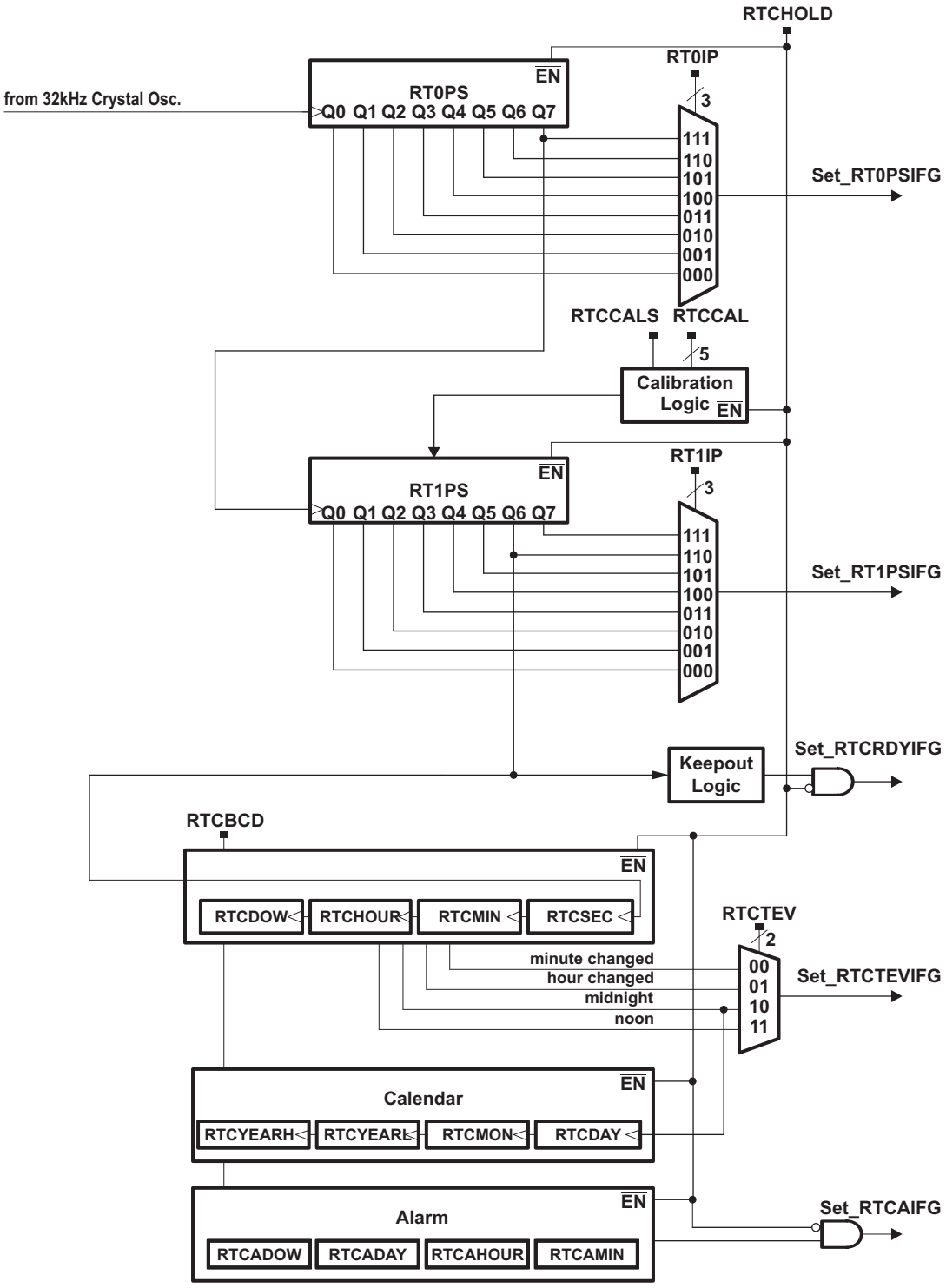


Figure 13-1. RTC_B Block Diagram

13.2 RTC_B Operation

The RTC_B 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 four as leap years. This algorithm is accurate from the year 1901 through 2099.

13.2.1 Real-Time Clock and Prescale Dividers

The prescale dividers, RT0PS and RT1PS, are automatically configured to provide a 1-s clock interval for the RTC_B. The low-frequency oscillator must be operated at 32768 Hz (nominal) for proper RTC_B operation. RT0PS is sourced from the low-frequency oscillator XT1. The output of RT0PS / 256 (Q7) is used to source RT1PS. RT1PS is further divider and the /128 output sources the real-time clock counter registers providing the required 1-second time interval. When RTCBCD = 1, BCD format is selected for the calendar registers. Setting RTCHOLD halts the real-time counters and prescale counters, RT0PS and RT1PS.

13.2.2 Real-Time Clock Alarm Function

The RTC_B 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.

Each alarm register contains an alarm enable (AE) bit 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.

- Example 1: A user wishes to set an alarm every hour at 15 minutes past the hour (that is, at 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 is enabled. When enabled, the RTCAIFG is 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.
- Example 2: 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 is enabled. When enabled, the RTCAIFG is set when the count transitions from 03:59:59 to 04:00:00.
- Example 3: 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 is enabled. Once enabled, the RTCAIFG is set when the time count transitions from 06:29:59 to 06:30:00. In this case, the alarm event occurs every day at 06:30:00.
- Example 4: 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 is enabled. Once enabled, the RTCAIFG is set when the time count transitions from 06:29:59 to 06:30:00 and the RTCDOW transitions from 1 to 2.
- Example 5: 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 is enabled. Once enabled, the RTCAIFG is set when the time count transitions from 06:29:59 to 06:30:00 and the RTCDAY equals 5.

NOTE: Setting the alarm

Prior to setting an initial alarm, all alarm registers including the AE bits should be cleared.

To prevent potential erroneous alarm conditions from occurring, the alarms should be disabled by clearing the RTCAIE, RTCAIFG, and AE bits prior to writing initial or new time values to the RTC time registers.

NOTE: Invalid alarm settings

Invalid alarm settings are not checked via hardware. It is the user's 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, RTCYEAR, RTCAMIN, RTCAHOUR, RTCADAY, and RTCADOW registers can result in unpredictable behavior.

13.2.3 Reading or Writing Real-Time Clock Registers

Because the system clock may in fact be asynchronous to the RTC_B clock source, special care must be used when accessing the real-time clock registers.

The real-time clock registers are updated once per second. To prevent reading any real-time clock register at the time of an update that could result in an invalid time being read, a keep-out window is provided. The keep-out window is centered approximately 128/32768 seconds around the update transition. The read-only RTCRDY bit is reset during the keep-out window period and set outside the keep-out 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 is 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 it can be reset with software.

NOTE: Reading or writing real-time clock registers

When the counter clock is asynchronous to the CPU clock, any read from any RTCSEC, RTCMIN, RTCHOUR, RTCDOW, RTCDAY, RTCMON, or RTCYEAR register while the RTCRDY is reset may result in invalid data being read. To safely read the counting registers, either polling of the RTCRDY bit or the synchronization procedure previously described 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 the RT0PS and RT1PS 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 1 second during a write. Writing of data outside the legal ranges or invalid time stamp combinations results in unpredictable behavior.

13.2.4 Real-Time Clock Interrupts

Six sources for interrupts are available, namely RT0PSIFG, RT1PSIFG, RTCRDYIFG, RTCTEVIFG, RTCAIFG, and RTCOFIFG. 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 (PC) 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 RTCAIE enables the interrupt. In addition to the user-programmable alarm, the RTC_B 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 or 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.

RT0PSIFG can be used to generate interrupt intervals selectable by the RT0IP bits. RT0PS is sourced with low-frequency oscillator clock 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.

RT1PSIFG can be used to generate interrupt intervals selectable by the RT1IP bits. RT1PS is sourced with the output of RT0PS, which is 128 Hz (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.

The RTCOFIFG bit flags a failure of the 32-kHz crystal oscillator. Its main purpose is to wake-up the CPU from LPM3.5 in case an oscillator failure occurred. On device with a backup-supply sub-system it also stores a failure event that occurred while the RTC was operating on the backup supply.

13.2.4.0.1 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_HND    ; Vector 6: RTCAIFG         5
           JMP RT0PSIFG_HND   ; Vector 8: RT0PSIFG        5
           JMP RT1PSIFG_HND   ; Vector A: RT1PSIFG        5
           JMP RTCOFIFG_HND   ; Vector C: RTCOFIFG        5
           RETI               ; Vector E: Reserved        5

RTCRDYIFG_HND                 ; Vector 2: RTCRDYIFG Flag
           ...                ; Task starts here
           RETI               ; Back to main program      5

RTCTEVIFG_HND                 ; Vector 4: RTCTEVIFG Flag
           ...                ; Task starts here
           RETI               ; Back to main program      5

RTCAIFG_HND                   ; Vector 6: RTCAIFG Flag
           ...                ; Task starts here
           RETI               ; Back to main program      5

RT0PSIFG_HND                  ; Vector 8: RT0PSIFG Flag
           ...                ; Task starts here
           RETI               ; Back to main program      5

RT1PSIFG_HND                  ; Vector A: RT1PSIFG Flag
           ...                ; Task starts here
           RETI               ; Back to main program      5
```

```
RTCOFIFG_HND                ; Vector C: RTCOFIFG Flag
           ...              ; Task starts here
           RETI             ; Back to main program      5
```

13.2.5 Real-Time Clock Calibration

The RTC_B module has calibration logic that allows for adjusting the crystal frequency in approximately +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 causes a ≈ +4-ppm adjustment. When RTCCALS is cleared, each RTCCALx LSB causes a ≈ –2-ppm adjustment. Calibration is available in calendar mode only. In counter mode (RTCMODE = 0), the calibration logic is disabled.

Calibration is accomplished by periodically adjusting the RT1PS counter based on the RTCCALS and RTCCALx settings. In calendar mode, the RT0PS divides the nominal 37268-Hz low-frequency (LF) crystal clock input by 256. A 60-minute period has 32768 cycles/sec × 60 sec/min × 60 min = 117964800 cycles. Therefore, a –2-ppm reduction in frequency (down calibration) approximately equates to adding an additional 256 cycles every 117964800 cycles (256/117964800 = 2.17 ppm). This is accomplished by holding the RT1PS counter for one additional clock of the RT0PS output within a 60-minute period. Similarly, a +4-ppm increase in frequency (up calibration) approximately equates to removing 512 cycles every 117964800 cycle (512/117964800 = 4.34 ppm). This is accomplished by incrementing the RT1PS counter for two additional clocks of the RT0PS output within a 60-minute period. Each RTCCALx calibration bit causes either 256 LF crystal clock cycles to be added every 60 minutes or 512 LF crystal clock cycles to be subtracted every 60 minutes, giving a frequency adjustment of approximately -2 ppm or +4 ppm, respectively.

To calibrate the frequency, the RTCCLK output signal is available at a pin. RTCCALF bits can be used to select the frequency rate of the output signal, either no signal, 512 Hz, 256 Hz, or 1 Hz.

The basic flow to calibrate the frequency is as follows:

1. Configure the RTCCLK pin.
2. Measure the RTCCLK output signal with an appropriate resolution frequency counter ; that is, within the resolution required.
3. Compute the absolute error in ppm: Absolute error (ppm) = $|10^6 (f_{MEASURED} - f_{RTCCLK})/f_{RTCCLK}|$, where f_{RTCCLK} is the expected frequency of 512 Hz, 256 Hz, or 1 Hz.
4. Adjust the frequency by performing the following:
   (a) If the frequency is too low, set RTCCALS = 1 and apply the appropriate RTCCALx bits, where RTCCALx = (Absolute Error) / 4.34 rounded to the nearest integer
   (b) If the frequency is too high, clear RTCCALS = 0 and apply the appropriate RTCCALx bits, where RTCCALx = (Absolute Error) / 2.17 rounded to the nearest integer

For example, assume that RTCCLK is configured to output at a frequency of 512 Hz. The measured RTCCLK is 511.9658 Hz. This frequency error is approximately 66.8 ppm too low. To increase the frequency by 66.8 ppm, RTCCALS would be set, and RTCCALx would be set to 15 (66.8/4.34). Similarly, assume that the measured RTCCLK is 512.0125 Hz. The frequency error is approximately 24.4 ppm too high. To decrease the frequency by 24.4 ppm, RTCCALS would be cleared, and RTCCAL would be set to 11 (24.4/2.17).

The calibration corrects only initial offsets and does not adjust for temperature and aging effects. These effects can be handled by periodically measuring temperature and using the crystal's characteristic curve to adjust the ppm based on temperature, as required. 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 affected by changes in the calibration settings since these output frequencies are generated prior to the calibration logic. The 1-Hz output frequency is affected by changes in the calibration settings. Because the frequency change is small and infrequent over a very long time interval, it can be difficult to observe.

13.2.6 Real-Time Clock Operation in LPMx.5 Low Power Mode

The regulator of the Power Management Module (PMM) is disabled upon entering LPMx.5, which causes most of the RTC_B configuration registers to be lost; only the counters and the backup RAM are retained. Table 13-1 lists the retained registers in LPMx.5. Also the configuration of the interrupts is stored so that the configured interrupts can cause a wakeup upon exit from LPMx.5. The interrupt flags RTCTEVIFG, RTCAIFG, RT1PSIFG, and RTCOFIFG can be used as RTC_B wake-up interrupt sources. After restoring the configuration registers (and clearing LOCKLPM5) the interrupts can be serviced as usual. The detailed flow is as follows:

1. Set all I/Os to general purpose I/Os and configure as needed. Optionally configure input interrupt pins for wake-up. Configure RTC_B interrupts for wake-up (set RTCTEVIE, RTCAIE, RT1PSIE, or RTCOFIE. If the alarm interrupt is also used as wake-up event, the alarm registers must be configured as needed).
2. Enter LPMx.5 with LPMx.5 entry sequence.

```
MOV #PMMKEY + PMMREGOFF, &PMMCTL0 ; Open PMM registers for write and set PMMREGOFF
                                  ;
BIS #LPM4,SR                      ; Enter LPMx.5 when PMMREGOFF is set
```

3. LOCKLPM5 is automatically set by hardware upon entering LPMx.5, the core voltage regulator is disabled, and all clocks are disabled except for the 32-kHz crystal oscillator clock if the RTC is enabled with RTCHOLD = 0.
4. An LPMx.5 wake-up event, such as an edge on a wake-up input pin, are an RTC_B interrupt event and start the BOR entry sequence together with the core voltage regulator. All peripheral registers are set to their default conditions. The I/O pin state remains locked as well as the interrupt configuration for the RTC_B.
5. The device can be configured. The I/O configuration and the RTC_B interrupt configuration that was not retained during LPMx.5 should be restored to the values prior to entering LPMx.5. Then the LOCKLPM5 bit can be cleared, this releases the I/O pin conditions as well as the RTC_B interrupt configuration.
6. After enabling I/O and RTC_B interrupts, the interrupt that caused the wake-up can be serviced.
7. To re-enter LPMx.5, the LOCKLPM5 bit must be cleared prior to re-entry, otherwise LPMx.5 will not be entered.

If the RTC is enabled (RTCHOLD = 0), the 32-kHz oscillator remains active during LPMx.5. The fault detection also remains functional. If a fault occurs during LPMx.5 and the RTCOFIE was set before entering LPMx.5, a wake-up event is issued.

13.3 Real-Time Clock Registers

The RTC_B module registers are listed in Table 13-1. This table also lists the retention during LPMx.5. Registers that are not retained during LPMx.5 must be restored after exit from LPMx.5. The base address for the RTC_B module registers can be found in the device-specific data sheet. The address offsets are given in Table 13-1.

NOTE: Most registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 13-1. RTC_B Real-Time Clock Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State	LPMx.5 / Backup Op.
Real-Time Clock Control 0, 1	RTCCTL01	Read/write	Word	00h	4000h	not retained
Real-Time Clock Control 0	RTCCTL0 or RTCCTL01_L	Read/write	Byte	00h	00h	not retained
Real-Time Clock Control 1	RTCCTL1 or RTCCTL01_H	Read/write	Byte	01h	40h	not retained
Real-Time Clock Control 2, 3	RTCCTL23	Read/write	Word	02h	0000h	retained
Real-Time Clock Control 2	RTCCTL2 or RTCCTL23_L	Read/write	Byte	02h	00h	retained
Real-Time Clock Control 3	RTCCTL3 or RTCCTL23_H	Read/write	Byte	03h	00h	retained
Real-Time Prescale Timer 0 Control	RTCPS0CTL	Read/write	Word	08h	0000h	not retained
	RTCPS0CTLL or RTCPS0CTL_L	Read/write	Byte	08h	00h	not retained
	RTCPS0CTLH or RTCPS0CTL_H	Read/write	Byte	09h	00h	not retained
Real-Time Prescale Timer 1 Control	RTCPS1CTL	Read/write	Word	0Ah	0000h	not retained
	RTCPS1CTLL or RTCPS1CTL_L	Read/write	Byte	0Ah	00h	not retained
	RTCPS0CTLH or RTCPS0CTL_H	Read/write	Byte	0Bh	00h	not retained
Real-Time Prescale Timer 0, 1 Counter	RTCPS	Read/write	Word	0Ch	none	retained
Real-Time Prescale Timer 0 Counter	RT0PS or RTCPS_L	Read/write	Byte	0Ch	none	retained
Real-Time Prescale Timer 1 Counter	RT1PS or RTCPS_H	Read/write	Byte	0Dh	none	retained
Real Time Clock Interrupt Vector	RTCIV	Read	Word	0Eh	0000h	not retained
Real-Time Clock Seconds, Minutes	RTCTIM0	Read/write	Word	10h	undefined	retained
Real-Time Clock Seconds	RTCSEC or RTCTIM0_L	Read/write	Byte	10h	undefined	retained
Real-Time Clock Minutes	RTCMIN or RTCTIM0_H	Read/write	Byte	11h	undefined	retained
Real-Time Clock Hour, Day of Week	RTCTIM1	Read/write	Word	12h	undefined	retained
Real-Time Clock Hour	RTCHOUR or RTCTIM1_L	Read/write	Byte	12h	undefined	retained
Real-Time Clock Day of Week	RTCDOW or RTCTIM1_H	Read/write	Byte	13h	undefined	retained

Table 13-1. RTC_B Real-Time Clock Registers (continued)

Register	Short Form	Register Type	Register Access	Address Offset	Initial State	LPMx.5 / Backup Op.
Real-Time Clock Date	RTCDATE	Read/write	Word	14h	undefined	retained
Real-Time Clock Day of Month	RTCDAY or RTCDATE_L	Read/write	Byte	14h	undefined	retained
Real-Time Clock Month	RTCMON or RTCDATE_H	Read/write	Byte	15h	undefined	retained
Real-Time Clock Year[1]	RTCYEAR	Read/write	Word	16h	undefined	retained
Real-Time Clock Minutes, Hour Alarm	RTCAMINHR	Read/write	Word	18h	undefined	retained
Real-Time Clock Minutes Alarm	RTCAMIN or RTCAMINHR_L	Read/write	Byte	18h	undefined	retained
Real-Time Clock Hours Alarm	RTCAHOUR or RTCAMINHR_H	Read/write	Byte	19h	undefined	retained
Real-Time Clock Day of Week, Day of Month Alarm	RTCADOWDAY	Read/write	Word	1Ah	undefined	retained
Real-Time Clock Day of Week Alarm	RTCADOW or RTCADOWDAY_L	Read/write	Byte	1Ah	undefined	retained
Real-Time Clock Day of Month Alarm	RTCADAY or RTCADOWDAY_H	Read/write	Byte	1Bh	undefined	retained
Binary-to-BCD conversion register	BIN2BCD	Read/write	Word	1Ch	00h	not retained
BCD-to-binary conversion register	BCD2BIN	Read/write	Word	1Eh	00h	not retained

(1) The year register RTCYEAR must not be accessed in byte mode!

Real-Time Clock Control 0 Register (RTCCTL0)

7	6	5	4	3	2	1	0
RTCOFIE(1)	**RTCTEVIE**(1)	**RTCAIE**(1)	**RTCRDYIE**	**RTCOFIFG**	**RTCTEVIFG**	**RTCAIFG**	**RTCRDYIFG**
rw-0	rw-0	rw-0	rw-0	rw-(0)	rw-(0)	rw-(0)	rw-(0)

RTCOFIE	Bit 7	32-kHz crystal oscillator fault interrupt enable. This interrupt can be used as LPMx.5 wake-up event. 0 Interrupt not enabled 1 Interrupt enabled. (LPMx.5 wake-up enabled.)
RTCTEVIE	Bit 6	Real-time clock time event interrupt enable. In modules supporting LPMx.5 this interrupt can be used as LPMx.5 wake-up event. 0 Interrupt not enabled 1 Interrupt enabled. (LPMx.5 wake-up enabled.)
RTCAIE	Bit 5	Real-time clock alarm interrupt enable. In modules supporting LPMx.5 this interrupt can be used as LPMx.5 wake-up event. 0 Interrupt not enabled 1 Interrupt enabled. (LPMx.5 wake-up enabled.)
RTCRDYIE	Bit 4	Real-time clock ready interrupt enable. 0 Interrupt not enabled 1 Interrupt enabled
RTCOFIFG	Bit 3	32-kHz crystal oscillator fault interrupt flag. This interrupt can be used as LPMx.5 wake-up event. It also indicates a clock failure during backup operation. 0 No interrupt pending 1 Interrupt pending. A 32-kHz crystal oscillator fault occurred after last reset.
RTCTEVIFG	Bit 2	Real-time clock time event interrupt flag. In modules supporting LPMx.5 this interrupt can be used as LPMx.5 wake-up event. 0 No time event occurred. 1 Time event occurred.
RTCAIFG	Bit 1	Real-time clock alarm interrupt flag. In modules supporting LPMx.5 this interrupt can be used as LPMx.5 wake-up event. 0 No time event occurred. 1 Time event occurred.
RTCRDYIFG	Bit 0	Real-time clock ready interrupt flag 0 RTC cannot be read safely. 1 RTC can be read safely.

(1) The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits itself; therefore, reconfiguration after wake-up from LPMx.5 before clearing LOCKLPM5 is required.

RTCCTL1, Real-Time Clock Control Register 1

7	6	5	4	3	2	1	0
RTCBCD	**RTCHOLD**[1]	**Reserved**	**RTCRDY**	**Reserved**	**Reserved**	**RTCTEVx**[1]	
rw-(0)	rw-(1)	r1	r-(1)	r0	r0	rw-(0)	rw-(0)

RTCBCD Bit 7 Real-time clock BCD select. Selects BCD counting for real-time clock.

- 0 Binary/hexadecimal code selected
- 1 BCD Binary coded decimal (BCD) code selected

RTCHOLD Bit 6 Real-time clock hold

- 0 Real-time clock is operational.
- 1 The calendar is stopped as well as the prescale counters, RT0PS and RT1PS.

Reserved Bit 5 Reserved. Always read as 1.

RTCRDY Bit 4 Real-time clock ready

- 0 RTC time values in transition
- 1 RTC time values safe for reading. This bit indicates when the real-time clock time values are safe for reading.

Reserved Bits 3-2 Reserved. Always read as 0.

RTCTEVx Bits 1-0 Real-time clock time event

RTCTEVx	Interrupt Interval
00	Minute changed
01	Hour changed
10	Every day at midnight (00:00)
11	Every day at noon (12:00)

(1) The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits itself; therefore, reconfiguration after wake-up from LPMx.5 before clearing LOCKLPM5 is required.

Real-Time Clock Control 2 Register (RTCCTL2)

7	6	5	4	3	2	1	0
RTCCALS	**Reserved**	**RTCCALx**					
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.
RTCCALx	Bits 5-0	Real-time clock calibration. Each LSB represents approximately +4-ppm (RTCCALS = 1) or a –2-ppm (RTCCALS = 0) adjustment in frequency.

Real-Time Clock Control 3 Register (RTCCTL3)

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

Reserved	Bits 7-2	Reserved. Always read as 0.
RTCCALFx	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 don't care. 00 No frequency output to RTCCLK pin 01 512 Hz 10 256 Hz 11 1 Hz

Real-Time Clock Seconds Register (RTCSEC) – 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

Real-Time Clock Seconds Register (RTCSEC) – 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

Real-Time Clock Minutes Register (RTCMIN) – 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

Real-Time Clock Minutes Register (RTCMIN) – 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

Real-Time Clock Hours Register (RTCHOUR) – 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

Real-Time Clock Hours Register (RTCHOUR) – 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

Real-Time Clock Day of Week Register (RTCDOW)

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

Real-Time Clock Day of Month Register (RTCDAY) – 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

Real-Time Clock Day of Month Register (RTCDAY) – 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

Real-Time Clock Month Register (RTCMON) – 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

Real-Time Clock Month Register (RTCMON) – 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

Real-Time Clock Year Register (RTCYEAR) – Hexadecimal Format

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**Year – high byte of 0 to 4095**			
r-0	r-0	r-0	r-0	rw	rw	rw	rw
7	6	5	4	3	2	1	0
Year – low byte of 0 to 4095							
rw	rw	rw	rw	rw	rw	rw	rw

Real-Time Clock Year Register (RTCYEAR) – BCD Format

15	14	13	12	11	10	9	8
0	**Century – high digit (0 to 4)**			**Century – low digit (0 to 9)**			
r-0	rw	rw	rw	rw	rw	rw	rw
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

Real-Time Clock Minutes Alarm Register (RTCAMIN) – Hexadecimal Format

7	6	5	4	3	2	1	0
AE	**0**	**Minutes (0 to 59)**					
rw	r-0	rw	rw	rw	rw	rw	rw

Real-Time Clock Minutes Alarm Register (RTCAMIN) – BCD Format

7	6	5	4	3	2	1	0
AE	**Minutes – high digit (0 to 5)**			**Minutes – low digit (0 to 9)**			
rw	rw	rw	rw	rw	rw	rw	rw

Real-Time Clock Hours Alarm Register (RTCAHOUR) – Hexadecimal Format

7	6	5	4	3	2	1	0
AE	**0**	**0**	**Hours (0 to 24)**				
rw	r-0	r-0	rw	rw	rw	rw	rw

Real-Time Clock Hours Alarm Register (RTCAHOUR) – 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	r-0	rw	rw	rw	rw	rw	rw

Real-Time Clock Day of Week Alarm Register (RTCADOW)

7	6	5	4	3	2	1	0
AE	**0**	**0**	**0**	**0**	**Day of week (0 to 6)**		
rw	r-0	r-0	r-0	r-0	rw	rw	rw

Real-Time Clock Day of Month Alarm Register (RTCADAY) – Hexadecimal Format

7	6	5	4	3	2	1	0
AE	**0**	**0**	**Day of month (1 to 28, 29, 30, 31)**				
rw	r-0	r-0	rw	rw	rw	rw	rw

Real-Time Clock Day of Month Alarm Register (RTCADAY) – 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	r-0	rw	rw	rw	rw	rw	rw

Real-Time Clock Prescale Timer 0 Control Register (RTCPS0CTL)

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
Reserved			RT0IPx[1]			RT0PSIE	RT0PSIFG
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-(0)

Reserved	Bit 15-5	Reserved. Always read as 0.
RT0IPx	Bits 4-2	Prescale timer 0 interrupt interval
	000	/2
	001	/4
	010	/8
	011	/16
	100	/32
	101	/64
	110	/128
	111	/256
RT0PSIE	Bit 1	Prescale timer 0 interrupt enable
	0	Interrupt not enabled
	1	Interrupt enabled
RT0PSIFG	Bit 0	Prescale timer 0 interrupt flag
	0	No time event occurred.
	1	Time event occurred.

(1) The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits itself; therefore, reconfiguration after wake-up from LPMx.5 before clearing LOCKLPM5 is required.

Real-Time Clock Prescale Timer 1 Control Register (RTCPS1CTL)

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
Reserved			RT1IPx[1]			RT1PSIE[1]	RT1PSIFG
r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-(0)

Reserved	Bits 15-5	Reserved. Always read as 0.
RT1IPx	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. (LPMx.5 wake-up enabled.)
RT1PSIFG	Bit 0	Prescale timer 1 interrupt flag. In modules supporting LPMx.5 this interrupt can be used as LPMx.5 wake-up event.
	0	No time event occurred.
	1	Time event occurred.

(1) The configuration of these bits is retained during LPMx.5 until LOCKLPM5 is cleared, but not the register bits itself; therefore, reconfiguration after wake-up from LPMx.5 before clearing LOCKLPM5 is required.

Real-Time Clock Prescale Timer 0 Counter Register (RTCPS0)

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

Real-Time Clock Prescale Timer 1 Counter Register (RTCPS1)

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

Real-Time Clock Interrupt Vector Register (RTCIV)

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**	**RTCIVx**			0
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

RTCIVx Bits 15-0 Real-time clock 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	RTC oscillator failure	RTCOFIFG	
0Eh	Reserved		Lowest

Binary-to-BCD Conversion Register (BIN2BCD)

15	14	13	12	11	10	9	8
BIN2BCDx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
BIN2BCDx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

BIN2BCDx Bits 15-0 Read: 16-bit BCD conversion of previously written 12-bit binary number
Write: 12-bit binary number to be converted

BCD-to-Binary Conversion Register (BCD2BIN)

15	14	13	12	11	10	9	8
BCD2BINx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
BCD2BINx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

BCD2BINx	Bits 15-0	Read:	12-bit binary conversion of previously written 16-bit BCD number
		Write:	16-bit BCD number to be converted

32-Bit Hardware Multiplier (MPY32)

This chapter describes the 32-bit hardware multiplier (MPY32). The MPY32 module is implemented in all devices.

Topic **Page**

14.1 32-Bit Hardware Multiplier (MPY32) Introduction

The MPY32 is a peripheral and is not part of the 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 MPY32 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 MPY32 block diagram is shown in Figure 14-1.

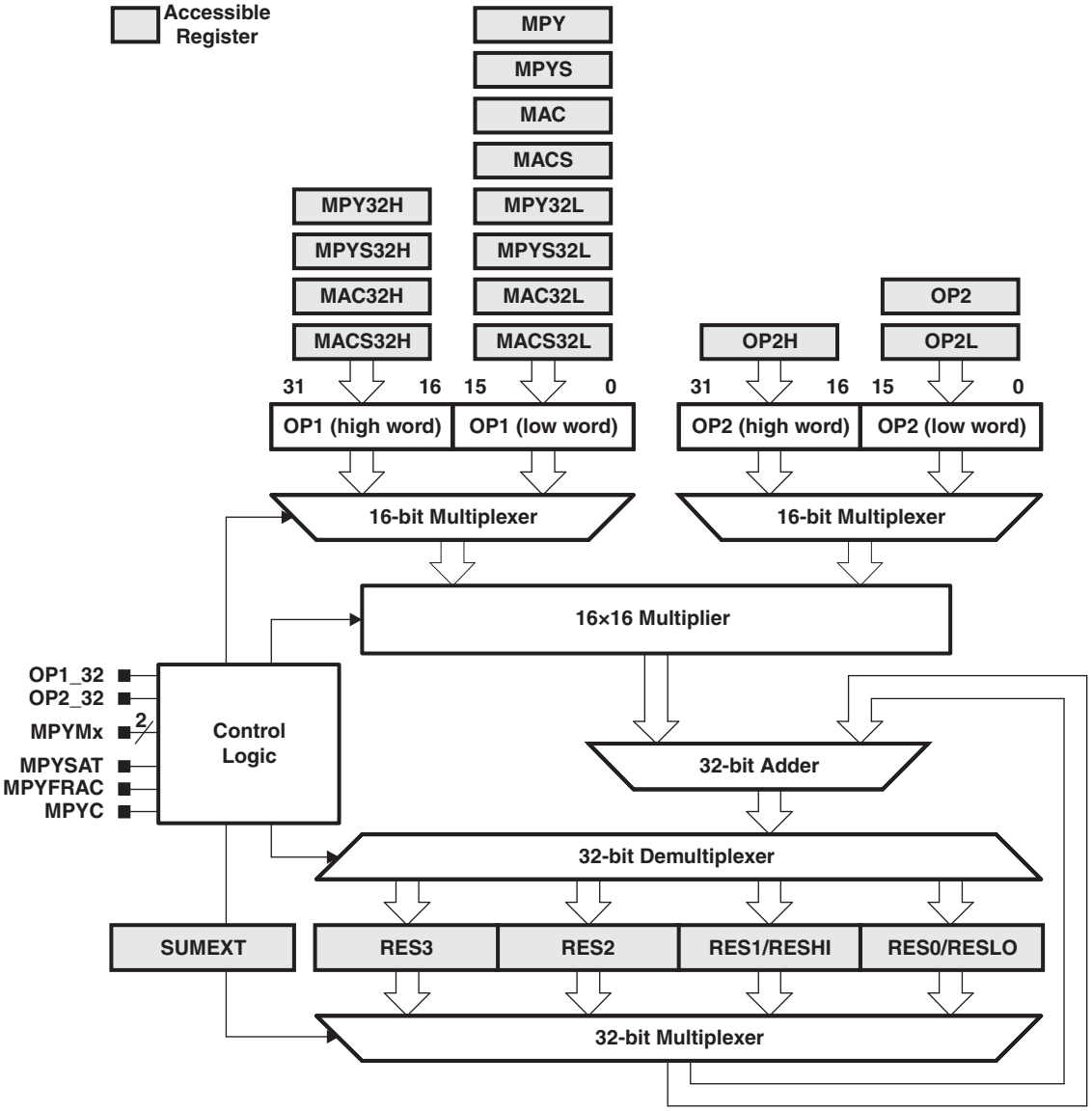


Figure 14-1. MPY32 Block Diagram

14.2 MPY32 Operation

The MPY32 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 14-1 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 must 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 14-1. 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

14.2.1 Operand Registers

Operand one (OP1) has 12 registers (see Table 14-2) 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 ignores MPY32H and assumes 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 rewrite the OP1 value to perform the operations.

Table 14-2. OP1 Registers

OP1 Register	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 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 14-3. OP2 Registers

OP2 Register	Operation
OP2	Start multiplication with 16-bit-wide OP2 – operand bits 0 up to 15
OP2L	Start multiplication with 32-bit-wide OP2 – operand bits 0 up to 15
OP2H	Continue multiplication with 32-bit-wide 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 automatically causes 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 as 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 renders any results invalid that are not ready at the time the new operand(s) are changed. Writing OP2 or OP2L aborts any ongoing calculation and starts a new operation. Results that are not ready at that time are also invalid 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, the complete 64-bit result should always be ready.

See Table 14-1 for how many CPU cycles are needed until a certain result register is ready and valid for each of the different modes.

14.2.2 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 register SUMEXT contents depend on the multiply operation and are listed in Table 14-4. 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 14-4 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 14-4. SUMEXT 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

14.2.2.1 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 (that is, using only 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 from the sign reflected by the SUMEXT register, an overflow or underflow occurred. User software must handle these conditions appropriately.

14.2.3 Software Examples

Examples for all multiplier modes follow. All 8×8 modes use the absolute address for the registers, because the assembler does 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 automatically causes 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
```

14.2.4 Fractional Numbers

The MPY32 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 14-2 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 (MSB) 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.

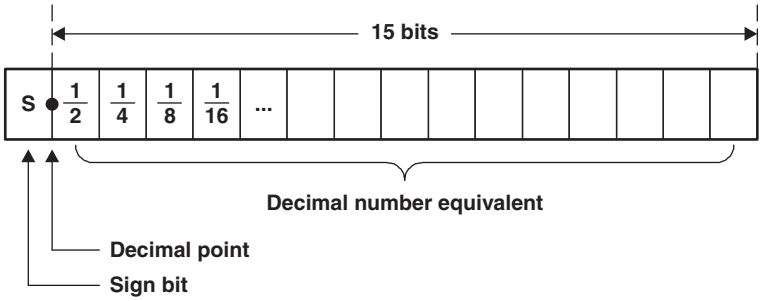


Figure 14-2. Q15 Format Representation

The range can be increased by shifting the decimal point to the right as shown in Figure 14-3. The signed Q14 format with 16 bits gives a range from –2.0 to 1.999938965 ≠ 2.0.

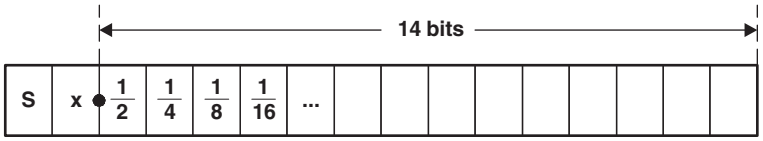


Figure 14-3. Q14 Format Representation

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.

14.2.4.1 Fractional Number Mode

Multiplying two fractional numbers using the default multiplication mode with MPYFRAC = 0 and MPYSAT = 0 gives a result with two 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 one bit, resulting in the final Q formatted result. This allows user software to switch between reading both the shifted (fractional) and the unshifted 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 nonfractional 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 14-5. 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

14.2.4.2 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 nonsaturated result.

With 16×16 operations, the saturation mode only applies to the least significant 32 bits, that is, 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 leads to unpredictable results.

With 32×32, 16×32, and 32×16 operations, the saturated result can only be calculated when RES3 is ready.

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 14-6. 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 14-4 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 MSB of the result. Secondly, if the fractional mode is enabled, it depends also on the two MSBs of the unshift result, that is, the result that is read with fractional mode disabled.

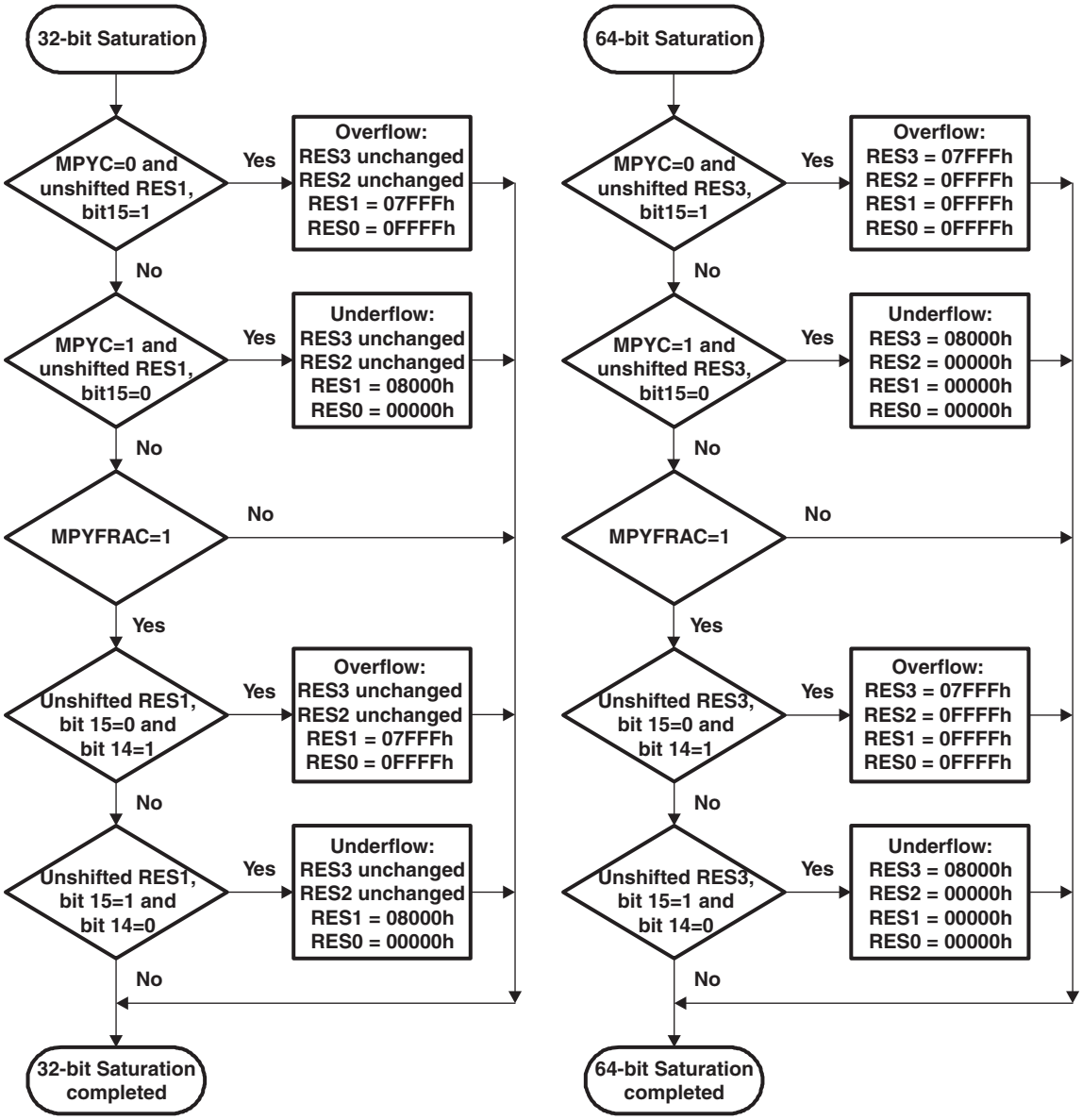


Figure 14-4. Saturation Flow Chart

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. Because the MSB of the unmodified result RES1 is 1 and MPYC = 0, the result is saturated according Figure 14-4.

NOTE: Validity of saturated result

The saturated result is only valid if the registers RES0 to RES3, the size of OP1 and OP2, 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 erroneously saturates the result.

14.2.5 Putting It All Together

Figure 14-5 shows the complete multiplication flow, depending on the various selectable modes for the MPY32 module.

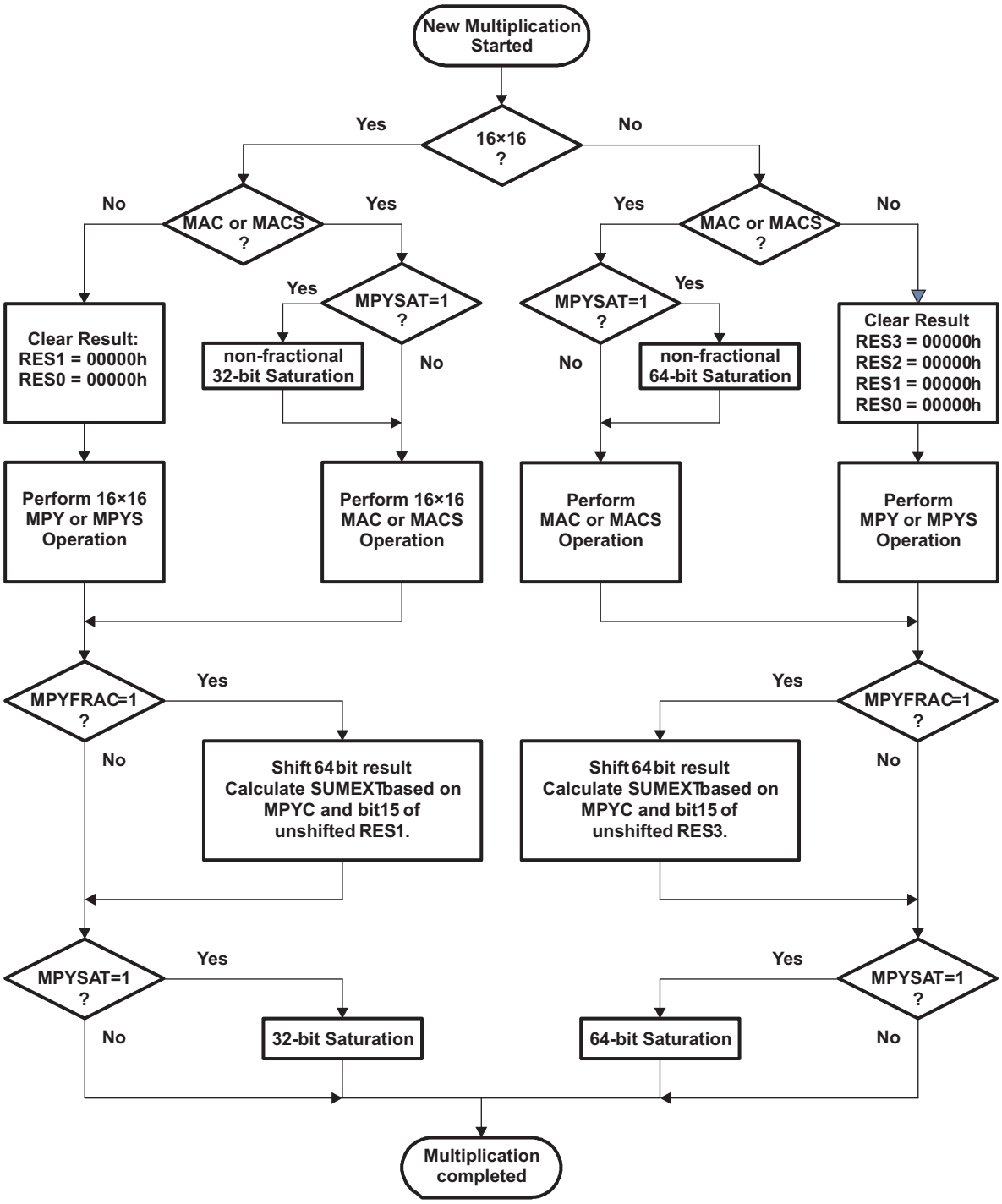


Figure 14-5. Multiplication Flow Chart

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 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 MSB 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 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.

14.2.6 Indirect Addressing of Result Registers

When using indirect or indirect autoincrement addressing mode to access the result registers and the multiplier requires three cycles until result availability according to Table 14-1, 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
```

14.2.7 Using Interrupts

If an interrupt occurs after writing OP, 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 MPY32, do not use the MPY32 in interrupt service routines, or use the save and restore functionality of the MPY32.

```
; 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
```

14.2.7.1 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. Because 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
```

14.2.8 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' (see the *DMA Controller* chapter for details).

14.3 MPY32 Registers

MPY32 registers are listed inTable 14-7. The base address can be found in the device-specific data sheet. The address offsets are listed inTable 14-7.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 14-7. MPY32 Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
16-bit operand one – multiply	MPY	Read/write	Word	00h	Undefined
	MPY_L	Read/write	Byte	00h	Undefined
	MPY_H	Read/write	Byte	01h	Undefined
8-bit operand one – multiply	MPY_B	Read/write	Byte	00h	Undefined
16-bit operand one – signed multiply	MPYS	Read/write	Word	02h	Undefined
	MPYS_L	Read/write	Byte	02h	Undefined
	MPYS_H	Read/write	Byte	03h	Undefined
8-bit operand one – signed multiply	MPYS_B	Read/write	Byte	02h	Undefined
16-bit operand one – multiply accumulate	MAC	Read/write	Word	04h	Undefined
	MAC_L	Read/write	Byte	04h	Undefined
	MAC_H	Read/write	Byte	05h	Undefined
8-bit operand one – multiply accumulate	MAC_B	Read/write	Byte	04h	Undefined
16-bit operand one – signed multiply accumulate	MACS	Read/write	Word	06h	Undefined
	MACS_L	Read/write	Byte	06h	Undefined
	MACS_H	Read/write	Byte	07h	Undefined
8-bit operand one – signed multiply accumulate	MACS_B	Read/write	Byte	06h	Undefined
16-bit operand two	OP2	Read/write	Word	08h	Undefined
	OP2_L	Read/write	Byte	08h	Undefined
	OP2_H	Read/write	Byte	09h	Undefined
8-bit operand two	OP2_B	Read/write	Byte	08h	Undefined
16x16-bit result low word	RESLO	Read/write	Word	0Ah	Undefined
	RESLO_L	Read/write	Byte	0Ah	Undefined
16x16-bit result high word	RESHI	Read/write	Word	0Ch	Undefined
16x16-bit sum extension register	SUMEXT	Read	Word	0Eh	Undefined
32-bit operand 1 – multiply – low word	MPY32L	Read/write	Word	10h	Undefined
	MPY32L_L	Read/write	Byte	10h	Undefined
	MPY32L_H	Read/write	Byte	11h	Undefined
32-bit operand 1 – multiply – high word	MPY32H	Read/write	Word	12h	Undefined
	MPY32H_L	Read/write	Byte	12h	Undefined
	MPY32H_H	Read/write	Byte	13h	Undefined
24-bit operand 1 – multiply – high byte	MPY32H_B	Read/write	Byte	12h	Undefined
32-bit operand 1 – signed multiply – low word	MPYS32L	Read/write	Word	14h	Undefined
	MPYS32L_L	Read/write	Byte	14h	Undefined
	MPYS32L_H	Read/write	Byte	15h	Undefined
32-bit operand 1 – signed multiply – high word	MPYS32H	Read/write	Word	16h	Undefined
	MPYS32H_L	Read/write	Byte	16h	Undefined
	MPYS32H_H	Read/write	Byte	17h	Undefined
24-bit operand 1 – signed multiply – high byte	MPYS32H_B	Read/write	Byte	16h	Undefined
32-bit operand 1 – multiply accumulate – low word	MAC32L	Read/write	Word	18h	Undefined

Table 14-7. MPY32 Registers (continued)

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
	MAC32L_L	Read/write	Byte	18h	Undefined
	MAC32L_H	Read/write	Byte	19h	Undefined
32-bit operand 1 – multiply accumulate – high word	MAC32H	Read/write	Word	1Ah	Undefined
	MAC32H_L	Read/write	Byte	1Ah	Undefined
	MAC32H_H	Read/write	Byte	1Bh	Undefined
24-bit operand 1 – multiply accumulate – high byte	MAC32H_B	Read/write	Byte	1Ah	Undefined
32-bit operand 1 – signed multiply accumulate – low word	MACS32L	Read/write	Word	1Ch	Undefined
	MACS32L_L	Read/write	Byte	1Ch	Undefined
	MACS32L_H	Read/write	Byte	1Dh	Undefined
32-bit operand 1 – signed multiply accumulate – high word	MACS32H	Read/write	Word	1Eh	Undefined
	MACS32H_L	Read/write	Byte	1Eh	Undefined
	MACS32H_H	Read/write	Byte	1Fh	Undefined
24-bit operand 1 – signed multiply accumulate – high byte	MACS32H_B	Read/write	Byte	1Eh	Undefined
32-bit operand 2 – low word	OP2L	Read/write	Word	20h	Undefined
	OP2L_L	Read/write	Byte	20h	Undefined
	OP2L_H	Read/write	Byte	21h	Undefined
32-bit operand 2 – high word	OP2H	Read/write	Word	22h	Undefined
	OP2H_L	Read/write	Byte	22h	Undefined
	OP2H_H	Read/write	Byte	23h	Undefined
24-bit operand 2 – high byte	OP2H_B	Read/write	Byte	22h	Undefined
32x32-bit result 0 – least significant word	RES0	Read/write	Word	24h	Undefined
	RES0_L	Read/write	Byte	24h	Undefined
32x32-bit result 1	RES1	Read/write	Word	26h	Undefined
32x32-bit result 2	RES2	Read/write	Word	28h	Undefined
32x32-bit result 3 – most significant word	RES3	Read/write	Word	2Ah	Undefined
MPY32 control register 0	MPY32CTL0	Read/write	Word	2Ch	Undefined
	MPY32CTL0_L	Read/write	Byte	2Ch	Undefined
	MPY32CTL0_H	Read/write	Byte	2Dh	00h

The registers listed in Table 14-8 are treated equally.

Table 14-8. Alternative Registers

Register	Alternative 1	Alternative 2
16-bit operand one – multiply	MPY	MPY32L
8-bit operand one – multiply	MPY_B or MPY_L	MPY32L_B or MPY32L_L
16-bit operand one – signed multiply	MPYS	MPYS32L
8-bit operand one – signed multiply	MPYS_B or MPYS_L	MPYS32L_B or MPYS32L_L
16-bit operand one – multiply accumulate	MAC	MAC32L
8-bit operand one – multiply accumulate	MAC_B or MAC_L	MAC32L_B or MAC32L_L
16-bit operand one – signed multiply accumulate	MACS	MACS32L
8-bit operand one – signed multiply accumulate	MACS_B or MACS_L	MACS32L_B or MACS32L_L
16x16-bit result low word	RESLO	RES0
16x16-bit result high word	RESHI	RES1

32-Bit Hardware Multiplier Control 0 Register (MPY32CTL0)

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

The REF module is a general purpose reference system that is used to generate voltage references required for other subsystems available on a given device such as digital-to-analog converters, analog-to-digital converters, or comparators. This chapter describes the REF module.

15.1 REF Introduction

The reference module (REF) is responsible for generation of all critical reference voltages that can be used by various analog peripherals in a given device. These include, but are not necessarily limited to, the ADC12_B and COMP_B modules dependent upon the particular device. The heart of the reference system is the bandgap from which all other references are derived by unity or non-inverting gain stages. The REF module consists of the bandgap and a non-inverting buffer stage which generates three voltage reference available in the system, namely 1.5 V, 2.0 V, and 2.5 V. In addition, when requested, a buffered bandgap voltage is also available.

Features of the REF include:

- Centralized, factory trimmed bandgap with excellent PSRR, temperature coefficient, and accuracy
- 1.5 V, 2.0 V, 2.5 V user selectable internal references
- Buffered bandgap voltage available to rest of system
- Power saving features

The block diagram of the REF module is shown in Figure 15-1.

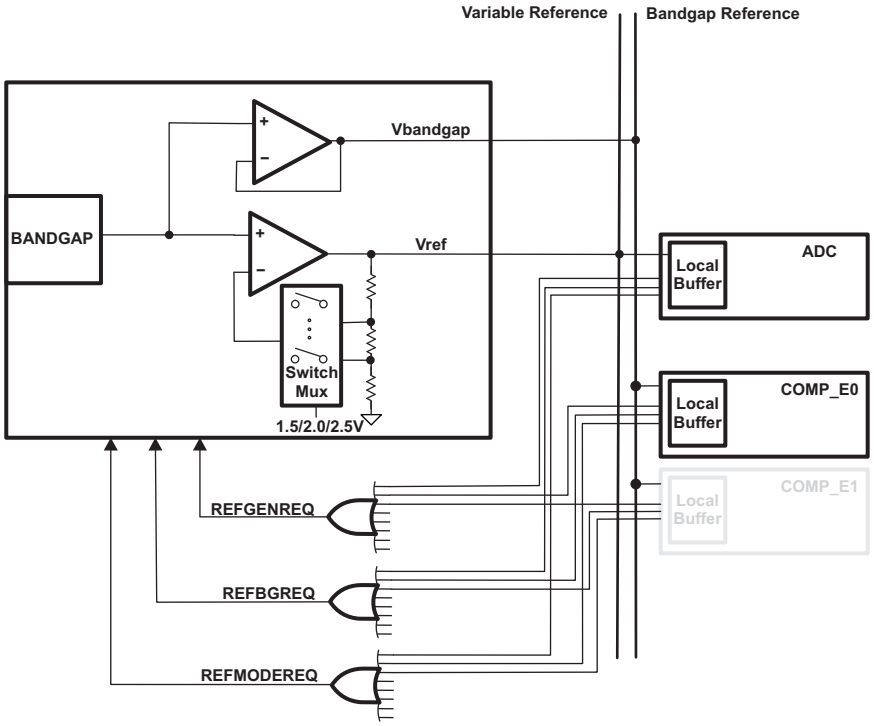


Figure 15-1. REF Block Diagram

15.2 Principle of Operation

The REF module provides all of the necessary voltage references that are used by various peripheral modules throughout the system.

The high-performance bandgap has very good accuracy (factory trimmed), low temperature coefficient, and high PSRR while operating at low power. The bandgap voltage is used to generate three voltages via a noninverting amplifier stage, namely 1.5 V, 2.0 V, and 2.5 V. One voltage can be selected at a time. One output is the variable reference line that can be used throughout the system. The variable reference line provides either 1.5 V, 2.0 V, or 2.5 V to the rest of the system. A second output of the REF module provides a buffered bandgap reference line that can be used by any module throughout the system. The REF module includes the temperature sensor circuitry. The temperature sensor is used by an ADC to measure a voltage proportional to temperature.

15.2.1 Low Power Operation

The REF module is capable of supporting low power applications such as LCD generation. Many of these applications do not require a very accurate reference, compared to data conversion, yet power is of prime concern. To support these kinds of applications, the bandgap is capable of being used in a sampled mode. This reduces the average power of the bandgap circuitry significantly, at the cost of accuracy. When not in sampled mode, the bandgap is in static mode. Its power is at its highest, but so is its accuracy.

Modules automatically can request static mode or sampled mode via their own individual request lines. In this way, the particular module determines what mode is appropriate for its proper operation and performance. Any one active module that requests static mode causes all other modules to use static mode, regardless if another module is requesting sampled mode. In other words, static mode always has higher priority over sampled mode.

15.2.2 REFCTL

The REFCTL registers provide a way to control the reference system from one centralized set of registers. REFCTL is used to control the reference system.

Table Table 15-1summarizes the REFCTL bits and their effect on the REF module.

Table 15-1. REF Control of Reference System (REFMSTR = 1) (Default)

REF Register Setting	Function
REFON	Setting this bit enables the REF module which includes the bandgap, the bandgap bias circuitry, and the 1.5-/2.0-/2.5-V buffer. Setting this bit causes the REF module to remain enabled regardless if any module has requested it. Clearing this bits disables the REF module only when there are no pending requests for any reference voltage.
REFVSEL	Selects 1.5 V, 2.0 V, or 2.5 V to be present on the variable reference line when REFON = 1 or it is requested by any module.
REFTCOFF	Setting this bit disables the temperature sensor (when available) to conserve power.

15.2.3 Reference System Requests

There are three basic reference system requests that are used by the reference system. Each module can use these requests to obtain the proper response from the reference system. The three basic requests are REFGENREQ, REFBGREQ, and REFMODEREQ.

A reference request signal, REFGENREQ, is available as an input into the REFGEN subsystem. This signal represents a logical OR of individual requests coming from the various modules in the system that require a voltage reference to be available on the variable reference line. When a module requires a voltage reference, it asserts its corresponding REGFENREQ signal. Once the REFGENREQ is asserted, the REFGEN subsystem is enabled. After the specified settling time, the variable reference line voltage is stable and ready for use. The REFVSEL settings determine which voltage is generated on the variable reference line.

In addition to the REFGENREQ, a second reference request signal, REFBGREQ is available. The REFBGREQ signal represents a logical OR of requests coming from the various modules that require the bandgap reference line. Once the REFBGREQ is asserted, the bandgap, along with its bias circuitry and local buffer, is enabled if it is not already enabled by a prior request.

The REFMODEREQ request signal is available that configures the bandgap and its bias circuitry to operate in a sampled or static mode of operation. The REFMODEREQ signal basically represents a logical AND of individual requests coming from the various analog modules. In reality, a REFMODEREQ occurs only if a module's REFGENREQ or REFBGQ is also asserted, otherwise it is a do not care. When REFMODEREQ = 1, the bandgap operates in sampled mode. When a module asserts its corresponding REFMODEREQ signal, it is requesting that the bandgap operate in sampled mode. Because REMODEREQ is a logical AND of all individual requests, any modules requesting static mode cause the bandgap to operate in static mode. The BGMODE bit can be used as an indicator of static or sampled mode of operation.

15.2.3.1 REFBGACT, REFGENACT, REFGENBUSY

Any module that is using the variable reference line causes REFGENACT to be set inside the REFCTL register. This bit is read only and indicates to the user that the REFGEN is active or off. Similarly, the REFBGACT is active any time one or more modules is actively utilizing the bandgap reference line and indicates to the user that the REFBG is active or off.

The REFGENBUSY signal, when asserted, indicates that a module is using the reference and cannot have any of it settings changed. For example, during an active ADC10_B conversion, the reference voltage level should not be changed. REFGENBUSY is asserted when there is an active ADC10_B conversion (ENC = 1) **or when the DAC12_A is actively converting (DAC12AMPx > 1 and DAC12SREFx = 0)**. REFGENBUSY when asserted, write protects the REFCTL register. This prevents the reference from being disabled or its level changed during any active conversion.

15.2.3.2 ADC10_B

For devices that contain an ADC10_B module, if the ADC is not sampling or converting but the REFON bit is set the REF module remains on.

15.3 REF Registers

The REF registers are listed in Table 15-2. The base address can be found in the device specific datasheet. The address offset is listed in Table 15-2.

NOTE: All registers have word or byte register access. For a generic register *ANYREG*, the suffix "_L" (*ANYREG_L*) refers to the lower byte of the register (bits 0 through 7). The suffix "_H" (*ANYREG_H*) refers to the upper byte of the register (bits 8 through 15).

Table 15-2. REF Registers

Register	Short Form	Register Type	Access	Address Offset	Initial State
REFCTL0	REFCTL0	Read/write	Word	00h	0080h
	REFCTL0_L	Read/write	Byte	00h	80h
	REFCTL0_H	Read/write	Byte	01h	00h

REFCTL0, REF Control Register 0

15	14	13	12	11	10	9	8
Reserved	**Reserved**	**Reserved**	**Reserved**	**BGMODE**	**REFGENBUSY**	**REFBGACT**	**REFGENACT**
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r-(0)
7	6	5	4	3	2	1	0
Reserved	**Reserved**	**REFVSEL**		**REFTCOFF**	**Reserved**	**Reserved**	**REFON**
r0	r0	rw-(0)	rw-(0)	rw-(0)	r0	r0	rw-(0)

Modifiable only when REFGENBUSY = 0

Reserved	Bits 15-12	Reserved. Always reads back 0.	
BGMODE	Bit 11	Bandgap mode. Read only.	
		0	Static mode.
		1	Sampled mode.
REFGENBUSY	Bit 10	Reference generator busy. Read only.	
		0	Reference generator not busy.
		1	Reference generator busy.
REFBGACT	Bit 9	Reference bandgap active. Read only.	
		0	Reference bandgap buffer not active.
		1	Reference bandgap buffer active.
REFGENACT	Bit 8	Reference generator active. Read only.	
		0	Reference generator not active.
		1	Reference generator active.
Reserved	Bit 7-6	Reserved. Always reads back 0.	
REFVSEL	Bits 5-4	Reference voltage level select.	
		0 0	1.5 V available when reference requested or REFON = 1
		0 1	2.0 V available when reference requested or REFON = 1
		1 x	2.5 V available when reference requested or REFON = 1
REFTCOFF	Bit 3	Temperature sensor disabled.	
		0	Temperature sensor enabled.
		1	Temperature sensor disabled to save power.
Reserved	Bit 2-1	Reserved. Always reads back 0.	
REFON	Bit 0	Reference enable.	
		0	Disables reference if no other reference requests are pending.
		1	Enables reference.

ADC10_B Module

The ADC10_B module is a high-performance 10-bit analog-to-digital converter (ADC). This chapter describes the operation of the ADC10_B module.

Topic **Page**

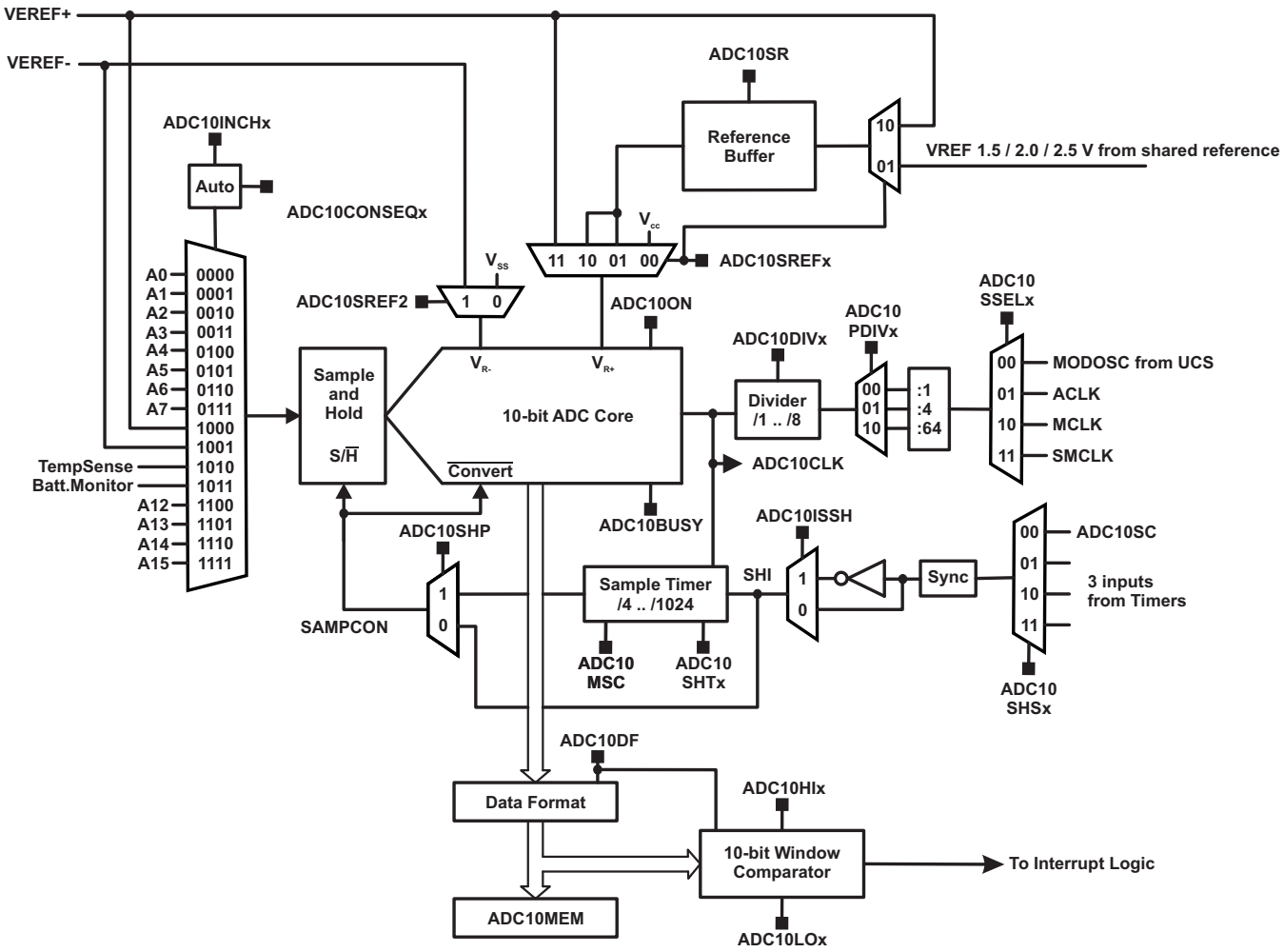
16.1 ADC10_B Introduction

The ADC10_B module supports fast 10-bit analog-to-digital conversions. The module implements a 10-bit SAR core together, sample select control and a window comparator.

ADC10_B features include:

- Greater than 200-ksps maximum conversion rate
- Monotonic 10-bit converter with no missing codes
- Sample-and-hold with programmable sampling periods controlled by software or timers
- Conversion initiation by software or different timers
- Software-selectable on chip reference using the REF module or external reference
- Twelve individually configurable external input channels
- Conversion channel for temperature sensor of the REF module
- Selectable conversion clock source
- Single-channel, repeat-single-channel, sequence, and repeat-sequence conversion modes
- Window comparator for low-power monitoring of input signals
- Interrupt vector register for fast decoding of six ADC interrupts (ADC10IFG0, ADC10TOVIFG, ADC10OVIFG, ADC10LOIFG, ADC10INIFG, ADC10HIIFG)

The block diagram of ADC10_B is shown in Figure 16-1. The on-chip generation is located in the reference module (see the device-specific data sheet).



A The MODOSC is part of the CS. See the CS chapter for more information.

B When using ADC10SHP = 0, no synchronisation of the trigger input is done.

Figure 16-1. ADC10_B Block Diagram

16.2 ADC10_B Operation

The ADC10_B module is configured with user software. The setup and operation of the ADC10_B is discussed in the following sections.

16.2.1 10-Bit ADC Core

The ADC core converts an analog input to its 10-bit digital representation and stores the result in the conversion register ADC10MEM0. The core uses two programmable/selectable voltage levels (V_{R+} and V_{R-}) to define the upper and lower limits of the conversion. The digital output (N_{ADC}) is full scale (03FFh) when the input signal is equal to or higher than V_{R+}, and zero when the input signal is equal to or lower than V_{R-}. The input channel and the reference voltage levels (V_{R+} and V_{R-}) are defined in the conversion-control memory. The conversion formula for the ADC result N_{ADC} is:

$$N_{ADC} = 1023 \times \frac{Vin - V_{R-}}{V_{R+} - V_{R-}}$$

The ADC10_B core is configured by the control registers ADC10CTL0, ADC10CTL1 and ADC10CTL2. The core is enabled with the ADC10ON bit. The ADC10_B can be turned off when not in use to save power. With few exceptions, the ADC10_B control bits can only be modified when ADC10ENC = 0. ADC10ENC must be set to 1 before any conversion can take place.

16.2.1.1 Conversion Clock Selection

The ADC10CLK is used both as the conversion clock and to generate the sampling period when the pulse sampling mode is selected. The ADC10_B source clock is selected using the ADC10SSELx bits. Possible ADC10CLK sources are SMCLK, MCLK, ACLK, and the MODOSC. The input clock can be divided from 1–512 using both the ADC10DIVx bits and the ADC10PDIVx bits.

MODOSC, generated internally in the UCS, is in the 5-MHz range, but varies with individual devices, supply voltage, and temperature. See the device-specific data sheet for the MODOSC specification.

The user must ensure that the clock chosen for ADC10CLK remains active until the end of a conversion. If the clock is removed during a conversion, the operation does not complete and any result is invalid.

16.2.2 ADC10_B Inputs and Multiplexer

The 12 external and 4 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 (see Figure 16-2). The input multiplexer is also a T-switch to minimize the coupling between channels. Channels that are not selected are isolated from the A/D and the intermediate node is connected to analog ground (AV_{SS}), so that the stray capacitance is grounded to eliminate crosstalk.

The ADC10_B uses the charge redistribution method. When the inputs are internally switched, the switching action may cause transients on the input signal. These transients decay and settle before causing errant conversion.

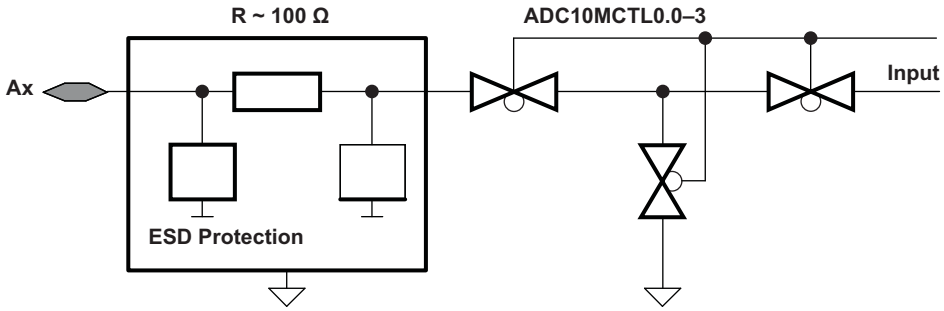


Figure 16-2. Analog Multiplexer

16.2.2.1 Analog Port Selection

The ADC10_B 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 part 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 ADC10_B function
```

16.2.3 Voltage Reference Generator

The ADC10_B module is designed to be used either with the on chip reference supplied by the REF module or an externally reference voltage supplied on external pins.

The on chip reference is capable of supplying 1.5 V, 2.0 V and 2.5 V. The exact working of this internal reference is explained in the corresponding section of this users guide.

External references may be supplied for V_{R+} and V_{R-} through pins V_{REF+}/Ve_{REF+} and V_{REF-}/Ve_{REF-} , respectively.

16.2.3.1 Internal Reference Low-Power Features

The on chip reference is designed for low-power applications. This reference includes a band-gap voltage source and a separate reference buffer both located in the REF-module. The current consumption of each is specified separately in the device-specific data sheet. The ADC10_B also contains an internal buffer for reference voltages. This buffer is automatically enabled, when the internal reference gets selected for V_{REF+} but it is also optionally available for Ve_{REF+}. The on chip reference from the REF-module needs to be enabled by software. Its settling time is ≤30 µs. See the REF module description for further information on the on-chip reference.

The reference buffer of the ADC10_B also has selectable speed versus power settings. When the maximum conversion rate is below 50 ksps, setting ADC10SR = 1 reduces the current consumption of the buffer approximately 50%.

16.2.4 Auto Power Down

The ADC10_B is designed for low-power applications. When the ADC10_B is not actively converting, the core is automatically disabled and automatically reenabled when needed. The MODOSC is also automatically enabled when needed and disabled when not needed.

16.2.5 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 ADC10SHSx bits and includes the following:

- ADC10SC bit
- Three timer outputs

The polarity of the SHI signal source can be inverted with the ADC10ISSH 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 11 ADC10CLK cycles in 10-bit resolution mode. One additional ADC10CLK is used for the window comparator. Two different sample-timing methods are defined by control bit ADC10SHP, extended sample mode, and pulse mode.

16.2.5.1 Extended Sample Mode

The extended sample mode is selected when ADC10SHP = 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 ADC10CLK (see Figure 16-3).

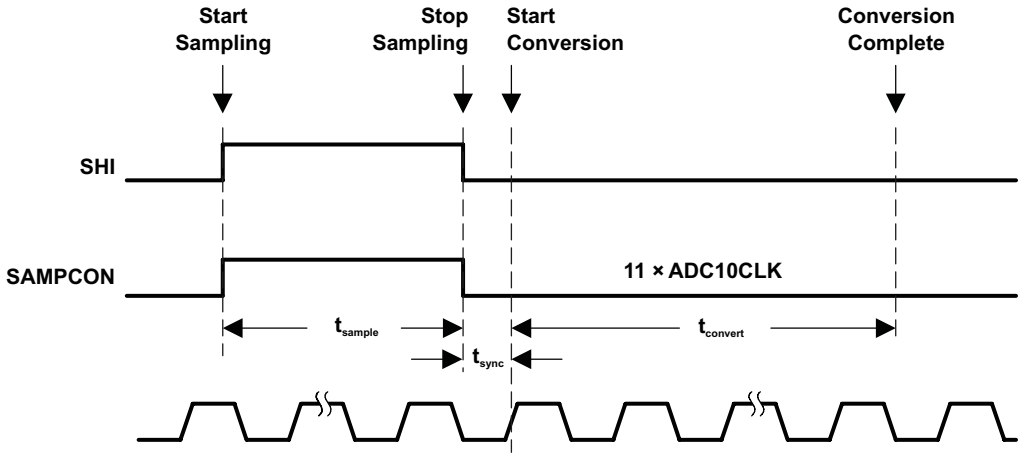


Figure 16-3. Extended Sample Mode

16.2.5.2 Pulse Sample Mode

The pulse sample mode is selected when ADC10SHP = 1. The SHI signal is used to trigger the sampling timer. The ADC10SHTx bits in ADC10CTL0 control the interval of the sampling timer that defines the SAMPCON sample period $t_{sample.}$ The sampling timer keeps SAMPCON high after synchronization with AD10CLK for a programmed interval t_{sample}. The total sampling time is t_{sample} plus t_{sync} (see Figure 16-4).

The ADC10SHTx bits select the sampling time in 4× multiples of ADC10CLK.

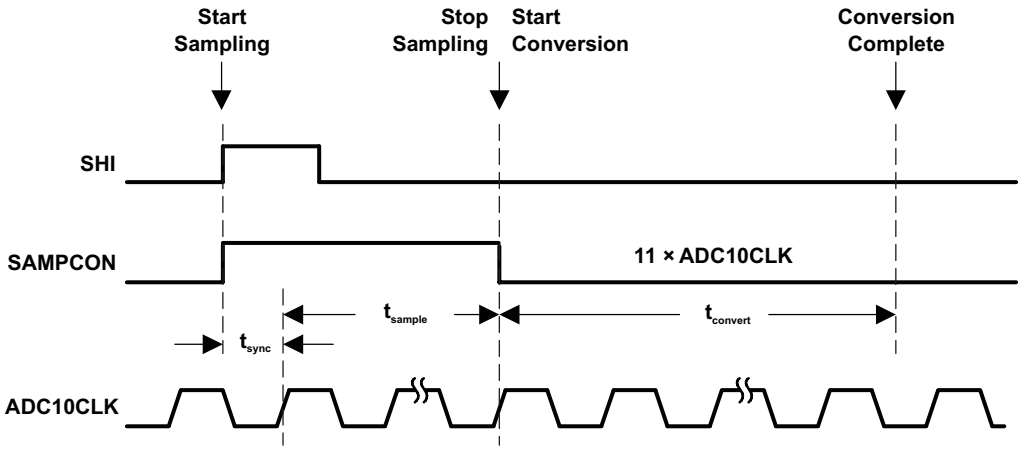


Figure 16-4. Pulse Sample Mode

16.2.5.3 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} (see Figure 16-5). An internal MUX-on input resistance R_I (see device specific datasheet) in series with capacitor C_I (see device specific datasheet) is seen by the source. The capacitor C_I voltage V_C must be charged to within one-half LSB of the source voltage V_S for an accurate 10-bit conversion.

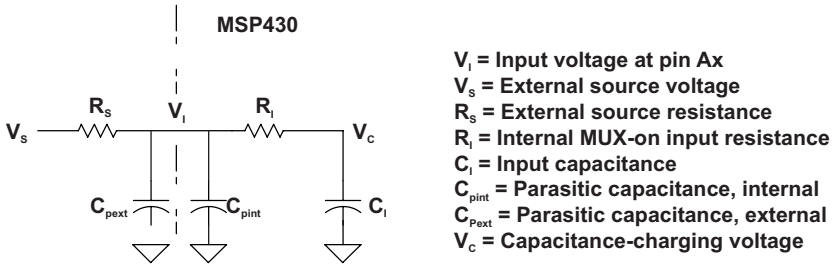


Figure 16-5. Analog Input Equivalent Circuit

The resistance of the source R_S and R_I affect t_{sample}. See the device specific datasheet for the t_{sample} limits.

16.2.6 Conversion Result

The conversion result is accessible using the ADC10MEM0 register independently of the conversion mode selected by the user. When a conversion result is written to ADC10MEM0, the ADC10IFG0 is set.

16.2.7 ADC10_B Conversion Modes

The ADC10_B has four operating modes selected by the CONSEQx bits as listed in Table 16-1.

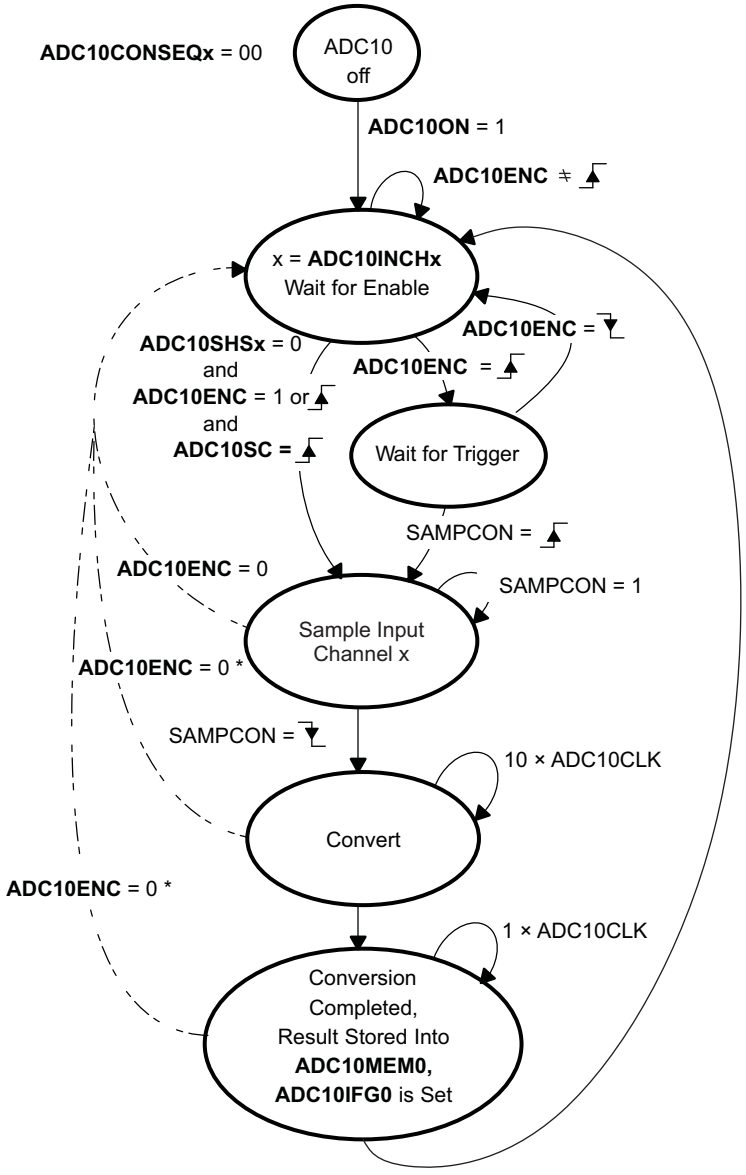
Table 16-1. Conversion Mode Summary

ADC10CONSEQx	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.

16.2.7.1 Single-Channel Single-Conversion Mode

A single channel selected by ADC10INCHx is sampled and converted once. The ADC result is written to ADC10MEM0. Figure 16-6 shows the flow of the single-channel single-conversion mode. When ADC10SC triggers a conversion, successive conversions can be triggered by the ADC10SC bit. When any other trigger source is used, ADC10ENC must be toggled between each conversion.

Resetting ADC10ON bit within a conversion causes the ADC10_B to go back into "ADC10 off" state. In this case, the value of the conversion register and the value of the interrupt flags is unpredictable.

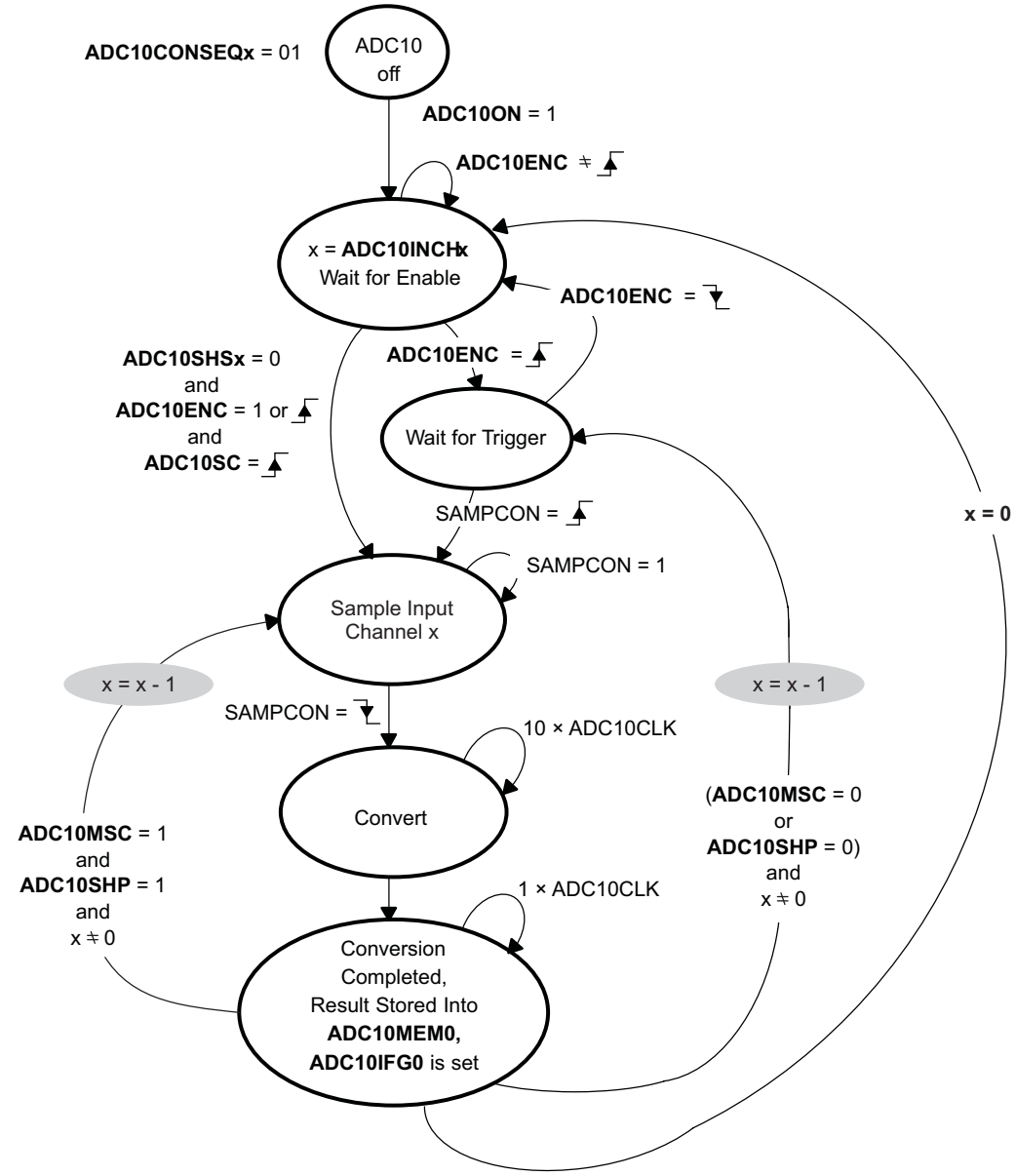


* Conversion result is unpredictable
x - pointer to the selected ADC10_A channel defined by **ADC10INCHx**
All bit- or registernames are marked with bold font, signals are noted in normal font

Figure 16-6. Single-Channel Single-Conversion Mode

16.2.7.2 Sequence-of-Channels Mode

A sequence of channels is sampled and converted once. The sequence begins with the channel selected by the ADC10INCHx bits and decrements to channel A0. Each ADC result is written to ADC10MEM0. The sequence stops after conversion of channel A0. Figure 16-7 shows the sequence-of-channels mode. When ADC10SC triggers a sequence, successive sequences can be triggered by the ADC10SC bit. When any other trigger source is used, ADC10ENC must be toggled between each sequence. As in all conversion modes, resetting ADC10ON bit within a conversion causes the ADC10_B to go back into "ADC10 off" state.

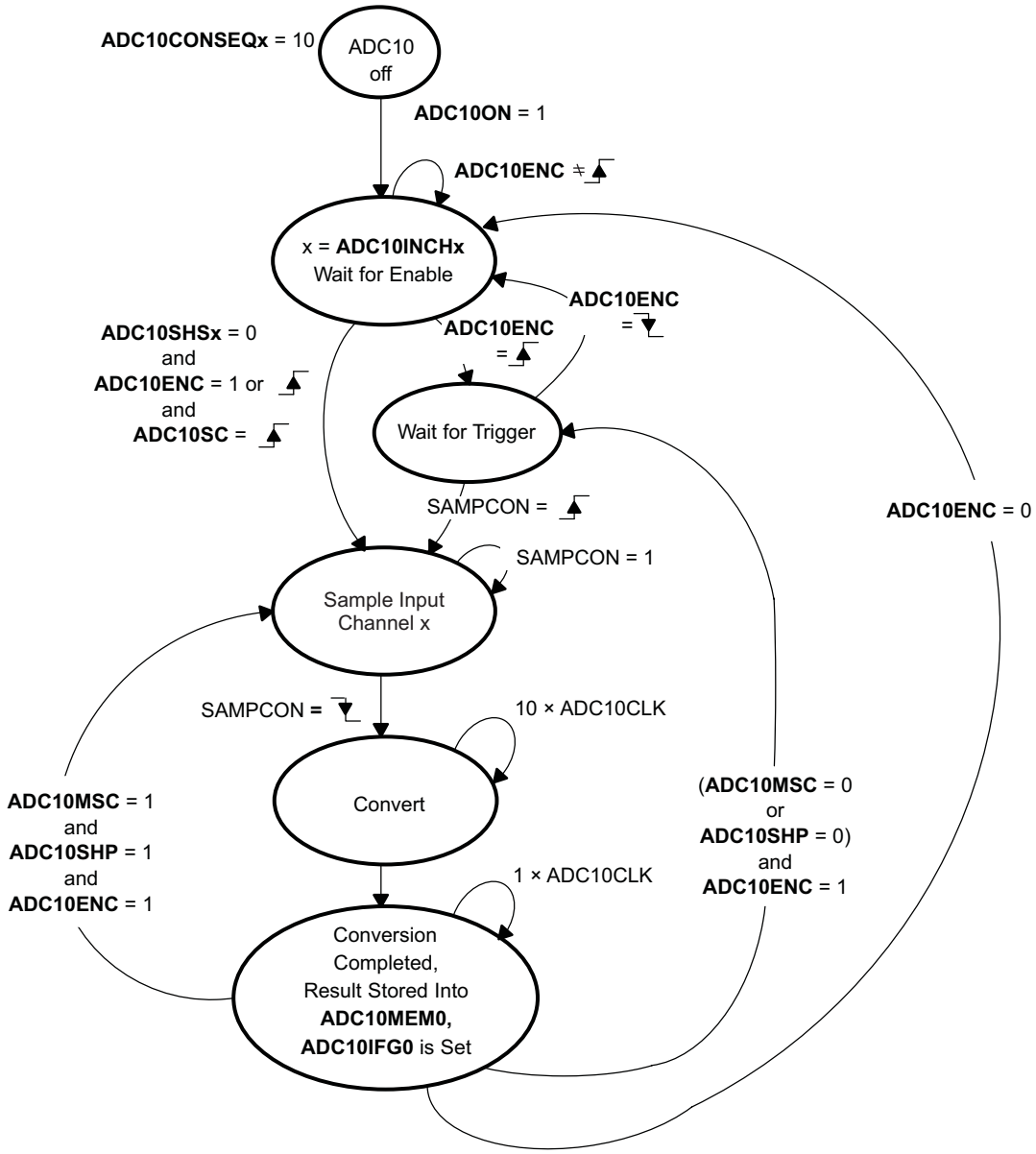


x - input channel Ax
All bit- or registernames are marked with bold font, signals are noted in normal font

Figure 16-7. Sequence-of-Channels Mode

16.2.7.3 Repeat-Single-Channel Mode

A single channel selected by ADC10INCHx is sampled and converted continuously. Each ADC result is written to ADC10MEM0. Figure 16-8 shows the repeat-single-channel mode.

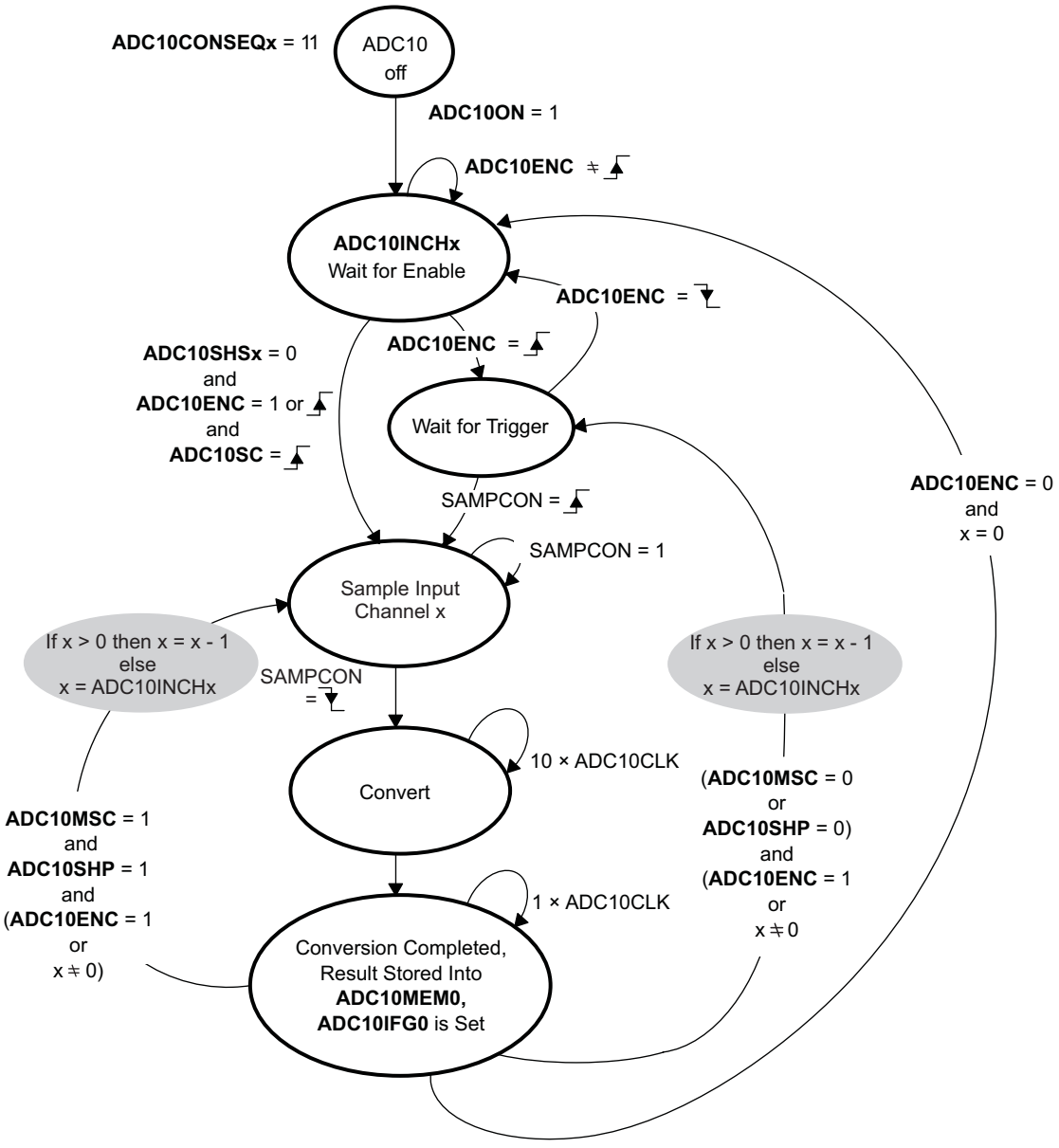


x - pointer to the selected ADC10_A channel defined by **ADC10INCHx**
All bit- or registernames are marked with bold font, signals are noted in normal font

Figure 16-8. Repeat-Single-Channel Mode

16.2.7.4 Repeat-Sequence-of-Channels Mode

A sequence of channels is sampled and converted repeatedly. The sequence begins with the channel selected by ADC10INCHx and decrements to channel A0. Each ADC result is written to ADC10MEM0. The sequence ends after conversion of channel A0, and the next trigger signal re-starts the sequence. Figure 16-9 shows the repeat-sequence-of-channels mode.



x - input channel Ax
All bit- or registernames are marked with bold font, signals are noted in normal font

Figure 16-9. Repeat-Sequence-of-Channels Mode

16.2.7.5 Using the Multiple Sample and Convert (ADC10MSC) Bit

To configure the converter to perform successive conversions automatically and as quickly as possible, a multiple sample and convert function is available. When ADC10MSC = 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 ADC10ENC bit is toggled in repeat-single-channel or repeated-sequence modes. The function of the ADC10ENC bit is unchanged when using the ADC10MSC bit.

16.2.7.6 Stopping Conversions

Stopping ADC10_B activity depends on the mode of operation. The recommended ways to stop an active conversion or conversion sequence are:

- Resetting ADC10ENC 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 ADC10ENC.
- Resetting ADC10ENC during repeat-single-channel operation stops the converter at the end of the current conversion.
- Resetting ADC10ENC during a sequence or repeat-sequence mode stops the converter at the end of the sequence.
- Any conversion mode may be stopped immediately by setting the CONSEQx = 0 and resetting the ADC10ENC bit. Conversion data are unreliable.

16.2.8 The Window Comparator

The window comparator allows to monitor analog signals without any CPU interaction. In the following list one can find the available Interrupt flags and the conditions, when they are asserted:

- The ADC10LO-Interrupt flag (ADC10LOIFG) gets set if the current result of the ADC10_B conversion is below the low threshold defined in register ADC10LO
- The ADC10HI-Interrupt flag (ADC10HIIFG) gets set if the current result of the ADC10_B conversion is greater than the high threshold defined in register ADC10HI
- The ADC10IN-Interrupt flag (ADC10INIFG) gets set if the current result of the ADC10_B conversion is in between the low threshold defined in register ADC10LO and the high threshold defined in ADC10HI

These Interrupts are generated independently of the conversion mode selected by the user.

The user always needs to ensure, that the values in the ADC10HI and ADC10LO registers are in the correct data format. If for example the binary data format is selected (ADC10DF = 0), then the thresholds in the threshold registers ADC10HI and ADC10LO also need to be entered binary coded. Changing the ADC10DF or the ADC10RES resets the threshold registers.

The interrupt flags need to be reset by the user software. The ADC10_B only updates the flags each time a new value is available in the ADC10MEM0. This update is only a set of the corresponding interrupt flag. When the user uses the window comparator flags he needs to ensure, that they get reset by software according to the application needs..

16.2.9 Using the Integrated Temperature Sensor

To use the on-chip temperature sensor, the user selects the analog input channel ADC10INCHx = 1010. Any other configuration is done as if an external channel was selected, including reference selection, conversion-mode selection, etc. The temperature sensor is located in the REF module of the device and needs to be activated by software.

The typical temperature sensor transfer function is shown in Figure 16-10. When using the temperature sensor, the sample period must be greater than 30 µs. The temperature sensor offset error can be large and may need to be calibrated for most applications (see the device-specific data sheet for parameters).

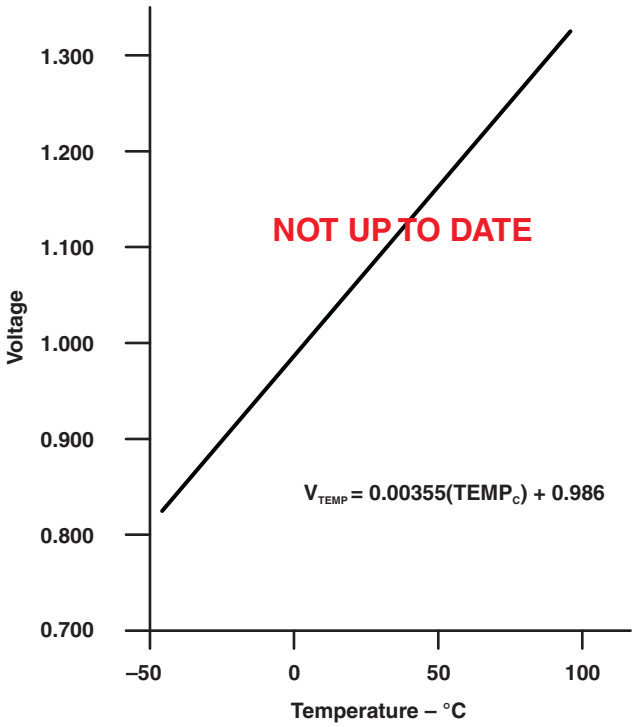


Figure 16-10. Typical Temperature Sensor Transfer Function

16.2.10 ADC10_B 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 ADC. The connections shown in Figure 16-11 prevent 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 recommended to achieve high accuracy.

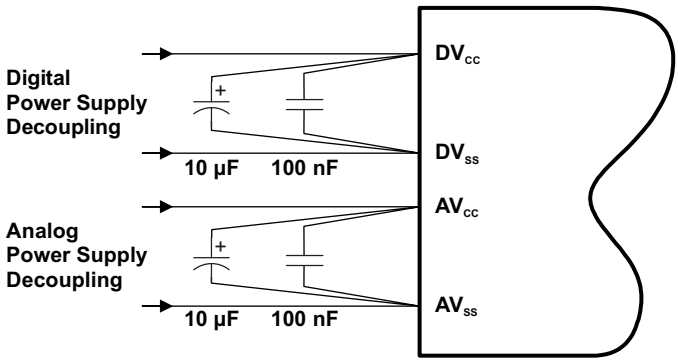


Figure 16-11. ADC10_B Grounding and Noise Considerations

16.2.11 ADC10_B Interrupts

The ADC10_B has 6 Interrupt sources:

- ADC10IFG0 : conversion ready Interrupt
- ADC10OVIFG : ADC10MEM0 overflow
- ADC10TOVIFG : ADC10_B conversion time overflow
- ADC10LOIFG, ADC10INIFG, ADC10HIIFG : window comparator Interrupt flags

The ADC10IFG0 bit is set when the ADC10MEM0 memory register is loaded with the conversion result. An Interrupt request is generated if ADC10IE0 bit and the GIE bit are set. The ADC10OV condition occurs when a conversion result is written to the ADC10MEM0 before its previous conversion result was read. The ADC10TOV condition is generated when another sample-and-conversion is requested before the current conversion is completed. The DMA is triggered after each conversion.

The window comparator Interrupt flags are set corresponding to the description in the Window Comparator section (see Section 16.2.8).

16.2.11.1 ADC10IV, Interrupt Vector Generator

All ADC10_B Interrupt sources are prioritized and combined to source a single Interrupt vector. The Interrupt vector register ADC10IV is used to determine which enabled ADC10_B Interrupt source requested an Interrupt.

The highest-priority enabled ADC10_B Interrupt generates a number in the ADC10IV register (see register description). This number can be evaluated or added to the program counter (PC) to automatically enter the appropriate software routine. Disabled ADC10_B Interrupts do not affect the ADC10IV value.

Read access of the ADC10IV register automatically resets the highest-pending Interrupt condition and flag. Only the ADC10IFG0 is not reset by this ADC10IV read access. ADC10IFG0 is automatically reset by reading the ADC10MEM0 register or may be reset with software.

Write access of the ADC10IV register clears all pending interrupt conditions and flags.

If another Interrupt is pending after servicing of an Interrupt, another Interrupt is generated. For example, if the ADC10OV, ADC10HIIFG and ADC10IFG0 Interrupts are pending when the Interrupt service routine accesses the ADC10IV register, the highest priority interrupt (ADC10OV Interrupt condition) is reset automatically. After the RETI instruction of the Interrupt service routine is executed, the ADC10HIIFG generates another Interrupt.

16.2.11.2 ADC10_B Interrupt Handling Software Example

The following software example shows the recommended use of the ADC10IV. The ADC10IV value is added to the PC to automatically jump to the appropriate routine.

- ADC10IFG0, ADC10TOV, and ADC10OV: 16 cycles

```
; Interrupt handler for ADC10_B.
INT_ADC10_B                             ; Enter Interrupt Service Routine
 ADD   &ADC10IV,PC                      ; Add offset to PC
 RETI                                   ; Vector  0: No Interrupt
 JMP   ADOV                             ; Vector  2: ADC10_B overflow
 JMP   ADTOV                            ; Vector  4: ADC10_B timing overflow
 JMP   ADHI                             ; Vector  6: ADC10_B window comparator high
Interrupt
 JMP   ADLO                             ; Vector  8: ADC10_B window comparator low
Interrupt
 JMP   ADIN                             ; Vector 10: ADC10_B window comparator in
Interrupt
;
; Handler for ADC10IFG0 starts here. No JMP required.
;
ADMEM  MOV &ADC10MEM0,xxx               ; Move result, flag is reset
       ...                              ; Other instruction needed?
       RETI                             ; Return ;
ADOV   ...                              ; Handle ADCMEM0 overflow
       RETI                             ; Return ;
ADTOV  ...                              ; Handle Conv. time overflow
       RETI                             ; Return ;
ADHI   ...                              ; Handle window comparator high Interrupt
       RETI                             ; Return ;
ADLO   ...                              ; Handle window comparator low Interrupt
       RETI                             ; Return ;
ADIN   ...                              ; Handle window comparator in window Interrupt
       RETI                             ; Return
```

16.3 ADC10_B Registers

The ADC10_B registers are listed in Table 16-2. The base address of the ADC10_B can be found in the device-specific data sheet. The address offset of each ADC10_B register is given in Table 16-2.

Table 16-2. ADC10_B Registers

Register	Short Form	Register Type	Address	Initial State
ADC10_B Control 0 register	ADC10CTL0	Read/write	00h	Reset with POR
ADC10_B Control 1 register	ADC10CTL1	Read/write	02h	Reset with POR
ADC10_B Control 2 register	ADC10CTL2	Read/write	04h	Reset with POR
ADC10_B Window Comparator Low Threshold register	ADC10LO	Read/write	06h	Reset with POR
ADC10_B Window Comparator High Threshold register	ADC10HI	Read/write	08h	Reset with POR
ADC10_B Memory Control register	ADC10MCTL0	Read/write	0Ah	Reset with POR
ADC10_B Conversion Memory register	ADC10MEM0	Read/write	12h	Unchanged
ADC10_B Interrupt Enable register	ADC10IE	Read/write	1Ah	Reset with POR
ADC10_B Interrupt Flag register	ADC10IFG	Read/write	1Ch	Reset with POR
ADC10_B Interrupt Vector register	ADC10IV	Read/write	1Eh	Reset with POR

ADC10_B Control Register 0 (ADC10CTL0)

15	14	13	12	11	10	9	8
Reserved				ADC10SHTx			
r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC10MSC	Reserved		ADC10ON	Reserved		ADC10ENC	ADC10SC
rw-(0)	r0	r0	rw-(0)	r0	r0	rw-(0)	rw-(0)

Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also when a conversion is active.

Reserved Bits 15-12 Reserved. Read back as 0.

ADC10SHTx Bits 11-8 ADC10_B sample-and-hold time. These bits define the number of ADC10CLK cycles in the sampling period for the ADC10.

ADC10SHTx Bits	ADC10CLK 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

ADC10MSC Bit 7 ADC10_B 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.

Reserved Bit 6-5 Reserved. Read back as 0.

ADC10ON Bit 4 ADC10_B on

0 ADC10_B off

1 ADC10_B on

Reserved Bits 3-2 Reserved. Read back as 0.

ADC10ENC Bit 1 ADC10_B enable conversion

0 ADC10_B disabled

1 ADC10_B enabled

ADC10SC Bit 0 ADC10_B start conversion. Software-controlled sample-and-conversion start. ADC10SC and ADC10ENC may be set together with one instruction. ADC10SC is reset automatically.

0 No sample-and-conversion-start

1 Start sample-and-conversion

ADC10_B Control Register 1 (ADC10CTL1)

15	14	13	12	11	10	9	8
Reserved				**ADC10SHSx**		**ADC10SHP**	**ADC10ISSH**
r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC10DIVx			**ADC10SSELx**		**ADC10CONSEQx**		**ADC10BUSY**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)

Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also when a conversion is active.

Reserved	Bits 15-12	Reserved. Read back as 0.	
ADC10SHSx	Bits 11-10	ADC10_B sample-and-hold source select	
		00	ADC10SC bit
		01	Timer trigger 0 - see device specific datasheet
		10	Timer trigger 1 - see device specific datasheet
		11	Timer trigger 2 - see device specific datasheet
ADC10SHP	Bit 9	ADC10_B 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.
ADC10ISSH	Bit 8	ADC10_B invert signal sample-and-hold	
		0	The sample-input signal is not inverted.
		1	The sample-input signal is inverted.
ADC10DIVx	Bits 7-5	ADC10_B clock divider	
		000	/1
		001	/2
		010	/3
		011	/4
		100	/5
		101	/6
		110	/7
		111	/8
ADC10SSELx	Bits 4-3	ADC10_B clock source select	
		00	MODCLK
		01	ACLK
		10	MCLK
		11	SMCLK
ADC10CONSEQx	Bits 2-1	ADC10_B conversion sequence mode select	
		00	Single-channel, single-conversion
		01	Sequence-of-channels
		10	Repeat-single-channel
		11	Repeat-sequence-of-channels
ADC10BUSY	Bit 0	ADC10_B busy. This bit indicates an active sample or conversion operation.	
		0	No operation is active.
		1	A sequence, sample, or conversion is active.

ADC10_B Control Register 2 (ADC10CTL2)

15	14	13	12	11	10	9	8
Reserved						ADC10PDIVx	
r0	r0	r0	r0	r0	r0	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
Reserved			ADC10RES	ADC10DF	ADC10SR	Reserved	Reserved
r0	r0	r0	rw-(1)	rw-(0)	rw-(0)	r0	rw-(0)

Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also when a conversion is active.

Reserved	Bits 15-10	Reserved. Read back as 0.
ADC10PDIVx	Bits 9-8	ADC10_B predivider. This bit predivides the selected ADC10_B clock source before it gets divided again using ADC10DIVx. 00 Predivide by 1 01 Predivide by 4 10 Predivide by 64 11 reserved
Reserved	Bits 7-5	Reserved. Read back as 0.
ADC10RES	Bit 4	ADC10_B resolution. This bit defines the conversion result resolution. 0 8 bit (10 clock cycle conversion time) 1 10 bit (12 clock cycle conversion time)
ADC10DF	Bit 3	ADC10_B 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 03FFh. 1 Signed binary (2s complement), left aligned. Theoretically the analog input voltage – V_{REF} results in 8000h, the analog input voltage + V_{REF} results in 7FC0h.
ADC10SR	Bit 2	ADC10_B sampling rate. This bit selects drive capability of the ADC10_B reference buffer for the maximum sampling rate. Setting ADC10SR reduces the current consumption of this buffer. 0 ADC10_B buffer supports up to ~200 ksps. 1 ADC10_B buffer supports up to ~50 ksps.
Reserved	Bit 1	Reserved. Read back as 0.
Reserved	Bit 0	Reserved. Must be written as 0.

ADC10_B Conversion Memory Register (ADC10MEM0)

15	14	13	12	11	10	9	8
0	0	0	0	0	0	Conversion Results	
r0	r0	r0	r0	r0	r0	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 10-bit conversion results are right justified. Bit 9 is the MSB. Bits 15–10 are 0 in 10-bit mode, and bits 15–8 are 0 in 8-bit mode. Writing to the conversion memory register corrupts the results. This data format is used if ADC10DF = 0.

ADC10_B Conversion Memory Register (ADC10MEM0), 2s-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**	**0**	**0**
rw	rw	r0	r0	r0	r0	r0	r0

Conversion Results Bits 15-0 The 10-bit conversion results are left justified, 2s-complement format. Bit 15 is the MSB. 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 ADC10DF = 1. The data is stored in the right-justified format and is converted to the left-justified 2s-complement format during read back. Writing to the conversion memory register corrupts the results.

ADC10_B Conversion Memory Control Register (ADC10MCTL0)

7	6	5	4	3	2	1	0
Reserved	**ADC10SREFx**			**ADC10INCHx**			
r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Can be modified only when ADC10ENC = 0. Resetting ADC10ENC = 0 by software and changing these fields immediately shows effect also when a conversion is active.

Reserved Bit 7 Reserved. Read back as 0.

ADC10SREFx Bits 6-4 Select reference. It is not recommended to change this setting while a conversion is ongoing.

- 000 V_{R+} = AVCC and V_{R-} = AVSS
- 001 V_{R+} = VREF and V_{R-} = AVSS
- 010 V_{R+} = VEREF+ buffered and V_{R-} = AVSS
- 011 V_{R+} = VEREF+ and V_{R-} = AVSS
- 100 V_{R+} = AVCC and V_{R-} = VEREF-
- 101 V_{R+} = VREF and V_{R-} = VEREF-
- 110 V_{R+} = VEREF+ buffered and V_{R-} = VEREF-
- 111 V_{R+} = VEREF+ and V_{R-} = VEREF-

ADC10INCHx Bits 3-0 Input channel select. Writing these bits select the channel for a single-conversion or the highest channel for a sequence of conversions. Reading these bits in ADC10CONSEQ = 01,11 returns the channel currently converted.

- 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, from REF module.
- 1011 $(AV_{CC} – AV_{SS}) / 2$
- 1100 A12
- 1101 A13
- 1110 A14
- 1111 A15

ADC10_B Window Comparator High Threshold Register (ADC10HI)

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

High Threshold	Bits 15-0	The 10-bit threshold value needs to be right justified. Bit 9 is the MSB. Bits 15–10 are 0 in 10-bit mode, and bits 15–8 are 0 in 8-bit mode. This data format is used if ADC10DF = 0.

ADC10_B Window Comparator High Threshold Register (ADC10HI), 2s-Complement Format

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

High Threshold	Bits 15-0	The 10-bit threshold value needs to be left justified if 2s-complement format is chosen. Bit 15 is the MSB. 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 ADC10DF = 1.

ADC10_B Window Comparator Low Threshold Register (ADC10LO)

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

Low Threshold	Bits 15-0	The 10-bit threshold value needs to be right justified. Bit 9 is the MSB. Bits 15–10 are 0 in 10-bit mode, and bits 15–8 are 0 in 8-bit mode. This data format is used if ADC10DF = 0.

ADC10_B Window Comparator Low Threshold Register (ADC10LO), 2s-Complement Format

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

Low Threshold	Bits 15-0	The 10-bit threshold value needs to be left justified if 2s-complement format is chosen. Bit 15 is the MSB. 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 ADC10DF = 1.

ADC10_B Interrupt Enable Register (ADC10IE)

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
Reserved		ADC10TOVIE	ADC10OVIE	ADC10HIIE	ADC10LOIE	ADC10INIE	ADC10IE0
r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Reserved	Bits 15-6	Reserved. Read back as 0.
ADC10TOVIE	Bit 5	ADC10_B conversion-time-overflow Interrupt enable. 0 Conversion time overflow Interrupt disabled 1 Conversion time overflow Interrupt enabled
ADC10OVIE	Bit 4	ADC10MEM0 overflow Interrupt enable. 0 Overflow Interrupt disabled 1 Overflow Interrupt enabled
ADC10HIIE	Bit 3	Interrupt enable for the above upper threshold Interrupt of the Window comparator. 0 Above upper threshold Interrupt disabled 1 Above upper threshold Interrupt enabled
ADC10LOIE	Bit 2	Interrupt enable for the below lower threshold Interrupt of the Window comparator. 0 Below lower threshold Interrupt disabled 1 Below lower threshold Interrupt enabled
ADC10INIE	Bit 1	Interrupt enable for the inside of window Interrupt of the Window comparator. 0 Inside of window Interrupt disabled 1 Inside of window Interrupt enabled
ADC10IE0	Bit 0	Interrupt enable. This bits enable or disable the Interrupt request for a completed ADC10_B conversion. 0 Interrupt disabled 1 Interrupt enabled

ADC10_B Interrupt Flag Register (ADC10IFG)

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
Reserved		ADC10TOVIFG	ADC10OVIFG	ADC10HIIFG	ADC10LOIFG	ADC10INIFG	ADC10IFG0
r0	r0	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Reserved Bits 15-6 Reserved. Read back as 0.

ADC10TOVIFG Bit 5 The ADC10TOVIFG is set when an ADC10_B conversion is triggered before the actual conversion has completed.

0 No Interrupt pending

1 Interrupt pending

ADC10OVIFG Bit 4 The ADC10OVIFG is set when the ADC10MEM0 register is written before the last conversion result has been read.

0 No Interrupt pending

1 Interrupt pending

ADC10HIIFG Bit 3 The ADC10HIIFG is set when the result of the current ADC10_B conversion is greater than the upper threshold defined by the Window Comparators upper threshold register.

0 No Interrupt pending

1 Interrupt pending

ADC10LOIFG Bit 2 The ADC10LOIFG is set when the result of the current ADC10_B conversion is below the lower threshold defined by the Window Comparators lower threshold register.

0 No Interrupt pending

1 Interrupt pending

ADC10INIFG Bit 1 The ADC10INIFG is set when the result of the current ADC10_B conversion is within the thresholds defined by the Window Comparators threshold registers.

0 No Interrupt pending

1 Interrupt pending

ADC10IFG0 Bit 0 The ADC10IFG0 is set when an ADC10_B conversion is completed. This bit gets reset, when the ADC10MEM0 get read, or may be reset by software.

0 No Interrupt pending

1 Interrupt pending

ADC10_B Interrupt Vector Register (ADC10IV)

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	ADC10IVx			0
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

ADC10IVx Bits 15-0 ADC10_B Interrupt vector value. It generates an value that can be used as address offset for fast interrupt service routine handling. Writing to this register clears all pending interrupt flags.

ADC10IV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No Interrupt pending	–	
002h	ADC10MEM0 overflow	ADC10OVIFG	Highest
004h	Conversion time overflow	ADC10TOVIFG	
006h	ADC10HI Interrupt flag	ADC10HIIFG	
008h	ADC10LO Interrupt flag	ADC10LOIFG	
00Ah	ADC10IN Interrupt flag	ADC10INIFG	
00Ch	ADC10_B memory Interrupt flag	ADC10IFG0	Lowest

Comparator_D

Comparator_D is an analog voltage comparator. This chapter describes the Comparator_D. Comparator_D covers general comparator functionality for up to 16 channels.

Topic **Page**

17.1 Comparator_D Introduction

The Comparator_D module supports precision slope analog-to-digital conversions, supply voltage supervision, and monitoring of external analog signals.

Features of Comparator_D include:

- Inverting and noninverting terminal input multiplexer
- Software-selectable RC filter for the comparator output
- Output provided to Timer_A capture input
- Software control of the port input buffer
- Interrupt capability
- Selectable reference voltage generator, voltage hysteresis generator
- Reference voltage input from shared reference
- Interrupt driven measurement system – low-power operation support

The Comparator_D block diagram is shown in Figure 17-1.

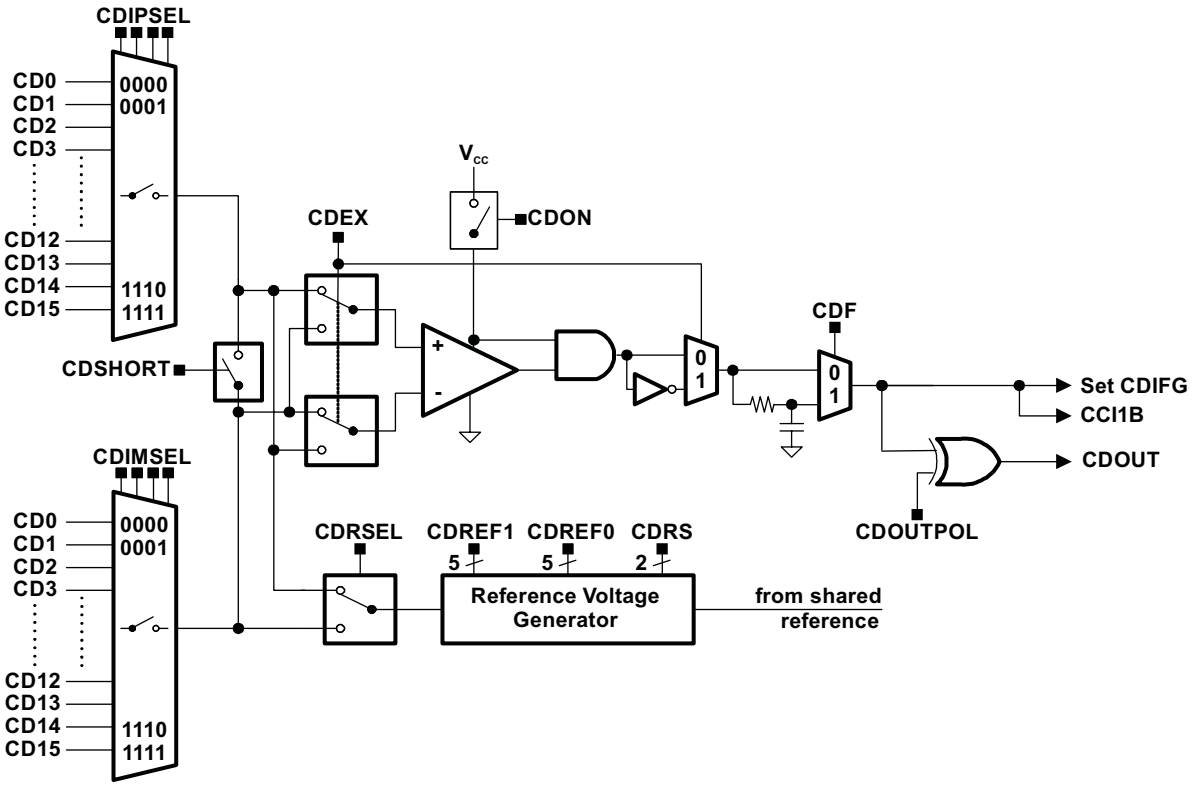


Figure 17-1. Comparator_D Block Diagram

17.2 Comparator_D Operation

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

17.2.1 Comparator

The comparator compares the analog voltages at the + and – input terminals. If the + terminal is more positive than the – terminal, the comparator output CDOUT is high. The comparator can be switched on or off using control bit CDON. The comparator should be switched off when not in use to reduce current consumption. When the comparator is switched off, CDOUT is always low. The bias current of the comparator is programmable.

17.2.2 Analog Input Switches

The analog input switches connect or disconnect the two comparator input terminals to associated port pins using the CDIPSELx and CDIMSELx bits. The comparator terminal inputs can be controlled individually. The CDIPSELx/CDIMSELx bits allow:

- Application of an external signal to the + and – terminals of the comparator
- Routing of an internal reference voltage to an associated output port pin
- Application of an external current source (e.g., resistor) to the + or – terminal of the comparator
- The mapping of both terminals of the internal multiplexer to the outside

Internally, the input switch is constructed as a T-switch to suppress distortion in the signal path.

NOTE: Comparator Input Connection

When the comparator is on, the input terminals should be connected to a signal, power, or ground. Otherwise, floating levels may cause unexpected interrupts and increased current consumption.

The CDEX bit controls the input multiplexer, permuting the input signals of the comparator's + and – terminals. Additionally, when the comparator terminals are permuted, the output signal from the comparator is inverted too. This allows the user to determine or compensate for the comparator input offset voltage.

17.2.3 Port Logic

The Px.y pins associated with a comparator channel are enabled by the CDIPSELx or CDIMSELx bits to disable its digital components while used as comparator input. Only one of the comparator input pins is selected as input to the comparator by the input multiplexer at a time.

17.2.4 Input Short Switch

The CDSHORT bit shorts the Comparator_D inputs. This can be used to build a simple sample-and-hold for the comparator as shown in Figure 17-2.

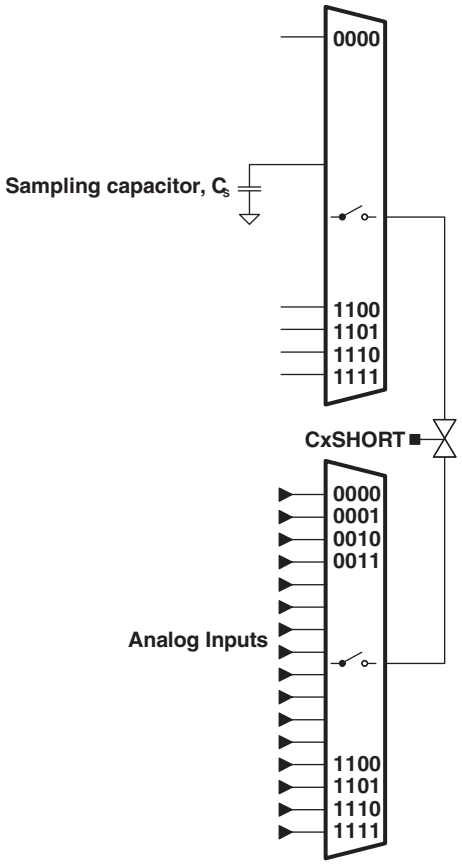


Figure 17-2. Comparator_D Sample-And-Hold

The required sampling time is proportional to the size of the sampling capacitor (C_S), the resistance of the input switches in series with the short switch (R_i), and the resistance of the external source (R_S). The total internal resistance (R_I) is typically in the range of TBD kΩ. The sampling capacitor C_S should be greater than 100 pF. The time constant, Tau, to charge the sampling capacitor C_S can be calculated with the following equation:

$$Tau = (R_I + R_S) \times C_S$$

Depending on the required accuracy, 3 to 10 Tau should be used as a sampling time. With 3 Tau the sampling capacitor is charged to approximately 95% of the input signals voltage level, with 5 Tau it is charged to more than 99%, and with 10 Tau the sampled voltage is sufficient for 12-bit accuracy.

17.2.5 Output Filter

The output of the comparator can be used with or without internal filtering. When control bit CDF is set, the output is filtered with an on-chip RC filter. The delay of the filter can be adjusted in four different steps.

All comparator outputs are oscillating if the voltage difference across the input terminals is small. Internal and external parasitic effects and cross coupling on and between signal lines, power supply lines, and other parts of the system are responsible for this behavior as shown in Figure 17-3. The comparator output oscillation reduces the accuracy and resolution of the comparison result. Selecting the output filter can reduce errors associated with comparator oscillation.

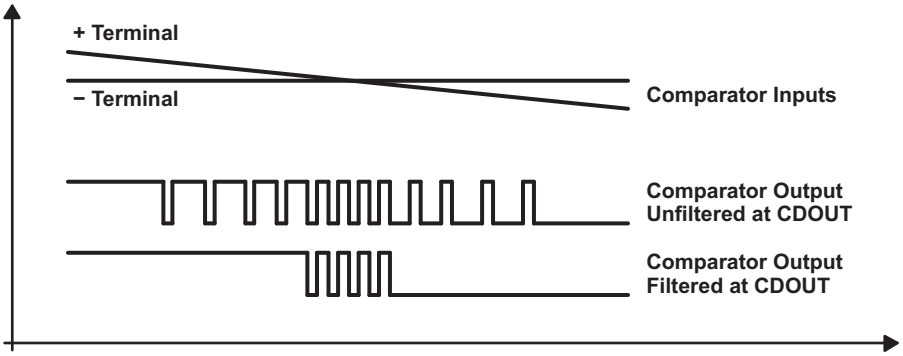


Figure 17-3. RC-Filter Response at the Output of the Comparator

17.2.6 Reference Voltage Generator

The Comparator_D reference block diagram is shown in Figure 17-4.

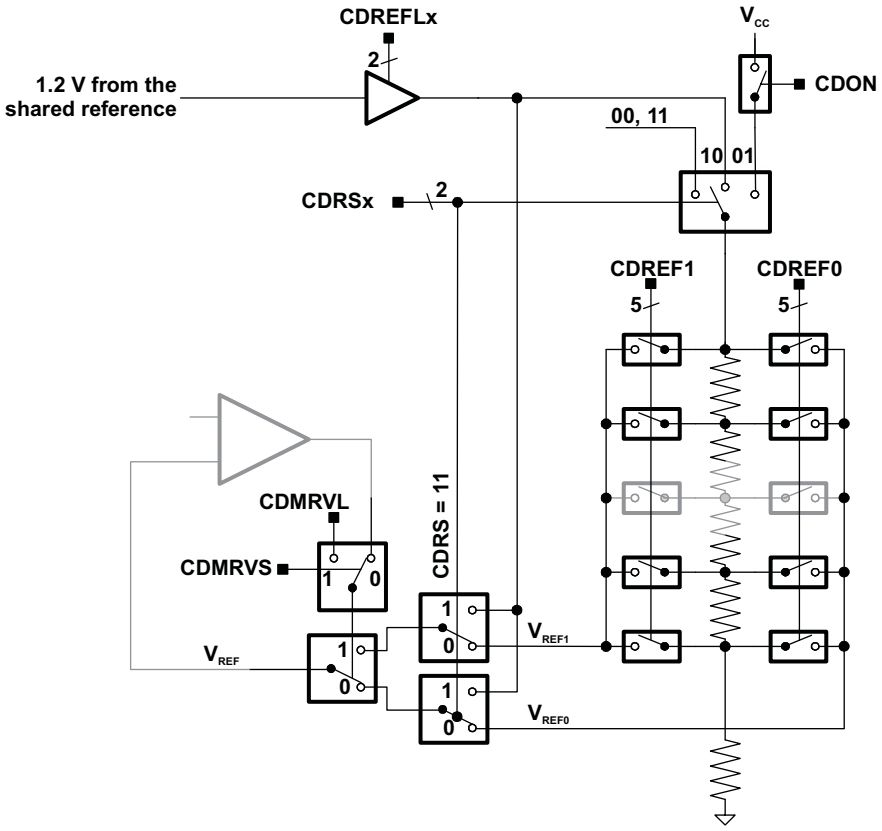


Figure 17-4. Reference Generator Block Diagram

The voltage reference generator is used to generate VREF, which can be applied to either comparator input terminal. The CDREF1x (VREF1) and CDREF0x (VREF0) bits control the output of the voltage generator. The CDRSEL bit selects the comparator terminal to which VREF is applied. If external signals are applied to both comparator input terminals, the internal reference generator should be turned off to reduce current consumption. The voltage reference generator can generate a fraction of the device's V_{CC} or of the voltage reference of the integrated precision voltage reference source. Vref1 is used while CDOUT is 1 and Vref0 is used while CDOUT is 0. This allows the generation of a hysteresis without using external components.

17.2.7 Comparator_D, Port Disable Register CDPD

The comparator input and output functions are multiplexed with the associated I/O port pins, which are digital CMOS gates. When analog signals are applied to digital CMOS gates, parasitic current can flow from V_{CC} to GND. This parasitic current occurs if the input voltage is near the transition level of the gate. Disabling the port pin buffer eliminates the parasitic current flow and therefore reduces overall current consumption.

The CDPDx bits, when set, disable the corresponding Px.y input buffer as shown in Figure 17-5. When current consumption is critical, any Px.y pin connected to analog signals should be disabled with their associated CDPDx bits.

Selecting an input pin to the comparator multiplexer with the CDIPSEL or CDIMSEL bits automatically disables the input buffer for that pin, regardless of the state of the associated CDPDx bit.

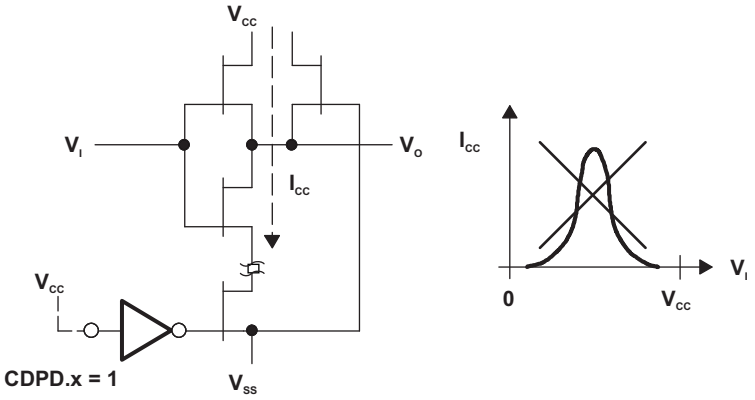


Figure 17-5. Transfer Characteristic and Power Dissipation in a CMOS Inverter/Buffer

17.2.8 Comparator_D Interrupts

One interrupt flag and one interrupt vector is associated with the Comparator_D.

The interrupt flag CDIFG is set on either the rising or falling edge of the comparator output, selected by the CDIES bit. If both the CDIE and the GIE bits are set, then the CDIFG interrupt flag generates an interrupt request.

17.2.9 Comparator_D Used to Measure Resistive Elements

The Comparator_D can be optimized to precisely measure resistive elements using single slope analog-to-digital conversion. For example, temperature can be converted into digital data using a thermistor, by comparing the thermistor's capacitor discharge time to that of a reference resistor as shown in Figure 17-6. A reference resister Rref is compared to Rmeas.

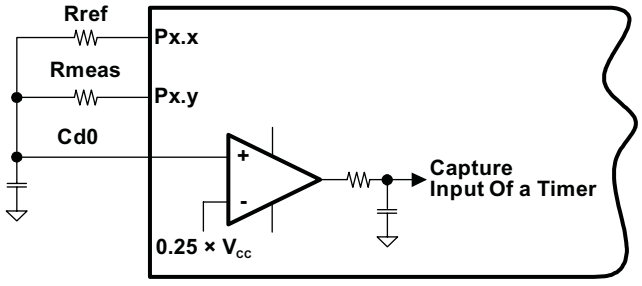


Figure 17-6. Temperature Measurement System

The resources used to calculate the temperature sensed by Rmeas are:

- Two digital I/O pins charge and discharge the capacitor.
- I/O is set to output high (V_{CC}) to charge capacitor, reset to discharge.
- I/O is switched to high-impedance input with CDPDx set when not in use.
- One output charges and discharges the capacitor via Rref.
- One output discharges capacitor via Rmeas.
- The + terminal is connected to the positive terminal of the capacitor.
- The – terminal is connected to a reference level, for example 0.25 × V_{CC}.
- The output filter should be used to minimize switching noise.
- CDOUT is used to gate a timer capturing capacitor discharge time.

More than one resistive element can be measured. Additional elements are connected to CD0 with available I/O pins and switched to high impedance when not being measured.

The thermistor measurement is based on a ratiometric conversion principle. The ratio of two capacitor discharge times is calculated as shown in Figure 17-7.

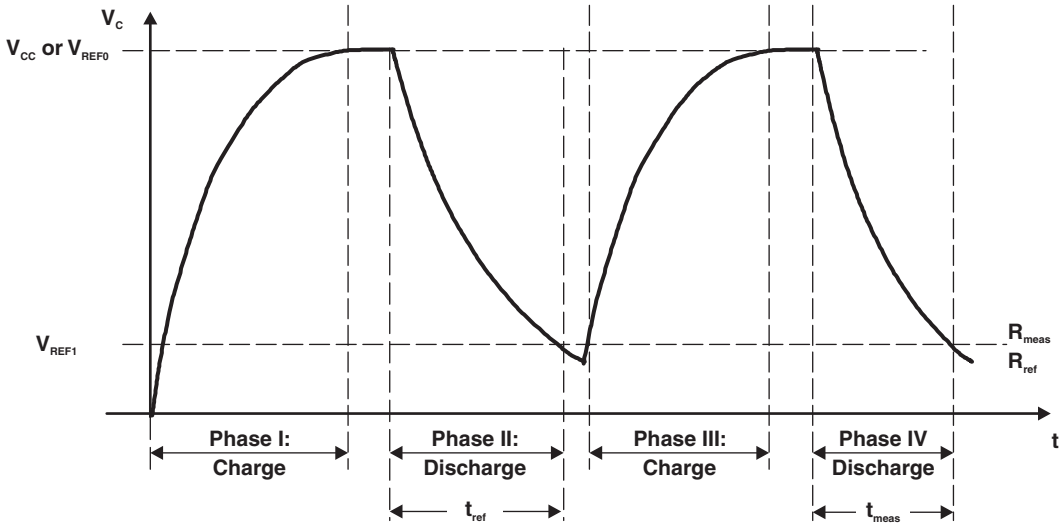


Figure 17-7. Timing for Temperature Measurement Systems

The V_{CC} voltage and the capacitor value should remain constant during the conversion, but are not critical since they cancel in the ratio:

$$\frac{N_{meas}}{N_{ref}} = \frac{-R_{meas} \times C \times \ln \frac{V_{ref1}}{V_{CC}}}{-R_{ref} \times C \times \ln \frac{V_{ref1}}{V_{CC}}}$$

$$\frac{N_{meas}}{N_{ref}} = \frac{R_{meas}}{R_{ref}}$$

$$R_{meas} = R_{ref} \times \frac{N_{meas}}{N_{ref}}$$

17.3 Comparator_D Registers

The Comparator_D registers are listed in Table 17-1. The base address of the Comparator_D module can be found in the device specific data sheet.

Table 17-1. Comparator_D Registers

Register	Short Form	Register Type	Address Offset	Initial State
Comparator_D control register 0	CDCTL0	Read/write	0x0000	Reset with PUC
Comparator_D control register 1	CDCTL1	Read/write	0x0002	Reset with PUC
Comparator_D control register 2	CDCTL2	Read/write	0x0004	Reset with PUC
Comparator_D control register 3	CDCTL3	Read/write	0x0006	Reset with POR
Comparator_D interrupt register	CDINT	Read/write	0x000C	Reset with PUC
Comparator_D interrupt vector word	CDIV	Read	0x000E	Reset with PUC

Comparator_D Control Register 0 (CDCTL0)

15	14	13	12	11	10	9	8
CDIMEN	**Reserved**			**CDIMSEL**			
rw-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
CDIPEN	**Reserved**			**CDIPSEL**			
rw-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0

CDIMEN	Bit 15	Channel input enable for the V– terminal of the comparator. 0 Selected analog input channel for V– terminal is disabled. 1 Selected analog input channel for V– terminal is enabled.
Reserved	Bits 14-12	Reserved
CDIMSEL	Bits 11-8	Channel input selected for the V– terminal of the comparator if CDIMEN is set to 1.
CDIPEN	Bit 7	Channel input enable for the V+ terminal of the comparator. 0 Selected analog input channel for V+ terminal is disabled. 1 Selected analog input channel for V+ terminal is enabled.
Reserved	Bits 6-4	Reserved
CDIPSEL	Bits 3-0	Channel input selected for the V+ terminal of the comparator if CDIPEN is set to 1.

Comparator_D, Control Register 1 (CDCTL1)

15	14	13	12	11	10	9	8
Reserved			**CDMRVS**	**CDMRVL**	**CDON**	**Reserved**	
r-0	r-0	r-0	rw-0	rw-0	rw-0	r-0	r-0

7	6	5	4	3	2	1	0
CDFDLY		**CDEX**	**CDSHORT**	**CDIES**	**CDF**	**CDOUTPOL**	**CDOUT**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-0

Reserved	Bits 15-13	Reserved
CDMRVS	Bit 12	This bit defines if the comparator output selects between VREF0 or VREF1 if CDRS = 00, 01, or 10. 0 Comparator output state selects between VREF0 or VREF1. 1 CDMRVL selects between VREF0 or VREF1.
CDMRVL	Bit 11	This bit is valid of CDMRVS is set to 1. 0 VREF0 is selected if CDRS = 00, 01, or 10. 1 VREF1 is selected if CDRS = 00, 01, or 10.
CDON	Bit 10	On. This bit turns the comparator on. When the comparator is turned off the Comparator_D consumes no power. 0 Off 1 On
Reserved	Bits 9-8	Reserved
CDFDLY	Bits 7-6	Filter delay. The filter delay can be selected in 4 steps. See the device specific data sheet for details. 00 Typical filter delay of TBD (450) ns 01 Typical filter delay of TBD (900) ns 10 Typical filter delay of TBD (1800) ns 11 Typical filter delay of TBD (3600) ns
CDEX	Bit 5	Exchange. This bit permutes the comparator 0 inputs and inverts the comparator 0 output.
CDSHORT	Bit 4	Input short. This bit shorts the + and – input terminals. 0 Inputs not shorted 1 Inputs shorted
CDIES	Bit 3	Interrupt edge select for CDIIFG and CDIFG 0 Rising edge for CDIFG, falling edge for CDIIFG 1 Falling edge for CDIFG, rising edge for CDIIFG
CDF	Bit 2	Output filter 0 Comparator_D output is not filtered 1 Comparator_D output is filtered
CDOUTPOL	Bit 1	Output polarity. This bit defines the CDOUT polarity. 0 Noninverted 1 Inverted
CDOUT	Bit 0	Output value. This bit reflects the value of the Comparator_D output. Writing this bit has no effect on the comparator output.

Comparator_D, Control Register 2 (CDCTL2)

15	14	13	12	11	10	9	8
CDREFACC	**CDREFL**		**CDREF1**				
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
CDRS		**CDRSEL**	**CDREF0**				
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

CDREFACC Bit 15 Reference accuracy. A reference voltage is requested only if CDREFL > 0.

- 0 Static mode
- 1 Clocked (low-power, low-accuracy) mode

CDREFL Bits 14-13 Reference voltage level

- 00 Reference amplifier is disabled. No reference voltage is requested.
- 01 1.5 V is selected as shared reference voltage input
- 10 2.0 V is selected as shared reference voltage input
- 11 2.5 V is selected as shared reference voltage input

CDREF1 Bits 12-8 Reference resistor tap 1. This register defines the tap of the resistor string while CDOUT = 1.

CDRS Bits 7-6 Reference source. This bit define if the reference voltage is derived from V_{CC} or from the precise shared reference.

- 00 No current is drawn by the reference circuitry.
- 01 V_{CC} applied to the resistor ladder
- 10 Shared reference voltage applied to the resistor ladder.
- 11 Shared reference voltage supplied to V_{CREF}. Resistor ladder is off.

CDRSEL Bit 5 Reference select. This bit selects which terminal the V_{CCREF} is applied to.
When CDEX = 0:

- 0 V_{REF} is applied to the + terminal
- 1 V_{REF} is applied to the – terminal

When CDEX = 1:

- 0 V_{REF} is applied to the – terminal
- 1 V_{REF} is applied to the + terminal

CDREF0 Bits 4-0 Reference resistor tap 0. This register defines the tap of the resistor string while CDOUT = 0.

Comparator_D, Control Register 3 (CDCTL3)

15	14	13	12	11	10	9	8
CDPD15	**CDPD14**	**CDPD13**	**CDPD12**	**CDPD11**	**CDPD10**	**CDPD9**	**CDPD8**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
CDPD7	**CDPD6**	**CDPD5**	**CDPD4**	**CDPD3**	**CDPD2**	**CDPD1**	**CDPD0**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

CDPDx Bit 15-0 Port disable. These bits individually disable the input buffer for the pins of the port associated with Comparator_D. The bit CDPDx disabled the port of the comparator channel x.

- 0 The input buffer is enabled.
- 1 The input buffer is disabled.

Comparator_D, Interrupt Control Register (CDINT)

15	14	13	12	11	10	9	8
Reserved						CDIIE	CDIE
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0
7	6	5	4	3	2	1	0
Reserved						CDIIFG	CDIFG
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

Reserved Bits 15-10 Reserved. Always read back 0.

CDIIE Bit 9 Comparator_D output interrupt enable inverted polarity

0 Interrupt is disabled

1 Interrupt is enabled

CDIE Bit 8 Comparator_D output interrupt enable

0 Interrupt is disabled

1 Interrupt is enabled

Reserved Bits 7-2 Reserved. Always read back 0.

CDIIFG Bit 1 Comparator_D output inverted interrupt flag. The bit CDIES defines the transition of the output setting this bit.

0 No interrupt pending

1 Output interrupt pending

CDIFG Bit 0 Comparator_D output interrupt flag. The bit CDIES defines the transition of the output setting this bit.

0 No interrupt pending

1 Output interrupt pending

Comparator_D, Interrupt Vector Word Register (CDIV)

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	CDIV		0
r0	r0	r0	r0	r0	r-(0)	r-(0)	r0

CDIV Bits 15-0 Comparator_D interrupt vector word register. The interrupt vector register reflects only interrupt flags whose interrupt enable bit are set. Reading the CDIV register clears the pending interrupt flag with the highest priority.

CDIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	–	–
02h	CDOUT interrupt	CDIFG	Highest
04h	CDOUT interrupt inverted polarity	CDIIFG	Lowest

Enhanced Universal Serial Communication Interface (eUSCI) – UART Mode

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

Topic **Page**

18.1 Enhanced Universal Serial Communication Interface A (eUSCI_A) Overview

The eUSCI_A module supports two serial communication modes:

- I^2C mode
- SPI mode

18.2 eUSCI_A Introduction – UART Mode

In asynchronous mode, the eUSCI_Ax modules connect the device 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-bit 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, transmit, start bit received, and transmit complete

Figure 18-1 shows the eUSCI_Ax when configured for UART mode.

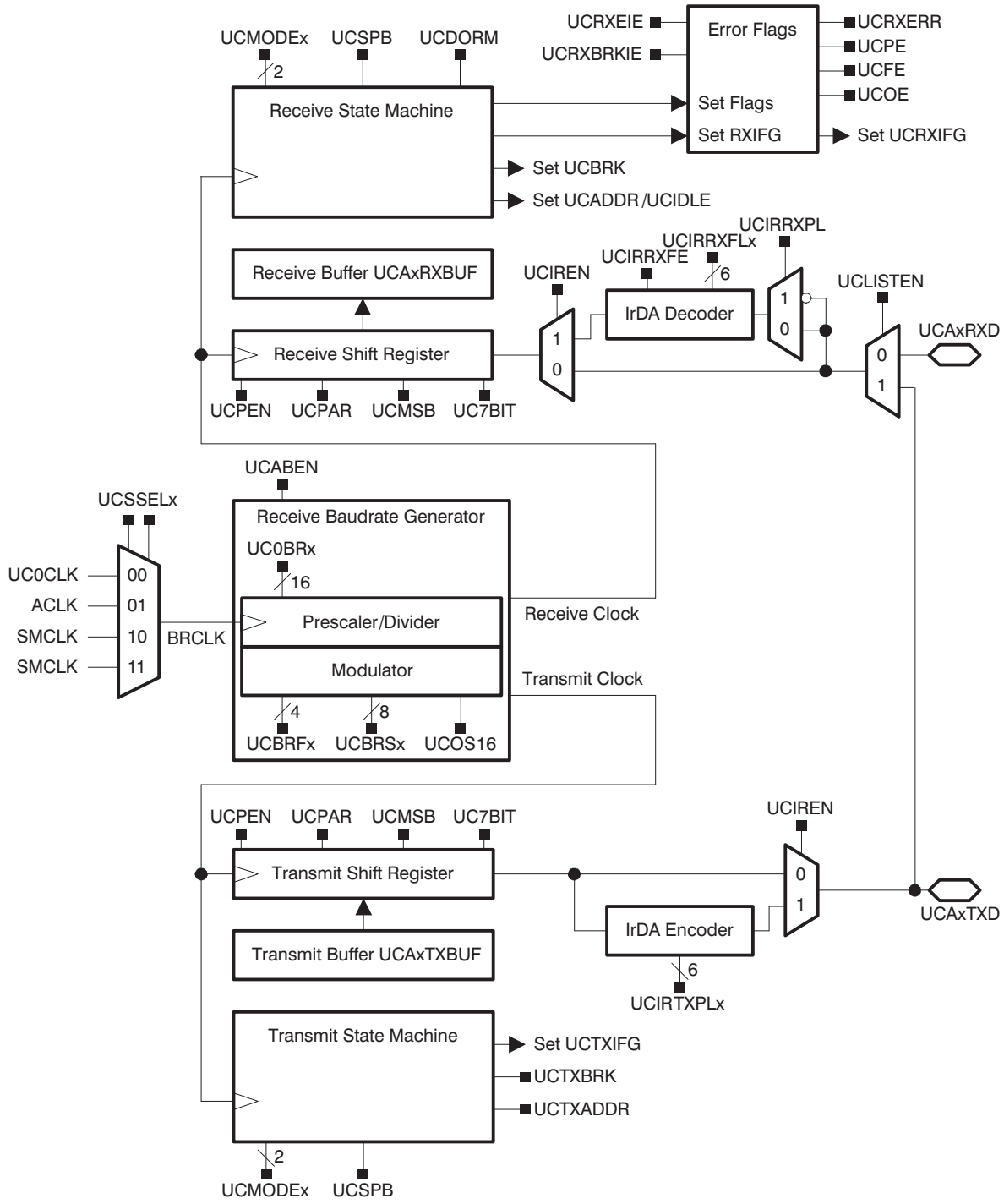


Figure 18-1. eUSCI_Ax Block Diagram – UART Mode (UCSYNC = 0)

18.3 eUSCI_A Operation – UART Mode

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

18.3.1 eUSCI_A Initialization and Reset

The eUSCI_A is reset by a PUC or by setting the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the eUSCI_A in a reset condition. When set, the UCSWRST bit sets the UCTXIFG bit and resets the UCRXIE, UCTXIE, UCRXIFG, UCRXERR, UCBRK, UCPE, UCOE, UCFE, UCSTOE, and UCBTOE bits. Clearing UCSWRST releases the eUSCI_A for operation.

NOTE: Initializing or reconfiguring the eUSCI_A module

The recommended eUSCI_A initialization/reconfiguration process is:

1. Set UCSWRST (`BIS.B #UCSWRST,&UCAxCTL1`).
2. Initialize all eUSCI_A 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.

18.3.2 Character Format

The UART character format (see Figure 18-2) consists of a start bit, seven or eight data bits, an even/odd/no parity bit, an address bit (address-bit mode), and one or two stop bits. The UCMSB bit controls the direction of the transfer and selects LSB or MSB first. LSB first is typically required for UART communication.

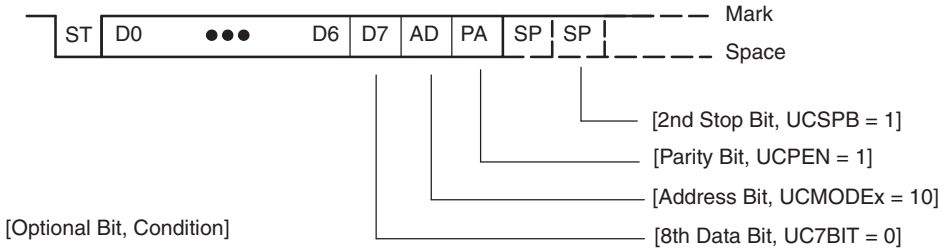


Figure 18-2. Character Format

18.3.3 Asynchronous Communication Format

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

18.3.3.1 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 (see Figure 18-3). An idle receive line is detected when ten or more continuous ones (marks) are received after the one or two stop bits of a character. The baud-rate generator is switched off after reception of an idle line until the next start edge is detected. When an idle line is detected, the UCIDLE bit is set.

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

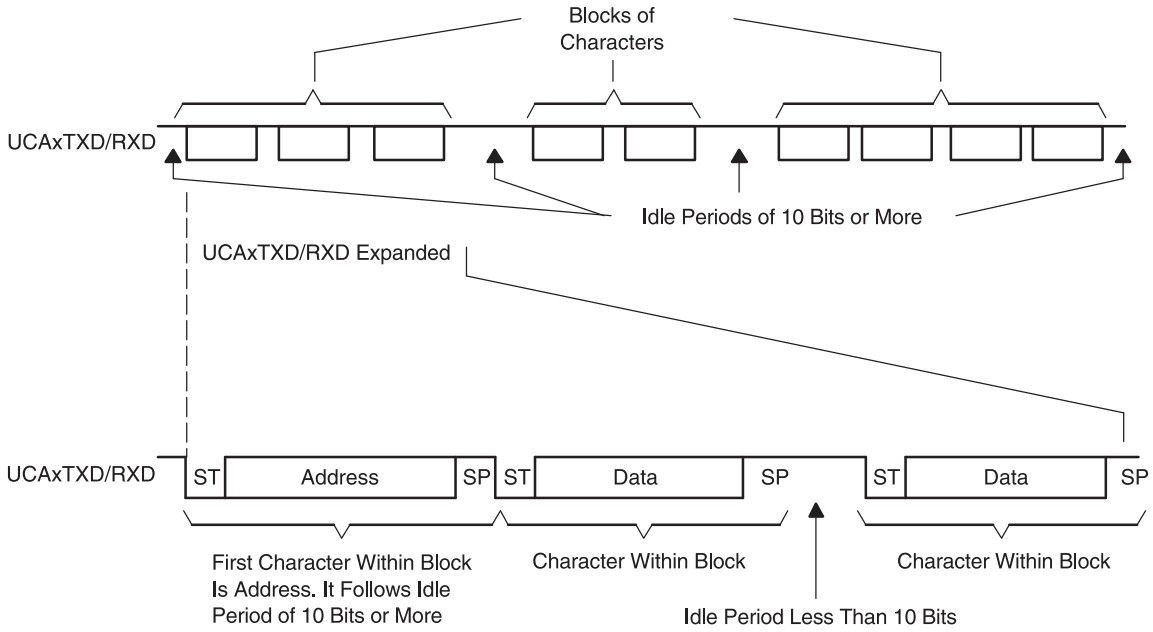


Figure 18-3. Idle-Line Format

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 are received. When UCDORM is cleared during the reception of a character, the receive interrupt flag is set after the reception completed. The UCDORM bit is not modified automatically by the eUSCI_A hardware.

For address transmission in idle-line multiprocessor format, a precise idle period can be generated by the eUSCI_A 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.

18.3.3.1.1 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 is misinterpreted as an address.

18.3.3.2 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 (see Figure 18-4). The first character in a block of characters carries a set address bit that indicates that the character is an address. The eUSCI_A 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 are received. The UCDORM bit is not modified by the eUSCI_A hardware automatically.

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

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

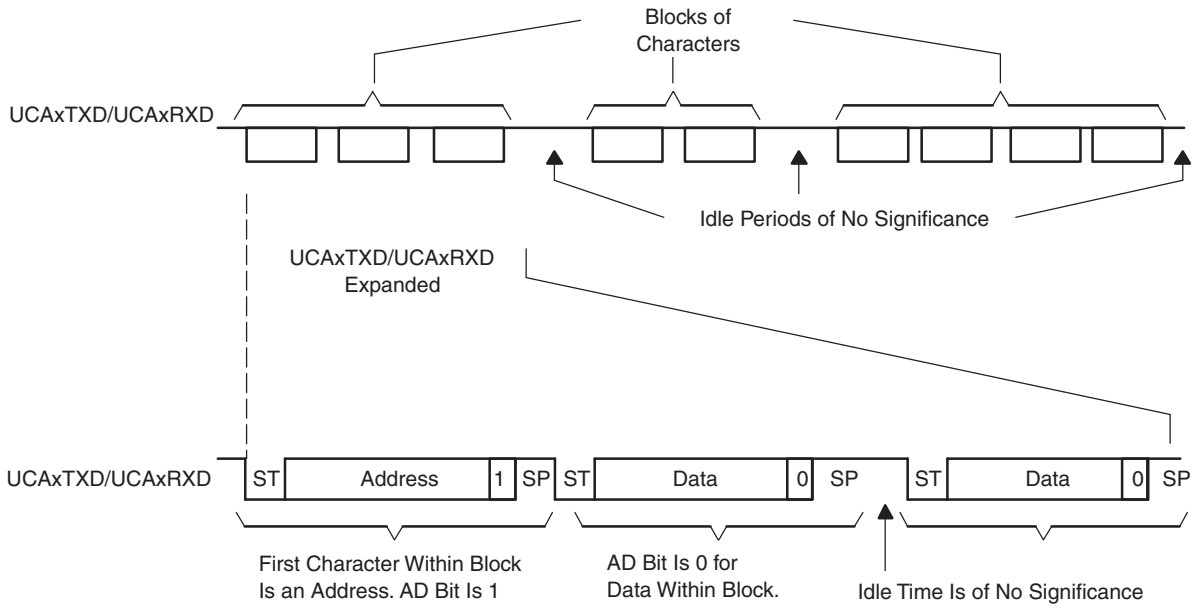


Figure 18-4. Address-Bit Multiprocessor Format

18.3.3.2.1 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 is also set. In this case, the value in UCAxRXBUF is 0h, because 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.

18.3.4 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 eUSCI_A cannot transmit data while receiving the break/sync field. The synch field follows the break as shown in Figure 18-5.

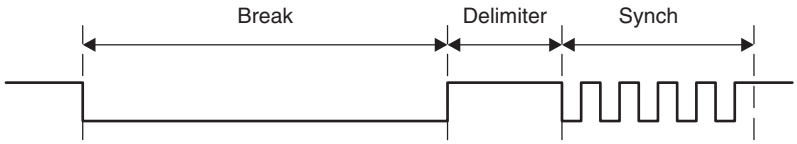


Figure 18-5. Auto Baud-Rate Detection – Break/Synch Sequence

For LIN conformance, the character format should be set to eight 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 (see Figure 18-6). 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 (UCAxBRW and UCAxMCTLW). If the length of the synch field exceeds the measurable time, the synch timeout error flag UCSTOE is set.

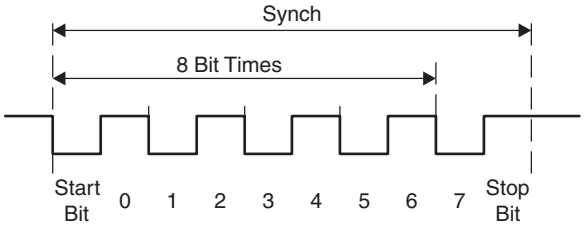


Figure 18-6. Auto Baud-Rate Detection – Synch Field

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 is received. The UCDORM bit is not modified by the eUSCI_A hardware automatically.

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

The counter used to detect the baud rate is limited to 0FFFFh (2^{16}) counts. This means the minimum baud rate detectable is 244 baud in oversampling mode and 15 baud in low-frequency mode. The highest detectable baudrate is 1 Mbaud.

The automatic baud-rate detection mode can be used in a full-duplex communication system with some restrictions. The eUSCI_A cannot transmit data while receiving the break/sync field and, if a 0h byte with framing error is received, any data transmitted during this time is corrupted. The latter case can be discovered by checking the received data and the UCFE bit.

18.3.4.1 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.

18.3.5 IrDA Encoding and Decoding

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

18.3.5.1 IrDA Encoding

The encoder sends a pulse for every zero bit in the transmit bit stream coming from the UART (see Figure 18-7). The pulse duration is defined by UCIRTXPLx bits specifying the number of one-half clock periods of the clock selected by UCIRTXCLK.

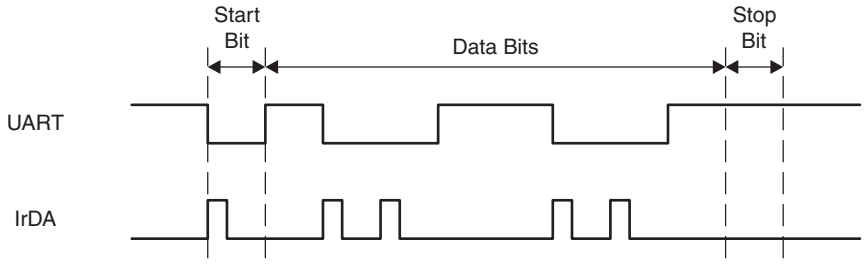


Figure 18-7. UART vs IrDA Data Format

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 six one-half clock cycles with UCIRTXPLx = 6 – 1 = 5.

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

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

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

18.3.5.2 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 the device is in active mode.

18.3.6 Automatic Error Detection

Glitch suppression prevents the eUSCI_A from being accidentally started. Any pulse on UCAxRXD shorter than the deglitch time t_t (selected by UCGLITx) is ignored (see the device-specific data sheet 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 eUSCI_A 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 eUSCI_A 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 18-1.

Table 18-1. 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 does not function properly. To detect overflows reliably, the following flow is recommended. After a character is received and UCAxRXIFG is set, first read UCAxSTAT to check the error flags including the overflow flag UCOE. Read UCAxRXBUF next. This clears all error flags except UCOE, if UCAxRXBUF was overwritten between the read access to UCAxSTAT and to UCAxRXBUF. Therefore, the UCOE flag should be checked after reading UCAxRXBUF to detect this condition. Note that, in this case, the UCRXERR flag is not set.

18.3.7 eUSCI_A Receive Enable

The eUSCI_A 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 is 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, the UART state machine returns to its idle state, and the baud rate generator is turned off.

18.3.7.1 Receive Data Glitch Suppression

Glitch suppression prevents the eUSCI_A from being accidentally started. Any glitch on UCAxRXD shorter than the deglitch time t_t is ignored by the eUSCI_A, and further action is initiated as shown in Figure 18-8 (see the device-specific data sheet for parameters). The deglitch time t_t can be set to four different values using the UCGLITx bits.

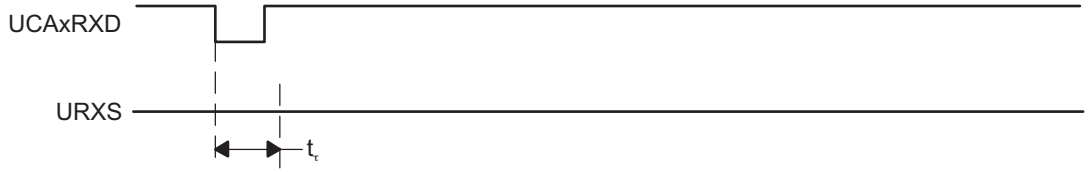


Figure 18-8. Glitch Suppression, eUSCI_A Receive Not Started

When a glitch is longer than t_t, or a valid start bit occurs on UCAxRXD, the eUSCI_A receive operation is started and a majority vote is taken (see Figure 18-9). If the majority vote fails to detect a start bit, the eUSCI_A halts character reception.

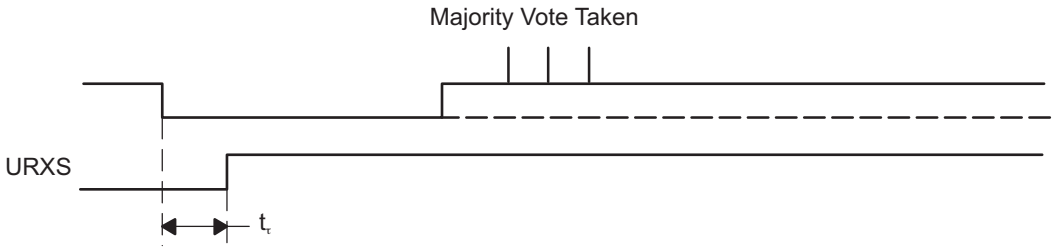


Figure 18-9. Glitch Suppression, eUSCI_A Activated

18.3.8 eUSCI_A Transmit Enable

The eUSCI_A 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.

18.3.9 UART Baud-Rate Generation

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

A quick setup for finding the correct baudrate settings for the eUSCI_A can be found in Section 18.3.10.

18.3.9.1 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 (for example, 9600 baud from a 32768-Hz 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 causes 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 eUSCI_A baud rate is one-third the UART source clock frequency BRCLK.

Timing for each bit is shown in Figure 18-10. 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.

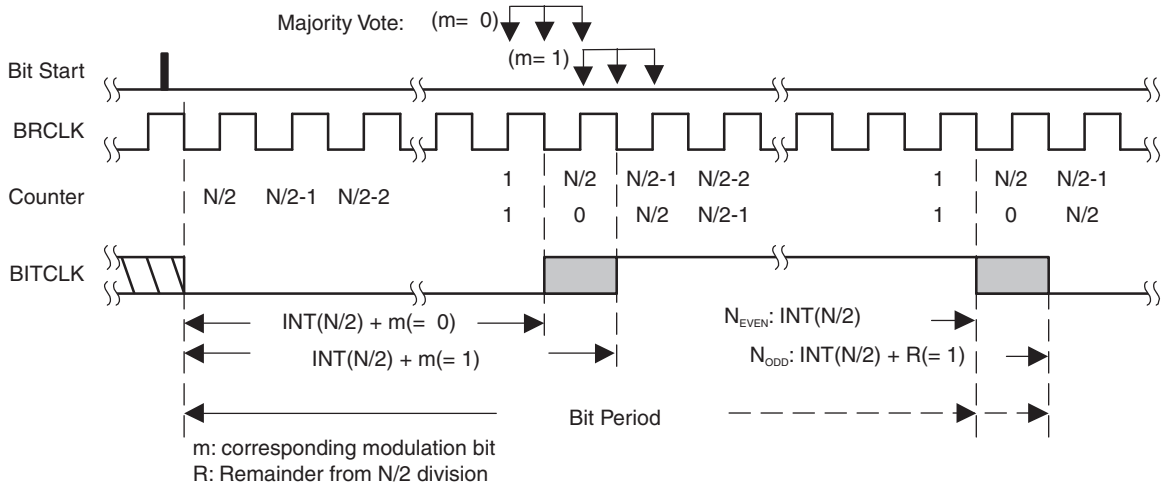


Figure 18-10. BITCLK Baud-Rate Timing With UCOS16 = 0

Modulation is based on the UCBRSx setting as shown in Table 18-2. 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 18-2. Modulation Pattern Examples

UCBRSx	Bit 0 (Start Bit)	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
0x00	0	0	0	0	0	0	0	0
0x01	0	0	0	0	0	0	0	1
				⋮				
0x35	0	0	1	1	0	1	0	1
0x36	0	0	1	1	0	1	1	0
0x37	0	0	1	1	0	1	1	1
				⋮				
0xff	1	1	1	1	1	1	1	1

The correct setting of UCBRSx can be found as described in Section 18.3.10.

18.3.9.2 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 by 16 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 eUSCI_A baud rate is 1/16 the UART source clock frequency BRCLK.

Modulation for BITCLK16 is based on the UCBRFx setting (see Table 18-3). 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 previously described.

Table 18-3. BITCLK16 Modulation Pattern

UCBRFx	Number 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

18.3.10 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 noninteger 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, it is recommended to use the oversampling baud-rate generation mode by setting UCOS16.

NOTE: Baudrate settings quick set up

To calculate the correct the correct settings for the baudrate generation, perform these steps:

1. Calculate $N = f_{BRCLK}/\text{Baudrate}$ [if N > 16 continue with step 3, otherwise with step 2]
2. OS16 = 0, UCBRx = INT(N) [continue with step 4]
3. OS16 = 1, UCBRx = INT(N/16), UCBRFx = INT([(N/16) – INT(N/16)] × 16)
4. UCBRSx can be found by looking up the fractional part of N (= N - INT(N)) in table Table 18-4
5. If OS16 = 0 was chosen, a detailed error calculation is recommended to be performed

Table 18-4 can be used as a lookup table for finding the correct UCBRSx modulation pattern for the corresponding fractional part of N. The values there are optimized for transmitting.

Table 18-4. UCBRSx Settings for Fractional Portion of $N = f_{BRCLK}/\text{Baudrate}$

Fractional Portion of N	UCBRSx (1)	Fractional Portion of N	UCBRSx (1)
0.0000	0x00	0.5002	0xAA
0.0529	0x01	0.5715	0x6B
0.0715	0x02	0.6003	0xAD
0.0835	0x04	0.6254	0xB5
0.1001	0x08	0.6432	0xB6
0.1252	0x10	0.6667	0xD6
0.1430	0x20	0.7001	0xB7
0.1670	0x11	0.7147	0xBB
0.2147	0x21	0.7503	0xDD
0.2224	0x22	0.7861	0xED
0.2503	0x44	0.8004	0xEE
0.3000	0x25	0.8333	0xBF
0.3335	0x49	0.8464	0xDF
0.3575	0x4A	0.8572	0xEF
0.3753	0x52	0.8751	0xF7
0.4003	0x92	0.9004	0xFB
0.4286	0x53	0.9170	0xFD
0.4378	0x55	0.9288	0xFE

(1) The UCBRSx setting in one row is valid from the fractional portion given in that row until the one in the next row

18.3.10.1 Low-Frequency Baud-Rate Mode Setting

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

UCBRx = INT(N)

The fractional portion is realized by the modulator with its UCBRSx setting. The recommended way of determining the correct UCBRSx is performing a detailed error calculation as explained in the following sections. However it is also possible to look up the correct settings in table with typical crystals (see Table 18-5).

18.3.10.2 Oversampling Baud-Rate Mode Setting

In the oversampling mode, the prescaler is set to:

UCBRx = INT(N/16)

and the first stage modulator is set to:

UCBRFx = INT([(N/16) – INT(N/16)] × 16)

The second modulation stage setting (UCBRSx) can be found by performing a detailed error calculation or by using Table 18-4 and the fractional part of N = f_{BRCLK}/Baudrate.

18.3.11 Transmit Bit Timing - Error calculation

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.

18.3.11.1 Low-Frequency Baud-Rate Mode Bit Timing

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

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

Where:

$m_{UCBRSx}[i]$ = Modulation of bit i of UCBRSx

18.3.11.2 Oversampling Baud-Rate Mode Bit Timing

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

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

Where:

$\sum_{j=0}^{15} m_{UCBRFx}[j]$ = Sum of ones from the corresponding row in Table 18-3

$m_{UCBRSx}[i]$ = Modulation of bit i of UCBRSx

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):

$Error_{TX}[i] = (t_{bit,TX}[i] – t_{bit,ideal,TX}[i]) × Baudrate × 100\%$

18.3.12 Receive Bit Timing – Error Calculation

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 eUSCI_A module. Figure 18-11 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.

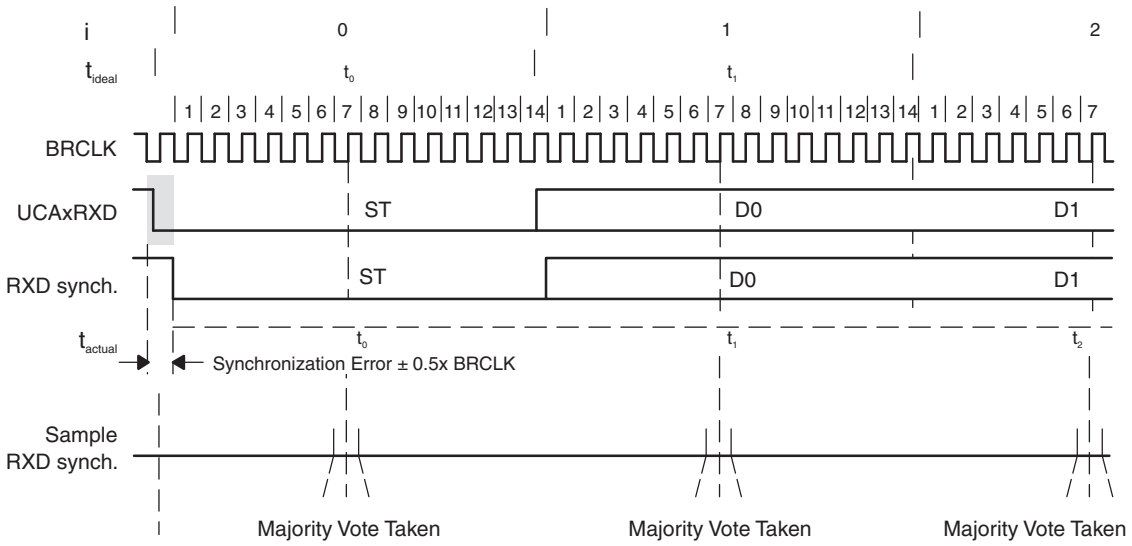


Figure 18-11. Receive Error

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 of UCBRSx

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

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

Where:

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

$\sum_{j=0}^{7 + m_{UCBRSx}[i]} m_{UCBRFx}[j]$ = Sum of ones from columns 0 to (7 + $m_{UCBRSx}[i]$) from the corresponding row in Table 18-3.

$m_{UCBRSx}[i]$ = Modulation of bit i of UCBRSx

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

$Error_{RX}[i] = (t_{bit,RX}[i] – t_{bit,ideal,RX}[i]) \times Baudrate \times 100\%$

18.3.13 Typical Baud Rates and Errors

Standard baud-rate data for UCBRx, UCBRSx, and UCBRFx are listed in Table 18-5 for a 32768-Hz crystal sourcing ACLK and typical SMCLK frequencies. Make sure that the selected BRCLK frequency does not exceed the device specific maximum eUSCI_A input frequency (see the device-specific data sheet).

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 18-5. Recommended Settings for Typical Crystals and Baudrates

BRCLK	Baudrate	UCOS16	UCBRx	UCFx	UCSx	TX error (%)		RX error (%)	
						neg	pos	neg	pos
32768	1200	1	1	11	0x25	-2.29	2.25	-2.56	5.35
32768	2400	0	13	-	0xB6	-3.12	3.91	-5.52	8.84
32768	4800	0	6	-	0xEE	-7.62	8.98	-21	10.25
32768	9600	0	3	-	0x92	-17.19	16.02	-23.24	37.3
1000000	9600	1	6	8	0x20	-0.48	0.64	-1.04	1.04
1000000	19200	1	3	4	0x2	-0.8	0.96	-1.84	1.84
1000000	38400	1	1	10	0x0	0	1.76	0	3.44
1000000	57600	0	17	-	0x4A	-2.72	2.56	-3.76	7.28
1000000	115200	0	8	-	0xD6	-7.36	5.6	-17.04	6.96
1048576	9600	1	6	13	0x22	-0.46	0.42	-0.48	1.23
1048576	19200	1	3	6	0xAD	-0.88	0.83	-2.36	1.18
1048576	38400	1	1	11	0x25	-2.29	2.25	-2.56	5.35
1048576	57600	0	18	-	0x11	-2	3.37	-5.31	5.55
1048576	115200	0	9	-	0x08	-5.37	4.49	-5.93	14.92
4000000	9600	1	26	0	0xB6	-0.08	0.16	-0.28	0.2
4000000	19200	1	13	0	0x84	-0.32	0.32	-0.64	0.48
4000000	38400	1	6	8	0x20	-0.48	0.64	-1.04	1.04
4000000	57600	1	4	5	0x55	-0.8	0.64	-1.12	1.76
4000000	115200	1	2	2	0xBB	-1.44	1.28	-3.92	1.68
4000000	230400	0	17	-	0x4A	-2.72	2.56	-3.76	7.28
4194304	9600	1	27	4	0xFB	-0.11	0.1	-0.33	0
4194304	19200	1	13	10	0x55	-0.21	0.21	-0.55	0.33
4194304	38400	1	6	13	0x22	-0.46	0.42	-0.48	1.23
4194304	57600	1	4	8	0xEE	-0.75	0.74	-2	0.87
4194304	115200	1	2	4	0x92	-1.62	1.37	-3.56	2.06
4194304	230400	0	18	-	0x11	-2	3.37	-5.31	5.55
8000000	9600	1	52	1	0x49	-0.08	0.04	-0.1	0.14
8000000	19200	1	26	0	0xB6	-0.08	0.16	-0.28	0.2
8000000	38400	1	13	0	0x84	-0.32	0.32	-0.64	0.48
8000000	57600	1	8	10	0xF7	-0.32	0.32	-1	0.36
8000000	115200	1	4	5	0x55	-0.8	0.64	-1.12	1.76
8000000	230400	1	2	2	0xBB	-1.44	1.28	-3.92	1.68
8000000	460800	0	17	-	0x4A	-2.72	2.56	-3.76	7.28
8388608	9600	1	54	9	0xEE	-0.06	0.06	-0.11	0.13
8388608	19200	1	27	4	0xFB	-0.11	0.1	-0.33	0
8388608	38400	1	13	10	0x55	-0.21	0.21	-0.55	0.33
8388608	57600	1	9	1	0xB5	-0.31	0.31	-0.53	0.78
8388608	115200	1	4	8	0xEE	-0.75	0.74	-2	0.87
8388608	230400	1	2	4	0x92	-1.62	1.37	-3.56	2.06
8388608	460800	0	18	-	0x11	-2	3.37	-5.31	5.55
12000000	9600	1	78	2	0x0	0	0	0	0.04

Table 18-5. Recommended Settings for Typical Crystals and Baudrates (continued)

BRCLK	Baudrate	UCOS16	UCBRx	UCFx	UCSx	TX error (%)		RX error (%)	
						neg	pos	neg	pos
12000000	19200	1	39	1	0x0	0	0	0	0.16
12000000	38400	1	19	8	0x65	-0.16	0.16	-0.4	0.24
12000000	57600	1	13	0	0x25	-0.16	0.32	-0.48	0.48
12000000	115200	1	6	8	0x20	-0.48	0.64	-1.04	1.04
12000000	230400	1	3	4	0x2	-0.8	0.96	-1.84	1.84
12000000	460800	1	1	10	0x0	0	1.76	0	3.44
16000000	9600	1	104	2	0xD6	-0.04	0.02	-0.09	0.03
16000000	19200	1	52	1	0x49	-0.08	0.04	-0.1	0.14
16000000	38400	1	26	0	0xB6	-0.08	0.16	-0.28	0.2
16000000	57600	1	17	5	0xDD	-0.16	0.2	-0.3	0.38
16000000	115200	1	8	10	0xF7	-0.32	0.32	-1	0.36
16000000	230400	1	4	5	0x55	-0.8	0.64	-1.12	1.76
16000000	460800	1	2	2	0xBB	-1.44	1.28	-3.92	1.68
16777216	9600	1	109	3	0xB5	-0.03	0.02	-0.05	0.06
16777216	19200	1	54	9	0xEE	-0.06	0.06	-0.11	0.13
16777216	38400	1	27	4	0xFB	-0.11	0.1	-0.33	0
16777216	57600	1	18	3	0x44	-0.16	0.15	-0.2	0.45
16777216	115200	1	9	1	0xB5	-0.31	0.31	-0.53	0.78
16777216	230400	1	4	8	0xEE	-0.75	0.74	-2	0.87
16777216	460800	1	2	4	0x92	-1.62	1.37	-3.56	2.06
20000000	9600	1	130	3	0x25	-0.02	0.03	0	0.07
20000000	19200	1	65	1	0xD6	-0.06	0.03	-0.1	0.1
20000000	38400	1	32	8	0xEE	-0.1	0.13	-0.27	0.14
20000000	57600	1	21	11	0x22	-0.16	0.13	-0.16	0.38
20000000	115200	1	10	13	0xAD	-0.29	0.26	-0.46	0.66
20000000	230400	1	5	6	0xEE	-0.67	0.51	-1.71	0.62
20000000	460800	1	2	11	0x92	-1.38	0.99	-1.84	2.8

18.3.14 Using the eUSCI_A Module in UART Mode With Low-Power Modes

The eUSCI_A module provides automatic clock activation for use with low-power modes. When the eUSCI_A clock source is inactive because the device is in a low-power mode, the eUSCI_A module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the eUSCI_A module returns to its idle condition. After the eUSCI_A module returns to the idle condition, control of the clock source reverts to the settings of its control bits.

18.3.15 eUSCI_A Interrupts

The eUSCI_A has only one interrupt vector that is shared for transmission and for reception.

18.3.15.1 eUSCI_A 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.

18.3.15.2 eUSCI_A 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 do not set UCRXIFG.
- When UCDORM = 1, nonaddress characters do not set UCRXIFG in multiprocessor modes. In plain UART mode, no characters are set UCRXIFG.
- When UCBRKIE = 1, a break condition sets the UCBRK bit and the UCRXIFG flag.

18.3.15.3 eUSCI_A Receive Interrupt Operation

Table 18-6 describes the I^2C state change interrupt flags.

Table 18-6. UART State Change Interrupt Flags

Interrupt Flag	Interrupt Condition
UCSTTIFG	START byte received interrupt. This flag is set when the UART module receives a START byte.
UCTXCPTIFG	Transmit complete interrupt. This flag is set, after the complete UART byte in the internal shift register including STOP bit got shifted out and UCAxTXBUF is empty.

18.3.15.4 UCAxIV, Interrupt Vector Generator

The eUSCI_A 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.

Read access of the UCAxIV register automatically resets the highest-pending Interrupt condition and flag. Write access of the UCAxIV register clears all pending Interrupt conditions and flags. If another interrupt flag is set, another interrupt is generated immediately after servicing the initial interrupt.

Example 18-1 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 eUSCI_A0.

Example 18-1. UCAxIV Software Example

```
#pragma vector = USCI_A0_VECTOR __interrupt void USCI_A0_ISR(void) {
    switch(__even_in_range(UCA0IV,18)) {
        case 0x00:      // Vector 0: No interrupts
                 break;
        case 0x02: ...  // Vector 2: UCSTTIFG
                 break;
        case 0x04: ...  // Vector 4: UCTXCPTIFG
                 break;
        case 0x06: ...  // Vector 6: UCTXIFG
                 break;
        case 0x08: ...  // Vector 8: UCRXIFG
                 break;
        default: break;
    }
}
```

18.4 eUSCI_A Registers – UART Mode

The eUSCI_A registers applicable in UART mode listed in Table 18-7. The base address can be found in the device-specific data sheet. The address offsets are listed in Table 18-7.

Table 18-7. eUSCI_Ax Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
eUSCI_Ax Control Word 0	UCAxCTLW0	Read/write	Word	00h	0001h
eUSCI_Ax Control 0	UCAxCTL0[1]	Read/write	Byte	01h	00h
eUSCI_Ax Control 1	UCAxCTL1	Read/write	Byte	00h	01h
eUSCI_Ax Control Word 1	UCAxCTLW1	Read/write	Word	02h	0003h
eUSCI_Ax Baud Rate Control Word	UCAxBRW	Read/write	Word	06h	0000h
eUSCI_Ax Baud Rate Control 0	UCAxBR0[1]	Read/write	Byte	06h	00h
eUSCI_Ax Baud Rate Control 1	UCAxBR1	Read/write	Byte	07h	00h
eUSCI_Ax Modulation Control Word	UCAxMCTLW	Read/write	Word	08h	00h
eUSCI_Ax Status	UCAxSTATW	Read/write	Word	0Ah	00h
eUSCI_Ax Receive Buffer	UCAxRXBUF	Read/write	Word	0Ch	00h
eUSCI_Ax Transmit Buffer	UCAxTXBUF	Read/write	Word	0Eh	00h
eUSCI_Ax Auto Baud Rate Control	UCAxABCTL	Read/write	Word	10h	00h
eUSCI_Ax IrDA Control	UCAxIRCTL	Read/write	Word	12h	0000h
eUSCI_Ax IrDA Transmit Control	UCAxIRTCTL	Read/write	Byte	12h	00h
eUSCI_Ax IrDA Receive Control	UCAxIRRCTL	Read/write	Byte	13h	00h
eUSCI_Ax Interrupt Enable	UCAxIE	Read/write	Word	1Ah	00h
eUSCI_Ax Interrupt Flag	UCAxIFG	Read/write	Word	1Ch	00h
eUSCI_Ax Interrupt Vector	UCAxIV	Read	Word	1Eh	0000h

(1) It is recommended to access these registers using 16-bit access. If 8-bit access is used, the corresponding bit names must be followed by "_H"

eUSCI_Ax Control Word Register 0 (UCAxCTLW0)

15	14	13	12	11	10	9	8
UCPEN	**UCPAR**	**UCMSB**	**UC7BIT**	**UCSPB**	**UCMODEx**		**UCSYNC = 0**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
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

UCPEN Bits 15 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 14 Parity select. UCPAR is not used when parity is disabled.

0 Odd parity

1 Even parity

UCMSB Bit 13 MSB first select. Controls the direction of the receive and transmit shift register.

0 LSB first

1 MSB first

UC7BIT Bit 12 Character length. Selects 7-bit or 8-bit character length.

0 8-bit data

1 7-bit data

UCSPB Bit 11 Stop bit select. Number of stop bits.

0 One stop bit

1 Two stop bits

UCMODEx Bits 10-9 eUSCI_A 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 8 Synchronous mode enable

0 Asynchronous mode

1 Synchronous mode

UCSSELx Bits 7-6 eUSCI_A 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 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 eUSCI_A into sleep mode.

0 Not dormant. All received characters set UCRXIFG.

1 Dormant. Only characters that are preceded by an idle-line or with address bit set UCRXIFG. In UART mode with automatic baud-rate detection, only the combination of a break and synch field sets UCRXIFG.

UCTXADDR Bit 2 Transmit address. Next frame to be transmitted is marked as address, depending on the selected multiprocessor mode.

0 Next frame transmitted is data.

1 Next frame transmitted is an address.

(continued)

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. eUSCI_A reset released for operation. 1 Enabled. eUSCI_A logic held in reset state.

eUSCI_Ax Control Word Register 1 (UCAxCTLW1)

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						**UCGLITx**	
r-0	r-0	r-0	r-0	r-0	r-0	rw-1	rw-1

Reserved	Bits 15-2	Reserved.
UCGLITx	Bits 1-0	Deglitch time 00 ~2 ns 01 ~50 ns 10 ~100 ns 11 ~200 ns

eUSCI_Ax Baud Rate Control Word (UCAxBRW)

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

UCBRx	Bits 15	Clock prescaler setting of the Baud rate generator

eUSCI_Ax Modulation Control Word (UCAxMCTLW)

15	14	13	12	11	10	9	8
UCBRSx							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
UCBRFx				**Reserved**			**UCOS16**
rw-0	rw-0	rw-0	rw-0	r0	r0	r0	rw-0

UCBRSx	Bits 15-8	Second modulation stage select. These bits hold a free modulation pattern for BITCLK.
UCBRFx	Bits 7-4	First modulation stage select. These bits determine the modulation pattern for BITCLK16 when UCOS16 = 1. Ignored with UCOS16 = 0. Table 18-3 shows the modulation pattern.
UCOS16	Bit 0	Oversampling mode enabled 0 Disabled 1 Enabled

eUSCI_Ax Status Register (UCAxSTAT)

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
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

Reserved	Bits 15-8	Reserved
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 does 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, or 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. UCADDR is cleared when UCAxRXBUF is read. 0 Received character is data. 1 Received character is an address.
UCIDLE		Idle line detected in idle-line multiprocessor mode. UCIDLE is cleared when UCAxRXBUF is read. 0 No idle line detected 1 Idle line detected
UCBUSY	Bit 0	eUSCI_A busy. This bit indicates if a transmit or receive operation is in progress. 0 eUSCI_A inactive 1 eUSCI_A transmitting or receiving

eUSCI_Ax Receive Buffer Register (UCAxRXBUF)

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
UCRXBUFx							
r	r	r	r	r	r	r	r

Reserved	Bits 15-8	Reserved
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.

eUSCI_Ax Transmit Buffer Register (UCAxTXBUF)

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
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.

eUSCI_Ax Auto Baud Rate Control Register (UCAxABCTL)

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		UCDELIMx		UCSTOE	UCBTOE	Reserved	UCABDEN
r-0	r-0	rw-0	rw-0	rw-0	rw-0	r-0	rw-0

Reserved	Bits 15-6	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.

eUSCI_Ax IrDA Control Word (UCAxIRTCTL)

15	14	13	12	11	10	9	8
UCIRRXFLx						**UCIRRXPL**	**UCIRRXFE**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
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

UCIRRXFLx	Bits 15-10	Receive filter length. The minimum pulse length for receive is given by: $t_{MIN} = (UCIRRXFLx + 4) / (2 \times f_{IRTXCLK})$
UCIRRXPL	Bit 9	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 8	IrDA receive filter enabled
		0 Receive filter disabled
		1 Receive filter enabled
UCIRTXPLx	Bits 7-2	Transmit pulse length Pulse length $t_{PULSE} = (UCIRTXPLx + 1) / (2 \times f_{IRTXCLK})$
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

eUSCI_Ax Interrupt Enable Register (UCAxIE)

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				**UCTXCPTIE**	**UCSTTIE**	**UCTXIE**	**UCRXIE**
r-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0

Reserved	Bits 15-4	Reserved
UCTXCPTIE	Bit 2	Transmit complete interrupt enable
		0 Interrupt disabled
		1 Interrupt enabled
UCSTTIE		Start bit 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

eUSCI_Ax Interrupt Flag Register (UCAxIFG)

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				UCTXCPTIFG	UCSTTIFG	UCTXIFG	UCRXIFG
r-0	r-0	r-0	r-0	rw-0	rw-0	rw-0	rw-0

Reserved Bits 15-4 Reserved.

UCTXCPTIFG Bit 3 Transmit ready interrupt enable. UCTXRDYIFG is set when the entire byte in the internal shift register got shifted out and UCAxTXBUF is empty.

0 No interrupt pending

1 Interrupt pending

UCSTTIFG Bit 2 Start bit interrupt flag. UCSTTIFG is set after a Start bit was received

0 No interrupt pending

1 Interrupt pending

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

eUSCI_Ax Interrupt Vector Register (UCAxIV)

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 eUSCI_A interrupt vector value

UCAxIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	Receive buffer full	UCRXIFG	Highest
04h	Transmit buffer empty	UCTXIFG	
06h	Start bit received	UCSTTIFG	
08h	Transmit complete	UCTXCPTIFG	Lowest

Enhanced Universal Serial Communication Interface (eUSCI) – SPI Mode

The enhanced universal serial communication interfaces, eUSCI_A and eUSCI_B, support multiple serial communication modes with one hardware module. This chapter discusses the operation of the synchronous peripheral interface (SPI) mode.

Topic **Page**

19.1 Enhanced Universal Serial Communication Interfaces (eUSCI_A, eUSCI_B) Overview

Both the eUSCI_A and the eUSCI_B support serial communication in SPI mode.

19.2 eUSCI Introduction – SPI Mode

In synchronous mode, the eUSCI connects the device 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-bit 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 19-1 shows the eUSCI when configured for SPI mode.

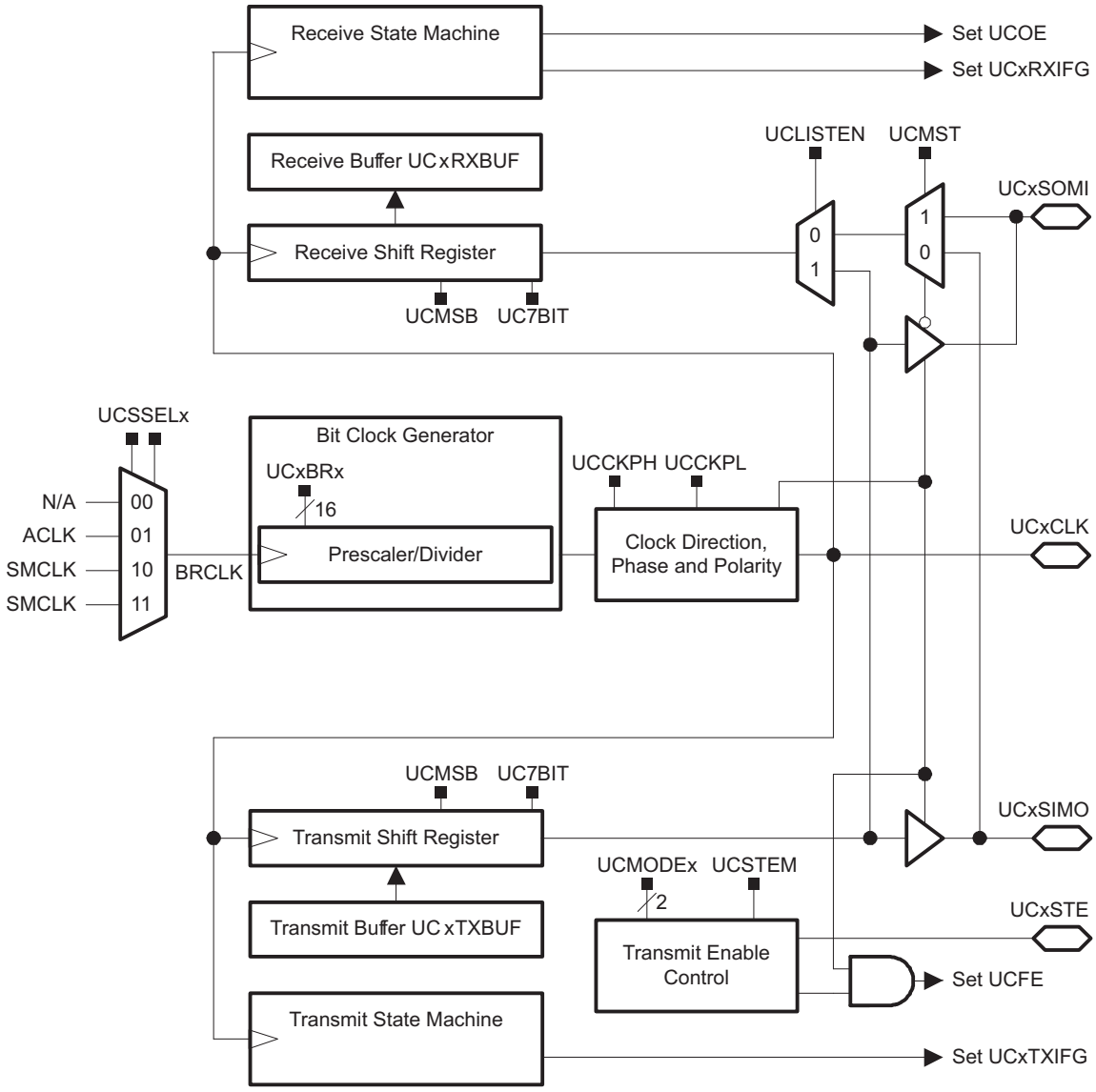


Figure 19-1. eUSCI Block Diagram – SPI Mode

19.3 eUSCI 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 controlled by the master, UCxSTE, is provided to enable a device to receive and transmit data.

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 – eUSCI 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 19-1 describes the UCxSTE operation.

Table 19-1. 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

19.3.1 eUSCI Initialization and Reset

The eUSCI is reset by a PUC or by the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the eUSCI 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 eUSCI for operation.

NOTE: Initializing or reconfiguring the eUSCI module

The recommended eUSCI initialization/reconfiguration process is:

1. Set UCSWRST (`BIS.B #UCSWRST,&UCxCTL1`).
2. Initialize all eUSCI 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.

19.3.2 Character Format

The eUSCI module in SPI mode supports 7-bit 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, MSB-first mode may be required.

NOTE: Character format for figures

Figures throughout this chapter use MSB-first format.

19.3.3 Master Mode

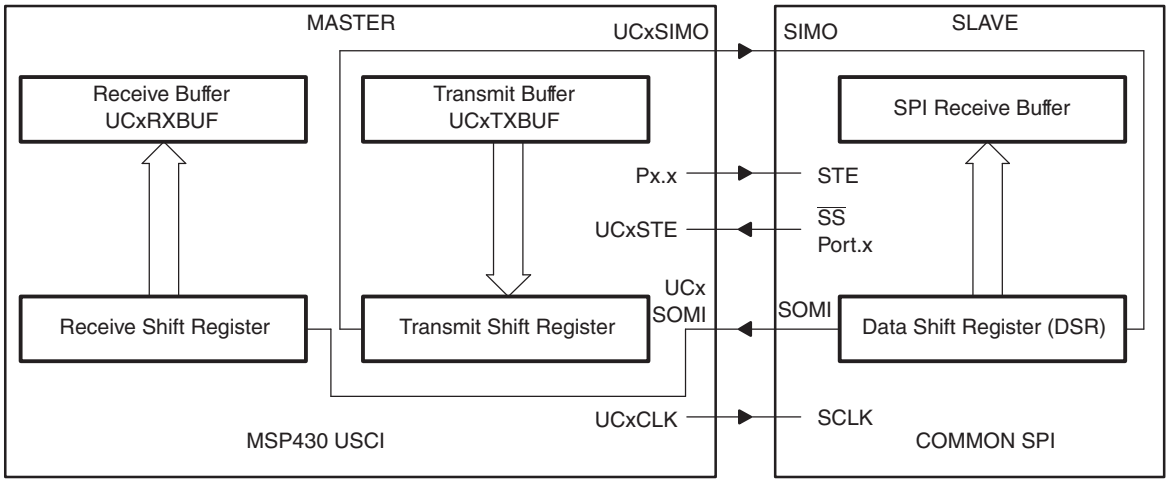


Figure 19-2. eUSCI Master and External Slave

Figure 19-2 shows the eUSCI as a master in both 3-pin and 4-pin configurations. The eUSCI initiates data transfer when data is moved to the transmit data buffer UCxTXBUF. The UCxTXBUF data is moved to the transmit (TX) shift register when the TX shift register is empty, initiating data transfer on UCxSIMO starting with either the MSB or LSB, 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 receive (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 eUSCI in master mode, data must be written to UCxTXBUF, because receive and transmit operations operate concurrently.

There two different options for configuring the eUSCI as a 4-pin master, which are described in the next sections:

- The fourth pin is used to prevent conflicts with other masters (UCSTEM = 0).
- The fourth pin is used to generate a slave enable signal (UCSTEM = 1).

The bit UCSTEM is used to select the corresponding mode.

19.3.3.1 4-Pin SPI Master Mode (UCSTEM = 0)

In 4-pin master mode with UCSTEM = 0, UCxSTE can be used to prevent conflicts with another master and controls the master as described in Table 19-1. When UCxSTE is in the master-inactive state and UCSTEM = 0:

- 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 is 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 rewritten 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.

19.3.3.2 4-Pin SPI Master Mode (UCSTEM = 1)

If UCSTEM = 1 in 4-pin master mode, the slave enable signal for a single slave is automatically generated. The corresponding behavior can be seen in Figure 19-4.

If multiple slaves are desired, this feature is not applicable and the software needs to use general purpose I/O pins instead.

19.3.4 Slave Mode

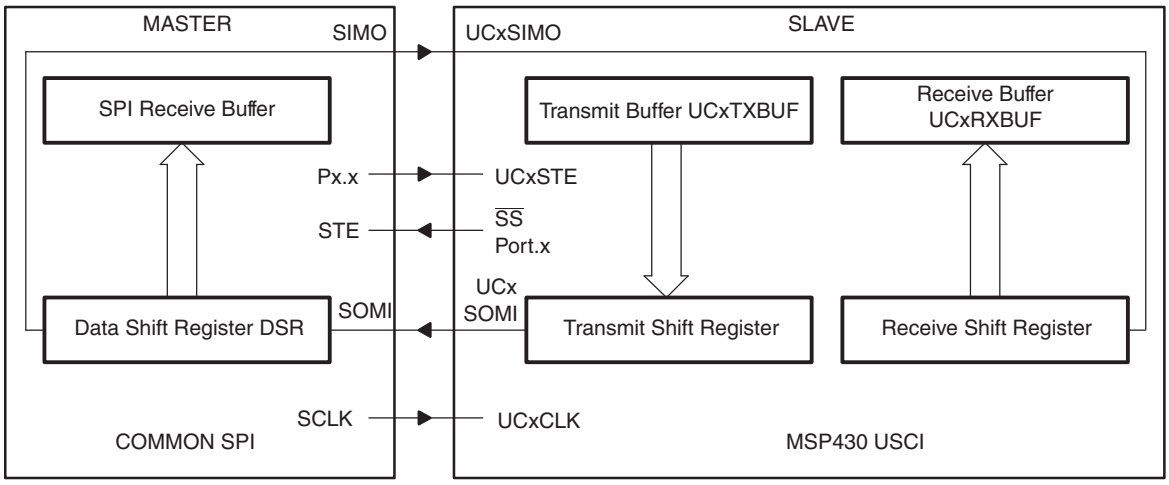


Figure 19-3. eUSCI Slave and External Master

Figure 19-3 shows the eUSCI as a slave in both 3-pin and 4-pin configurations. UCxCLK is used as the input for the SPI clock and must be supplied by the external master. The data-transfer rate is determined by this clock and not by the internal bit clock generator. Data written to UCxTXBUF and moved to the TX shift register before the start of UCxCLK is transmitted on UCxSOMI. Data on UCxSIMO is shifted into the receive shift register on the opposite edge of UCxCLK and moved to UCxRXBUF when the set number of bits are received. When data is moved from the RX shift register to UCxRXBUF, the 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.

19.3.4.1 4-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.

19.3.5 SPI Enable

When the eUSCI 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 eUSCI immediately and any active transfer is terminated.

19.3.5.1 Transmit Enable

In master mode, writing to UCxTXBUF activates the bit clock generator, and the data begins 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.

19.3.5.2 Receive Enable

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

19.3.6 Serial Clock Control

UCxCLK is provided by the master on the SPI bus. When UCMST = 1, the bit clock is provided by the eUSCI 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 eUSCI 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 eUSCI 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 eUSCI_A. The UCAxCLK/UCBxCLK frequency is given by:

$f_{BitClock} = f_{BRCLK}/UCBRx$

19.3.6.1 Serial Clock Polarity and Phase

The polarity and phase of UCxCLK are independently configured via the UCCKPL and UCCKPH control bits of the eUSCI. Timing for each case is shown in Figure 19-4.

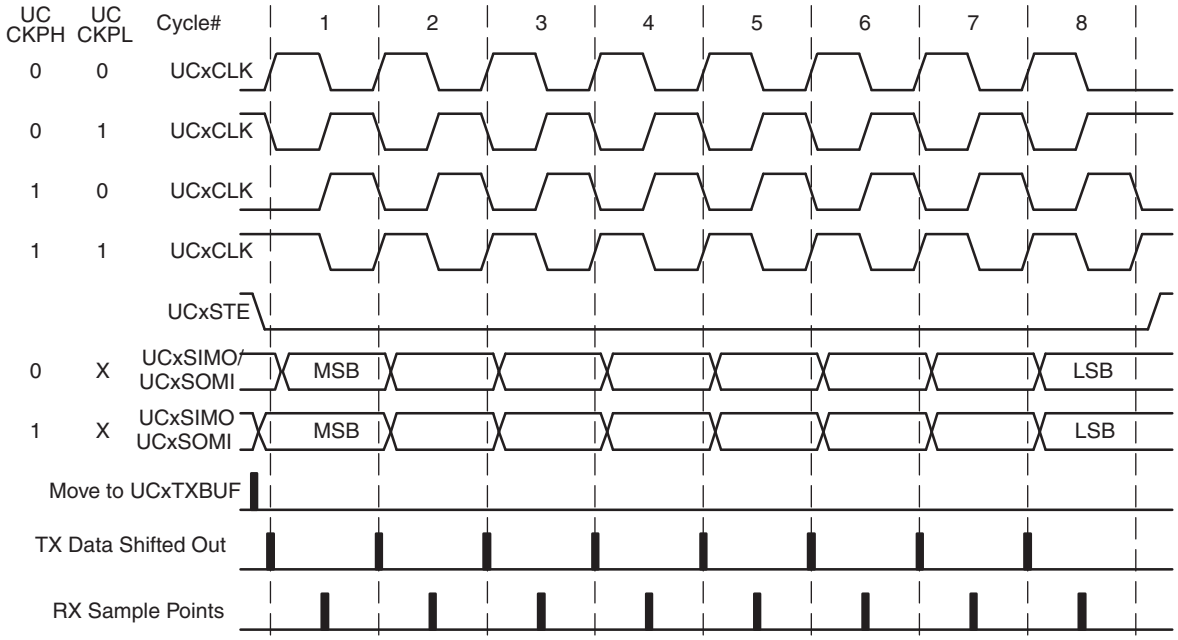


Figure 19-4. eUSCI SPI Timing With UCMSB = 1

19.3.7 Using the SPI Mode With Low-Power Modes

The eUSCI module provides automatic clock activation for use with low-power modes. When the eUSCI clock source is inactive because the device is in a low-power mode, the eUSCI module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the eUSCI module returns to its idle condition. After the eUSCI 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 eUSCI 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.

When receiving multiple bytes as a slave in LPM4 the wakeup time of the CPU needs to be considered. If the wake-up time of the CPU is, for example, 150 µs (see device-specific data-sheet), it needs to be ensured that the CPU serves the TXIFG of the first received byte before the second byte is completely received by the eUSCI_A or eUSCI_B. Otherwise an overrun error occurs.

19.3.8 SPI Interrupts

The eUSCI has only one interrupt vector that is shared for transmission and for reception. eUSCI_Ax and eUSCI_Bx do not share the same interrupt vector.

19.3.8.1 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 UCxTXBUF when UCTXIFG = 0 may result in erroneous data transmission.

19.3.8.2 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.

19.3.8.3 UCxIV, Interrupt Vector Generator

The eUSCI 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 (PC) 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.

19.3.8.3.1 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 eUSCI_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
```

19.4 eUSCI Registers – SPI Mode

The eUSCI registers applicable in SPI mode are listed in Table 19-2 and Table 19-3. The base addresses can be found in the device-specific data sheet. The address offsets are listed in Table 19-2 and Table 19-3.

Table 19-2. eUSCI_Ax Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
eUSCI_Ax Control Word 0	UCAxCTLW0	Read/write	Word	00h	0001h
eUSCI_Ax Control 1	UCAxCTL1	Read/write	Byte	00h	01h
eUSCI_Ax Control 0	UCAxCTL0	Read/write	Byte	01h	00h
eUSCI_Ax Bit Rate Control Word	UCAxBRW	Read/write	Word	06h	0000h
eUSCI_Ax Bit Rate Control 0	UCAxBR0	Read/write	Byte	06h	00h
eUSCI_Ax Bit Rate Control 1	UCAxBR1	Read/write	Byte	07h	00h
eUSCI_Ax Status	UCAxSTATW	Read/write	Word	0Ah	00h
eUSCI_Ax Receive Buffer	UCAxRXBUF	Read/write	Word	0Ch	00h
eUSCI_Ax Transmit Buffer	UCAxTXBUF	Read/write	Word	0Eh	00h
eUSCI_Ax Interrupt Enable	UCAxIE	Read/write	Word	1Ah	00h
eUSCI_Ax Interrupt Flag	UCAxIFG	Read/write	Word	1Ch	02h
eUSCI_Ax Interrupt Vector	UCAxIV	Read	Word	1Eh	0000h

Table 19-3. eUSCI_Bx Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
eUSCI_Bx Control Word 0	UCBxCTLW0	Read/write	Word	00h	01C1h
eUSCI_Bx Control 1	UCBxCTL1	Read/write	Byte	00h	C1h
eUSCI_Bx Control 0	UCBxCTL0	Read/write	Byte	01h	01h
eUSCI_Bx Bit Rate Control Word	UCBxBRW	Read/write	Word	06h	0000h
eUSCI_Bx Bit Rate Control 0	UCBxBR0	Read/write	Byte	06h	00h
eUSCI_Bx Bit Rate Control 1	UCBxBR1	Read/write	Byte	07h	00h
eUSCI_Bx Status	UCBxSTATW	Read/write	Word	08h	00h
eUSCI_Bx Receive Buffer	UCBxRXBUF	Read/write	Word	0Ch	00h
eUSCI_Bx Transmit Buffer	UCBxTXBUF	Read/write	Word	0Eh	00h
eUSCI_Bx Interrupt Enable	UCBxIE	Read/write	Word	2Ah	00h
eUSCI_Bx Interrupt Flag	UCBxIFG	Read/write	Word	2Ch	02h
eUSCI_Bx Interrupt Vector	UCBxIV	Read	Word	2Eh	0000h

eUSCI_Ax Control Register 0 (UCAxCTLW0)
eUSCI_Bx Control Register 0 (UCBxCTLW0)

15	14	13	12	11	10	9	8
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]

7	6	5	4	3	2	1	0
UCSSELx		**Reserved**				**UCSTEM**	**UCSWRST**
rw-0[1] rw-1[2]	rw-0[1] rw-1[2]	rw-0[1] r0[2]	rw-0	rw-0	rw-0	rw-0	rw-1

UCCKPH	Bit 15	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 14	Clock polarity select 0 The inactive state is low. 1 The inactive state is high.
UCMSB	Bit 13	MSB first select. Controls the direction of the receive and transmit shift register. 0 LSB first 1 MSB first
UC7BIT	Bit 12	Character length. Selects 7-bit or 8-bit character length. 0 8-bit data 1 7-bit data
UCMST	Bit 11	Master mode select 0 Slave mode 1 Master mode
UCMODEx	Bits 10-9	eUSCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. 00 3-pin SPI 01 4-pin SPI with UCxSTE active high: Slave enabled when UCxSTE = 1 10 4-pin SPI with UCxSTE active low: Slave enabled when UCxSTE = 0 11 I^2C mode
UCSYNC	Bit 8	Synchronous mode enable 0 Asynchronous mode 1 Synchronous mode
UCSSELx	Bits 7-6	eUSCI 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
Reserved	Bits 5-2	reserved
UCSTEM	Bit 1	STE mode select in master mode. This byte is ignored in slave or 3-wire mode. 0 STE-pin is used to prevent conflicts with other masters 1 STE-pin is used to generate the enable signal for a 4-wire slave
UCSWRST	Bit 0	Software reset enable 0 Disabled. eUSCI reset released for operation. 1 Enabled. eUSCI logic held in reset state.

(1) UCAxCTL0 (eUSCI_Ax)
(2) UCBxCTL0 (eUSCI_Bx)
(1) UCAxCTL0 (eUSCI_Ax)
(2) UCBxCTL0 (eUSCI_Bx)

eUSCI_Ax Bit Rate Control Register 1 (UCAxBRW)
eUSCI_Bx Bit Rate Control Register 1 (UCBxBRW)

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

UCBRx	Bits 7-0	Bit clock prescaler setting.

eUSCI_Ax Modulation Control Register (UCAxMCTL)

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
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 15-0	Write as 0

eUSCI_Ax Status Register (UCAxSTATW)
eUSCI_Bx Status Register (UCBxSTATW)

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
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

Reserved	Bits 15-8	Reserved
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 does not function correctly. 0 No error 1 Overrun error occurred.
Unused	Bits 4-1	Unused
UCBUSY	Bit 0	eUSCI busy. This bit indicates if a transmit or receive operation is in progress. 0 eUSCI inactive 1 eUSCI transmitting or receiving

(1) UCAxSTAT (eUSCI_Ax)
(2) UCBxSTAT (eUSCI_Bx)

eUSCI_Ax Receive Buffer Register (UCAxRXBUF)
eUSCI_Bx Receive Buffer Register (UCBxRXBUF)

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
UCRXBUFx							
r	r	r	r	r	r	r	r

Reserved	Bits 15-8	Reserved
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.

eUSCI_Ax Transmit Buffer Register (UCAxTXBUF)
eUSCI_Bx Transmit Buffer Register (UCBxTXBUF)

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
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

Reserved	Bits 15-8	Reserved
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.

eUSCI_Ax Interrupt Enable Register (UCAxIE)
eUSCI_Bx Interrupt Enable Register (UCBxIE)

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
Reserved						**UCTXIE**	**UCRXIE**
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

Reserved	Bits 15-2	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

eUSCI_Ax Interrupt Flag Register (UCAxIFG)
eUSCI_Bx Interrupt Flag Register (UCBxIFG)

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
Reserved						**UCTXIFG**	**UCRXIFG**
r-0	r-0	r-0	r-0	r-0	r-0	rw-1	rw-0

Reserved	Bits 15-2	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

eUSCI_Ax Interrupt Vector Register (UCAxIV)
eUSCI_Bx Interrupt Vector Register (UCBxIV)

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 eUSCI 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

Enhanced Universal Serial Communication Interface (eUSCI) – I^2C Mode

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

Topic **Page**

20.1 Enhanced Universal Serial Communication Interface B (eUSCI_B) Overview

The eUSCI_B module supports two serial communication modes:

- I^2C mode
- SPI mode

If more than one eUSCI_B module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two eUSCI_B modules, they are named eUSCI0_B and eUSCI1_B.

20.2 eUSCI_B Introduction – I^2C Mode

In I^2C mode, the eUSCI_B module provides an interface between the device and I^2C-compatible devices connected by 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 eUSCI_B module through the 2-wire I^2C interface.

The eUSCI_B I^2C mode features include:

- 7-bit and 10-bit device addressing modes
- General call
- START/RESTART/STOP
- Multi-master transmitter/receiver mode
- Slave receiver/transmitter mode
- Standard mode up to 100 kbps and fast mode up to 400 kbps support
- Programmable UCxCLK frequency in master mode
- Designed for low power
- 8-bit byte counter with interrupt capability and automatic STOP assertion
- Up to four hardware slave addresses, each having its own interrupt and DMA trigger
- Mask register for slave address and address received interrupt
- Clock low timeout interrupt to avoid bus stalls
- Slave operation in LPM4
- Slave receiver START detection for auto wake-up from LPMx modes (not LPM3.5 and LPM4.5)

Figure 20-1 shows the eUSCI_B when configured in I^2C mode.

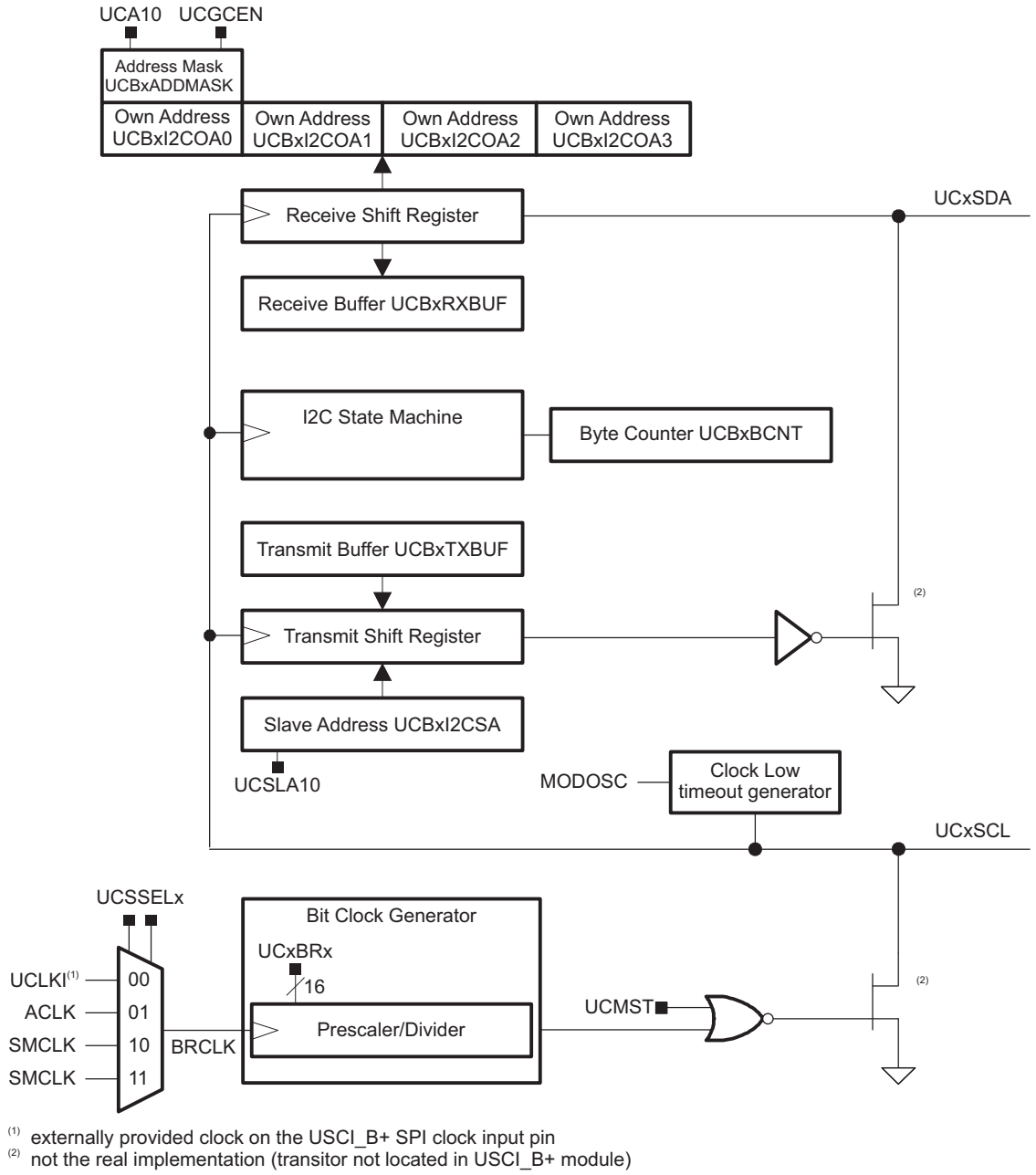


Figure 20-1. eUSCI_B Block Diagram – I²C Mode

20.3 eUSCI_B Operation – I²C Mode

The I²C mode supports any slave or master I²C-compatible device. Figure 20-2 shows an example of an I²C bus. Each I²C device is recognized by a unique address and can operate as either a transmitter or a receiver. A device connected to the I²C bus can be considered as the master or the slave when performing data transfers. A master initiates a data transfer and generates the clock signal SCL. Any device addressed by a master is considered a slave.

I²C data is communicated using the serial data (SDA) pin and the serial clock (SCL) pin. Both SDA and SCL are bidirectional and must be connected to a positive supply voltage using a pullup resistor.

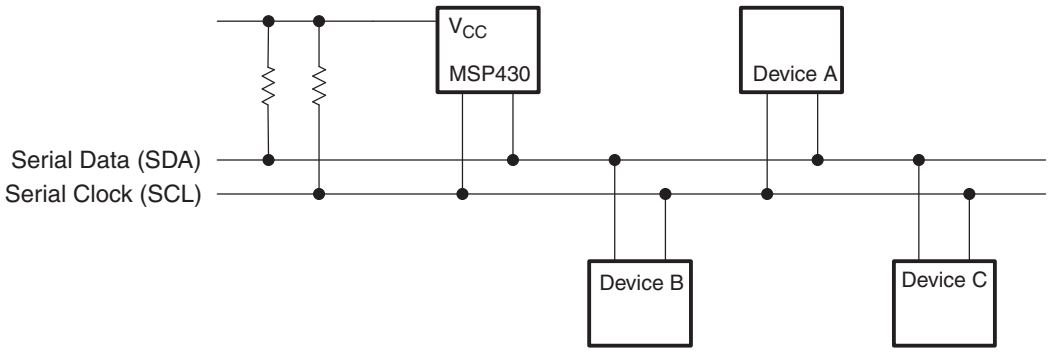


Figure 20-2. I²C Bus Connection Diagram

NOTE: SDA and SCL levels

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

20.3.1 eUSCI_B Initialization and Reset

The eUSCI_B is reset by a PUC or by setting the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the eUSCI_B in a reset condition. To select I²C operation, the UCMODEx bits must be set to 11. After module initialization, it is ready for transmit or receive operation. Clearing UCSWRST releases the eUSCI_B for operation.

Configuring and reconfiguring the eUSCI_B module should be done when UCSWRST is set to avoid unpredictable behavior. Setting UCSWRST in I²C mode has the following effects:

- I²C communication stops.
- SDA and SCL are high impedance.
- UCBxSTAT, bits 15-9 and 6-4 are cleared.
- Registers UCBxIE and UCBxIFG are cleared.
- All other bits and registers remain unchanged.

NOTE: Initializing or re-configuring the eUSCI_B module

The recommended eUSCI_B initialization/reconfiguration process is:

1. Set UCSWRST (`BIS.B #UCSWRST,&UCxCTL1`).
2. Initialize all eUSCI_B registers with UCSWRST = 1 (including UCxCTL1).
3. Configure ports.
4. Clear UCSWRST via software (`BIC.B #UCSWRST,&UCxCTL1`).
5. Enable interrupts (optional).

20.3.2 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 MSB first as shown in Figure 20-3.

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 ninth SCL clock.

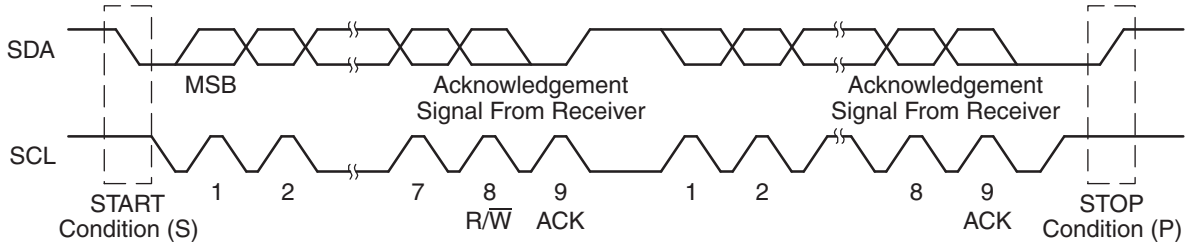


Figure 20-3. I²C Module Data Transfer

START and STOP conditions are generated by the master and are shown in Figure 20-3. 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 (see Figure 20-4). The high and low state of SDA can change only when SCL is low, otherwise START or STOP conditions are generated.

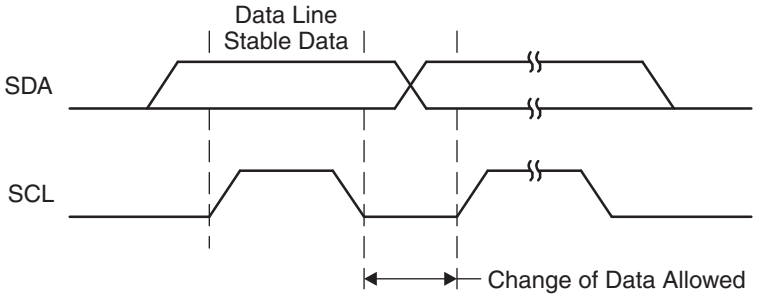


Figure 20-4. Bit Transfer on I²C Bus

20.3.3 I²C Addressing Modes

The I²C mode supports 7-bit and 10-bit addressing modes.

20.3.3.1 7-Bit Addressing

In the 7-bit addressing format (see Figure 20-5), 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.

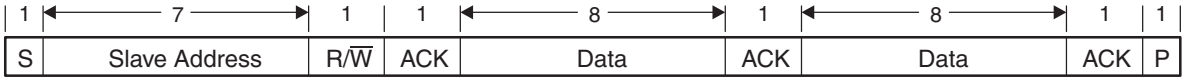


Figure 20-5. I²C Module 7-Bit Addressing Format

20.3.3.2 10-Bit Addressing

In the 10-bit addressing format (see Figure 20-6), 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 eight bits of the 10-bit slave address, followed by the ACK bit and the 8-bit data. See I2C Slave 10-bit Addressing Mode and I2C Master 10-bit Addressing Mode for details how to use the 10-bit addressing mode with the eUSCI_B module.

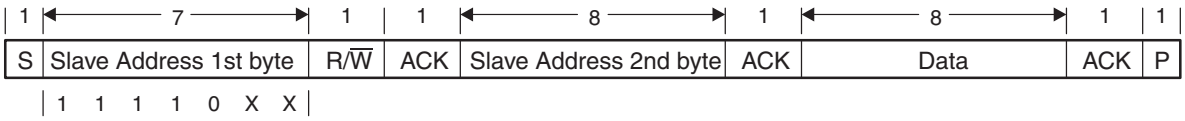


Figure 20-6. I²C Module 10-Bit Addressing Format

20.3.3.3 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 20-7.

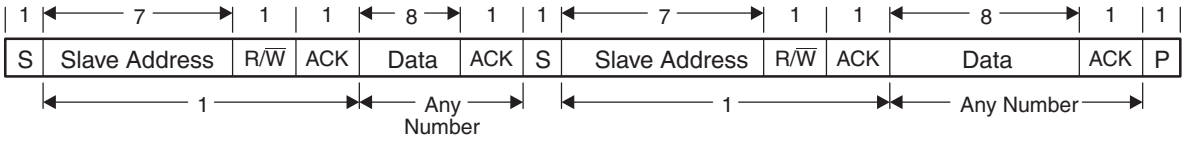


Figure 20-7. I²C Module Addressing Format With Repeated START Condition

20.3.4 I²C Quick Setup

This section gives a quick introduction into the operation of the eUSCI_B in I2C mode. The basic steps to start communication are described and shown as a software example. More detailed information about the possible configurations and details can be found in Section 20.3.5.

The latest code examples can be found on the MSP430 web under "Code Examples".

To set up the eUSCI_B as a master transmitter that transmits to a slave with the address 0x12h, only a few steps are needed (see Example 20-1).

Example 20-1. Master TX With 7-Bit Address

```
UCBxCTL1 |= UCSWRST;           // put eUSCI_B in reset state
UCBxCTLW0 |= UCMODE_3 + UCMST; // I2C master mode
UCBxBRW = 0x0008;              // baudrate = SMCLK / 8
UCBxCTLW1 = UCASTP_2;          // autom. STOP assertion
UCBxTBCNT = 0x07;              // TX 7 bytes of data
UCBxI2CSA = 0x0012;            // address slave is 12hex
P2SEL |= 0x03;                 // configure I2C pins (device specific)
UCBxCTL1 &= ^UCSWRST;          // eUSCI_B in operational state
UCBxIE |= UCTXIE;              // enable TX-interrupt
GIE;                           // general interrupt enable
...
// inside the eUSCI_B TX interrupt service routine
UCBxTXBUF = 0x77;              // fill TX buffer
```

As shown in the code example, all configurations must be done while UCSWRST is set. To select the I²C operation of the eUSCI_B, UCMODE must be set accordingly. The baudrate of the transmission is set by writing the correct divider in the UCBxBRW register. The default clock selected is SMCLK. How many bytes are transmitted in one frame is controlled by the byte counter threshold register UCBxTBCNT together with the UCASTPx bits.

The slave address to send to is specified in the UCBxI2CSA register. Finally, the ports must be configured. This step is device dependent; see the data sheet for the pins that must be used.

Each byte that is to be transmitted must be written to the UCBxTXBUF inside the interrupt service routine. The recommended structure of the interrupt service routine can be found in Example 20-3.

Example 20-2 shows the steps needed to set up the eUSCI_B as a slave with the address 0x12h that is able to receive and transmit data to the master.

Example 20-2. Slave RX With 7-Bit Address

```
UCBxCTL1 |= UCSWRST;         // eUSCI_B in reset state
UCBxCTLW0 |= UCMODE_3;       // I2C slave mode
UCBxI2COA0 = 0x0012;         // own address is 12hex
P2SEL |= 0x03;               // configure I2C pins (device specific)
UCBxCTL1 &= ^UCSWRST;        // eUSCI_B in operational state
UCBxIE |= UCTXIE + UCRXIE;   // enable TX&RX-interrupt
GIE;                         // general interrupt enable
...
// inside the eUSCI_B TX interrupt service routine
UCBxTXBUF = 0x77;            // send 077h
...
// inside the eUSCI_B RX interrupt service routine
data = UCBxRXBUF;            // data is the internal variable
```

As shown in Example 20-2, all configurations must be done while UCSWRST is set. For the slave, I²C operation is selected by setting UCMODE. The slave address is specified in the UCBxI2COA0 register. To enable the interrupts for receive and transmit requests, the according bits in UCBxIE and, at the end, GIE need to be set. Finally the ports must be configured. This step is device dependent; see the data sheet for the pins that are used.

The RX interrupt service routine is called for every byte received by a master device. The TX interrupt service routine is executed each time the master requests a byte. The recommended structure of the interrupt service routine can be found in Example 20-3.

20.3.5 I²C Module Operating Modes

In I²C mode, the eUSCI_B 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 20-8 shows how to interpret the time-line figures. Data transmitted by the master is represented by grey rectangles; data transmitted by the slave is represented by white rectangles. Data transmitted by the eUSCI_B module, either as master or slave, is shown by rectangles that are taller than the others.

Actions taken by the eUSCI_B module are shown in grey rectangles with an arrow indicating where in 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.

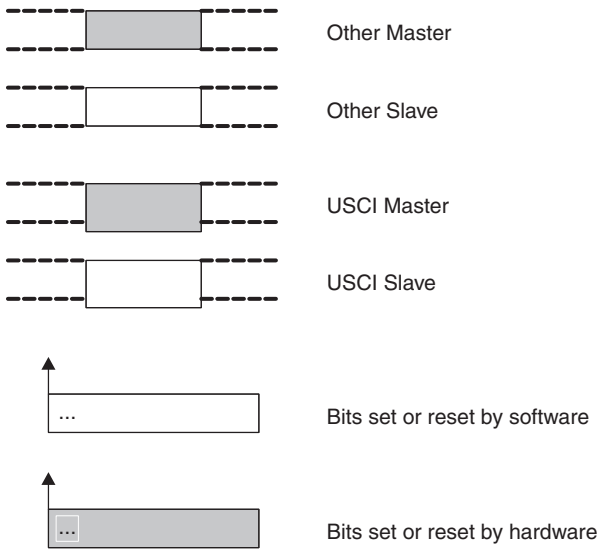


Figure 20-8. I²C Time-Line Legend

20.3.5.1 Slave Mode

The eUSCI_B 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 eUSCI_B module must 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/$\overline{W}$ bit received together with the slave address.

The eUSCI_B slave address is programmed with the UCBxI2COA0 register. Support for multiple slave addresses is explained in . 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 eUSCI_B module receives the transmitted address and compares it against its own address stored in UCBxI2COA0. The UCSTTIFG flag is set when address received matches the eUSCI_B slave address.

20.3.5.1.1 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 does 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 eUSCI_B module is automatically configured as a transmitter and UCTR and UCTXIFG0 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 and the data is transmitted. As soon as the data is transferred into the shift register, the UCTXIFG0 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 followed by a STOP condition, the UCSTPIFG flag is set. If the NACK is followed by a repeated START condition, the eUSCI_B I²C state machine returns to its address-reception state.

Figure 20-9 shows the slave transmitter operation.

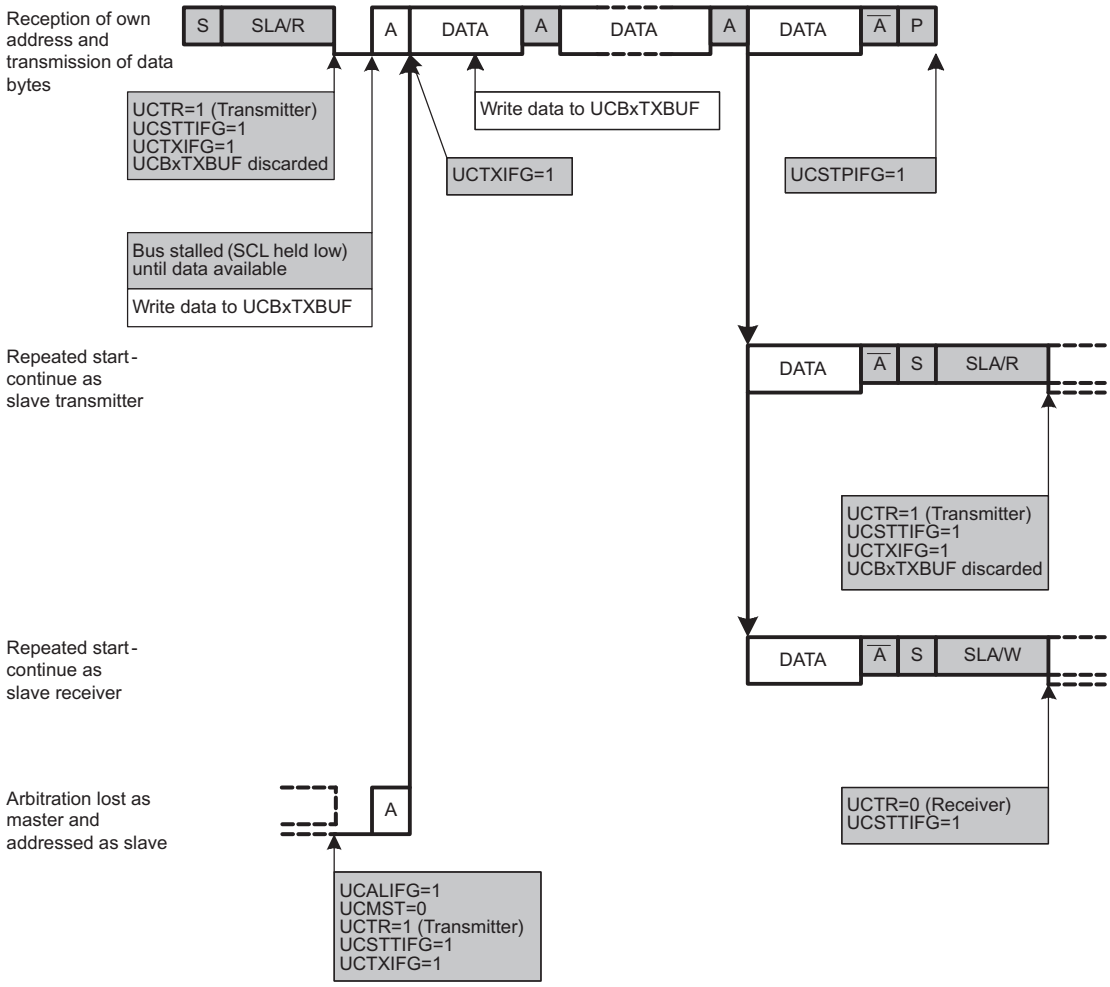


Figure 20-9. I²C Slave Transmitter Mode

20.3.5.1.2 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 receives data from the master, the eUSCI_B module is automatically configured as a receiver and UCTR is cleared. After the first data byte is received, the receive interrupt flag UCRXIFG0 is set. The eUSCI_B module automatically acknowledges the received data and can receive the next data byte.

If the previous data was 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 is released, a NACK is transmitted immediately, and UCBxRXBUF is loaded with the last received data. Because the previous data was not read, that data is lost. To avoid loss of data, the UCBxRXBUF must 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 eUSCI_B I²C state machine returns to its address-reception state.

Figure 20-10 shows the I²C slave receiver operation.

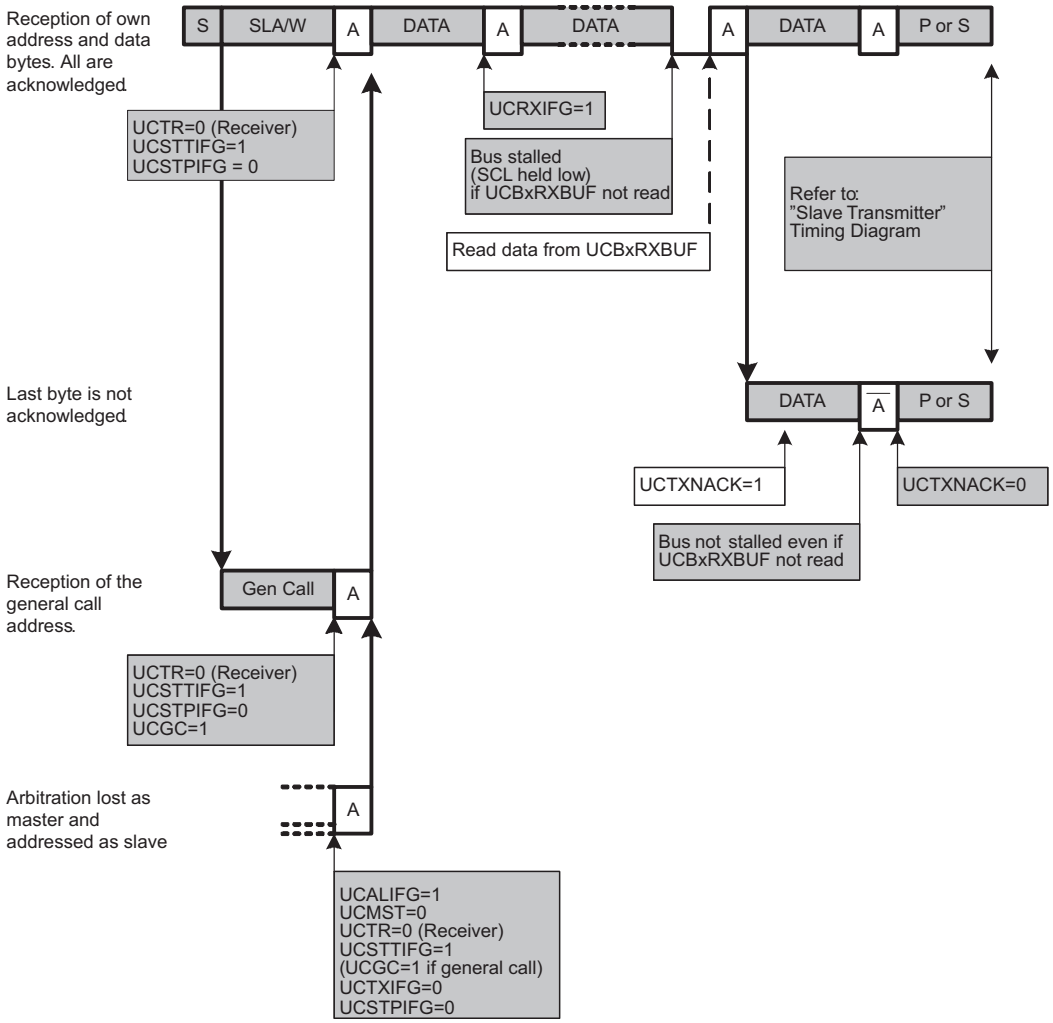


Figure 20-10. I²C Slave Receiver Mode

20.3.5.1.3 I²C Slave 10-Bit Addressing Mode

The 10-bit addressing mode is selected when UCA10 = 1 and is as shown in Figure 20-11. In 10-bit addressing mode, the slave is in receive mode after the full address is received. The eUSCI_B 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 sets the UCSTTIFG flag if it was previously cleared by software, and the eUSCI_B modules switches to transmitter mode with UCTR = 1.

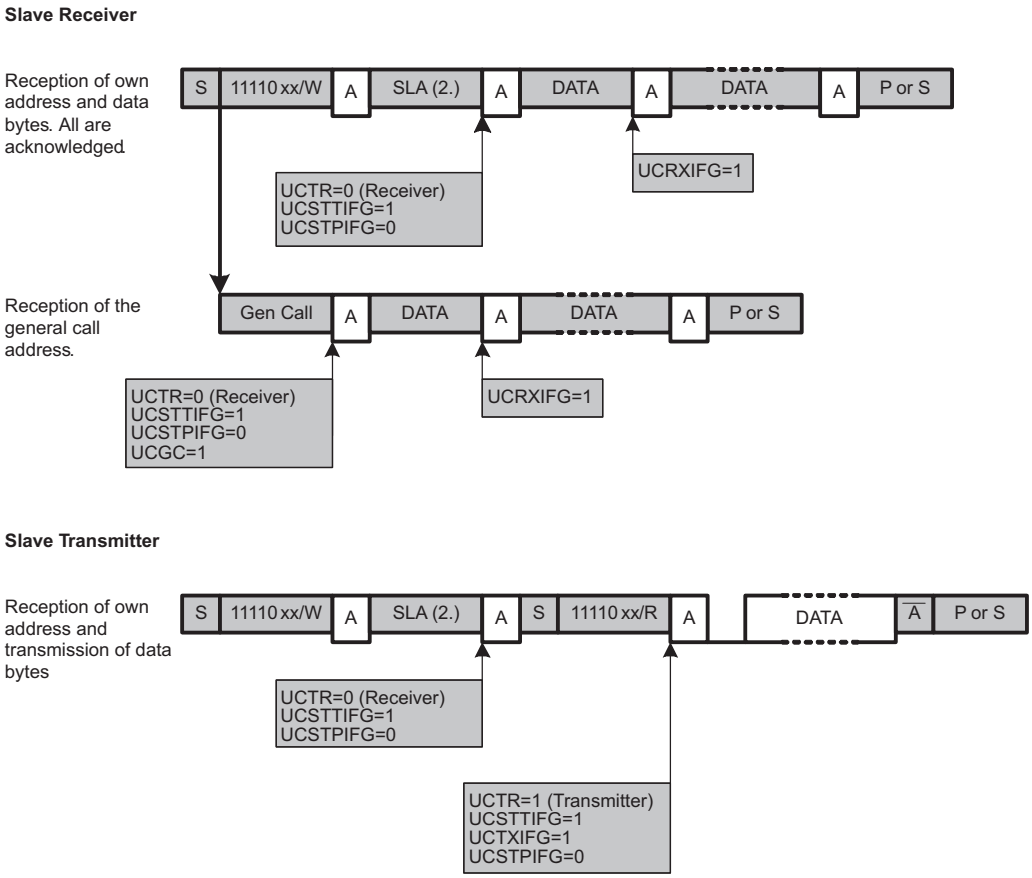


Figure 20-11. I²C Slave 10-Bit Addressing Mode

20.3.5.2 Master Mode

The eUSCI_B module is configured as an I²C master by selecting the I²C 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 UCBxI2COA0 register. Support for multiple slave addresses is explained in . When UCA10 = 0, 7-bit addressing is selected. When UCA10 = 1, 10-bit addressing is selected. The UCGCEN bit selects if the eUSCI_B module responds to a general call.

NOTE: Addresses and multi-master systems

In master mode with own-address detection enabled (UCOAEN = 1)—especially in multi-master systems—it is not allowed to specify the same address in the own address and slave address register (UCBxI2CSA = UCBxI2COAx). This would mean that the eUSCI_B addresses itself.

The user software must ensure that this situation does not occur. There is no hardware detection for this case, and the consequence is unpredictable behavior of the eUSCI_B.

20.3.5.2.1 I²C 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 eUSCI_B module waits until the bus is available, then generates the START condition, and transmits the slave address. The UCTXIFG0 bit is set when the START condition is generated and the first data to be transmitted can be written into UCBxTXBUF. **The UCTXSTT flag is cleared as soon as the complete address is sent.**

The data written into UCBxTXBUF is transmitted if arbitration is not lost during transmission of the slave address. UCTXIFG0 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:

- No automatic STOP is generated
- The UCTXSTP bit is not set
- The UCTXSTT bit is not set

Setting UCTXSTP generates a STOP condition after the next acknowledge from the slave. If UCTXSTP is set during the transmission of the slave address or while the eUSCI_B module waits for data to be written into UCBxTXBUF, a STOP condition is generated, even if no data was transmitted to the slave. **In this case, the UCSTPIFG is set.** When transmitting a single byte of data, the UCTXSTP bit must be set while the byte is being transmitted or any time after transmission begins, without writing new data into UCBxTXBUF. Otherwise, only the address is transmitted. When the data is transferred from the buffer to the shift register, UCTXIFG0 is set, indicating data transmission has begun, and the UCTXSTP bit may be set. When UCASTPx = 10 is set, the byte counter is used for STOP generation and the user does not need to set the UCTXSTP. **This is recommended when transmitting only one byte.**

Setting UCTXSTT generates 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 is discarded. If this data should be transmitted after a repeated START, it must be written into UCBxTXBUF again. Any set UCTXSTT or UCTXSTP is also discarded.

Figure 20-12 shows the I²C master transmitter operation.

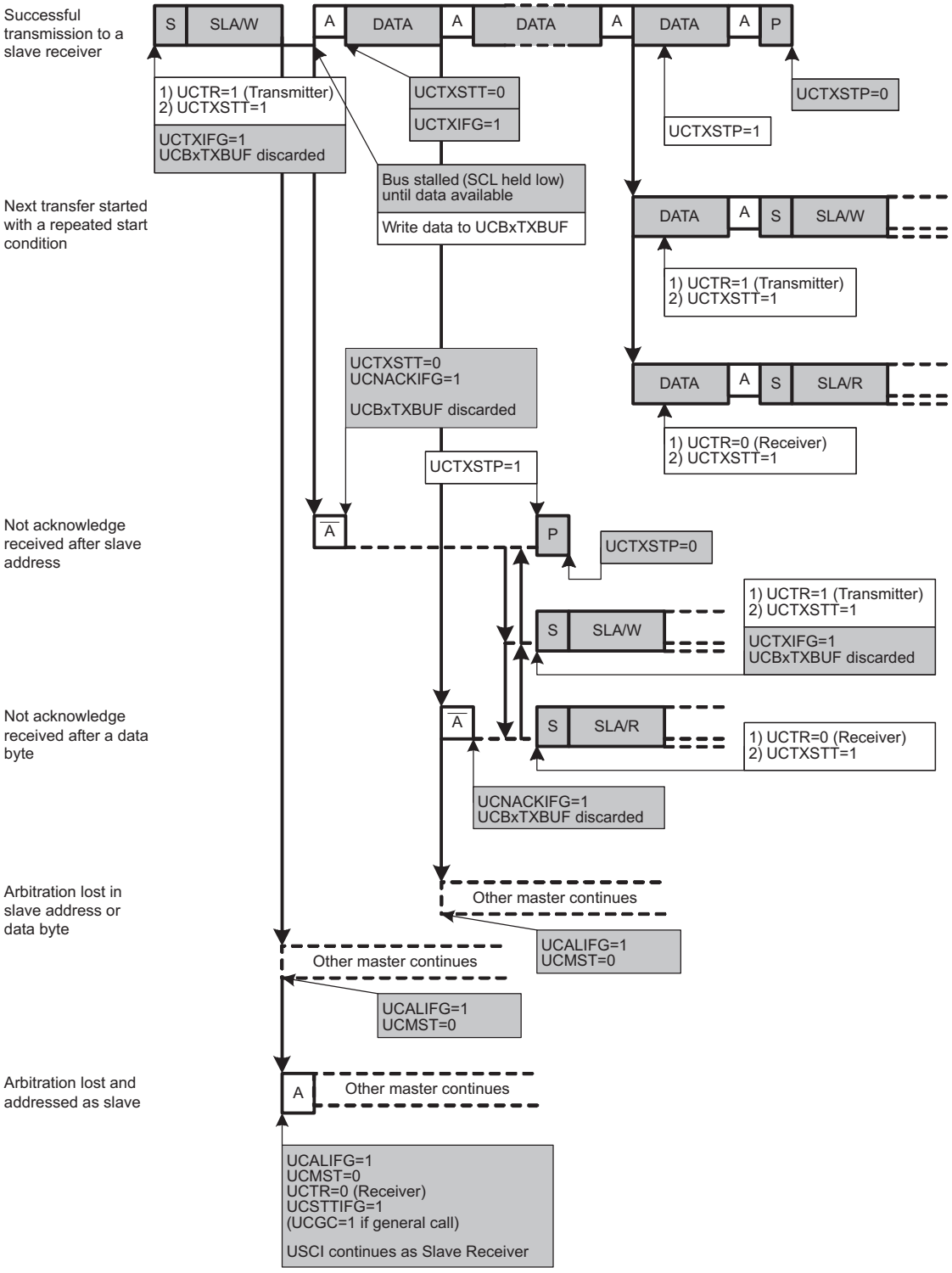


Figure 20-12. I²C Master Transmitter Mode

20.3.5.2.2 I²C 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 eUSCI_B module checks if the bus is available, generates the START condition, and transmits the slave address. The UCTXSTT flag is cleared as soon as the complete address is sent.

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, as long as:

- No automatic STOP is generated
- The UCTXSTP bit is not set
- The UCTXSTT bit is not set

If a STOP condition was generated by the eUSCI_B module, the UCSTPIFG is 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.

A STOP condition is either generated by the automatic STOP generation or by setting the UCTXSTP bit. The next byte received from the slave is followed by a NACK and a STOP condition. This NACK occurs immediately if the eUSCI_B module is currently waiting for UCBxRXBUF to be read.

If a RESTART is sent, UCTR may be set or cleared to configure transmitter or receiver, and a different slave address may be written into UCBxI2CSA if desired.

Figure 20-13 shows the I²C master receiver operation.

NOTE: Consecutive master transactions without repeated START

When performing multiple consecutive I²C 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 I²C transaction is initiated with setting UCTXSTT = 1. Otherwise, the current transaction might be affected.

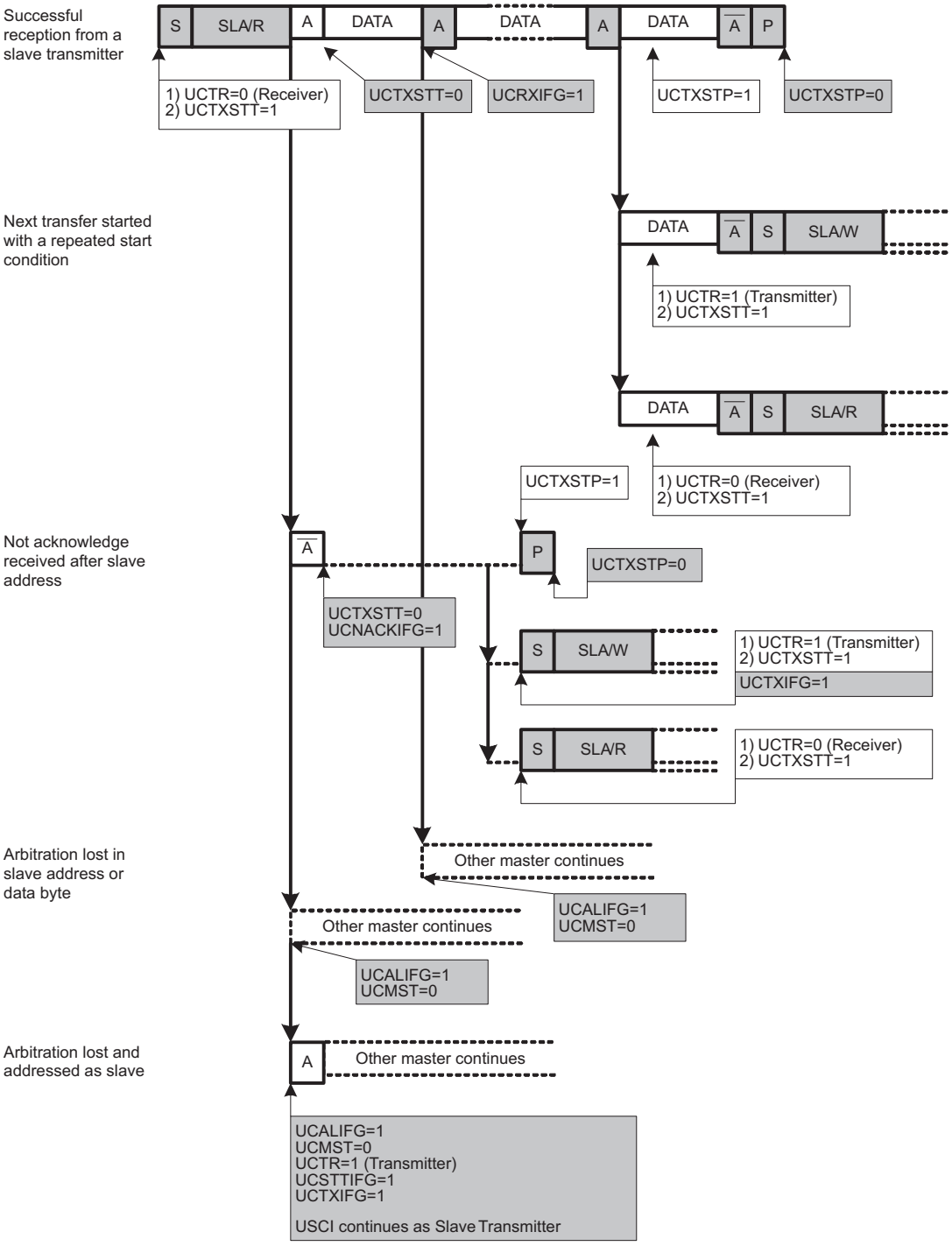


Figure 20-13. I²C Master Receiver Mode

20.3.5.2.3 I²C Master 10-Bit Addressing Mode

The 10-bit addressing mode is selected when UCSLA10 = 1 and is shown in Figure 20-14.

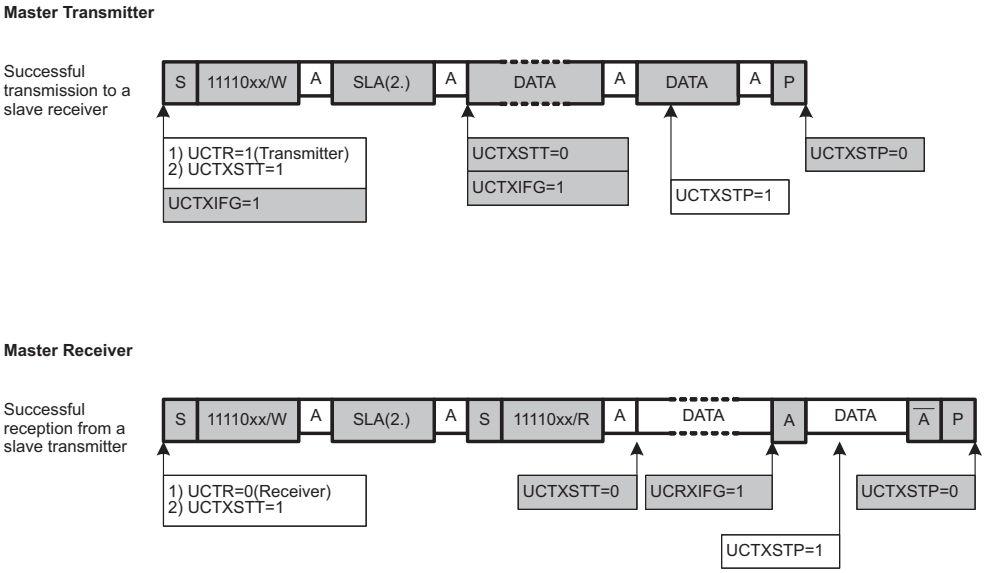


Figure 20-14. I²C Master 10-Bit Addressing Mode

20.3.5.3 Arbitration

If two or more master transmitters simultaneously start a transmission on the bus, an arbitration procedure is invoked. Figure 20-15 shows 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.

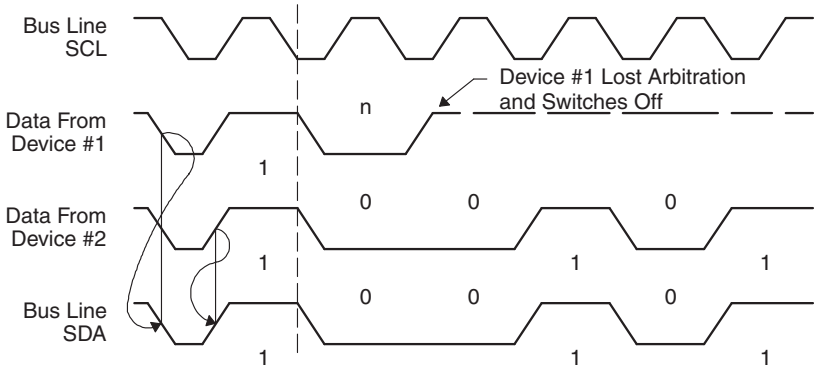


Figure 20-15. Arbitration Procedure Between Two Master Transmitters

There is an undefined condition if the arbitration procedure is still in progress when one master sends a repeated START or a STOP condition while the other master is still sending data. In other words, the following combinations result in an undefined condition:

- Master 1 sends a repeated START condition and master 2 sends a data bit.
- Master 1 sends a STOP condition and master 2 sends a data bit.
- Master 1 sends a repeated START condition and master 2 sends a STOP condition.

20.3.6 Glitch Filtering

According to the I²C standard, both the SDA and the SCL line need to be glitch filtered. The eUSCI_B module provides the UCGLITx bits to configure the length of this glitch filter:

Table 20-1. Glitch Filter Length Selection Bits

UCGLITx	Corresponding Glitch Filter Length on SDA and SCL	According to I²C Standard
00	Pulses of max 50-ns length are filtered	yes
01	Pulses of max 25-ns length are filtered.	no
10	Pulses of max 12.5-ns length are filtered.	no
11	Pulses of max 6.25-ns length are filtered.	no

20.3.7 I²C Clock Generation and Synchronization

The I²C clock SCL is provided by the master on the I²C bus. When the eUSCI_B is in master mode, BITCLK is provided by the eUSCI_B 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 eUSCI_B 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 eUSCI_B 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 20-16 shows the clock synchronization. This allows a slow slave to slow down a fast master.

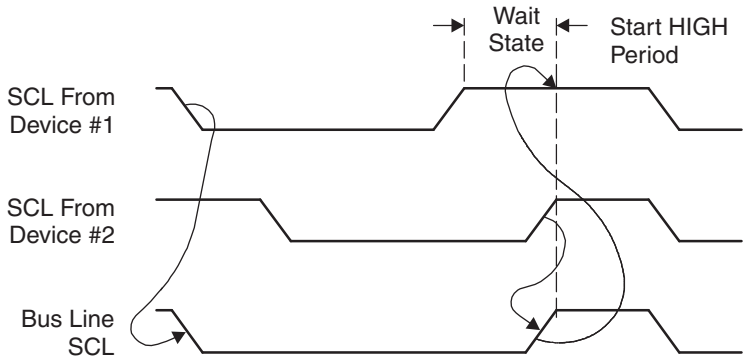


Figure 20-16. Synchronization of Two I²C Clock Generators During Arbitration

20.3.7.1 Clock Stretching

The eUSCI_B 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 eUSCI_B module already released SCL due to the following conditions:

- eUSCI_B is acting as master and a connected slave drives SCL low.

- eUSCI_B is acting as master and another master drives SCL low during arbitration.

The UCSCLLOW bit is also active if the eUSCI_B 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 be 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.

20.3.7.2 Avoiding Clock Stretching

Even though clock stretching is part of the I2C specification, there are applications in which clock stretching should be avoided.

The clock is stretched by the eUSCI_B under the following conditions:

- The internal shift register is expecting data, but the TXIFG is still pending
- The internal shift register is full, but the RXIFG is still pending
- The arbitration lost interrupt is pending
- UCSWACK is selected and UCBxI2COA0 did cause a match

To avoid clock stretching, all of these situations for clock stretch either need to be avoided or the corresponding interrupt flags need to be processed before the actual clock stretch can occur.

Using the DMA (on devices that contain a DMA) is the most secure way to avoid clock stretching. If no DMA is available, the software must ensure that the corresponding interrupts are serviced in time before the clock is stretched.

In slave transmitter mode, the TXIFG is set only after the reception of the direction bit; therefore, there is only a short amount of time for the software to write the TXBUF before a clock stretch occurs. This situation can be remedied by using the early Transmit Interrupt (see).

20.3.7.3 Clock Low Timeout

The UCCLTOIFG interrupt allows the software to react if the clock is low longer than a defined time. It is possible to detect the situation, when a clock is stretched by a master or slave for a too long time. The user can then, for example, reset the eUSCI_B module by using the UCSWRST bit.

The clock low timeout feature is enabled using the UCCLTO bits. It is possible to select one of three predefined times for the clock low timeout. If the clock has been low longer than the time defined with the UCCLTO bits and the eUSCI_B was actively receiving or transmitting, the UCCLTOIFG is set and an interrupt request is generated if UCCLTOIE and GIE are set as well. The UCCLTOIFG is set only once, even if the clock is stretched a multiple of the time defined in UCCLTO.

20.3.8 Byte Counter

The eUSCI_B module supports hardware counting of the bytes received or transmitted. The counter is automatically active and counts up for each byte seen on the bus in both master and slave mode.

The byte counter is incremented at the second bit position of each byte independently of the following ACK or NACK. A START or RESTART condition resets the counter value to zero. Address bytes do not increment the counter. The byte counter is also incremented at the second byte position, if an arbitration lost occurs during the first bit of data.

20.3.8.1 Byte Counter Interrupt

If UCASTPx = 01 or 10 the UCBCNTIFG is set when the byte counter threshold value UCBxTBCNT is reached in both master- and slave-mode. Writing zero to UCBxTBCNT does not generate an interrupt.

Because the UCBCNTIFG has a lower interrupt priority than the UCBTXIFG and UCBRXIFG, it is recommended to only use it for protocol control together with the DMA handling the received and transmitted bytes. Otherwise the application must have enough processor bandwidth to ensure that the UCBCNT interrupt routine is executed in time to generate for example a RESTART.

20.3.8.2 Automatic STOP Generation

When the eUSCI_B module is configured as a master, the byte counter can be used for automatic STOP generation by setting the UCASTPx = 10. Before starting the transmission using UCTXSTT, the byte counter threshold UCBxTBCNT must be set to the number of bytes that are to be transmitted or received. After the number of bytes that are configured in UCBxTBCNT have been transmitted, the eUSCI_B automatically generates a STOP condition.

UCBxTBCNT cannot be used if the user wants to transmit the slave address only without any data. In this case, it is recommended to set UCTXSTT and UCTXSTP at the same time.

20.3.9 Multiple Slave Addresses

The eUSCI_B module supports two different ways of implementing multiple slave addresses at the same time:

- Hardware support for up to 4 different slave addresses, each with its own interrupt flag and DMA trigger
- Software support for up to 2^{10} different slave addresses all sharing one interrupt

20.3.9.1 Multiple Slave Address Registers

The registers UCBxI2COA0, UCBxI2COA1, UCBxI2COA2, and UCBxI2COA3 contain four slave addresses. Up to four address registers are compared against a received 7- or 10-bit address. Each slave address must be activated by setting the UCAOEN bit in the corresponding UCBxI2COAx register. Register UCBxI2COA3 has the highest priority if the address received on the bus matches more than one of the slave address registers. The priority decreases with the index number of the address register, so that UCBxI2COA0 in combination with the address mask has the lowest priority.

When one of the slave registers matches the 7- or 10-bit address seen on the bus, the address is acknowledged. In the following the corresponding receive- or transmit-interrupt flag (UCTXIFGx or UCRXIFGx) to the received address is updated. The state change interrupt flags are independent of the address comparison result. They are updated according to the bus condition.

20.3.9.2 Address Mask Register

The Address Mask Register can be used when the eUSCI_B is configured in slave or in multiple-master mode. To activate this feature, at least one bit of the address mask in register UCBxADDMASK must be cleared.

If the received address matches the own address in UCBxI2COA0 on all bit positions not masked by UCBxADDMASK the eUSCI_B considers the seen address as its own address and sends an acknowledge. The user has the choice to either automatically acknowledge the address seen on the bus or to evaluate this address and send the acknowledge in software using UCTXACK. The selection between these options is done using the UCSWACK bit. If the software is used for generation of the ACK of the slave address, it is recommended to use the UCSTTIFG. The received address can be found in the UCBxADDRX register.

A slave address seen on the bus is automatically acknowledged by the eUSCI_B module, if it matches any of the slave addresses defined in UCBxI2COA1 to UCBxI2COA3.

NOTE: UCSWACK and slave-transmitter

If the user selects manual acknowledge of slave addresses, the TXIFG is set if the slave is addressed as a transmitter. If the user decides not to acknowledge the address, the TXIFG0 also must be reset.

20.3.10 Using the eUSCI_B Module in I²C Mode With Low-Power Modes

The eUSCI_B module provides automatic clock activation for use with low-power modes. When the eUSCI_B clock source is inactive because the device is in a low-power mode, the eUSCI_B module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the eUSCI_B module returns to its idle condition. After the eUSCI_B module returns to the idle condition, control of the clock source reverts to the settings of its control bits.

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

20.3.11 eUSCI_B Interrupts in I²C Mode

The eUSCI_B has only one interrupt vector that is shared for transmission, reception, and the state change.

Each interrupt flag has its own interrupt enable bit. When an interrupt is enabled and the GIE bit is set, the interrupt flag generates an interrupt request. DMA transfers are controlled by the UCTXIFGx and UCRXIFGx flags on devices with a DMA controller. It is possible to react on each slave address with an individual DMA channel.

All interrupt flags are not cleared automatically, but they need to be cleared together by user interactions (for example, reading the UCRXBUF clears UCRXIFGx). If the user wants to use an interrupt flag he needs to ensure that the flag has the correct state before the corresponding interrupt is enabled.

20.3.11.1 I²C Transmit Interrupt Operation

The UCTXIFG0 interrupt flag is set whenever the transmitter is able to accept a new byte. When operating as a slave with multiple slave addresses, the UCTXIFGx flags are set corresponding to which address was received before. If, for example, the slave address specified in register UCBxI2COA3 did match the address seen on the bus, the UCTXIFG3 indicates that the UCBxTXBUF is ready to accept a new byte.

When operating in master mode with automatic STOP generation (UCASTPx = 10), the UCTXIFG0 is set as many times as defined in UCBxTBCNT.

An interrupt request is generated if UCTXIEx and GIE are also set. UCTXIFGx is automatically reset if a write to UCBxTXBUF occurs or if the UCALIFG is cleared. UCTXIFGx is set when:

- Master mode: UCTXSTT was set by the user
- Slave mode: own address was received(UCETXINT = 0) or START was received (UCETXINT = 1)

UCTXIEx is reset after a PUC or when UCSWRST = 1.

20.3.11.2 Early I²C Transmit Interrupt

Setting the UCETXINT causes UCTXIFG0 to be sent out automatically when a START condition is sent and the eUSCI_B is configured as slave. In this case, it is not allowed to enable the other slave addresses UCBxI2COA1-UCBxI2COA3. This allows the software more time to handle the UCTXIFG0 compared to the normal situation, when UCTXIFG0 is sent out after the slave address match was detected. Situations where the UCTXIFG0 was set and afterward no slave address match occurred need to be handled in software. The use of the byte counter is recommended to handle this.

20.3.11.3 I²C Receive Interrupt Operation

The UCRXIFG0 interrupt flag is set when a character is received and loaded into UCBxRXBUF. When operating as a slave with multiple slave addresses, the UCRXIFGx flag is set corresponding to which address was received before.

An interrupt request is generated if UCRXIEx and GIE are also set. UCRXIFGx and UCRXIEx are reset after a PUC signal or when UCSWRST = 1. UCRXIFGx is automatically reset when UCxRXBUF is read.

20.3.11.4 I²C State Change Interrupt Operation

Table 20-2 describes the I²C state change interrupt flags.

Table 20-2. I²C State Change Interrupt Flags

Interrupt Flag	Interrupt Condition
UCALIFG	Arbitration-lost. Arbitration can be lost when two or more transmitters start a transmission simultaneously, or when the eUSCI_B operates as master but is addressed as a slave by another master in the system. The UCALIFG flag is set when arbitration is lost. When UCALIFG is set, the UCMST bit is cleared and the I²C controller becomes a slave.
UCNACKIFG	Not-acknowledge interrupt. This flag is set when an acknowledge is expected but is not received. UCNACKIFG is used in master mode only.
UCCLTOIFG	Clock low timeout. This interrupt flag is set, if the clock is held low longer than defined by the UCCLTO bits.
UCBIT9IFG	This interrupt flag is generated each time the eUSCI_B is transferring 9th clock cycle of a byte of data. This gives the user the possibility to follow the I²C communication in software if wanted. The UCBIT9IFG is not set for address information.
UCBCNTIFG	Byte counter interrupt. This flag is set when the byte counter value reaches the value defined in UCBxTBCNT and UCASTPx = 01 or 10. This bit allows to organize following communications, especially if a RESTART will be issued.
UCSTTIFG	START condition detected interrupt. This flag is set when the I²C module detects a START condition together with its own address(1). UCSTTIFG is used in slave mode only.
UCSTPIFG	STOP condition detected interrupt. This flag is set when the I²C module detects a STOP condition on the bus. UCSTPIFG is used in slave and master mode.

(1) The address evaluation includes the address mask register if it is used.

20.3.11.5 UCBxIV, Interrupt Vector Generator

The eUSCI_B 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 PC to automatically enter the appropriate software routine. Disabled interrupts do not affect the UCBxIV value.

Read access 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.

Write access of the ADC10IV register clears all pending Interrupt conditions and flags.

Example 20-3 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 eUSCI0_B.

Example 20-3. UCBxIV Software Example

```
#pragma vector = USCI_B0_VECTOR __interrupt void USCI_B0_ISR(void) {
    switch(__even_in_range(UCB0IV,18))    {
        case 0x00:     // Vector 0: No interrupts break;
        case 0x02: ... // Vector 2: ALIFG break;
        case 0x04: ... // Vector 4: NACKIFG break;
        case 0x06: ... // Vector 6: STTIFG break;
        case 0x08: ... // Vector 8: STPIFG break;
        case 0x0a: ... // Vector 10: RXIFG3 break;
        case 0x0c: ... // Vector 14: TXIFG3 break;
        case 0x0e: ... // Vector 16: RXIFG2 break;
        case 0x10: ... // Vector 18: TXIFG2 break;
        case 0x12: ... // Vector 20: RXIFG1 break;
        case 0x14: ... // Vector 22: TXIFG1 break;
        case 0x16: ... // Vector 24: RXIFG0 break;
        case 0x18: ... // Vector 26: TXIFG0 break;
        case 0x1a: ... // Vector 28: BCNTIFG break;
        case 0x1c: ... // Vector 30: clock low timeout break;
        case 0x1e: ... // Vector 32: 9th bit break;
        default: break;
    }
}
```

20.4 eUSCI_B Registers – I²C Mode

The eUSCI_B registers applicable in I²C mode are listed in Table 20-3. The base address can be found in the device-specific data sheet. The address offsets are listed in Table 20-3.

Table 20-3. eUSCIx_B Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
eUSCI_Bx Control Word 0	UCBxCTLW0	Read/write	Word	00h	01C1h
eUSCI_Bx Control 1	UCBxCTL1	Read/write	Byte	00h	C1h
eUSCI_Bx Control 0	UCBxCTL0	Read/write	Byte	01h	01h
eUSCI_Bx Control Word 1	UCBxCTLW1	Read/write	Word	02h	0000h
eUSCI_Bx Bit Rate Control Word	UCBxBRW	Read/write	Word	06h	0000h
eUSCI_Bx Bit Rate Control 0	UCBxBR0	Read/write	Byte	06h	00h
eUSCI_Bx Bit Rate Control 1	UCBxBR1	Read/write	Byte	07h	00h
eUSCI_Bx Status Word	UCBxSTATW	Read	Word	08h	0000h
eUSCI_Bx Status	UCBxSTAT	Read	Byte	08h	00h
eUSCI_Bx Byte Counter Register	UCBxBCNT	Read	Byte	09h	00h
eUSCI_Bx Byte Counter Threshold Register	UCBxTBCNT	Read/Write	Word	0Ah	00h
eUSCI_Bx Receive Buffer	UCBxRXBUF	Read/write	Word	0Ch	00h
eUSCI_Bx Transmit Buffer	UCBxTXBUF	Read/write	Word	0Eh	00h
eUSCI_Bx I²C Own Address 0	UCBxI2COA0	Read/write	Word	14h	0000h
eUSCI_Bx I²C Own Address 1	UCBxI2COA1	Read/write	Word	16h	0000h
eUSCI_Bx I²C Own Address 2	UCBxI2COA2	Read/write	Word	18h	0000h
eUSCI_Bx I²C Own Address 3	UCBxI2COA3	Read/write	Word	1Ah	0000h
eUSCI_Bx Received Address Register	UCBxADDRX	Read	Word	1Ch	
eUSCI_Bx Address Mask Register	UCBxADDMASK	Read/write	Word	1Eh	03FFh
eUSCI_Bx I²C Slave Address	UCBxI2CSA	Read/write	Word	20h	0000h
eUSCI_Bx Interrupt Enable	UCBxIE	Read/write	Word	2Ah	0000h
eUSCI_Bx Interrupt Flag	UCBxIFG	Read/write	Word	2Ch	2A02h
eUSCI_Bx Interrupt Vector	UCBxIV	Read	Word	2Eh	0000h

eUSCI_Bx Control Word Register 0 (UCBxCTLW0)

15	14	13	12	11	10	9	8
UCA10	**UCSLA10**	**UCMM**	**Reserved**	**UCMST**	**UCMODEx = 11**		**Reserved**
rw-0	rw-0	rw-0	r0	rw-0	rw-0	rw-0	r1
7	6	5	4	3	2	1	0
UCSSELx		**UCTXACK**	**UCTR**	**UCTXNACK**	**UCTXSTP**	**UCTXSTT**	**UCSWRST**
rw-1	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

Modify only when eUSCI_B is in reset state (UCSWRST = 1)

UCA10 Bit 15 Own addressing mode select
- 0 Own address is a 7-bit address.
- 1 Own address is a 10-bit address.

UCSLA10 Bit 14 Slave addressing mode select
- 0 Address slave with 7-bit address
- 1 Address slave with 10-bit address

UCMM Bit 13 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

Reserved Bit 12 Reserved

UCMST Bit 11 Master mode select. When a master loses 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 10-9 eUSCI_B 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 I²C mode

Reserved Bit 8 Reserved

UCSSELx Bits 7-6 eUSCI_B clock source select. These bits select the BRCLK source clock. These bits are ignored in slave mode.
- 00 UCLKI
- 01 ACLK
- 10 SMCLK
- 11 SMCLK

UCTXACK Bit 5 Transmit ACK condition in slave mode with enabled address mask register. After the UCSTTIFG has been set, the user needs to set or reset he UCTXACK flag to continue with the I2C protocol. The clock is stretched until the UCBxCTL1 register has been written. This bit is cleared automatically after the ACK has been send.
- 0 Do not acknowledge the slave address
- 1 Acknowledge the slave address

UCTR Bit 4 Transmitter/receiver
- 0 Receiver
- 1 Transmitter

UCTXNACK Bit 3 Transmit a NACK. UCTXNACK is automatically cleared after a NACK is transmitted. Only for slave receiver mode.
- 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. This bit is a don't care, if automatic UCASTPx is different from 01 or 10.
- 0 No STOP generated
- 1 Generate STOP

(continued)

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. eUSCI_B released for operation. 1 Enabled. eUSCI_B logic held in reset state.

eUSCI_Bx Control Word Register (UCBxCTLW1)

15	14	13	12	11	10	9	8
Reserved							**UCETXINT**
r0	r0	r0	r0	r0	r0	r0	rw-0
7	6	5	4	3	2	1	0
UCCLTO		**UCSTPNACK**	**UCSWACK**	**UCASTPx**		**UCGLITx**	
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Reserved	Bits 15-0	Reserved
UCETXINT	Bit 9	Early UCTXIFG0. Only in slave mode. When this bit is set the slave addresses defined in UCxI2COA1-UCxI2COA3 need to be disabled. 0 UCTXIFGx is set after an address match with UCxI2COAx and the direction bit indicating slave transmit 1 UCTXIFG0 is set for each START condition.
UCCLTO	Bits 7-6	Clock low timeout select. 00 disable Clock low timeout counter 01 135 000 MODCLK cycles (≠28 ms) 10 150 000 MODCLK cycles (≠31 ms) 11 165 000 MODCLK cycles (≠34 ms)
UCSTPNACK	Bit 5	The UCSTPNACK bit allows to make the eUSCI_B master acknowledge the last byte in master receiver mode as well. This is not conform to the I²C specification and should only be used for slaves, which automatically release the SDA after a fixed packet length. 0 Send a non-acknowledge before the STOP condition as a master receiver (conform to I²C standard) 1 All bytes are acknowledged by the eUSCI_B when configured as master receiver.
UCSWACK	Bit 4	Using this bit it is possible to select, whether the eUSCI_B module triggers the sending of the ACK of the address or if it is controlled by software. 0 The address acknowledge of the slave is controlled by the eUSCI_B module 1 The user needs to trigger the sending of the address ACK by issuing UCTXACK.
UCASTPx	Bit 3-2	Automatic STOP condition generation. In slave mode only UCBCNTIFG is available. 00 No automatic STOP generation. The STOP condition is generated after the user sets the UCTXSTP bit. The value in UCBxTBCNT is a don't care. 01 UCBCNTIFG is set with the byte counter reaches the threshold defined in UCBxTBCNT. 10 A STOP condition is generated automatically after the byte counter value reached UCBxTBCNT. UCBCNTIFG is set with the byte counter reaching the threshold. 11 Reserved
UCGLITx	Bits 1-0	Deglitch time 00 50 ns 01 25 ns 10 12.5 ns 11 6.25 ns

eUSCI_Bx Baud Rate Control Register 1 (UCBxBR1)

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

UCBRx	Bits 15-0	Bit clock prescaler.

eUSCI_Bx Status Word Register (UCBxSTATW)

15	14	13	12	11	10	9	8
UCBCNTx							
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	UCSCLLOW	UCGC	UCBBUSY	Reserved			
r0	r-0	r-0	r-0	r-0	r0	r0	r0

UCBCNTx	Bits 15-8	Hardware byte counter value. Reading this register returns the number of bytes received or transmitted on the I²C-Bus since the last START or RESTART. There is no synchronization of this register done. When reading UCBxBCNT during the first bit position, a faulty readback can occur.
Reserved	Bit 7	Reserved
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
Reserved	Bits 3-0	Reserved

eUSCI_Bx Byte Counter Threshold Register (UCBxTBCNT)

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

Reserved	Bits 15-8	Reserved
UCTBCNTx	Bits 7-0	The byte counter threshold value is used to set the number of I²C data bytes after which the automatic STOP or the UCSTPIFG should occur. This value is evaluated only if UCASTPx is different from 00.

eUSCI_Bx Receive Buffer Register (UCBxRXBUF)

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

Reserved	Bits 15-8	Reserved
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 the UCRXIFGx flags.

eUSCI_Bx Transmit Buffer Register (UCBxTXBUF)

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
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 the UCTXIFGx flags.

eUSCI_Bx I²C Own Address Registers (UCBxI2COAx)

15	14	13	12	11	10	9	8
UCGCEN	**0**	**0**	**0**	**0**	**UCOAEN**	**I2COAx**	
rw-0	r0	r0	r0	r0	rw-0	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. This bit is only available in UCBxI2COA0. 0 Do not respond to a general call 1 Respond to a general call
UCOAEN	Bit 10	Own Address enable register. With this register it can be selected if the I²C slave-address related to this register UCBxI2COAx is evaluated or not. 0 The slave address defined in I2COAx is disabled 1 The slave address defined in I2COAx is enabled
I2COAx	Bits 9-0	I²C own address. The I2COAx bits contain the local address of the eUSCIx_B I²C 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.

eUSCI_Bx I²C Received Address Register (UCBxADDRX)

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

ADDRXx	Bits 9-0	Received Address Register. This register contains the last received slave address on the bus. Using this register and the address mask register it is possible to react on more than one slave address using one eUSCI_B module.

eUSCI_Bx I²C Address Mask Register (UCBxADDMASK)

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

ADDMASKx	Bits 9-0	Address Mask Register. By clearing the corresponding bit of the own address, this bit is a don't care when comparing the address on the bus to the own address. Using this method, it is possible to react on more than one slave address. When all bits of ADDMASKx are set, the address mask feature is deactivated.

eUSCI_Bx I²C Slave Address Register (UCBxI2CSA)

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	I²C slave address. The I2CSAx bits contain the slave address of the external device to be addressed by the eUSCIx_B module. It is only used in master mode. The address is right justified. In 7-bit slave addressing mode, bit 6 is the MSB and bits 9-7 are ignored. In 10-bit slave addressing mode, bit 9 is the MSB.

eUSCI_Bx I²C Interrupt Enable Register (UCBxIE)

15	14	13	12	11	10	9	8
Reserved	**UCBIT9IE**	**UCTXIE3**	**UCRXIE3**	**UCTXIE2**	**UCRXIE2**	**UCTXIE1**	**UCRXIE1**
r0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
UCCLTOIE	**UCBCNTIE**	**UCNACKIE**	**UCALIE**	**UCSTPIE**	**UCSTTIE**	**UCTXIE0**	**UCRXIE0**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Reserved	Bits 15-9	Reserved
UCBIT9IE	Bit 14	Bit position 9 interrupt enable 0 Interrupt disabled 1 Interrupt enabled
UCTXIE3	Bit 13	Transmit Interrupt enable 3 0 Interrupt disabled 1 Interrupt enabled
UCTXIE3	Bit 12	Receive Interrupt enable 3 0 Interrupt disabled 1 Interrupt enabled
UCTXIE2	Bit 11	Transmit Interrupt enable 2 0 Interrupt disabled 1 Interrupt enabled
UCRXIE2	Bit 10	Receive Interrupt enable 2 0 Interrupt disabled 1 Interrupt enabled
UCTXIE1	Bit 9	Transmit Interrupt enable 1 0 Interrupt disabled 1 Interrupt enabled
UCRXIE1	Bit 8	Receive Interrupt enable 1 0 Interrupt disabled 1 Interrupt enabled
UCCLTOIE	Bit 7	Clock low timeout interrupt enable. 0 Interrupt disabled 1 Interrupt enabled
UCBCNTIE	Bit 6	Byte counter interrupt enable. 0 Interrupt disabled 1 Interrupt enabled
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
UCTXIE0	Bit 1	Transmit interrupt enable 0 0 Interrupt disabled 1 Interrupt enabled
UCRXIE0	Bit 0	Receive interrupt enable 0 0 Interrupt disabled 1 Interrupt enabled

eUSCI_Bx I²C Interrupt Flag Register (UCBxIFG)

15	14	13	12	11	10	9	8
Reserved	**UCBIT9IFG**	**UCTXIFG3**	**UCRXIFG3**	**UCTXIFG2**	**UCRXIFG2**	**UCTXIFG1**	**UCRXIFG1**
r0	rw-0	rw-1	rw-0	rw-1	rw-0	rw-1	rw-0
7	6	5	4	3	2	1	0
UCCLTOIFG	**UCBCNTIFG**	**UCNACKIFG**	**UCALIFG**	**UCSTPIFG**	**UCSTTIFG**	**UCTXIFG0**	**UCRXIFG0**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1	rw-0

Reserved Bits 15-9 Reserved

UCBIT9IFG Bit 14 Bit position 9 interrupt flag

0 No interrupt pending

1 Interrupt pending

UCTXIFG3 Bit 13 eUSCI_B transmit interrupt flag 3. UCTXIFG3 is set when UCBxTXBUF is empty in slave mode, if the slave address defined in UCBxI2COA3 was on the bus in the same frame.

0 No interrupt pending

1 Interrupt pending

UCRXIFG3 Bit 12 Receive interrupt flag 2. UCRXIFG2 is set when UCBxRXBUF has received a complete byte in slave mode and if the slave address defined in UCBxI2COA2 was on the bus in the same frame.

0 No interrupt pending

1 Interrupt pending

UCTXIFG2 Bit 11 eUSCI_B transmit interrupt flag 2. UCTXIFG2 is set when UCBxTXBUF is empty in slave mode, if the slave address defined in UCBxI2COA2 was on the bus in the same frame.

0 No interrupt pending

1 Interrupt pending

UCRXIFG2 Bit 10 Receive interrupt flag 2. UCRXIFG2 is set when UCBxRXBUF has received a complete byte in slave mode and if the slave address defined in UCBxI2COA2 was on the bus in the same frame.

0 No interrupt pending

1 Interrupt pending

UCTXIFG1 Bit 9 eUSCI_B transmit interrupt flag 1. UCTXIFG1 is set when UCBxTXBUF is empty in slave mode, if the slave address defined in UCBxI2COA1 was on the bus in the same frame.

0 No interrupt pending

1 Interrupt pending

UCRXIFG1 Bit 8 Receive interrupt flag 1. UCRXIFG1 is set when UCBxRXBUF has received a complete byte in slave mode and if the slave address defined in UCBxI2COA1 was on the bus in the same frame.

0 No interrupt pending

1 Interrupt pending

UCCLTOIFG Bit 7 Clock low timeout interrupt flag

0 No interrupt pending

1 Interrupt pending

UCBCNTIFG Bit 6 Byte counter interrupt flag. When using this interrupt the user needs to ensure enough processing bandwidth(see).

0 No interrupt pending

1 Interrupt pending

UCNACKIFG Bit 5 Not-acknowledge received interrupt flag. This flag only is updated when operating in master mode.

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

0 No interrupt pending

1 Interrupt pending

UCSTTIFG Bit 2 START condition interrupt flag

0 No interrupt pending

1 Interrupt pending

(continued)

UCTXIFG0	Bit 1	eUSCI_B transmit interrupt flag 0. UCTXIFG0 is set when UCBxTXBUF is empty in master mode or in slave mode, if the slave address defined in UCBxI2COA0 was on the bus in the same frame. 0 No interrupt pending 1 Interrupt pending
UCRXIFG0	Bit 0	eUSCI_B receive interrupt flag 0. UCRXIFG0 is set when UCBxRXBUF has received a complete character in master mode or in slave mode, if the slave address defined in UCBxI2COA0 was on the bus in the same frame. 0 No interrupt pending 1 Interrupt pending

eUSCI_Bx Interrupt Vector Register (UCBxIV)

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**	**UCIVx**				**0**
r0	r0	r0	r0	r-0	r-0	r-0	r0

UCIVx Bits 15-0 eUSCI_B interrupt vector value. It generates an value that can be used as address offset for fast interrupt service routine handling. Writing to this register clears all pending interrupt flags.

UCBxIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
000h	No interrupt pending	–	
002h	Arbitration lost	UCALIFG	Highest
004h	Not acknowledgment	UCNACKIFG	
006h	Start condition received	UCSTTIFG	
008h	Stop condition received	UCSTPIFG	
00Ah	Slave 3 Data received	UCRXIFG3	
00Ch	Slave 3 Transmit buffer empty	UCTXIFG3	
00Eh	Slave 2 Data received	UCRXIFG2	
010h	Slave 2 Transmit buffer empty	UCTXIFG2	
012h	Slave 1 Data received	UCRXIFG1	
014h	Slave 1 Transmit buffer empty	UCTXIFG1	
016h	Data received	UCRXIFG0	
018h	Transmit buffer empty	UCTXIFG0	
01Ah	Byte counter zero	UCBCNTIFG	
01Ch	Clock low timeout	UCCLTOIFG	
01Eh	9th bit position	UCBIT9IFG	Lowest

Embedded Emulation Module (EEM)

This chapter describes the embedded emulation module (EEM) that is implemented in all flash devices.

Topic **Page**

21.1 Embedded Emulation Module (EEM) Introduction

Every MSP430 flash-based microcontroller implements an 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 21.3, the EEM Configurations section, and the device-specific 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 21-1 shows a simplified block diagram of the largest currently-available EEM implementation.

For more details on how the features of the EEM can be used together with the IAR Embedded Workbench ™ debugger or with Code Composer Essentials (CCE), see the application report *Advanced Debugging Using the Enhanced Emulation Module* (SLAA393) at www.msp430.com. Most other debuggers supporting the MSP430 devices have the same or a similar feature set. For details, see the user's guide of the applicable debugger.

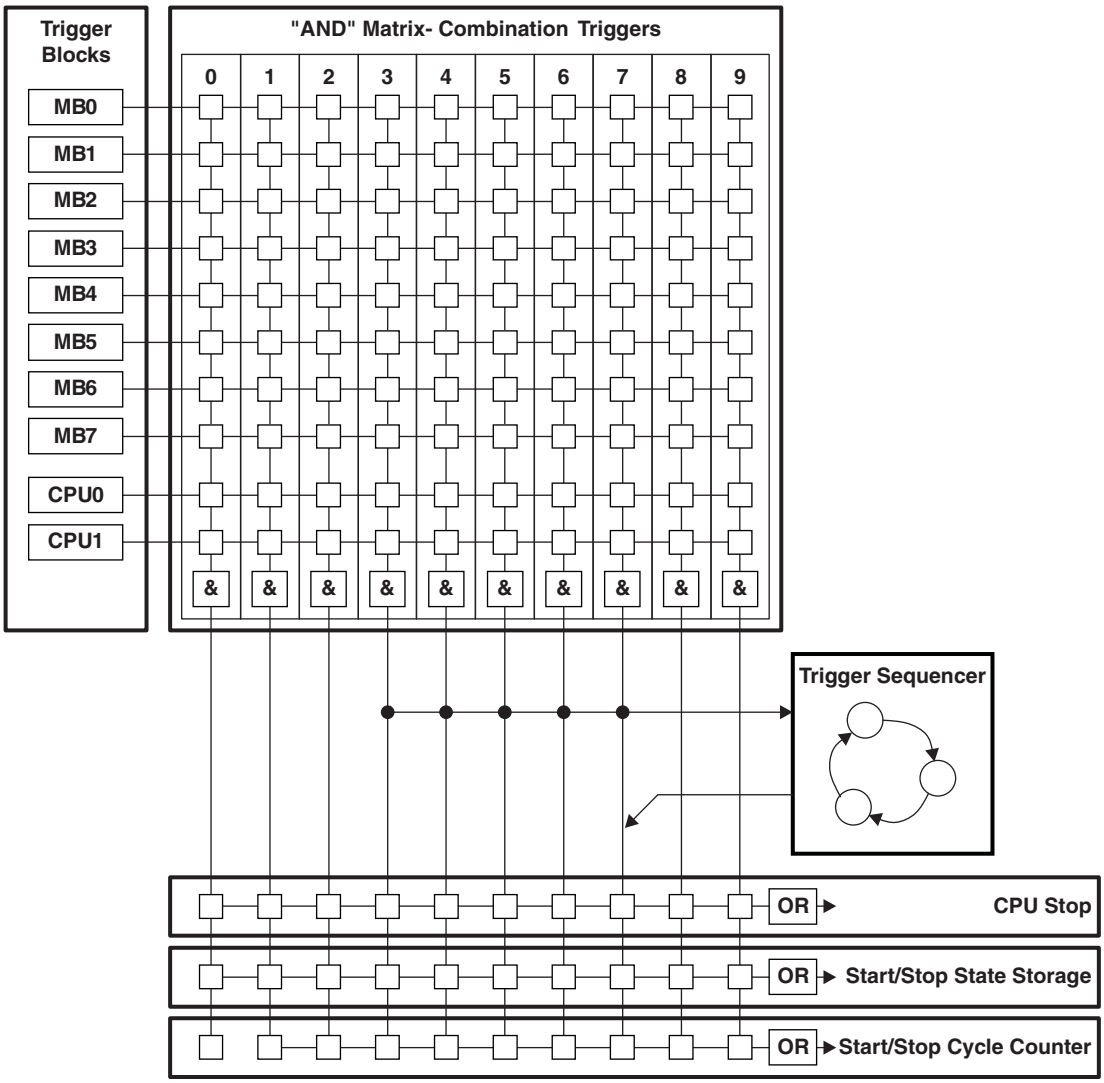


Figure 21-1. Large Implementation of EEM

21.2 EEM Building Blocks

21.2.1 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 cause various reactions other than 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.

21.2.2 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.

21.2.3 State Storage (Internal Trace Buffer)

The state storage function uses a built-in buffer to store MAB, MDB, and CPU control signal information (that is, 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.

21.2.4 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.

21.2.5 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 (for example, 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.

21.3 EEM Configurations

Table 21-1 gives an overview of the EEM configurations. The implemented configuration is device dependent, and details can be found in the device-specific data sheet and these documents:

Advanced Debugging Using the Enhanced Emulation Module (EEM) With CCS Version 4 (SLAA393)

IAR Embedded Workbench Version 3+ for MSP430 User's Guide (SLAU138)

Code Composer Studio v4.2 for MSP430 User's Guide (SLAU157).

Table 21-1. 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 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**	
Audio	www.ti.com/audio	Communications and Telecom	www.ti.com/communications
Amplifiers	amplifier.ti.com	Computers and Peripherals	www.ti.com/computers
Data Converters	dataconverter.ti.com	Consumer Electronics	www.ti.com/consumer-apps
DLP® Products	www.dlp.com	Energy and Lighting	www.ti.com/energy
DSP	dsp.ti.com	Industrial	www.ti.com/industrial
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Security	www.ti.com/security
Logic	logic.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Power Mgmt	power.ti.com	Transportation and Automotive	www.ti.com/automotive
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com	Wireless	www.ti.com/wireless-apps
RF/IF and ZigBee® Solutions	www.ti.com/lprf		
	TI E2E Community Home Page		e2e.ti.com

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