MSP430x09x Family

User's Guide

Literature Number: SLAU321
September 2010

Contents

List of Figures

List of Tables

Read This First

About This Manual

This manual describes the modules and peripherals of the 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, Compact System Control Module (CSYS)

The Compact System Control Module (CSYS) is integrated into various devices with different feature sets. It provides start-up functionality and interrupt support functions.

The following list shows the basic feature set of CSYS.

- Power on reset (BOR/POR) handling
- Power up clear (PUC) handling
- NMI (SNMI/UNMI) event source selection and management
- Address decoding
- Providing an user data exchange mechanism via the JTAG Mailbox (JMB)
- Configuration management (device descriptors)
- Providing interrupt vector generators for resets and NMIs

Topic **Page**

1.1 Compact System Control Module Introduction

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

1.2 Principle of Operation

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

1.2.1 Device Descriptor Table

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

1.2.2 Start-Up Code (SUC)

The start-up code is always executed after a BOR. The SUC provides basic routines for testing ROM mask and other test related functions. SUC gives control immediately to the user code on normal startups (no test functions invoked). All POR and PUC requests fire an security BOR during SUC execution. This insures completion of SUC before entering user code.

1.2.3 Boot Loader Code

The MSP430 boot loader is software that is executed after startup on certain devices (e.g. MSP430L092). This boot loader enables the user to store its code to external memory accessible via I2C or SPI. The boot loader scans the external memory devices for a valid code signature an then loads the associated code into the internal RAM memory and invokes it. A set functions supporting this was introduced to support code development and testing during the prototyping phase of the final product.

1.2.4 JTAG Mailbox (JMB) System

The CSYS module provides the capability to exchange user data via the regular JTAG test/debug interface. The idea behind the JTAG mailbox system is to have a direct interface to the CPU during debugging, programming and test that is identical for all MSP430 devices of this family and uses only few or no user application resources. The JTAG interface was chosen because it is available on all MSP430 devices and is a dedicated resource for debugging, programming, and test.

Applications of the JTAG mailbox system are:

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

1.3 Memory Map – Uses and Abilities

The memory map shown in Table 1-1 represents the MSP430x09x device. Though the address ranges differs from device to device, overall behavior remains the same.

Table 1-1. Memory Map

Address		Name/Purpose	Properties				
			Mirrored Memory location optional	Generates PUC on fetch access	Protectable against write access	Protectable against read access	May generate NMI on read/write/fetch
00000h - 00FFFFh		Peripherals (with gaps)					
	00000h - 000FFh	Reserved for system -extension					
	00100h - 00FEFh	Peripherals		X			
	00FF0h - 00FF3h	Descriptor type		X			
	00FF4h - 00FF7h	Start address of descriptor structure		X			
01C00h - 023FFh		RAM 2k					
	01C00 - 01C7F	Calibration RAM (lockable)			X		
	01C80 - 0237F	Application Memory (lockable)			X		
	02380 - 023FF	Appl. Memory (non lockable)					
	02380 - 023FF	Alternate Interrupt Vectors					
02400h - 0F7FFh		Vacant Memory					X
0F800h - 0FFFFh		Program Memory					
	0F800h - 0F87Fh	Start-Up Code (SUC) memory mirror	X				
	0F880h - 0FF7Fh	Application Program					
	0FF80h - 0FFFFh	Interrupt Vectors					
10000h - FFFFFh		Vacant Memory					X

1.3.1 *Vacant Memory Space*

Accesses to vacant memory space generates an NMI interrupt. Reads from vacant memory results in the value 3FFFh. In the case of a fetch, this is taken as JMP $. Fetch accesses from vacant peripheral space result in a PUC.

1.3.2 *Start-Up Code (SUC)*

After a BOR, the memory location 0F800h is the reset vector to start the start-up code. The start-up code evaluates if test routines are to be invoked an executes them before the control is handed over to the application code. The SUC also checks for a password to allow JTAG access of the MSP430x09x application code for debug purposes.

1.3.3 SYS Interrupt Vector Generators

The CSYS module collects all user NMI (UNMI) sources, system NMI (SNMI) sources, and BOR/POR/PUC sources of all the other modules. They are combined into three interrupt vectors. The interrupt vector registers SYSRSTIV, SYSSNIV, and SYSUNIV are used to determine which flags requested an interrupt or a BOR/POR/PUC reset. The interrupt with the highest priority of a group, when enabled, generates a number in the corresponding SYSRSTIV, SYSSNIV, or 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, or SYSUNIV values. A read access to the SYSRSTIV, SYSSNIV, or SYSUNIV register automatically resets the highest pending interrupt flag of that register. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt. A write access to the SYSRSTIV, SYSSNIV, or SYSUNIV register automatically resets all pending interrupt flags of the group.

1.3.3.1 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 MSP430x09x device. Vectors can change in priority for a given device. See the device-specific data sheet for the vector locations. All vectors should be coded symbolically to allow for easy portability of code.

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

1.4 Interrupts

Interrupt priorities are fixed and defined by the arrangement of the modules in the connection chain as shown in Figure 1-1. Interrupt priorities determine what interrupt is taken when more than one interrupt is pending simultaneously.

There are three types of interrupts:

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

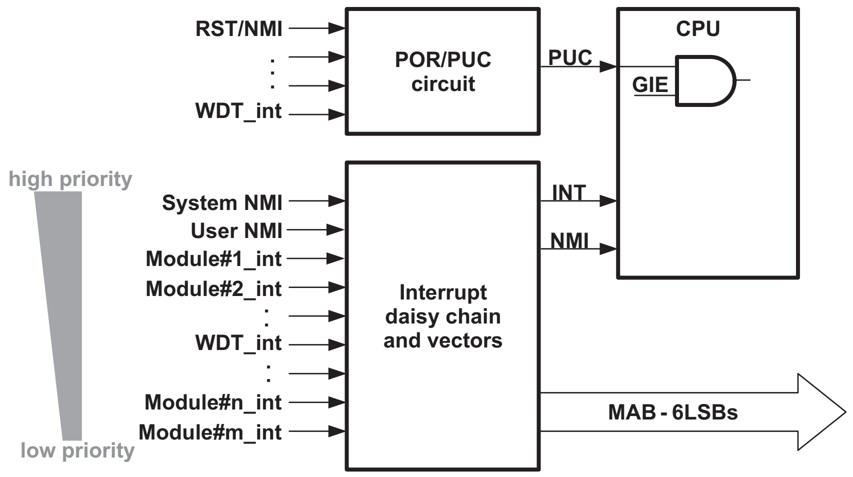


Figure 1-1. Interrupt Priority

1.4.1 (Non)-Maskable Interrupts (NMI)

The MSP430x09x family supports two levels of NMI interrupts, system NMI (SNMI) and user NMI (UNMI). In general, (non)-maskable NMI interrupts are not masked by the general interrupt enable bit (GIE). The user NMI sources are enabled by individual interrupt enable bits (NMIIE, ACCVIE, and OFIE). When a user NMI interrupt is accepted, other NMIs of that level are automatically disabled to prevent nesting of consecutive NMIs of the same level. Program execution begins at the address stored in the (non)-maskable interrupt vector as shown in Table 1-4. To allow software backward compatibility to users of earlier MSP430 families, the software may, but does not need to, re-enable user NMI sources. The block diagram for NMI sources is shown in Figure 1-2.

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

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

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

- SVM supply voltage fault
- Vacant memory access
- JTAG mailbox event

1.4.2 SNMI Timing

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

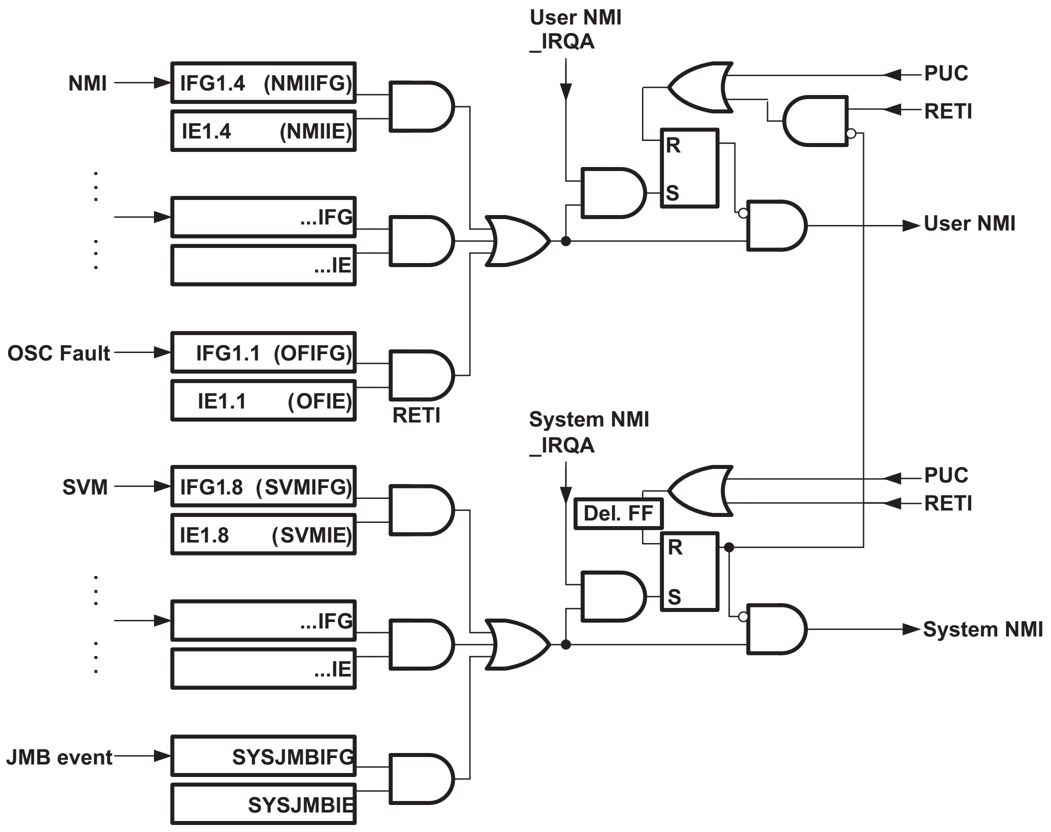


Figure 1-2. NMI Interrupts with Reentrance Protection

1.4.3 Maskable Interrupts

Maskable interrupts are caused by peripherals with interrupt capability. Each maskable interrupt source can be disabled individually by an interrupt enable bit, or all maskable interrupts can be disabled by the general interrupt enable (GIE) bit in the status register (SR).

Each individual peripheral interrupt is discussed in its respective module chapter of this manual.

1.4.3.1 Interrupt Processing

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

1.4.3.2 Interrupt Acceptance

The interrupt latency is 6 cycles, starting with the acceptance of an interrupt request and lasting until the start of execution of the first instruction of the interrupt-service routine, as shown in Figure 1-3. The interrupt logic executes the following:

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

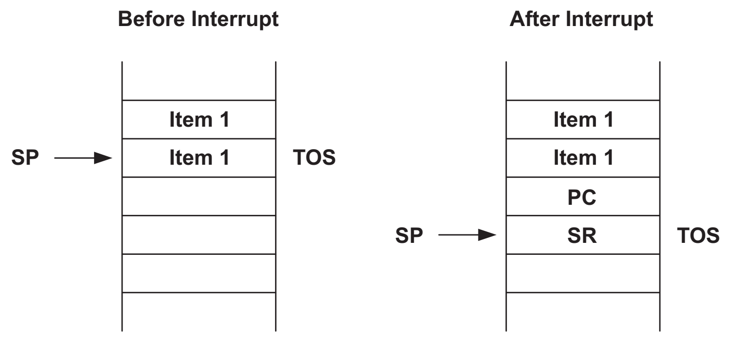


Figure 1-3. Interrupt Processing

1.4.3.3 Return From Interrupt

The interrupt handling routine terminates with the instruction:

`RETI` (return from an interrupt service routine)

The return from the interrupt takes 5 cycles to execute the following actions and is shown in Figure 1-4

1. The SR with all previous settings pops from the stack. All previous settings of GIE, CPUOFF, etc. are now in effect, regardless of the settings used during the interrupt service routine.
2. The PC pops from the stack and begins execution at the point where it was interrupted.

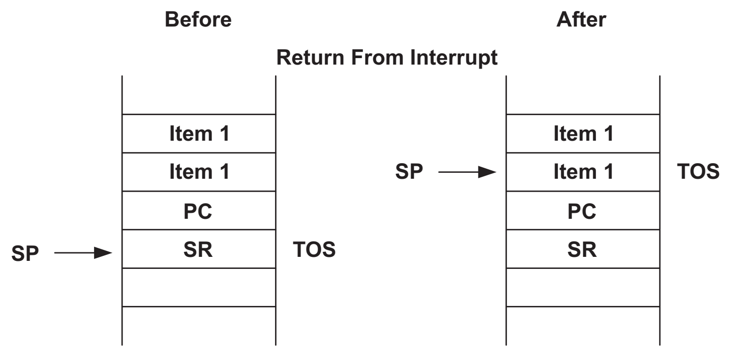


Figure 1-4. Return From Interrupt

1.4.3.4 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.

1.4.3.5 Interrupt Nesting of NMIs

A user NMI is always be able to interrupt the service program of any maskable interrupt. A user NMI is not able to interrupt another user NMI. A system NMI is always able to interrupt the services program of any maskable interrupt and any user NMI. A system NMI is not able to interrupt another system NMI. Any reset (BOR, POR, or PUC) is able to interrupt any ongoing program and restart the system.

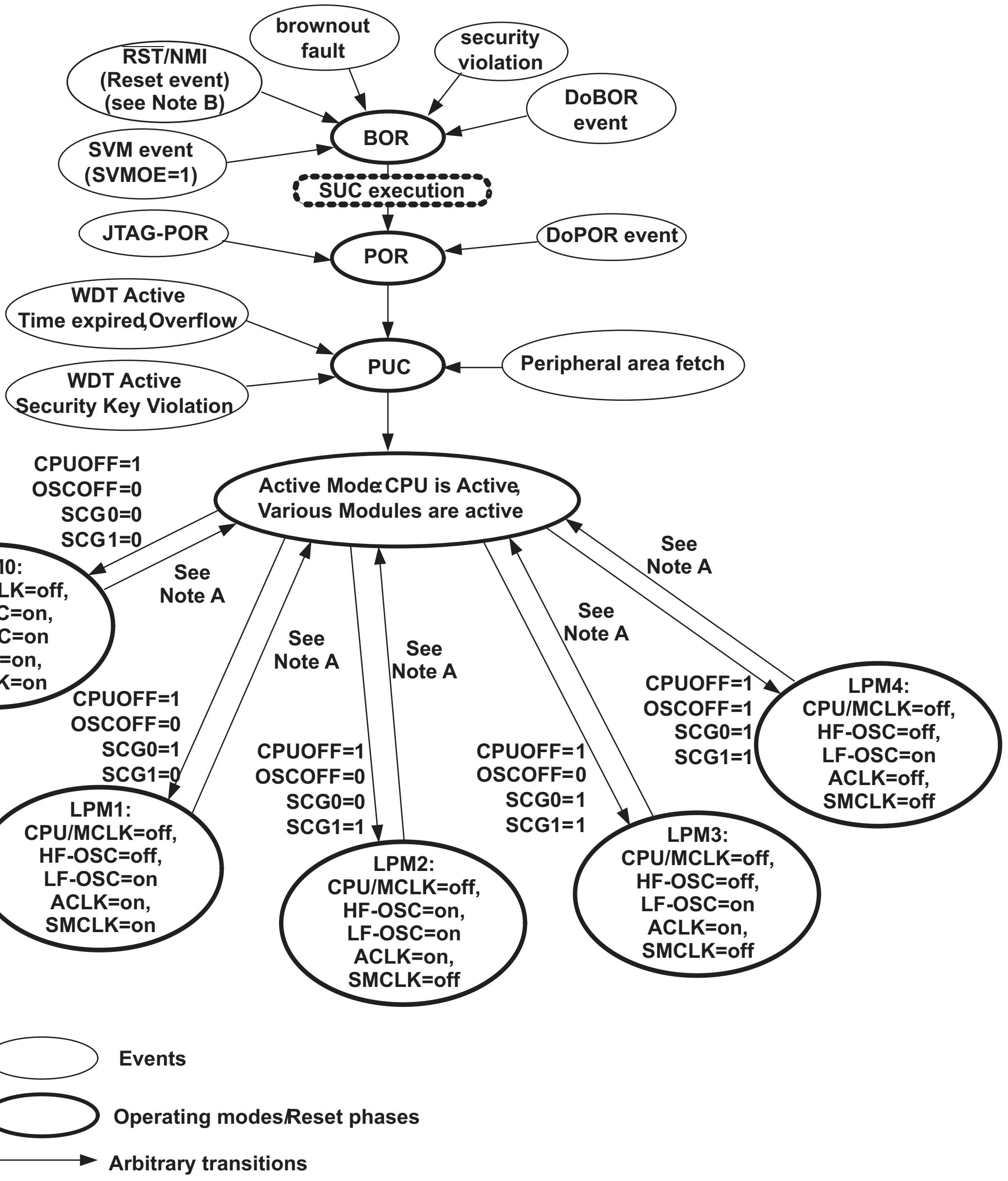
1.5 Operating Modes

The MSP430 family is designed for ultra low power applications and uses different operating modes. The operating modes for the MSP430x09x family are shown in Figure 1-5 and Table 1-2. See the device-specific data sheet for the operating modes available.

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 status register. The advantage of including the CPUOFF, OSCOFF, SCG0, and SCG1 mode-control bits in the status register is that the present operating mode is saved onto the stack during an interrupt service routine. Program flow returns to the previous operating mode if the saved SR value is not altered during the interrupt service routine. Program flow can be returned to a different operating mode by manipulating the saved SR value on the stack inside of the interrupt service routine. The mode-control bits and the stack can be accessed with any instruction. When setting any of the mode-control bits, the selected operating mode takes effect immediately. Peripherals operating with any disabled clock are disabled until the clock becomes active. The peripherals may also be disabled with their individual control register settings. All I/O port pins and RAM/registers are unchanged. Wake-up is possible through all enabled interrupts.


brownout fault
security violation
RST/NMI (Reset event) (see Note B)
DoBOR event
SVM event (SVMOE=1)
BOR
SUC execution
JTAG-POR
DoPOR event
POR
WDT Active Time expired, Overflow
PUC
Peripheral area fetch
WDT Active Security Key Violation
CPUOFF=1 OSCOFF=0 SCG0=0 SCG1=0
Active Mode: CPU is Active, Various Modules are active
See Note A
LPM0: CPU/MCLK=off, HF-OSC=on, LF-OSC=on ACLK=on, SMCLK=on
CPUOFF=1 OSCOFF=0 SCG0=1 SCG1=0
LPM1: CPU/MCLK=off, HF-OSC=off, LF-OSC=on ACLK=on, SMCLK=on
CPUOFF=1 OSCOFF=0 SCG0=0 SCG1=1
LPM2: CPU/MCLK=off, HF-OSC=on, LF-OSC=on ACLK=on, SMCLK=off
CPUOFF=1 OSCOFF=0 SCG0=1 SCG1=1
LPM3: CPU/MCLK=off, HF-OSC=off, LF-OSC=on ACLK=on, SMCLK=off
CPUOFF=1 OSCOFF=1 SCG0=1 SCG1=1
LPM4: CPU/MCLK=off, HF-OSC=off, LF-OSC=on ACLK=off, SMCLK=off
Events
Operating modes/Reset phases
Arbitrary transitions

A Any enabled interrupt and NMI performs this transition.

B An enabled reset always restarts the device.

Figure 1-5. Operation Modes for the MSP430x09x Family

Table 1-2. Operating Modes for the MSP430x09x Family

SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is enabled. MCLK, ACLK, SMCLK, and VLOCLK are active.
0	0	0	1	LPM0	CPU is disabled. MCLK is inactive. ACLK, SMCLK, and VLOCLK are active. HF-OSC is on.
0	1	0	1	LPM1	CPU is disabled. MCLK is inactive. ACLK, SMCLK, and VLOCLK are active. HF-OSC is off (LF-OSC is used instead where HF-OSC is selected).
1	0	0	1	LPM2	CPU is disabled. MCLK and SMCLK are inactive. ACLK and VLOCLK are active. HF-OSC is on.
1	1	0	1	LPM3	CPU is disabled. MCLK and SMCLK are inactive. ACLK and VLOCLK are active. HF-OSC is off (LF-OSC is used instead where HF-OSC is selected).
1	1	1	1	LPM4	CPU is disabled. MCLK, ACLK, and SMCLK are inactive. VLOCLK is active. HF-OSC is off. LF-OSC is on.

1.5.1 Entering and Exiting Low-Power Modes

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

- Enter interrupt service routine:
  - The PC and SR are stored on the stack
  - The CPUOFF, SCG1, and OSCOFF bits are automatically reset (SCG0 remains as is)
- 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
```

1.6 Principles for Low-Power Applications

Often, the most important factor for reducing power consumption is using the MSP430's clock system to maximize the time in LPM3 or LPM4 modes whenever possible.

- Use interrupts to wake the processor and control program flow.
- Peripherals should be switched on only when needed.
- Use low-power integrated peripheral modules in place of software driven functions. For example Timer0_A3 and Timer1_A3 can automatically generate PWM and capture external timing with no CPU resources.
- Calculated branching and fast table lookups should be used in place of flag polling and long software calculations.
- Avoid frequent subroutine and function calls due to overhead.
- For longer software routines, single-cycle CPU registers should be used.

1.7 Connection of Unused Pins

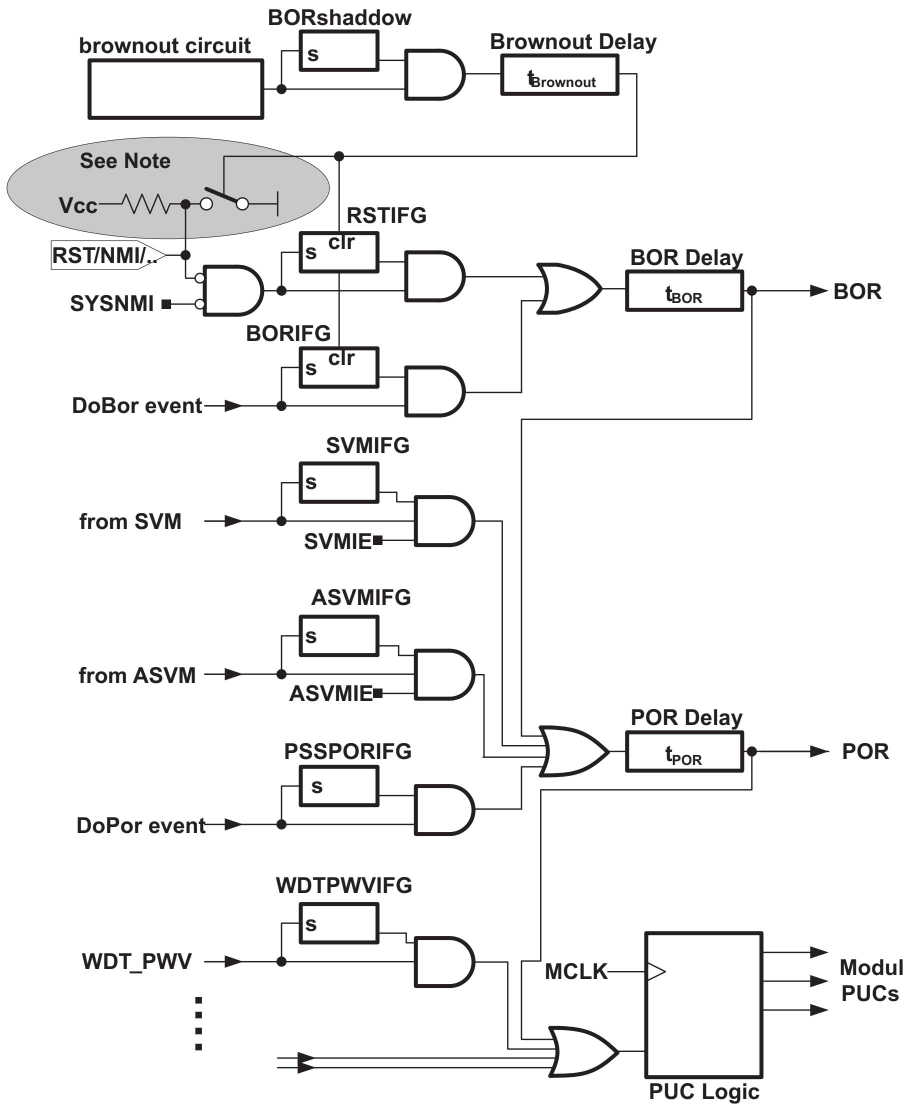
The correct termination of all unused pins is listed in Table 1-3.

Table 1-3. Connection of Unused Pins

Pin	Potential	Comment
TDI/TMS/TCK	Open	When used for JTAG function
$\overline{\text{RST}}$/NMI/SVMOUT	V_{CC} or V_{SS}	10-nF capacitor to GND/V_{SS}
Px.0 to Px.7	Open	Switched to port function, output direction
TDO	Open	Convention: leave TDO terminal as JTAG function

1.8 Reset and Subtypes

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



NOTE: See Figure 1-7

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

1.9 RST/NMI/SVMOUT Logic

The exact signal path from brownout circuit to the reset logic is shown in Figure 1-7. The external RST/NMI/SVMOUT terminal is pulled low on a brownout condition. The RST terminal can be used as reset source for the rest of the application. Setting SVMOE high enables the internal SVM logic to fire resets as soon a SVM event is detected. SVM events can be undervoltage or overvoltage conditions.

PortsOn is a control signal that enables logical output terminals. On a brownout condition all logical output terminals (except RST/NMI/SVMOUT) are forced to high impedance. SVMPD enables the circuit to force those logical output terminals to high impedance during on SVM events. SVMPO captures those events and has to be cleared to reenable the logical outputs. Brownout clears SVMPO.

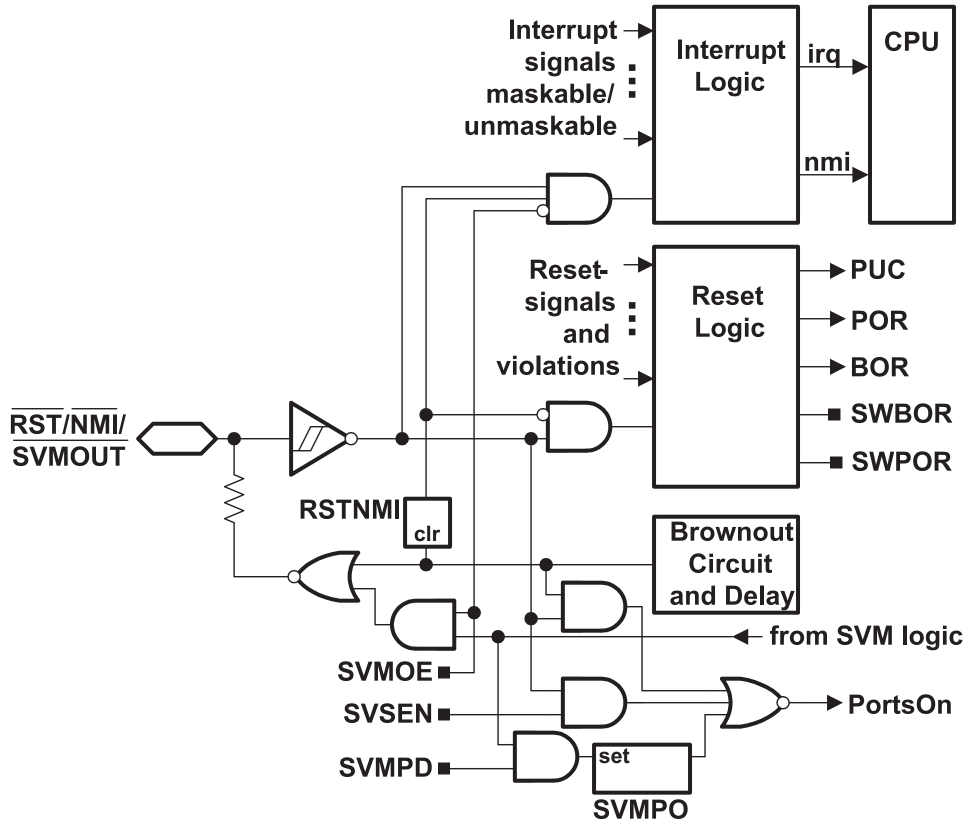


Figure 1-7. RST/NMI/SVMOUT Circuit

1.10 Interrupt Vectors

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

Table 1-4. Interrupt Sources, Flags, and Vectors

Interrupt Source	Interrupt Flag	System Interrupt	Word Address	Priority
Reset: Power-up, external reset, watchdog	WDTIFG, KEYV	… Reset	… 0FFFEh	Highest
System NMI		(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	SWI	(maskable)	⋮	Lowest

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

1.11 Special Function Registers

The special function registers, SFR, are listed in Table 1-6. The base address for the SFR registers is listed in Table 1-5.

Table 1-5. SFR Base Address

Module	Base address
SFR	00100h

Table 1-6. Special Function Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Interrupt enable register	SFRIE1	read/write	word	00h	0000h
	SFRIE1_L (IE1)	read/write	byte	00h	00h
	SFRIE1_H (IE2)	read/write	byte	01h	00h
Interrupt flag register	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 register	SFRRPCR	read/write	word	04h	000Eh
	SFRRPCR_L	read/write	byte	04h	0Eh
	SFRRPCR_H	read/write	byte	05h	00h

SFRIFG1, SFRIFG1_L, SFRIFG1_H, Interrupt Flag Register

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

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

Reserved	Bits 15-9	Reserved. Reads back as 0.
SVMIFG	Bit 8	SVM interrupt flag. This bit signals that the A-POOL comparator signaled an SVM event either low voltage or high voltage depending on setup 0 No interrupt pending 1 Interrupt pending
JMBOUTIFG	Bit 7	JTAG mailbox output interrupt flag 0 No interrupt pending. When in 16-bit mode (JMBMODE = 0), this bit is cleared automatically when JMBO0 has been written by the CPU. When in 32-bit mode (JMBMODE = 1), this bit is cleared automatically when both JMBO0 and JMBO1 have been written by the CPU. This bit is also cleared when the associated vector in SYSSNIV has been read 1 Interrupt pending, JMBO registers are ready for new messages. In 16-bit mode (JMBMODE = 0) JMBO0 has been received by JTAG. In 32-bit mode (JMBMODE = 1) , JMBO0 and JMBO1 have been received by JTAG
JMBINIFG	Bit 6	JTAG mailbox input interrupt flag 0 No interrupt pending. When in 16-bit mode (JMBMODE = 0), this bit is cleared automatically when JMBI0 is read by the CPU. When in 32-bit mode (JMBMODE = 1), this bit is cleared automatically when both JMBI0 and JMBI1 have been read by the CPU. This bit is also cleared when the associated vector in SYSSNIV has been read 1 Interrupt pending, a message is waiting in the JMBIN registers. In 16-bit mode (JMBMODE = 0) when JMBI0 has been written by JTAG. In 32 bit mode (JMBMODE = 1) when JMBI0 and JMBI1 have been written by JTAG
Reserved	Bit 5	Reserved. Reads back as 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 as 0.
OFIFG	Bit 1	Oscillator fault interrupt flag 0 No interrupt pending 1 Interrupt pending
WDTIFG	Bit 0	Watchdog timer interrupt flag. In watchdog mode, WDTIFG remains set until reset by software. In interval mode, WDTIFG is reset automatically by servicing the interrupt, or can be reset by software. Because other bits in ~IFG1 may be used for other modules, it is recommended to clear WDTIFG by using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instructions. 0 No interrupt pending 1 Interrupt pending

SFRIE1, SFRIE1_L, SFRIE1_H, Interrupt Enable Register

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

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

Reserved	Bits 15-9	Reserved. Reads back as 0.
SVMIE	Bit 8	SVM interrupt enable flag. 0 Interrupts disabled 1 Interrupts enabled
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
Reserved	Bit 5	Reserved. Reads back as 0.
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 as 0.
OFIE	Bit 1	Oscillator fault interrupt enable flag. 0 Interrupts disabled 1 Interrupts enabled
WDTIE	Bit 0	Watchdog timer interrupt enable. This bit enables the WDTIFG interrupt for interval timer mode. It is not necessary to set this bit for watchdog mode. Because other bits in ~IE1 may be used for other modules, it is recommended to set or clear this bit using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instruction 0 Interrupts disabled 1 Interrupts enabled

SFRRPCR, SFRRPCR_H, SFRRPCR_L, Reset Pin Control Register

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

7	6	5	4	3	2	1	0
Reserved				**SYSRSTRE**	**SYSRSTUP**	**SYSNMIES**	**SYSNMI**

Reserved	Bits 15-4	Reserved. Reads back as 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
SYSNMIES	Bit 1	NMI edge select. This bit selects the interrupt edge for the NMI interrupt when SYSNMI = 1. Modifying this bit can trigger an NMI. Modify this bit when SYSNMI = 0 to avoid triggering an accidental NMI 0 NMI on rising edge 1 NMI on falling edge
SYSNMI	Bit 0	NMI select. This bit selects the function for the $\overline{\text{RST}}$/NMI pin 0 Reset function 1 NMI function

1.12 CSYS Registers

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

Table 1-7. CSYS Base Address

Module	Base Address
CSYS	00180h

Table 1-8. SYS Configuration Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
System control register	SYSCTL	read/write	word	00h	0000h
	SYSCTLL	read/write	byte	00h	00h
	SYSCTLH	read/write	byte	01h	00h
JTAG mailbox control register	SYSJMBC	read/write	word	06h	000Ch
	SYSJMBCL	read/write	byte	06h	0Ch
	SYSJMBH	read/write	byte	07h	00h
JTAG mailbox input register 0	SYSJMBI0	read/write	word	08h	0000h
	SYSJMBI0L	read/write	byte	08h	00h
	SYSJMBI0H	read/write	byte	09h	00h
JTAG mailbox input register 1	SYSJMBI1	read/write	word	0Ah	0000h
	SYSJMBI1L	read/write	byte	0Ah	00h
	SYSJMBI1H	read/write	byte	0Bh	00h
JTAG mailbox output register 0	SYSJMBO0	read/write	word	0Ch	0000h
	SYSJMBO0L	read/write	byte	0Ch	00h
	SYSJMBO0H	read/write	byte	0Dh	00h
JTAG mailbox output register 1	SYSJMBO1	read/write	word	0Eh	0000h
	SYSJMBO1L	read/write	byte	0Eh	00h
	SYSJMBO1H	read/write	byte	0Fh	00h
System configuration register	SYSCNF	read/write	word	10h	0300h
	SYSCNFL	read/write	byte	10h	00h
	SYSCNFH	read/write	byte	11h	03h
Bus error vector generator	SYSBERRIV	read only	word	18h	0000h
User NMI vector generator	SYSUNIV	read only	word	1Ah	0000h
System NMI vector generator	SYSSNIV	read only	word	1Ch	0000h
Reset vector generator	SYSRSTIV	read only	word	1Eh	0002h

SYSCTL, SYSCTL_L, SYSCTL_H, SYS Control Register

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved		**SYSTEDIS**	**ETUNLOCK DIS**	**Reserved**		**SYSJTAGDIS**	**Reserved**
r0	r0	rw-[0]	rw-[0]	r0	r0	rw-[0]	r0

7	6	5	4	3	2	1	0
Reserved		**SYSJTAGPIN**	**Reserved**				**SYSRIVECT**
r0	r0	r1	r0	r0	r0	r0	rw-[0]

Reserved	Bits 15-14	Reserved. Reads back as 0.
SYSTEDIS	Bit 13	Internally used for control purposes
ETUNLOCKDIS	Bit 12	ET wrapper control bit. This bit is electrically "ORed" with the ETUNLOCKDIS bit from JTAG to generate the final ETUNLOCKDIS signal. If one of those bits is set to unlock, then the unlock function is enabled. 0 Unlocked ET wrapper locks after access again except ETUNLOCDIS in JTAG is set. 1 Unlocked ET wrapper stays unlocked after accesses
Reserved	Bits 11-10	Reserved. Reads back as 0.
SYSJTAGDIS	Bit 9	JTAG disable 0 JTAG enabled 1 JTAG disabled
Reserved	Bits 8-6	Reserved. Reads back as 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 set, it remains set until a BOR occurs. 0 SBW is primary JTAG interface 1 Explicit 4-wire JTAG is primary JTAG interface
Reserved	Bits 4-1	Reserved. Reads back as 0.
SYSRIVECT	Bit 0	RAM based Interrupt Vectors 0 Interrupt vectors generated with end address: top of lower 64k memory 0FFFFh 1 Interrupt vectors generated with end address: top of RAM

SYSJMBC, SYSJMBC_L, SYSBMBC_H, JTAG Mailbox Control Register

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

7	6	5	4	3	2	1	0
JMBCLR1OFF	**JMBCLR0OFF**	**Reserved**	**JMBMODE**	**JMBMODE**	**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 as 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 as 0.
JMBMODE	Bit 4	This bit defined the operation mode of JMB for JMBI0/1 and JMBO0/1. Before switching this bit pad and flush out any partial content to avoid data drops 0 16 bit transfers using JMBO0 and JMBI0 only 1 32 bit transfers using JMBO0/1 and JMBI0/1
JMBMODE	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, etc.) 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, etc.) 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 JMBI1. This flag is cleared automatically on read of JMBI0 when JMBCLR0OFF = 0 (auto clear mode). On JMBCLR0OFF = 1 JMBIN1FG needs to be cleared by SW 0 JMBI0 has no new data 1 JMBI0 has new data available

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

15	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
MSGLO							
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

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

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
MSGLO							
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

SYSCNF, SYSCNFL, SYSCNFH SYS Configuration Register

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved						**RAMLCK1**	**RAMLCK0**
r0	r0	r0	r0	r0	r0	rw-[1]	rw-[1]

7	6	5	4	3	2	1	0
Reserved		**SVMEN**	**SVMPD**	**SVMPO**	**SVMOE**	**Reserved**	
r0	r0	rw-0	rw-0	rw-{0}	rw-0	r0	r0

Reserved	Bits 15-10	Reserved. Reads back as 0.	
RAMLCK1	Bit 9	Write lock enable for application's code RAM	
		0	Write accesses to code RAM are possible
		1	Write accesses to code RAM are ignored
RAMLCK0	Bit 8	Write lock enable for configuration RAM (128 Bytes of RAM)	
		0	Write accesses to configuration RAM are possible
		1	Write accesses to configuration RAM are ignored
Reserved	Bits 7-6	Reserved. Reads back as 0.	
SVMEN	Bit 5	SVM input enable	
		0	SVM input disabled
		1	SVM input enabled
SVMPD	Bit 4	SVM based port disable	
		0	SVM event has no impact to the SVMPO bit
		1	SVM event sets SVMPO that forces all PORTS (except $\overline{\text{RST}}$/NMI/SVMOUT) in high impedance
SVMPO	Bit 3	SVM based Ports off flag. Can be set by SVM when SVMPD = 1	
		0	PortsOn signal is not forced to zero
		1	PortsOn signal is forced to zero
SVMOE	Bit 2	SVM output enable	
		0	SVM events do not drive $\overline{\text{RST}}$/NMI/SVMOUT
		1	SVM event pulls $\overline{\text{RST}}$/NMI/SVMOUT pin low
Reserved	Bits 1-0	Reserved. Reads back as 0.	

SYSBERRIV, SYSBERRIV_H, SYSBERRIV_L, Bus Error Interrupt Vector Register

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

7	6	5	4	3	2	1	0
0	**0**	**0**	**SYSBERRVEC**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

SYSBERRVEC	Bits 4-1	Bus error interrupt vector. It generates a value that can be used as address offset for fast interrupt service routine handling for bus errors. Check the data sheet of the particular device for the corresponding bus error table. Writing to this register clears all pending bus error interrupt flags. Reading this register clears the highest pending bus error (displaying this register with the debugger does not affect its content).	
		0000h	No interrupt pending
		0002h...	Valid bus error from peripheral (see device-specific data sheet)

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.

SYSUNIV, SYSUNIV_H, SYSUNIV_L, User NMI Interrupt Vector Register

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

7	6	5	4	3	2	1	0
0	**0**	**0**	**SYSUNVEC**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

SYSUNVEC Bits 4-1 User NMI interrupt vector. It 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 interrupt flags. Reading this register clears the highest pending interrupt flag (displaying this register with the debugger does not affect its content).

Value	Interrupt Type
0000h	No interrupt pending
0002h	NMIIFG interrupt pending (highest priority)
0004h	OFIFG interrupt pending
0006h	Bus error interrupt pending (check SYSBERRIV)
⋮	Reserved for future extensions

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

SYSSNIV, SYSSNIV_H, SYSSNIV_L, SYS NMI Interrupt Vector Register

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

7	6	5	4	3	2	1	0
0	**0**	**0**	**SYSSNVEC**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

SYSSNVEC Bits 4-1 System NMI interrupt vector. It 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 interrupt flags. Reading this register clears the highest pending interrupt flag (displaying this register with the debugger does not affect its content).

Value	Interrupt Type
0000h	No interrupt pending
0002h	SVMIFG interrupt pending (highest priority)
0004h	VMAIFG interrupt pending
0006h	JMBINIFG interrupt pending
0008h	JMBOUTIFG interrupt pending
⋮	Reserved for future extensions

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

SYSRSTIV, SYSRSTIV_H, SYSRSTIV_L, SYS Reset Interrupt Vector Register

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

7	6	5	4	3	2	1	0
0	**0**	**0**	**SYSRSTVEC**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

SYSRSTVEC	Bits 4-1	Reset interrupt vector. It 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. Reading this register clears the highest pending interrupt flag (displaying this register with the debugger does not affect its content). Signals that alter the $\overline{\text{RST}}$/NMI pin generate a double event. A SVMBOR generates an SVMBOR vector followed by an $\overline{\text{RST}}$/NMI BOR vector; a brownout also causes a $\overline{\text{RST}}$/NMI vector).

Value	Interrupt Type
0000h	No interrupt pending
0002h	Brownout (BOR) (highest priority)
0004h	$\overline{\text{RST}}$/NMI (BOR)
0006h	DoBOR (BOR)
0008h	Security violation (BOR)
000Ah	DoPOR (POR)
000Ch	WDT time out (PUC)
000Eh	WDT key violation (PUC)
0010h	PERF peripheral/configuration area fetch (PUC)
⋮	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.

1.13 CSYS PMM Register Replica

The CSYS module hosts the control bits PMMSWBOR and PMMSWPOR for devices that do not have an own power management module (PMM). The function of those bits are explained in the PMMCTL0 definition.

Power Management Control 0 Register, PMMCTL0

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

7	6	5	4	3	2	1	0
Reserved				**PMMSWPOR**	**PMMSWBOR**	**Reserved**	
r0	r0	r0	r0	rw-(0)	rw-[0]	r0	r0

PMMKEY	Bits 15-8	PMM password. Always read as 096h. Must be written with 0A5h or a PUC is generated.
Reserved	Bits 7-4	Reserved. Reads back as 0.
PMMSWPOR	Bit 3	Software power-on reset. Setting this bit to one triggers a POR. This bit is self clearing.
PMMSWBOR	Bit 2	Software brownout reset. Setting this bit to one triggers a BOR. This bit is self clearing.
Reserved	Bits 1-0	Reserved. Reads back as 0.

Compact Clock System (CCS)

The compact clock system (CCS) module provides the clocks for the MSP430x09x device family. This chapter describes the operation of the CCS module.

Topic **Page**

2.1 Compact Clock System (CCS) Introduction

The CCS module is used to generate the system clocks required for the MSP430x09x family devices. Using two internal clock signals and one external, the user can select the best balance of performance and low power consumption. The Compact Clock System module can be configured to operate without any external components.

The Compact Clock System includes three clock sources

- HFCLK: From an internal high-frequency oscillator (HF-OSC) in the 1-MHz range that can be trimmed to 3% accuracy by the user software.
- LFCLK: From an internal low-frequency oscillator (LF-OSC) in the 20-kHz range that is designed for low power with consumption currents in the sub-microampere range.
- CLKIN: Optional external clock source that can be operated from dc up to 4-MHz range.

Four clock signals are available from the CCS module:

- ACLK: Auxiliary clock. ACLK is sourced from HFCLK, LFCLK, or CLKIN. ACLK is software selectable for individual peripheral modules. ACLK can be divided by 1, 2, 4, 8, 16 or 32.
- MCLK: Master clock. MCLK is sourced from HFCLK, LFCLK, or CLKIN. 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 sourced from HFCLK, LFCLK, or CLKIN. SMCLK is software selectable for individual peripheral modules. SMCLK can be divided by 1, 2, 4, 8,16 or 32.
- VLOCLK: Low-frequency clock as permanent clock source.

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

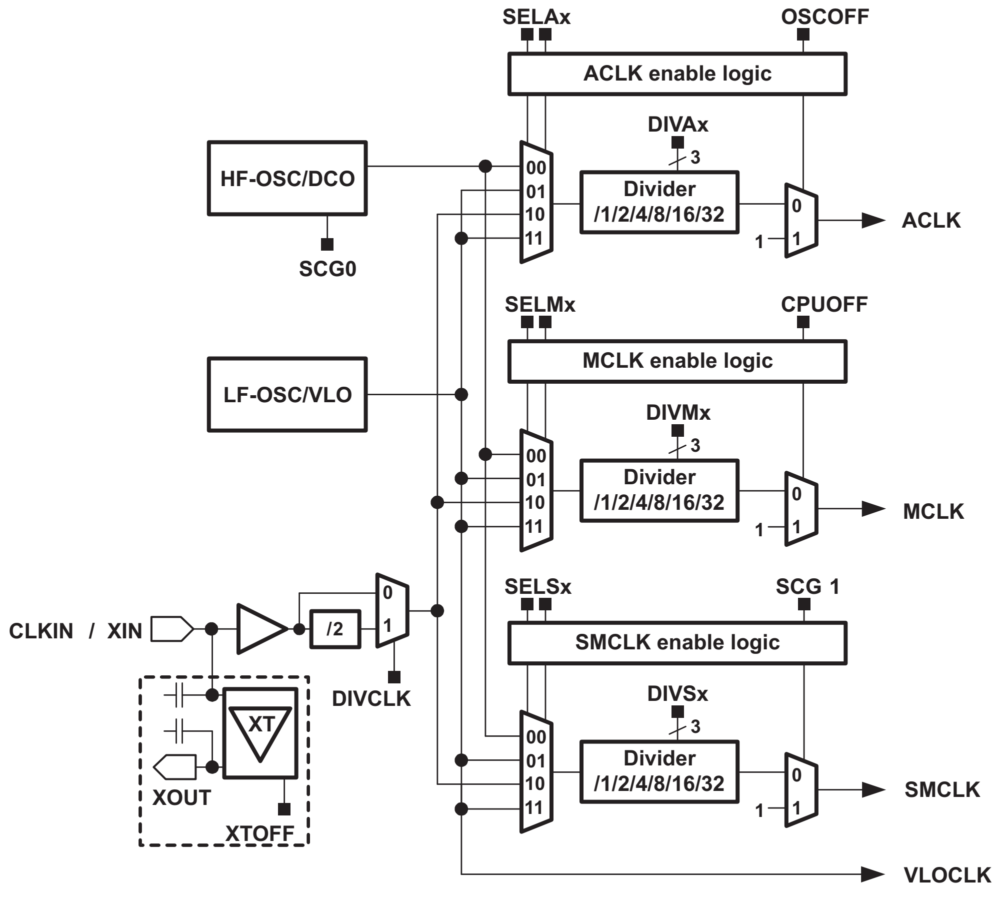


Figure 2-1. CCS Block Diagram

2.2 CCS Module Operation

After a PUC, the CSS module default configuration is:

- HF-OSC is selected as the source for ACLK. ACLK defaults to divide by 8.
- HF-OSC is selected as the source for MCLK. MCLK defaults to divide by 8
- HF-OSC is selected as the source for SMCLK. SMCLK defaults to divide by 8.

Status register control bits (SCG1, OSCOFF, and CPUOFF) configure the MSP430 operating modes and enable or disable portions of the CCS module (see System Resets, Interrupts, and Operating Modes chapter). The CCSCTLx registers configure the CCS module. The CCS module can be configured or reconfigured by software at any time during program execution.

2.2.1 Operation From Low-Power Modes Requested by Peripheral Modules

Peripheral modules can request a clock from the CCS module if their state of operation still requires an operational clock. A peripheral module asserts one of three possible clock request signals: ACLK_REQ, MCLK_REQ, or SMCLK_REQ.

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

2.2.2 Internal Low-Frequency Oscillator

The internal ultra-low-voltage low-frequency oscillator (LF-OSC) provides a typical frequency of 20 kHz (see device-specific data sheet for parameters) without requiring a crystal. The LF-OSC provides for a low-cost ultra-low-voltage clock source for applications that do not need an accurate time base.

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

2.2.3 Internal Trimmable High-Frequency Oscillator

The internal trimmable ultra-low-voltage high-frequency oscillator (HF-OSC) can be used for cost-sensitive applications in which an external clock source is not required or desired. The high-frequency oscillator may be internally trimmed and provides a stable frequency that can be used as input into all clock trees. The typical frequency of the HF-OSC is 1 MHz.

HF-OSC is selected when it is used to source ACLK, MCLK, or SMCLK (SELA = 0, SELM = 0, or SELS = 0).

2.2.4 External Clock Source

CLKIN may be used with external clock signals on the CLKIN-pin by selecting CLKIN as source. When used with an external signal, the external frequency must meet the data sheet parameters.

CLKIN is selected under any of the following conditions:

- CLKIN/XIN is a source for ACLK (SELA = 2 and OSCOFF = 0)
- CLKIN/XIN is a source for MCLK (SELM = 2 and CPUOFF = 0)
- CLKIN/XIN is a source for SMCLK (SELS = 2 and SCG1 = 0)

The CLKIN/XIN pin is shared with a general-purpose I/O port. CLKIN/XIN is used as clock source. CLKIN/XIN is not checked for the presence of a valid clock. When CLKIN/XIN is configured as analog input pin CLKIN/XIN cannot be used as clock input.

2.2.5 Compact Clock System Module Fail-Safe Operation

The Compact Clock System module incorporates an oscillator-fault fail-safe feature. This feature detects an oscillator fault for HF-OSC and LF-OSC as shown in Figure 2-2. The available fault conditions are:

- HF-OSC fault (HFOFFG)
- X-OSC fault (XOFFG)

The oscillator fault bits are set if the corresponding oscillator is turned on and not operating properly. The fault bits remain set as long as the fault condition exists and need to be cleared by software.

A fault of the HF-OSC is detected if no transition on HFCLK is sensed for a period of up to five LFCLK cycles. If this is detected, all clock trees sourced by HFCLK are sourced by LFCLK instead.

A fault of the CLKIN/XIN is detected when the optional oscillator on XIN stops. If this is detected, all clock trees sourced by CLKIN/XIN are sourced by LFCLK instead.

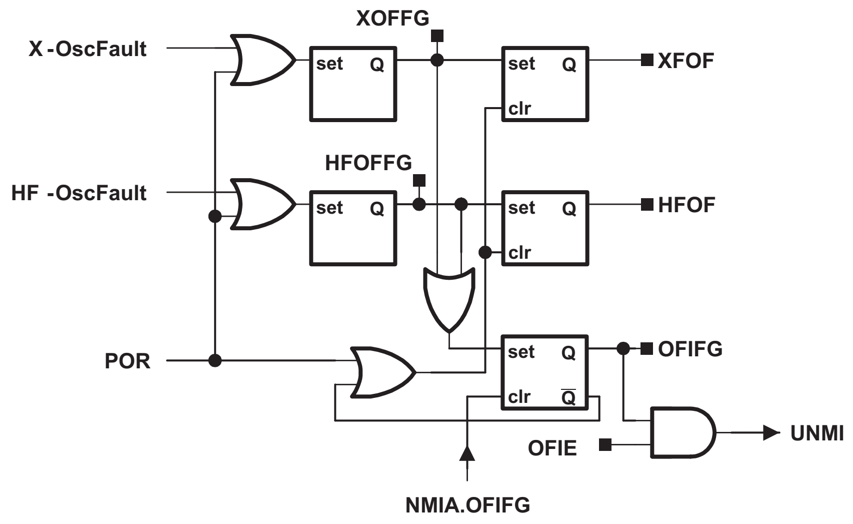


Figure 2-2. Oscillator Fault Logic for Devices With HF-OSC

2.3 CCS Module Registers

The CCS module registers and their address offsets are listed in Table 2-2. The base for the CCS registers is listed in Table 2-1. The password defined in the CCSCTL0 register controls access to the CCS 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 CCSCTL0 upper byte. Word accesses to CCSCTL0 with a wrong password triggers a PUC. A write access to a register other than CCSCTL0 while write access is not enabled causes a PUC.

Table 2-1. CCS Register Base Address

Module	Base Address
CCS	001A0h

Table 2-2. CCS Control Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Compact clock system control 0	CCSCTL0	read/write	word	00h	9600h
	CCSCTL0_L		byte	00h	00h
	CCSCTL0_H		byte	01h	96h
Compact clock system control 1	CCSCTL1	read/write	word	02h	0001h
	CCSCTL1_L		byte	02h	01h
	CCSCTL1_H		byte	03h	00h
Compact clock system control 2	CCSCTL2	read/write	word	04h	0028h
	CCSCTL2_L		byte	04h	28h
	CCSCTL2_H		byte	05h	00h
Compact clock system control 4	CCSCTL4	read/write	word	08h	0100h
	CCSCTL4_L		byte	08h	00h
	CCSCTL4_H		byte	09h	01h
Compact clock system control 5	CCSCTL5	read/write	word	0Ah	0333h
	CCSCTL5_L		byte	0Ah	33h
	CCSCTL5_H		byte	0Bh	03h
Compact clock system control 6	CCSCTL6	read/write	word	0Ch	0001h
	CCSCTL6_L		byte	0Ch	01h
	CCSCTL6_H		byte	0Dh	00h
Compact clock system control 7	CCSCTL7	read/write	word	0Eh	0003h
	CCSCTL7_L		byte	0Eh	03h
	CCSCTL7_H		byte	0Fh	00h
Compact clock system control 8	CCSCTL8	read/write	word	10h	0007h
	CCSCTL8_L		byte	10h	07h
	CCSCTL8_H		byte	11h	00h

CCSCTL0, Compact Clock System Control 0 Register

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

CCSKEY	Bits 15-8	CCSKEY password. Value 0x00h locks Control Register. Must always be written with A5h to unlock or 00h to lock. Any other values generate a PUC. Always read as 96h.
Reserved	Bits 7-0	Reserved. Reads back as 0.

CCSCTL1, Compact Clock System Control 1 Register

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							**DIVCLK**
r0	r0	r0	r0	r0	r0	r0	rw-[1]

Reserved	Bits 15-1	Reserved. Reads back as 0.
DIVCLK	Bit 0	Clock division for CLKIN / X-OSC
		0 CLKIN / X-OSC is directly used for clock generation
		1 CLKIN/ X-OSC is divided by two for clock generation

CCSCTL2, Compact Clock System Control 2 Register

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	**FSELx**						
r0	rw-[0]	rw-[1]	rw-[0]	rw-[1]	rw-[0]	rw-[0]	rw-[0]

Reserved	Bits 15-7	Reserved. Reads back as 0.
FSELx	Bits 6-0	Frequency trimming of the HF-OSC (applies only for devices that feature the HF-OSC)

0000000	Highest adjustable frequency
⋮	⋮
0101000	Center frequency
⋮	⋮
1111111	Lowest adjustable frequency

CCSCTL4, Compact Clock System Control 4 Register

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

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

Reserved	Bits 15-10	Reserved. Reads back as 0.
SELAx	Bits 9-8	Select the ACLK source 00 HFCLK / DCO 01 LFCLK / VLO 10 CLKIN / X-OSC 11 Reserved (defaults to LFCLK / VLO)
Reserved	Bits 7-6	Reserved. Reads back as 0.
SELSx	Bits 5-4	Select the SMCLK source 00 HFCLK / DCO 01 LFCLK / VLO 10 CLKIN / X-OSC 11 Reserved (defaults to LFCLK / VLO)
Reserved	Bits 3-2	Reserved. Reads back as 0.
SELMx	Bits 1-0	Select the MCLK source 00 HFCLK / DCO 01 LFCLK / VLO 10 CLKIN / X-OSC 11 Reserved (defaults to LFCLK / VLO)

CCSCTL5, Compact Clock System Control 5 Register

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

7	6	5	4	3	2	1	0
Reserved	DIVSx			Reserved	DIVMx		
r0	rw-0	rw-1	rw-1	r0	rw-0	rw-1	rw-1

Reserved	Bits 15-11	Reserved. Reads back as 0.
DIVAx	Bits 10-8	ACLK source divider
		000 /1
		001 /2
		010 /4
		011 /8
		100 /16
		101 /32
		110 Reserved, defaults to /32
		111 Reserved, defaults to /32
Reserved	Bit 7	Reserved. Reads back as 0.
DIVSx	Bits 6-4	SMCLK source divider
		000 /1
		001 /2
		010 /4
		011 /8
		100 /16
		101 /32
		110 Reserved, defaults to /32
		111 Reserved, defaults to /32
Reserved	Bit 3	Reserved. Reads back as 0.
DIVMx	Bits 2-0	MCLK source divider
		000 /1
		001 /2
		010 /4
		011 /8
		100 /16
		101 /32
		110 Reserved, defaults to /32
		111 Reserved, defaults to /32

CCSCTL6, Compact Clock System Control 6 Register

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							XTOFF
r0	r0	r0	r0	r0	r0	r0	rw-1

Reserved	Bits 15-1	Reserved. Reads back as 0.
XTOFF	Bit 0	XT off. This bit turns off the XT oscillator.
		0 XT is on if XT is selected via the port selection.
		1 XT is off if it is not used as a source for ACLK, MCLK, or SMCLK.

CCSCTL7, Compact Clock System Control 7 Register

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						**XOFFG**	**HFOFFG**
r0	r0	r0	r0	r0	r0	rw-(1)	rw-(1)

Reserved	Bits 15-2	Reserved. Reads back as 0.
XOFFG	Bit 1	Crystal oscillator (X-OSC) fault flag. If this bit is set, the OFIFG flag is also set. XOFFG is set if a X-OSC fault condition exists. The XOFFG can be cleared via software. If the X-OSC fault condition still remains, the XOFFG remains set. 0 No fault condition occurred after the last reset. 1 X-OSC fault. A X-OSC fault occurred after the last reset.
HFOFFG	Bit 0	High-frequency oscillator (HF-OSC) fault flag. If this bit is set, the OFIFG flag is also set. HFOFFG is set if a HF-OSC fault condition exists. The HFOFFG can be cleared via software. If the HF-OSC fault condition still remains, the HFOFFG remains set. 0 No fault condition occurred after the last reset. 1 HF-OSC fault. A HF-OSC fault occurred after the last reset.

CCSCTL8, Compact Clock System Control 8 Register

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					**SMCLKREQEN**	**MCLKREQEN**	**ACLKREQEN**
r0	r0	r0	r0	r0	rw-(1)	rw-(1)	rw-(1)

Reserved	Bits 15-3	Reserved. Reads back as 0.
SMCLKREQEN	Bit 2	SMCLK clock request enable. Peripheral modules can request the SMCLK to remain switched on. 0 SMCLK is turned off if the respective low power mode is entered. 1 SMCLK remains active in case a peripheral is using it even if it is selected to be switched off according to the current low–power mode.
MCLKREQEN	Bit 1	MCLK clock request enable. Peripheral modules can request the MCLK to remain switched on. 0 MCLK is turned off if the respective low power mode is entered. 1 MCLK remains active in case a peripheral is using it even if it is selected to be switched off according to the current low–power mode.
ACLKREQEN	Bit 0	ACLK clock request enable. Peripheral modules can request the ACLK to remain switched on. Refer to the device specific data sheet which peripheral supports the clock request. 0 ACLK is turned off if the respective low power mode is entered. 1 ACLK remains active in case a peripheral is using it even if it is selected to be switched off according to the current low–power mode.

CPU

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

Topic **Page**

3.1 CPU Introduction

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

The CPU features include:

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

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

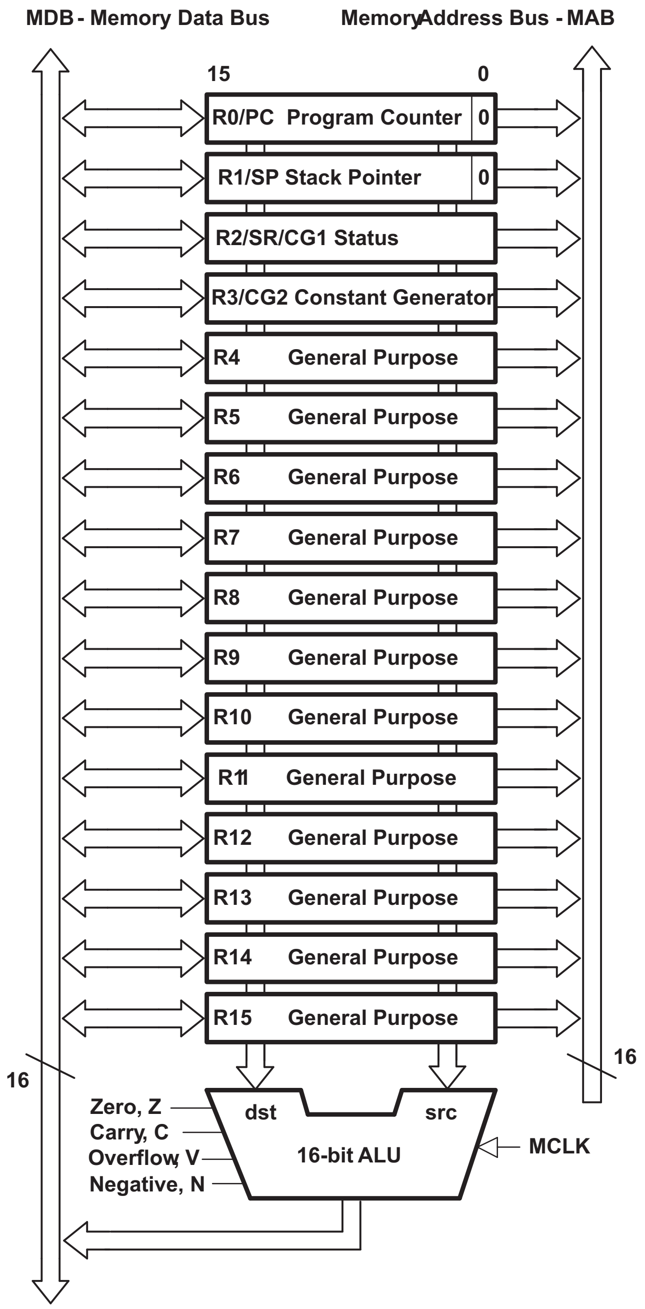


Figure 3-1. CPU Block Diagram

3.2 CPU Registers

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

3.2.1 Program Counter (PC)

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

Figure 3-2. Program Counter

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Program Counter Bits 15 to 1															0

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

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

3.2.2 Stack Pointer (SP)

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

Figure 3-4 shows stack usage.

Figure 3-3. Stack Counter

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Stack Pointer Bits 15 to 1															0

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

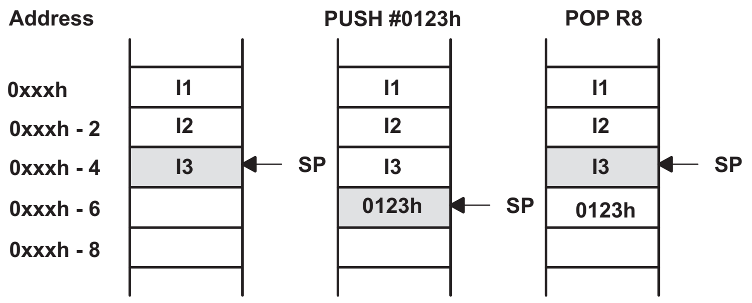
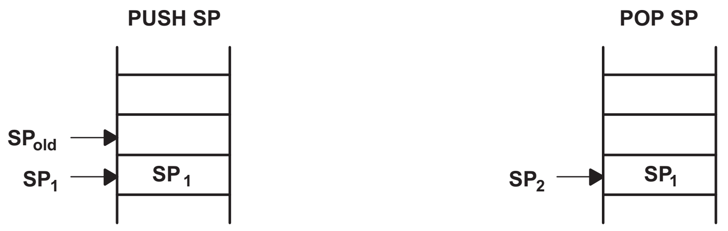


Figure 3-4. Stack Usage

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



The stack pointer is changed after a PUSH SP instruction.

The stack pointer is not changed after a POP SP instruction. The POP SP instruction places SP_1 into the stack pointer SP (SP_2=SP_1)

Figure 3-5. PUSH SP - POP SP Sequence

3.2.3 Status Register (SR)

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

Figure 3-6. Status Register Bits

15–9	8	7	6	5	4	3	2	1	0
Reserved	V	SCG1	SCG0	OSC OFF	CPU OFF	GIE	N	Z	C
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 3-1 describes the status register bits.

Table 3-1. Description of Status Register Bits

Bit	Description	
V	Overflow bit. This bit is set when the result of an arithmetic operation overflows the signed-variable range.	
	`ADD(.B),ADDC(.B)`	Set when:
		Positive + Positive = Negative
		Negative + Negative = Positive
		Otherwise reset
	`SUB(.B),SUBC(.B),CMP(.B)`	Set when:
		Positive – Negative = Negative
		Negative – Positive = Positive
		Otherwise reset
SCG1	System clock generator 1. When set, turns off the SMCLK.	
SCG0	System clock generator 0. When set, turns off the DCO dc generator, if DCOCLK is not used for MCLK or SMCLK.	
OSCOFF	Oscillator Off. When set, turns off the LFXT1 crystal oscillator, when LFXT1CLK is not use for MCLK or SMCLK.	
CPUOFF	CPU off. When set, turns off the CPU.	
GIE	General interrupt enable. When set, enables maskable interrupts. When reset, all maskable interrupts are disabled.	
N	Negative bit. Set when the result of a byte or word operation is negative and cleared when the result is not negative.	
	Word operation: N is set to the value of bit 15 of the result.	
	Byte operation: N is set to the value of bit 7 of the result.	
Z	Zero bit. Set when the result of a byte or word operation is 0 and cleared when the result is not 0.	
C	Carry bit. Set when the result of a byte or word operation produced a carry and cleared when no carry occurred.	

3.2.4 Constant Generator Registers CG1 and CG2

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

Table 3-2. Values of Constant Generators CG1, CG2

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

The constant generator advantages are:

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

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

3.2.4.1 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
```

3.2.5 General-Purpose Registers R4 to R15

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

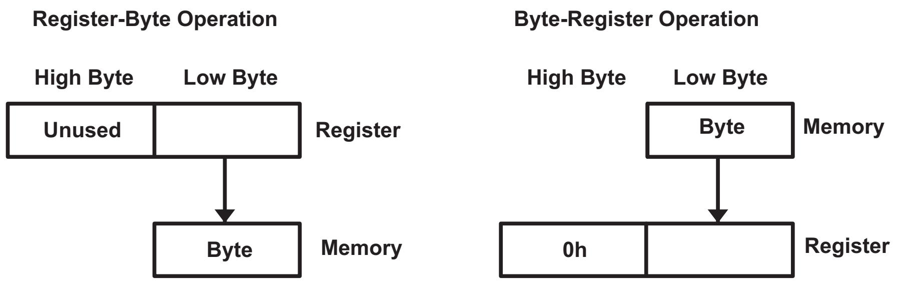


Figure 3-7. Register-Byte/Byte-Register Operations

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

3.3 Addressing Modes

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

Table 3-3. Source/Destination Operand Addressing Modes

As/Ad	Addressing Mode	Syntax	Description
00/0	Register mode	Rn	Register contents are operand
01/1	Indexed mode	X(Rn)	(Rn + X) points to the operand. X is stored in the next word.
01/1	Symbolic mode	ADDR	(PC + X) points to the operand. X is stored in the next word. Indexed mode X(PC) is used.
01/1	Absolute mode	&ADDR	The word following the instruction contains the absolute address. X is stored in the next word. Indexed mode X(SR) is used.
10/-	Indirect register mode	@Rn	Rn is used as a pointer to the operand.
11/-	Indirect autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards by 1 for .B instructions and by 2 for .W instructions.
11/-	Immediate mode	#N	The word following the instruction contains the immediate constant N. Indirect autoincrement mode @PC+ is used.

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

NOTE: Use of Labels *EDE, TONI, TOM, and LEO*

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

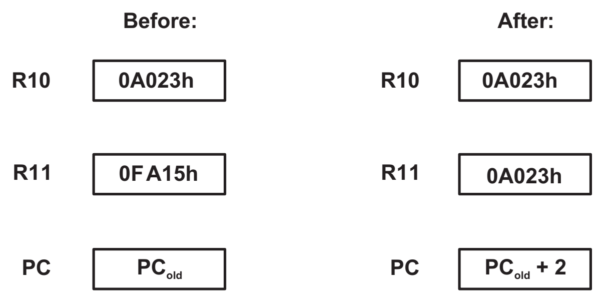
3.3.1 Register Mode

The register mode is described in Table 3-4.

Table 3-4. Register Mode Description

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

Length:	One or two words
Operation:	Move the content of R10 to R11. R10 is not affected.
Comment:	Valid for source and destination
Example:	`MOV R10,R11`



NOTE: Data in Registers

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

3.3.2 Indexed Mode

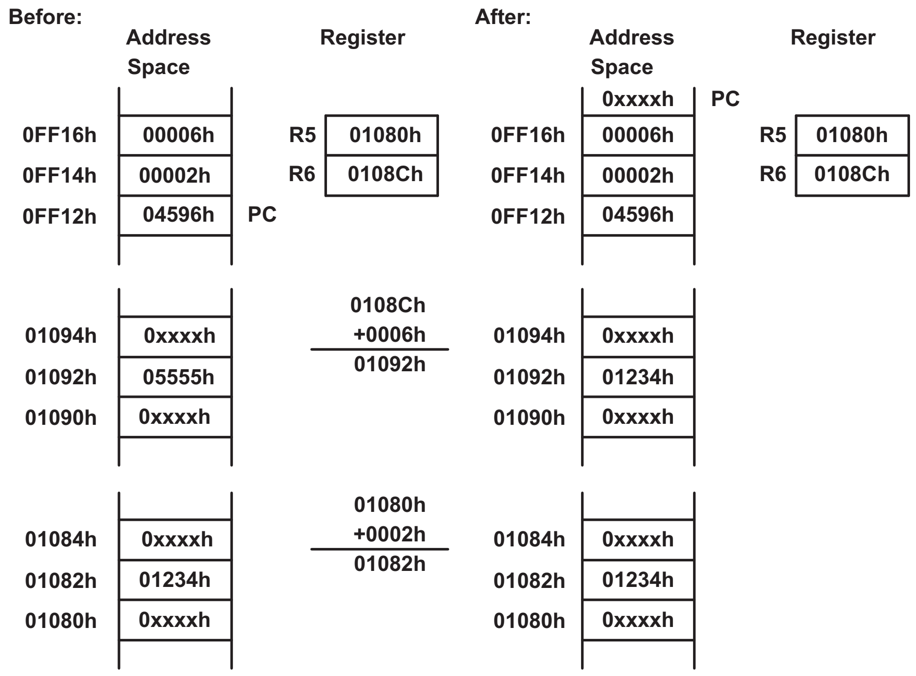
The indexed mode is described in Table 3-5.

Table 3-5. Indexed Mode Description

Assembler Code	Content of ROM

`MOV 2(R5),6(R6)`	`MOV X(R5),Y(R6)`
	X = 2
	Y = 6

Length:	Two or three words
Operation:	Move the contents of the source address (contents of R5 + 2) to the destination address (contents of R6 + 6). The source and destination registers (R5 and R6) are not affected. In indexed mode, the program counter is incremented automatically so that program execution continues with the next instruction.
Comment:	Valid for source and destination
Example:	`MOV 2(R5),6(R6);`



3.3.3 Symbolic Mode

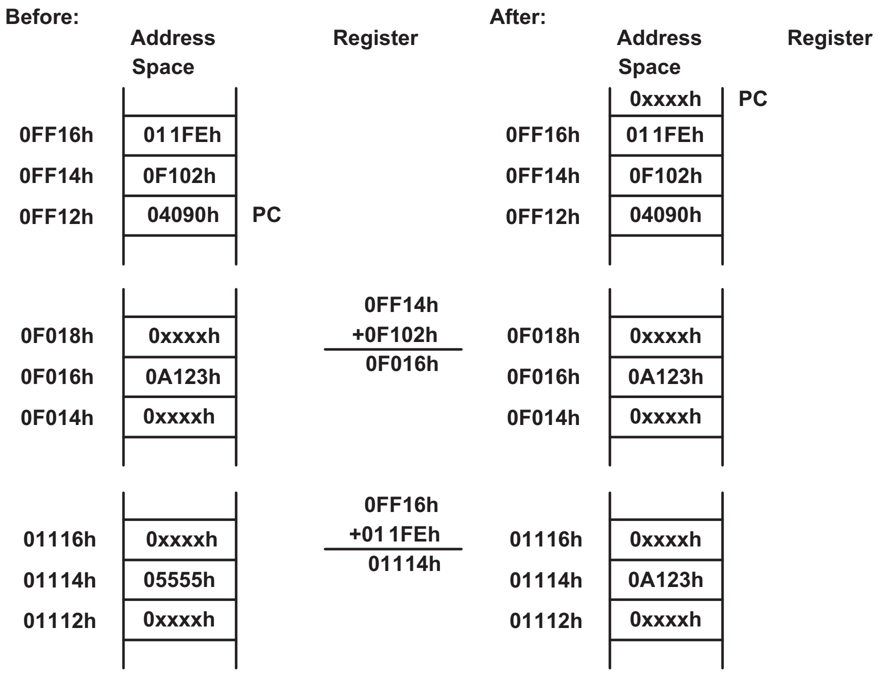
The symbolic mode is described in Table 3-6.

Table 3-6. Symbolic Mode Description

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

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

```
MOV  EDE,TONI  ;Source address EDE = 0F016h
               ;Dest. address TONI = 01114h
```



3.3.4 Absolute Mode

The absolute mode is described in Table 3-7.

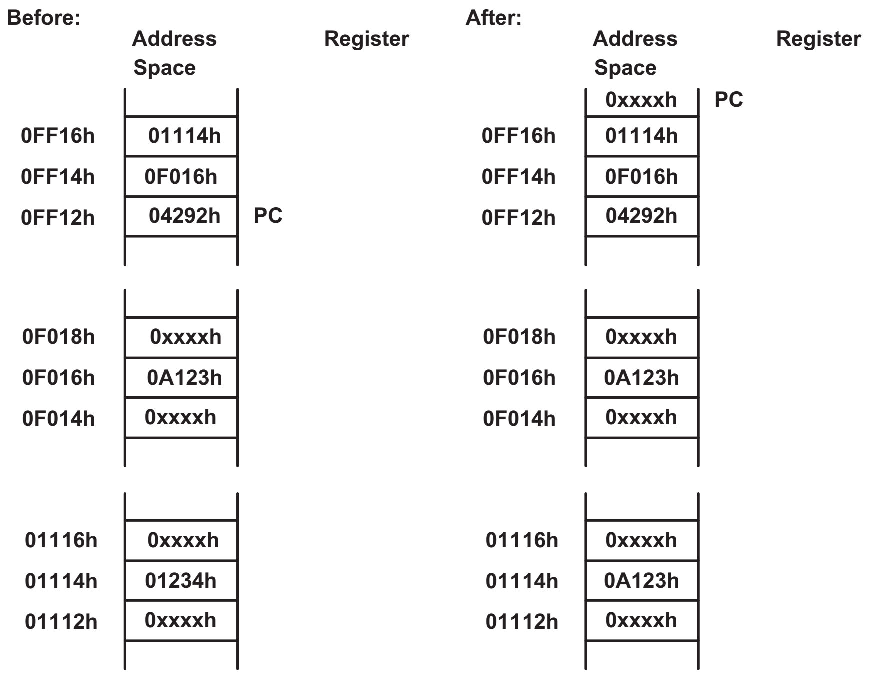
Table 3-7. Absolute Mode Description

Assembler Code	Content of ROM

`MOV &EDE,&TONI`	`MOV X(0),Y(0)`
	X = EDE
	Y = TONI

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

```
MOV  &EDE,&TONI  ;Source address EDE = 0F016h
                 ;Dest. address TONI = 01114h
```



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

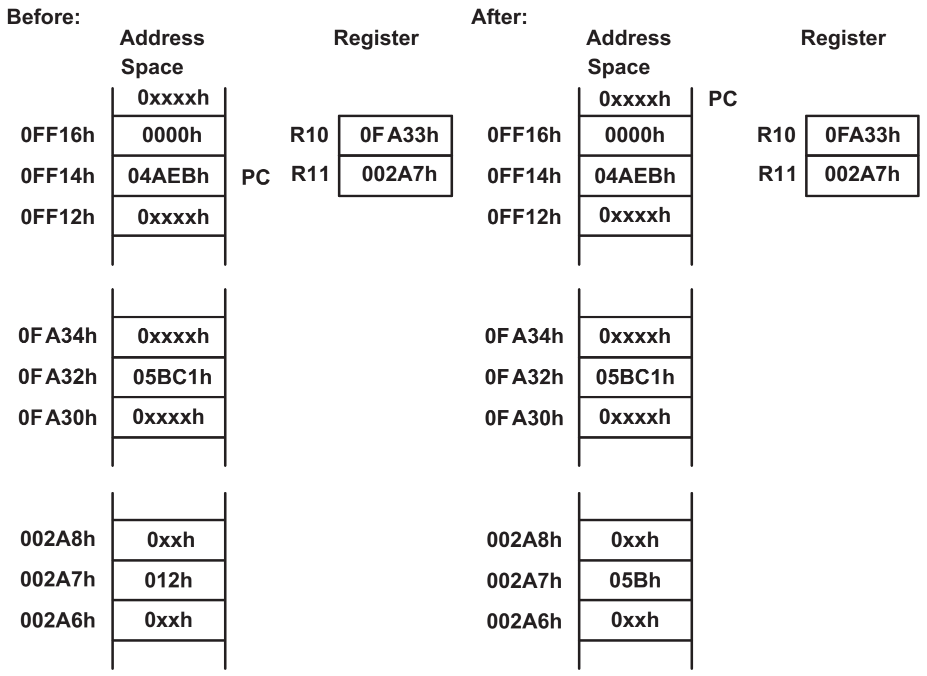
3.3.5 Indirect Register Mode

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

Table 3-8. Indirect Mode Description

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

Length:	One or two words
Operation:	Move the contents of the source address (contents of R10) to the destination address (contents of R11). The registers are not modified.
Comment:	Valid only for source operand. The substitute for destination operand is 0(Rd).
Example:	`MOV.B @R10,0(R11)`



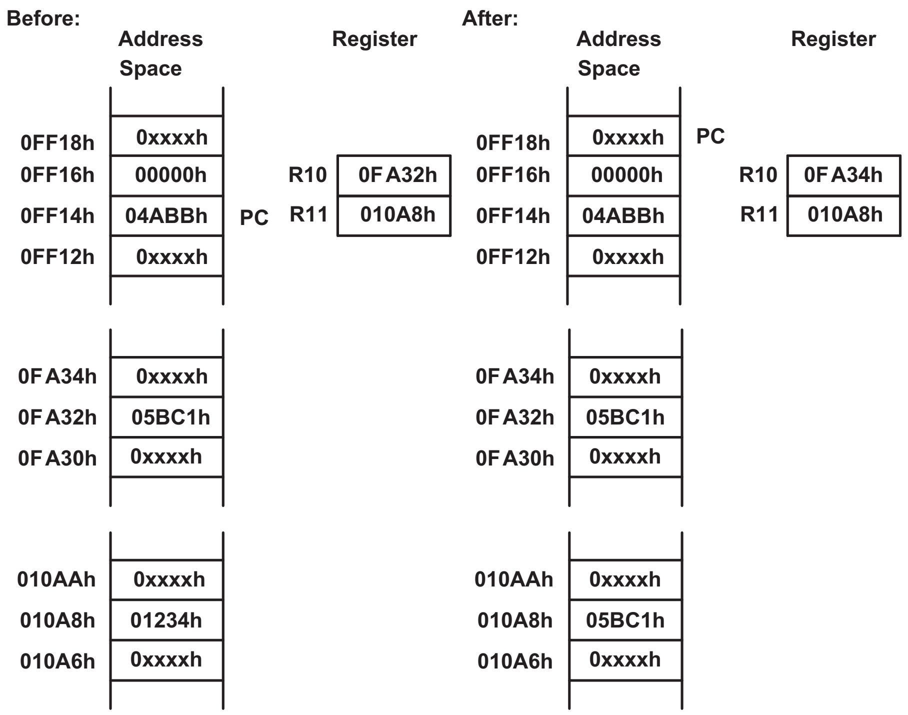
3.3.6 Indirect Autoincrement Mode

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

Table 3-9. Indirect Autoincrement Mode Description

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

Length:	One or two words
Operation:	Move the contents of the source address (contents of R10) to the destination address (contents of R11). Register R10 is incremented by 1 for a byte operation, or 2 for a word operation after the fetch; it points to the next address without any overhead. This is useful for table processing.
Comment:	Valid only for source operand. The substitute for destination operand is 0(Rd) plus second instruction INCD Rd.
Example:	`MOV @R10+,0(R11)`



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

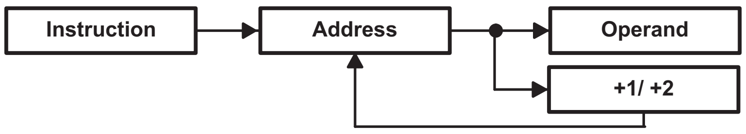


Figure 3-8. Operand Fetch Operation

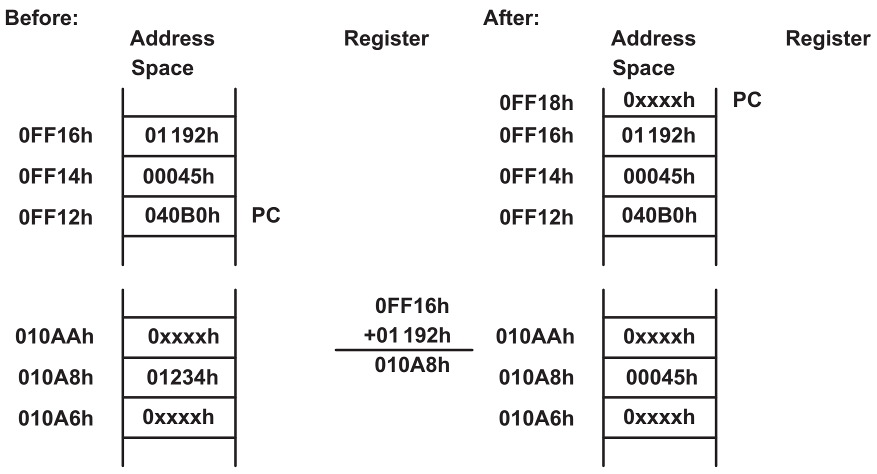
3.3.7 Immediate Mode

The immediate mode is described in Table 3-10.

Table 3-10. Immediate Mode Description

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

Length:	Two or three words It is one word less if a constant of CG1 or CG2 can be used.
Operation:	Move the immediate constant 45h, which is contained in the word following the instruction, to destination address TONI. When fetching the source, the program counter points to the word following the instruction and moves the contents to the destination.
Comment:	Valid only for a source operand.
Example:	`MOV #45h,TONI`



3.4 Instruction Set

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

There are three core-instruction formats:

- Dual-operand
- Single-operand
- Jump

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

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

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

NOTE: Destination Address

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

3.4.1 Double-Operand (Format I) Instructions

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

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

Figure 3-9. Double Operand Instruction Format

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

Table 3-11. Double Operand Instructions

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

*	The status bit is affected
–	The status bit is not affected
0	The status bit is cleared
1	The status bit is set

NOTE: Instructions CMP and SUB

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

3.4.2 Single-Operand (Format II) Instructions

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

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

Figure 3-10. Single Operand Instruction Format

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

Table 3-12. Single Operand Instructions

Mnemonic	S-Reg, D-Reg	Operation	Status Bits			
			V	N	Z	C
`RRC(.B)`	`dst`	C → MSB →.......LSB → C	*	*	*	*
`RRA(.B)`	`dst`	MSB → MSB →....LSB → C	0	*	*	*
`PUSH(.B)`	`src`	SP – 2 → SP, src → @SP	-	-	-	-
`SWPB`	`dst`	Swap bytes	-	-	-	-
`CALL`	`dst`	SP – 2 → SP, PC+2 → @SP	-	-	-	-
		dst → PC				
`RETI`		TOS → SR, SP + 2 → SP	*	*	*	*
		TOS → PC,SP + 2 → SP				
`SXT`	`dst`	Bit 7 → Bit 8........Bit 15	0	*	*	*

*	The status bit is affected
–	The status bit is not affected
0	The status bit is cleared
1	The status bit is set

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

3.4.3 Jumps

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

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

Figure 3-11. Jump Instruction Format

Table 3-13 lists and describes the jump instructions

Table 3-13. Jump Instructions

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

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

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

3.4.4 Instruction Set

***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 Bit

N: Set if result is negative, reset if positive

Z: Set if result is zero, reset otherwise

C: Set if dst was incremented from 0FFFFh to 0000, reset otherwise

Set if dst was incremented from 0FFh to 00, reset otherwise

V: Set if an arithmetic overflow occurs, otherwise reset

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

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

```
ADD   @R13,0(R12)   ; Add LSDs
ADC   2(R12)        ; Add carry to MSD
```

Example The 8-bit counter pointed to by R13 is added to a 16-bit counter pointed to by R12.

```
ADD.B   @R13,0(R12)   ; Add LSDs
ADC.B   1(R12)        ; Add carry to MSD
```

ADD[.W]	Add source to destination
ADD.B	Add source to destination

Syntax	`ADD src,dst  or  ADD.W src,dst`
	`ADD.B src,dst`

Operation	src + dst -> dst
Description	The source operand is added to the destination operand. The source operand is not affected. The previous contents of the destination are lost.
Status Bits	N: Set if result is negative, reset if positive Z: Set if result is zero, reset otherwise C: Set if there is a carry from the result, cleared if not V:Set if an arithmetic overflow occurs, otherwise reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.

Example R5 is increased by 10. The jump to TONI is performed on a carry.

```
ADD     #10,R5
JC      TONI      ; Carry occurred
......            ; No carry
```

Example R5 is increased by 10. The jump to TONI is performed on a carry.

```
ADD.B   #10,R5    ; Add 10 to Lowbyte of R5
JC      TONI      ; Carry occurred, if (R5) ≥ 246 [0Ah+0F6h]
......            ; No carry
```

ADDC[.W] Add source and carry to destination

ADDC.B Add source and carry to destination

Syntax

```
ADDC src,dst  or  ADDC.W src,dst
ADDC.B src,dst
```

Operation src + dst + C -> dst

Description The source operand and the carry bit (C) are added to the destination operand. The source operand is not affected. The previous contents of the destination are lost.

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

Z: Set if result is zero, reset otherwise

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

V: Set if an arithmetic overflow occurs, otherwise reset

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

Example The 32-bit counter pointed to by R13 is added to a 32-bit counter, eleven words (20/2 + 2/2) above the pointer in R13.

```
ADD      @R13+,20(R13)   ; ADD LSDs with no carry in
ADDC     @R13+,20(R13)   ; ADD MSDs with carry
...                      ; resulting from the LSDs
```

Example The 24-bit counter pointed to by R13 is added to a 24-bit counter, eleven words above the pointer in R13.

```
ADD.B    @R13+,10(R13)   ; ADD LSDs with no carry in
ADDC.B   @R13+,10(R13)   ; ADD medium Bits with carry
ADDC.B   @R13+,10(R13)   ; ADD MSDs with carry
...                      ; resulting from the LSDs
```

AND[.W] Source AND destination

AND.B Source AND destination

Syntax

```
AND src,dst  or  AND.W src,dst
AND.B src,dst
```

Operation

```
src .AND. dst -> dst
```

Description The source operand and the destination operand are logically ANDed. The result is placed into the destination.

Status Bits

N: Set if result MSB is set, reset if not set

Z: Set if result is zero, reset otherwise

C: Set if result is not zero, reset otherwise (= .NOT. Zero)

V: Reset

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

Example The bits set in R5 are used as a mask (#0AA55h) for the word addressed by TOM. If the result is zero, a branch is taken to label TONI.

```
MOV     #0AA55h,R5    ; Load mask into register R5
AND     R5,TOM        ; mask word addressed by TOM with R5
JZ      TONI          ;
......                ; Result is not zero
;
;
;   or
;
;
AND     #0AA55h,TOM
JZ      TONI
```

Example The bits of mask #0A5h are logically ANDed with the low byte TOM. If the result is zero, a branch is taken to label TONI.

```
AND.B   #0A5h,TOM      ; mask Lowbyte TOM with 0A5h
JZ      TONI           ;
......                 ; Result is not zero
```

BIC[.W] Clear bits in destination

BIC[.W] Clear bits in destination

Syntax

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

Operation .NOT.src .AND. dst -> dst

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

Status Bits Status bits are not affected.

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

Example The six MSBs of the RAM word LEO are cleared.

```
BIC    #0FC00h,LEO  ; Clear 6 MSBs in MEM(LEO)
```

Example The five MSBs of the RAM byte LEO are cleared.

```
BIC.B  #0F8h,LEO    ; Clear 5 MSBs in Ram location LEO
```

BIS[.W]	Set bits in destination
BIS.B	Set bits in destination

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

Operation	src .OR. dst -> dst
Description	The source operand and the destination operand are logically ORed. The result is placed into the destination. The source operand is not affected.
Status Bits	Status bits are not affected.
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The six LSBs of the RAM word TOM are set.

```
BIS    #003Fh,TOM   ; set the six LSBs in RAM location TOM
```

Example The three MSBs of RAM byte TOM are set.

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

BIT[.W] Test bits in destination

BIT.B Test bits in destination

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

Operation src .AND. dst

Description The source and destination operands are logically ANDed. The result affects only the status bits. The source and destination operands are not affected.

Status Bits N: Set if MSB of result is set, reset otherwise

Z: Set if result is zero, reset otherwise

C: Set if result is not zero, reset otherwise (.NOT. Zero)

V: Reset

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

Example If bit 9 of R8 is set, a branch is taken to label TOM.

```
BIT   #0200h,R8   ; bit 9 of R8 set?
JNZ   TOM   ; Yes, branch to TOM
...   ; No, proceed
```

Example If bit 3 of R8 is set, a branch is taken to label TOM.

```
BIT.B   #8,R8
JC   TOM
```

Example A serial communication receive bit (RCV) is tested. Because the carry bit is equal to the state of the tested bit while using the BIT instruction to test a single bit, the carry bit is used by the subsequent instruction; the read information is shifted into register RECBUF.

```
;
; Serial communication with LSB is shifted first:
                     ; xxxx   xxxx   xxxx   xxxx
BIT.B   #RCV,RCCTL   ; Bit info into carry
RRC     RECBUF       ; Carry -> MSB of RECBUF
                     ; cxxx    xxxx
......               ; repeat previous two instructions
......               ; 8 times
                     ; cccc   cccc
                     ; ^         ^
                     ; MSB       LSB
; Serial communication with MSB shifted first:
BIT.B   #RCV,RCCTL   ; Bit info into carry
RLC.B   RECBUF       ; Carry -> LSB of RECBUF
                     ; xxxx   xxxc
......               ; repeat previous two instructions
......               ; 8 times
                     ; cccc   cccc
                     ; |
                     ; MSB       LSB
```

***BR, BRANCH** Branch to .... destination

Syntax `BR dst`

Operation dst -> PC

Emulation `MOV dst,PC`

Description An unconditional branch is taken to an address anywhere in the 64K address space. All source addressing modes can be used. The branch instruction is a word instruction.

Status Bits Status bits are not affected.

Example Examples for all addressing modes are given.

```
BR  #EXEC  ; Branch to label EXEC or direct branch (e.g. #0A4h)
           ; Core instruction MOV @PC+,PC
BR  EXEC   ; Branch to the address contained in EXEC
           ; Core instruction MOV X(PC),PC
           ; Indirect address
BR  &EXEC  ; Branch to the address contained in absolute
           ; address EXEC
           ; Core instruction MOV X(0),PC
           ; Indirect address
BR  R5     ; Branch to the address contained in R5
           ; Core instruction MOV R5,PC
           ; Indirect R5
BR  @R5    ; Branch to the address contained in the word
           ; pointed to by R5.
           ; Core instruction MOV @R5+,PC
           ; Indirect, indirect R5
BR  @R5+   ; Branch to the address contained in the word pointed
           ; to by R5 and increment pointer in R5 afterwards.
           ; The next time--S/W flow uses R5 pointer--it can
           ; alter program execution due to access to
           ; next address in a table pointed to by R5
           ; Core instruction MOV @R5,PC
           ; Indirect, indirect R5 with autoincrement
BR  X(R5)  ; Branch to the address contained in the address
           ; pointed to by R5 + X (e.g. table with address
           ; starting at X). X can be an address or a label
           ; Core instruction MOV X(R5),PC
           ; Indirect, indirect R5 + X
```

CALL Subroutine

Syntax `CALL dst`

Operation

dst -> tmp	dst is evaluated and stored
SP - 2 -> SP	
PC -> @SP	PC updated to TOS
tmp -> PC	dst saved to PC

Description A subroutine call is made to an address anywhere in the 64K address space. All addressing modes can be used. The return address (the address of the following instruction) is stored on the stack. The call instruction is a word instruction.

Status Bits Status bits are not affected.

Example Examples for all addressing modes are given.

```
CALL  #EXEC  ; Call on label EXEC or immediate address (e.g. #0A4h)
             ; SP-2 -> SP, PC+2 -> @SP, @PC+ -> PC
CALL  EXEC   ; Call on the address contained in EXEC
             ; SP-2 -> SP, PC+2 ->SP, X(PC) -> PC
             ; Indirect address
CALL  &EXEC  ; Call on the address contained in absolute address
             ; EXEC
             ; SP-2 -> SP, PC+2 -> @SP, X(0) -> PC
             ; Indirect address
CALL  R5     ; Call on the address contained in R5
             ; SP-2 -> SP, PC+2 -> @SP, R5 -> PC
             ; Indirect R5
CALL  @R5    ; Call on the address contained in the word
             ; pointed to by R5
             ; SP-2 -> SP, PC+2 -> @SP, @R5 -> PC
             ; Indirect, indirect R5
CALL  @R5+   ; Call on the address contained in the word
             ; pointed to by R5 and increment pointer in R5.
             ; The next time S/W flow uses R5 pointer
             ; it can alter the program execution due to
             ; access to next address in a table pointed to by R5
             ; SP-2 -> SP, PC+2 -> @SP, @R5 -> PC
             ; Indirect, indirect R5 with autoincrement
CALL  X(R5)  ; Call on the address contained in the address pointed
             ; to by R5 + X (e.g. table with address starting at X)
             ; X can be an address or a label
             ; SP-2 -> SP, PC+2 -> @SP, X(R5) -> PC
             ; Indirect, indirect R5 + X
```

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

Syntax

```
CLR dst  or  CLR.W dst
CLR.B dst
```

Operation 0 -> dst

Emulation

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

Description The destination operand is cleared.

Status Bits Status bits are not affected.

Example RAM word TONI is cleared.

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

Example Register R5 is cleared.

```
CLR    R5
```

Example RAM byte TONI is cleared.

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

***CLRC**	Clear carry bit
Syntax	`CLRC`
Operation	0 -> C
Emulation	`BIC #1,SR`
Description	The carry bit (C) is cleared. The clear carry instruction is a word instruction.
Status Bits	N: Not affected Z: Not affected C: Cleared V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The 16-bit decimal counter pointed to by R13 is added to a 32-bit counter pointed to by R12.

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

***CLRN** Clear negative bit

Syntax `CLRN`

Operation 0 -> N

or

(.NOT.src .AND. dst -> dst)

Emulation `BIC #4,SR`

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

Status Bits N: Reset to 0

Z: Not affected

C: Not affected

V: Not affected

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

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

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

***CLRZ**	Clear zero bit
Syntax	`CLRZ`
Operation	0 -> Z or (.NOT.src .AND. dst -> dst)
Emulation	`BIC #2,SR`
Description	The constant 02h is inverted (0FFFDh) and logically ANDed with the destination operand. The result is placed into the destination. The clear zero bit instruction is a word instruction.
Status Bits	N: Not affected Z: Reset to 0 C: Not affected V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	The zero bit in the status register is cleared. `CLRZ`

CMP[.W] Compare source and destination

CMP.B Compare source and destination

Syntax

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

Operation dst + .NOT.src + 1

or

(dst - src)

Description The source operand is subtracted from the destination operand. This is accomplished by adding the 1s complement of the source operand plus 1. The two operands are not affected and the result is not stored; only the status bits are affected.

Status Bits N: Set if result is negative, reset if positive (src ≥ dst)

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

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

V: Set if an arithmetic overflow occurs, otherwise reset

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

Example R5 and R6 are compared. If they are equal, the program continues at the label EQUAL.

```
CMP   R5,R6   ; R5 = R6?
JEQ   EQUAL   ; YES, JUMP
```

Example Two RAM blocks are compared. If they are not equal, the program branches to the label ERROR.

```
     MOV   #NUM,R5      ; number of words to be compared
     MOV   #BLOCK1,R6   ; BLOCK1 start address in R6
     MOV   #BLOCK2,R7   ; BLOCK2 start address in R7
L$1  CMP   @R6+,0(R7)   ; Are Words equal? R6 increments
     JNZ   ERROR        ; No, branch to ERROR
     INCD  R7           ; Increment R7 pointer
     DEC   R5           ; Are all words compared?
     JNZ   L$1          ; No, another compare
```

Example The RAM bytes addressed by EDE and TONI are compared. If they are equal, the program continues at the label EQUAL.

```
CMP.B   EDE,TONI   ; MEM(EDE) = MEM(TONI)?
JEQ     EQUAL      ; YES, JUMP
```

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

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

Syntax

```
DADC dst  or  DADC.W src,dst
DADC.B dst
```

Operation dst + C -> dst (decimally)

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

Description The carry bit (C) is added decimally to the destination.

Status Bits

N: Set if MSB is 1

Z: Set if dst is 0, reset otherwise

C: Set if destination increments from 9999 to 0000, reset otherwise

Set if destination increments from 99 to 00, reset otherwise

V: Undefined

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

Example The four-digit decimal number contained in R5 is added to an eight-digit decimal number pointed to by R8.

```
CLRC                ; Reset carry
                    ; next instruction’s start condition is defined
DADD     R5,0(R8)   ; Add LSDs + C
DADC     2(R8)      ; Add carry to MSD
```

Example The two-digit decimal number contained in R5 is added to a four-digit decimal number pointed to by R8.

```
CLRC                ; Reset carry
                    ; next instruction’s start condition is defined
DADD.B   R5,0(R8)   ; Add LSDs + C
DADC.B   1(R8)      ; Add carry to MSDs
```

DADD[.W]	Source and carry added decimally to destination
DADD.B	Source and carry added decimally to destination

Syntax

```
DADD src,dst  or  DADD.W src,dst
DADD.B src,dst
```

Operation

src + dst + C -> dst (decimally)

Description

The source operand and the destination operand are treated as four binary coded decimals (BCD) with positive signs. The source operand and the carry bit (C)are added decimally to the destination operand. The source operand is not affected. The previous contents of the destination are lost. The result is not defined for non-BCD numbers.

Status Bits

N: Set if the MSB is 1, reset otherwise

Z: Set if result is zero, reset otherwise

C: Set if the result is greater than 9999

Set if the result is greater than 99

V: Undefined

Mode Bits

OSCOFF, CPUOFF, and GIE are not affected.

Example

The eight-digit BCD number contained in R5 and R6 is added decimally to an eight-digit BCD number contained in R3 and R4 (R6 and R4 contain the MSDs).

```
CLRC                ; clear carry
DADD    R5,R3       ; add LSDs
DADD    R6,R4       ; add MSDs with carry
JC      OVERFLOW    ; If carry occurs go to error handling routine
```

Example

The two-digit decimal counter in the RAM byte CNT is incremented by one.

```
CLRC              ; clear carry
DADD.B  #1,CNT
```

or

```
SETC
DADD.B  #0,CNT    ; equivalent to DADC.B CNT
```

***DEC[.W]** Decrement destination

***DEC.B** Decrement destination

Syntax

```
DEC dst  or  DEC.W dst
DEC.B dst
```

Operation dst - 1 -> dst

Emulation `SUB #1,dst`

Emulation `SUB.B #1,dst`

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

Status Bits

N: Set if result is negative, reset if positive

Z: Set if dst contained 1, reset otherwise

C: Reset if dst contained 0, set otherwise

V: Set if an arithmetic overflow occurs, otherwise reset.

Set if initial value of destination was 08000h, otherwise reset.

Set if initial value of destination was 080h, otherwise reset.

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

Example R10 is decremented by 1.

```
     DEC     R10    ; Decrement R10

; Move a block of 255 bytes from memory location starting with EDE to
memory location starting with
; TONI. Tables should not overlap: start of destination address TONI
must not be within the range EDE
; to EDE+0FEh

     MOV     #EDE,R6
     MOV     #255,R10
L$1  MOV.B   @R6+,TONI-EDE-1(R6)
     DEC     R10
     JNZ     L$1
```

Do not transfer tables using the routine above with the overlap shown in Figure 3-12.

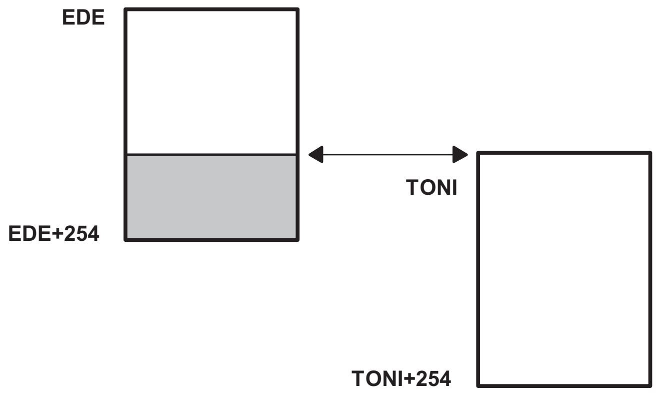


Figure 3-12. Decrement Overlap

***DECD[.W]** Double-decrement destination

***DECD.B** Double-decrement destination

Syntax

```
DECD dst  or  DECD.W dst
DECD.B dst
```

Operation dst - 2 -> dst

Emulation `SUB #2,dst`

Emulation `SUB.B #2,dst`

Description The destination operand is decremented by two. The original contents are lost.

Status Bits

N: Set if result is negative, reset if positive

Z: Set if dst contained 2, reset otherwise

C: Reset if dst contained 0 or 1, set otherwise

V: Set if an arithmetic overflow occurs, otherwise reset.

Set if initial value of destination was 08001 or 08000h, otherwise reset.

Set if initial value of destination was 081 or 080h, otherwise reset.

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

Example R10 is decremented by 2.

```
      DECD   R10      ; Decrement R10 by two
; Move a block of 255 words from memory location starting with EDE to
; memory location starting with TONI
; Tables should not overlap: start of destination address TONI must not be
; within the range EDE to EDE+0FEh

      MOV    #EDE,R6
      MOV    #510,R10
L$1   MOV    @R6+,TONI-EDE-2(R6)
      DECD   R10
      JNZ    L$1
```

Example Memory at location LEO is decremented by two.

```
DECD.B   LEO   ; Decrement MEM(LEO)
```

Decrement status byte STATUS by two.

```
DECD.B   STATUS
```

***DINT** Disable (general) interrupts

Syntax `DINT`

Operation 0 -> GIE

or

(0FFF7h .AND. SR -> SR / .NOT.src .AND. dst -> dst)

Emulation `BIC #8,SR`

Description All interrupts are disabled.

The constant 08h is inverted and logically ANDed with the status register (SR). The result is placed into the SR.

Status Bits Status bits are not affected.

Mode Bits GIE is reset. OSCOFF and CPUOFF are not affected.

Example The general interrupt enable (GIE) bit in the status register is cleared to allow a nondisrupted move of a 32-bit counter. This ensures that the counter is not modified during the move by any interrupt.

```
DINT                ; All interrupt events using the GIE bit are disabled
NOP
MOV   COUNTHI,R5    ; Copy counter
MOV   COUNTLO,R6
EINT                ; All interrupt events using the GIE bit are enabled
```

NOTE: Disable Interrupt

If any code sequence needs to be protected from interruption, the DINT should be executed at least one instruction before the beginning of the uninterruptible sequence, or should be followed by a NOP instruction.

***EINT** Enable (general) interrupts

Syntax `EINT`

Operation 1 -> GIE

or

(0008h .OR. SR -> SR / .src .OR. dst -> dst)

Emulation `BIS #8,SR`

Description All interrupts are enabled.

The constant #08h and the status register SR are logically ORed. The result is placed into the SR.

Status Bits Status bits are not affected.

Mode Bits GIE is set. OSCOFF and CPUOFF are not affected.

Example The general interrupt enable (GIE) bit in the status register is set.

```
; Interrupt routine of ports P1.2 to P1.7
; P1IN is the address of the register where all port bits are read. P1IFG is
; the address of the register where all interrupt events are latched.

         PUSH.B   &P1IN
         BIC.B    @SP,&P1IFG   ; Reset only accepted flags
         EINT                  ; Preset port 1 interrupt flags stored on stack
                               ; other interrupts are allowed
         BIT      #Mask,@SP
         JEQ      MaskOK       ; Flags are present identically to mask: jump
         ......
MaskOK   BIC      #Mask,@SP
         ......
         INCD     SP           ; Housekeeping: inverse to PUSH instruction
                               ; at the start of interrupt subroutine. Corrects
                               ; the stack pointer.
         RETI
```

NOTE: Enable Interrupt

The instruction following the enable interrupt instruction (EINT) is always executed, even if an interrupt service request is pending when the interrupts are enable.

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

Syntax

```
INC dst   or   INC.W dst
INC.B dst
```

Operation dst + 1 -> dst

Emulation `ADD #1,dst`

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

Status Bits

N: Set if result is negative, reset if positive

Z: Set if dst contained 0FFFFh, reset otherwise

Set if dst contained 0FFh, reset otherwise

C: Set if dst contained 0FFFFh, reset otherwise

Set if dst contained 0FFh, reset otherwise

V: Set if dst contained 07FFFh, reset otherwise

Set if dst contained 07Fh, reset otherwise

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

Example The status byte, STATUS, of a process is incremented. When it is equal to 11, a branch to OVFL is taken.

```
INC.B   STATUS
CMP.B   #11,STATUS
JEQ     OVFL
```

***INCD[.W]** Double-increment destination

***INCD.B** Double-increment destination

Syntax

```
INCD dst  or  INCD.W dst
INCD.B dst
```

Operation dst + 2 -> dst

Emulation `ADD #2,dst`

Emulation `ADD.B #2,dst`

Example The destination operand is incremented by two. The original contents are lost.

Status Bits

N: Set if result is negative, reset if positive

Z: Set if dst contained 0FFFEh, reset otherwise

Set if dst contained 0FEh, reset otherwise

C: Set if dst contained 0FFFEh or 0FFFFh, reset otherwise

Set if dst contained 0FEh or 0FFh, reset otherwise

V: Set if dst contained 07FFEh or 07FFFh, reset otherwise

Set if dst contained 07Eh or 07Fh, reset otherwise

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

Example The item on the top of the stack (TOS) is removed without using a register.

```
PUSH     R5     ; R5 is the result of a calculation, which is stored
                ; in the system stack
INCD     SP     ; Remove TOS by double-increment from stack
                ; Do not use INCD.B, SP is a word-aligned register
RET
```

Example The byte on the top of the stack is incremented by two.

```
INCD.B   0(SP)  ; Byte on TOS is increment by two
```

***INV[.W]** Invert destination

***INV.B** Invert destination

Syntax

```
INV dst
INV.B dst
```

Operation .NOT.dst -> dst

Emulation `XOR #0FFFFh,dst`

Emulation `XOR.B #0FFh,dst`

Description The destination operand is inverted. The original contents are lost.

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

Z: Set if dst contained 0FFFFh, reset otherwise

Set if dst contained 0FFh, reset otherwise

C: Set if result is not zero, reset otherwise (= .NOT. Zero)

Set if result is not zero, reset otherwise (= .NOT. Zero)

V: Set if initial destination operand was negative, otherwise reset

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

Example Content of R5 is negated (twos complement).

```
MOV    #00AEh,R5  ;                        R5 = 000AEh
INV    R5         ; Invert R5,             R5 = 0FF51h
INC    R5         ; R5 is now negated,     R5 = 0FF52h
```

Example Content of memory byte LEO is negated.

```
MOV.B  #0AEh,LEO  ;                        MEM(LEO) = 0AEh
INV.B  LEO        ; Invert LEO,            MEM(LEO) = 051h
INC.B  LEO        ; MEM(LEO) is negated,   MEM(LEO) = 052h
```

JC Jump if carry set

JHS Jump if higher or same

Syntax
```
JC label
JHS label
```

Operation If C = 1: PC + 2 offset -> PC

If C = 0: execute following instruction

Description The status register carry bit (C) is tested. If it is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If C is reset, the next instruction following the jump is executed. JC (jump if carry/higher or same) is used for the comparison of unsigned numbers (0 to 65536).

Status Bits Status bits are not affected.

Example The P1IN.1 signal is used to define or control the program flow.

```
BIT.B   #02h,&P1IN   ; State of signal -> Carry
JC      PROGA        ; If carry=1 then execute program routine A
......               ; Carry=0, execute program here
```

Example R5 is compared to 15. If the content is higher or the same, branch to LABEL.

```
CMP     #15,R5
JHS     LABEL        ; Jump is taken if R5 >= 15
......               ; Continue here if R5 < 15
```

JEQ, JZ Jump if equal, jump if zero

Syntax

```
JEQ label
JZ label
```

Operation If Z = 1: PC + 2 offset -> PC

If Z = 0: execute following instruction

Description The status register zero bit (Z) is tested. If it is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If Z is not set, the instruction following the jump is executed.

Status Bits Status bits are not affected.

Example Jump to address TONI if R7 contains zero.

```
TST   R7              ; Test R7
JZ    TONI            ; if zero: JUMP
```

Example Jump to address LEO if R6 is equal to the table contents.

```
CMP   R6,Table(R5)    ; Compare content of R6 with content of
                      ; MEM (table address + content of R5)
JEQ   LEO             ; Jump if both data are equal
......                ; No, data are not equal, continue here
```

Example Branch to LABEL if R5 is 0.

```
TST   R5
JZ    LABEL
......
```

JGE	Jump if greater or equal
Syntax	`JGE label`
Operation	If (N .XOR. V) = 0 then jump to label: PC + 2 P offset -> PC If (N .XOR. V) = 1 then execute the following instruction
Description	The status register negative bit (N) and overflow bit (V) are tested. If both N and V are set or reset, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If only one is set, the instruction following the jump is executed. This allows comparison of signed integers.
Status Bits	Status bits are not affected.
Example	When the content of R6 is greater or equal to the memory pointed to by R7, the program continues at label EDE.

```
CMP   @R7,R6   ; R6 >= (R7)?, compare on signed numbers
JGE   EDE      ; Yes, R6 >= (R7)
......         ; No, proceed
......
......
```

JL Jump if less

Syntax `JL label`

Operation If (N .XOR. V) = 1 then jump to label: PC + 2 offset -> PC

If (N .XOR. V) = 0 then execute following instruction

Description The status register negative bit (N) and overflow bit (V) are tested. If only one is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If both N and V are set or reset, the instruction following the jump is executed.

This allows comparison of signed integers.

Status Bits Status bits are not affected.

Example When the content of R6 is less than the memory pointed to by R7, the program continues at label EDE.

```
CMP   @R7,R6   ; R6 < (R7)?,  compare on signed numbers
JL    EDE      ; Yes, R6 < (R7)
......         ; No, proceed
......
......
```

JMP	Jump unconditionally
Syntax	`JMP label`
Operation	PC + 2 × offset -> PC
Description	The 10-bit signed offset contained in the instruction LSBs is added to the program counter.
Status Bits	Status bits are not affected.
Hint	This one-word instruction replaces the BRANCH instruction in the range of –511 to +512 words relative to the current program counter.

JN Jump if negative

Syntax `JN label`

Operation if N = 1: PC + 2 ×offset -> PC

if N = 0: execute following instruction

Description The negative bit (N) of the status register is tested. If it is set, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If N is reset, the next instruction following the jump is executed.

Status Bits Status bits are not affected.

Example The result of a computation in R5 is to be subtracted from COUNT. If the result is negative, COUNT is to be cleared and the program continues execution in another path.

```
      SUB   R5,COUNT   ; COUNT - R5 -> COUNT
      JN    L$1        ; If negative continue with COUNT=0 at PC=L$1
      ......           ; Continue with COUNT>=0
      ......
      ......
      ......
L$1   CLR   COUNT
      ......
      ......
      ......
```

JNC Jump if carry not set

JLO Jump if lower

Syntax

```
JNC label
JLO label
```

Operation

if C = 0: PC + 2 offset -> PC

if C = 1: execute following instruction

Description

The status register carry bit (C) is tested. If it is reset, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If C is set, the next instruction following the jump is executed. JNC (jump if no carry/lower) is used for the comparison of unsigned numbers (0 to 65536).

Status Bits

Status bits are not affected.

Example

The result in R6 is added in BUFFER. If an overflow occurs, an error handling routine at address ERROR is used.

```
        ADD     R6,BUFFER   ; BUFFER + R6 -> BUFFER
        JNC     CONT        ; No carry, jump to CONT
ERROR   ......              ; Error handler start
        ......
        ......
        ......
CONT    ......              ; Continue with normal program flow
        ......
        ......
```

Example

Branch to STL2 if byte STATUS contains 1 or 0.

```
        CMP.B   #2,STATUS
        JLO     STL 2       ; STATUS < 2
        ......              ; STATUS >= 2, continue here
```

JNE	Jump if not equal
JNZ	Jump if not zero

Syntax	`JNE label`
	`JNZ label`

Operation	If Z = 0: PC + 2 a offset -> PC If Z= 1: execute following instruction
Description	The status register zero bit (Z) is tested. If it is reset, the 10-bit signed offset contained in the instruction LSBs is added to the program counter. If Z is set, the next instruction following the jump is executed.
Status Bits	Status bits are not affected.
Example	Jump to address TONI if R7 and R8 have different contents.

```
CMP   R7,R8   ; COMPARE R7 WITH R8
JNE   TONI    ; if different: jump
......        ; if equal, continue
```

MOV[.W] Move source to destination

MOV.B Move source to destination

Syntax

```
MOV src,dst  or  MOV.W src,dst
MOV.B src,dst
```

Operation src -> dst

Description The source operand is moved to the destination.

The source operand is not affected. The previous contents of the destination are lost.

Status Bits Status bits are not affected.

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

Example The contents of table EDE (word data) are copied to table TOM. The length of the tables must be 020h locations.

```
       MOV   #EDE,R10                ; Prepare pointer
       MOV   #020h,R9                ; Prepare counter
Loop   MOV   @R10+,TOM-EDE-2(R10)    ; Use pointer in R10 for both tables
       DEC   R9                      ; Decrement counter
       JNZ   Loop                    ; Counter not 0, continue copying
       ......                        ; Copying completed
       ......
       ......
```

Example The contents of table EDE (byte data) are copied to table TOM. The length of the tables should be 020h locations

```
       MOV     #EDE,R10              ; Prepare pointer
       MOV     #020h,R9              ; Prepare counter
Loop   MOV.B   @R10+,TOM-EDE-1(R10)  ; Use pointer in R10 for
                                     ; both tables
       DEC   R9                      ; Decrement counter
       JNZ   Loop                    ; Counter not 0, continue
                                     ; copying
       ......                        ; Copying completed
       ......
       ......
```

***NOP** No operation

Syntax `NOP`

Operation None

Emulation `MOV #0, R3`

Description No operation is performed. The instruction may be used for the elimination of instructions during the software check or for defined waiting times.

Status Bits Status bits are not affected.

The NOP instruction is mainly used for two purposes:

- To fill one, two, or three memory words
- To adjust software timing

NOTE: Emulating No-Operation Instruction

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

```
MOV  #0,R3          ; 1 cycle, 1 word
MOV  0(R4),0(R4)    ; 6 cycles, 3 words
MOV  @R4,0(R4)      ; 5 cycles, 2 words
BIC  #0,EDE(R4)     ; 4 cycles, 2 words
JMP  $+2            ; 2 cycles, 1 word
BIC  #0,R5          ; 1 cycle, 1 word
```

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

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

***POP.B** Pop byte from stack to destination

Syntax

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

Operation

@SP -> temp

SP + 2 -> SP

temp -> dst

Emulation `MOV @SP+,dst or MOV.W @SP+,dst`

Emulation `MOV.B @SP+,dst`

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

Status Bits Status bits are not affected.

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

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

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

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

Example The contents of R7 is restored from the stack.

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

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

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

NOTE: The System Stack Pointer

The system stack pinter (SP) is always incremented by two, independent of the byte suffix.

PUSH[.W]	Push word onto stack
PUSH.B	Push byte onto stack

Syntax

```
PUSH src  or  PUSH.W src
PUSH.B src
```

Operation

SP - 2 -> SP

src -> @SP

Description The stack pointer is decremented by two, then the source operand is moved to the RAM word addressed by the stack pointer (TOS).

Status Bits Status bits are not affected.

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

Example The contents of the status register and R8 are saved on the stack.

```
PUSH     SR      ; save status register
PUSH     R8      ; save R8
```

Example The contents of the peripheral TCDAT is saved on the stack.

```
PUSH.B   &TCDAT  ; save data from 8-bit peripheral module,
                 ; address TCDAT, onto stack
```

NOTE: System Stack Pointer

The System stack pointer (SP) is always decremented by two, independent of the byte suffix.

***RET**	Return from subroutine
Syntax	`RET`
Operation	@SP -> PC SP + 2 -> SP
Emulation	`MOV @SP+,PC`
Description	The return address pushed onto the stack by a CALL instruction is moved to the program counter. The program continues at the code address following the subroutine call.
Status Bits	Status bits are not affected.

RETI	Return from interrupt
Syntax	`RETI`
Operation	TOS -> SR SP + 2 -> SP TOS -> PC SP + 2 -> SP
Description	The status register is restored to the value at the beginning of the interrupt service routine by replacing the present SR contents with the TOS contents. The stack pointer (SP) is incremented by two. The program counter is restored to the value at the beginning of interrupt service. This is the consecutive step after the interrupted program flow. Restoration is performed by replacing the present PC contents with the TOS memory contents. The stack pointer (SP) is incremented.
Status Bits	N: Restored from system stack Z: Restored from system stack C: Restored from system stack V: Restored from system stack
Mode Bits	OSCOFF, CPUOFF, and GIE are restored from system stack.
Example	Figure 3-13 illustrates the main program interrupt.

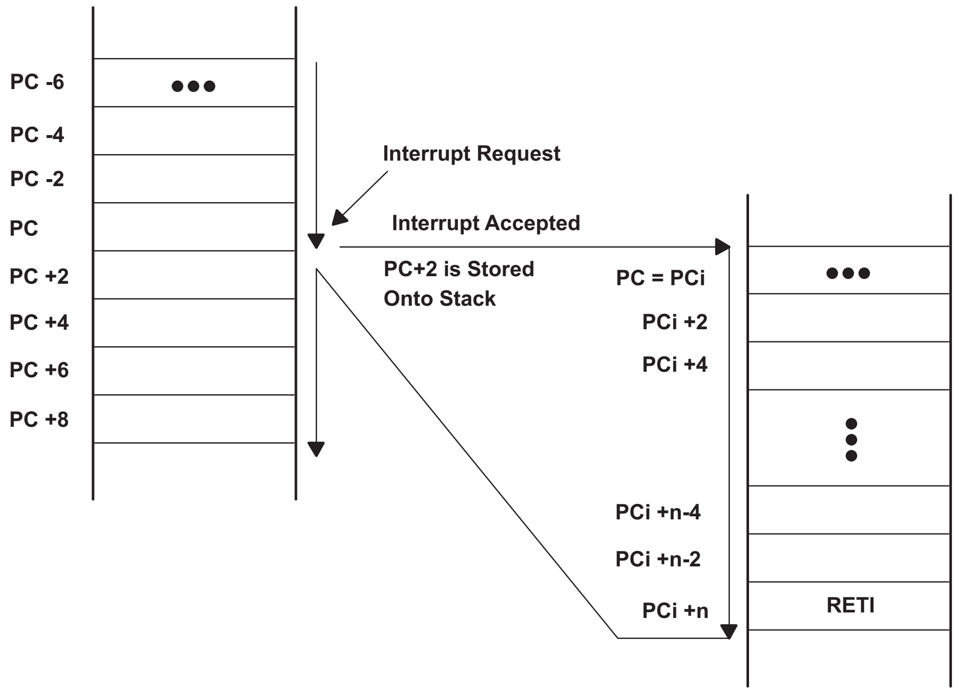


Figure 3-13. Main Program Interrupt

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

***RLA.B** Rotate left arithmetically

Syntax

```
RLA dst or RLA.W dst
RLA.B dst
```

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

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

Description The destination operand is shifted left one position as shown in Figure 3-14. The MSB is shifted into the carry bit (C) and the LSB is filled with 0. The RLA instruction acts as a signed multiplication by 2.

An overflow occurs if dst ≥ 04000h and dst < 0C000h before operation is performed: the result has changed sign.

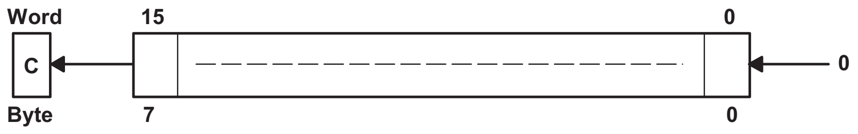


Figure 3-14. Destination Operand – Arithmetic Shift Left

An overflow occurs if dst ≥ 040h and dst < 0C0h before the operation is performed: the result has changed sign.

Status Bits

N: Set if result is negative, reset if positive

Z: Set if result is zero, reset otherwise

C: Loaded from the MSB

V: Set if an arithmetic overflow occurs:

the initial value is 04000h ≤ dst < 0C000h; reset otherwise

Set if an arithmetic overflow occurs:

the initial value is 040h ≤ dst < 0C0h; reset otherwise

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

Example R7 is multiplied by 2.

```
RLA     R7   ; Shift left R7  (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 instruction:

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

It must be substituted by:

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

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

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

Operation	C <- MSB <- MSB-1 .... LSB+1 <- LSB <- C
Emulation	`ADDC dst,dst`
Description	The destination operand is shifted left one position as shown in Figure 3-15. The carry bit (C) is shifted into the LSB and the MSB is shifted into the carry bit (C).

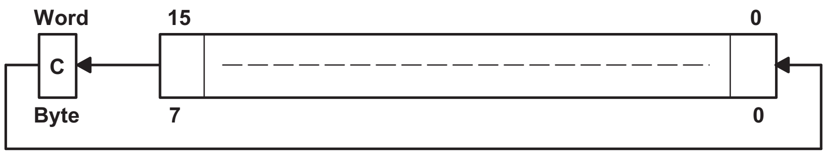


Figure 3-15. Destination Operand-Carry Left Shift

Status Bits

N: Set if result is negative, reset if positive

Z: Set if result is zero, reset otherwise

C: Loaded from the MSB

V: Set if an arithmetic overflow occurs

the initial value is 04000h ≤ dst < 0C000h; reset otherwise

Set if an arithmetic overflow occurs:

the initial value is 040h ≤ dst < 0C0h; reset otherwise

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

Example R5 is shifted left one position.

```
RLC     R5          ; (R5 x 2) + C -> R5
```

Example The input P1IN.1 information is shifted into the LSB of R5.

```
BIT.B   #2,&P1IN    ; Information -> Carry
RLC     R5          ; Carry=P0in.1 -> LSB of R5
```

Example The MEM(LEO) content is shifted left one position.

```
RLC.B   LEO         ; Mem(LEO) x 2 + C -> Mem(LEO)
```

NOTE: RLC and RLC.B Substitution

The assembler does not recognize the instruction:

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

It must be substitued by:

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

RRA[.W] Rotate right arithmetically

RRA.B Rotate right arithmetically

Syntax

```
RRA dst  or  RRA.W dst
RRA.B dst
```

Operation MSB -> MSB, MSB -> MSB-1, ... LSB+1 -> LSB, LSB -> C

Description The destination operand is shifted right one position as shown in Figure 3-16. The MSB is shifted into the MSB, the MSB is shifted into the MSB-1, and the LSB+1 is shifted into the LSB.

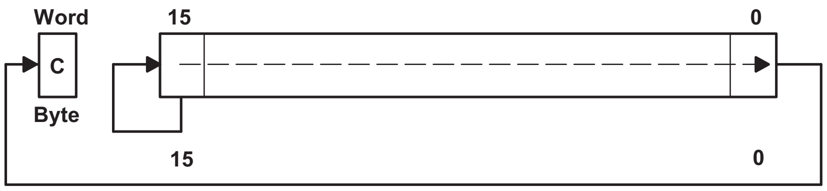


Figure 3-16. Destination Operand – Arithmetic Right Shift

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

Z: Set if result is zero, reset otherwise

C: Loaded from the LSB

V: Reset

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

Example R5 is shifted right one position. The MSB retains the old value. It operates equal to an arithmetic division by 2.

```
RRA   R5   ; R5/2 -> R5
;   The value in R5 is multiplied by 0.75 (0.5 + 0.25).
;
PUSH    R5        ; Hold R5 temporarily using stack
RRA     R5        ; R5 x 0.5  ->  R5
ADD     @SP+,R5   ; R5 x 0.5 + R5 = 1.5 x R5  -> R5
RRA     R5        ; (1.5 x R5) x 0.5 = 0.75 x R5  -> R5
......
```

Example The low byte of R5 is shifted right one position. The MSB retains the old value. It operates equal to an arithmetic division by 2.

```
RRA.B    R5       ; R5/2 -> R5: operation is on low byte only
                  ; High byte of R5 is reset
PUSH.B   R5       ; R5 x 0.5  ->  TOS
RRA.B   @SP       ; TOS x 0.5 = 0.5 x R5 x 0.5 = 0.25 x R5  -> TOS
ADD.B   @SP+,R5   ; R5 x 0.5 + R5 x 0.25 = 0.75 x R5  -> R5
......
```

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

Syntax

```
RRC dst  or  RRC.W dst
RRC dst
```

Operation C -> MSB -> MSB-1 .... LSB+1 -> LSB -> C

Description The destination operand is shifted right one position as shown in Figure 3-17. The carry bit (C) is shifted into the MSB, the LSB is shifted into the carry bit (C).

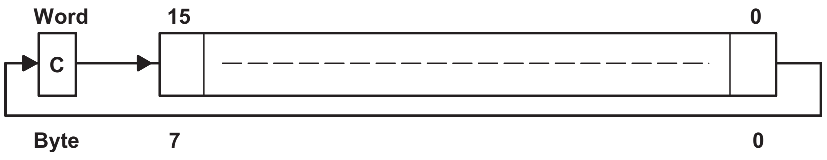


Figure 3-17. Destination Operand—Carry Right Shift

Status Bits

N: Set if result is negative, reset if positive

Z: Set if result is zero, reset otherwise

C: Loaded from the LSB

V: Reset

Mode Bits OSCOFF, CPUOFF, and GIEare not affected.

Example R5 is shifted right one position. The MSB is loaded with 1.

```
SETC          ; Prepare carry for MSB
RRC     R5    ; R5/2 + 8000h -> R5
```

Example R5 is shifted right one position. The MSB is loaded with 1.

```
SETC          ; Prepare carry for MSB
RRC.B   R5    ; R5/2 + 80h -> R5; low byte of R5 is used
```

***SBC[.W]** Subtract source and borrow/.NOT. carry from destination

***SBC.B** Subtract source and borrow/.NOT. carry from destination

Syntax

```
SBC dst  or  SBC.W dst
SBC.B dst
```

Operation

dst + 0FFFFh + C -> dst

dst + 0FFh + C -> dst

Emulation

```
SUBC #0,dst SUBC.B #0,dst
```

Description The carry bit (C) is added to the destination operand minus one. The previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive

Z: Set if result is zero, reset otherwise

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

Set to 1 if no borrow, reset if borrow.

V: Set if an arithmetic overflow occurs, reset otherwise.

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

Example The 16-bit counter pointed to by R13 is subtracted from a 32-bit counter pointed to by R12.

```
SUB     @R13,0(R12)    ; Subtract LSDs
SBC     2(R12)         ; Subtract carry from MSD
```

Example The 8-bit counter pointed to by R13 is subtracted from a 16-bit counter pointed to by R12.

```
SUB.B   @R13,0(R12)    ; Subtract LSDs
SBC.B   1(R12)         ; Subtract carry from MSD
```

NOTE: Borrow Implementation

The borrow is treated as a .NOT. carry:	Borrow	Carry bit
	Yes	0
	No	1

***SETC**	Set carry bit
Syntax	`SETC`
Operation	1 -> C
Emulation	`BIS #1,SR`
Description	The carry bit (C) is set.
Status Bits	N: Not affected Z: Not affected C: Set V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	Emulation of the decimal subtraction: Subtract R5 from R6 decimally Assume that R5 = 03987h and R6 = 04137h

```
DSUB   ADD    #06666h,R5   ; Move content R5 from 0-9 to 6-0Fh
                           ; R5 = 03987h + 06666h = 09FEDh
       INV    R5           ; Invert this (result back to 0-9)
                           ; R5 = .NOT. R5 = 06012h
       SETC                ; Prepare carry = 1
       DADD   R5,R6        ; Emulate subtraction by addition of:
                           ; (010000h - R5 - 1)
                           ; R6 = R6 + R5 + 1
                           ; R6 = 0150h
```

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

***SETZ**	Set zero bit
Syntax	`SETZ`
Operation	1 -> Z
Emulation	`BIS #2,SR`
Description	The zero bit (Z) is set.
Status Bits	N: Not affected Z: Set C: Not affected V: Not affected
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.

SUB[.W]	Subtract source from destination
SUB.B	Subtract source from destination

Syntax

```
SUB src,dst  or  SUB.W src,dst
SUB.B src,dst
```

Operation

dst + .NOT.src + 1 -> dst

or

[(dst - src -> dst)]

Description

The source operand is subtracted from the destination operand by adding the source operand's 1s complement and the constant 1. The source operand is not affected. The previous contents of the destination are lost.

Status Bits

N: Set if result is negative, reset if positive

Z: Set if result is zero, reset otherwise

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

Set to 1 if no borrow, reset if borrow.

V: Set if an arithmetic overflow occurs, otherwise reset

Mode Bits

OSCOFF, CPUOFF, and GIE are not affected.

Example

See example at the SBC instruction.

Example

See example at the SBC.B instruction.

NOTE: Borrow Is Treated as a .NOT.

The borrow is treated as a .NOT. carry:

Borrow	Carry bit
Yes	0
No	1

SUBC[.W], SBB[.W] Subtract source and borrow/.NOT. carry from destination

SUBC.B, SBB.B Subtract source and borrow/.NOT. carry from destination

Syntax

```
SUBC     src,dst    or    SUBC.W   src,dst    or
SBB      src,dst    or    SBB.W    src,dst
SUBC.B   src,dst    or    SBB.B    src,dst
```

Operation dst + .NOT.src + C -> dst

or

(dst - src - 1 + C -> dst)

Description The source operand is subtracted from the destination operand by adding the source operand's 1s complement and the carry bit (C). The source operand is not affected. The previous contents of the destination are lost.

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

Z: Set if result is zero, reset otherwise.

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

Set to 1 if no borrow, reset if borrow.

V: Set if an arithmetic overflow occurs, reset otherwise.

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

Example Two floating point mantissas (24 bits) are subtracted.

LSBs are in R13 and R10, MSBs are in R12 and R9.

```
SUB.W    R13,R10     ; 16-bit part, LSBs
SUBC.B   R12,R9      ; 8-bit part, MSBs
```

Example The 16-bit counter pointed to by R13 is subtracted from a 16-bit counter in R10 and R11(MSD).

```
SUB.B    @R13+,R10   ; Subtract LSDs without carry
SUBC.B   @R13,R11    ; Subtract MSDs with carry
...                  ; resulting from the LSDs
```

NOTE: Borrow Implementation

The borrow is treated as a .NOT. carry:

Borrow	Carry bit
Yes	0
No	1

SWPB Swap bytes

Syntax `SWPB dst`

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

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

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

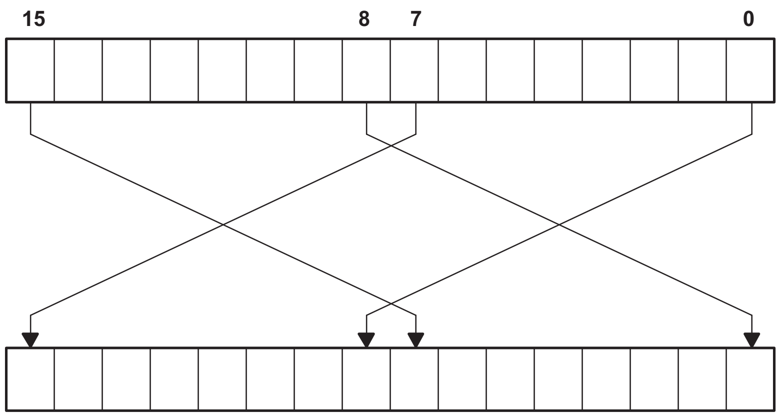


Figure 3-18. Destination Operand Byte Swap

Example

```
MOV     #040BFh,R7    ; 0100000010111111 -> R7
SWPB    R7            ; 1011111101000000 in R7
```

Example The value in R5 is multiplied by 256. The result is stored in R5,R4.

```
SWPB    R5            ;
MOV     R5,R4         ; Copy the swapped value to R4
BIC     #0FF00h,R5    ; Correct the result
BIC     #00FFh,R4     ; Correct the result
```

SXT Extend Sign

Syntax `SXT dst`

Operation Bit 7 -> Bit 8 .... Bit 15

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

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

Z: Set if result is zero, reset otherwise

C: Set if result is not zero, reset otherwise (.NOT. Zero)

V: Reset

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

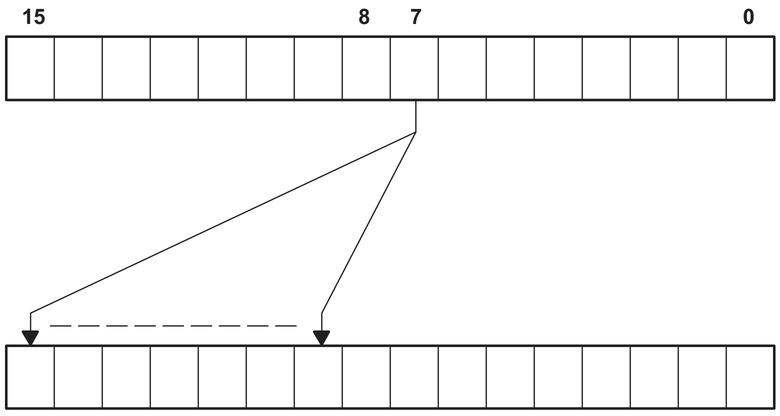


Figure 3-19. Destination Operand Sign Extension

Example R7 is loaded with the P1IN value. The operation of the sign-extend instruction expands bit 8 to bit 15 with the value of bit 7.

R7 is then added to R6.

```
MOV.B   &P1IN,R7    ; P1IN = 080h:    .... .... 1000 0000
SXT     R7          ; R7 = 0FF80h:    1111 1111 1000 0000
```

***TST[.W]** Test destination

***TST.B** Test destination

Syntax

```
TST dst  or  TST.W dst
TST.B dst
```

Operation

dst + 0FFFFh + 1

dst + 0FFh + 1

Emulation

```
CMP #0,dst CMP.B #0,dst
```

Description The destination operand is compared with zero. The status bits are set according to the result. The destination is not affected.

Status Bits

N: Set if destination is negative, reset if positive

Z: Set if destination contains zero, reset otherwise

C: Set

V: Reset

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

Example R7 is tested. If it is negative, continue at R7NEG; if it is positive but not zero, continue at R7POS.

```
         TST     R7       ; Test R7
         JN      R7NEG    ; R7 is negative
         JZ      R7ZERO   ; R7 is zero
R7POS    ......           ; R7 is positive but not zero
R7NEG    ......           ; R7 is negative
R7ZERO   ......           ; R7 is zero
```

Example The low byte of R7 is tested. If it is negative, continue at R7NEG; if it is positive but not zero, continue at R7POS.

```
         TST.B   R7       ; Test low byte of R7
         JN      R7NEG    ; Low byte of R7 is negative
         JZ      R7ZERO   ; Low byte of R7 is zero
R7POS    ......           ; Low byte of R7 is positive but not zero
R7NEG    .....            ; Low byte of R7 is negative
R7ZERO   ......           ; Low byte of R7 is zero
```

XOR[.W] Exclusive OR of source with destination

XOR.B Exclusive OR of source with destination

Syntax

```
XOR src,dst  or  XOR.W src,dst
XOR.B src,dst
```

Operation src .XOR. dst -> dst

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

Status Bits

N: Set if result MSB is set, reset if not set

Z: Set if result is zero, reset otherwise

C: Set if result is not zero, reset otherwise (= .NOT. Zero)

V: Set if both operands are negative

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

Example The bits set in R6 toggle the bits in the RAM word TONI.

```
XOR     R6,TONI  ; Toggle bits of word TONI on the bits set in R6
```

Example The bits set in R6 toggle the bits in the RAM byte TONI.

```
XOR.B   R6,TONI  ; Toggle bits of byte TONI on the bits set in
                 ; low byte of R6
```

Example Reset to 0 those bits in low byte of R7 that are different from bits in RAM byte EDE.

```
XOR.B   EDE,R7   ; Set different bit to "1s"
INV.B   R7       ; Invert Lowbyte, Highbyte is 0h
```

3.4.5 Instruction Cycles and Lengths

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

3.4.5.1 Interrupt and Reset Cycles

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

Table 3-14. Interrupt and Reset Cycles

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

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

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

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

	No. of Cycles				
Addressing Mode	RRA, RRC SWPB, SXT	PUSH	CALL	Length of Instruction	Example
Rn	1	3	4	1	`SWPB R5`
@Rn	3	4	4	1	`RRC @R9`
@Rn+	3	5	5	1	`SWPB @R10+`
#N	(See note)	4	5	2	`CALL #0F000h`
X(Rn)	4	5	5	2	`CALL 2(R7)`
EDE	4	5	5	2	`PUSH EDE`
&EDE	4	5	5	2	`SXT &EDE`

NOTE: Instruction Format II Immediate Mode

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

3.4.5.3 Format-III (Jump) Instruction Cycles and Lengths

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

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

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

Table 3-16. Format 1 Instruction Cycles and Lengths

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

3.4.6 Instruction Set Description

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

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

Figure 3-20. Core Instruction Map

Table 3-17. MSP430 Instruction Set

Mnemonic		Description		V	N	Z	C
`ADC(.B)` [(1)]	`dst`	Add C to destination	dst + C → dst	*	*	*	*
`ADD(.B)`	`src,dst`	Add source to destination	src + dst → dst	*	*	*	*
`ADDC(.B)`	`src,dst`	Add source and C to destination	src + dst + C → dst	*	*	*	*
`AND(.B)`	`src,dst`	AND source and destination	src .and. dst → dst	0	*	*	*
`BIC(.B)`	`src,dst`	Clear bits in destination	not.src .and. dst → dst	-	-	-	-
`BIS(.B)`	`src,dst`	Set bits in destination	src .or. dst → dst	-	-	-	-
`BIT(.B)`	`src,dst`	Test bits in destination	src .and. dst	0	*	*	*
`BR` [(1)]	`dst`	Branch to destination	dst → PC	-	-	-	-
`CALL`	`dst`	Call destination	PC+2 → stack, dst → PC	-	-	-	-
`CLR(.B)` [(1)]	`dst`	Clear destination	0 → dst	-	-	-	-
`CLRC` [(1)]		Clear C	0 → C	-	-	-	0
`CLRN` [(1)]		Clear N	0 → N	-	0	-	-
`CLRZ` [(1)]		Clear Z	0 → Z	-	-	0	-
`CMP(.B)`	`src,dst`	Compare source and destination	dst - src	*	*	*	*
`DADC(.B)` [(1)]	`dst`	Add C decimally to destination	dst + C → dst (decimally)	*	*	*	*
`DADD(.B)`	`src,dst`	Add source and C decimally to dst	src + dst + C → dst (decimally)	*	*	*	*
`DEC(.B)` [(1)]	`dst`	Decrement destination	dst - 1 → dst	*	*	*	*

(1) Emulated Instruction

Table 3-17. MSP430 Instruction Set (continued)

Mnemonic		Description		V	N	Z	C
`DECD(.B)` [1]	`dst`	Double-decrement destination	dst - 2 → dst	*	*	*	*
`DINT` [1]		Disable interrupts	0 → GIE	-	-	-	-
`EINT` [1]		Enable interrupts	1 → GIE	-	-	-	-
`INC(.B)` [1]	`dst`	Increment destination	dst +1 → dst	*	*	*	*
`INCD(.B)` [1]	`dst`	Double-increment destination	dst+2 → dst	*	*	*	*
`INV(.B)` [1]	`dst`	Invert destination	not.dst → dst	*	*	*	*
`JC/JHS`	`label`	Jump if C set/Jump if higher or same		-	-	-	-
`JEQ/JZ`	`label`	Jump if equal/Jump if Z set		-	-	-	-
`JGE`	`label`	Jump if greater or equal		-	-	-	-
`JL`	`label`	Jump if less		-	-	-	-
`JMP`	`label`	Jump	PC + 2 × offset → PC	-	-	-	-
`JN`	`label`	Jump if N set		-	-	-	-
`JNC/JLO`	`label`	Jump if C not set/Jump if lower		-	-	-	-
`JNE/JNZ`	`label`	Jump if not equal/Jump if Z not set		-	-	-	-
`MOV(.B)`	`src,dst`	Move source to destination	src → dst	-	-	-	-
`NOP` [2]		No operation		-	-	-	-
`POP(.B)` [2]	`dst`	Pop item from stack to destination	@SP → dst, SP+2 → SP	-	-	-	-
`PUSH(.B)`	`src`	Push source onto stack	SP - 2 → SP, src → @SP	-	-	-	-
`RET` [2]		Return from subroutine	@SP → PC, SP + 2 → SP	-	-	-	-
`RETI`		Return from interrupt		*	*	*	*
`RLA(.B)` [2]	`dst`	Rotate left arithmetically		*	*	*	*
`RLC(.B)` [2]	`dst`	Rotate left through C		*	*	*	*
`RRA(.B)`	`dst`	Rotate right arithmetically		0	*	*	*
`RRC(.B)`	`dst`	Rotate right through C		*	*	*	*
`SBC(.B)` [2]	`dst`	Subtract not(C) from destination	dst + 0FFFFh + C → dst	*	*	*	*
`SETC` [2]		Set C	1 → C	-	-	-	1
`SETN` [2]		Set N	1 → N	-	1	-	-
`SETZ` [2]		Set Z	1 → C	-	-	1	-
`SUB(.B)`	`src,dst`	Subtract source from destination	dst + .not.src + 1 → dst	*	*	*	*
`SUBC(.B)`	`src,dst`	Subtract source and not(C) from dst	dst + .not.src + C → dst	*	*	*	*
`SWPB`	`dst`	Swap bytes		-	-	-	-
`SXT`	`dst`	Extend sign		0	*	*	*
`TST(.B)` [2]	`dst`	Test destination	dst + 0FFFFh + 1	0	*	*	1
`XOR(.B)`	`src,dst`	Exclusive OR source and destination	src .xor. dst → dst	*	*	*	*

[2] Emulated Instruction

Versatile I/O Port

This chapter describes the operation of the versatile I/O ports. The versatile port combo P1/P2 is implemented in all MSP430x09x devices.

Topic **Page**

4.1 Versatile I/O Ports (VersaPorts) and Digital I/O Ports

MSP430 devices have digital I/O ports or versatile I/O ports implemented. See the device-specific data sheet to determine which port type is available for each device. VersaPorts are used where higher flexibility of the possible configuration is required. This is typically the case in lower pin count devices with a high number of internal modules. Versatile I/O ports and digital I/O ports with different port identifiers can coexist; they use the same address space reserved for ports in general.

4.2 Versatile I/O Port Introduction

Most ports have eight I/O pins; however, some ports may support fewer. See the device-specific data sheet for ports available. Every I/O pin is individually configurable for input or output direction, and each I/O line can be individually read or written to. All ports have individually configurable pullup or pulldown resistors.

All versatile I/O ports have interrupt capability. Each interrupt for a port Px I/O line can be individually enabled and configured to provide an interrupt on a rising edge or falling edge of an input signal. All I/O lines from each versatile port source a single interrupt vector.

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, etc. are associated with the names PA, PB, etc., respectively. When writing to port PA with word operations, all 16 bits are written to the port. Writing to the lower byte of the PA port using byte operations leaves the upper byte unchanged. Similarly, writing to the upper byte of the PA port using byte instructions leaves the lower byte unchanged. Ports PB, PC, etc., behave similarly. The unused bits of ports that are not fully equipped are a do-not-care when writing to them.

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, etc., behave similarly. Unused bits are read as zeros when reading from a port that is not fully equipped.

The I/O features include:

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

Figure 4-1 shows a typical basic logic for a versatile I/O. The direction of a port is determined either by PxDIR or by the module itself. All module inputs use the same Module x IN.

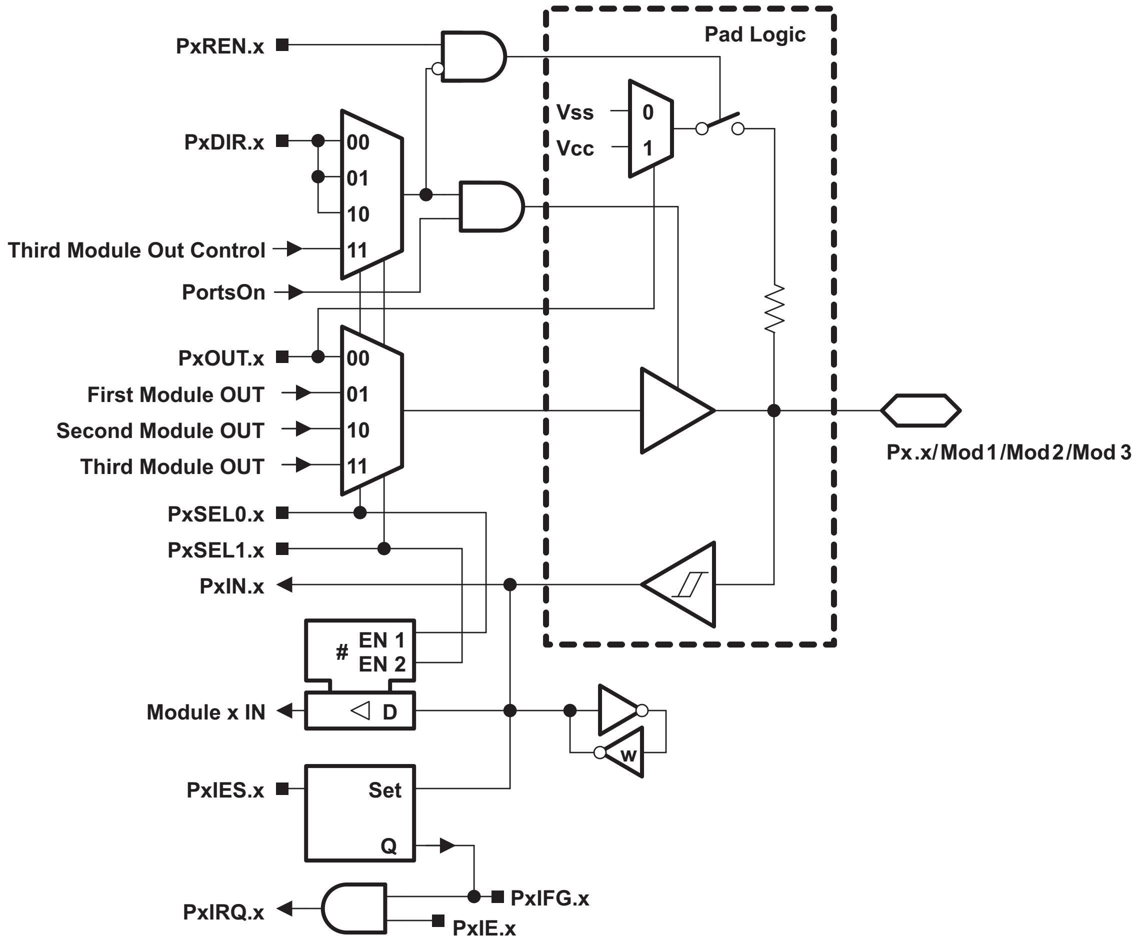


Figure 4-1. Typical Schematic of the Port Logic

4.3 Versatile I/O Port Operation

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

4.3.1 *Input Register PxIN*

Each bit in each PxIN register reflects the value of the input signal at the corresponding I/O pin when the pin is configured as I/O function. These registers are read only.

Bit = 0: The input is low

Bit = 1: The input is high

4.3.2 *Output Registers PxOUT*

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

Bit = 0: The output is low

Bit = 1: The output is high

If the pin's pullup/down resistor is enabled, the corresponding bit in the PxOUT register selects pullup or pulldown.

Bit = 0: The pin is pulled down

Bit = 1: The pin is pulled up

4.3.3 Direction Registers PxDIR

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

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

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

4.3.4 Pullup/Pulldown Resistor Enable Registers PxREN

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

Bit = 0: Pullup/down resistor disabled

Bit = 1: Pullup/down resistor enabled

Table 4-1 summarizes the usage of PxDIRx, PxRENx, and PxOUTx for proper I/O configuration.

Table 4-1. I/O Configuration

PxDIRx	PxRENx	PxOUTx	I/O Configuration
0	0	x	Input
0	1	0	Input with pulldown resistor
0	1	1	Input with pullup resistor
1	x	x	Output

4.3.5 Function Select Registers PxSELxx

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 4-2 shows how to select the various module functions.

Table 4-2. I/O Function Selection

PxSEL0x	PxSEL1x	I/O Configuration
0	0	I/O port function is selected for the pin
0	1	Primary module function is selected
1	0	Secondary module function is selected
1	1	Tertiary module function is selected

Setting the PxSELxx bits to a module function does not automatically set the pin direction. Other peripheral module functions may require the PxDIRx bits to be configured according to the direction needed for the module function. See the pin schematics in the device-specific data sheet.

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 PxSELxx is other than 00, the internal input signal follows the signal at the pin for all connected modules. However, if PxSELxx = 00, the input to the peripherals maintain the value of the input signal at the device pin before the PxSELxx bits were reset.

4.3.6 Versatile I/O Port Interrupts

Each pin of the versatile port has interrupt capability, configured with the PxIFG, PxIE, and PxIES registers. All port interrupt flags are prioritized, with bit 0 being the highest, and combined to source a single interrupt vector. The highest priority enabled interrupt generates a number in the PxIV register. This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled Px interrupts do not affect the PxIV value.

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

Bit = 0: No interrupt is pending

Bit = 1: An interrupt is pending

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

NOTE: PxIFG Flags When Changing PxOUT, PxDIR, or PxREN

Writing to PxOUT, PxDIR or PxREN can result in setting the corresponding PxIFG flags

Any access, read or write, of the PxIV 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 PxIFG.0 has the highest priority. If the PxIFG.0 and PxIFG.2 flags are set when the interrupt service routine accesses the PxIV register, PxIFG.0 is reset automatically. After the RETI instruction of the interrupt service routine is executed, the PxIFG.2 will generate another interrupt.

4.3.6.1 P1IV 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. For the other ports this is similar.

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

```
;Interrupt handler for P1IFGx                          Cycles
P1_HND:   ...            ; Interrupt latency           6
          ADD &P1IV,PC   ; Add offset to Jump table    3
          RETI           ; Vector 0: No interrupt      5
          JMP P1_0_HND   ; Vector 2: Port 1 bit 0      2
          JMP P1_1_HND   ; Vector 4: Port 1 bit 1      2
          JMP P1_2_HND   ; Vector 6: Port 1 bit 2      2
          JMP P1_3_HND   ; Vector 8: Port 1 bit 3      2
          JMP P1_4_HND   ; Vector 10: Port 1 bit 4     2
          JMP P1_5_HND   ; Vector 12: Port 1 bit 5     2
          JMP P1_6_HND   ; Vector 14: Port 1 bit 6     2
          JMP P1_7_HND   ; Vector 16: Port 1 bit 7     2
P1_7_HND                 ; Vector 16: Port 1 bit 7
          ...            ; Task starts here
          RETI           ; Back to main program        5
P1_6_HND                 ; Vector 14: Port 1 bit 6
          ...            ; Task starts here
          RETI           ; Back to main program        5
P1_5_HND                 ; Vector 12: Port 1 bit 5
          ...            ; Task starts here
          RETI           ; Back to main program        5
P1_4_HND                 ; Vector 10: Port 1 bit       4
          ...            ; Task starts here
          RETI           ; Back to main program        5
P1_3_HND                 ; Vector 8: Port 1 bit 3
          ...            ; Task starts here
          RETI           ; Back to main program        5
P1_2_HND                 ; Vector 6: Port 1 bit 2
          ...            ; Task starts here
          RETI           ; Back to main program        5
```

```
P1_1_HND                 ; Vector 4: Port 1 bit 1
          ...            ; Task starts here
          RETI           ; Back to main program       5
P1_0_HND                 ; Vector 2: Port 1 bit 0
          ...            ; Task starts here
          RETI           ; Back to main program       5
```

4.3.6.2 Interrupt Edge Select Registers PxIES

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

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

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

NOTE: Writing to PxIESx

Writing to PxIES can result in setting the corresponding interrupt flags (see Table 4-3).

Table 4-3. Writing to PxIESx

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

4.3.6.3 Interrupt Enable PxIE

Each PxIE bit enables the associated PxIFG interrupt flag.

Bit = 0: The interrupt is disabled

Bit = 1: The interrupt is enabled

4.3.7 Configuring Unused Port Pins

Unused I/O pins should be configured as I/O function, output direction, and left unconnected on the PC board, to prevent a floating input and reduce power consumption. The value of the PxOUT bit is don't care, since the pin is unconnected. Alternatively, the integrated pullup/pulldown resistor can be enabled by setting the PxREN bit of the unused pin to prevent the floating input. See Chapter 1 for termination of unused pins.

4.4 Versatile I/O Port Registers

The digital I/O registers are listed in Table 4-4, along with their base addresses. The address offset is given in Table 4-5.

Table 4-4. Versatile I/O Ports Base Address

Module	Base Address
VersaPort combo P1/P2	0200h
VersaPort combo P3/P4	0220h
VersaPort combo P5/P6	0240h
VersaPort combo P7/P8	0280h
VersaPort combo P9/P10	02A0h
VersaPort combo P11/P12	02C0h

Table 4-5. Versatile I/O Port Control Registers

Port	Register	Short Form	Register Type	Address Offset	Initial State
P1[1]	P1 Interrupt Vector	P1IV	read/write	0Eh	0000h
P2[1]	P2 Interrupt Vector	P2IV	read/write	1Eh	0000h
P1[1]	Input	P1IN	read only	00h	-
	Output	P1OUT	read/write	02h	Unchanged[2]
	Direction	P1DIR	read/write	04h	00h[2]
	Resistor Enable	P1REN	read/write	06h	00h[2]
	Port Select 0	P1SEL0	read/write	0Ah	00h[2]
	Port Select 1	P1SEL1	read/write	0Ch	00h[2]
	Interrupt Edge Select	P1IES	read/write	18h	Unchanged
	Interrupt Enable	P1IE	read/write	1Ah	00h
	Interrupt Flag	P1IFG	read/write	1Ch	00h
P2[1]	Input	P2IN	read only	01h	-
	Output	P2OUT	read/write	03h	Unchanged[2]
	Direction	P2DIR	read/write	05h	00h[2]
	Resistor Enable	P2REN	read/write	07h	00h[2]
	Port Select 0	P2SEL0	read/write	0Bh	00h[2]
	Port Select 1	P2SEL1	read/write	0Dh	00h[2]
	Interrupt Edge Select	P2IES	read/write	19h	Unchanged
	Interrupt Enable	P2IE	read/write	1Bh	00h
	Interrupt Flag	P2IFG	read/write	1Dh	00h

(1) Similar for port combos P3/P4, P5/P6, etc.
(2) Unless otherwise noted in device specific data sheet

P1IV, Port 1 Interrupt Vector Register

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

7	6	5	4	3	2	1	0
0	**0**	**0**	**P1IVx**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

P1IVx	Bits 4-1	Port 1 interrupt vector value. Writing to this register clears all pending interrupt flags

P1IV	Interrupt Source	Interrupt Flag	Priority
00h	No interrupt pending	-	
02h	Port 1.0 interrupt	P1IFG.0	Highest
04h	Port 1.1 interrupt	P1IFG.1	
06h	Port 1.2 interrupt	P1IFG.2	
08h	Port 1.3 interrupt	P1IFG.3	
0Ah	Port 1.4 interrupt	P1IFG.4	
0Ch	Port 1.5 interrupt	P1IFG.5	
0Eh	Port 1.6 interrupt	P1IFG.6	
10h	Port 1.7 interrupt	P1IFG.7	Lowest

P2IV, Port 2 Interrupt Vector Register

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

7	6	5	4	3	2	1	0
0	**0**	**0**	**P2IVx**				**0**
r0	r0	r0	r-0	r-0	r-0	r-0	r0

P2IVx	Bits 4-1	Port 2 interrupt vector value. Writing to this register clears all pending interrupt flags

P1IV	Interrupt Source	Interrupt Flag	Priority
00h	No interrupt pending	-	
02h	Port 1.0 interrupt	P2IFG.0	Highest
04h	Port 1.1 interrupt	P2IFG.1	
06h	Port 1.2 interrupt	P2IFG.2	
08h	Port 1.3 interrupt	P2IFG.3	
0Ah	Port 1.4 interrupt	P2IFG.4	
0Ch	Port 1.5 interrupt	P2IFG.5	
0Eh	Port 1.6 interrupt	P2IFG.6	
10h	Port 1.7 interrupt	P2IFG.7	Lowest

PxIN, Port x Input Register

7	6	5	4	3	2	1	0
PxIN							
r	r	r	r	r	r	r	r

PxIN	Bits 7-0	Port x input register. Read only.

PxOUT, Port x Output Register

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

PxOUT Bits 7-0 Port x output register

When I/O configured to output mode:

0 The output is low

1 The output is high

When I/O configured to input mode and pullups/pulldowns enabled:

0 Pulldown selected

1 Pullup selected

PxDIR, Port x Direction Register

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

PxDIR Bits 7-0 Port x direction register

0 Port configured as input

1 Port configured as output

PxREN, Port x Resistor Enable Register

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

PxREN Bits 7-0 Port x Pullup/pulldown resistor enable register

0 Pullup/pulldown disabled

1 Pullup/pulldown enabled

PxSEL0, PxSEL1, Port x Function Select Registers

7	6	5	4	3	2	1	0
PxSEL0/1							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

PxSEL0/1 Bits 7-0 Port x function select register. These two eight bit register determine the function of each port pin. The chosen functions for a particular pin are device dependent. Check the port schematics of the device data sheet.

PxSEL0.x	PxSEL1.x	I/O Configuration
0	0	I/O Port function is selected for the pin
0	1	primary module function is selected
1	0	secondary module function is selected
1	1	ternary module function is selected

PxIES, Port x Interrupt Edge Select Register

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 register

0 PxIFGx flag is set with a low-to-high transition

1 PxIFGx flag is set with a high-to-low transition

PxIE, Port x Interrupt Enable Register

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

PxIE	Bits 7-0	Port x interrupt enable register	
		0	Corresponding port interrupt disabled
		1	Corresponding port interrupt enabled

PxIFG, Port x Interrupt Flag Register

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 register	
		0	No interrupt is pending
		1	Interrupt is pending

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**

5.1 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 5-1.

NOTE: Watchdog timer powers up active.

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

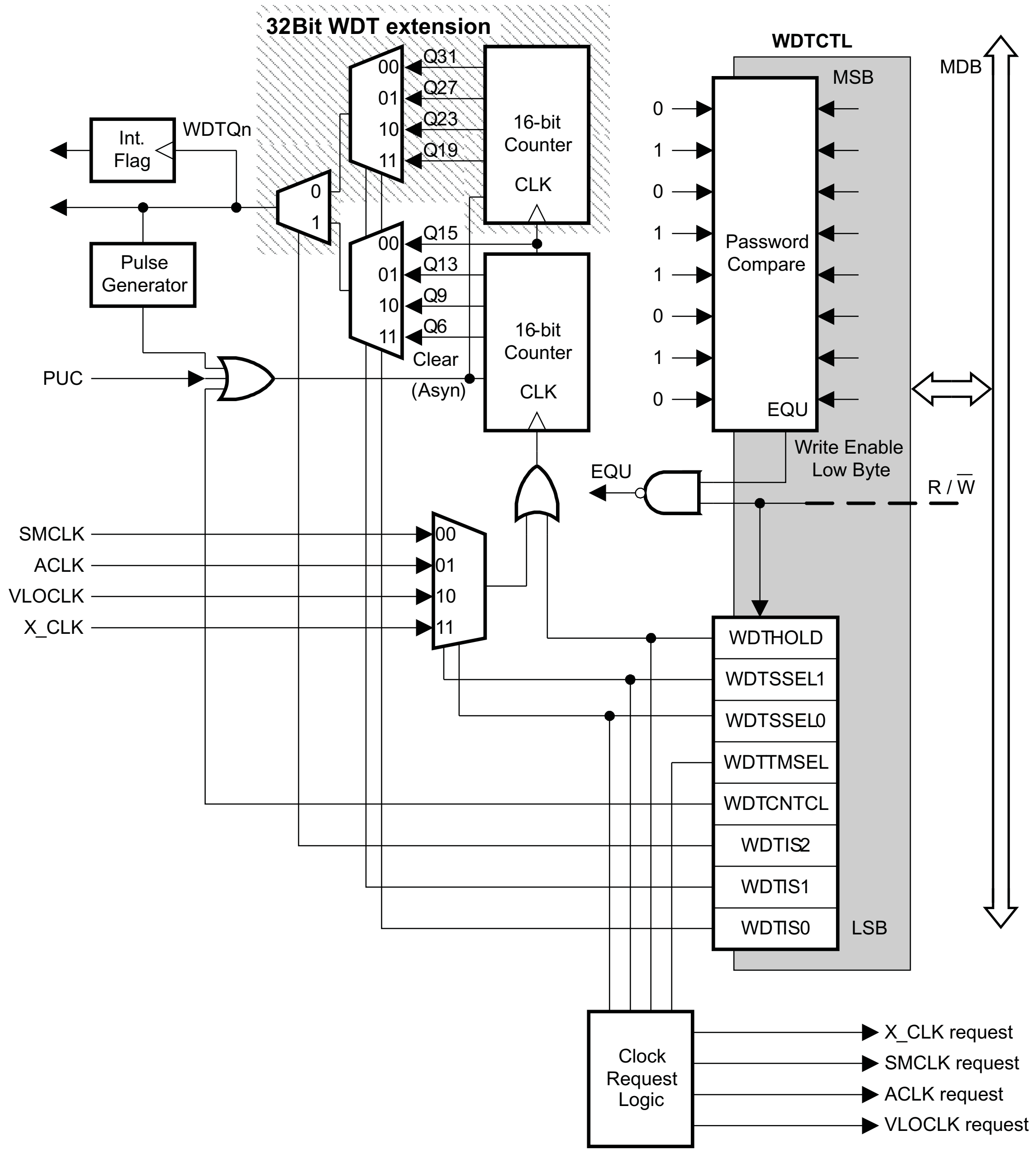


Figure 5-1. Watchdog Timer Block Diagram

5.2 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.

5.2.1 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.

5.2.2 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.

5.2.3 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.

5.2.4 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.

5.2.5 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.

5.2.6 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.

5.2.7 Software Examples

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

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

5.3 WDT_A Registers

The watchdog timer module registers are listed in Table 5-1. The base register or the watchdog timer module registers and special function registers (SFRs) can be found in device-specific data sheets. The address offset is given in Table 5-1.

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 5-1. 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; VLOCLK in devices that do not support X_CLK
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)

Chapter 6
SLAU321–September 2010

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**

6.1 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 6-1.

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.

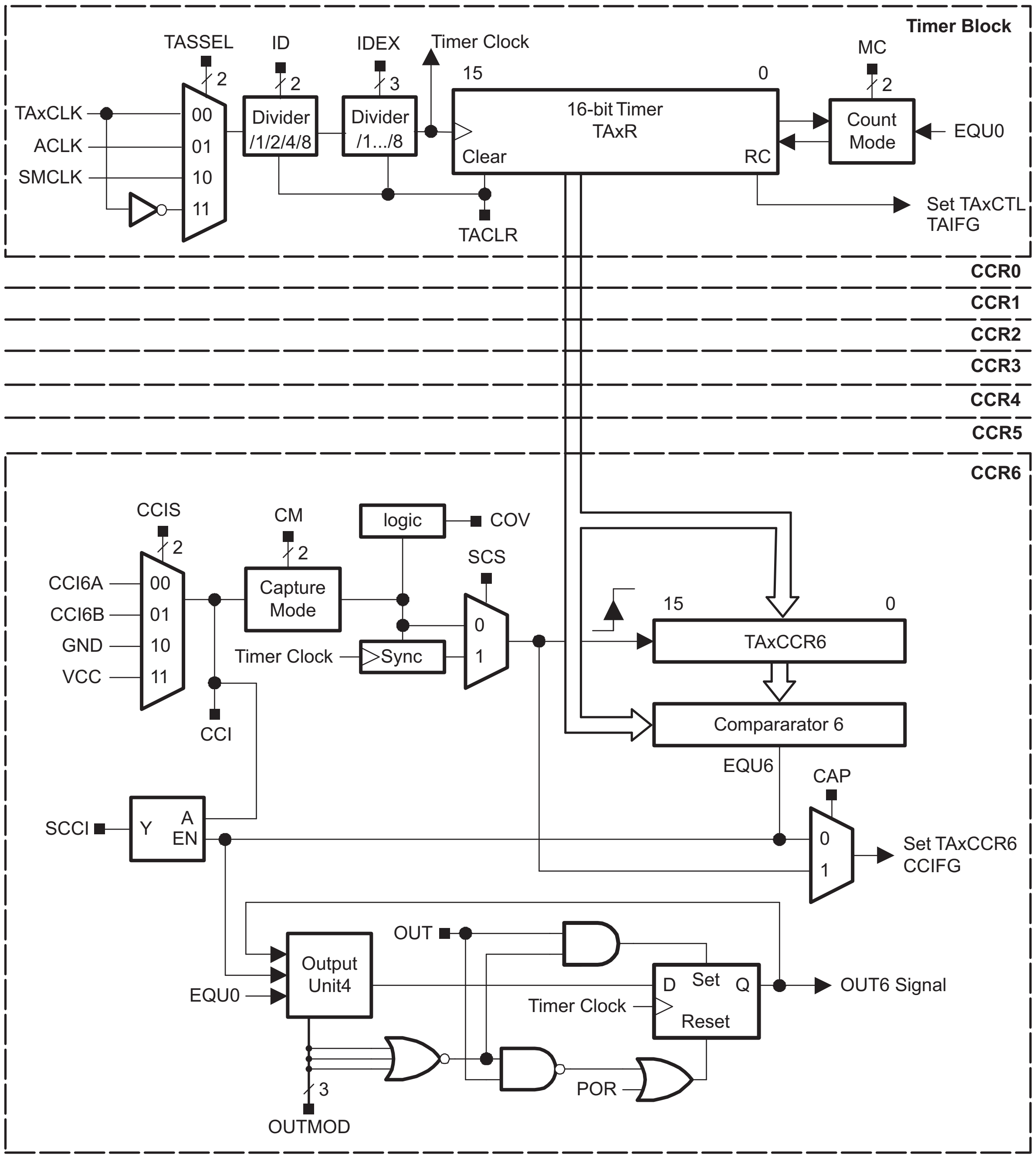


Figure 6-1. Timer_A Block Diagram

6.2 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.

6.2.1 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.

6.2.1.1 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 IDEX 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 IDEX bits.

6.2.2 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.

6.2.3 Timer Mode Control

The timer has four modes of operation: stop, up, continuous, and up/down (see Table 6-1). The operating mode is selected with the MC bits.

Table 6-1. 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.

6.2.3.1 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 6-2). 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.

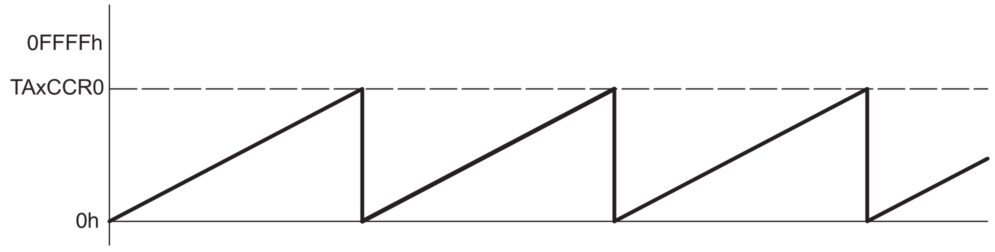


Figure 6-2. Up Mode

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 6-3 shows the flag set cycle.

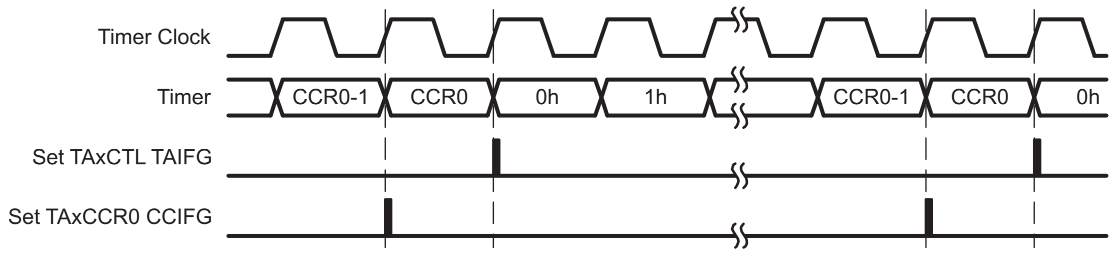


Figure 6-3. Up Mode Flag Setting

6.2.3.1.1 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.

6.2.3.2 Continuous Mode

In the continuous mode, the timer repeatedly counts up to 0FFFFh and restarts from zero as shown in Figure 6-4. The capture/compare register TAxCCR0 works the same way as the other capture/compare registers.

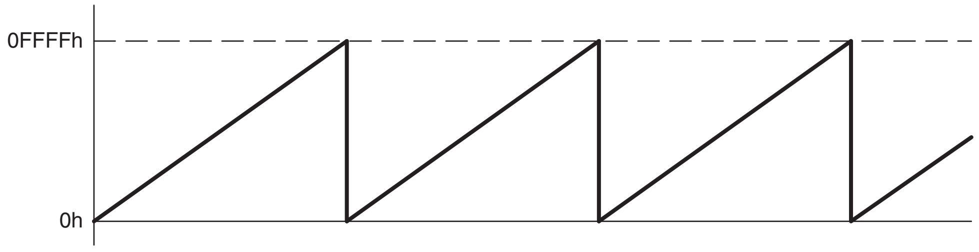


Figure 6-4. Continuous Mode

The TAIFG interrupt flag is set when the timer *counts* from 0FFFFh to zero. Figure 6-5 shows the flag set cycle.

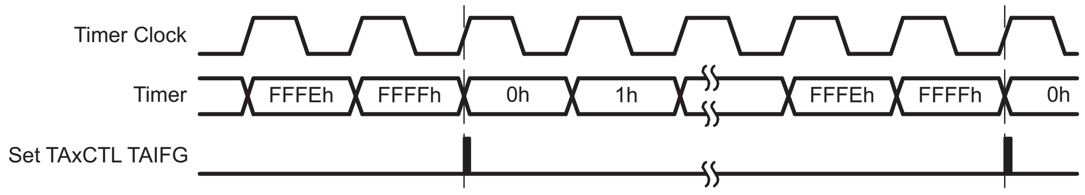


Figure 6-5. Continuous Mode Flag Setting

6.2.3.3 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 6-6 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.

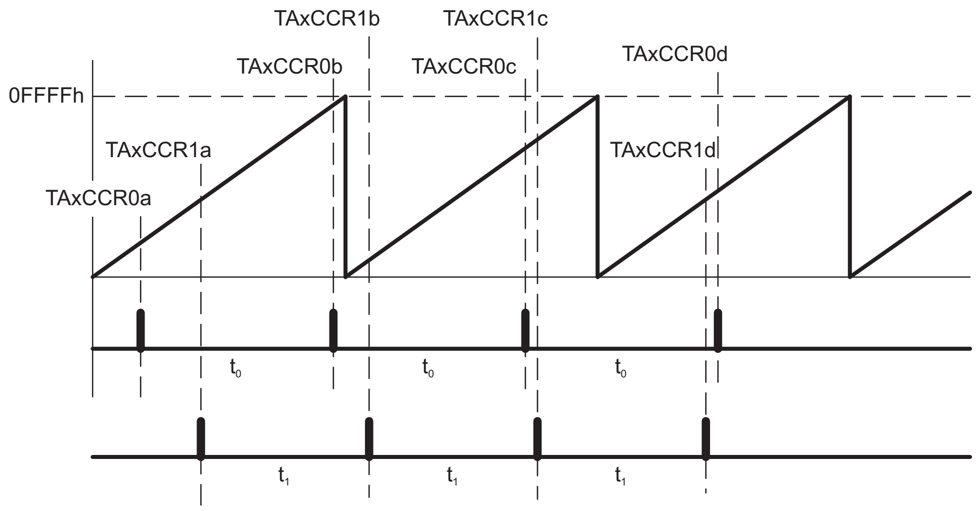


Figure 6-6. Continuous Mode Time Intervals

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.

6.2.3.4 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 6-7). The period is twice the value in TAxCCR0.

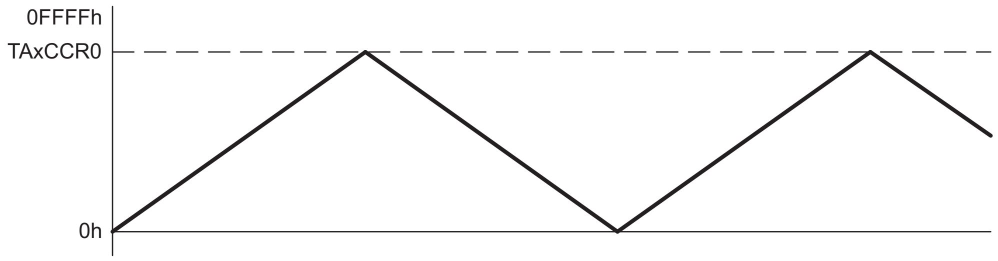


Figure 6-7. Up/Down Mode

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 6-8 shows the flag set cycle.

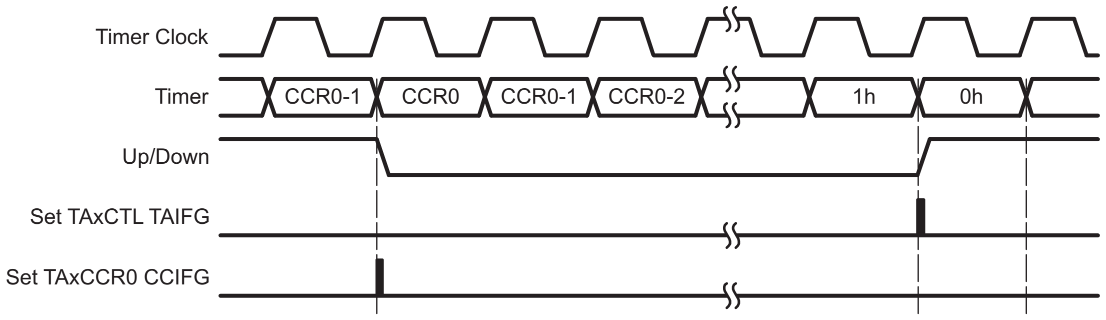


Figure 6-8. Up/Down Mode Flag Setting

6.2.3.4.1 *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.

6.2.3.5 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 6-9, the t_{dead} is:

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

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.

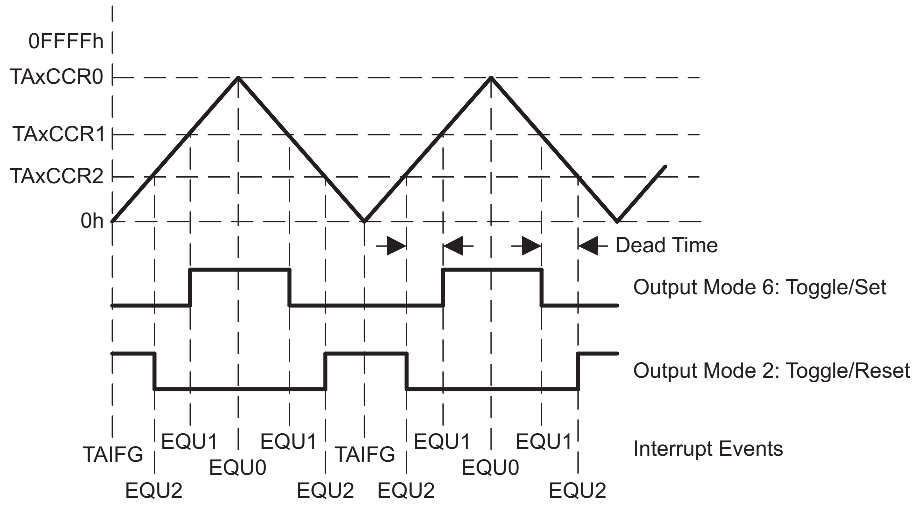


Figure 6-9. Output Unit in Up/Down Mode

6.2.4 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.

6.2.4.1 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 6-10).

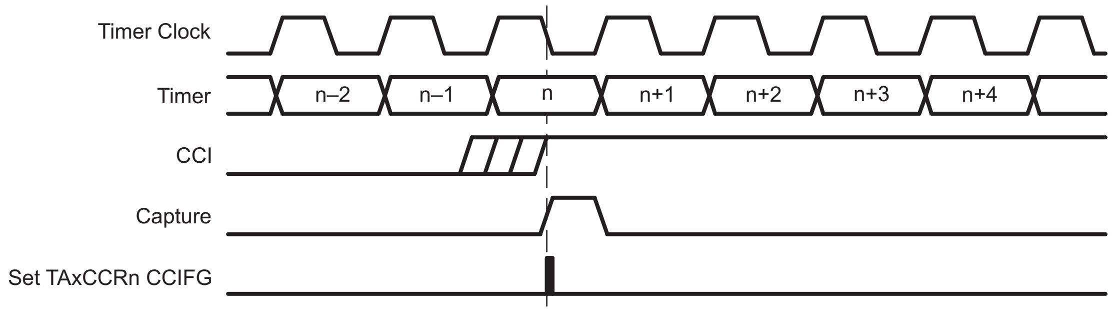
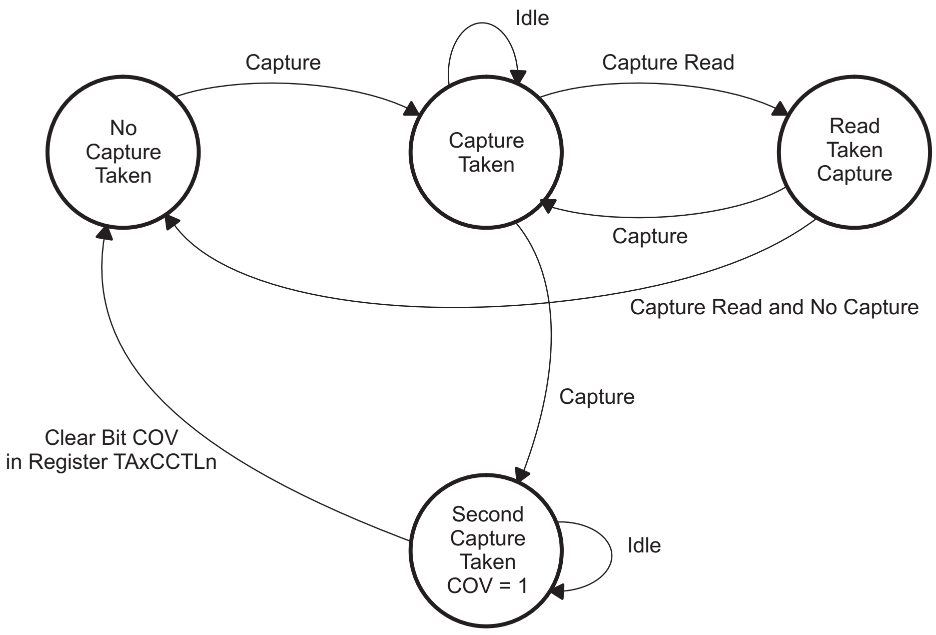


Figure 6-10. Capture Signal (SCS = 1)

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 6-11. COV must be reset with software.



Idle
Capture
Capture Read
No Capture Taken
Capture Taken
Read Taken Capture
Capture
Capture Read and No Capture
Capture
Clear Bit COV in Register TAxCCTLn
Second Capture Taken COV = 1
Idle

Figure 6-11. Capture Cycle

6.2.4.1.1 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.

6.2.4.2 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.

6.2.5 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.

6.2.5.1 Output Modes

The output modes are defined by the OUTMOD bits and are described in Table 6-2. 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 6-2. 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.

6.2.5.1.1 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 6-12 using TAxCCR0 and TAxCCR1.

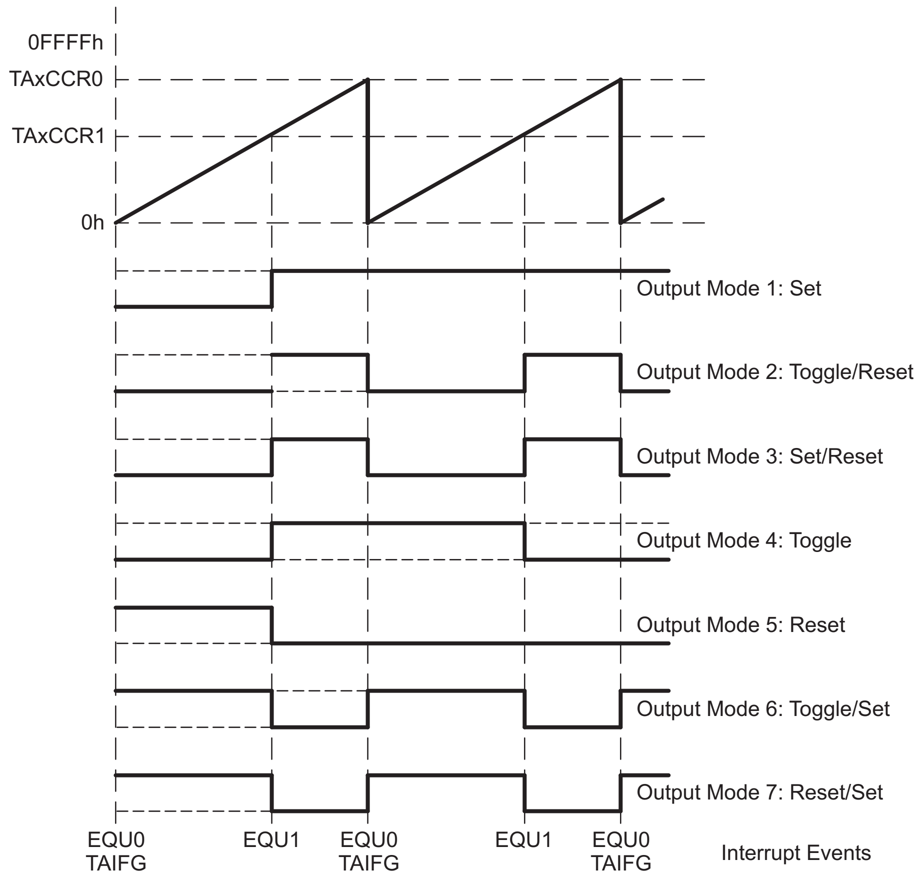


Figure 6-12. Output Example – Timer in Up Mode

6.2.5.1.2 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 6-13 using TAxCCR0 and TAxCCR1.

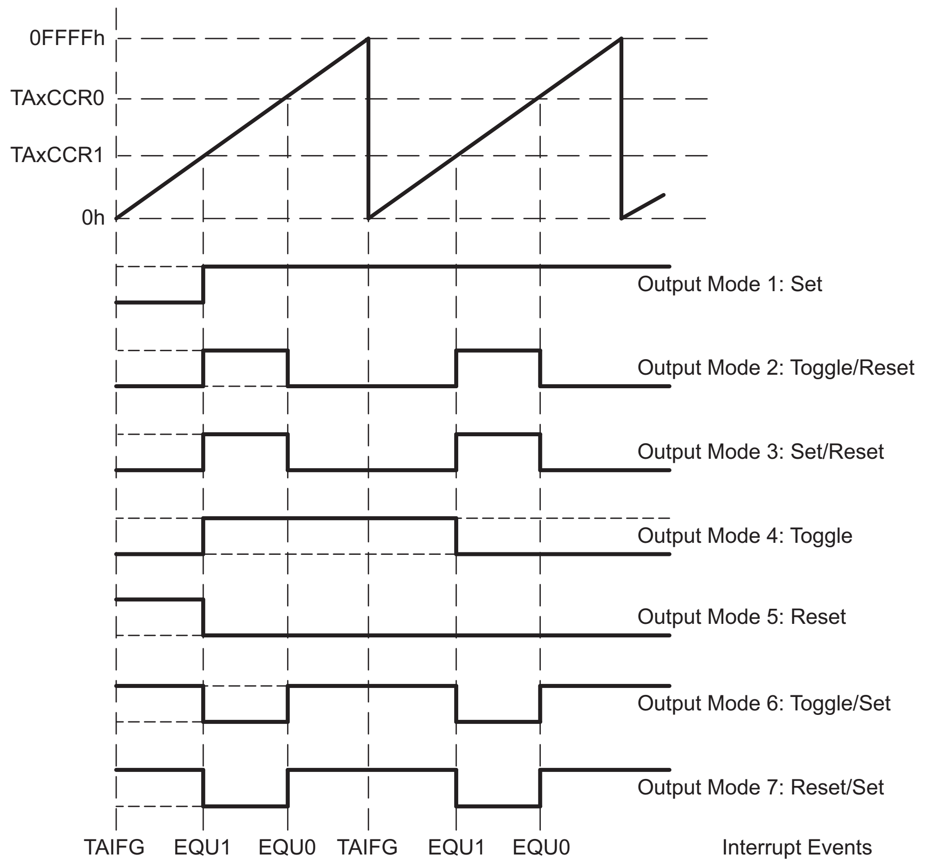


Figure 6-13. Output Example – Timer in Continuous Mode

6.2.5.1.3 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 6-14 using TAxCCR0 and TAxCCR2.

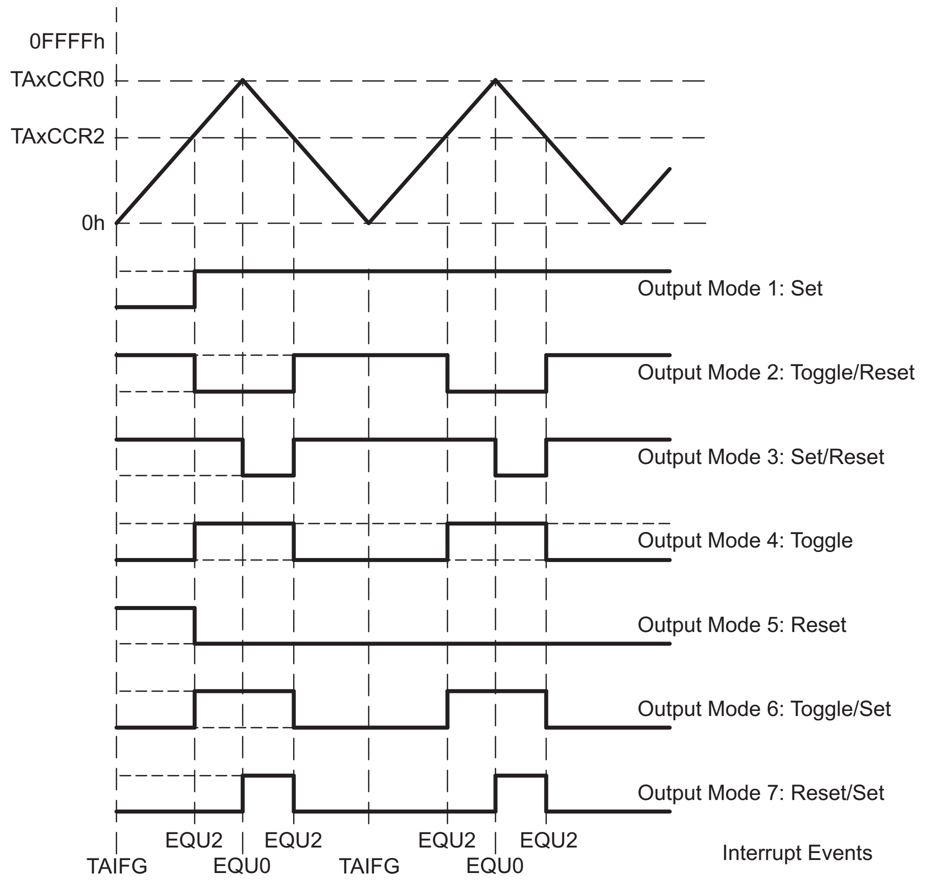


Figure 6-14. Output Example – Timer in Up/Down Mode

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
```

6.2.6 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.

6.2.6.1 TAxCCR0 Interrupt

The TAxCCR0 CCIFG flag has the highest Timer_A interrupt priority and has a dedicated interrupt vector as shown in Figure 6-15. The TAxCCR0 CCIFG flag is automatically reset when the TAxCCR0 interrupt request is serviced.

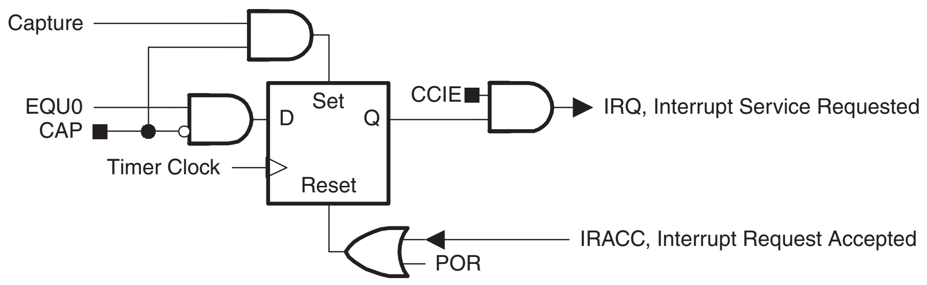

Capture
EQU0
CAP
Timer Clock
D
Set
Q
Reset
CCIE
IRQ, Interrupt Service Requested
IRACC, Interrupt Request Accepted
POR

Figure 6-15. Capture/Compare TAxCCR0 Interrupt Flag

6.2.6.2 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.

6.2.6.2.1 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
```

6.3 Timer_A Registers

Timer_A registers are listed in Table 6-3 for the largest configuration available. The base address can be found in the device-specific data sheet. The address offsets are listed in Table 6-3.

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 6-3. Timer_A Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Timer_A Control	TAxCTL	Read/write	Word	00h	0000h
	TAxCTL_L	Read/write	Byte	00h	00h
	TAxCTL_H	Read/write	Byte	01h	00h
Timer_A Capture/Compare Control 0	TAxCCTL0	Read/write	Word	02h	0000h
	TAxCCTL0_L	Read/write	Byte	02h	00h
	TAxCCTL0_H	Read/write	Byte	03h	00h
Timer_A Capture/Compare Control 1	TAxCCTL1	Read/write	Word	04h	0000h
	TAxCCTL1_L	Read/write	Byte	04h	00h
	TAxCCTL1_H	Read/write	Byte	05h	00h
Timer_A Capture/Compare Control 2	TAxCCTL2	Read/write	Word	06h	0000h
	TAxCCTL2_L	Read/write	Byte	06h	00h
	TAxCCTL2_H	Read/write	Byte	07h	00h
Timer_A Capture/Compare Control 3	TAxCCTL3	Read/write	Word	08h	0000h
	TAxCCTL3_L	Read/write	Byte	08h	00h
	TAxCCTL3_H	Read/write	Byte	09h	00h
Timer_A Capture/Compare Control 4	TAxCCTL4	Read/write	Word	0Ah	0000h
	TAxCCTL4_L	Read/write	Byte	0Ah	00h
	TAxCCTL4_H	Read/write	Byte	0Bh	00h
Timer_A Capture/Compare Control 5	TAxCCTL5	Read/write	Word	0Ch	0000h
	TAxCCTL5_L	Read/write	Byte	0Ch	00h
	TAxCCTL5_H	Read/write	Byte	0Dh	00h
Timer_A Capture/Compare Control 6	TAxCCTL6	Read/write	Word	0Eh	0000h
	TAxCCTL6_L	Read/write	Byte	0Eh	00h
	TAxCCTL6_H	Read/write	Byte	0Fh	00h
Timer_A Counter	TAxR	Read/write	Word	10h	0000h
	TAxR_L	Read/write	Byte	10h	00h
	TAxR_H	Read/write	Byte	11h	00h
Timer_A Capture/Compare 0	TAxCCR0	Read/write	Word	12h	0000h
	TAxCCR0_L	Read/write	Byte	12h	00h
	TAxCCR0_H	Read/write	Byte	13h	00h
Timer_A Capture/Compare 1	TAxCCR1	Read/write	Word	14h	0000h
	TAxCCR1_L	Read/write	Byte	14h	00h
	TAxCCR1_H	Read/write	Byte	15h	00h
Timer_A Capture/Compare 2	TAxCCR2	Read/write	Word	16h	0000h
	TAxCCR2_L	Read/write	Byte	16h	00h
	TAxCCR2_H	Read/write	Byte	17h	00h

Table 6-3. Timer_A Registers (continued)

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
Timer_A Capture/Compare 3	TAxCCR3	Read/write	Word	18h	0000h
	TAxCCR3_L	Read/write	Byte	18h	00h
	TAxCCR3_H	Read/write	Byte	19h	00h
Timer_A Capture/Compare 4	TAxCCR4	Read/write	Word	1Ah	0000h
	TAxCCR4_L	Read/write	Byte	1Ah	00h
	TAxCCR4_H	Read/write	Byte	1Bh	00h
Timer_A Capture/Compare 5	TAxCCR5	Read/write	Word	1Ch	0000h
	TAxCCR5_L	Read/write	Byte	1Ch	00h
	TAxCCR5_H	Read/write	Byte	1Dh	00h
Timer_A Capture/Compare 6	TAxCCR6	Read/write	Word	1Eh	0000h
	TAxCCR6_L	Read/write	Byte	1Eh	00h
	TAxCCR6_H	Read/write	Byte	1Fh	00h
Timer_A Interrupt Vector	TAxIV	Read only	Word	2Eh	0000h
	TAxIV_L	Read only	Byte	2Eh	00h
	TAxIV_H	Read only	Byte	2Fh	00h
Timer_A Expansion 0	TAxEX0	Read/write	Word	20h	0000h
	TAxEX0_L	Read/write	Byte	20h	00h
	TAxEX0_H	Read/write	Byte	21h	00h

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 IDEX 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

(continued)

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**		**IDEX**	
r0	r0	r0	r0	r0	rw-(0)	rw-(0)	rw-(0)

Unused	Bits 15-3	Unused. Read only. Always read as 0.	
IDEX	Bits 2-0	Input divider expansion. These bits along with the 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

Chapter 7

SLAU321 – September 2010

ADC, DAC, Comparator, SVM, ASVM, Analog Functions Pool Module (A-POOL)

The Analog Functions Pool (A-POOL) module is integrated into various devices with different feature sets. It provides the necessary functions to implement ADCs, DACs, and SVMs with various features.

Some elementary functions of A-POOL include:

- Reference voltage source
- Comparator
- Eight-bit elementary DAC
- Successive approximation register (SAR)
- Support for integer and fractional number representation

Topic **Page**

7.1 Analog-Functions Pool Module Introduction

The Analog-Functions Pool Module (A-POOL) provides the necessary elements to build DACs, ADCs, and other analog functions.

7.2 Principle of Operation

Analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) are complex analog functions that consists of analog and digital components, some types use compensation methods and auto-zero (AZ) mechanisms to eliminate error sources. Modern converters provide automatic range control and other advanced features. A-POOL has none of those complex functions as ready modules; instead, it provides analog and analog-oriented digital elementary functions that can be used to build complex analog functions like DACs, ADCs, and SVMs of different kinds when combined through software.

7.2.1 Analog Elementary Functions

A summary of analog capabilities is listed below. Not all of those functions may be available at the same time.

- Internal low-voltage reference source
- Ability to use external voltage reference
- Four independent analog inputs
- Two analog buffered outputs for DAC and reference
- Independently programmable voltage dividers for analog inputs
- Low-offset analog comparator with two speed ranges and complementary output
- Eight-bit DAC element (non-compensated)
- High-precision dual-range supply voltage divider
- Internal temperature sensor with temperature-proportional sensor voltage
- Analog signal paths allowing cross compensation on all eight internal sources

7.2.2 Digital Elementary Functions

A summary of the analog oriented digital functions is listed below. Not all of those functions may be available at the same time.

- Control elements for all analog signal paths, input voltage dividers, and other resistive paths
- Control elements for enabling and setting the operation mode of reference source, comparator, and analog buffer elements
- Saturating logic for the 8-bit up/down read/write counter
- Clock multiplexer allowing two independent time-based ramps
- Programmable clock prescaler for division rates 2^N, up to 32
- Start/stop logic for counter clock with flexible event control
- Write and read logic for integer and fractional number representation
- Digital deglitching filter on comparator output
- Successive approximation register (SAR)

7.3 A-POOL Analog Components and Paths

The elements provided by A-POOL are suitable for complex function with up to 8-bit dynamics and resolution. The multiplexers allow for high flexibility in input selections. All input signals are available on the inverting and non-inverting input of the comparator, allowing elimination offsets and dynamic errors by cross compensation. The voltages carried on the internal analog paths range from ground (V_{SS}) to approximately 256 mV. Higher voltage levels are scaled to this range. Figure 7-1 shows the block diagram of the A-POOL analog components and analog signal paths.

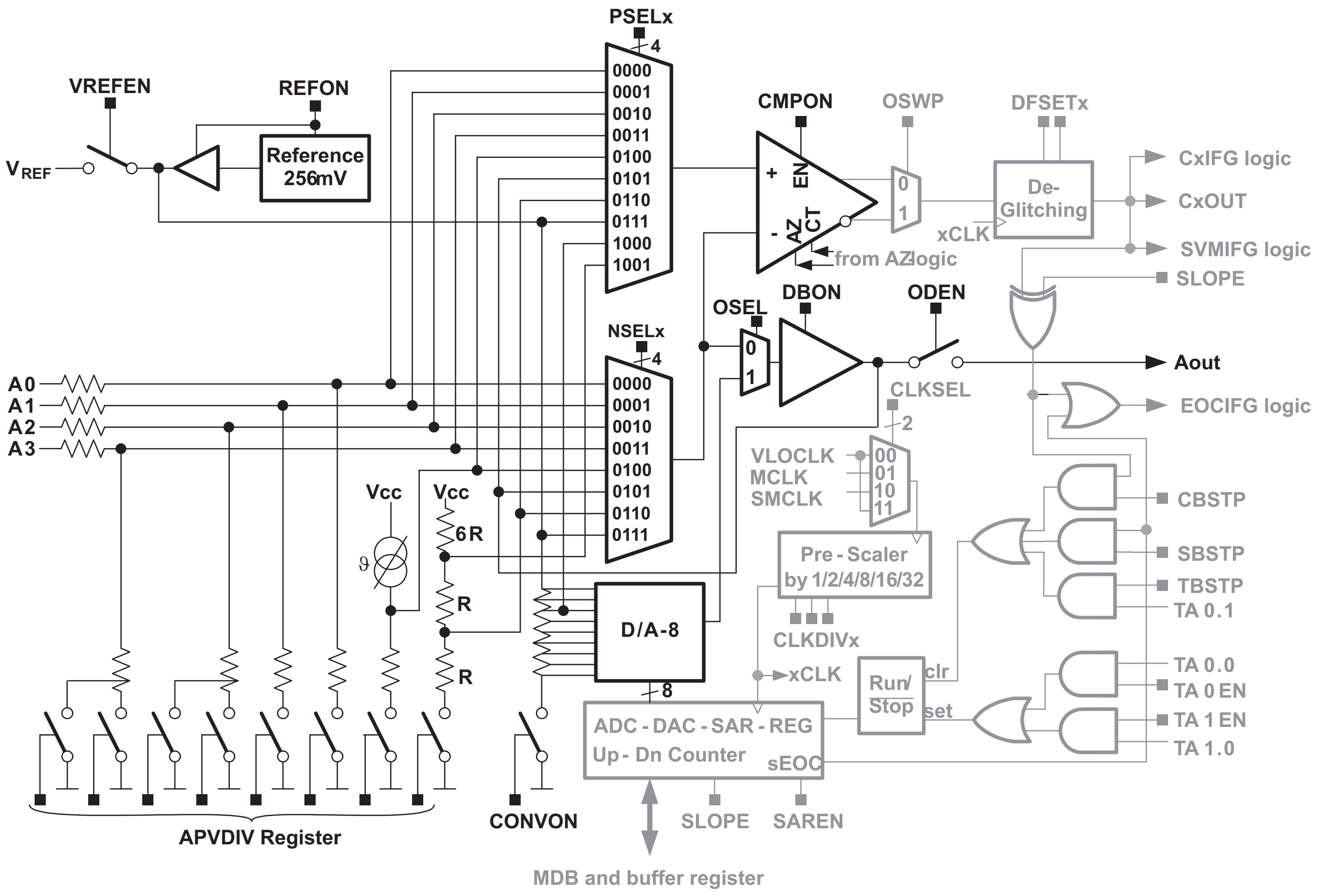


Figure 7-1. A-POOL Analog Components and Signal Paths (With Digital Components in Gray)

7.3.1 Reference Voltage Source

A-POOL provides an internally buffered ultra-low-voltage (ULV) reference voltage source. This reference source operates at a very low voltage of approximately 256 mV, allowing supply voltages far below 1 V. The reference voltage can be trimmed by the user to compensate small mismatches. DAC conversions, ADC conversions, supply-voltage monitoring, and temperature measurements require a reference voltage. For those functions, REFON and VREFEN are set to 1, unless an external reference voltage is applied on VREF. The internal reference source generates an interrupt (REFOKIFG) as soon as the voltage has settled after switching the reference on; thus, time-consuming delay loops can be avoided.

7.3.2 Internal vs External Reference Voltage Source

To improve the overall accuracy of A-POOL or to scale the A-POOL functions to another reference voltage range, an external reference voltage source can be connected on the external VREF terminal. The internal reference voltage source should then remain off (REFON = 0, VREFEN = 1).

7.3.3 Temperature Sensor

The internal temperature sensor of A-POOL generates a voltage proportional to the temperature measured. The temperature sensor is powered from the internal reference voltage source only. Temperature measurement with an external reference voltage source is not possible.

7.3.4 Input Voltage Dividers

Many voltages in single-battery cell applications are higher than the nominal 256 mV used as internal operation range for A-POOL-main. High-precision voltage dividers allow an extension of the voltage range to 500 mV, 1 V and 2 V for various analog inputs. The predefined nominal division ratios are selectable (1, 2, 4, or 8; for higher accuracy, use the true division ratio given in the device-specific data sheet).

7.3.5 Comparator in Non-Compensated Mode

The A-POOL comparator is designed to operate from ground (V_{SS}) to the nominal 256 mV. This comparator can be operated in non compensated mode when CTEN = 1, CT-mode (continuous time). In CT-mode, the full speed of the comparator is used, at the cost of a small internal offset voltage. The offset voltage can be compensated in most applications with help of proper software using the same techniques known from chopper stabilized amplifiers, etc. Figure 7-2 shows that the comparator of A-POOL in CT-mode and the model with optimal comparator and the error sources offset voltage and overdrive.

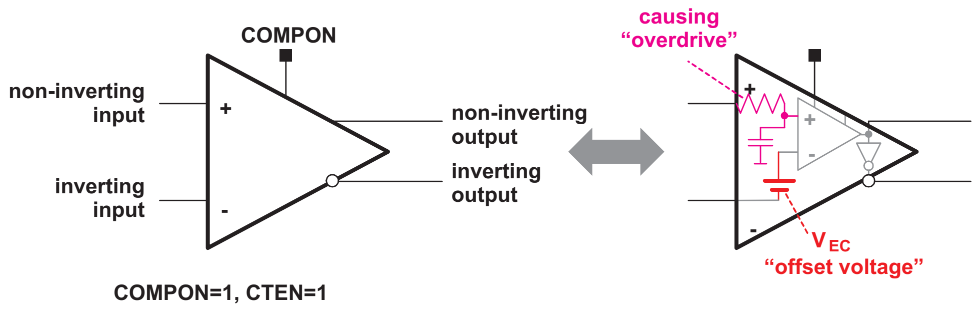


Figure 7-2. Comparator in Non-Compensated Mode

About overdrive: Overdrive is the input difference voltage needed to get a distinct result from the comparator within a defined response time. The higher the overdrive, the faster the reaction. Overdrive is an auxiliary parameter to describe the dynamic behavior of a comparator. The overdrive of real comparators is nonlinear; it depends on the voltage range, temperature and supply voltage.

About offset voltage: Offset voltage is a DC error voltage that needs to be overcome to cause a change on the comparator's output.

Overdrive and offset effects are independent from each other and are superimposed on real comparators. The comparator in A-POOL-main is a real comparator; compensation methods are shown later.

7.3.6 Comparator in Compensated Mode

The A-POOL comparator may be operated in compensated mode by setting CTEN = 0 and AZCMP = 1. In this mode, the comparator changes its operation mode cyclicly between auto-zero phase, recovery phase, and compare phase as shown in Figure 7-3. Inserting cyclic auto-zero phases between compare phases eliminates the errors from the internal offset voltage of the comparator. The output signal of the comparator is valid only during the compare phase (50% of the time). The overall response of the comparator appears to be slower in this mode (based on frequency and phase of xCLK to the observed signal). An auto-zero cycle can be requested by software by setting AZSWREQ = 1, and the successive approximation register (SAR) also requests an AZ cycle.

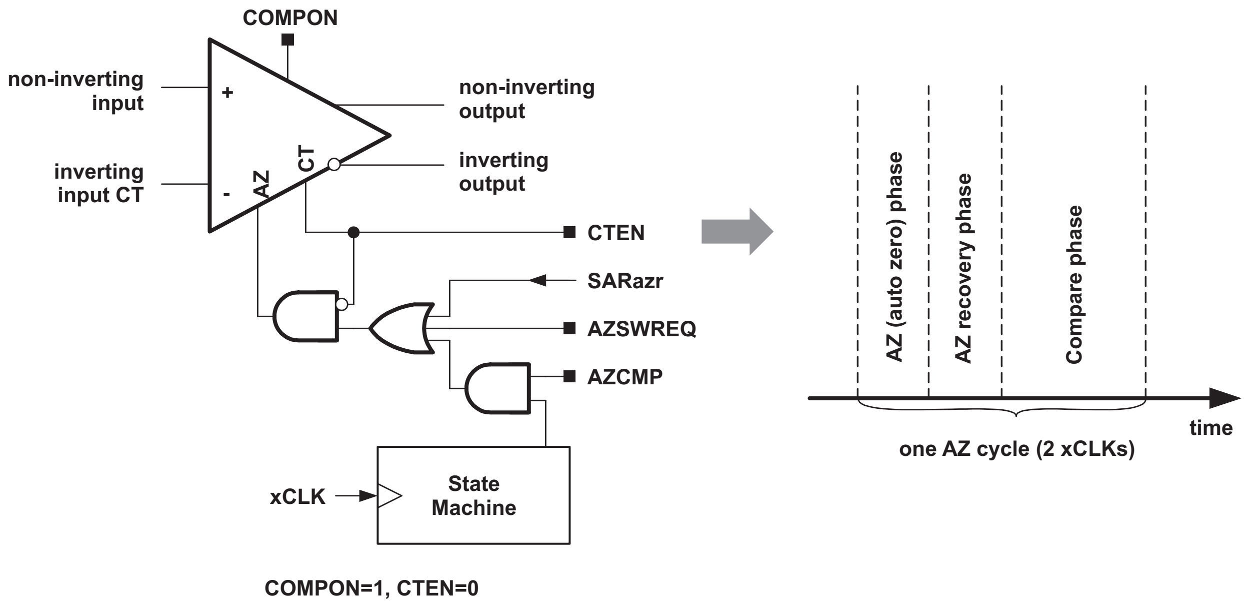


Figure 7-3. Comparator in Compensated Mode

7.3.7 DAC and Output Buffer

The DAC converts the eight bits from the conversion register directly to an analog voltage ranging from 1 mV for value 0 up to 256 mV for value 255 (in eight bit unsigned integer representation). The output buffer provides enough strength to drive high impedance loads. This output is available internally and externally when ODEN = 1.The DAC output buffer may shows a small voltage offset that can be compensated (as shown in the application examples). High accuracy on AOUT is reached when the output voltage is above 20 mV.

7.4 A-POOL Digital Components and Paths

The digital components of A-POOL and their paths are shown in Figure 7-4.

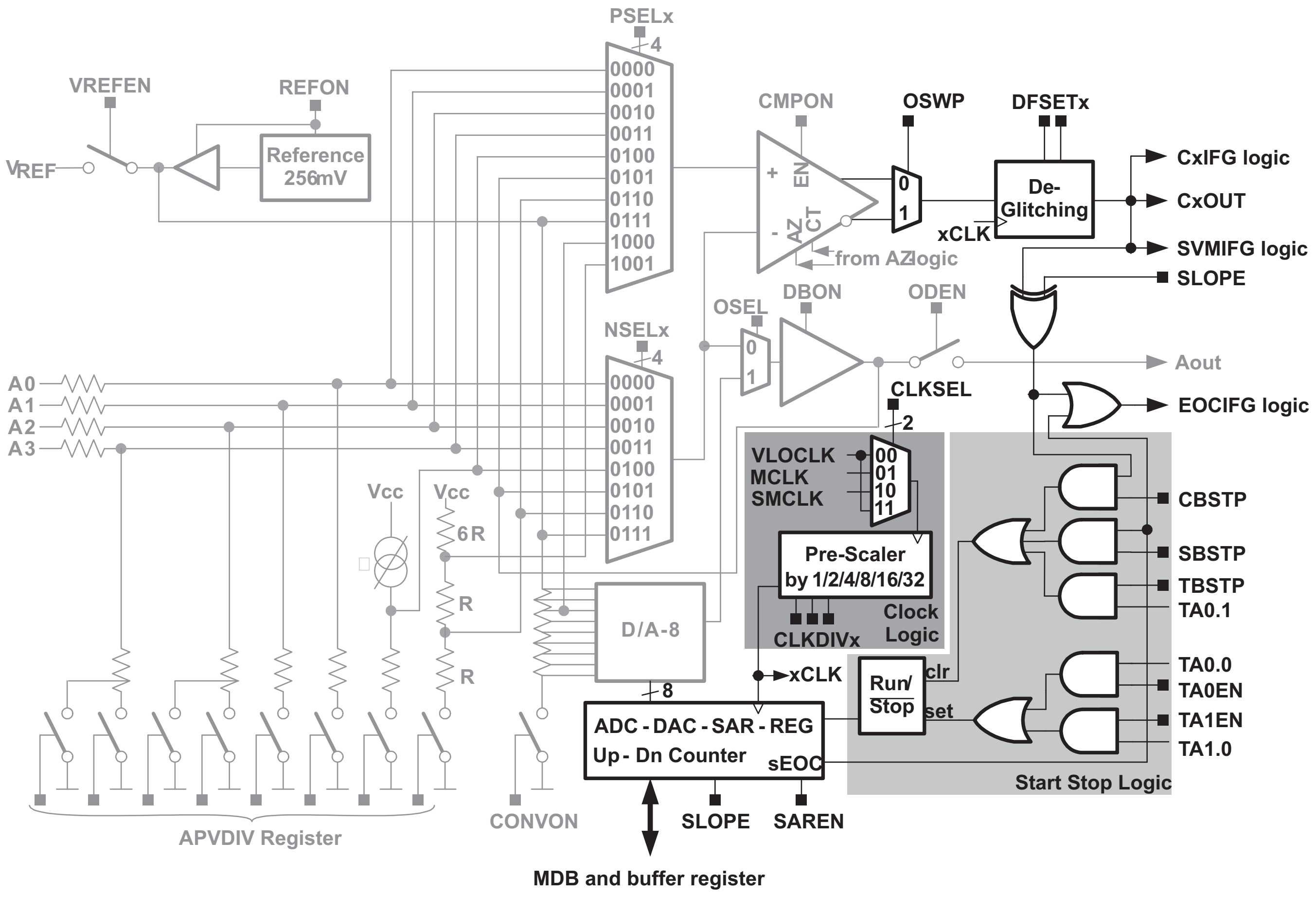


Figure 7-4. A-POOL Digital Components (With Analog Components in Gray)

7.4.1 *Deglitching filter*

The deglitching filter on the comparator output reduces unstable conditions that are introduced from input voltages that contain harmonics and other distortions. Four settings allow stable switching responses.

Table 7-1. Deglitching Filter

DFSETx	Behavior
00	No deglitching, the comparator output is feedthrough.
01	Two out of two consecutive samples, based on xCLK, must be high to generate a high output.
10	Majority vote of two out of three consecutive sample, based on xCLK, must be high to generate a high output.
11	Majority vote on three of five consecutive samples, based on xCLK, must be high to generate a high output.

7.4.2 Clock Logic and Prescaler

xCLK, the operation clock for the conversion counter, SAR logic, and deglitching filter, is provided by the clock logic. The CLKSEL bits determine if VLOCLK, MCLK, or SMCLK is selected to feed the prescaler. CLKDIVx determines the division rate from the selected clock to xCLK.

The operation clock for A-POOL is enabled as soon it is required for internal operation; otherwise, it remains off to conserve power.

Table 7-2. Clock enable for A-POOL in Various Operation Cases

RUNSTOP	CTEN	AZCMP	AZSWREQ	A-POOL Clock	Mode
1	x	x	x	Enabled	Any conversion forced/automatic
x	x	1	x	Enabled	Any auto-zero operation
x	x	x	1	Enabled	During software auto-zero operation

7.4.3 Conversion Register and Conversion Buffer Register

Conversion Register and conversion buffer register are real 8-bit registers that are accessed via the registers APINT/APFRACT and APINTB/APFRACTB. These registers were introduced to allow both number representations that are commonly used in digital control and signal processing and to allow interim results to be calculated by the CPU with 16-bit dynamics, while only the eight most "important" bits are used for conversions.

Setting ATBU = 1 enables an automatic update of the DAC register on a TA0.1 pulse. This is required in signal-processing algorithms and some control applications to avoid spectral distortions in the DAC path. Setting EOCBU updates the ADC buffer with the last ADC conversion value. This is helpful when multi-channel ADC conversions are done.

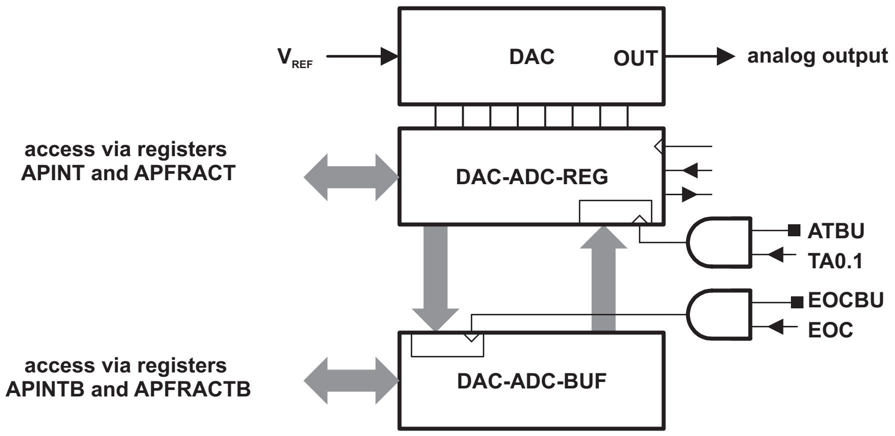

V_REF
DAC
OUT
analog output
access via registers APINT and APFRACT
DAC-ADC-REG
ATBU
TA0.1
EOCBU
EOC
access via registers APINTB and APFRACTB
DAC-ADC-BUF

Figure 7-5. Conversion Register and Conversion Buffer

7.4.4 Fractional and Integer Numbers

The register set of A-POOL features a register pair for integer numbers and fractional numbers. Unsigned integer numbers can be read from and written to the APINT / APINTB registers, signed fractional numbers of format Q7 and Q15 can be read from and written to the APFRACT / APFRACTB registers. The true data width of A-POOL is 8 bit, but word-wide integer and fractional numbers may still be used for digital control applications. The bit assignment of the APINT / APINTB and APFRACT / APFRACTB registers is done automatically by accessing the appropriate registers shown in Table 7-3.

Table 7-3. Fractional and Integer Values used with A-POOL

Number Type	Use Register	Number Range (Decimal)	Number Range Used (Hex)	A-POOL Internal Resolution
8-bit integer	APINT_L , APINTB_L	0 to 255	00h to FFh	8 bits in APINT_L
16-bit integer	APINT, APINTB	0 to 255	0000h to 00FFh	8 bits in APINT_L
Q7 fractional	APFRACT_H, APFRACTB_H	-128 to 127	80h to 7Fh	8 bits in APFRACT_H
Q15 fractional	APFRACT, APFRACTB	-32768 to 32767	8000h to 7F00h	8 bits in APFRACT_H

Table 7-4. Integer, Q7, Q15 and Corresponding Internal Voltages

8/16-Bit Integer Representation	Q7 Fractional Representation	Q15 Fractional Representation	Internal Equivalent Voltage Based on V_{REF} = 256 mV
00h	80h	8000h to 80FFh	1 mV
⋮	⋮	⋮	⋮
7Fh	FFh	FF00h to FFFFh	128 mV
80h	00h	0000h to 00FFh	129 mV
⋮	⋮	⋮	⋮
FFh	7Fh	7F00h to 7FFFh	256 mV

7.4.5 Numeric Saturation and End of Conversion Indication

A-POOL uses an 8-bit-wide up/down counter as ramp generator. Access to this counter is through the APINT / APINTB and APFRACT / APFRACTB registers. The values 0 and 255 represent the end-of-conversion range. The up/down counter suppresses a wraparound when SBSTP is set. The end-of-conversion flag, EOCIFG, is set independent from the value of SBSTP.

Table 7-5. Saturation Schemes for Up and Down Ramps[1]

Conditions	SLOPE = 0, SBSTP = 0			SLOPE = 0, SBSTP = 1			SLOPE = 1, SBSTP = 0			SLOPE = 1, SBSTP = 1		
	Counter Value		EOCs Ind.	Counter Value		EOCs Ind.	Counter Value		EOCs Ind.	Counter Value		EOCs Ind.
Clock	DEC	HEX		DEC	HEX		DEC	HEX		DEC	HEX	
Q…	…	…	…	…	…	…	…	…	…	…	…	…
Q_N	253	FDh	0	253	FDh	0	2	02h	0	2	02h	0
Q_{N+1}	254	FEh	0	254	FEh	0	1	01h	0	1	01h	0
Q_{N+2}	255	FFh	1	255	FFh	1	0	00h	1	0	00h	1
Q_{N+3}	0	00h	0	255	FFh	0	255	FFh	0	0	00h	0
Q_{N+4}	1	01h	0	255	FFh	0	254	FEh	0	0	00h	0
Q_{N+5}	2	02h	0	255	FFh	0	253	FDh	0	0	00h	0

(1) EOCs = end of conversion due to saturation

7.4.6 *Interrupt Logic*

A-POOL can set up to four different interrupt flags for various purposes. Table 7-6 summarizes types and sources for the interrupt flags.

Table 7-6. Interrupt Logic Behavior

Trigger Event	CxIFG	SVMIFG	EOCIFG	REFOKIFG
ADC-DAC counting to end of range	–	–	Synchronous interrupt	–
SAR state machine end-state	–	–	Synchronous interrupt	–
Regular crossover (no ramp, no SAR)	Synchronous interrupt	Asynchronous interrupt	–	–
VREF reaching correct voltage after turning on	–	–	–	Synchronous interrupt

An internal state machine prevents unintentional interrupt generation during SAR conversions, auto-zero compensations and ramp based operations.

7.5 Simple Application Examples With A-POOL-main

The following examples show some methods for building complex functions from the available elementary functions. In most cases, more than two different programming styles for event detection can be applied. Here, wait periods are implemented as LPM0 phases that are terminated by an interrupt event. This method results in simple example code compared to concurrent CPU operation while interrupt-controlled data acquisition is done in the background, and it avoids power-consuming event polling. In general, all programming styles can be applied.

7.5.1 *DAC Operation for Classical Digital Control Purposes*

A DAC operation for classical digital control purposes (P-I-D and combos) is a simple write to the APINT register or APFRACT register while REFON, OSEL, DBEN, and ODEN bits are set to 1. A corresponding analog voltage with a range of 1 mV to 256 mV is driven on AOUT. Instead of using the internal reference voltage, an external reference voltage may be applied to VREF (set REFON = 0). A compensation of the offset voltages of the DAC output buffer is not required, as most digital control loop algorithms accept those offsets as actuator deviation or part of the disturbance signal. For digital control loops, a moderate resolution and monotonic behavior are more important.

7.5.2 *ADC Conversions Without Error Compensation*

An ADC conversion for digital control purposes (P-I-D and combos) is a single ramp conversion using the ADC-DAC-REG with the DAC as a ramp generator. In the example below, the counter is cleared by software, conversion is triggered by software, and the comparator stops the counter as soon the ramp voltage crosses the selected input voltage.

A compensation of the offset voltages of the DAC output buffer is not required, as most digital control loops algorithms accept those offsets as sensor deviation or part of the disturbance signal. For digital control loops, a moderate resolution and monotonic behavior are more important.

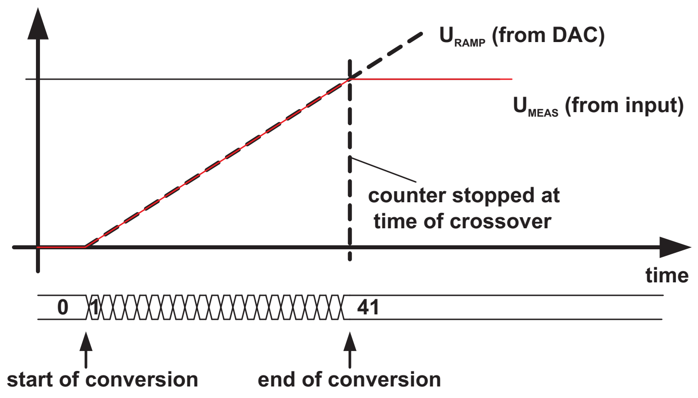


Figure 7-6. Simple ADC Conversion Principle

Example 7-1. Simple ADC Conversion

```
...
;A-POOL Interrupt handler for simple ADC conversions
APHNDL:   MOV.W   #0,&APIV        ;clear all pending interrupts
          BIC.W   #CPUOFF,0(SP)   ;exit LPM0 and return
          RETI
...
.
...
;A-POOL simple SW triggered ADC conversion function for 8 bit integer
;on channel A0 with int. Vref, .5V range, low power, event driven
SADC8:    MOV.B   #1,&APIE        ;enable end of conversion interrupt
          MOV.W   #21E0h,&APCNF   ;enable required anal. elements
          BIS.B   #A0DIV,&APVDIV  ;set the voltage divider to 500 mV
          MOV.W   #0,&APINT       ;clear ADC-DAC-REG
          MOV.W   #5504h,&APCTL   ;set channels and start conversion
          BIS.W   #CPUOFF,SR      ;enter LPM0 and wait
          MOV.W   &APINT,R12      ;handover int. value in R12
          RET                     ;return to calling function
```

7.5.3 ADC Conversions With Overdrive Compensation

The counter that generates the ramp voltage is not stopped immediately on the crossover point with the input voltage due to overdrive. Some "input difference voltage" is required for some time to cause a comparator reaction. In the example below, this "overdrive" is building up gradually after the crossover point, as the input signal can be seen constant compared to the ramp voltage. The product of "overdrive" and time would normally be a "pulse"; in the case of nonlinear behavior, it has the characteristics of an "inertia". In the up-ramp case (see Figure 7-7), this inertia causes a delayed end of conversion, resulting a value that is higher than the actual voltage. In the down-ramp case (see Figure 7-8), this inertia causes a delayed end of conversion, resulting a value that is lower than the actual value. Adding both values together provides a 9-bit conversion value that is overdrive compensated.

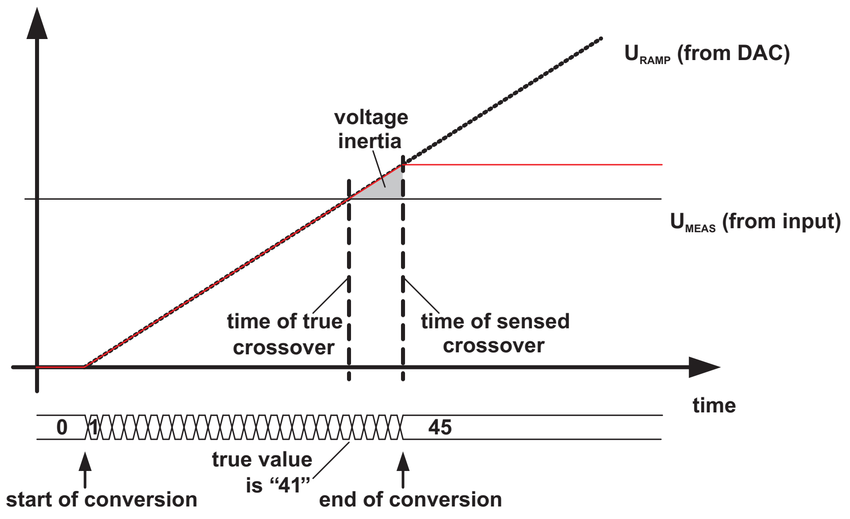


Figure 7-7. Overdrive Compensation Up-Ramp

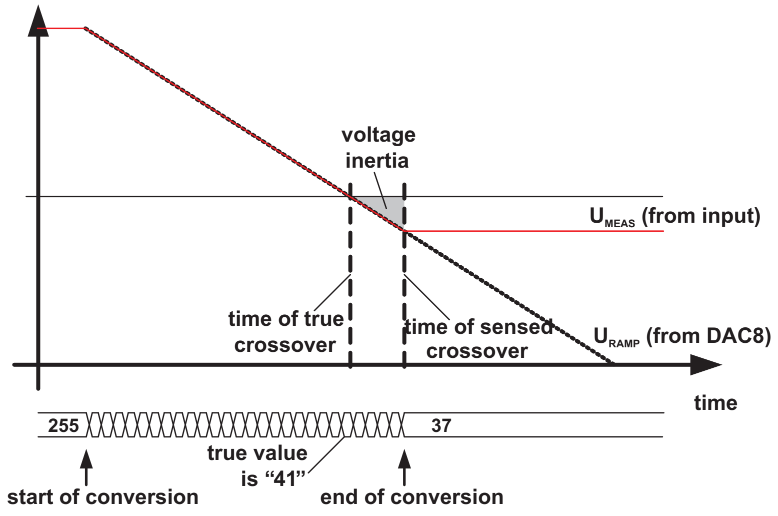


Figure 7-8. Overdrive Compensation Down-Ramp

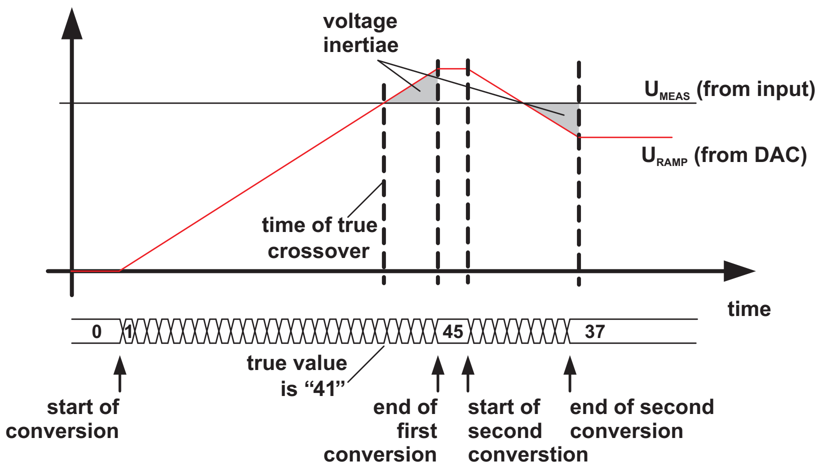


Figure 7-9. Overdrive Compensation by Up/Down-Ramp Concatenation

Example 7-2. ADC Conversion With Overdrive Compensation

```
.
...
;A-POOL Interrupt handler for simple ADC conversions
APHNDL:   MOV.W   #0,&APIV                ;clear all pending interrupts
          BIC.W   #CPUOFF,0(SP)           ;exit LPM0 and return
          RETI
...
.
...
;A-POOL SW triggered overdrive comp. ADC conversion for 9 bit integer
;.. on channel A0 with int. Vref, .5V range, low power, event driven
ICADC9:   MOV.B   #1,&APIE                ;enable end of conversion interrupt
          MOV.W   #21E0h,&APCNF           ;enable required anal. elements
          BIS.B   #A0DIV,&APVDIV          ;set the voltage divider to 500 mV
          MOV.W   #0,&APINT               ;clear ADC-DAC-REG
          MOV.W   #5504h,&APCTL           ;set channels and start conversion
          BIS.W   #CPUOFF,SR              ;enter LPM0 and wait
          MOV.W   &APINT,R12              ;handover int. value in R12
          BIS.W   #RUNSTOP+SLOPE,&APCTL   ;set down and start conversion
          BIS.W   #CPUOFF,SR              ;enter LPM0 and wait
          ADD.W   &APINT,R12              ;compose 9 bit result in R12
          RET                             ;return to calling function
```

7.5.4 ADC Conversions With Offset Compensation

The offset voltage of the comparator is added to the divided input voltage when an ADC conversion is done with the switches in position "A"; The offset voltage of the comparator is subtracted from the divided input voltage when an ADC conversion is done with the switches in position "B". Adding both results together gives a 9-bit conversion value that is offset compensated.

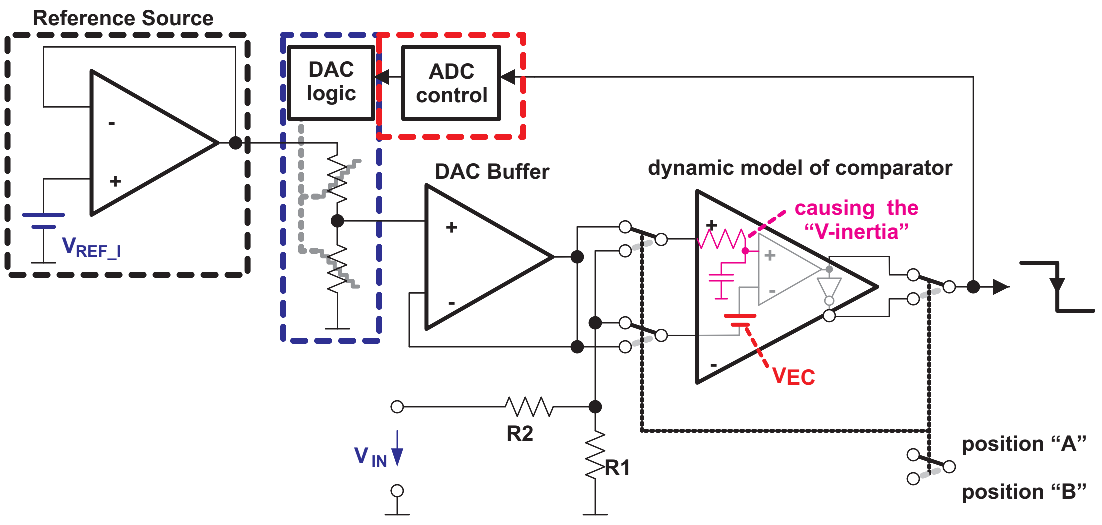


Figure 7-10. Comparator Offset

Example 7-3. ADC Conversion With Offset Compensation

```
.
...
;A-POOL Interrupt handler for simple ADC conversions
APHNDL:   MOV.W   #0,&APIV        ;clear all pending interrupts
          BIC.W   #CPUOFF,0(SP)   ;exit LPM0 and return
          RETI
...
.
...
;A-POOL SW triggered offset comp. ADC conversion for 9 bit integer
;.. on channel A0 with int. Vref, .5V range, low power, event driven
OCADC9:   MOV.B   #1,&APIE        ;enable end of conversion interrupt
          MOV.W   #21E0h,&APCNF   ;enable required anal. elements
          BIS.B   #A0DIV,&APVDIV  ;set the voltage divider to 500 mV
          MOV.W   #0,&APINT       ;clear ADC-DAC-REG
          MOV.W   #5504h,&APCTL   ;set channels and start conversion
          BIS.W   #CPUOFF,SR      ;enter LPM0 and wait
          MOV.W   &APINT,R12      ;handover int. value in R12
          MOV.W   #0,&APINT       ;clear ADC-DAC-REG
          MOV.W   #0556h,&APCTL   ;cross channels and start conversion
          BIS.W   #CPUOFF,SR      ;enter LPM0 and wait
          ADD.W   &APINT,R12      ;compose 9 bit result in R12
          RET             ;return to calling function
```

7.5.5 Evaluation of DAC Buffer Offset

The DAC output buffer shows a small offset voltage V_{EO} that can be compensated again when compared against the comparator's offset voltage V_{EC}.

7.5.6 ADC Conversions for Measuring

For measuring purposes, compensation of all error sources might be required. Inertia compensation and offset compensation can be done as separate measurements, and all results are then added to get a 10-bit value (then averaged to 8-bit resolution). In the example below, concatenated conversions are done to save time and energy. V_{EO} that has been evaluated before (typically 5 to 10 counts) is used to correct the values after measurement.

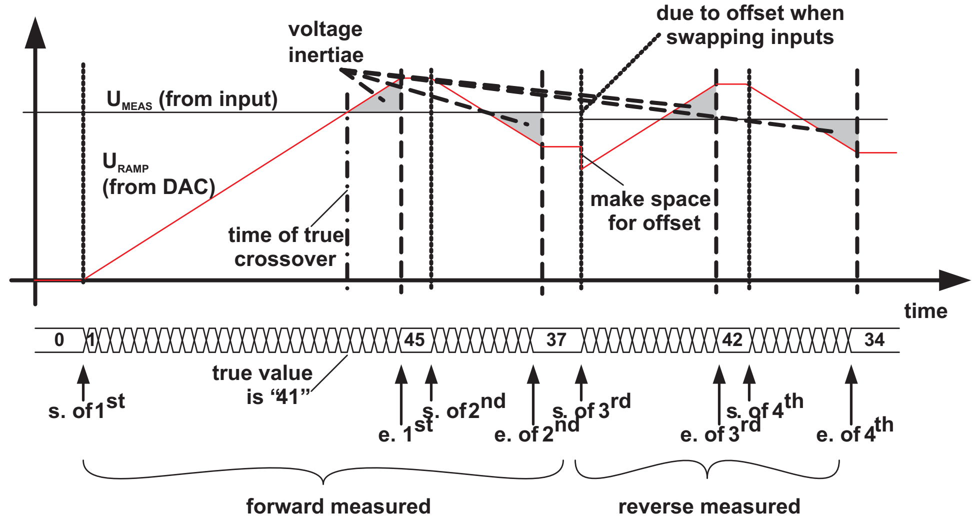


Figure 7-11. ADC Conversion With Overall Compensation

Example 7-4. ADC Conversion With Overall Compensation

```
.
...
;A-POOL Interrupt handler for simple ADC conversions
APHNDL:    MOV.W   #0,&APIV      ;clear all pending interrupts
           BIC.W   #CPUOFF,0(SP)  ;exit LPM0 and return
           RETI
...
.
...
;A-POOL SW triggered overall comp. ADC conversion for 10 bit integer
;.. on channel A0 with int. Vref, .5V range, low power, event driven
IOCADC10:  MOV.B   #1,&APIE               ;enable end of conversion interrupt
           MOV.W   #21E0h,&APCNF          ;enable required anal. elements
           BIS.B   #A0DIV,&APVDIV         ;set the voltage divider to 500 mV
           MOV.W   #0,&APINT              ;clear ADC-DAC-REG
           MOV.W   #5504h,&APCTL          ;set channels and start conversion
           BIS.W   #CPUOFF,SR             ;enter LPM0 and wait
           MOV.W   &APINT,R12             ;compose final value in R12
           BIS.W   #RUNSTOP+SLOPE,&APCTL  ;set down & start conversion
           BIS.W   #CPUOFF,SR             ;enter LPM0 and wait
           ADD.W   &APINT,R12             ;compose final value in R12
           CMP.W   #15,&APINT             ;check if range save for offset space
           JL      IOCADCa                ;no space (lower 15 mV are not comp.)
           SUB.W   #15,&APINT             ;make space for offset
IOCADCa:   MOV.W   #0556h,&APCTL          ;set channels and start conversion
           BIS.W   #CPUOFF,SR             ;enter LPM0 and wait
           ADD.W   &APINT,R12             ;compose final value in R12
           BIS.W   #RUNSTOP+SLOPE,&APCTL  ;set down and start conversion
           BIS.W   #CPUOFF,SR             ;enter LPM0 and wait
           ADD.W   &APINT,R12             ;compose final value in R12
           SUB.W   &VEO,R12               ;subtract pre-eval VEO
           RET                            ;return to calling function
```

7.5.7 Windowed ADC Conversions

Input voltages close to the upper range end on an up-ramp measurement are slow as usually the ramp voltage starts at zero. If the expected input voltage range can be estimated, then the ADC-DAC-REG is initialized with the lower range value. This speeds up the conversion.

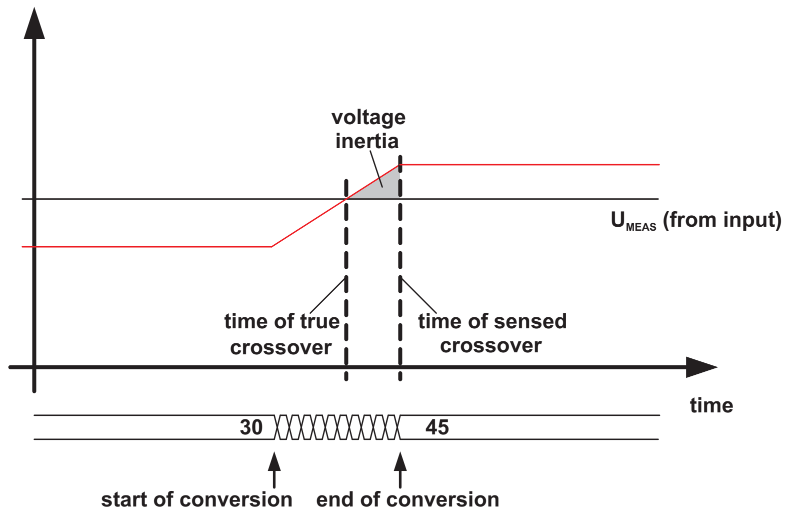


Figure 7-12. Windowed ADC Conversion

Regular ADC conversions terminate when the comparator detects the crossover of ramp voltage and input voltage. Input voltages close to the upper or lower range end might not terminate properly. Using the saturation logic as second termination mechanism resolves those situations.

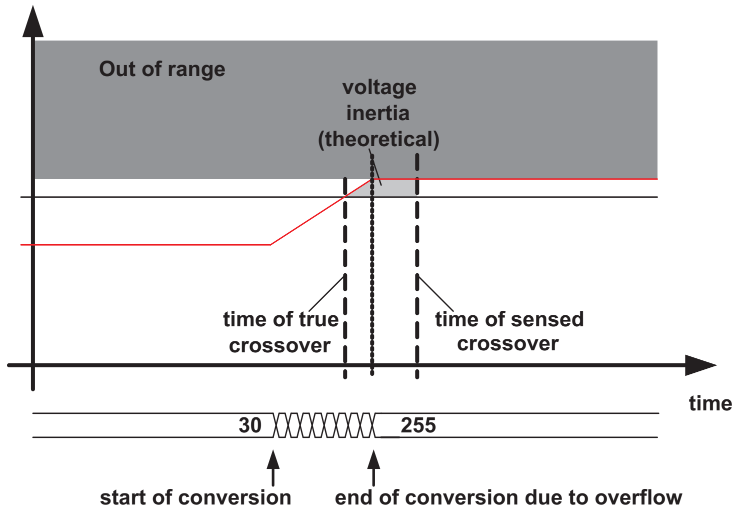


Figure 7-13. ADC Conversions Range Terminated

7.5.8 Full Analog Signal Chain Setup With Interleaved SVM Operations

A-POOL is flexible enough to build up a full signal chain with ADC, DAC, and observing the supply voltage in background while the main program runs the user application. The following example shows one possible solution.

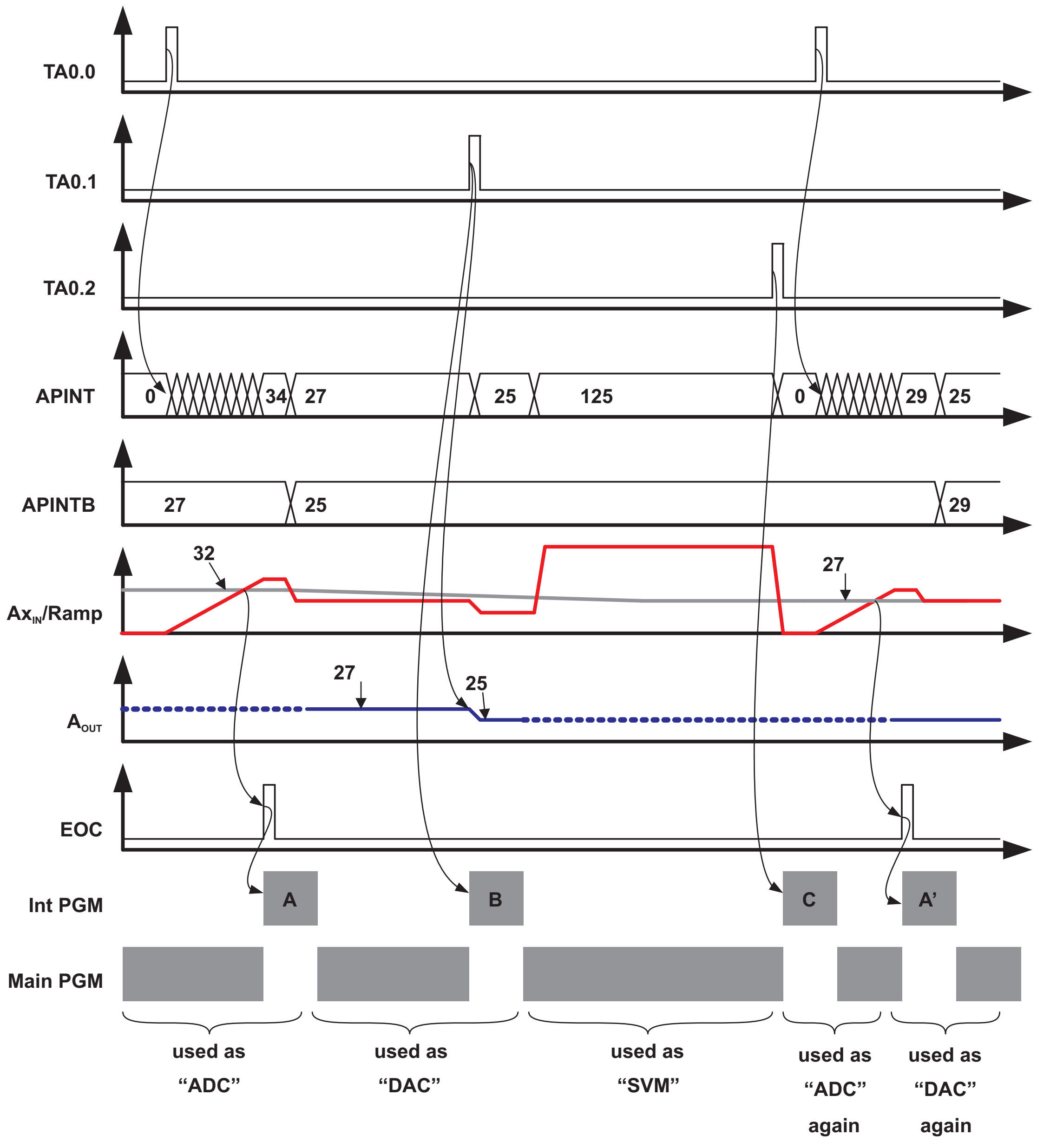


Figure 7-14. Full Signal-Chain Timing Diagram

The tasks of the various interrupt service routines for example in Figure 7-14:

Task A

- Pick up the ADC conversion value from APINT; the value is 34 instead of the true 32 due to offset.
- Put the old DAC value location in APINT
- Set ODEN (turn on AOUT)
- Put the new DAC value in APINTB
- Copy new DAC value into old DAC value location

Task B

- Prepare multiplexer for SVM task, Clear SVMIFG
- Clear ODEN (make AOUT high impedance; VOUT is kept by external capacitor)
- Set SVM level and SVMIE

Task C

- Clear SVMIE
- Prepare multiplexer for ADC measurement
- Clear APINT

7.5.9 Multiple ADC Channels

A-POOL is able to make measurements on multiple analog inputs. In Figure 7-15 the buffered conversion value is picked up from APINTB (when EOCBU = 1).

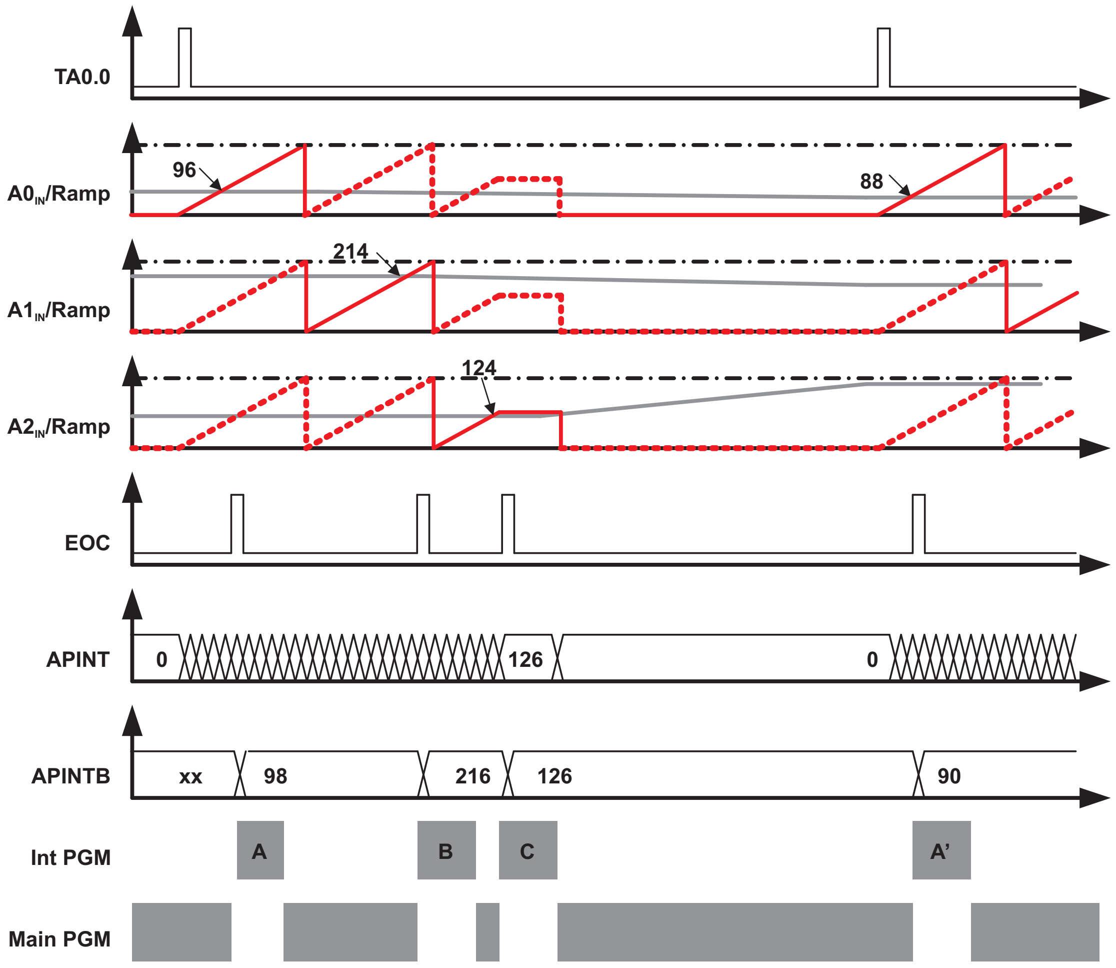


Figure 7-15. Multichannel ADC Conversion

The tasks of the various interrupt service routines for example in Figure 7-15:

Task A

- Pick up the ADC conversion value for "channel 0" from APINTB; the value is 98 instead of the true 96 due to offset.
- Change multiplexers to "channel 1"

Task B

- Pick up the ADC conversion value for "channel 1" from APINTB; the value is 216 instead of the true 214 due to offset.
- Change multiplexers to "channel 2"
- Turn on CBSTP to stop counter on next comparator match

Task C

- Pick up the ADC conversion value for "channel 2" from APINTB; the value is 126 instead of the true 124 due to offset.
- Change multiplexers to "channel 0"
- Turn off CBSTP to enable multi channels again

7.6 A-POOL Control Registers

The control registers of A-POOL are listed in Table 7-8. The base address for the A-POOL registers is listed in Table 7-7.

Table 7-7. A-POOL Base Address

Module	Base address
A-POOL	001A0h

Table 7-8. A-POOL Control Registers

Register	Short Form	Register Type	Register Access	Address Offset	Initial State
A-POOL configuration register	APCNF	read/write	word	00h	2000h
	APCNF_L		byte	00h	00h
	APCNF_H		byte	01h	20h
A-POOL control register	APCTL	read/write	word	02h	F0F0h
	APCTL_L		byte	02h	F0h
	APCTL_H		byte	03h	F0h
A-POOL operation mode register	APOMR	read/write	word	04h	0000h
	APOMR_L		byte	04h	00h
	APOMR_H		byte	05h	00h
A-POOL voltage divider register	APVDIV	read/write	word	06h	0000h
	APDIV_L		byte	06h	00h
	APDIV_H		byte	07h	00h
A-POOL trimming register	APTRIM	read/write	word	08h	xx00h
	APTRIM_L		byte	08h	00h
	APTRIM_H		byte	09h	xxh
A-POOL integer conversion register	APINT	read/write	word	10h	0000h
	APINT_L		byte	10h	00h
	APINT_H		byte	11h	00h
A-POOL integer conversion buffer	APINTB	read/write	word	12h	0000h
	APINTB_L		byte	12h	00h
	APINTB_H		byte	13h	00h
A-POOL fractional conversion register	APFRACT	read/write	word	14h	8000h
	APFRACT_L		byte	14h	00h
	APFRACT_H		byte	15h	80h
A-POOL fractional conversion buffer	APFRACTB	read/write	word	16h	8000h
	APFRACTB_L		byte	16h	00h
	APFRACTB_H		byte	17h	80h
A-POOL interrupt flag register	APIFG	read/write	word	1Ah	0000h
	APIFG_L		byte	1Ah	00h
	APIFG_H		byte	1Bh	00h
A-POOL interrupt enable register	APIE	read/write	word	1Ch	0000h
	APIE_L		byte	1Ch	00h
	APIE_H		byte	1Dh	00h
A-POOL interrupt vector register	APIV	read/write	word	1Eh	0000h
	APIV_L		byte	1Eh	00h
	APIV_H		byte	1Fh	00h
Interrupt flag 1	IFG1	read/write	word/byte		
Interrupt enable 1	IE1	read/write	word/byte		

APCNF, A-POOL Configuration Register

15 7	14 6	13 5	12 4	11 3	10 2	9 1	8 0
Reserved	**VREFEN**	**REFON**	**A3PSEL**	**ATBU**	**EOCBU**	**CLKSELx**	
r0	rw-0	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
CONVON	**DBON**	**CMPON**	**Reserved**	**DFSETx**		**TA1EN**	**TA0EN**
rw-0	rw-0	rw-0	r0	rw-0	rw-0	rw-0	rw-0

Reserved	Bit 15	Reserved. Reads back as 0.	
VREFEN	Bit 14	Reference voltage pin enable	
		0	VREF terminal disabled
		1	VREF terminal enabled
REFON	Bit 13	Internal voltage reference enable	
		0	Internal reference is off
		1	Internal reference is on
A3PSEL	Bit 12	This bit determines which pin analog input A3 is accessed with.	
		0	A3 is taken from the pin that is able to drive AOUT
		1	
ATBU	Bit 11	Automatic update of conversion register from conversion buffer on TA0.1 event enable bit	
		0	Disabled
		1	Enabled
EOCBU	Bit 10	Enable bit for loading conversion buffer from conversion register on “end of conversion” (EOC)	
		0	Disabled
		1	Enabled
CLKSELx	Bits 9-8	Conversion clock (A-POOL master clock) select	
		00	VLOCLK
		01	MCLK
		10	SMCLK
		11	Reserved, defaults to VLOCLK
CONVON	Bit 7	Enable for converter’s resistor ladder	
		0	Resistor ladder is off
		1	Resistor ladder is on
DBON	Bit 6	DAC buffer enable signal	
		0	DAC buffer is off
		1	DAC buffer is on
CMPON	Bit 5	Comparator enable	
		0	Comparator off
		1	Comparator on
Reserved	Bit 4	Reserved. Reads back as 0.	
DFSETx	Bits 3-2	Deglitching filter setting	
		00	No filtering (straight signal from comparator)
		01	Two of two
		10	Two of three
		11	Three of five
TA1EN	Bit 1	Timer_A1 trigger enable	
		0	Disabled
		1	Enabled
TA0EN	Bit 0	Timer_A0 trigger enable	
		0	Disabled
		1	Enabled

APCTL, A-POOL Control Register

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
PSELx				**TBSTP**	**CBSTP**	**SBSTP**	**RUNSTOP**
rw-1	rw-1	rw-1	rw-1	rw-0	rw-0	rw-0	rw-0

7	6	5	4	3	2	1	0
NSELx				**SLOPE**	**OSEL**	**OSWP**	**ODEN**
rw-1	rw-1	rw-1	rw-1	rw-0	rw-0	rw-0	rw-0

PSELx	Bits 15-12	Reference input select. These control bits select the source for the noninverting input of the comparator	
		0000	Analog input A0 is selected
		0001	Analog input A1 is selected
		0010	Analog input A2 is selected
		0011	Analog input A3 is selected
		0100	Temperature sensor is selected
		0101	DAC buffer output is selected
		0110	Supply voltage divider (2 / 1.8-V range)
		0111	Voltage reference is selected
		1000	224-mV reference tap is selected
		1001	Supply voltage divider for (1 / 0.9-V range)
		⋮	Reserved; defaults to ground
		1111	No input signal is selected (multiplexer open)
TBSTP	Bit 11	Timer based conversion stop enable for Timer_A0	
		0	Disabled
		1	Enabled
CBSTP	Bit 10	Comparator based conversion stop enable	
		0	Disabled
		1	Enabled
SBSTP	Bit 9	Saturation based conversion stop enable	
		0	Disabled
		1	Enabled
RUNSTOP	Bit 8	Converter run/stop. This bit can be changed to force the desired state and can be read to check its state.	
		0	Stopped
		1	Running
NSELx	Bits 7-4	Analog input select. These control bits select the source for the inverting input of the comparator	
		0000	Analog input A0 is selected
		0001	Analog input A1 is selected
		0010	Analog input A2 is selected
		0011	Analog input A3 is selected
		0100	Temperature sensor is selected
		0101	DAC buffer output is selected
		0110	Supply voltage divider is selected
		0111	Voltage reference is selected
		⋮	Reserved; defaults to ground
		1111	No input signal is selected (multiplexer open)
SLOPE	Bit 3	Slop select of converter	
		0	Rising slope (counting up)
		1	Falling slope (counting down)

(continued)

OSEL	Bit 2	Output buffer select
		0 Analog input MUX is selected to drive AOUT
		1 DAC is selected to drive AOUT
OSWP	Bit 1	Output swap
		0 Straight comparator output is used
		1 Inverted comparator output is used
ODEN	Bit 0	Output driver enable
		0 AOUT is disabled
		1 AOUT is enabled

APVDIV, A-POOL Voltage Divider Control Register

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

7	6	5	4	3	2	1	0
VCCDIVEN	**TMPSEN**	**A3DIVx**		**A2DIVx**		**A1DIV**	**A0DIV**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Reserved	Bits 15-10	Reserved. Reads back 0.
BUSY	Bit 9	Busy flag indicates that internal state machine is performing a conversion such as SAR or Up/Down ramp. The Busy flag is cleared when its operation has stopped and is ready for new operations.
COMPOUT	Bit 8	Reads back the state of the comparator after the deglitching filter
VCCDIVEN	Bit 7	V_{CC} voltage divider enable
		0 V_{CC} voltage divider off
		1 V_{CC} voltage divider on
TMPSEN	Bit 6	Temperature sensor enable
		0 Temperature sensor off
		1 Temperature sensor on
A3DIVx	Bits 5-4	Analog channel 3 voltage divider control
		00 250-mV input voltage range
		01 1-V input voltage range
		10 2-V input voltage range
		11 Reserved
A2DIVx	Bits 3-2	Analog channel 2 voltage divider control
		00 250-mV input voltage range
		01 1-V input voltage range
		10 2-V input voltage range
		11 Reserved
A1DIV	Bit 1	Analog channel 1 voltage divider control
		0 250-mV input voltage range
		1 500-mV input voltage range
A0DIV	Bit 0	Analog channel 0 voltage divider control
		0 250-mV input voltage range
		1 500-mV input voltage range

APTRIM, A-POOL Trimming Register

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
REFTRIM_PTATx				**REFTRIM_GAINx**			**Reserved**
r	r	r	r	r0	r0	r0	r0
▨ rw-[1]	▨ rw-[0]	▨ rw-[0]	▨ rw-[0]				

7	6	5	4	3	2	1	0
Reserved							**REFTSEL**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-[0]

▨ Condition when REFTSEL = 1

REFTRIM_PTATx	Bits 15-12	Reference trimming bits for fine calibrating the reference voltage source; PTAT part
		If REFTSEL = 0, read reflects the calibration value of the fuse element, write has no effect
		If REFTSEL = 1, read returns the value of the calibration register, write values with bit 9 set to one will cause a write into the calibration register.
REFTRIM_GAINx	Bits 11-9	Reference trimming bits for fine calibrating the reference voltage source; gain part
		If REFTSEL = 0, read reflects the calibration value of the fuse element, write has no effect
		If REFTSEL = 1, read returns the value of the calibration register, write values with bit 9 set to one will cause a write into the calibration register.
Reserved	Bits 8-1	Reserved. Reads back 0.
REFTSEL	Bit 0	Selects between the register bank used for the reference trimming
		0 Fuse elements are selected (write has no effect)
		1 Trimming register is selected

APOMR, A-POOL Operation Mode Register

15	14	13	12	11	10	9	8
7	6	5	4	3	2	1	0
Reserved				**SVMINH**	**AZSWREQ**	**AZCMP**	**CTEN**
r0	r0	r0	r0	rw-[0]	rw-[0]	rw-[0]	rw-[0]

7	6	5	4	3	2	1	0
Reserved				**SAREN**	**CLKDIVx**		
r0	r0	r0	r0	rw-[0]	rw-[0]	rw-[0]	rw-[0]

Reserved	Bits 15-12	Reserved. Reads back as 0.
SVMINH	Bit 11	This bit suppresses the generation of an SVM interrupt event. This is recommended to be use before swapping the input/output channels of A-POOL to avoid generating erroneous SVM interrupts.
AZSWREQ	Bit 10	Software request for auto-zero phase
AZCMP	Bit 9	Set comparator to clocked zero compensated long term comparison
CTEN	Bit 8	Continuous time mode of comparator

CTEN	AZCMP	AZSWREQ	Mode of Comparator
0	0	0	Auto-zero mode for requesting units (e.g., SAR)
0	X	1	Comparator forced to AZ phase
0	1	0	Comparator put in long-term clocked AZ comparison
1	X	X	Continuous time mode for all operations

Reserved	Bits 7-4	Reserved. Reads back as 0.	
SAREN	Bit 3	SAR conversion enable (instead of ramp generation)	
		0	Disabled
		1	Enabled
CLKDIVx	Bit 2-0	Prescaler control for ADC/DAC, digital filter, AZ comparator modes	
		000	Division ratio = 1
		001	Division ratio = 2
		010	Division ratio = 4
		011	Division ratio = 8
		100	Division ratio = 16
		101	Division ratio = 32
		⋮	Reserved; defaults to Division ratio = 1

APINT, A-POOL Integer Conversion Value Register

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
INTVAL							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Reserved	Bits 15-8	Reserved. Reads back as 0.
INTVAL	Bits 7-0	Conversion value in unsigned integer format. The value ranges from 0h to FFh when read or written as byte and from 0h to FF00h when read or written as word

APINTB, A-POOL Integer Conversion Value Buffer

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
INTVAL							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Reserved	Bits 15-8	Reserved. Reads back as 0.
INTVAL	Bits 7-0	Buffer value in unsigned integer format. The value ranges from 0h to FFh when read or written as byte and from 0h to FF00h when read or written as word

APFRACT, A-POOL Fractional Conversion Value Register

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

7	6	5	4	3	2	1	0
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0

FRACTVAL	Bits 15-8	Conversion value in signed fractional format. The value ranges from 80h to 7Fh when read or written as byte and from 8000h to 7F00h when read or written as word
Reserved	Bits 7-0	Reserved. Reads back as 0.

APFRACTB, A-POOL Fractional Conversion Value Buffer

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

7	6	5	4	3	2	1	0
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0

FRACTVAL	Bits 15-8	Buffer value in signed fractional format. The value ranges from 80h to 7Fh when read or written as byte and from 8000h to 7F00h when read or written as word
Reserved	Bits 7-0	Reserved. Reads back as 0.

APIFG, A-Pool Interrupt Flag Register

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				**REFOKIFG**	**CRIFG**	**CFIFG**	**EOCIFG**
r0	r0	r0	r0	rw-0	rw-0	rw-0	rw-0

Reserved	Bits 15-4	Reserved. Reads back as 0.
REFOKIFG	Bit 3	Reference voltage ready interrupt flag 0 No interrupt pending 1 Interrupt pending
CRIFG	Bit 2	Comparator rising edge interrupt flag 0 No interrupt pending 1 Interrupt pending
CFIFG	Bit 1	Comparator falling edge interrupt flag 0 No interrupt pending 1 Interrupt pending
EOCIFG	Bit 0	End of conversion interrupt flag 0 No interrupt pending 1 Interrupt pending

APIE, A-Pool Interrupt Enable Register

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				**REFOKIE**	**CRIE**	**CFIE**	**EOCIE**
r0	r0	r0	r0	rw-0	rw-0	rw-0	rw-0

Reserved	Bits 15-4	Reserved. Reads back as 0.
REFOKIE	Bit 3	Reference voltage ready interrupt enable 0 Interrupt disabled 1 Interrupt enabled
CRIE	Bit 2	Comparator rising edge interrupt enable 0 Interrupt disabled 1 Interrupt enabled
CFIE	Bit 1	Comparator falling edge interrupt enable 0 Interrupt disabled 1 Interrupt enabled
EOCIE	Bit 0	End of conversion interrupt enable 0 Interrupt disabled 1 Interrupt enabled

APIV, A-Pool Interrupt Vector Register

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

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**		**APIVx**		**0**
r0	r0	r0	r0	r-0	r-0	r-0	r0

APIVx Bits 3-1 Analog Pool Interrupt vector value. It generates a value that can be used as address offset for fast interrupt service routine handling. Writing to this register clears all pending interrupt flags. Reading this register clears the highest pending interrupt flag (displaying this register with the debugger does not affect its content).

APIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending		
02h	End of conversion (EOC) interrupt	EOCIFG	Highest
04h	Comparator falling edge interrupt	CFIFG	⋮
06h	Comparator rising edge interrupt	CRIFG	⋮
08h	Reference OK interrupt	REFOKIFG	Lowest

IFG1, Interrupt Flag Register 1

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

SVMIFG Bit 8 SVM interrupt flag. This bit signals that the A-POOL comparator signaled an SVM event either low voltage or high voltage depending on setup.

0	No interrupt pending
1	Interrupt pending

IE1, Interrupt Enable Register 1

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

SVMIE Bit 8 SVM interrupt enable flag

0	Interrupts disabled
1	Interrupts enabled

MSP430L092 Loader Code (Quick Start)

The MSP430L092 device is a development and prototyping and small series family member of the MSP430x09x device family. It contains a special loader code stored in its internal ROM memory. This chapter describes how the MSP430L092 loader code is used to build an autonomous microcontroller solution. The loader approach is chosen as nonvolatile memory is not available for native ultra-low supply voltages. For detailed information on the loader code, see the *MSP430L092 Loader Code User's Guide* (SLAU324).

Topic **Page**

8.1 Loader Code Introduction

The loader code in the MSP430L092 is TI-provided ROM firmware that provides a series of services. It enables customers to build autonomous applications without the need for a custom ROM mask. Such an application consists of an MSP430 device containing the loader (for example, MSP430L092) and a SPI memory device (for example, '95512 or '25AA40). These and similar memory devices are available from various manufacturers.

The major use cases for an application with a loader device and external SPI memory for native 0.9-V supply voltage are late development, prototyping, and small series production. Table 8-1 and Figure 8-1 list various debugging scenarios possible for ultra-low supply voltage. A loader approach is the only choice for an autonomous application with MSP430L092, as no nonvolatile memories are available on the market for native ultra-low supply voltages.

Table 8-1. Debugging Scenarios With MSP430x09x Devices

Use Case	Early Development	Late Development	Prototyping	Small Series	Mass Production
No. of Units	up to 10	up to 100	up to 1000	up to 100000	100000+
Device	MSP430L092	MSP430L092	MSP430L092	MSP430L092	MSP430C091/C092
Cost	High	Medium	Medium	Medium	Low
Code in ...	IDE / RAM	External Memory / RAM	External Memory / RAM	External Memory / RAM	External Memory / ROM / RAM
Galv. Sep.	No	Yes	Yes	Yes	Yes
Code Size	up to 1984B (typical)	up to 1984B (typical)	up to 1984B (typical)	up to 1984B (typical)	up to 1984B (typical)
RAM Size	up to 64B (typical)	up to 64B (typical)	up to 64B (typical)	up to 64B (typical)	up to 1024/2048B (typical)
Overlays	Supported by 'L092	Supported	Supported	Supported	Depends on customer code

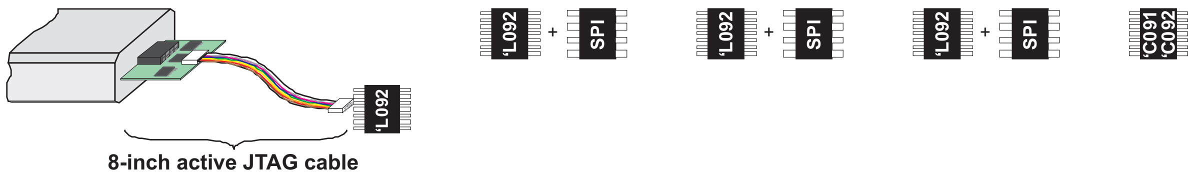


Figure 8-1. Debugging Scenarios With MSP430x09x Devices

The user can determine the type of SPI memory device used together with an MSP430 device with loader code. SPI-EEPROMs, SPI-Flash, SPI-SRAM, SPI-FRAM, and SPI-byte alterable flash devices with supply voltages range from 1.8 V to 6 V and various memory sizes have been seen on the market.

8.1.1 Typical Two-Chip Application

An application with the MSP430L092 device can be as simple as shown in Figure 8-2. The loader code initializes the MSP430 device and generates an external clock on port P1.2 that allows an external boost converter generating the necessary supply voltage for the SPI device containing the user program. After approximately 500 µs, the loader starts to load the user code into the 'L092 RAM memory. After a successful load procedure, the user code is started. During the code loading process, the LED used to stabilize the voltage for the SPI device lights up briefly. The LED may be used later for regular signaling purposes; the SPI device is then kept inactive.

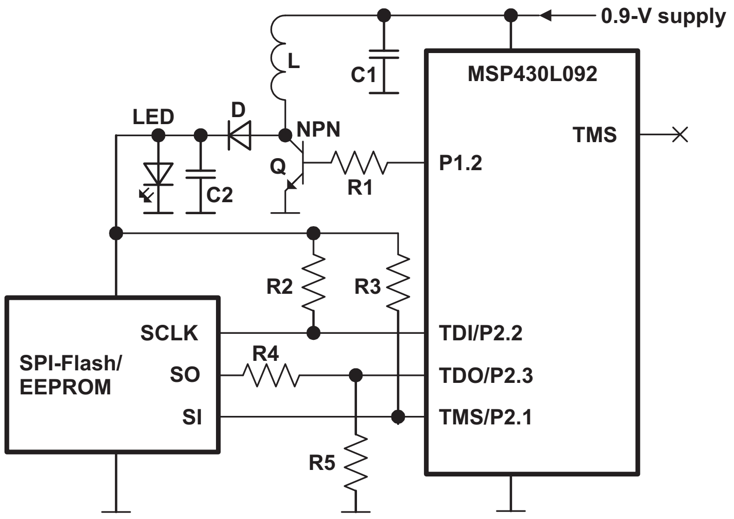


Figure 8-2. Component Optimized Application Circuit for 0.9-V Supply

8.1.2 Code Generation, Conventions, and Restrictions

The application code is generated with the standard tools for MSP430. The user application may (but is not required to) use other services provided by the loader API. If the API's services are used, then special conventions must be followed; otherwise, the user code can be written without any restrictions.

8.1.3 Start-Up Behavior and Timing

Immediately after startup, devices with the loader code behave like devices with any other user code. After V_{CC} ramp-up or reset release, control is given to the start-up code (SUC). This code performs initialization and verifies device integrity. It then passes control to the code in ROM by branching to the location the ROM-Code start vector is pointing at, in this case, the loader code. The loader performs its initialization and turns on a 250-kHz PWM signal on port P1.2. Approximately 500 µs later, the user application code residing in the external SPI memory is loaded into the internal MSP430 RAM. When stored in the external SPI memory, the application code is embedded inside a data container that is protected with checksums. During the loading process, the checksum is verified. If the checksum is correct, the loader code passes control to the application code loaded in RAM.

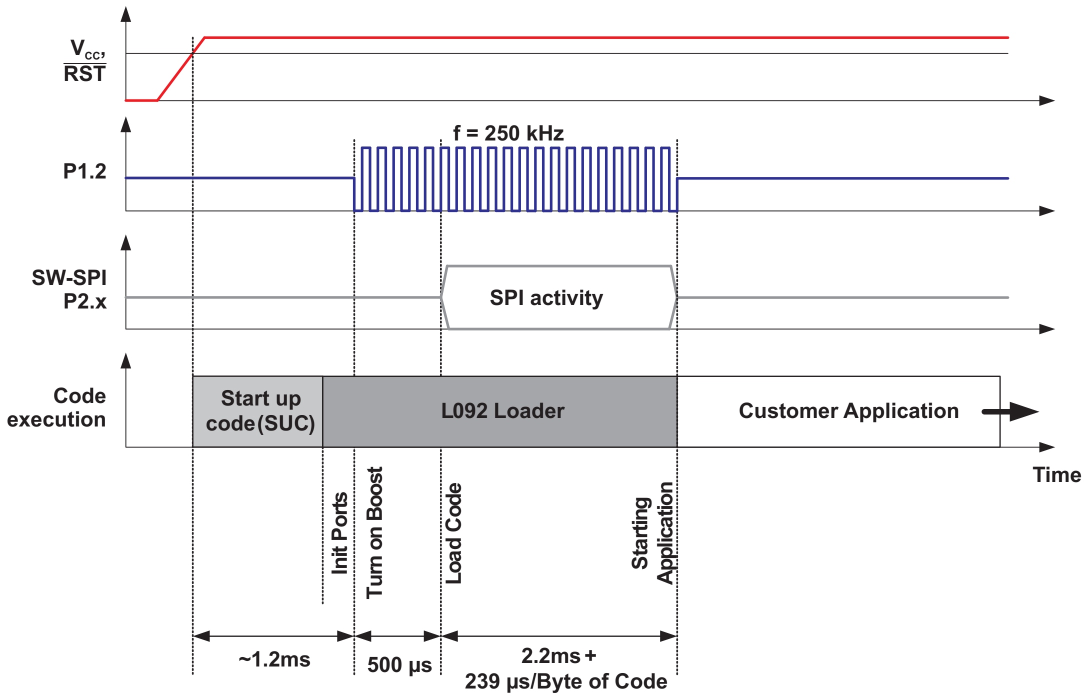


Figure 8-3. Timing of Successful Load Operation

NOTE: The ROM-code start vector is located at 0xF840 for the MSP430x09x devices, and it is a reduced length 16-bit address pointer (in this example, it points to the start of the loader).

8.1.4 Failsafe Mechanism

If no SPI device is connected, or the SPI power is not generated, or an error during user code load is detected, a visible error signature is generated. The voltage stabilization LED blinks three times with a frequency of approximately 1 Hz. The user's application code is not executed, and the device enters LPM4 state. This prevents the device from executing erroneous code.

8.1.5 Data Structure of the SPI Memory

The application code is kept in external SPI memory when using the loader approach. One-bit wide SPI devices with 16-bit and 24-bit address range are supported. At location 0x0, a format indication is expected for both address types. The loader code automatically adapts its SPI address width to the identified SPI memory device size by checking for the format indicator at memory location at address 0x0 and 0x1. The first boot data/program container is expected at address 0x2. Other data/program containers may be stored anywhere in the SPI memory. Loading data/program containers from SPI addresses below 0x800 automatically causes an LED turn off operation and a password check with stop for debugging purposes. Data/program containers loaded from SPI addresses ≥ 0x800 do not cause an automated LED-off operation; this area is typically used for overlay programming.

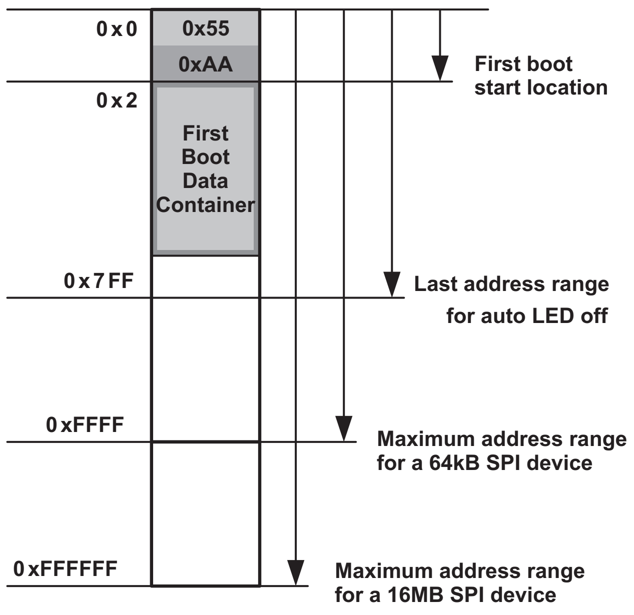


Figure 8-4. Data Structures in SPI Memory

8.1.6 Data/Program Containers

A data or program container is a structure stored in SPI memory that contains data or program (code) elements as payload (see Figure 8-5).

The header of the data container consists of the 16-bit length field, a 16-bit destination address where the code is supposed to be loaded to, and a 16-bit start address field that is invoked after code load.

The length field represents the size of the payload in bytes. The payload itself is always of even length. Zero padding at the end of payload is used if the length field contains an odd value. The theoretical maximum block length is 65536 bytes.

The load address points to the MSP430 memory location the payload is loaded to (when not overridden). This is between 0x0 and 0xFFFF.

The start address points to the start of code when loaded into MSP430 memory in the case of the first bootable data/program container for proper operation. For all the other containers loaded later, it may point outside the loaded destination address.

The trailer of the container provides two copies of the checksum that is built over header and payload. The checksum is calculated using a word-wide XOR operation initialized with zero.

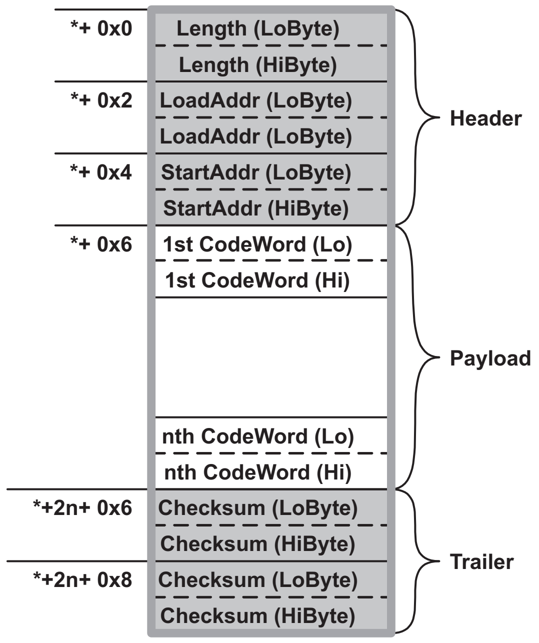


Figure 8-5. Data/Program Container

8.1.7 Interrupt Handling

The loader should allow an application program to use all interrupt resources of a device. A hardware interrupt causes the interrupt service routine to be called from the location that the corresponding interrupt vector is pointing at. In the case of the loader, a simple instruction (SW-stub) is placed there to forward the control to the interrupt service routine that the secondary interrupt vector is pointing at. The secondary interrupt vector is a software element in RAM that points to the user's interrupt service routine (see Figure 8-6). Such SW-stubs and secondary interrupt vectors are implemented for all interrupt sources. The secondary interrupt vectors are initialized to point to a dummy interrupt handler. This ensures that all interrupts, even unexpected ones, are terminated before the user code takes control. The dummy interrupt handler counts the number of unexpected interrupts.

Slight differences are to be expected between the 'L092 behavior and the 'C091/'C092 behavior.

- The interrupt response of the 'L092 takes four cycles longer due to the deviation by the SW-stub.
- Unexpected interrupts are always terminated on the 'L092. If the user code does not manage all interrupts, the 'L092 loader terminates them. The same user code on the 'C091/C092, however, is defenseless against unexpected interrupts.

NOTE: It is strongly recommended to terminate all interrupt vectors.

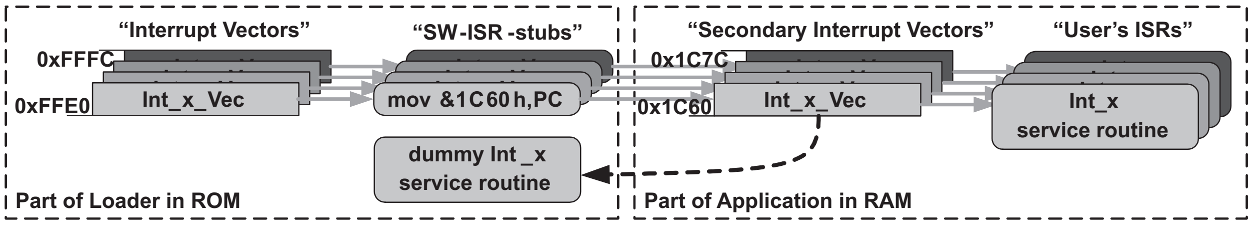


Figure 8-6. Secondary Interrupt Vectors

The secondary interrupt vectors are listed in Table 8-2. The secondary interrupt vectors provide a vector field, similar to the INTVECS section, allowing a dynamic lookup of interrupt handlers used on devices with loader code.

Table 8-2. Secondary Interrupt Vectors

Register	Short Form	Register Type	Register Access	Address (in 'L092)	Initial State
Interrupt 0xF Vector	INT0FIV2	read/write	word	1C60h	F8xx (user-defined)
Interrupt …	INT…	read/write	word	…	…
Interrupt 0x3 Vector	INT03IV2	read/write	word	1C78h	F8xx (user-defined)
User NMI Vector	UNMIV2	read/write	word	1C7Ah	F8xx (user-defined)
System NMI Vector	SNMIV2	read/write	word	1C7Ch	F8xx (user-defined)
Reset Vector	RSTIV2	read/write	word	1C7Eh	F8xx (user-defined)

8.2 Target Hardware

Devices with the loader, like the MSP430L092, require target hardware to operate—Figure 8-2 shown in the introduction is such a target hardware optimized for a particular device. A more generic block diagram for such target hardware is shown in Figure 8-7. It is the user's choice to select one of the proposed SPI-device voltage supply booster circuits and adapter networks, or develop custom circuits. It is also the user's choice to select the type of SPI memory device used.

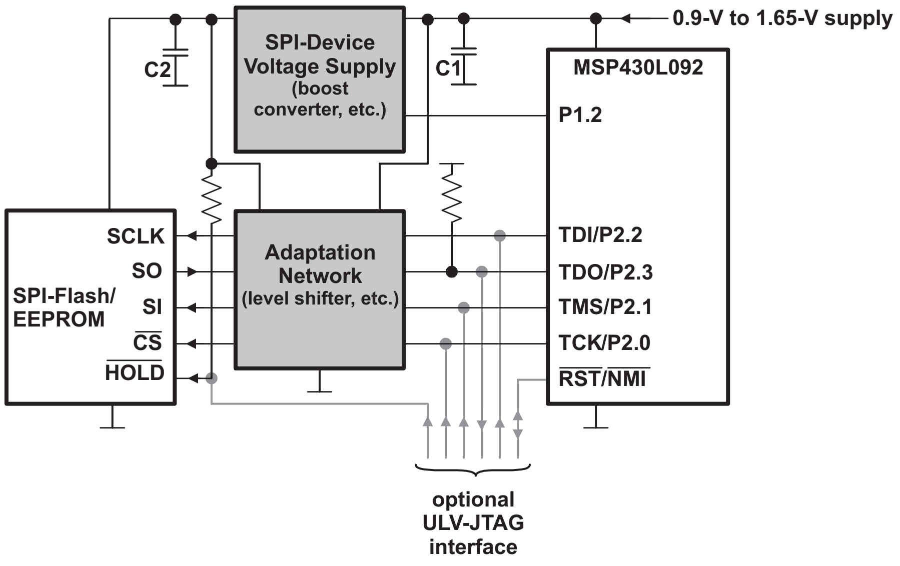

0.9-V to 1.65-V supply
SPI-Device Voltage Supply (boost converter, etc.)
C2
C1
MSP430L092
P1.2
SCLK
SO
SI
CS
HOLD
SPI-Flash/ EEPROM
Adaptation Network (level shifter, etc.)
TDI/P2.2
TDO/P2.3
TMS/P2.1
TCK/P2.0
RST/NMI
optional ULV-JTAG interface

Figure 8-7. Generic Block Diagram of Target Hardware

8.2.1 *Selection of the SPI Devices Supported by the Loader*

Most SPI memory devices share a common command set. Only a common subset of commands is used by the loader software (see Table 8-3). Special device-dependent commands are not used.

Table 8-3. SPI Commands Used by Loader

SPI Command	Code	EEPROMS	Flash	FRAM
Read Status Register	0x05	Used	Used	Used
Write Status Register	0x01	Used	Used	Used
Write Enable	0x06	Used	Used	Used
Read Memory	0x03	Used	Used	Used
Write Memory	0x02	Used	Used	Used
Bulk Erase	0xC7	Ignored	Used	Ignored
Read Stream	0x03, ...	Used	Used	Used

8.2.2 Booster Converters

The circuits in Figure 8 to Figure 13 represent a variety of booster circuits that have been verified and can be used to generate SPI-device supply voltages from 1.9 V to 6 V.

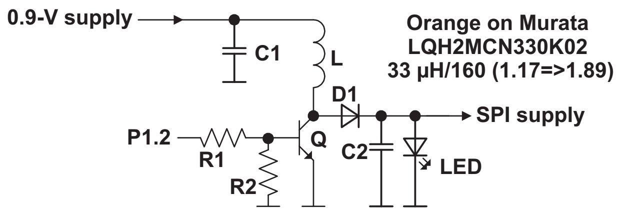


Figure 8. Booster Converter Type A

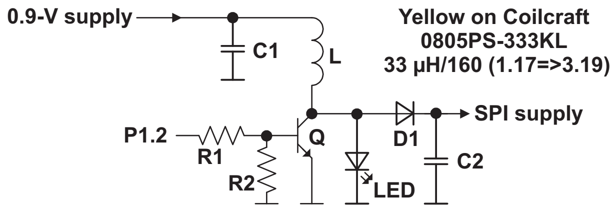


Figure 9. Booster Converter Type B

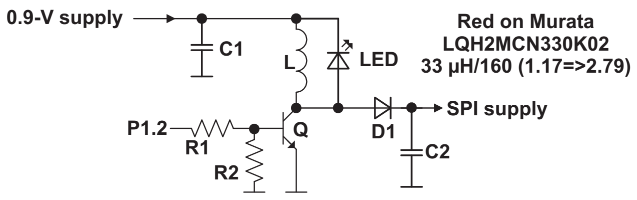


Figure 10. Booster Converter Type C

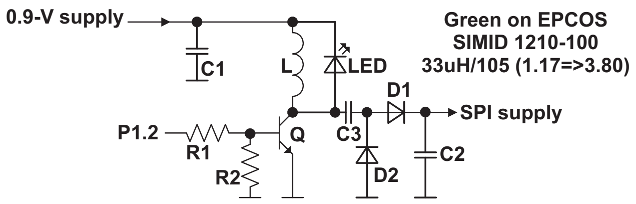


Figure 11. Booster Converter Type D

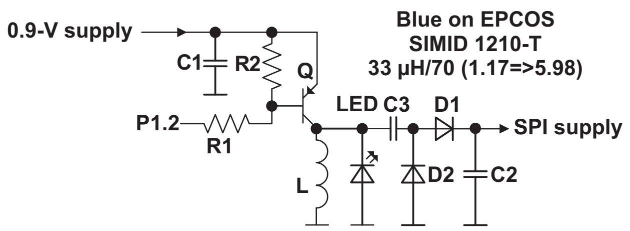


Figure 12. Booster Converter Type E

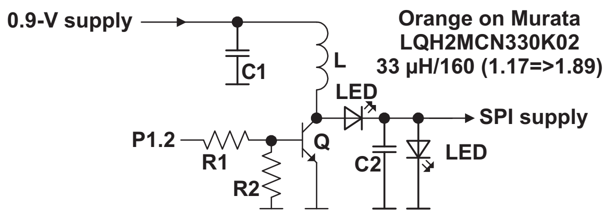


Figure 13. Booster Converter Type F

Table 8-4. Values of Components

Component	Value	Component	Value
R1	1k	D1	1N4148
R2	47k	D2	1N4148
C1	330 nF	L	33 µH / 160 mA
C2	330 nF	Q	BC807 / BC817
C3	10 nF		

8.2.3 Adaptation Networks

The circuits shown in Figure 14 through Figure 17 in represent a variety of adaptation networks circuits suitable for level adaptation for an SPI device being supplied from 1.8 V to 6 V.

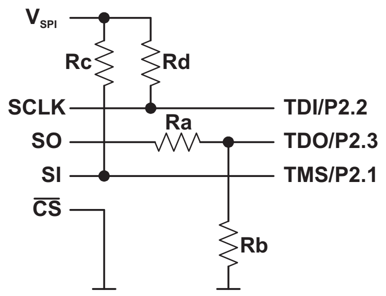


Figure 14. Adaptation Network Type A

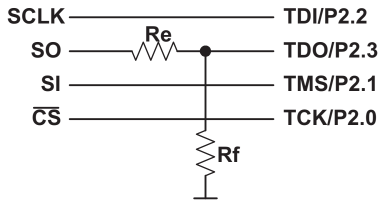


Figure 15. Adaptation Network Type B

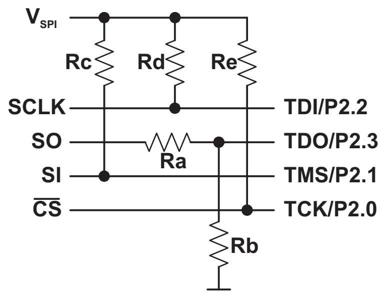


Figure 16. Adaptation Network Type C

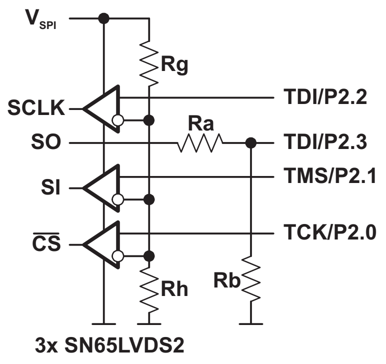


Figure 17. Adaptation Network Type D

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Communications and Telecom	www.ti.com/communications
DSP	dsp.ti.com	Computers and Peripherals	www.ti.com/computers
Clocks and Timers	www.ti.com/clocks	Consumer Electronics	www.ti.com/consumer-apps
Interface	interface.ti.com	Energy	www.ti.com/energy
Logic	logic.ti.com	Industrial	www.ti.com/industrial
Power Mgmt	power.ti.com	Medical	www.ti.com/medical
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video and Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless-apps