# RF430FRL15xH Family

# Technical Reference Manual

TEXAS INSTRUMENTS

Literature Number: SLAU506
December 2014

# Contents

## List of Figures

## List of Tables

*Chapter 1*

*SLAU506–December 2014*

# *System Resets, Interrupts, and Operating Modes, Compact System Control Module (CSYS_A)*

This chapter describes the Compact System Control Module (CSYS_A) which provides start-up functionality and interrupt support functions.

**Topic** **Page**

## 1.1 Compact System Control Module Introduction

The CSYS_A module is responsible for interaction between various modules throughout the system. The functions described in CSYS_A 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_A provides.

The following list shows the basic feature set of CSYS_A:

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

## 1.2 Principle of Operation

The CSYS_A 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_A modules.

### 1.2.1 Device Descriptor Table

Each device 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 Boot Code

The start-up code (SUC) 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 initiate a security BOR during SUC execution. This ensures completion of SUC before entering user code. If no user code exists, Loader Code will be started.

### 1.2.3 Loader Code

The Loader Code allows programming of the device through several interfaces, including JTAG, SPI, $I^2C$, and RF.

### 1.2.4 JTAG Mailbox (JMB) System

The CSYS_A 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 devices of this family and that uses only a 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

Table 1-1 shows a sample memory map. This map represents the RF430FRL15xH device, and other devices may vary.

**Table 1-1. Memory Map**

| | | | Mirrored memory location optional | Generates PUC on fetch access | Protectable against write access | Protectable against read access | May generate NMI on read/write/fetch |
|---|---|---|---|---|---|---|---|
| **Address** | | **Name or Purpose** | **Properties** | | | | |
| 00000h-00FFFh | | Peripherals (with gaps) | | | | | |
| | 00000h-000FFh | Reserved for system extension | | | | | |
| | 00100h-00FEFh | Peripherals | | X | | | |
| | 00FF0h-00FF3h | Descriptor type | | X | | | |
| | 00FF4h-00FF7h | Start address of descriptor structure | | X | | | |
| 01C00h-023FFh | | RAM 4k | | | | | |
| | 01C00h-01C7Fh | Calibration RAM (lockable) | | | X | | |
| | 01C80h-0237Fh | Application memory (lockable) | | | X | | |
| | 02380h-023FFh | Application memory (non lockable) | | | | | |
| | 02380h-023FFh | 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 SYS Interrupt Vector Generators

The CSYS_A module collects all user NMI (UNMI) sources, system NMI (SNMI) sources, and BOR, POR, and 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, or 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 (see Example 1-1). 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.

Example 1-1 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. Example 1-1 is for a generic RF430FRL15xH 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.

**Example 1-1. Recommended Use of SYSSNIV**

```
SNI_ISR:    ADD   &SYSSNIV,PC   ; Add offset to jump table
            RETI                ; Vector 0: No interrupt
            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
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

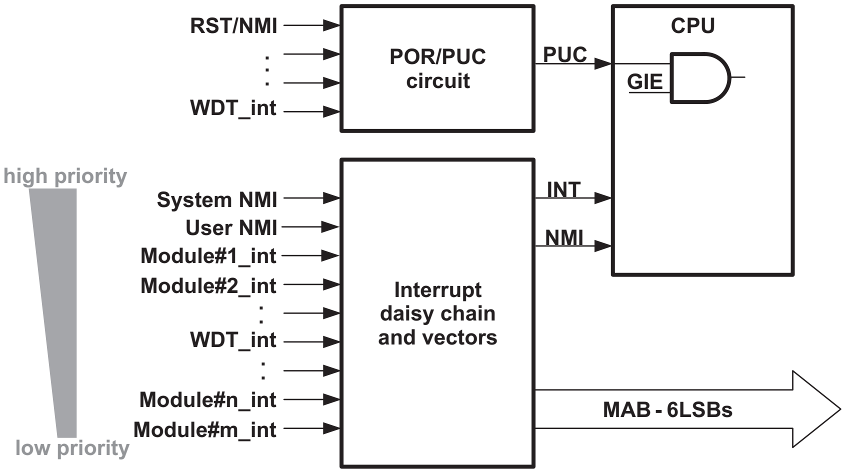


**Figure 1-1. Interrupt Priority**

### 1.4.1 (Non)-Maskable Interrupts (NMI)

The RF430FRL15xH 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:

- 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.

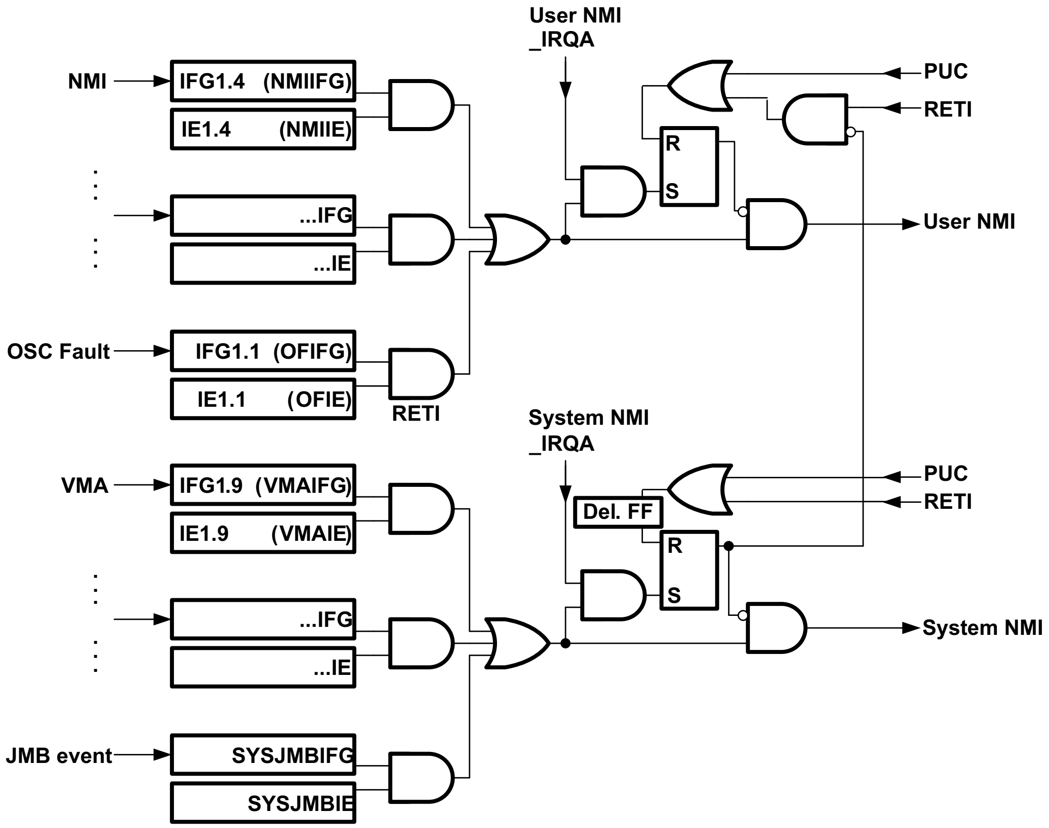


**Figure 1-2. NMI Interrupts with Reentrance Protection**

### *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 user's guide.

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

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

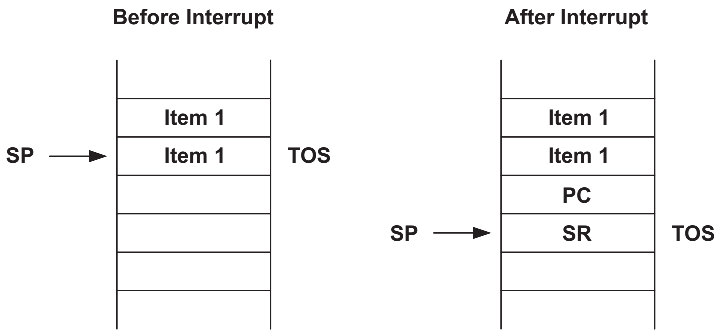


**Figure 1-3. Interrupt Processing**

### 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 five 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 stored in the SR are now in effect, regardless of the settings used during the interrupt service routine.
2. The PC pops from the stack and begins execution where it was interrupted.

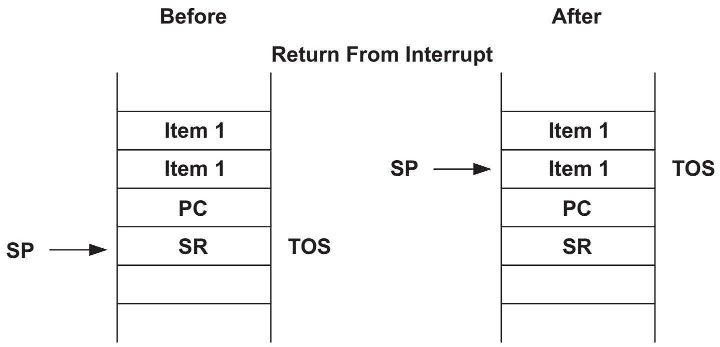


**Figure 1-4. Return From Interrupt**

#### 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 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.

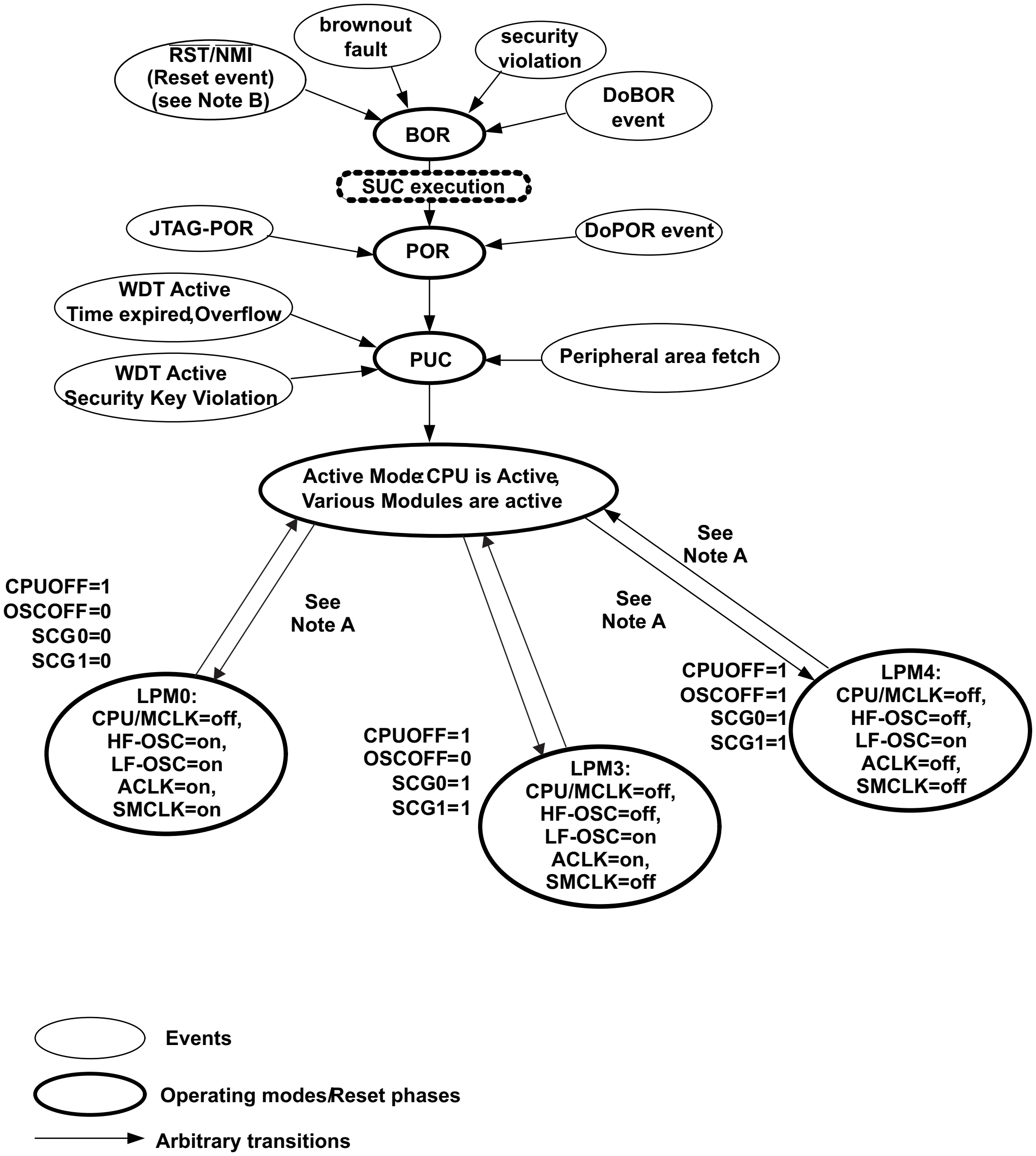
## 1.5 Operating Modes

The operating modes for the RF430FRL15xH 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, LPM3, and 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.



A Any enabled interrupt and NMI performs this transition.

B An enabled reset always restarts the device.

**Figure 1-5. Operating Modes for the RF430FRL15xH Family**

**Table 1-2. Operating Modes**

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

### 1.5.1 Entering and Exiting Low-Power Modes

An enabled interrupt event wakes the device from low-power operating modes LPM0 through LPM4. The program flow for entering and exiting LPM0 through LPM4 is shown in Example 1-2 and described here:

- 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.

***Example 1-2. Entering and Exiting Low-Power Modes***

```
                                        ; 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 clock system to maximize the time in LPM3 or LPM4.

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

## 1.7 Connection of Unused Pins

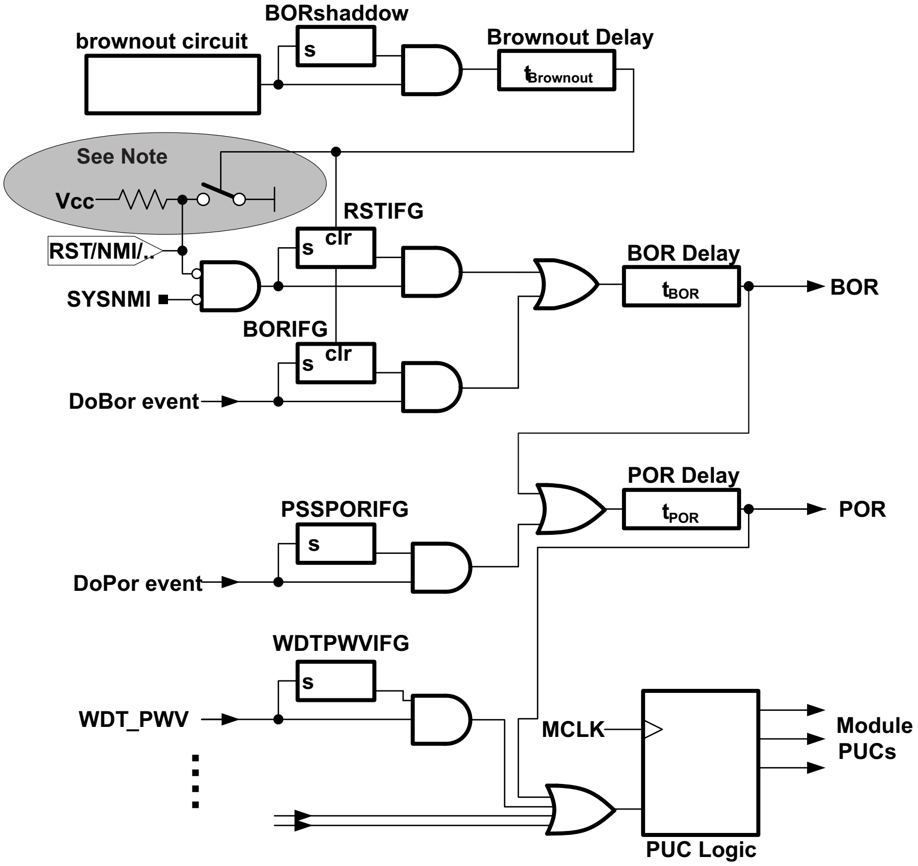
The correct termination of all unused pins is listed in Table 1-3.

**Table 1-3. Connection of Unused Pins**

| Pin | Potential | Comment |
|---|---|---|
| TDI/TMS/TCK | Open | When used for JTAG function |
| $\overline{\text{RST}}$/$\overline{\text{NMI}}$ | $V_{CC}$ 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 resets only the CPU and restarts program execution.



NOTE: See Figure 1-7

**Figure 1-6. BOR, POR, PUC Reset Circuit**

## 1.9 RST/NMI Logic

The reset / nmi control logic is shown in Figure 1-7. The SYSNMI bit defines the function of the RST/NMI terminal. After startup the terminal is configured to have reset functionality. Setting SYSNMI = 1 configures the terminal as an active-low NMI input. During brownout condition the BOR signal, coming from the brownout logic, is set to 1 and the external RST/NMI terminal is pulled low (VSS). With this the user can observe if a reset occurs. Furthermore the SYSNMI bit is set to its reset state (0h), thus the NMI functionality is disabled. The user can trigger a reset by pulling RST/NMI terminal low (SYSNMI = 0). If brownout is not active, the user can trigger an NMI interrupt by pulling RST/NMI terminal low (SYSNMI = 1).

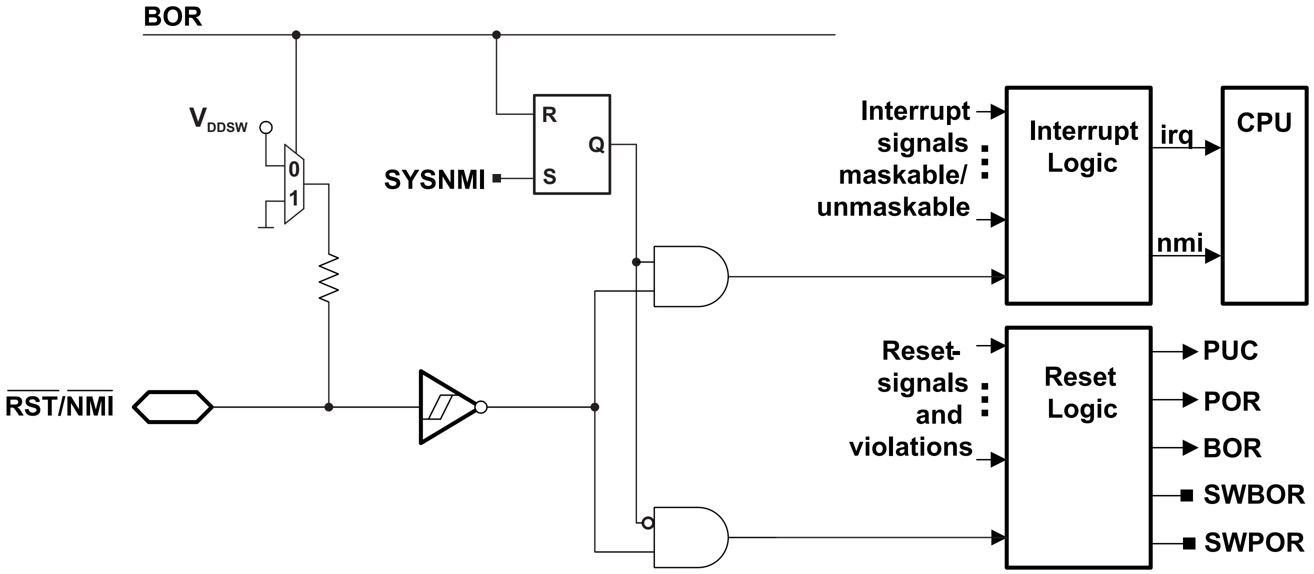


**Figure 1-7. RST/NMI Circuit**

## 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:<br>Power-up, external reset,<br>watchdog | WDTIFG, KEYV | ...<br>Reset | ...<br>0FFFEh | Highest |
| System NMI | | (non)-maskable | 0FFFCh | |
| User NMI:<br>NMI, oscillator fault | NMIIFG<br>OFIFG | ...<br>(non)-maskable<br>(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 $\overline{\text{RST}}/\overline{\text{NMI}}$ 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 (SFRs)

Table 1-5 lists the special function registers.

**Table 1-5. Special Function Registers**

| Offset | Acronym | Register | Type | Access | Reset |
|---|---|---|---|---|---|
| 00h | SFRIE1 | Interrupt enable register | read/write | word | 0000h |
| 00h | SFRIE1_L (IE1) | | read/write | byte | 00h |
| 01h | SFRIE1_H (IE2) | | read/write | byte | 00h |
| 02h | SFRIFG1 | Interrupt flag register | read/write | word | 0082h |
| 02h | SFRIFG1_L (IFG1) | | read/write | byte | 82h |
| 03h | SFRIFG1_H (IFG2) | | read/write | byte | 00h |
| 04h | SFRRPCR | Reset pin control register | read/write | word | 000Eh |
| 04h | SFRRPCR_L | | read/write | byte | 0Eh |
| 05h | SFRRPCR_H | | read/write | byte | 00h |

### 1.11.1 SFRIFG1 Register

Interrupt Flag Register

**Figure 1-8. SFRIFG1 (SFRIFG1_L, SFRIFG1_H) 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 |
|---|---|---|---|---|---|---|---|
| **JMBOUTIFG** | **JMBINIFG** | **Reserved** | **NMIIFG** | **VMAIFG** | **Reserved** | **OFIFG** | **WDTIFG** |
| rw-1 | rw-0 | r0 | rw-0 | rw-0 | r0 | rw-1 | rw-0 |

**Table 1-6. SFRIFG1 (SFRIFG1_L, SFRIFG1_H) Register Description**

| Bit | Field | Description |
|---|---|---|
| 15-8 | Reserved | Reserved. Reads back as 0. |
| 7 | JMBOUTIFG | JTAG mailbox output interrupt flag<br>0b = No interrupt pending. When in 16-bit mode (JMBMODE = 0), this bit is cleared automatically when SYSJMBO0 has been written by the CPU. When in 32-bit mode (JMBMODE = 1), this bit is cleared automatically when both SYSJMBO0 and SYSJMBO1 have been written by the CPU. This bit is also cleared when the associated vector in SYSSNIV has been read.<br>1b = Interrupt pending, JMBO registers are ready for new messages. In 16-bit mode (JMBMODE = 0) SYSJMBO0 has been received by JTAG. In 32-bit mode (JMBMODE = 1) , SYSJMBO0 and SYSJMBO1 have been received by JTAG. |
| 6 | JMBINIFG | JTAG mailbox input interrupt flag<br>0b = No interrupt pending. When in 16-bit mode (JMBMODE = 0), this bit is cleared automatically when SYSJMBI0 is read by the CPU. When in 32-bit mode (JMBMODE = 1), this bit is cleared automatically when both SYSJMBI0 and SYSJMBI1 have been read by the CPU. This bit is also cleared when the associated vector in SYSSNIV has been read.<br>1b = Interrupt pending, a message is waiting in the JMBIN registers. In 16-bit mode (JMBMODE = 0) when SYSJMBI0 has been written by JTAG. In 32 bit mode (JMBMODE = 1) when SYSJMBI0 and SYSJMBI1 have been written by JTAG. |
| 5 | Reserved | Reserved. Reads back as 0. |
| 4 | NMIIFG | NMI pin interrupt flag<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 3 | VMAIFG | Vacant memory access interrupt flag<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 2 | Reserved | Reserved. Reads back as 0. |
| 1 | OFIFG | Oscillator fault interrupt flag<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 0 | WDTIFG | Watchdog timer interrupt flag. In watchdog mode, WDTIFG remains set until reset by software. In interval mode, WDTIFG is reset automatically by servicing the interrupt, or can be reset by software. Because other bits in ~IFG1 may be used for other modules, it is recommended to clear WDTIFG by using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instructions.<br>0b = No interrupt pending<br>1b = Interrupt pending |

### 1.11.2 SFRIE1 Register

Interrupt Enable Register

**Figure 1-9. SFRIE1 (SFRIE1_L, SFRIE1_H) 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 |
| JMBOUTIE | JMBINIE | Reserved | NMIIE | VMAIE | Reserved | OFIE | WDTIE |
| rw-0 | rw-0 | r0 | rw-0 | rw-0 | r0 | rw-0 | rw-0 |

**Table 1-7. SFRIE1 (SFRIE1_L, SFRIE1_H) Register Description**

| Bit | Field | Description |
|---|---|---|
| 15-8 | Reserved | Reserved. Reads back as 0. |
| 7 | JMBOUTIE | JTAG mailbox output interrupt enable flag<br>0b = Interrupts disabled<br>1b = Interrupts enabled |
| 6 | JMBINIE | JTAG mailbox input interrupt enable flag<br>0b = Interrupts disabled<br>1b = Interrupts enabled |
| 5 | Reserved | Reserved. Reads back as 0. |
| 4 | NMIIE | NMI pin interrupt enable flag<br>0b = Interrupts disabled<br>1b = Interrupts enabled |
| 3 | VMAIE | Vacant memory access interrupt enable flag<br>0b = Interrupts disabled<br>1b = Interrupts enabled |
| 2 | Reserved | Reserved. Reads back as 0. |
| 1 | OFIE | Oscillator fault interrupt enable flag<br>0b = Interrupts disabled<br>1b = Interrupts enabled |
| 0 | WDTIE | 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.<br>0b = Interrupts disabled<br>1b = Interrupts enabled |

### *1.11.3 SFRRPCR Register*

Reset Pin Control Register

**Figure 1-10. SFRRPCR (SFRRPCR_H, SFRRPCR_L) 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 | | | | SYSRSTRE | SYSRSTUP | SYSNMIES | SYSNMI |
| r0 | r0 | r0 | r0 | r1 | r1 | r1 | rw-0 |

**Table 1-8. SFRRPCR (SFRRPCR_H, SFRRPCR_L) Register Description**

| Bit | Field | Description |
|---|---|---|
| 15-4 | Reserved | Reserved. Reads back as 0. |
| 3 | SYSRSTRE | Reset pin resistor enable<br>0b = Pullup or pulldown (see bit 2) resistor at the $\overline{\text{RST}}$/NMI pin is disabled<br>1b = Pullup or pulldown (see bit 2) resistor at the $\overline{\text{RST}}$/NMI pin is enabled |
| 2 | SYSRSTUP | Reset resistor pin pullup / pulldown<br>0b = Pulldown is selected<br>1b = Pullup is selected |
| 1 | SYSNMIES | 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.<br>0b = NMI on rising edge<br>1b = NMI on falling edge |
| 0 | SYSNMI | NMI select. This bit selects the function for the $\overline{\text{RST}}$/NMI pin.<br>0b = Reset function<br>1b = NMI function |

## 1.12 CSYS_A Registers

The CSYS_A registers are listed in Table 1-9. 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 1-9. CSYS_A Configuration Registers**

| Offset | Acronym | Register | Type | Access | Reset |
|---|---|---|---|---|---|
| 00h | SYSCTL | System control register | read/write | word | 0020h |
| 00h | SYSCTL_L | | read/write | byte | 20h |
| 01h | SYSCTL_H | | read/write | byte | 00h |
| 06h | SYSJMBC | JTAG mailbox control register | read/write | word | 000Ch |
| 06h | SYSJMBC_L | | read/write | byte | 0Ch |
| 07h | SYSJMB_H | | read/write | byte | 00h |
| 08h | SYSJMBI0 | JTAG mailbox input register 0 | read/write | word | 0000h |
| 08h | SYSJMBI0_L | | read/write | byte | 00h |
| 09h | SYSJMBI0_H | | read/write | byte | 00h |
| 0Ah | SYSJMBI1 | JTAG mailbox input register 1 | read/write | word | 0000h |
| 0Ah | SYSJMBI1_L | | read/write | byte | 00h |
| 0Bh | SYSJMBI1_H | | read/write | byte | 00h |
| 0Ch | SYSJMBO0 | JTAG mailbox output register 0 | read/write | word | 0000h |
| 0Ch | SYSJMBO0_L | | read/write | byte | 00h |
| 0Dh | SYSJMBO0_H | | read/write | byte | 00h |
| 0Eh | SYSJMBO1 | JTAG mailbox output register 1 | read/write | word | 0000h |
| 0Eh | SYSJMBO1_L | | read/write | byte | 00h |
| 0Fh | SYSJMBO1_H | | read/write | byte | 00h |
| 10h | SYSCNF | System configuration register | read/write | word | 0F00h |
| 10h | SYSCNF_L | | read/write | byte | 00h |
| 11h | SYSCNF_H | | read/write | byte | 0Fh |
| 18h | SYSBERRIV | Bus error vector generator | read only | word | 0000h |
| 1Ah | SYSUNIV | User NMI vector generator | read only | word | 0000h |
| 1Ch | SYSSNIV | System NMI vector generator | read only | word | 0000h |
| 1Eh | SYSRSTIV | Reset vector generator | read only | word | 0002h |

### 1.12.1 SYSCTL Register

SYS Control Register

**Figure 1-11. SYSCTL (SYSCTL_L, SYSCTL_H) Register**

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="8">Reserved</td></tr>
<tr><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td></tr>
</table>

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="2">Reserved</td><td>SYSJTAGPIN</td><td colspan="4">Reserved</td><td>SYSRIVECT</td></tr>
<tr><td>r0</td><td>r0</td><td>r1</td><td>r0</td><td>r0</td><td>r0</td><td>r0</td><td>rw-[0]</td></tr>
</table>

**Table 1-10. SYSCTL (SYSCTL_L, SYSCTL_H) Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | Reserved | R | 0h | Reserved. Reads back as 0. |
| 7-6 | Reserved | R | 0h | Reserved. Reads back as 0. |
| 5 | SYSJTAGPIN | R | 1h | 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.<br>0b = SBW is primary JTAG interface<br>1b = Explicit 4-wire JTAG is primary JTAG interface |
| 4-1 | Reserved | R | 0h | Reserved. Reads back as 0. |
| 0 | SYSRIVECT | RW | 0h | RAM based interrupt vectors<br>0b = Interrupt vectors generated with end address: top of lower 64k memory 0FFFFh<br>1b = Interrupt vectors generated with end address: top of RAM |

### 1.12.2 SYSJMBC Register

JTAG Mailbox Control Register

**Figure 1-12. SYSJMBC (SYSJMBC_L, SYSBMBC_H) 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 | JMBOUT1FG | JMBOUT0FG | JMBIN1FG | JMBIN0FG |
| rw-(0) | rw-(0) | r0 | rw-0 | r-(1) | r-(1) | rw-(0) | rw-(0) |

**Table 1-11. SYSJMBC (SYSJMBC_L, SYSBMBC_H) Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | Reserved | R | 0h | Reserved. Reads back as 0. |
| 7 | JMBCLR1OFF | RW | 0h | Incoming JTAG Mailbox 1 flag auto-clear disable<br>0b = JMBIN1FG cleared on read of SYSJMBI1 register<br>1b = JMBIN1FG cleared by software |
| 6 | JMBCLR0OFF | RW | 0h | Incoming JTAG Mailbox 0 flag auto-clear disable<br>0b = JMBIN0FG cleared on read of SYSJMBI0 register<br>1b = JMBIN0FG cleared by software |
| 5 | Reserved | R | 0h | Reserved. Reads back as 0. |
| 4 | JMBMODE | RW | 0h | This bit defined the operation mode of JMB for SYSJMBI0, SYSJMBI1, SYSJMBO0, and SYSJMBO1. Before switching this bit pad and flush out any partial content to avoid data drops.<br>0b = 16-bit transfers using SYSJMBO0 and SYSJMBI0 only<br>1b = 32-bit transfers using SYSJMBI0, SYSJMBI1, SYSJMBO0, and SYSJMBO1 |
| 3 | JMBOUT1FG | R | 1h | Outgoing JTAG Mailbox 1 flag. This bit is cleared automatically when a message is written to the upper byte of SYSJMBO1 or as word access (for example, by the CPU or DMA) and is set after the message is read by JTAG.<br>0b = SYSJMBO1 is not ready to receive new data<br>1b = SYSJMBO1 is ready to receive new data |
| 2 | JMBOUT0FG | R | 1h | Outgoing JTAG Mailbox 0 flag. This bit is cleared automatically when a message is written to the upper byte of SYSJMBO0 or as word access (for example, by the CPU or DMA) and is set after the message is read by JTAG.<br>0b = SYSJMBO0 is not ready to receive new data<br>1b = SYSJMBO0 is ready to receive new data |
| 1 | JMBIN1FG | RW | 0h | Incoming JTAG Mailbox 1 flag. This bit is set when a new message (provided via JTAG) is available in SYSJMBI1. This flag is cleared automatically on read of SYSJMBI1 when JMBCLR1OFF = 0 (auto clear mode). On JMBCLR1OFF = 1 JMBIN1FG needs to be cleared by software.<br>0b = SYSJMBI1 has no new data<br>1b = SYSJMBI1 has new data available |
| 0 | JMBIN0FG | RW | 0h | Incoming JTAG Mailbox 0 flag. This bit is set when a new message (provided via JTAG) is available in SYSJMBI1. This flag is cleared automatically on read of SYSJMBI0 when JMBCLR0OFF = 0 (auto clear mode). On JMBCLR0OFF = 1, JMBIN1FG needs to be cleared by software.<br>0b = SYSJMBI0 has no new data<br>1b = SYSJMBI0 has new data available |

### 1.12.3 SYSJMBI0 and SYSJMBI1 Registers

JTAG Mailbox Input 0 and JTAG Mailbox Input 1 Registers

**Figure 1-13. SYSJMBI0 (SYSJMBI0_L, SYSJMBI0_H) and SYSJMBI1 (SYSJMBI1_L, SYSJMBI1_H) Registers**

| 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> |

**Table 1-12. SYSJMBI0 (SYSJMBI0_L, SYSJMBI0_H) and SYSJMBI1 (SYSJMBI1_L, SYSJMBI1_H) Registers Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | MSGHI | R | 0h | JTAG mailbox incoming message high byte |
| 7-0 | MSGLO | R | 0h | JTAG mailbox incoming message low byte |

### 1.12.4 SYSJMBO0 and SYSJMBO1 Registers

JTAG Mailbox Output 0 and JTAG Mailbox Output 1 Registers

**Figure 1-14. SYSJMBO0 (SYSJMBO0_L, SYSJMBO0_H) and SYSJMBO1 (SYSJMBO1_L, SYSJMBO1_H) Registers**

| 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> |

**Table 1-13. SYSJMBO0 (SYSJMBO0_L, SYSJMBO0_H) and SYSJMBO1 (SYSJMBO1_L, SYSJMBO1_H) Registers Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | MSGHI | RW | 0h | JTAG mailbox outgoing message high byte |
| 7-0 | MSGLO | RW | 0h | JTAG mailbox outgoing message low byte |

### 1.12.5 SYSCNF Register

SYS Configuration Register

**Figure 1-15. SYSCNF (SYSCNF_L, SYSCNF_H) Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | | | ROMDEVMODE | DEVRAMLCK | FRAMLCK2 | FRAMLCK1 | FRAMLCK0 |
| r0 | r0 | r0 | rw-[0] | rw-[1] | rw-[1] | rw-[1] | rw-[1] |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

**Table 1-14. SYSCNF (SYSCNF_L, SYSCNF_H) Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-13 | Reserved | R | 0h | Reserved. Reads back as 0. |
| 12 | ROMDEVMODE | RW | 0h | Enable ROM development mode by re-mapping the upper 3.5kB of the ROM to RAM.<br>0b = ROM Development mode not active<br>1b = ROM Development mode active. Upper 3.5kB of ROM are mapped to RAM |
| 11 | DEVRAMLCK | RW | 1h | Write protect re-mapped ROM development area (upper 3.5kB)<br>0b = ROM development area writeable<br>1b = ROM development area not writeable |
| 10 | FRAMLCK2 | RW | 1h | Write protect FRAM bank A (see device-specific data sheet for address space)<br>0b = FRAM area is writeable<br>1b = FRAM area is not writeable |
| 9 | FRAMLCK1 | RW | 1h | Write protect FRAM bank B (see device-specific data sheet for address space)<br>0b = FRAM area is writeable<br>1b = FRAM area is not writeable |
| 8 | FRAMLCK0 | RW | 1h | Write protect FRAM bank C (see device-specific data sheet for address space)<br>0b = FRAM area is writeable<br>1b = FRAM area is not writeable |
| 7-0 | Reserved | R | 0h | Reserved. Reads back as 0. |

### 1.12.6 SYSBERRIV Register

Bus Error Interrupt Vector Register

**Figure 1-16. SYSBERRIV (SYSBERRIV_H, SYSBERRIV_L) Register**

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

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

**Table 1-15. SYSBERRIV (SYSBERRIV_H, SYSBERRIV_L) Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | SYSBERRVEC | R | 0h | Bus error interrupt vector. 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).<br>0000h = No interrupt pending<br>0002h and higher = Valid bus error from peripheral (see device-specific data sheet)<br>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. |

### 1.12.7 SYSUNIV Register

User NMI Interrupt Vector Register

**Figure 1-17. SYSUNIV (SYSUNIV_H, SYSUNIV_L) Register**

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

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

**Table 1-16. SYSUNIV (SYSUNIV_H, SYSUNIV_L) Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | SYSUNVEC | R | 0h | 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).<br>0000h = No interrupt pending<br>0002h = NMIIFG interrupt pending (highest priority)<br>0004h = OFIFG interrupt pending<br>0006h = Bus error interrupt pending (check SYSBERRIV)<br>0008h and higher = Reserved for future extensions<br>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. |

### 1.12.8 SYSSNIV Register

SYS NMI Interrupt Vector Register

**Figure 1-18. SYSSNIV (SYSSNIV_H, SYSSNIV_L) Register**

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

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

**Table 1-17. SYSSNIV (SYSSNIV_H, SYSSNIV_L) Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | SYSSNVEC | R | 0h | 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).<br>0000h = No interrupt pending<br>0002h = Reserved<br>0004h = VMAIFG interrupt pending<br>0006h = JMBINIFG interrupt pending<br>0008h = JMBOUTIFG interrupt pending<br>000Ah and higher = Reserved for future extensions<br>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. |

### 1.12.9 SYSRSTIV Register

SYS Reset Interrupt Vector Register

**Figure 1-19. SYSRSTIV (SYSRSTIV_H, SYSRSTIV_L) Register**

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

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

**Table 1-18. SYSRSTIV (SYSRSTIV_H, SYSRSTIV_L) Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | SYSRSTVEC | R | 0h | 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).<br>0000h = No interrupt pending<br>0002h = Brownout (BOR) (highest priority)<br>0004h = $\overline{\text{RST}}$/NMI (BOR)<br>0006h = DoBOR (BOR)<br>0008h = Security violation (BOR)<br>000Ah = DoPOR (POR)<br>000Ch = WDT time out (PUC)<br>000Eh = WDT key violation (PUC)<br>0010h = PERF peripheral/configuration area fetch (PUC)<br>0012h and higher = Reserved for future extensions<br>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. |

# Radio-Frequency Power Management Module (RFPMM)

This chapter describes the function of the radio-frequency power management module (RFPMM) and its registers.

Topic Page

## 2.1 RFPMM Introduction

The radio-frequency power management module (RFPMM) features include:

- Supply from either RF field or battery
- Ultra-low-voltage operation: 1.5 V (typical)
- Generation of $V_{DD2X}$ voltage (2.9 V typical) for external circuitry
- Voltage supervision for supply voltages and support voltages
- Software-accessible power-fail indicators
- I/O protection during power-fail condition

The RFPMM manages all functions related to the power supply and its supervision for the device. The RFPMM manages the supply power from two sources and provides power supply for other modules such as the digital core, digital I/Os, FRAM memory, and the sensor AFE.

The device can be supplied by the 13.56-MHz field, an external battery, or both.

The power management also provides reference voltages and currents, bias currents, and voltage supervision instances for monitoring proper voltage levels. Figure 2-1 shows the block diagram of the RFPMM.

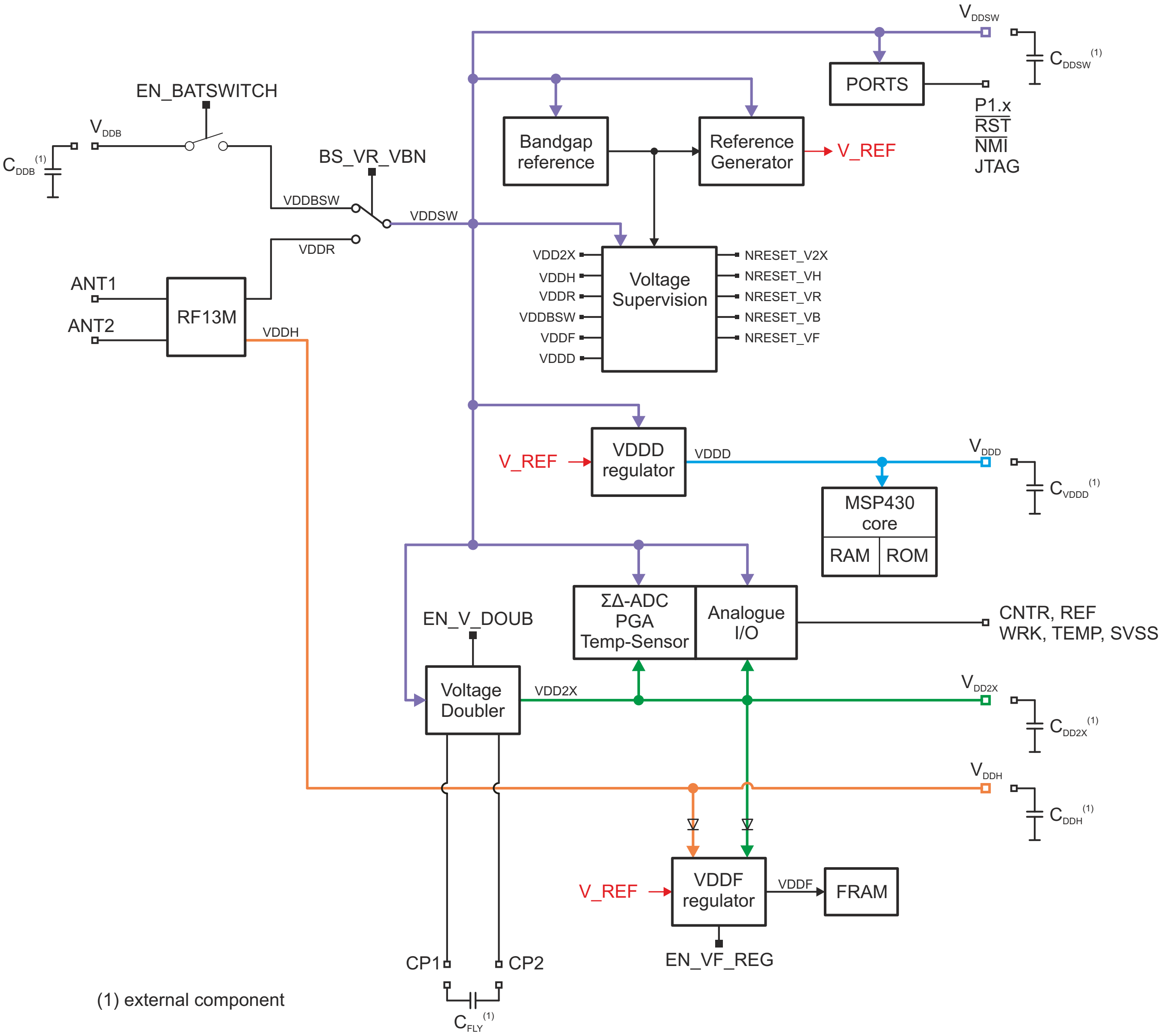


**Figure 2-1. RFPMM Block Diagram**

## 2.2 Supply Voltages

### 2.2.1 Battery Voltage $V_{DDB}$ and Battery Switch

If the battery switch is closed by setting EN_BATSWITCH = 1, the device can be sourced by an external battery voltage $V_{DDB}$. The maximum voltage drop over the battery switch from $V_{DDB}$ to $V_{DDBSW}$ is $V_{DROP}$ (see the device-specific data sheet).

If the battery switch is open, only a very low leakage current $I_{BASVBAT}$ drains the battery to ensure long battery life (see the device-specific data sheet).

### 2.2.2 Rectified Antenna Voltage $V_{DDH}$

The rectified antenna voltage $V_{DDH}$ can be used to supply the nonvolatile memory through the FRAM voltage regulator. The FRAM voltage regulator can also be sourced by $V_{DD2X}$ (for details, see Section 2.4.2).

$V_{DDH}$ is directly derived from the 13.56-MHz field. $V_{DDH}$ is not regulated and is directly dependent on the field strength of the 13.56-MHz field.

### 2.2.3 Regulated Rectified Antenna Voltage $V_{DDR}$

$V_{DDR}$ is the second supply voltage for the system. It is indirectly derived from the 13.56-MHz field and the rectified $V_{DDH}$ voltage. It is regulated and can be used to power the system.

### 2.2.4 Power Supply Switch

The device can be powered by $V_{DDR}$ , $V_{DDB}$, or directly by $V_{DDSW}$. If device is powered by $V_{DDR}$ or $V_{DDB}$, the power supply switch selects the higher of the two voltages. Assuming that the battery voltage is constant, two cases are possible: $V_{DDR}$ is rising from a level below $V_{DDB}$, or it is falling from a level above $V_{DDB}$.

The first case is shown in Figure 2-2 a). $V_{DDR} << V_{DDB}$ , power supply switch voltage $V_{DDSW}$ is connected to $V_{DDB}$. $V_{DDR}$ is rising, $V_{DDSW}$ remains connected to $V_{DDB}$ until $V_{DDR} = V_{SW+}$. At this point power supply switch connects $V_{DDSW}$ to $V_{DDR}$. With rising $V_{DDR}$, $V_{DDSW}$ rises too.

The second case is shown in Figure 2-2 b). $V_{DDR} >> V_{DDB,}$ power supply switch voltage $V_{DDSW}$ is connected to $V_{DDR}$. With falling $V_{DDR}$, $V_{DDB}$ is connected to $V_{DDSW}$ as soon as falls below negative hysteresis level $V_{SW-}$ ($V_{DDR} \geq V_{SW-}$).

Above description assumes that $V_{DDB}$ is constant, and $V_{DDR}$ is rising or falling. The power supply switch behaves in the same way if $V_{DDR}$ is constant and $V_{DDB}$ is rising or falling. In this case, the naming of $V_{DDR}$ and $V_{DDB}$ in Figure 2-2 must be exchanged.

An external buffer capacitor $C_{VDDSW}$ is necessary on the $V_{DDSW}$ pin (see the device-specific data sheet). The position of the power supply switch is indicated by the BS_VR_VBN bit in the RFPMMCTL1 register.

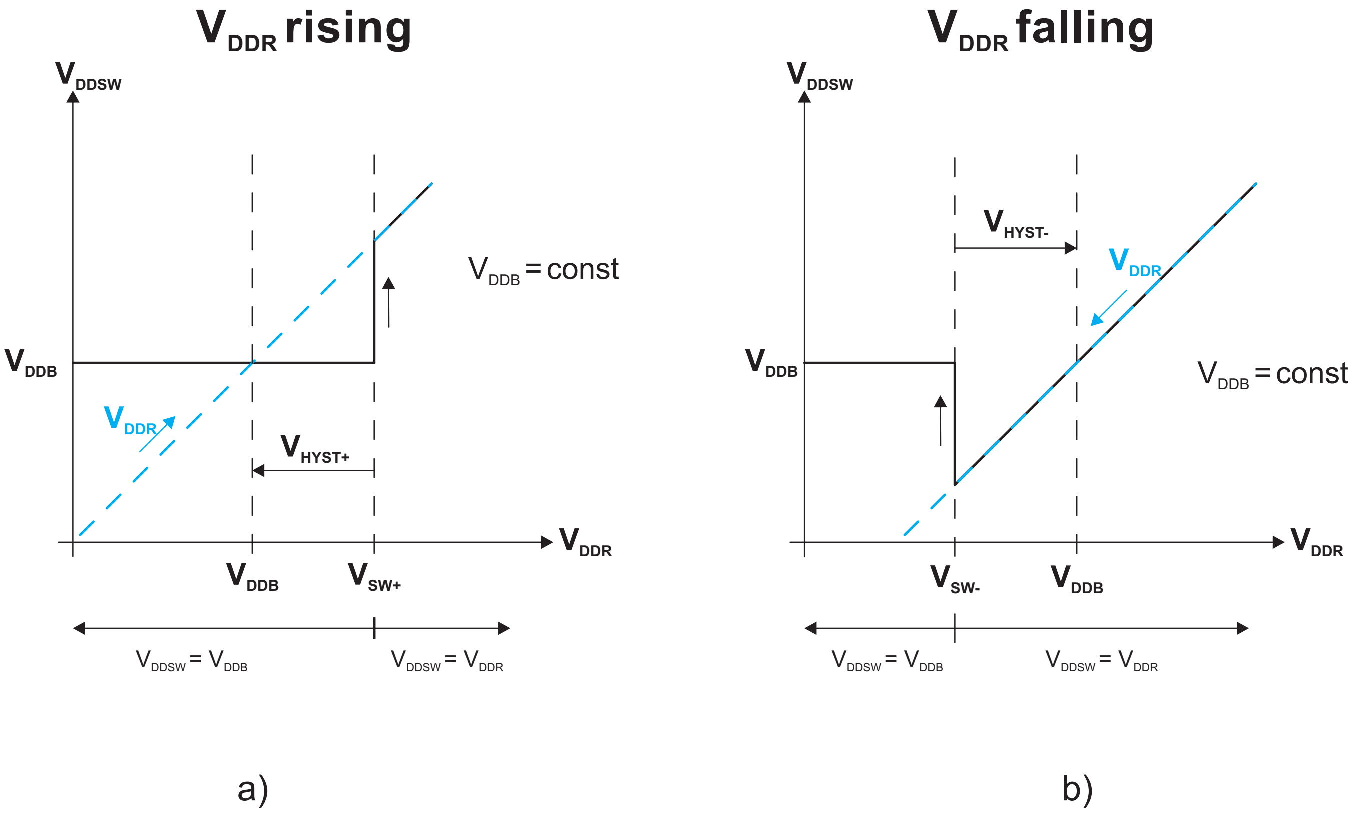


**Figure 2-2. Hysteresis of $V_{DDSW}$**

## 2.3 RFPMM Modes

The RFPMM module supports the following operating modes:

- Not Supplied
- RF Only
- RF and Battery
- Battery Only

### *2.3.1 Not Supplied*

In this mode, the battery switch is off, and the chip is not supplied from the RF-AFE. If a battery is connected to the system, a maximum leakage current of $I_{BASVBAT}$ drains the battery (see the *RF430FRL15xH Firmware User's Guide* SLAU603).

The battery switch can be enabled only if the device is supplied by the RF-AFE and if the correct RF-command is sent (see the *RF430FRL15xH Firmware User's Guide* SLAU603).

### *2.3.2 RF Only*

When the chip is supplied by the RF field, communication from and to the reader is enabled. The power management is supplied by rectified and regulated RF voltage $V_{DDR}$. Thus the power supply switch is closed between $V_{DDR}$ and $V_{DDSW}$.

The bandgap reference, the voltage supervision circuits, the digital core voltage regulator, and the bias generation circuits are active. The FRAM regulator is sourced by the rectified voltage $V_{DDH}$ and is active only during FRAM access. The voltage doubler is disabled, as is the reference generation circuit (see the *13-MHz RF Communication Module (RF13M)* chapter).

ISO communication is performed during in mode, and FRAM access for data read and write is possible. The battery switch stays disabled until an RF command enables it (see the *13-MHz RF Communication Module (RF13M)* chapter). The battery switch can be enabled only during RF Only mode.

### 2.3.3 RF and Battery

To limit the battery current consumption in this mode, the $V_{DDSW}$ is supplied by $V_{DDR}$ from the RF-AFE. Otherwise, continuous RF communication would degrade the battery and reduce its lifetime.

If an RF field is exposed to the chip, the measurement can be corrupted; a flag may mask the result as incorrect.

### 2.3.4 Battery Only

During this mode, no RF field is present, the battery switch is enabled, and a valid battery voltage is present. The battery voltage is supervised by the voltage supervision circuit (see Section 2.5).

If the battery voltage drops below a certain voltage level, the information is stored in the nonvolatile memory before reset (NRESET_VB).

## 2.4 Voltage Regulators

### 2.4.1 $V_{DDD}$ Voltage Regulator

The $V_{DDD}$ voltage regulator provides a constant voltage $V_{DDD}$ for all digital modules, including the MSP430 core and its peripherals such as ROM and RAM. $V_{DDD}$ is directly connected to the brownout reset; therefore, if $V_{DDD}$ drops below $V_{TH-}$, a brownout reset is triggered.

### 2.4.2 $V_{DDF}$ Voltage Regulator

The FRAM voltage regulator generates a constant voltage $V_{DDF}$ for the nonvolatile memory (FRAM). This regulator is sourced either by the rectified voltage from the RF ($V_{DDH}$) if the battery switch is not enabled or by the charge pump output voltage $V_{DD2X}$. When executing code in RAM or ROM, it is possible to turn on the $V_{DDF}$ voltage regulator by setting EN_VF_REG = 1 in the RFPMMCTL0 register. Setting EN_VF_REG = 0 returns control to the CPU again.

### 2.4.3 $V_{DD2X}$ Voltage Regulator (Voltage Doubler)

The voltage doubler generates the voltage $V_{DD2X}$ (2.9 V typical) from a lower battery voltage (approximately 1.5 V). $V_{DD2X}$ supplies the external circuitry. If the device is supplied by battery only, the FRAM voltage regulator is also supplied by $V_{DD2X}$. The $V_{DD2X}$ voltage regulator can be enabled by setting EN_V_DOUB in the RFPMMCTL0 register.

## 2.5 Voltage Supervision

### 2.5.1 Brownout Reset (BOR), Software BOR, Software POR

The primary function of the brownout reset (BOR) circuit occurs when the device is powering up. It is functional during power-up ramp, generating a POR that initializes the system. The BOR circuit sustains this reset until the input power is sufficient for the logic and for proper reset of the system. The BOR is linked directly to $V_{DDD}$ to ensure proper operation of the digital core.

In an application, it may be necessary to cause a BOR via software. Set PMMSWBOR to cause a software-driven BOR. PMMBORIFG is set accordingly. Note that a BOR also initiates a POR and PUC. PMMBORIFG can be cleared by software or by reading SYSRSTIV. Similarly, it is possible to cause a POR in software by setting PMMSWPOR. PMMPORIFG is set accordingly. A POR also initiates a PUC. PMMPORIFG can be cleared by software or by reading SYSRSTIV. Both PMMSWBOR and PMMSWPOR are self clearing. See the CSYS_A module for complete descriptions of BOR, POR, and PUC resets.

### 2.5.2 Reset Flags

The voltage supervision circuit monitors several internal voltages and provides reset flags that the application can use. Table 2-1 lists these voltages and their corresponding reset flags. The supervision circuit detects if one of these voltages falls below a certain threshold and generates the corresponding reset flag. The flags can be found in the RFPMMCTL1 register.

**Table 2-1. Voltages and Reset Signals**

| Supervised Voltage | Reset Flag | Function |
|---|---|---|
| VDD2X | NRESET_V2X | High voltage generated by charge pump |
| VDDH | NRESET_VH | Optional rectified voltage from RF-AFE |
| VDDR | NRESET_VR | Regulated voltage from RF-AFE |
| VDDBSW | NRESET_VB | Battery voltage after battery switch |
| VDDF | NRESET_VF | Regulated voltage for FRAM memory |

## 2.6 Interrupt Handling

The RFPMM module has the following interrupt sources:

- RFPMMIFGV2X
- RFPMMIFGVH
- RFPMMIFGVR
- RFPMMIFGVB
- RFPMMIFGVF

The RFPMMIFGx bits are set when the level of the corresponding voltage falls below a certain threshold level. An interrupt request is generated if the RFPMMIEx bit and the GIE bit are set.

### 2.6.1 RFPMMIV, Interrupt Vector Generator

All RFPMM interrupt sources are prioritized and combined to source a single interrupt vector. RFPMMIV is used to determine which enabled RFPMM interrupt source requested an interrupt. The highest priority RFPMM interrupt request that is enabled generates a number in the RFPMMIV register (see the register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled RFPMM interrupts do not affect the RFPMMIV value.

Any access, read or write, of the RFPMMIV register has no effect on the RFPMMIFGx flags. The RFPMMIFGx flags are reset by clearing the flags in software.

If another interrupt is pending after servicing of an interrupt, another interrupt is generated. For example, if the RFPMMIFGV2X and one or more RFPMMIFGx interrupts are pending when the interrupt service routine accesses the RFPMMIV register, the RFPMMIFGV2X interrupt condition is serviced first and the corresponding flag must be cleared in software. After the RETI instruction of the interrupt service routine is executed, the highest priority RFPMMIFGx pending generates another interrupt request.

## 2.7 RFPMM Registers

Table 2-2 lists the memory-mapped registers for the RFPMM. See the device-specific data manual for the base memory address of these registers. All other register offset addresses not listed in Table 2-2 should be considered as reserved locations and the register contents should not be modified.

**Table 2-2. RFPMM Registers**

| Offset | Acronym | Register Name | Type | Access | Reset | Section |
|---|---|---|---|---|---|---|
| 00h | RFPMMCTL0 | RF PMM Control Register 0 | read/write | Word | 9600h | Section 2.7.1 |
| | RFPMMCTL0_L | | read/write | Byte | 00h | |
| | RFPMMCTL0_H | | read/write | Byte | 96h | |
| 02h | RFPMMCTL1 | RF PMM Control Register 1 | read/write | Word | 001Fh | Section 2.7.2 |
| | RFPMMCTL1_L | | read/write | Byte | 1Fh | |
| | RFPMMCTL1_H | | read/write | Byte | 00h | |
| 04h | RFPMMIE | RF PMM Interrupt Enable Register | read/write | Word | 0000h | Section 2.7.3 |
| | RFPMMIE_L | | read/write | Byte | 00h | |
| | RFPMMIE_H | | read/write | Byte | 00h | |
| 06h | RFPMMIFG | RF PMM Interrupt Flag Register | read/write | Word | 0000h | Section 2.7.4 |
| | RFPMMIFG_L | | read/write | Byte | 00h | |
| | RFPMMIFG_H | | read/write | Byte | 00h | |
| 08h | RFPMMIV | RF PMM Interrupt Vector Register | read/write | Word | 0000h | Section 2.7.5 |
| | RFPMMIV_L | | read/write | Byte | 00h | |
| | RFPMMIV_H | | read/write | Byte | 00h | |

### 2.7.1 RFPMMCTL0 Register

RF Power Management Module Control Register 0

**Figure 2-3. RFPMMCTL0 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| RFPMMPW | | | | | | | |
| 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 | EN_V_DOUB | EN_BATSWITCH | EN_VF_REG | PMMSWPOR | PMMSWBOR | Reserved | |
| r0 | rw-{0} | rw-{0} | rw-{0} | rw-(0) | rw-[0] | r0 | r0 |

**Table 2-3. RFPMMCTL0 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | RFPMMPW | RW | 96h | RFPMMCTL0 password. Must be written as A5h to enable write access to configuration registers. Always reads as 96h. |
| 7 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 6 | EN_V_DOUB | RW | 0h | Enables the voltage regulator for the digital core VDD<br>0b = Inactive<br>1b = Active |
| 5 | EN_BATSWITCH | RW | 0h | Enables the battery switch<br>0b = Inactive<br>1b = Active |
| 4 | EN_VF_REG | RW | 0h | Enables the voltage regulator for the FRAM memory VDDF permanently<br>0b = CPU controlled<br>1b = Enables FRAM voltage regulator permanently |
| 3 | PMMSWPOR | RW | 0h | Software power-on reset. Set this bit to 1 to trigger a POR. This bit is self clearing. |
| 2 | PMMSWBOR | RW | 0h | Software brownout reset. Set this bit to 1 to trigger a BOR. This bit is self clearing. |
| 1-0 | Reserved | R | 0h | Reserved. Always reads as 0. |

### 2.7.2 RFPMMCTL1 Register

RF Power Management Module Control Register 1

**Figure 2-4. RFPMMCTL1 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | BS_VR_VBN |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r-{0} |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | NRESET_V2X | NRESET_VH | NRESET_VR | NRESET_VB | NRESET_VF |
| r0 | r0 | r0 | r-{0} | r-{1} | r-{1} | r-{1} | r-{1} |

**Table 2-4. RFPMMCTL1 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-9 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 8 | BS_VR_VBN | R | 0h | Indicates position of the bulk switch<br>0b = $V_{DDSW}$ connected to $V_{DDB}$ (battery voltage)<br>1b = $V_{DDSW}$ connected to $V_{DDR}$ (rectified RF voltage) |
| 7-5 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 4 | NRESET_V2X | R | 0h | Active-low reset signal of $V_{DD2X}$ voltage<br>0b = Reset active<br>1b = Reset inactive |
| 3 | NRESET_VH | R | 1h | Active-low reset signal of $V_{DDH}$ voltage<br>0b = Reset active<br>1b = Reset inactive |
| 2 | NRESET_VR | R | 1h | Active-low reset signal of $V_{DDR}$ voltage<br>0b = Reset active<br>1b = Reset inactive |
| 1 | NRESET_VB | R | 1h | Active-low reset signal of $V_{DDBSW}$ voltage<br>0b = Reset active<br>1b = Reset inactive |
| 0 | NRESET_VF | R | 1h | Active-low reset signal of $V_{DDF}$ voltage<br>0b = Reset active<br>1b = Reset inactive |

### *2.7.3 RFPMMIE Register*

RF Power Management Module Interrupt Enable Register

**Figure 2-5. RFPMMIE Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | RFPMMIEV2X | RFPMMIEVH | RFPMMIEVR | RFPMMIEVB | RFPMMIEVF |
| r0 | r0 | r0 | rw-{0} | rw-{0} | rw-{0} | rw-{0} | rw-{0} |

**Table 2-5. RFPMMIE Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-5 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 4 | RFPMMIEV2X | RW | 0h | Interrupt enable voltage regulator VDD2X<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 3 | RFPMMIEVH | RW | 0h | Interrupt enable voltage regulator VDDF<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 2 | RFPMMIEVR | RW | 0h | Interrupt enable voltage regulator VDDR<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 1 | RFPMMIEVB | RW | 0h | Interrupt enable voltage regulator VDDB<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 0 | RFPMMIEVF | RW | 0h | Interrupt enable voltage regulator VDDF<br>0b = Interrupt disabled<br>1b = Interrupt enabled |

### 2.7.4 RFPMMIFG Register

RF Power Management Module Interrupt Flag Register

**Figure 2-6. RFPMMIFG Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | RFPMMIFGV2X | RFPMMIFGVH | RFPMMIFGVR | RFPMMIFGVB | RFPMMIFGVF |
| r0 | r0 | r0 | rw-{0} | rw-{0} | rw-{0} | rw-{0} | rw-{0} |

**Table 2-6. RFPMMIFG Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-5 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 4 | RFPMMIFGV2X | RW | 0h | Interrupt flag voltage regulator VDD2X<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 3 | RFPMMIFGVH | RW | 0h | Interrupt flag voltage regulator VDDH<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 2 | RFPMMIFGVR | RW | 0h | Interrupt flag voltage regulator VDDR<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 1 | RFPMMIFGVB | RW | 0h | Interrupt flag voltage regulator VDDB<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 0 | RFPMMIFGVF | RW | 0h | Interrupt flag voltage regulator VDDF<br>0b = No interrupt pending<br>1b = Interrupt pending |

### 2.7.5 *RFPMMIV Register*

RF Power Management Module Interrupt Vector Register

**Figure 2-7. RFPMMIV Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| RFPMMIV | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RFPMMIV | | | | | | | |
| r0 | r0 | r0 | r0 | rw-{0} | rw-{0} | rw-{0} | r0 |

**Table 2-7. RFPMMIV Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | | R | 0h | RFPMM interrupt vector table<br>00h = No interrupt pending<br>02h = Reset from VDD2X voltage regulator; Priority: Highest<br>04h = Reset from VDDH voltage regulator<br>06h = Reset from VDDR voltage regulator<br>08h = Reset from VDDB voltage regulator<br>0Ah = Reset from VDDF voltage regulator; Priority: Lowest |

Chapter 3
SLAU506–December 2014

# Compact Clock System (CCS)

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

**Topic** **Page**

## 3.1 Compact Clock System (CCS) Introduction

The CCS module generates the system clocks required for the device. The CCS uses two internal clock signals and up to two external signals as sources, which allows the user to select the best balance of performance and low power consumption. The CCS module can be configured to operate without any external components.

The CCS uses three clock sources:

- HFCLK: From an internal high-frequency oscillator (HF-OSC)
- LFCLK: From an internal low-frequency oscillator (LF-OSC)
- CLKIN/XIN: Optional external clock source or external oscillator

The CCS generates four clock signals:

- ACLK: Auxiliary clock. ACLK is sourced from HFCLK, LFCLK, or CLKIN/XIN. 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/XIN. 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/XIN. 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.

Figure 3-1 shows the block diagram of the CCS module for the RF430FRL15xH devices.

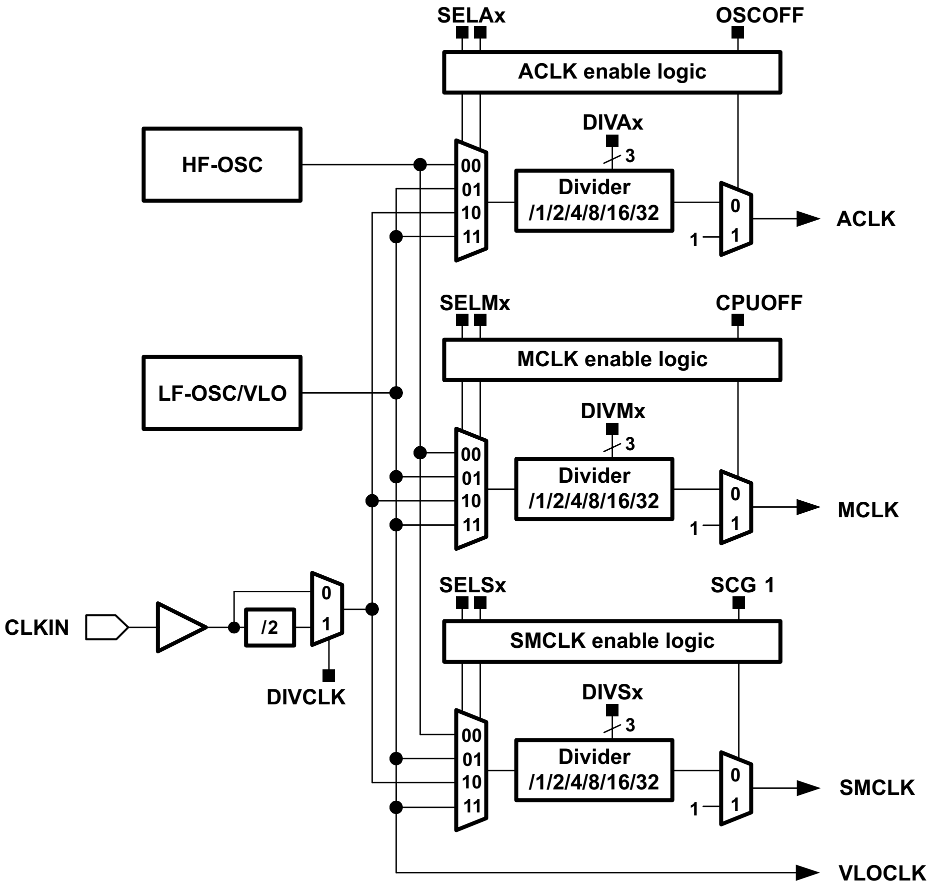


**Figure 3-1. CCS Block Diagram, RF430FRL15xH**

## 3.2 CCS Module Operation

After a PUC, the CSS module default configuration is:

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

Status register control bits (SCG0, SCG1, OSCOFF, and CPUOFF) configure the device operating modes and enable or disable portions of the CCS module (see the *System Resets, Interrupts, and Operating Modes* chapter). The CCSCTLx registers configure the CCS module. Software can configure or reconfigure the CCS at any time during program execution.

### 3.2.1 Operation From Low-Power Modes Requested by Peripheral Modules

During low-power modes that typically disable a clock signal, 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 causes that clock's off signal to be overridden. This override does not change the setting of clock-off control bit. For example, a peripheral module might require the MCLK source, even when MCLK is disabled by the CPUOFF bit. The module requests the MCLK source by setting the MCLK_REQ bit. This causes the CCS to ignore the CPUOFF bit and to generate the MCLK signal for the requesting peripheral module.

### 3.2.2 Internal Low-Frequency Oscillator (LF-OSC)

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

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

### 3.2.3 Internal High-Frequency Oscillator (HF-OSC)

The HF-OSC can be used for cost-sensitive applications that do not require an external clock source. The HF-OSC can be internally trimmed and provides a stable frequency that can be used as input into all clock trees.

The HFCLK is selected when the CCS uses it to source ACLK, MCLK or SMCLK (SELA = 0, SELM = 0, or SELS = 0, respectively).

### 3.2.4 Trimming of HF-OSC and LF-OSC

The HF-OSC and the LF-OSC use the same configuration register to store the trim value. Therefore, only one oscillator can be trimmed. After the low-frequency oscillator is trimmed, the high-frequency oscillator uses the same trim value. Likewise, after the high-frequency oscillator is trimmed, the low-frequency oscillator uses the same trim value. As a result of this implementation, the oscillator that was not trimmed has a larger frequency shift.

### 3.2.5 External Clock Source

An external clock source can be either a digital clock connected to CLKIN or a crystal oscillator connected to XIN and XOUT. CLKIN is selected when it is used to source ACLK, MCLK, or SMCLK (SELA = 2 SELM = 2, or SELS = 2, respectively). To use an external oscillator, XTOFF must also be set to 0. When using an external signal, the frequency and voltage levels 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.

### 3.2.6 CCS Module Fail-Safe Operation

The CCS module incorporates an oscillator-fault fail-safe feature. This feature detects an oscillator fault CLKIN/XIN as shown in Figure 3-2. The available fault conditions are:

- 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 must be cleared by software.

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

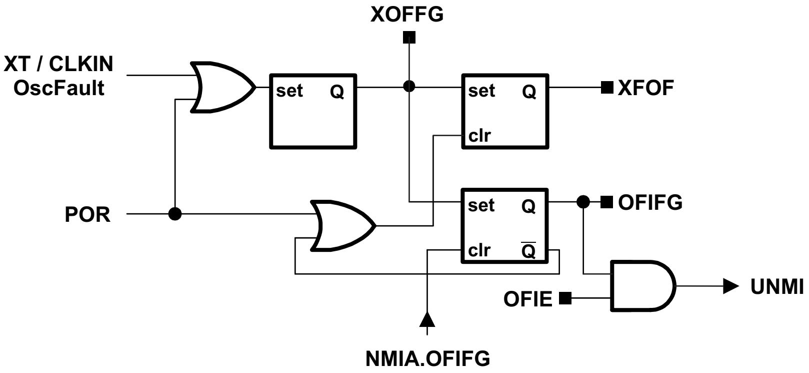


**Figure 3-2. Oscillator Fault Logic for Devices With HF-OSC**

## 3.3 CCS Registers

The CCS module registers and their address offsets are listed in Table 3-1. The base address for the CCS registers is listed in the device-specific data sheet.

The password defined in the CCSCTL0 register controls access to the CCS registers. After the correct password is written, write access is enabled. Write access is disabled by writing a wrong password in byte mode to the CCSCTL0 upper byte. Word access to CCSCTL0 with a wrong password causes a PUC. Write access to any CCS register other than CCSCTL0 while write access is not enabled causes a PUC.

**Table 3-1. CCS Registers**

| Offset | Acronym | Register Name | Type | Access | Reset | Section |
|---|---|---|---|---|---|---|
| 00h | CCSCTL0 | Compact clock system control 0 | read/write | word | 9600h | Section 3.3.1 |
| 00h | CCSCTL0_L | | | byte | 00h | |
| 01h | CCSCTL0_H | | | byte | 96h | |
| 02h | CCSCTL1 | Compact clock system control 1 | read/write | word | 0001h | Section 3.3.2 |
| 02h | CCSCTL1_L | | | byte | 01h | |
| 03h | CCSCTL1_H | | | byte | 00h | |
| 08h | CCSCTL4 | Compact clock system control 4 | read/write | word | 0100h | Section 3.3.3 |
| 08h | CCSCTL4_L | | | byte | 00h | |
| 09h | CCSCTL4_H | | | byte | 01h | |
| 0Ah | CCSCTL5 | Compact clock system control 5 | read/write | word | 0333h | Section 3.3.4 |
| 0Ah | CCSCTL5_L | | | byte | 33h | |
| 0Bh | CCSCTL5_H | | | byte | 03h | |
| 0Ch | CCSCTL6 | Compact clock system control 6 | read/write | word | 00C1h | Section 3.3.5 |
| 0Ch | CCSCTL6_L | | | byte | C1h | |
| 0Dh | CCSCTL6_H | | | byte | 00h | |
| 0Eh | CCSCTL7 | Compact clock system control 7 | read/write | word | 0041h | Section 3.3.6 |
| 0Eh | CCSCTL7_L | | | byte | 41h | |
| 0Fh | CCSCTL7_H | | | byte | 00h | |
| 10h | CCSCTL8 | Compact clock system control 8 | read/write | word | 0007h | Section 3.3.7 |
| 10h | CCSCTL8_L | | | byte | 07h | |
| 11h | CCSCTL8_H | | | byte | 00h | |

### *3.3.1 CCSCTL0 Register*

Compact Clock System Control 0 Register

**Figure 3-3. CCSCTL0 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CCSKEY | | | | | | | |
| 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 |

**Table 3-2. CCSCTL0 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | CCSKEY | RW | 96h | CCSKEY password. Write 00h to lock register access. Write A5h to unlock register access. A write of any other value causes a PUC. Always reads as 96h. |
| 7-0 | Reserved | R | 0h | Reserved. Always reads as 0. |

### *3.3.2 CCSCTL1 Register*

Compact Clock System Control 1 Register

**Figure 3-4. CCSCTL1 Register**

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

**Table 3-3. CCSCTL1 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-1 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 0 | DIVCLK | RW | 1h | Clock division for CLKIN or X-OSC<br>0b = CLKIN or X-OSC is directly used for clock generation<br>1b = CLKIN or X-OSC is divided by two for clock generation |

### 3.3.3 CCSCTL4 Register

Compact Clock System Control 4 Register

**Figure 3-5. CCSCTL4 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 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 |

**Table 3-4. CCSCTL4 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-10 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 9-8 | SELAx | RW | 1h | Select the ACLK source<br>00b = HFCLK / DCO<br>01b = LFCLK / VLO<br>10b = CLKIN / X-OSC<br>11b = Reserved (defaults to LFCLK / VLO) |
| 7-6 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 5-4 | SELSx | RW | 0h | Select the SMCLK source<br>00b = HFCLK / DCO<br>01b = LFCLK / VLO<br>10b = CLKIN / X-OSC<br>11b = Reserved (defaults to LFCLK / VLO) |
| 3-2 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 1-0 | SELMx | RW | 0h | Select the MCLK source<br>00b = HFCLK / DCO<br>01b = LFCLK / VLO<br>10b = CLKIN / X-OSC<br>11b = Reserved (defaults to LFCLK / VLO) |

### 3.3.4 CCSCTL5 Register

Compact Clock System Control 5 Register

**Figure 3-6. CCSCTL5 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| 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 |

**Table 3-5. CCSCTL5 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-11 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 10-8 | DIVAx | RW | 3h | ACLK source divider<br>000b = Divide by 1<br>001b = Divide by 2<br>010b = Divide by 4<br>011b = Divide by 8<br>100b = Divide by 16<br>101b = Divide by 32<br>110b = Reserved, defaults to divide by 32<br>111b = Reserved, defaults to divide by 32 |
| 7 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 6-4 | DIVSx | RW | 3h | SMCLK source divider<br>000b = Divide by 1<br>001b = Divide by 2<br>010b = Divide by 4<br>011b = Divide by 8<br>100b = Divide by 16<br>101b = Divide by 32<br>110b = Reserved, defaults to divide by 32<br>111b = Reserved, defaults to divide by 32 |
| 3 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 2-0 | DIVMx | RW | 3h | MCLK source divider<br>000b = Divide by 1<br>001b = Divide by 2<br>010b = Divide by 4<br>011b = Divide by 8<br>100b = Divide by 16<br>101b = Divide by 32<br>110b = Reserved, defaults to divide by 32<br>111b = Reserved, defaults to divide by 32 |

### 3.3.5 CCSCTL6 Register

Compact Clock System Control 6 Register

**Figure 3-7. CCSCTL6 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| XTDRIVE | | XTS | XTBYPASS | Reserved | | | XTOFF |
| rw-1 | rw-1 | rw-0 | rw-0 | r0 | r0 | r0 | rw-1 |

**Table 3-6. CCSCTL6 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 7-6 | XTDRIVE | RW | 1h | The XT oscillator current can be adjusted to its drive needs.<br>00b = Lowest current consumption for XT1 LF mode. XT1 oscillator operating range in HF mode is 4 MHz to 8 MHz.<br>01b = Increased drive strength for XT1 LF mode. XT1 oscillator operating range in HF mode is 8 MHz to 16 MHz.<br>10b = Increased drive capability for XT1 LF mode. XT1 oscillator operating range in HF mode is 16 MHz to 24 MHz.<br>11b = Maximum drive capability and maximum current consumption for XT1 LF mode. XT1 oscillator operating range in HF mode is 24 MHz to 32 MHz. |
| 5 | XTS | RW | 0h | XT mode select<br>0b = Low-frequency mode<br>1b = High-frequency mode |
| 4 | XTBYPASS | RW | 0h | XT bypass select<br>0b = XT sourced internally<br>1b = XT sourced externally from pin |
| 3-1 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 0 | XTOFF | RW | 1h | XT off. This bit turns off the XT oscillator.<br>0b = XT is on if XT is selected via the port selection.<br>1b = XT is off if it is not used as a source for ACLK, MCLK, or SMCLK. |

### 3.3.6 CCSCTL7 Register

Compact Clock System Control 7 Register

**Figure 3-8. CCSCTL7 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | ENSTFCNT1 | Reserved | | | | | XOFFG |
| r0 | rw-(1) | r0 | r0 | r0 | r0 | r0 | rw-(1) |

**Table 3-7. CCSCTL7 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-7 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 6 | ENSTFCNT1 | RW | 1h | Enable start counter for XT1<br>0b = Startup fault counter disabled. Counter is cleared.<br>1b = Startup fault counter enabled. |
| 5-1 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 0 | XOFFG | RW | 1h | External Clock (CLKIN) or 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.<br>0b = No fault condition occurred after the last reset.<br>1b = X-OSC fault. An X-OSC fault occurred after the last reset. |

### 3.3.7 CCSCTL8 Register

Compact Clock System Control 8 Register

**Figure 3-9. CCSCTL8 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | SMCLKREQEN | MCLKREQEN | ACLKREQEN |
| r0 | r0 | r0 | r0 | r0 | rw-(1) | rw-(1) | rw-(1) |

**Table 3-8. CCSCTL8 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-3 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 2 | SMCLKREQEN | RW | 1h | SMCLK clock request enable. Peripheral modules can request the SMCLK to remain on.<br>0b = SMCLK is off if that is typical for the current low-power mode.<br>1b = SMCLK remains active if a peripheral is using it, even if it is typically off in the current low-power mode. |
| 1 | MCLKREQEN | RW | 1h | MCLK clock request enable. Peripheral modules can request the MCLK to remain on.<br>0b = MCLK is off if that is typical for the current low-power mode.<br>1b = MCLK remains active if a peripheral is using it, even if it is typically off in the current low-power mode. |
| 0 | ACLKREQEN | RW | 1h | ACLK clock request enable. Peripheral modules can request the ACLK to remain on. See the device-specific data sheet to determine which peripherals support the clock request.<br>0b = ACLK is off if that is typical for the current low-power mode.<br>1b = ACLK remains active if a peripheral is using it, even if it is typically off in the current low-power mode. |

Chapter 4

SLAU506–December 2014

# CPU

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

**Topic** **Page**

## 4.1 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 4-1.

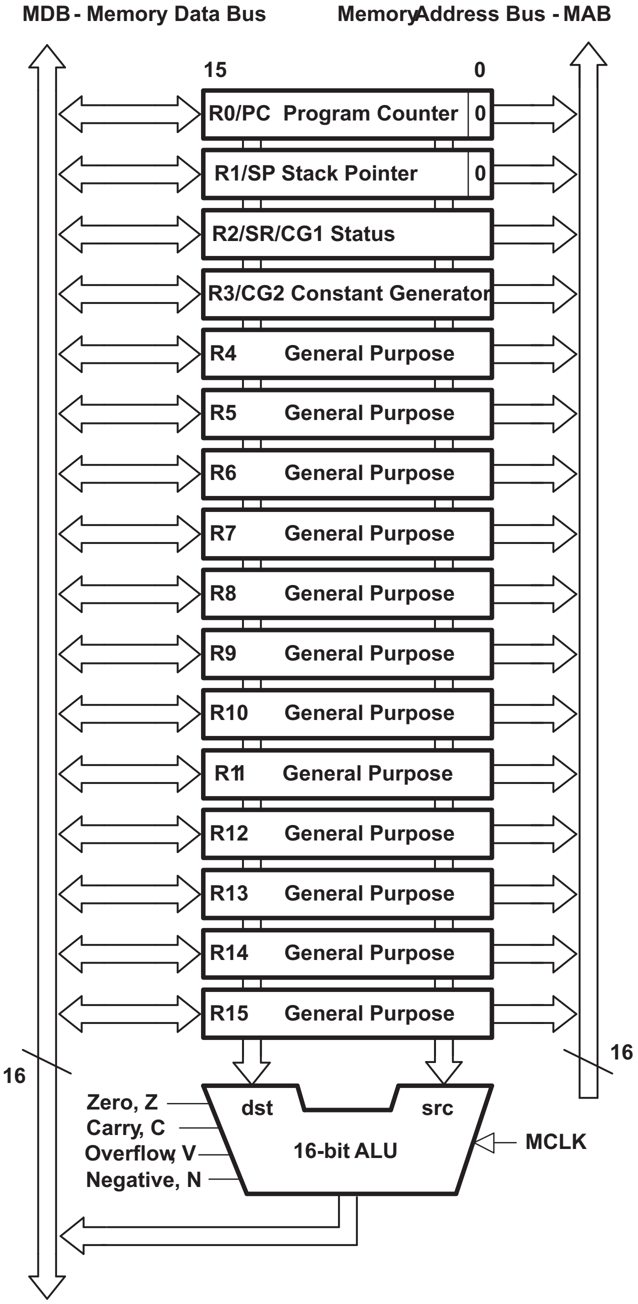


**Figure 4-1. CPU Block Diagram**

## 4.2 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.

### *4.2.1 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 4-2 shows the program counter.

**Figure 4-2. 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
```

### 4.2.2 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 4-3 shows the SP. The SP is initialized into RAM by the user, and is aligned to even addresses.

Figure 4-4 shows stack usage.

**Figure 4-3. 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
```

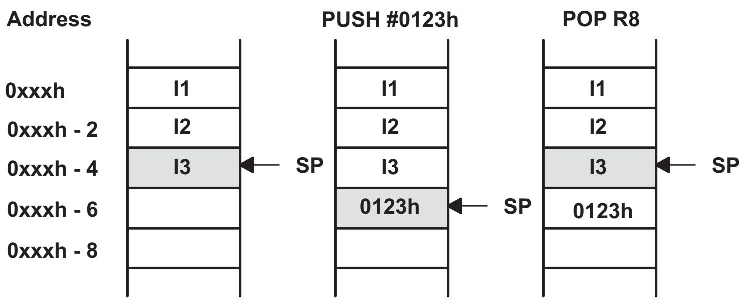
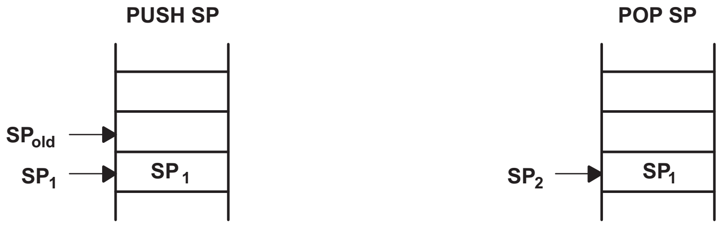


**Figure 4-4. Stack Usage**

The special cases of using the SP as an argument to the PUSH and POP instructions are described and shown in Figure 4-5.



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 4-5. PUSH SP - POP SP Sequence**

### 4.2.3 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 4-6 shows the SR bits.

**Figure 4-6. Status Register Bits**

| 15 | 14 | 13 | 12 | 11 | 10 | 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 4-1 describes the status register bits.

**Table 4-1. 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:<br>Positive + Positive = Negative<br>Negative + Negative = Positive<br>Otherwise reset |
| | `SUB(.B),SUBC(.B),CMP(.B)` | Set when:<br>Positive – Negative = Negative<br>Negative – Positive = Positive<br>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.<br>Word operation: N is set to the value of bit 15 of the result.<br>Byte operation: N is set to the value of bit 7 of the result. | |
| Z | Zero bit. 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. | |

### 4.2.4 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 4-2.

**Table 4-2. Values of Constant Generators CG1, CG2**

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

### 4.2.4.1 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
```

## 4.2.5 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 4-7.

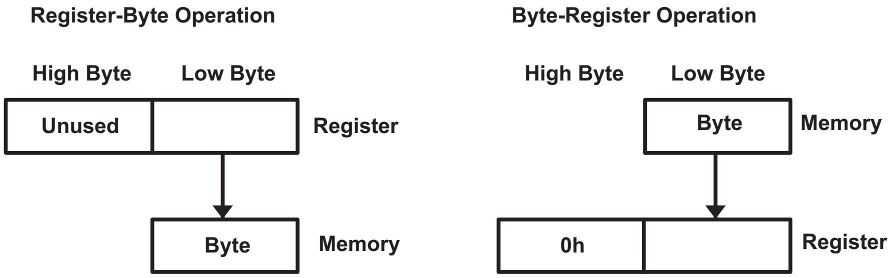


**Figure 4-7. Register-Byte/Byte-Register Operations**

| 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) |

## 4.3 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 4-3 describe the contents of the As (source) and Ad (destination) mode bits.

**Table 4-3. 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.

### *4.3.1 Register Mode*

The register mode is described in Table 4-4.

**Table 4-4. Register Mode Description**

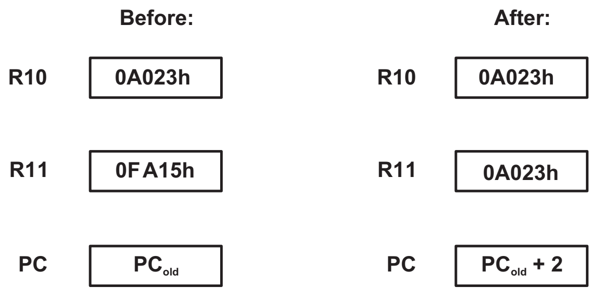
| Assembler Code | Content of ROM |
|---|---|
| `MOV    R10,R11` | `MOV   R10,R11` |

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.

### 4.3.2 Indexed Mode

The indexed mode is described in Table 4-5.

**Table 4-5. Indexed Mode Description**

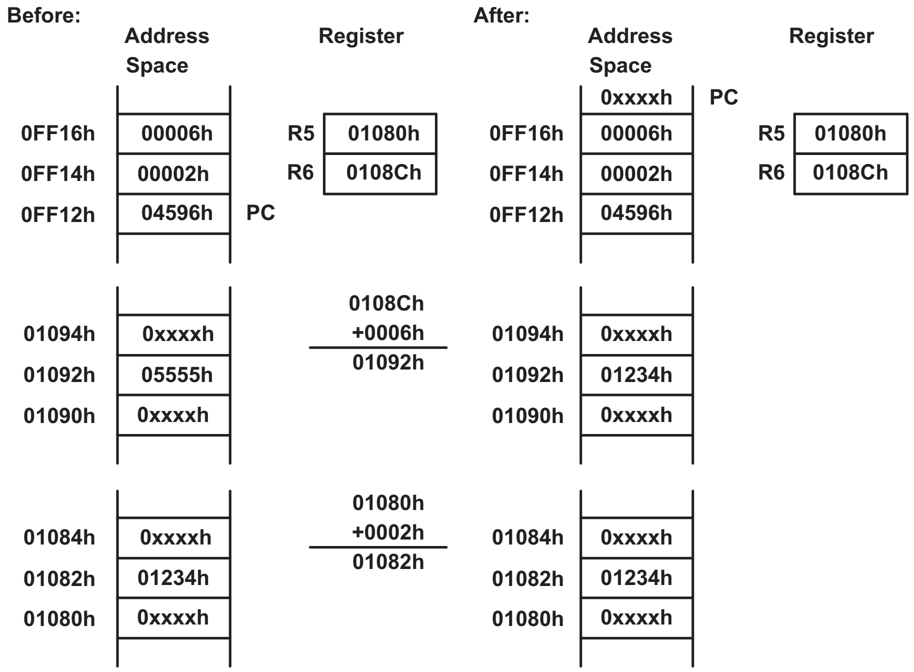
| Assembler Code | Content of ROM |
|---|---|
| `MOV 2(R5),6(R6)` | `MOV X(R5),Y(R6)` |
| | X = 2 |
| | Y = 6 |

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);`



### 4.3.3 Symbolic Mode

The symbolic mode is described in Table 4-6.

**Table 4-6. Symbolic Mode Description**

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

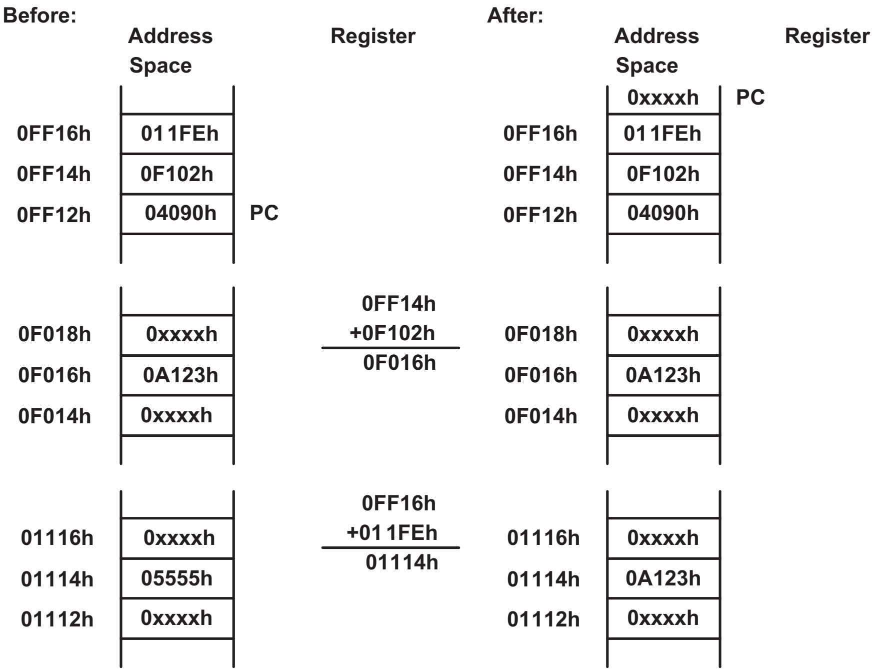
Length: Two or three words

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
```



### 4.3.4 Absolute Mode

The absolute mode is described in Table 4-7.

**Table 4-7. Absolute Mode Description**

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

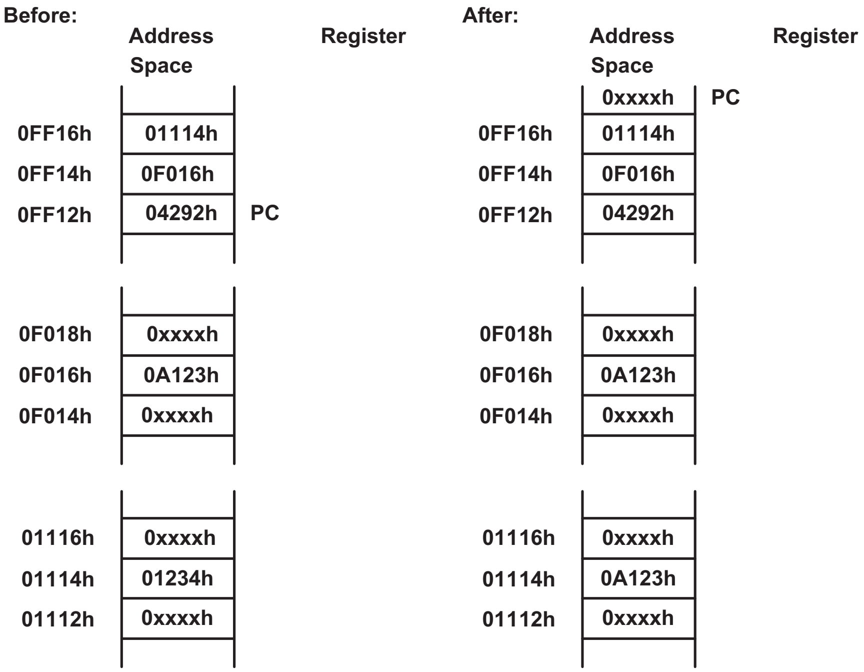
Length: Two or three words

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

Comment: Valid for source and destination

Example:

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



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

### 4.3.5 Indirect Register Mode

The indirect register mode is described in Table 4-8.

**Table 4-8. Indirect Mode Description**

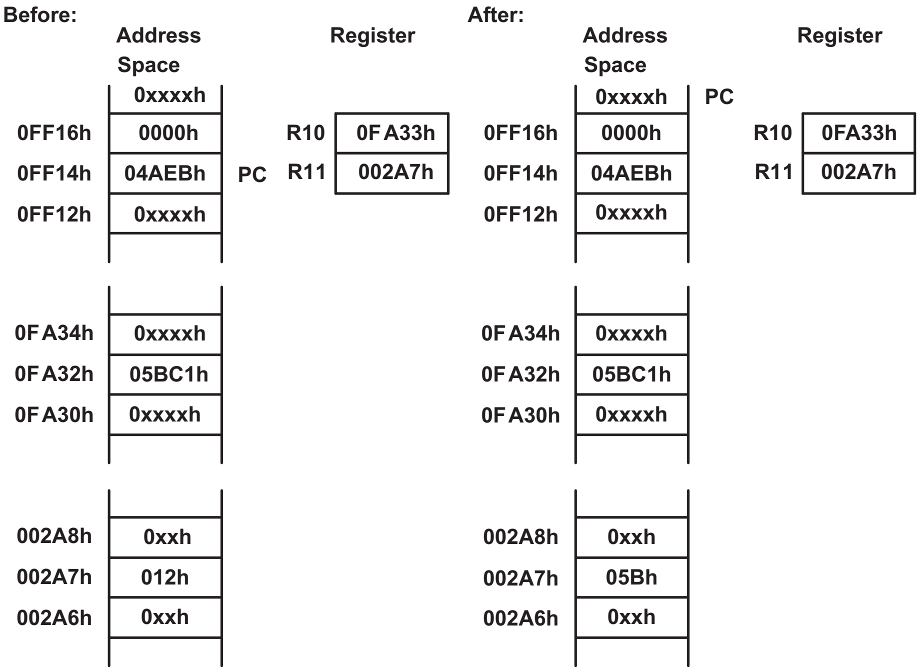
| Assembler Code | Content of ROM |
|---|---|
| `MOV @R10,0(R11)` | `MOV @R10,0(R11)` |

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)`



### 4.3.6 Indirect Autoincrement Mode

The indirect autoincrement mode is described in Table 4-9.

**Table 4-9. Indirect Autoincrement Mode Description**

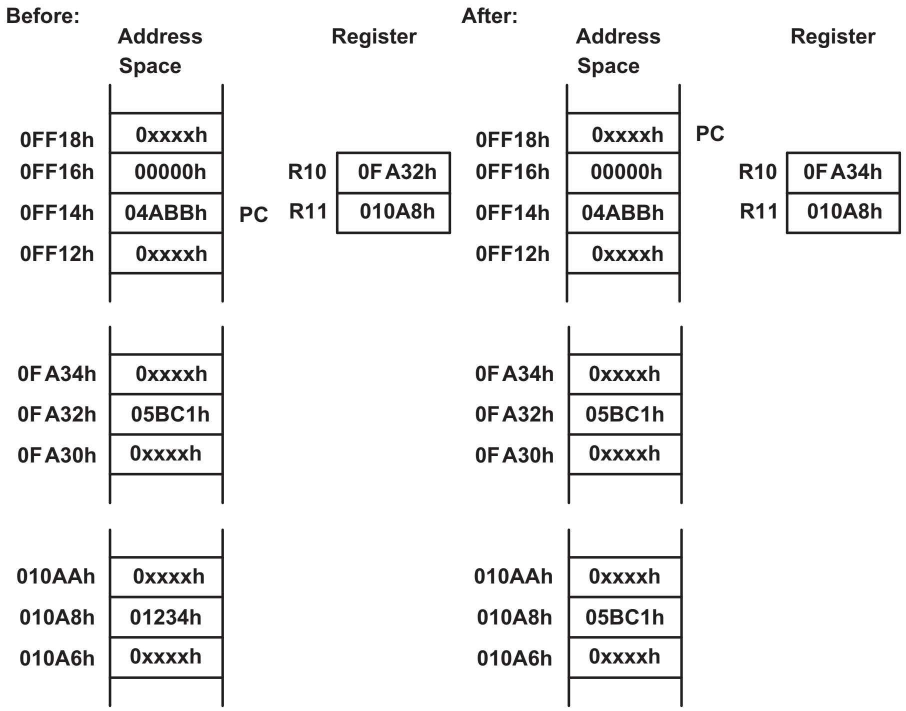
| Assembler Code | Content of ROM |
|---|---|
| `MOV @R10+,0(R11)` | `MOV @R10+,0(R11)` |

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 4-8.

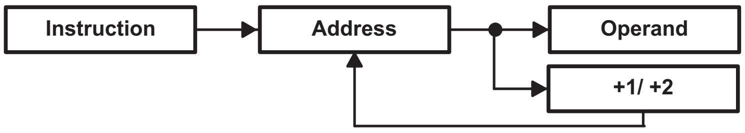


**Figure 4-8. Operand Fetch Operation**

### 4.3.7 Immediate Mode

The immediate mode is described in Table 4-10.

**Table 4-10. Immediate Mode Description**

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

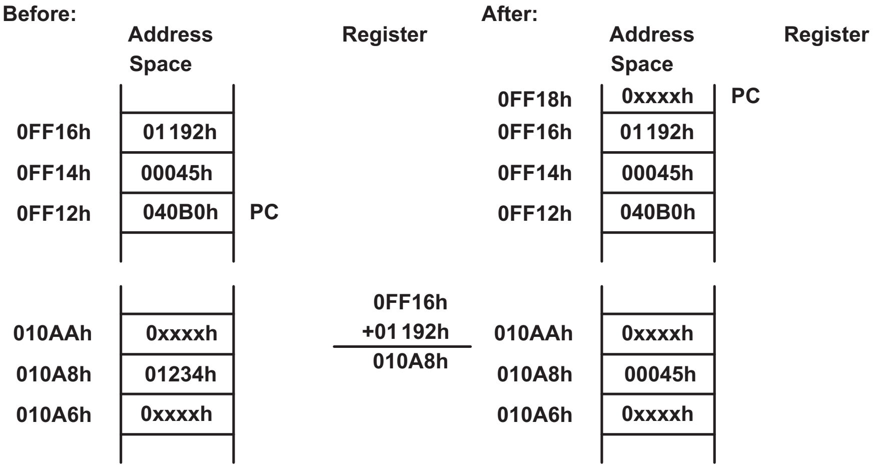
Length: Two or three words

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`



## 4.4 Instruction Set

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

There are three core-instruction formats:

- Dual-operand
- Single-operand
- Jump

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

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

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

**NOTE: Destination Address**

Destination addresses are valid anywhere in the memory map. However, when using an instruction that modifies the contents of the destination, the user must ensure the destination address is writable. 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.

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

Figure 4-9 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 4-9. Double Operand Instruction Format**

Table 4-11 lists and describes the double operand instructions.

**Table 4-11. Double Operand Instructions**

| Mnemonic | S-Reg,<br>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.

### 4.4.2 Single-Operand (Format II) Instructions

Figure 4-10 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 4-10. Single Operand Instruction Format**

Table 4-12 lists and describes the single operand instructions.

**Table 4-12. Single Operand Instructions**

| Mnemonic | S-Reg,<br>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.

### *4.4.3 Jumps*

Figure 4-11 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 4-11. Jump Instruction Format**

Table 4-13 lists and describes the jump instructions

**Table 4-13. 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:

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

### 4.4.4 Instruction Details

The following sections describe each of the MSP430 instructions.

### 4.4.4.1 ADC

***ADC[.W]** Add carry to destination

***ADC.B** Add carry to destination

**Syntax**

```
ADC dst  or  ADC.W dst
ADC.B dst
```

**Operation** dst + C → dst

**Emulation** `ADDC #0,dst ADDC.B #0,dst`

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

**Status 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
```

### 4.4.4.2 ADD

**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
```

### 4.4.4.3 ADDC

**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
```

#### 4.4.4.4 AND

**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
```

### 4.4.4.5 BIC

**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
```

### 4.4.4.6 BIS

**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
```

### 4.4.4.7 BIT

**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
```

### 4.4.4.8 BR, BRANCH

***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
```

### 4.4.4.9 CALL

**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
```

#### 4.4.4.10 CLR

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

***CLR.B** Clear destination

**Syntax**

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

**Operation** 0 → dst

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

**Description** The destination operand is cleared.

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

**Example** RAM word TONI is cleared.

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

**Example** Register R5 is cleared.

```
CLR    R5
```

**Example** RAM byte TONI is cleared.

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

### 4.4.4.11 CLRC

***CLRC** Clear carry bit

**Syntax** `CLRC`

**Operation** 0 → C

**Emulation** `BIC #1,SR`

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

**Status Bits**
N: Not affected

Z: Not affected

C: Cleared

V: Not affected

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

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

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

### 4.4.4.12 CLRN

***CLRN** Clear negative bit

**Syntax** `CLRN`

**Operation** 0 → N

or

(.NOT.src .AND. dst → dst)

**Emulation** `BIC #4,SR`

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

**Status Bits** N: Reset to 0

Z: Not affected

C: Not affected

V: Not affected

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

**Example** The Negative bit in the 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
```

### 4.4.4.13 CLRZ

***CLRZ** Clear zero bit

**Syntax** `CLRZ`

**Operation** 0 → Z

or

(.NOT.src .AND. dst → dst)

**Emulation** `BIC #2,SR`

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

**Status Bits** N: Not affected

Z: Reset to 0

C: Not affected

V: Not affected

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

**Example** The zero bit in the status register is cleared.

```
CLRZ
```

### 4.4.4.14 CMP

**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
```

### 4.4.4.15 DADC

***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
```

### 4.4.4.16 DADD

**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
```

### 4.4.4.17 DEC

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

***DEC.B** Decrement destination

**Syntax**

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

**Operation** dst - 1 → dst

**Emulation**

```
SUB   #1,dst
SUB.B #1,dst
```

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

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

Z: Set if dst contained 1, reset otherwise

C: Reset if dst contained 0, set otherwise

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

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

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

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

**Example** R10 is decremented by 1.

```
     DEC     R10    ; Decrement R10
; Move a block of 255 bytes from memory location starting with EDE
; to memory location starting with TONI. Tables should not overlap:
; start of destination address TONI must not be within the range
; EDE to EDE+0FEh
     MOV     #EDE,R6
     MOV     #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 4-12.

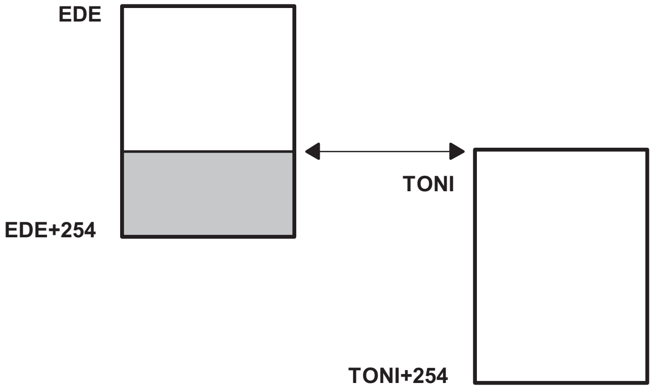


**Figure 4-12. Decrement Overlap**

### 4.4.4.18 DECD

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

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

**Syntax**

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

**Operation** dst - 2 → dst

**Emulation**

```
SUB   #2,dst
SUB.B #2,dst
```

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

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

Z: Set if dst contained 2, reset otherwise

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

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

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

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

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

**Example** R10 is decremented by 2.

```
      DECD   R10      ; Decrement R10 by two
; Move a block of 255 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
```

### 4.4.4.19 DINT

***DINT** Disable (general) interrupts

**Syntax** `DINT`

**Operation** 0 → GIE

or

(0FFF7h .AND. SR → SR / .NOT.src .AND. dst → dst)

**Emulation** `BIC #8,SR`

**Description** All interrupts are disabled.

The constant 08h is inverted and logically ANDed with the 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.

### 4.4.4.20 EINT

***EINT** Enable (general) interrupts

**Syntax** `EINT`

**Operation** 1 → GIE

or

(0008h .OR. SR → SR / .src .OR. dst → dst)

**Emulation** `BIS #8,SR`

**Description** All interrupts are enabled.

The constant #08h and the 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.

### 4.4.4.21 INC

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

***INC.B** Increment destination

**Syntax**

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

**Operation** dst + 1 → dst

**Emulation**

```
ADD #1,dst
```

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

**Status Bits**

N: Set if result is negative, reset if positive

Z: Set if dst contained 0FFFFh, reset otherwise

Set if dst contained 0FFh, reset otherwise

C: Set if dst contained 0FFFFh, reset otherwise

Set if dst contained 0FFh, reset otherwise

V: Set if dst contained 07FFFh, reset otherwise

Set if dst contained 07Fh, reset otherwise

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

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

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

### 4.4.4.22 INCD

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

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

**Syntax**

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

**Operation** dst + 2 → dst

**Emulation**

```
ADD   #2,dst
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
```

### 4.4.4.23 INV

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

***INV.B** Invert destination

**Syntax**

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

**Operation** .NOT.dst → dst

**Emulation**

```
XOR   #0FFFFh,dst
XOR.B #0FFh,dst
```

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

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

Z: Set if dst contained 0FFFFh, reset otherwise

Set if dst contained 0FFh, reset otherwise

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

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
```

### 4.4.4.24 JC, JHS

**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
```

### 4.4.4.25 JEQ, JZ

**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
......
```

### 4.4.4.26 JGE

**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
......
......
```

### 4.4.4.27 JL

**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
......
......
```

### 4.4.4.28 JMP

|  |  |
|---|---|
| **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. |

### 4.4.4.29 JN

**JN** Jump if negative

**Syntax** `JN label`

**Operation** if N = 1: PC + 2 ×offset → PC

if N = 0: execute following instruction

**Description** The negative bit (N) 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
      ......
      ......
      ......
```

#### 4.4.4.30 JNC, JLO

**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
```

### 4.4.4.31 JNE, JNZ

**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
```

### 4.4.4.32 MOV

**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
       ......
       ......
```

### 4.4.4.33 NOP

***NOP** No operation

**Syntax** `NOP`

**Operation** None

**Emulation** `MOV #0, R3`

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

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

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 occurs with the watchdog timer (address 120h), because the security key was not used.

### 4.4.4.34 POP

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

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

**Syntax**

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

**Operation**

@SP → temp

SP + 2 → SP

temp → dst

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

**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.

### 4.4.4.35 PUSH

**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.

### 4.4.4.36 RET

***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.

### 4.4.4.37 RETI

**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 4-13 illustrates the main program interrupt.

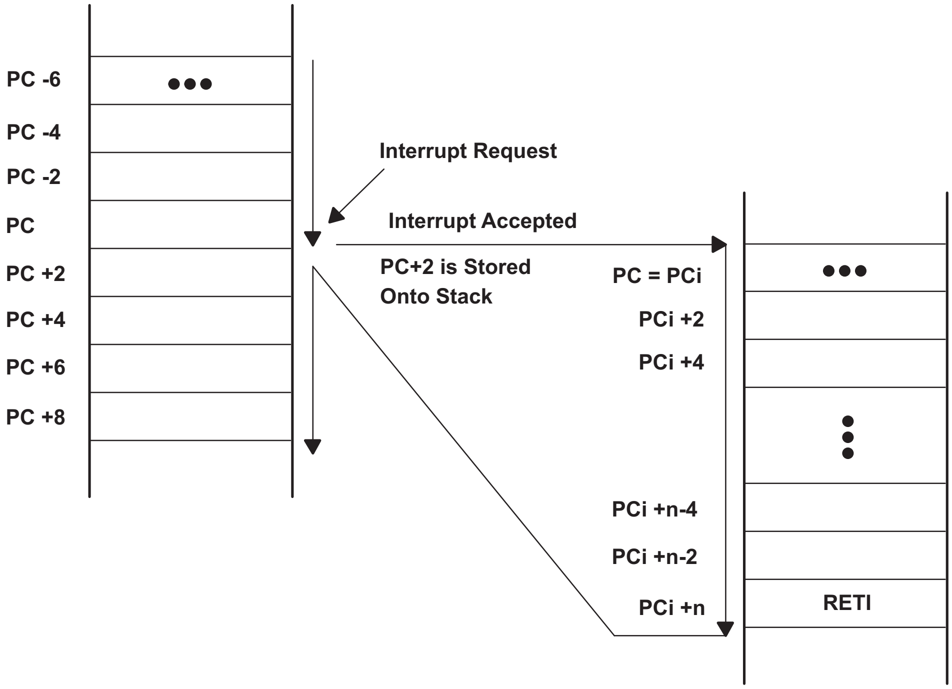


**Figure 4-13. Main Program Interrupt**

### 4.4.4.38 RLA

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

***RLA.B** Rotate left arithmetically

**Syntax**

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

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

**Emulation**

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

**Description** The destination operand is shifted left one position as shown in Figure 4-14. 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.

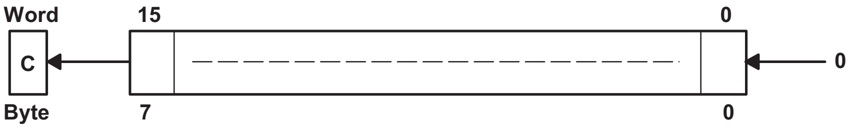


**Figure 4-14. Destination Operand – Arithmetic Shift Left**

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
```

### 4.4.4.39 RLC

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

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

**Syntax**

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

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

**Emulation**

```
ADDC dst,dst
```

**Description** The destination operand is shifted left one position as shown in Figure 4-15. The carry bit (C) is shifted into the LSB and the MSB is shifted into the carry bit (C).

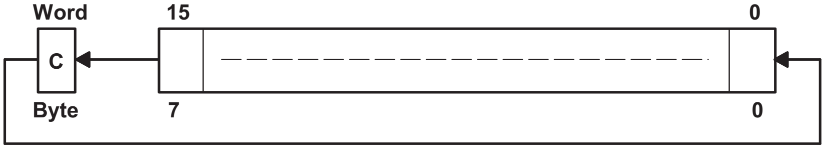


**Figure 4-15. Destination Operand-Carry Left Shift**

**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`

### 4.4.4.40 RRA

**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 4-16. The MSB is shifted into the MSB, the MSB is shifted into the MSB-1, and the LSB+1 is shifted into the LSB.

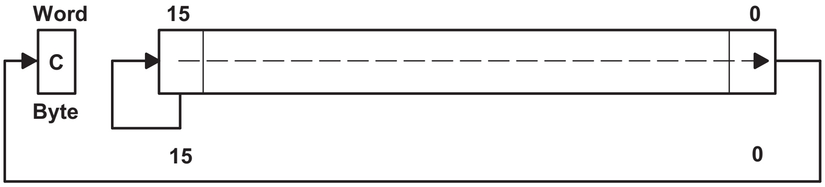


**Figure 4-16. Destination Operand – Arithmetic Right Shift**

**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
......
```

### 4.4.4.41 RRC

**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 4-17. The carry bit (C) is shifted into the MSB, the LSB is shifted into the carry bit (C).

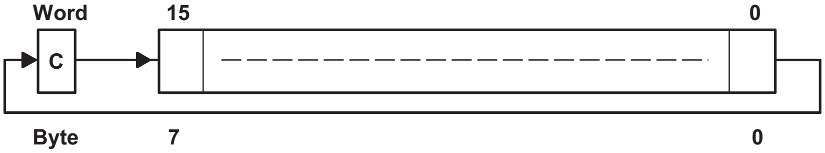


**Figure 4-17. Destination Operand—Carry Right Shift**

**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
```

#### 4.4.4.42 SBC

***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 |

#### 4.4.4.43 SETC

***SETC** Set carry bit

**Syntax** `SETC`

**Operation** 1 → C

**Emulation** `BIS #1,SR`

**Description** The carry bit (C) is set.

**Status Bits** N: Not affected

Z: Not affected

C: Set

V: Not affected

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

**Example** Emulation of the decimal subtraction:

Subtract R5 from R6 decimally

Assume that R5 = 03987h and R6 = 04137h

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

### 4.4.4.44 SETN

***SETN** Set negative bit

**Syntax** `SETN`

**Operation** 1 → N

**Emulation** `BIS #4,SR`

**Description** The negative bit (N) is set.

**Status Bits** N: Set

Z: Not affected

C: Not affected

V: Not affected

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

### 4.4.4.45 SETZ

| | |
|---|---|
| ***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<br>Z: Set<br>C: Not affected<br>V: Not affected |
| **Mode Bits** | OSCOFF, CPUOFF, and GIE are not affected. |

#### 4.4.4.46 SUB

**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 |

### 4.4.4.47 SUBC, SBB

**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
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 |

#### 4.4.4.48 SWPB

**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 4-18.

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

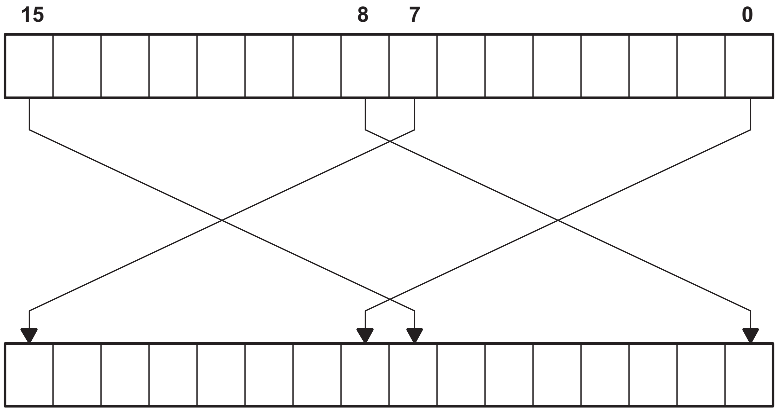


**Figure 4-18. Destination Operand Byte Swap**

**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
```

#### 4.4.4.49 SXT

**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 4-19.

**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.

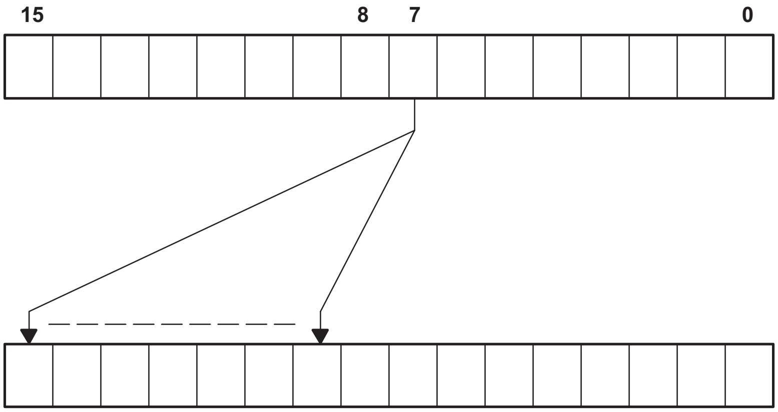


**Figure 4-19. Destination Operand Sign Extension**

**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
```

### 4.4.4.50 TST

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

***TST.B** Test destination

**Syntax**

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

**Operation**

dst + 0FFFFh + 1

dst + 0FFh + 1

**Emulation**

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

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

**Status Bits**

N: Set if destination is negative, reset if positive

Z: Set if destination contains zero, reset otherwise

C: Set

V: Reset

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

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

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

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

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

### 4.4.4.51 XOR

**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
```

## 4.4.5 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.

### 4.4.5.1 Interrupt and Reset Cycles

Table 4-14 lists the CPU cycles for interrupt overhead and reset.

**Table 4-14. 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 | - |

### 4.4.5.2 Format-II (Single Operand) Instruction Cycles and Lengths

Table 4-15 lists the length and CPU cycles for all addressing modes of format-II instructions.

**Table 4-15. Format-II Instruction Cycles and Lengths**

| | No. of Cycles | | | | |
|---|---|---|---|---|---|
| Addressing Mode | RRA, RRC<br>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.

### 4.4.5.3 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.

### 4.4.5.4 Format-I (Double Operand) Instruction Cycles and Lengths

Table 4-16 lists the length and CPU cycles for all addressing modes of format-I instructions.

**Table 4-16. 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` |

### 4.4.6 Instruction Set Description

The instruction map is shown in Figure 4-20 and the complete instruction set is summarized in Table 4-17.

| | 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 4-20. Core Instruction Map**

**Table 4-17. 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 4-17. 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 [1] | | No operation | | - | - | - | - |
| POP(.B) [1] | 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 [1] | | Return from subroutine | @SP → PC, SP + 2 → SP | - | - | - | - |
| RETI | | Return from interrupt | | * | * | * | * |
| RLA(.B) [1] | dst | Rotate left arithmetically | | * | * | * | * |
| RLC(.B) [1] | 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

Chapter 5

SLAU506–December 2014

# FRAM Controller (FRCTL)

This chapter describes the operation of the FRAM memory controller.

**Topic** **Page**

## 5.1 FRAM Introduction

FRAM memory is a nonvolatile memory that reads and writes like standard SRAM. The FRAM memory features include:

- Byte or word write access
- Automatic and programmable wait state control with independent wait state settings for access and cycle times
- Error Correction Code (ECC) with bit error correction, extended bit error detection, and flag indicators
- Cache for fast read
- Power control for disabling FRAM if it is not used

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

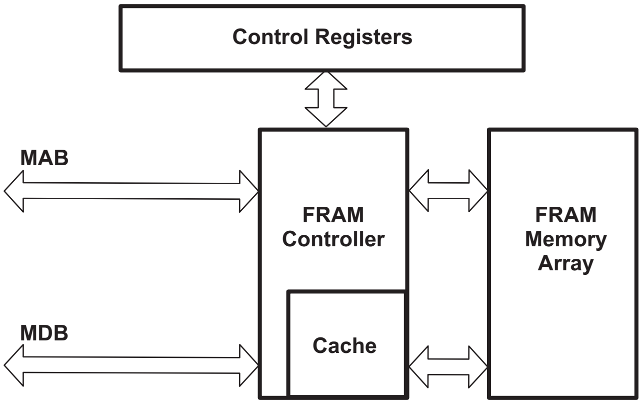


**Figure 5-1. FRCTL Block Diagram**

## 5.2 FRCTL Module Operation

The FRAM module can be read in a similar fashion to SRAM and needs no special requirements. Similarly, any writes to unprotected segments can be written in the same fashion as SRAM.

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

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

## 5.3 Programming FRAM Memory Devices

There are two options for programming an RF430 FRAM device. All options support in-system programming.

- Program using the JTAG interface
- Program using a custom solution (SPI, I$^2$C, or RF)

### 5.3.1 Programming FRAM Memory Through JTAG

Devices can be programmed through the JTAG port. The JTAG interface requires access to TDI, TDO, TMS, TCK, TEST, ground, and optionally VCC and $\overline{\text{RST}}$/NMI. For more details, see the *MSP430 Programming Via the JTAG Interface User's Guide* (SLAU320).

### 5.3.2 Programming FRAM Memory By Custom Solution

The ability of the CPU to write to its own FRAM memory allows for in-system and external custom programming solutions. The user can choose to provide data to the device through any means available (for example, SPI, I$^2$C, or RF). User-developed software can receive the data and program the FRAM memory. Because this type of solution is developed by the user, it can be completely customized to fit the application needs for programming or updating the FRAM memory.

## 5.4 Wait State Control

The system clock for the CPU or DMA may exceed the FRAM access and cycle time requirements. For these scenarios, a wait state generator mechanism is implemented. When required, the system clock or CPU is held until the FRAM access and cycle time constraints are met.

The complete FRAM cycle time is defined by the access time can be defined in the NACCESS[2:0] control bits. If the clock is set higher than the maximum FRAM access frequency, NACCESS[2:0] have to be set properly to permit correct FRAM access.

The NACCESS bits can be used to define an integer number of CPU cycles required for access time described in the data sheet. For NACCESS[2:0] only 0b and 1b are valid values.

### 5.4.1 Wait State and Cache Hit

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

### 5.4.2 Safe Access

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

Safe Access is active when the user configures the NACCESS[2:0] bits to a value that does not meet the required FRAM timing for the given clock setting. In this case, the Safe Access logic ensures the correct timing for the access.

## 5.5 FRAM ECC

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

## 5.6 FRAM Write Back

All reads from FRAM requires a write back of the previously read content. This write back is performed under all circumstances without any interaction from a user.

## 5.7 FRAM Power Control

The FRAM power control feature is not available.

## 5.8 FRAM Cache

The FRAM controller implements a read cache to provide a speed benefit when running the CPU at higher speeds than the FRAM supports without wait states. The cache implemented is a 2-way associative cache with 4 cache lines of 64 bit size. Memory read accesses on consecutive addresses can be executed without wait states when they are within the same cache line.

## 5.9 FRCTL Registers

The FRCTL registers and their address offsets are listed in Table 5-1 . The base address of the FRCTL module can be found in the device-specific data sheet.

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

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

**Table 5-1. FRCTL Registers**

| Offset | Acronym | Register Name | Type | Access | Reset | Section |
|---|---|---|---|---|---|---|
| 00h | FRCTL0 | FRAM Controller Control 0 | Read/write | Word | 9608h | Section 5.9.1 |
| 00h | FRCTL0_L | | Read/Write | Byte | 08h | |
| 01h | FRCTL0_H | | Read/Write | Byte | 96h | |
| 04h | GCCTL0 | General Control 0 | Read/write | Word | 0006h | Section 5.9.2 |
| 04h | GCCTL0_L | | Read/Write | Byte | 06h | |
| 05h | GCCTL0_H | | Read/Write | Byte | 00h | |
| 06h | GCCTL1 | General Control 1 | Read/write | Word | 0000h | Section 5.9.3 |
| 06h | GCCTL1_L | | Read/Write | Byte | 00h | |
| 07h | GCCTL1_H | | Read/Write | Byte | 00h | |

### *5.9.1 FRCTL0 Register*

FRAM Controller Control Register 0

**Figure 5-2. FRCTL0 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| FRCTLPW | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | NACCESS | | | Reserved | | | |
| r-0 | rw-[0] | rw-[0] | rw-[0] | r-0 | r-0 | r-0 | r-0 |

**Table 5-2. FRCTL0 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | FRCTLPW | RW | 96h | FRCTLPW Password. Always read as 96h. Must be written as A5h or a PUC is generated on word write. After a correct password is written and MPU register access is enabled, a wrong password write in byte mode disables the access and no PUC is generated. |
| 7 | Reserved | R | 0h | Reserved. Always read 0. |
| 6-4 | NACCESS | RW | 0h | Wait state generator access time control. Each wait state adds a N integer multiple increase of the IFCLK period where N = 0 through 7. N = 0 implies no wait states.<br>000b = No Wait States<br>001b = 1 Wait State<br>010b = Reserved<br>011b = Reserved<br>100b = Reserved<br>101b = Reserved<br>110b = Reserved<br>111b = Reserved |
| 3-0 | Reserved | R | 0h | Reserved. Always read 0. |

### 5.9.2 GCCTL0 Register

General Control Register 0

**Figure 5-3. GCCTL0 Register**

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

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UBDRSTEN | UBDIE | CBDIE | Reserved | | | | |
| rw-[0] | rw-[0] | rw-[0] | r-0 | r-0 | r-0 | r-0 | r-0 |

**Table 5-3. GCCTL0 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | Reserved | R | 0h | Reserved. Always read 0. |
| 7 | UBDRSTEN | RW | 0h | Enable Power Up Clear (PUC) reset if FRAM uncorrectable bit error detected.<br>The bits UBDRSTEN and UBDIE are mutual exclusive and are not allowed to be set simultaneously. Only one error handling can be selected at one time.<br>0b = PUC not initiated on uncorrectable bit detection flag.<br>1b = PUC initiated on uncorrectable bit detection flag. Generates vector in SYSRSTIV. |
| 6 | UBDIE | RW | 0h | Enable NMI event if uncorrectable bit error detected.<br>The bits UBDRSTEN and UBDIE are mutual exclusive and are not allowed to be set simultaneously. Only one error handling can be selected at one time.<br>0b = Uncorrectable bit detection interrupt disabled.<br>1b = Uncorrectable bit detection interrupt enabled. Generates vector in SYSSNIV. |
| 5 | CBDIE | RW | 0h | Enable NMI event if correctable bit error detected.<br>0b = Correctable bit detection interrupt disabled.<br>1b = Correctable bit detection interrupt enabled. Generates vector in SYSSNIV. |
| 4-0 | Reserved | R | 0h | Reserved. Always read 0. |

### 5.9.3 GCCTL1 Register

General Control Register 1

**Figure 5-4. GCCTL1 Register**

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

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | UBDIFG | CBDIFG | Reserved |
| r-0 | r-0 | r-0 | r-0 | r-0 | rw-[0] | rw-[0] | r-0 |

**Table 5-4. GCCTL1 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-3 | Reserved | R | 0h | Reserved. Always read 0. |
| 2 | UBDIFG | RW | 0h | FRAM uncorrectable bit error flag. This interrupt flag is set if an uncorrectable bit error has been detected in the FRAM memory error detection logic. This bit is cleared by software or by reading the system NMI vector word SYSSNIV if it is the highest pending interrupt flag. This bit is write 0 only and write 1 has no effect.<br>0b = No interrupt pending<br>1b = Interrupt pending. Can be cleared by user or by reading SYSSNIV. |
| 1 | CBDIFG | RW | 0h | FRAM correctable bit error flag. This interrupt flag is set if a correctable bit error has been detected and corrected in the FRAM memory error detection logic. This bit is cleared by software or by reading the system NMI vector word SYSSNIV if it is the highest pending interrupt flag. This bit is write 0 only and write 1 has no effect.<br>0b = No interrupt pending<br>1b = Interrupt pending. Can be cleared by user or by reading SYSSNIV |
| 0 | Reserved | R | 0h | Reserved. Always read 0. |

Chapter 6

SLAU506–December 2014

# Versatile I/O Port

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

**Topic** **Page**

## 6.1 Versatile I/O Ports (VersaPorts) and Digital I/O Ports

RF430 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.

## 6.2 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 the ports that are 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 pull-up or pull-down 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. The port pair P1+P2 is called port PA, the port pair P3+P4 is called port PB, and so on. 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, and so on, behave similarly. The unused bits of ports that are not fully equipped are don't 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, and so on, 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 6-1 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.

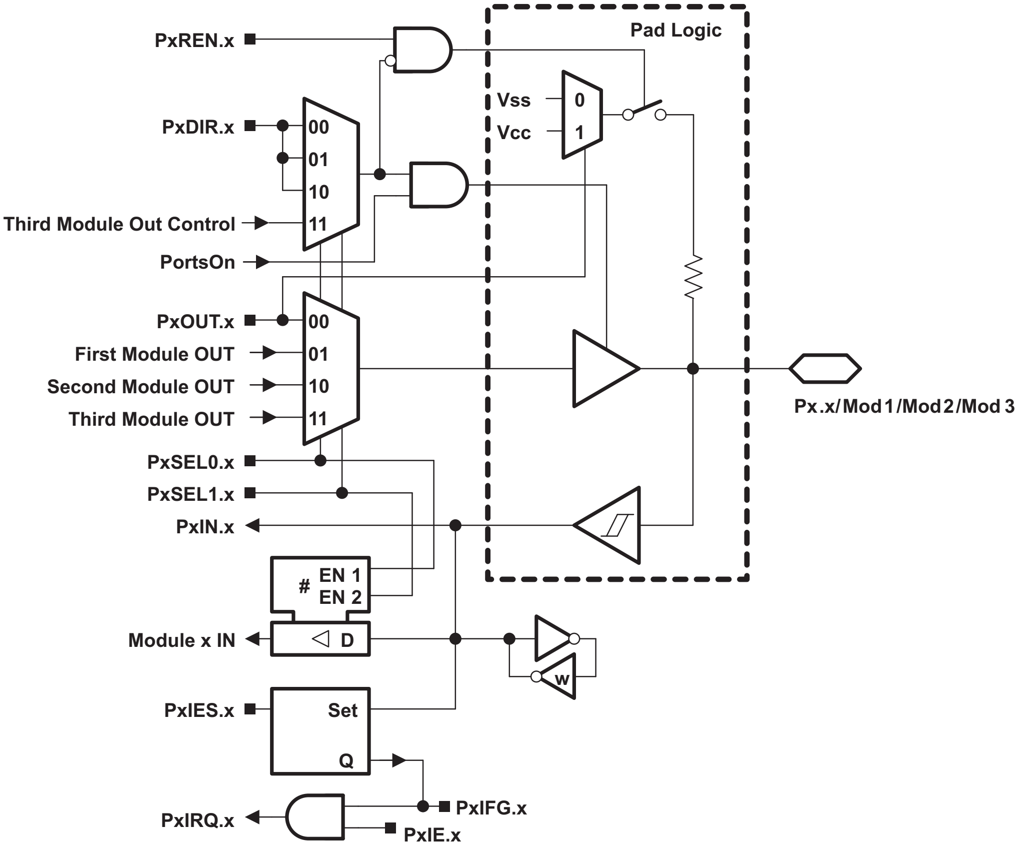


Figure 6-1. Typical Schematic of Port Logic

## 6.3 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 described in the following sections.

### 6.3.1 *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. See Table 6-1 for a summary of I/O configuration settings.

Bit = 0: The input is low

Bit = 1: The input is high

### 6.3.2 *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. See Table 6-1 for a summary of I/O configuration settings.

Bit = 0: The output is low

Bit = 1: The output is high

If the pin's pullup or pulldown resistor is enabled (see PxREN), 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

### 6.3.3 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. See Table 6-1 for a summary of I/O configuration settings.

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

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

### 6.3.4 Pullup or 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. See Table 6-1 for a summary of I/O configuration settings.

Bit = 0: Pullup or pulldown resistor disabled

Bit = 1: Pullup or pulldown resistor enabled

Table 6-1 summarizes the use of PxDIR.x, PxREN.x, and PxOUT.x for proper I/O configuration.

**Table 6-1. I/O Configuration**

| PxDIR.x | PxREN.x | PxOUT.x | 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 |

### 6.3.5 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 6-2 shows how to select the various module functions.

**Table 6-2. I/O Function Selection**

| 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 | Tertiary module function is selected |

Setting the PxSELx.x bits to a module function does not automatically set the pin direction. Other peripheral module functions may require the PxDIR bits to be configured according to the direction needed for the module function (see the pin schematics in the device-specific 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 PxSELx.x is other than 00, the internal input signal follows the signal at the pin for all connected modules. However, if PxSELx.x = 00, the input to the peripherals maintain the value of the input signal at the device pin before the PxSELx.x bits were reset.

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

Bit = 0: No interrupt is pending

Bit = 1: An interrupt is pending

Only transitions, not static levels, cause interrupts. If any PxIFG.x flag becomes set during a Px interrupt service routine or is set after the RETI instruction of a Px interrupt service routine is executed, the set PxIFG.x 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 generates another interrupt.

#### 6.3.6.1 P1IV Software Example

Example 6-1 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 CPU cycles that are required for each instruction. The software overhead for different interrupt sources includes interrupt latency and return-from-interrupt cycles, but not the task handling itself.

***Example 6-1. P1IV Software Example***

```
;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
```

*Example 6-1. P1IV Software Example (continued)*

```
          ...            ; 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
```

#### 6.3.6.2 Interrupt Edge Select Registers PxIES

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

Bit = 0: The PxIFG.x flag is set with a low-to-high transition

Bit = 1: The PxIFG.x flag is set with a high-to-low transition

**NOTE: Writing to PxIES**

Writing to PxIES can result in setting the corresponding interrupt flags (see Table 6-3).

**Table 6-3. Writing to PxIES**

| PxIES.x | PxIN.x | PxIFG.x |
|---|---|---|
| 0 → 1 | 0 | May be set |
| 0 → 1 | 1 | Unchanged |
| 1 → 0 | 0 | Unchanged |
| 1 → 0 | 1 | May be set |

#### 6.3.6.3 Interrupt Enable PxIE

Each PxIE bit enables the associated PxIFG interrupt flag.

Bit = 0: The interrupt is disabled

Bit = 1: The interrupt is enabled

### *6.3.7 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, because the pin is unconnected. Alternatively, the integrated pullup or pulldown resistor can be enabled by setting the PxREN bit of the unused pin to prevent the floating input. See Section 1.7 for termination of unused pins.

## 6.4 Versatile I/O Port Registers

The Versatile I/O registers are listed in Table 6-4, along with their base addresses. The address offset is given in Table 6-5.

**Table 6-4. 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 6-5. Versatile I/O Port Control Registers**

| Offset | Acronym | Port | Register Name | Type | Reset | Section |
|---|---|---|---|---|---|---|
| 0Eh | P1IV | P1[1] | P1 Interrupt Vector | read/write | 0000h | Section 6.4.1 |
| 1Eh | P2IV | P2[1] | P2 Interrupt Vector | read/write | 0000h | Section 6.4.2 |
| 00h | P1IN | P1[1] | Input | read only | - | Section 6.4.3 |
| 02h | P1OUT | | Output | read/write | Unchanged[2] | Section 6.4.4 |
| 04h | P1DIR | | Direction | read/write | 00h[2] | Section 6.4.5 |
| 06h | P1REN | | Resistor Enable | read/write | 00h[2] | Section 6.4.6 |
| 0Ah | P1SEL0 | | Port Select 0 | read/write | 00h[2] | Section 6.4.7 |
| 0Ch | P1SEL1 | | Port Select 1 | read/write | 00h[2] | Section 6.4.8 |
| 18h | P1IES | | Interrupt Edge Select | read/write | Unchanged | Section 6.4.9 |
| 1Ah | P1IE | | Interrupt Enable | read/write | 00h | Section 6.4.10 |
| 1Ch | P1IFG | | Interrupt Flag | read/write | 00h | Section 6.4.11 |
| 01h | P2IN | P2[1] | Input | read only | - | Section 6.4.3 |
| 03h | P2OUT | | Output | read/write | Unchanged[2] | Section 6.4.4 |
| 05h | P2DIR | | Direction | read/write | 00h[2] | Section 6.4.5 |
| 07h | P2REN | | Resistor Enable | read/write | 00h[2] | Section 6.4.6 |
| 0Bh | P2SEL0 | | Port Select 0 | read/write | 00h[2] | Section 6.4.7 |
| 0Dh | P2SEL1 | | Port Select 1 | read/write | 00h[2] | Section 6.4.8 |
| 19h | P2IES | | Interrupt Edge Select | read/write | Unchanged | Section 6.4.9 |
| 1Bh | P2IE | | Interrupt Enable | read/write | 00h | Section 6.4.10 |
| 1Dh | P2IFG | | Interrupt Flag | read/write | 00h | Section 6.4.11 |

[1] Similar for port combos P3/P4, P5/P6, and so on.

[2] Unless otherwise noted in the device-specific data sheet

### 6.4.1 P1IV Register

Port 1 Interrupt Vector Register

**Figure 6-2. P1IV Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| P1IVx | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

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

**Table 6-6. P1IV Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | P1IVx | R | 0h | Port 1 interrupt vector value. Writing to this register clears all pending interrupt flags<br>00h = No interrupt pending<br>02h = Interrupt Source = Port 1.0 interrupt; Interrupt Flag = P1IFG.0; Priority = Highest<br>04h = Interrupt Source = Port 1.1 interrupt; Interrupt Flag = P1IFG.1<br>06h = Interrupt Source = Port 1.2 interrupt; Interrupt Flag = P1IFG.2<br>08h = Interrupt Source = Port 1.3 interrupt; Interrupt Flag = P1IFG.3<br>0Ah = Interrupt Source = Port 1.4 interrupt; Interrupt Flag = P1IFG.4<br>0Ch = Interrupt Source = Port 1.5 interrupt; Interrupt Flag = P1IFG.5<br>0Eh = Interrupt Source = Port 1.6 interrupt; Interrupt Flag = P1IFG.6<br>10h = Interrupt Source = Port 1.7 interrupt; Interrupt Flag = P1IFG.7; Priority = Lowest |

### 6.4.2 P2IV Register

Port 2 Interrupt Vector Register

**Figure 6-3. P2IV Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| P2IVx | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

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

**Table 6-7. P2IV Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | P2IVx | R | 0h | Port 2 interrupt vector value. Writing to this register clears all pending interrupt flags<br>00h = No interrupt pending<br>02h = Interrupt Source = Port 2.0 interrupt; Interrupt Flag = P2IFG.0; Priority = Highest<br>04h = Interrupt Source = Port 2.1 interrupt; Interrupt Flag = P2IFG.1<br>06h = Interrupt Source = Port 2.2 interrupt; Interrupt Flag = P2IFG.2<br>08h = Interrupt Source = Port 2.3 interrupt; Interrupt Flag = P2IFG.3<br>0Ah = Interrupt Source = Port 2.4 interrupt; Interrupt Flag = P2IFG.4<br>0Ch = Interrupt Source = Port 2.5 interrupt; Interrupt Flag = P2IFG.5<br>0Eh = Interrupt Source = Port 2.6 interrupt; Interrupt Flag = P2IFG.6<br>10h = Interrupt Source = Port 2.7 interrupt; Interrupt Flag = P2IFG.7; Priority = Lowest |

### 6.4.3 *PxIN Register*

Port x Input Register

**Figure 6-4. PxIN Register**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| PxIN | | | | | | | |

**Table 6-8. PxIN Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 7-0 | PxIN | R | 0h | Port x input register. Read only. |

### 6.4.4 *PxOUT Register*

Port x Output Register

**Figure 6-5. PxOUT Register**

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

**Table 6-9. PxOUT Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 7-0 | PxOUT | RW | 0h | Port x output register<br>When I/O configured to output mode:<br>0b = The output is low<br>1b = The output is high<br>When I/O configured to input mode and pullups/pulldowns enabled:<br>0b = Pulldown selected<br>1b = Pullup selected |

### 6.4.5 PxDIR Register

Port x Direction Register

**Figure 6-6. PxDIR 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 |

**Table 6-10. PxDIR Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 7-0 | PxDIR | RW | 0h | Port x direction register<br>0b = Port configured as input<br>1b = Port configured as output |

### 6.4.6 PxREN Register

Port x Resistor Enable Register

**Figure 6-7. PxREN 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 |

**Table 6-11. PxREN Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 7-0 | PxREN | RW | 0h | Port x Pullup or pulldown resistor enable register<br>0b = Pullup or pulldown disabled<br>1b = Pullup or pulldown enabled |

### 6.4.7 PxSEL0 Register

Port x Function Select Register 0

**Figure 6-8. PxSEL0 Register**

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

**Table 6-12. PxSEL0 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 7-0 | PxSEL0 | RW | 0h | Port x function select register. These two 8-bit registers determine the function of each port pin. The chosen functions for a particular pin are device dependent (see the port schematics of the device-specific data sheet).<br>See Table 6-14 for valid values. |

### 6.4.8 PxSEL1 Register

Port x Function Select Register 1

**Figure 6-9. PxSEL1 Register**

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

**Table 6-13. PxSEL1 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 7-0 | PxSEL1 | RW | 0h | Port x function select register. These two 8-bit registers determine the function of each port pin. The chosen functions for a particular pin are device dependent (see the port schematics of the device-specific data sheet).<br>See Table 6-14 for valid values. |

**Table 6-14. PxSEL0 and PxSEL1 Values**

| 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 | Tertiary module function is selected |

### 6.4.9 PxIES Register

Port x Interrupt Edge Select Register

**Figure 6-10. PxIES Register**

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

**Table 6-15. PxIES Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 7-0 | PxIES | RW | 0h | Port x interrupt edge select register<br>0b = PxIFGx flag is set with a low-to-high transition<br>1b = PxIFGx flag is set with a high-to-low transition |

### 6.4.10 PxIE Register

Port x Interrupt Enable Register

**Figure 6-11. PxIE 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 |

**Table 6-16. PxIE Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 7-0 | PxIE | RW | 0h | Port x interrupt enable register<br>0b = Corresponding port interrupt disabled<br>1b = Corresponding port interrupt enabled |

### 6.4.11 PxIFG Register

Port x Interrupt Flag Register

**Figure 6-12. PxIFG 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 |

**Table 6-17. PxIFG Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 7-0 | PxIFG | RW | 0h | Port x interrupt flag register<br>0b = No interrupt is pending<br>1b = Interrupt is pending |

Chapter 7

SLAU506–December 2014

# CRC Module

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

**Topic** **Page**

## 7.1 Cyclic Redundancy Check (CRC) Module Introduction

The CRC module produces a signature for a given sequence of data values. The signature is generated through a feedback path from data bits 0, 4, 11, and 15 (see Figure 7-1). The CRC signature is based on the polynomial given in the CRC-CCITT-BR polynomial (see Equation 1) .

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

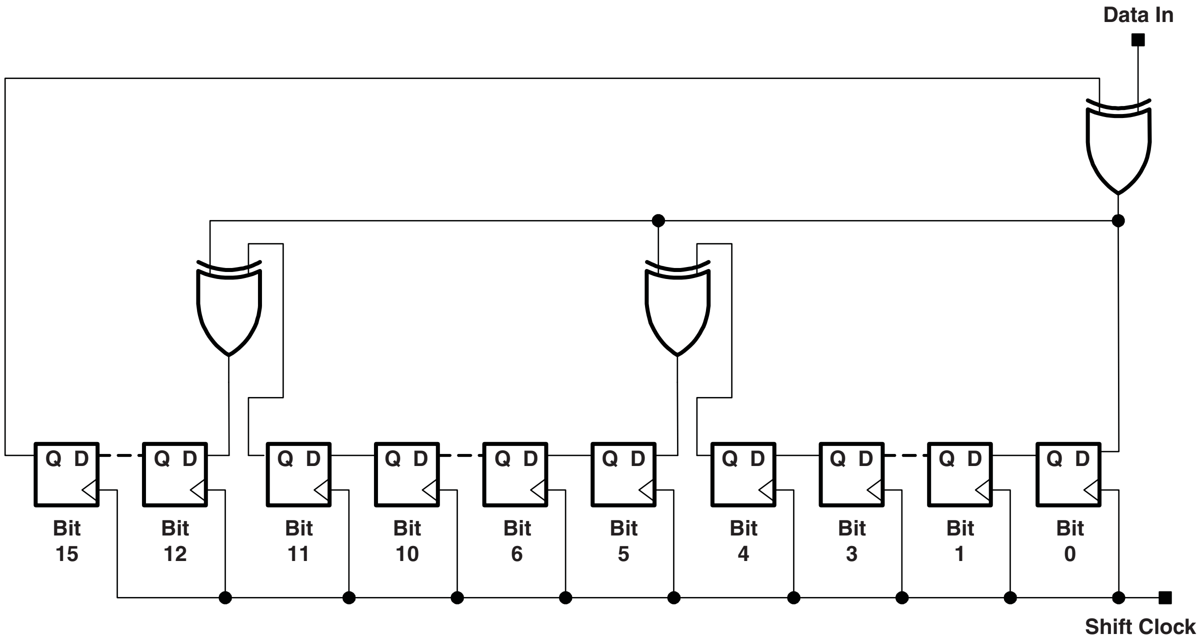


**Figure 7-1. LFSR Implementation of CRC-CCITT Standard, Bit 0 is the MSB of the Result**

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

## 7.2 CRC Standard and Bit Order

The definitions of the various CRC standards were done in the era of main frame computers, and by convention bit 0 was treated as the MSB. Today, as in most microcontrollers such as the MSP430, bit 0 normally denotes the LSB. In , the bit convention shown is as given in the original standards i.e. bit 0 is the MSB. The fact that bit 0 is treated for some as LSB, and for others as MSB, continues to cause confusion. The CRC16 module therefore provides a bit reversed register pair for CRC16 operations to support both conventions.

## 7.3 CRC Checksum Generation

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

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

### *7.3.1 CRC Implementation*

To allow parallel processing of the CRC, the linear feedback shift register (LFSR) functionality is implemented with an XOR tree. This implementation shows the identical behavior as the LFSR approach after 8 bits of data are shifted in when the LSB is 'shifted' in first. The generation of a signature calculation has to be started by writing a seed to the CRCINIRES register to initialize the register. Software or hardware (for example, the DMA) can transfer data to the CRCDI or CRCDIRB register (for example, from memory). The value in CRCDI or CRCDIRB is then included into the signature, and the result is available in the signature result registers at the next read access (CRCINIRES and CRCRESR). The signature can be generated using word or byte data.

If a word data is processed, the lower byte at the even address is used at the first clock (MCLK) cycle. During the second clock cycle, the higher byte is processed. Thus, it takes two clock cycles to process word data, while it takes only one clock (MCLK) cycle to process byte data.

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

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

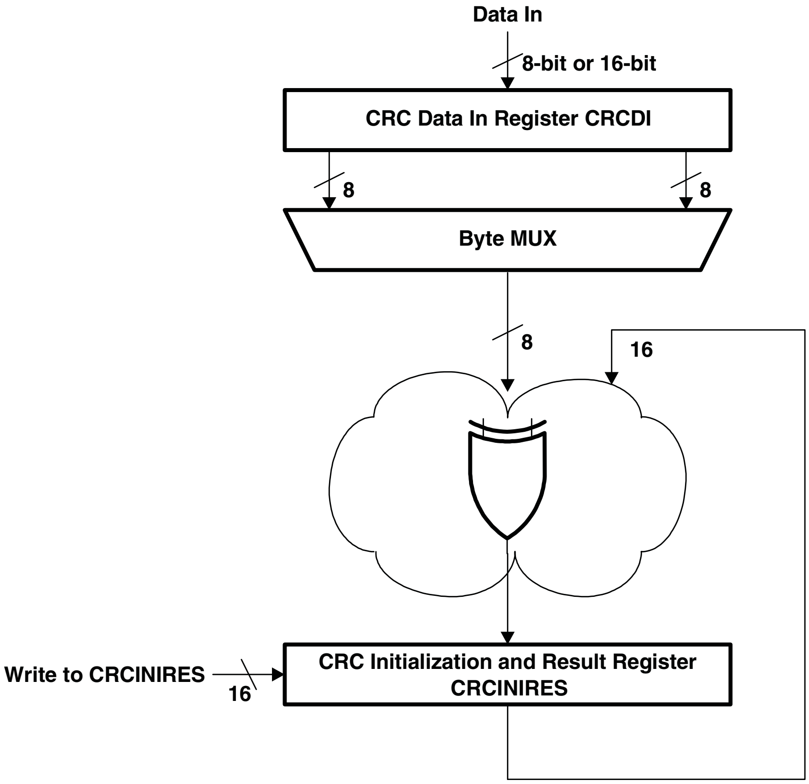

Data In
8-bit or 16-bit
CRC Data In Register CRCDI
8
8
Byte MUX
8
16
Write to CRCINIRES
16
CRC Initialization and Result Register
CRCINIRES

**Figure 7-2. Implementation of CRC-CCITT Using the CRCDI and CRCINIRES Registers**

### 7.3.2 Assembler Examples

Example 7-1 demonstrates the operation of the on-chip CRC.

**Example 7-1. General Assembler Example**

```
    ...
    PUSH   R4                   ; Save registers
    PUSH   R5
    MOV    #StartAddress,R4     ; StartAddress < EndAddress
    MOV    #EndAddress,R5
    MOV    &INIT, &CRCINIRES    ; INIT to CRCINIRES
L1  MOV    @R4+,&CRCDI          ; Item to Data In register
    CMP    R5,R4                ; End address reached?
    JLO    L1                   ; No
    MOV    &Check_Sum,&CRCDI    ; Yes, Include checksum
    TST    &CRCINIRES           ; Result = 0?
    JNZ    CRC_ERROR            ; No, CRCRES <> 0: error
    ...                         ; Yes, CRCRES=0:
                                ; information ok.
    POP    R5                   ; Restore registers
    POP    R4
```

The details of the implemented CRC algorithm are shown by the data sequences in Example 7-2 using word or byte accesses and the CRC data-in as well as the CRC data-in reverse byte registers.

### *Example 7-2. Reference Data Sequence*

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

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

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

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

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

## 7.4 CRC Registers

The CRC module registers are listed in Table 7-1. The base address can be found in the device-specific data sheet. The address offset is given in Table 7-1.

**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 7-1. CRC Registers**

| Offset | Acronym | Register Name | Type | Access | Reset | Section |
|---|---|---|---|---|---|---|
| 00h | CRCDI | CRC Data In | Read/write | Word | 0000h | Section 7.4.1 |
| 00h | CRCDI_L | | Read/write | Byte | 00h | |
| 01h | CRCDI_H | | Read/write | Byte | 00h | |
| 02h | CRCDIRB | CRC Data In Reverse Byte | Read/write | Word | 0000h | Section 7.4.2 |
| 02h | CRCDIRB_L | | Read/write | Byte | 00h | |
| 03h | CRCDIRB_H | | Read/write | Byte | 00h | |
| 04h | CRCINIRES | CRC Initialization and Result | Read/write | Word | FFFFh | Section 7.4.3 |
| 04h | CRCINIRES_L | | Read/write | Byte | FFh | |
| 05h | CRCINIRES_H | | Read/write | Byte | FFh | |
| 06h | CRCRESR | CRC Result Reverse | Read only | Word | FFFFh | Section 7.4.4 |
| 06h | CRCRESR_L | | Read/write | Byte | FFh | |
| 07h | CRCRESR_H | | Read/write | Byte | FFh | |

### 7.4.1 *CRCDI Register*

CRC Data In Register

**Figure 7-3. CRCDI Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CRCDI | | | | | | | |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |

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

**Table 7-2. CRCDI Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | CRCDI | RW | 0h | CRC data in. Data written to the CRCDI register is included to the present signature in the CRCINIRES register according to the CRC-CCITT standard. |

### 7.4.2 *CRCDIRB Register*

CRC Data In Reverse Register

**Figure 7-4. CRCDIRB Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CRCDIRB | | | | | | | |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |

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

**Table 7-3. CRCDIRB Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | CRCDIRB | RW | 0h | CRC data in reverse byte. Data written to the CRCDIRB register is included to the present signature in the CRCINIRES and CRCRESR registers according to the CRC-CCITT standard. Reading the register returns the register CRCDI content. |

### 7.4.3 CRCINIRES Register

CRC Initialization and Result Register

**Figure 7-5. CRCINIRES Register**

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

**Table 7-4. CRCINIRES Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | CRCINIRES | RW | FFFFh | CRC initialization and result. This register holds the current CRC result (according to the CRC-CCITT standard). Writing to this register initializes the CRC calculation with the value written to it. The value just written can be read from CRCINIRES register. |

### 7.4.4 CRCRESR Register

CRC Reverse Result Register

**Figure 7-6. CRCRESR Register**

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

**Table 7-5. CRCRESR Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | CRCRESR | R | FFFFh | CRC reverse result. This register holds the current CRC result (according to the CRC-CCITT standard). The order of bits is reverse (for example, CRCINIRES[15] = CRCRESR[0]) to the order of bits in the CRCINIRES register (see example code). |

Chapter 8
SLAU506–December 2014

# 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**

## 8.1 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 8-1.

**NOTE: Watchdog timer powers up active.**

After a PUC, the WDT_A module is automatically configured in the watchdog mode with an initial approximately 32-ms reset interval using the SMCLK. The user must set up or halt the WDT_A before the initial reset interval expires.

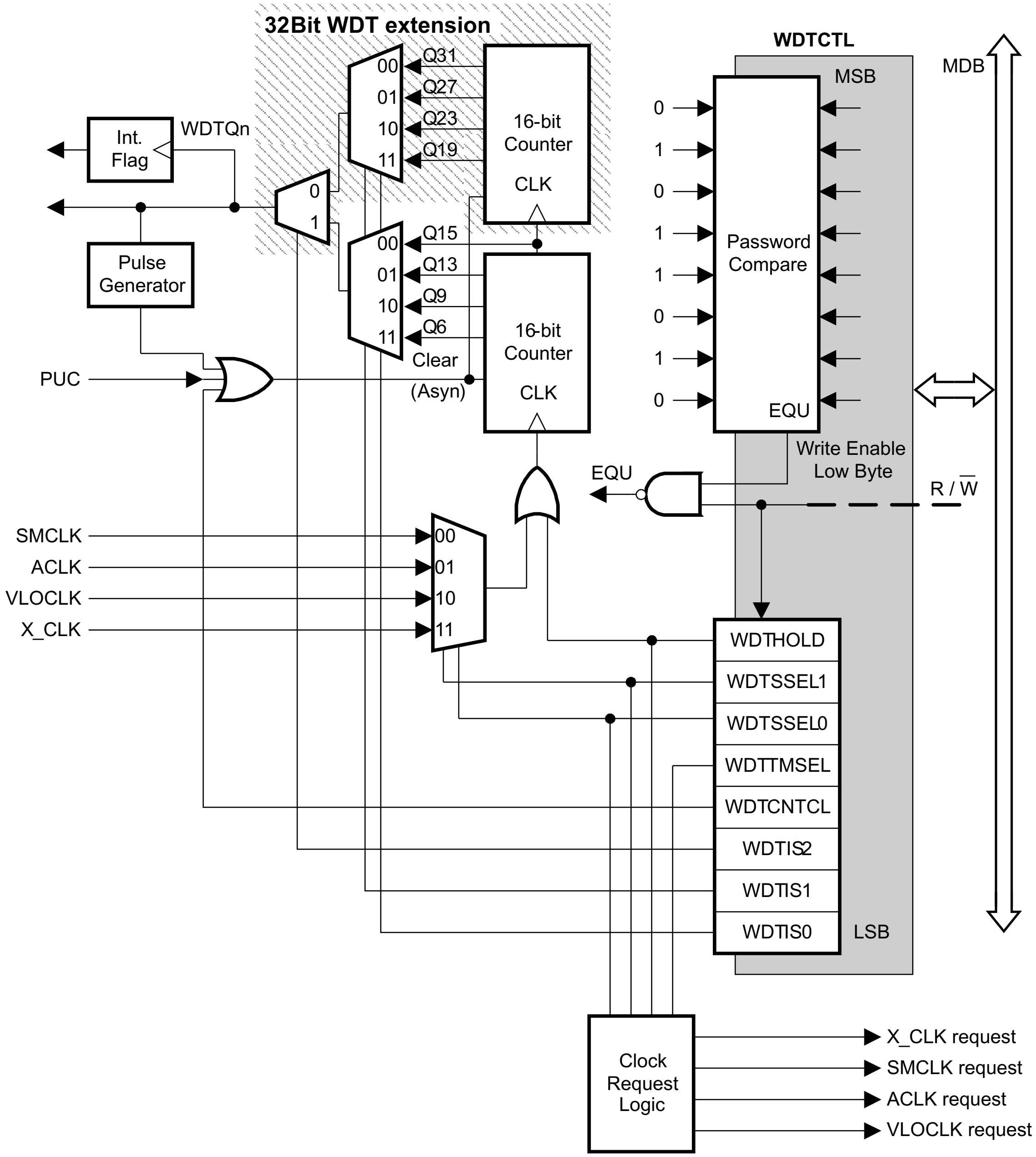


Figure 8-1. Watchdog Timer Block Diagram

## 8.2 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. A write to WDTCTL with any value other than 05Ah in the upper byte is a password violation and causes 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.

### *8.2.1 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.

### *8.2.2 Watchdog Mode*

After a PUC condition, the WDT module is configured in the watchdog mode with an initial 32-ms (approximate) reset interval using the SMCLK. The user must set up, halt, or clear the watchdog timer before this initial reset interval expires, 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.

### *8.2.3 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.

### *8.2.4 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.

### 8.2.5 Fail-Safe Features

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.

In watchdog mode the WDT_A prevents LPMx.5 because in LPMx.5 the WDT_A cannot operate.

When the WDT_A module is used in interval timer mode, there are no fail-safe features.

### 8.2.6 Operation in Low-Power Modes

The devices have several low-power modes. Different clock signals are available in different low-power modes. The requirements of the application and the type of clocking that is used determine how the WDT_A should be configured. For example, the WDT_A should not be configured in watchdog mode with a clock source that is originally sourced from DCO, XT1 in high-frequency mode, or XT2 via SMCLK or ACLK if the user wants to use low-power mode 3. In this case, SMCLK or ACLK would remain enabled, increasing the current consumption of LPM3. When the watchdog timer is not required, the WDTHOLD bit can be used to hold the WDTCNT, reducing power consumption.

Any write operation to WDTCTL must be a word operation with 05Ah (WDTPW) in the upper byte (see Example 8-1).

***Example 8-1. Writes to WDTCTL***

```
; Periodically clear an active watchdog
MOV #WDTPW+WDTIS2+WDTIS1+WDTCNTCL,&WDTCTL
;
; Change watchdog timer interval
MOV #WDTPW+WDTCNTCL+SSEL,&WDTCTL
;
; Stop the watchdog
MOV #WDTPW+WDTHOLD,&WDTCTL
;
; Change WDT to interval timer mode, clock/8192 interval
MOV #WDTPW+WDTCNTCL+WDTTMSEL+WDTIS2+WDTIS0,&WDTCTL
```

## 8.3 WDT_A Registers

The watchdog timer module registers are listed in Table 8-1. The base address for the watchdog timer module registers and special function registers (SFRs) can be found in the device-specific data sheets. The address offset is given in Table 8-1.

**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 8-1. WDT_A Registers**

| Offset | Acronym | Register Name | Type | Access | Reset | Section |
|---|---|---|---|---|---|---|
| 0Ch | WDTCTL | Watchdog Timer Control | Read/write | Word | 6904h | Section 8.3.1 |
| 0Ch | WDTCTL_L | | Read/write | Byte | 04h | |
| 0Dh | WDTCTL_H | | Read/write | Byte | 69h | |

### 8.3.1 WDTCTL Register

Watchdog Timer Control Register

**Figure 8-2. WDTCTL Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| WDTPW | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

| 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 |

**Table 8-2. WDTCTL Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | WDTPW | RW | 69h | Watchdog timer password. Always read as 069h. Must be written as 05Ah, or a PUC is generated. |
| 7 | WDTHOLD | RW | 0h | Watchdog timer hold. This bit stops the watchdog timer. Setting WDTHOLD = 1 when the WDT is not in use conserves power.<br>0b = Watchdog timer is not stopped<br>1b = Watchdog timer is stopped |
| 6-5 | WDTSSEL | RW | 0h | Watchdog timer clock source select<br>00b = SMCLK<br>01b = ACLK<br>10b = VLOCLK<br>11b = X_CLK, same as VLOCLK if not defined differently in data sheet |
| 4 | WDTTMSEL | RW | 0h | Watchdog timer mode select<br>0b = Watchdog mode<br>1b = Interval timer mode |
| 3 | WDTCNTCL | RW | 0h | Watchdog timer counter clear. Setting WDTCNTCL = 1 clears the count value to 0000h. WDTCNTCL is automatically reset.<br>0b = No action<br>1b = WDTCNT = 0000h |
| 2-0 | WDTIS | RW | 4h | Watchdog timer interval select. These bits select the watchdog timer interval to set the WDTIFG flag or generate a PUC.<br>000b = Watchdog clock source / $2^{31}$ (18:12:16 at 32.768 kHz)<br>001b = Watchdog clock source / $2^{27}$ (01:08:16 at 32.768 kHz)<br>010b = Watchdog clock source / $2^{23}$ (00:04:16 at 32.768 kHz)<br>011b = Watchdog clock source / $2^{19}$ (00:00:16 at 32.768 kHz)<br>100b = Watchdog clock source / $2^{15}$ (1 s at 32.768 kHz)<br>101b = Watchdog clock source / $2^{13}$ (250 ms at 32.768 kHz)<br>110b = Watchdog clock source / $2^{9}$ (15.625 ms at 32.768 kHz)<br>111b = Watchdog clock source / $2^{6}$ (1.95 ms at 32.768 kHz) |

Chapter 9

SLAU506–December 2014

# 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**

## 9.1 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 9-1.

**NOTE: Use of the word *count***

*Count* is used throughout this chapter. It means the counter must be in the process of counting for the action to take place. If a particular value is directly written to the counter, 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.

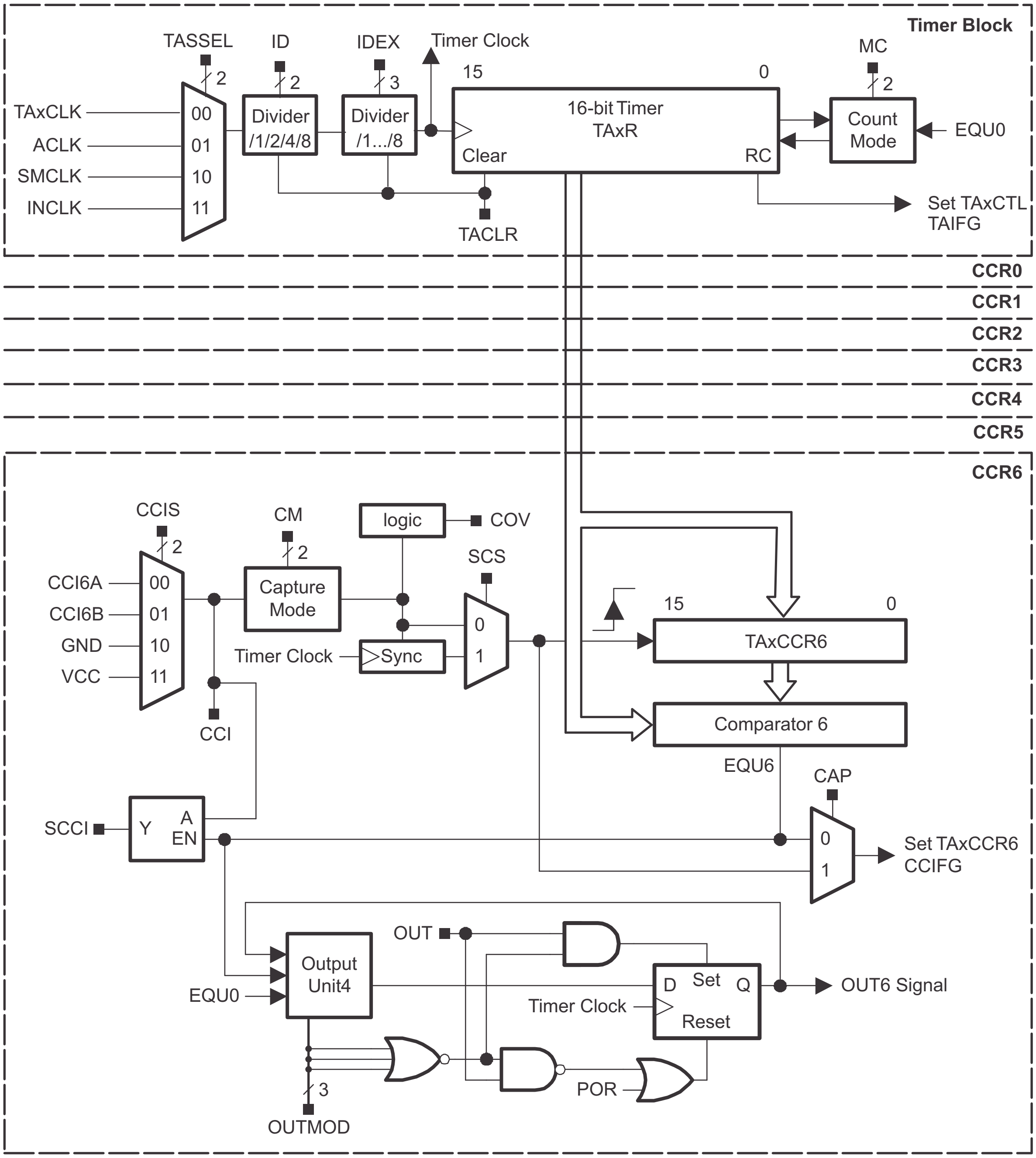


**Figure 9-1. Timer_A Block Diagram**

## 9.2 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.

### 9.2.1 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.

#### 9.2.1.1 Clock Source Select and Divider

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

**NOTE: Timer_A dividers**

After programming ID or TAIDEX bits, set the TACLR bit. This clears the contents of TAxR and resets the clock divider logic to a defined state. The clock dividers are implemented as down counters. Therefore, when the TACLR bit is cleared, the timer clock immediately begins clocking at the first rising edge of the Timer_A clock source selected with the TASSEL bits and continues clocking at the divider settings set by the ID and TAIDEX bits.

### 9.2.2 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.

### 9.2.3 Timer Mode Control

The timer has four modes of operation: stop, up, continuous, and up/down (see Table 9-1). The operating mode is selected with the MC bits.

**Table 9-1. Timer Modes**

| MC | 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. |

#### 9.2.3.1 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 9-2). 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.

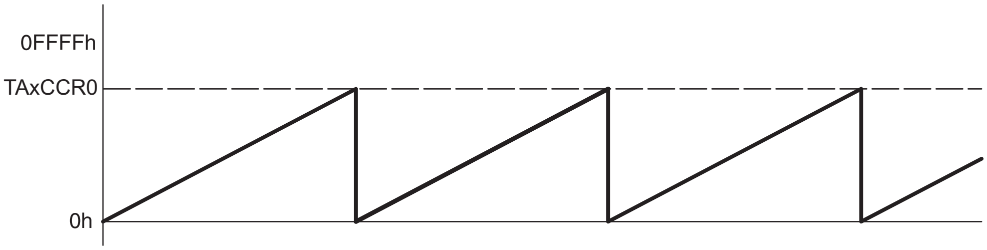


**Figure 9-2. Up Mode**

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 9-3 shows the flag set cycle.

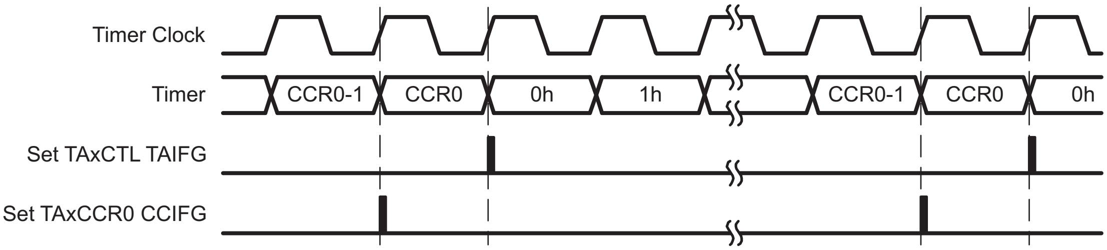


**Figure 9-3. Up Mode Flag Setting**

#### *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.

### 9.2.3.2 Continuous Mode

In the continuous mode, the timer repeatedly counts up to 0FFFFh and restarts from zero as shown in Figure 9-4. The capture/compare register TAxCCR0 works the same way as the other capture/compare registers.

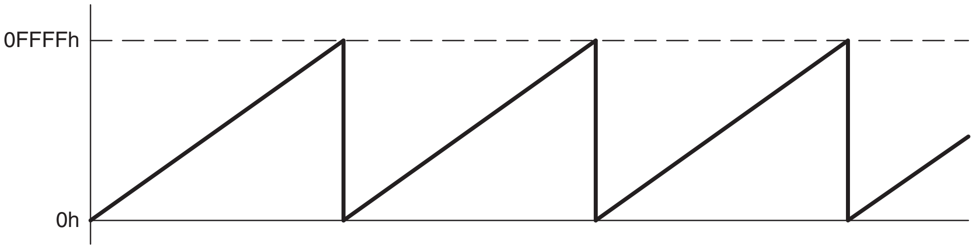


**Figure 9-4. Continuous Mode**

The TAIFG interrupt flag is set when the timer *counts* from 0FFFFh to zero. Figure 9-5 shows the flag set cycle.

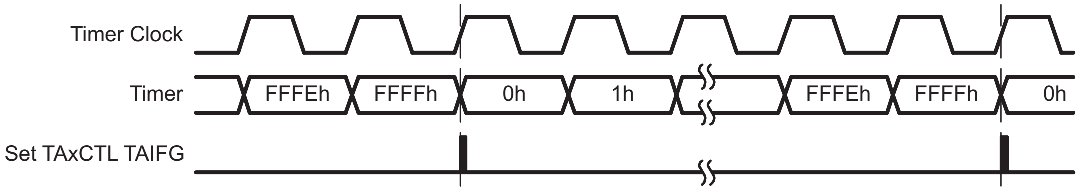


**Figure 9-5. Continuous Mode Flag Setting**

### 9.2.3.3 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 9-6 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.

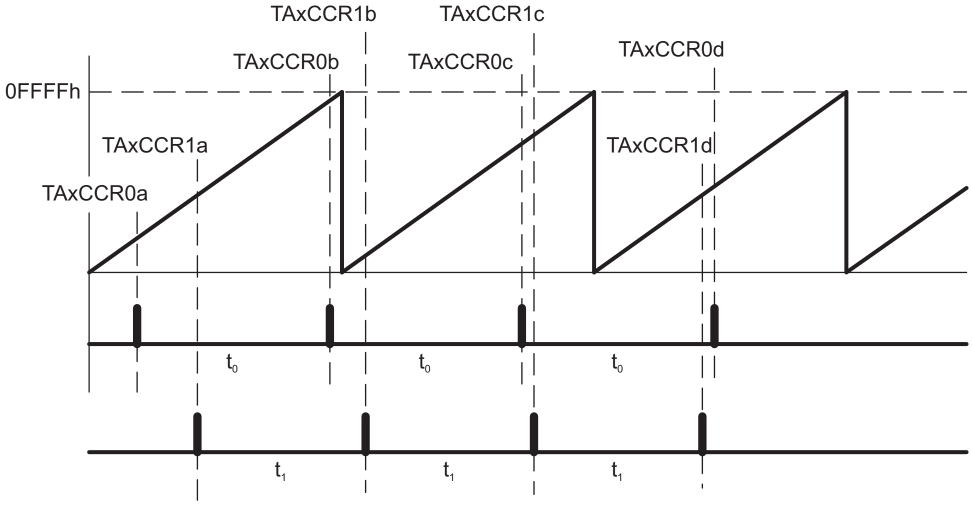


**Figure 9-6. Continuous Mode Time Intervals**

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.

### 9.2.3.4 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 9-7). The period is twice the value in TAxCCR0.

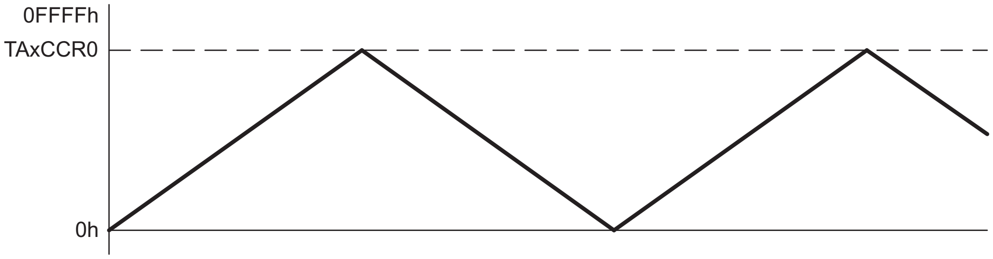


**Figure 9-7. Up/Down Mode**

The count direction is latched. This allows the timer to be stopped and then restarted in the same direction it was counting before it was stopped. If this is not desired, the 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 9-8 shows the flag set cycle.

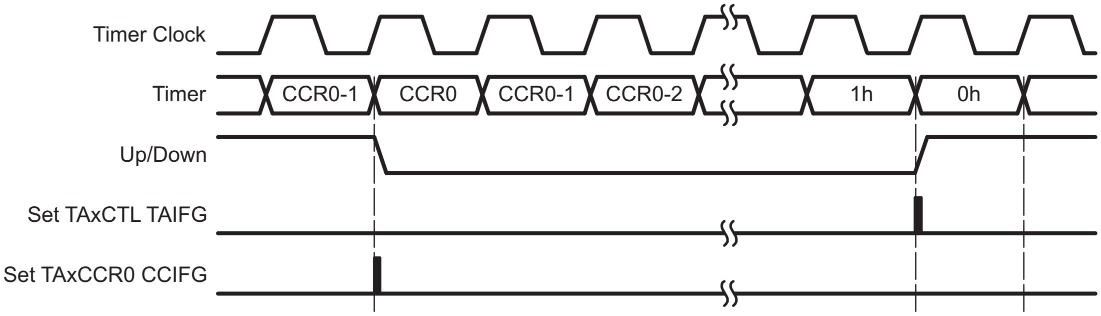


**Figure 9-8. Up/Down Mode Flag Setting**

### *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 effect 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.

### 9.2.3.5 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 9-9, the $t_{dead}$ is:

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

Where:

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

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

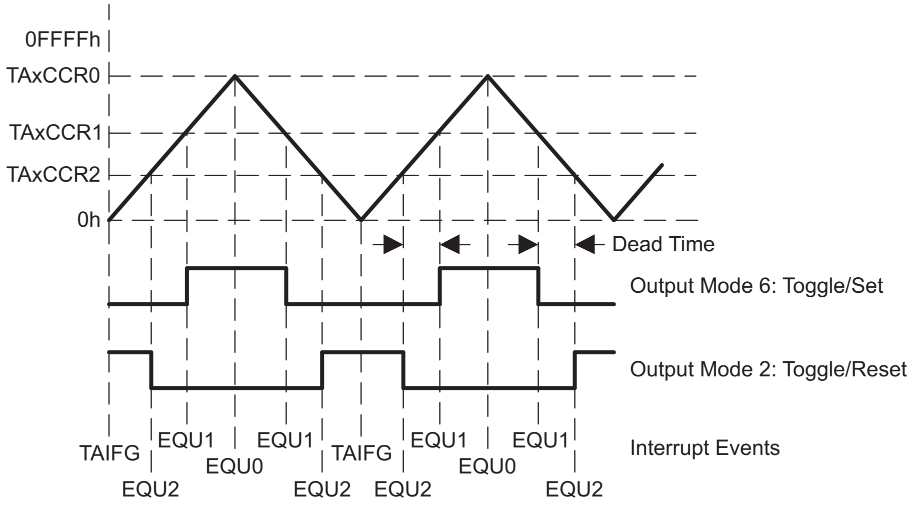


**Figure 9-9. Output Unit in Up/Down Mode**

## 9.2.4 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.

### 9.2.4.1 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 9-10).

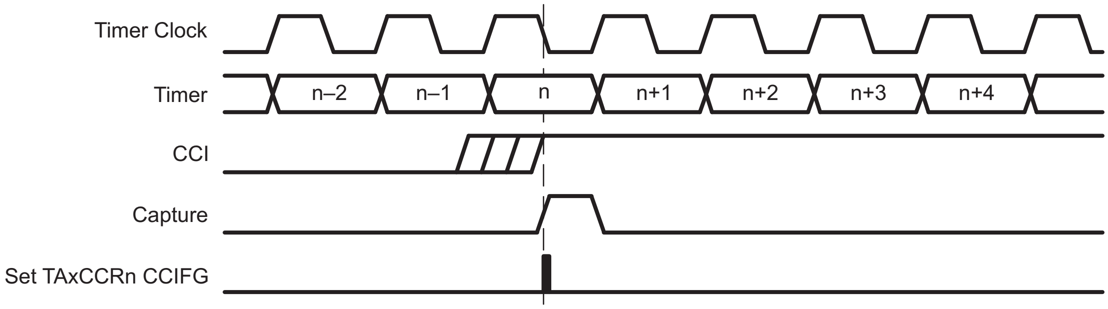


**Figure 9-10. Capture Signal (SCS = 1)**

**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 9-11. COV must be reset with software.

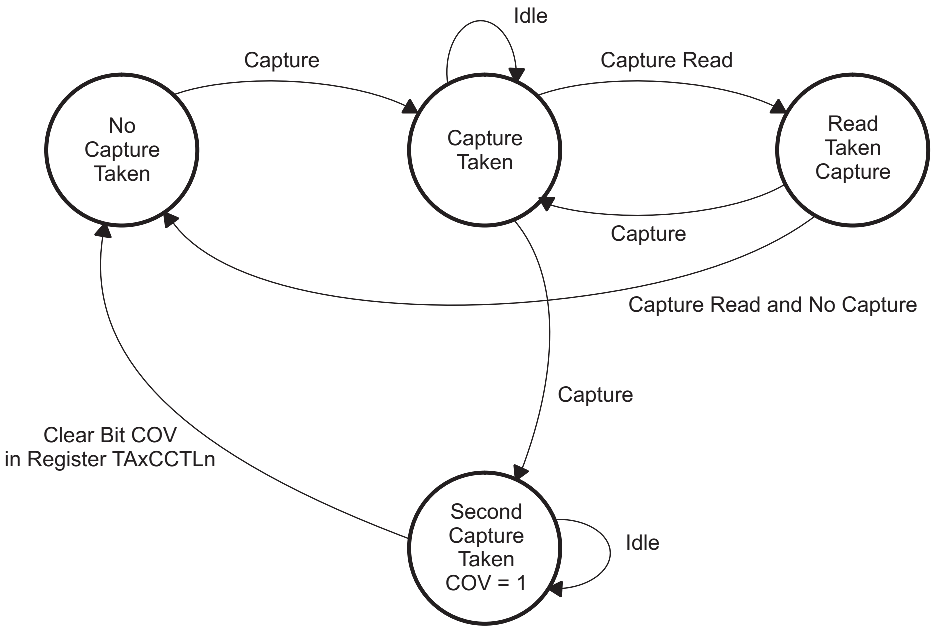

Idle
Capture
Capture Read
No Capture Taken
Capture Taken
Read Taken Capture
Capture
Capture Read and No Capture
Capture
Clear Bit COV in Register TAxCCTLn
Second Capture Taken COV = 1
Idle

**Figure 9-11. Capture Cycle**

### Capture Initiated by Software

Captures can be initiated by software. The CMx bits can be set for capture on both edges. Software then sets 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.

### 9.2.4.2 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.

## *9.2.5 Output Unit*

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

### 9.2.5.1 Output Modes

The output modes are defined by the OUTMOD bits and are described in Table 9-2. 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 9-2. 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. |

### 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 9-12 using TAxCCR0 and TAxCCR1.

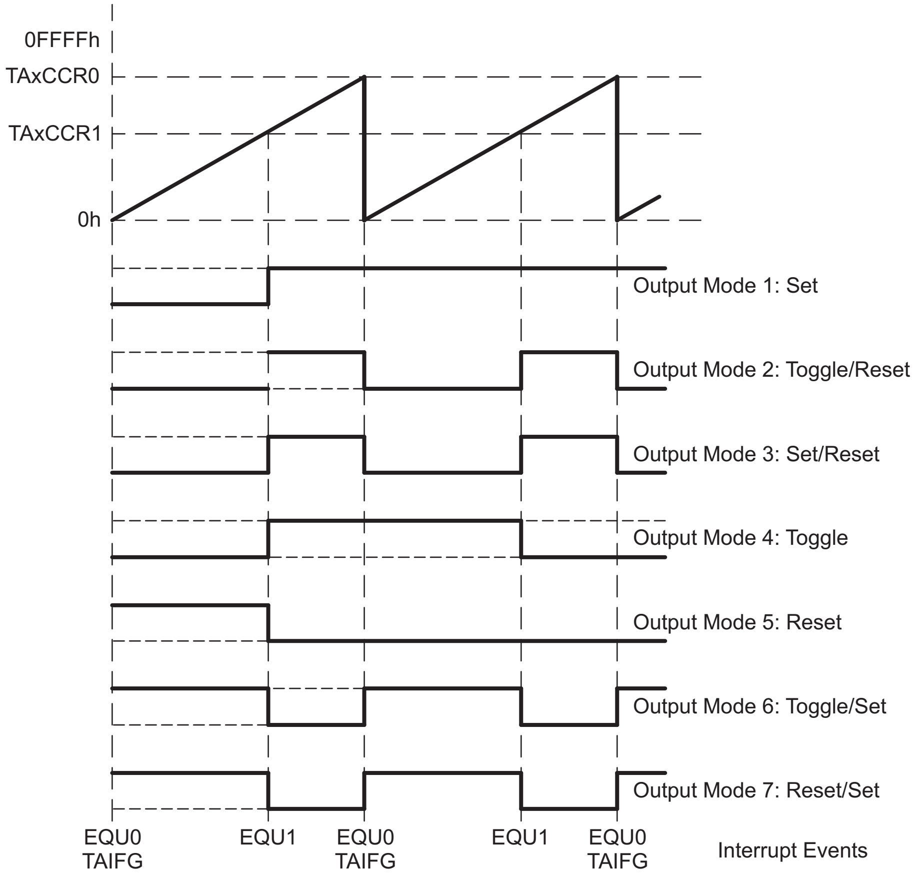


**Figure 9-12. Output Example – Timer in Up Mode**

### 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 9-13 using TAxCCR0 and TAxCCR1.

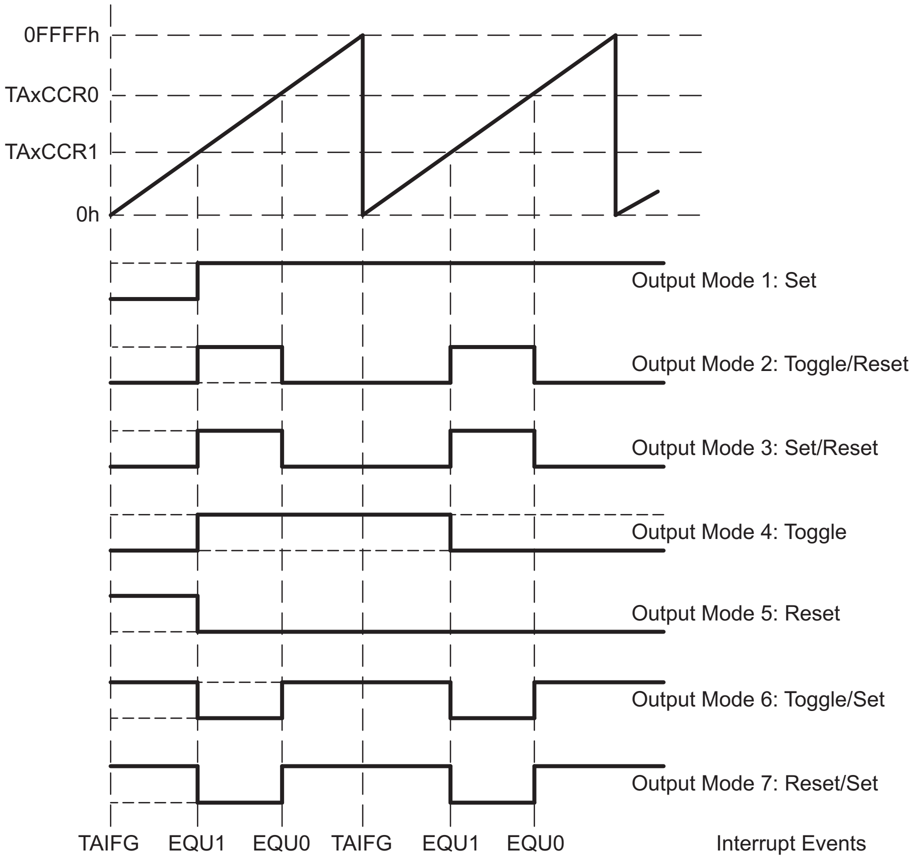


**Figure 9-13. Output Example – Timer in Continuous Mode**

### 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 9-14 using TAxCCR0 and TAxCCR2.

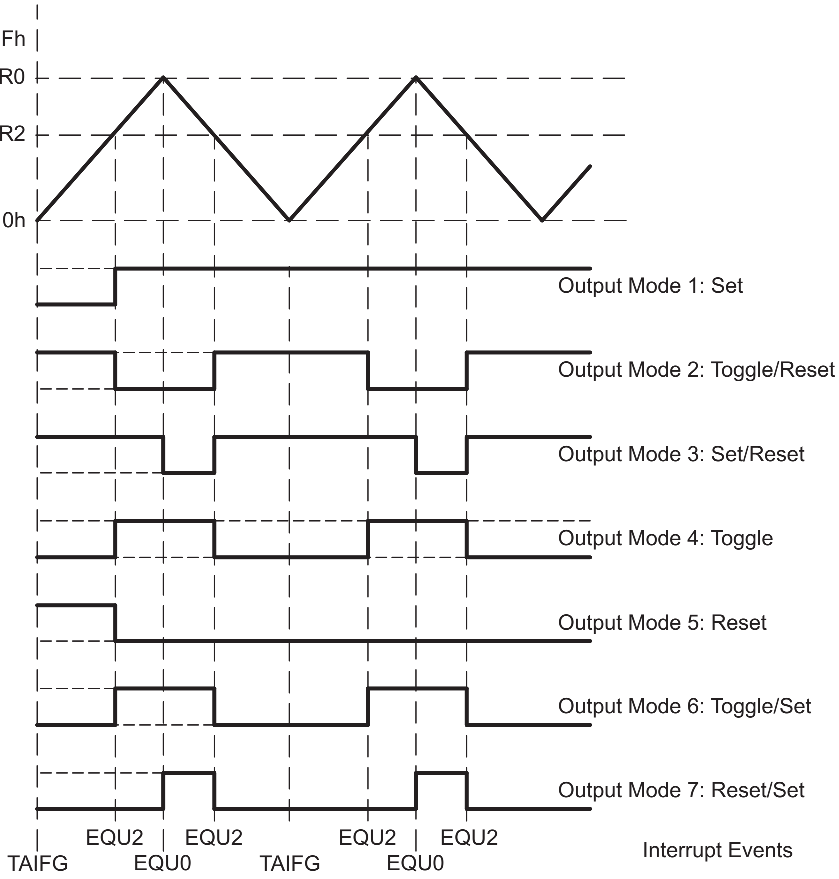


**Figure 9-14. Output Example – Timer in Up/Down Mode**

**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
```

### *9.2.6 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.

#### 9.2.6.1 TAxCCR0 Interrupt

The TAxCCR0 CCIFG flag has the highest Timer_A interrupt priority and has a dedicated interrupt vector as shown in Figure 9-15. The TAxCCR0 CCIFG flag is automatically reset when the TAxCCR0 interrupt request is serviced.

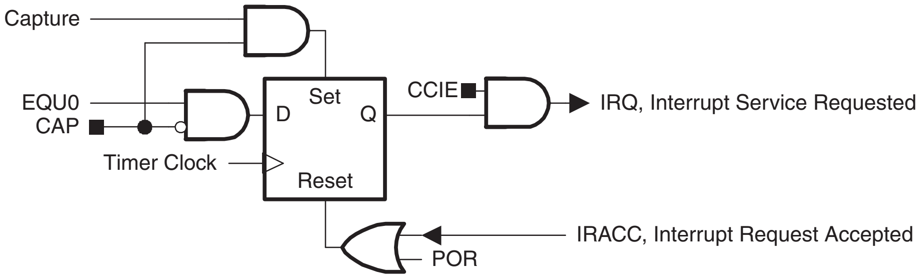

Capture
EQU0
CAP
Timer Clock
Set
D
Q
Reset
CCIE
IRQ, Interrupt Service Requested
IRACC, Interrupt Request Accepted
POR

**Figure 9-15. Capture/Compare TAxCCR0 Interrupt Flag**

#### 9.2.6.2 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.

### 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
```

## 9.3 Timer_A Registers

Timer_A registers are listed in Table 9-3 for the largest configuration available. The base address can be found in the device-specific data sheet.

**Table 9-3. Timer_A Registers**

| Offset | Acronym | Register Name | Type | Access | Reset | Section |
|---|---|---|---|---|---|---|
| 00h | TAxCTL | Timer_Ax Control | Read/write | Word | 0000h | Section 9.3.1 |
| 02h | TAxCCTL0 | Timer_Ax Capture/Compare Control 0 | Read/write | Word | 0000h | Section 9.3.3 |
| 04h | TAxCCTL1 | Timer_Ax Capture/Compare Control 1 | Read/write | Word | 0000h | Section 9.3.3 |
| 06h | TAxCCTL2 | Timer_Ax Capture/Compare Control 2 | Read/write | Word | 0000h | Section 9.3.3 |
| 08h | TAxCCTL3 | Timer_Ax Capture/Compare Control 3 | Read/write | Word | 0000h | Section 9.3.3 |
| 0Ah | TAxCCTL4 | Timer_Ax Capture/Compare Control 4 | Read/write | Word | 0000h | Section 9.3.3 |
| 0Ch | TAxCCTL5 | Timer_Ax Capture/Compare Control 5 | Read/write | Word | 0000h | Section 9.3.3 |
| 0Eh | TAxCCTL6 | Timer_Ax Capture/Compare Control 6 | Read/write | Word | 0000h | Section 9.3.3 |
| 10h | TAxR | Timer_Ax Counter | Read/write | Word | 0000h | Section 9.3.2 |
| 12h | TAxCCR0 | Timer_Ax Capture/Compare 0 | Read/write | Word | 0000h | Section 9.3.4 |
| 14h | TAxCCR1 | Timer_Ax Capture/Compare 1 | Read/write | Word | 0000h | Section 9.3.4 |
| 16h | TAxCCR2 | Timer_Ax Capture/Compare 2 | Read/write | Word | 0000h | Section 9.3.4 |
| 18h | TAxCCR3 | Timer_Ax Capture/Compare 3 | Read/write | Word | 0000h | Section 9.3.4 |
| 1Ah | TAxCCR4 | Timer_Ax Capture/Compare 4 | Read/write | Word | 0000h | Section 9.3.4 |
| 1Ch | TAxCCR5 | Timer_Ax Capture/Compare 5 | Read/write | Word | 0000h | Section 9.3.4 |
| 1Eh | TAxCCR6 | Timer_Ax Capture/Compare 6 | Read/write | Word | 0000h | Section 9.3.4 |
| 2Eh | TAxIV | Timer_Ax Interrupt Vector | Read only | Word | 0000h | Section 9.3.5 |
| 20h | TAxEX0 | Timer_Ax Expansion 0 | Read/write | Word | 0000h | Section 9.3.6 |

### *9.3.1 TAxCTL Register*

Timer_Ax Control Register

**Figure 9-16. TAxCTL Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | 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 | | Reserved | TACLR | TAIE | TAIFG |
| rw-(0) | rw-(0) | rw-(0) | rw-(0) | rw-(0) | w-(0) | rw-(0) | rw-(0) |

**Table 9-4. TAxCTL Register Description**

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

### 9.3.2 TAxR Register

Timer_Ax Counter Register

**Figure 9-17. TAxR Register**

| 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) |

**Table 9-5. TAxR Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | TAxR | RW | 0h | Timer_A register. The TAxR register is the count of Timer_A. |

### *9.3.3 TAxCCTLn Register*

Timer_Ax Capture/Compare Control n Register

**Figure 9-18. TAxCCTLn Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| CM | | CCIS | | SCS | SCCI | Reserved | 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) |

**Table 9-6. TAxCCTLn Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-14 | CM | RW | 0h | Capture mode<br>00b = No capture<br>01b = Capture on rising edge<br>10b = Capture on falling edge<br>11b = Capture on both rising and falling edges |
| 13-12 | CCIS | RW | 0h | Capture/compare input select. These bits select the TAxCCR0 input signal. See the device-specific data sheet for specific signal connections.<br>00b = CCIxA<br>01b = CCIxB<br>10b = GND<br>11b = VCC |
| 11 | SCS | RW | 0h | Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.<br>0b = Asynchronous capture<br>1b = Synchronous capture |
| 10 | SCCI | RW | 0h | Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit. |
| 9 | Reserved | R | 0h | Reserved. Reads as 0. |
| 8 | CAP | RW | 0h | Capture mode<br>0b = Compare mode<br>1b = Capture mode |
| 7-5 | OUTMOD | RW | 0h | Output mode. Modes 2, 3, 6, and 7 are not useful for TAxCCR0 because EQUx = EQU0.<br>000b = OUT bit value<br>001b = Set<br>010b = Toggle/reset<br>011b = Set/reset<br>100b = Toggle<br>101b = Reset<br>110b = Toggle/set<br>111b = Reset/set |
| 4 | CCIE | RW | 0h | Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag.<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 3 | CCI | R | 0h | Capture/compare input. The selected input signal can be read by this bit. |
| 2 | OUT | RW | 0h | Output. For output mode 0, this bit directly controls the state of the output.<br>0b = Output low<br>1b = Output high |

**Table 9-6. TAxCCTLn Register Description (continued)**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 1 | COV | RW | 0h | Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software.<br>0b = No capture overflow occurred<br>1b = Capture overflow occurred |
| 0 | CCIFG | RW | 0h | Capture/compare interrupt flag<br>0b = No interrupt pending<br>1b = Interrupt pending |

### 9.3.4 TAxCCRn Register

Timer_A Capture/Compare n Register

**Figure 9-19. TAxCCRn Register**

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

**Table 9-7. TAxCCRn Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | TAxCCR0 | RW | 0h | Compare mode: TAxCCRn holds the data for the comparison to the timer value in the Timer_A Register, TAR.<br>Capture mode: The Timer_A Register, TAR, is copied into the TAxCCRn register when a capture is performed. |

### 9.3.5 TAxIV Register

Timer_Ax Interrupt Vector Register

**Figure 9-20. TAxIV Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| TAIV | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TAIV | | | | | | | |
| r0 | r0 | r0 | r0 | r-(0) | r-(0) | r-(0) | r0 |

**Table 9-8. TAxIV Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | TAIV | R | 0h | Timer_A interrupt vector value<br>00h = No interrupt pending<br>02h = Interrupt Source: Capture/compare 1; Interrupt Flag: TAxCCR1 CCIFG; Interrupt Priority: Highest<br>04h = Interrupt Source: Capture/compare 2; Interrupt Flag: TAxCCR2 CCIFG<br>06h = Interrupt Source: Capture/compare 3; Interrupt Flag: TAxCCR3 CCIFG<br>08h = Interrupt Source: Capture/compare 4; Interrupt Flag: TAxCCR4 CCIFG<br>0Ah = Interrupt Source: Capture/compare 5; Interrupt Flag: TAxCCR5 CCIFG<br>0Ch = Interrupt Source: Capture/compare 6; Interrupt Flag: TAxCCR6 CCIFG<br>0Eh = Interrupt Source: Timer overflow; Interrupt Flag: TAxCTL TAIFG; Interrupt Priority: Lowest |

### 9.3.6 TAxEX0 Register

Timer_Ax Expansion 0 Register

**Figure 9-21. TAxEX0 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | TAIDEX[1] | | |
| r0 | r0 | r0 | r0 | r0 | rw-(0) | rw-(0) | rw-(0) |

[1] After programming TAIDEX bits and configuration of the timer, set TACLR bit to ensure proper reset of the timer divider logic.

**Table 9-9. TAxEX0 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-3 | Reserved | R | 0h | Reserved. Reads as 0. |
| 2-0 | TAIDEX | RW | 0h | Input divider expansion. These bits along with the ID bits select the divider for the input clock.<br>000b = Divide by 1<br>001b = Divide by 2<br>010b = Divide by 3<br>011b = Divide by 4<br>100b = Divide by 5<br>101b = Divide by 6<br>110b = Divide by 7<br>111b = Divide by 8 |

Chapter 10
SLAU506–December 2014

# SD14 Module

The SD14 module is a multi-channel sigma-delta analog-to-digital converter with up to 14 bits of resolution. This chapter describes SD14 ADC and its analog input circuits.

**Topic** **Page**

## 10.1 SD14 Module Introduction

The SD14 module consists of an on-chip temperature sensor, a resistive bias interface, a programmable gain amplifier (PGA), and a sigma-delta analog-to-digital converter (ADC).

The ADC has up to eight fully differential multiplexed analog input pairs. The converter is based on a first-order sigma-delta modulator whose output is oversampled followed by a digital decimation filter. Two types of filter are available: a cascaded integrator-comb (CIC) filter with programmable rate change from 32 to 2048, and a moving average filter with programmable number of samples. Additional filtering can be done in software.

Features of the SD14 module include:

- 14-bit first-order sigma-delta architecture
- Up to eight multiplexed differential analog inputs
- Programmable sigma-delta output sampling frequency
- Selectable digital filter: CIC filter or moving average filter
- Programmable filter parameter
- Low power
- On-chip temperature sensor
- Resistive bias interface to measure thermistors and reference resistors

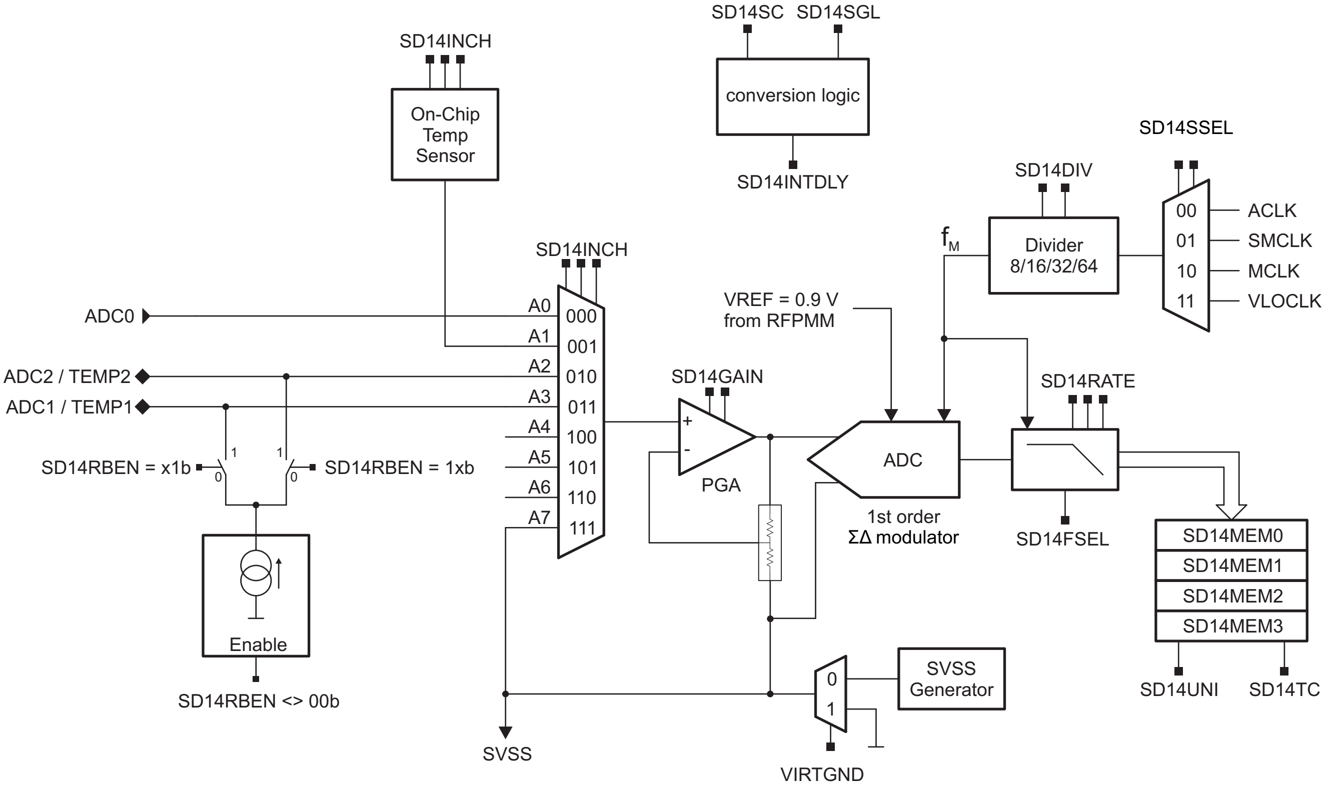


**Figure 10-1. SD14 Block Diagram**

## 10.2 SD14 ADC Operation

The SD14 module is configured with user software. The configuration and operation of the SD14 is discussed in the following sections.

### 10.2.1 Principle of Operation

A sigma-delta analog-to-digital converter basically consists of two parts: the analog part (called the modulator) and the digital part (called the decimation filter). The modulator of the SD14 provides a bit stream of zeros and ones to the digital decimation filter. The digital filter averages the bitstream from the modulator over a given number of bits (specified by the oversampling rate) and provides samples at a reduced rate for further processing to the CPU.

Averaging can be used to increase the signal-to-noise performance of a conversion (see Figure 10-2 a and b). With a conventional ADC, each factor-of-4 oversampling can improve the SNR by approximately 6 dB or 1 bit. To achieve a 14-bit resolution out of a simple 1-bit ADC would require an impractical oversampling rate of $4^{13}$ = 67.108.864. To overcome this limitation the sigma-delta modulator implements a technique called noise-shaping—due to an implemented feedback loop and integrators, the quantization noise is pushed to higher frequencies and, thus, much lower oversampling rates are sufficient to achieve high resolutions (see Figure 10-2 c).

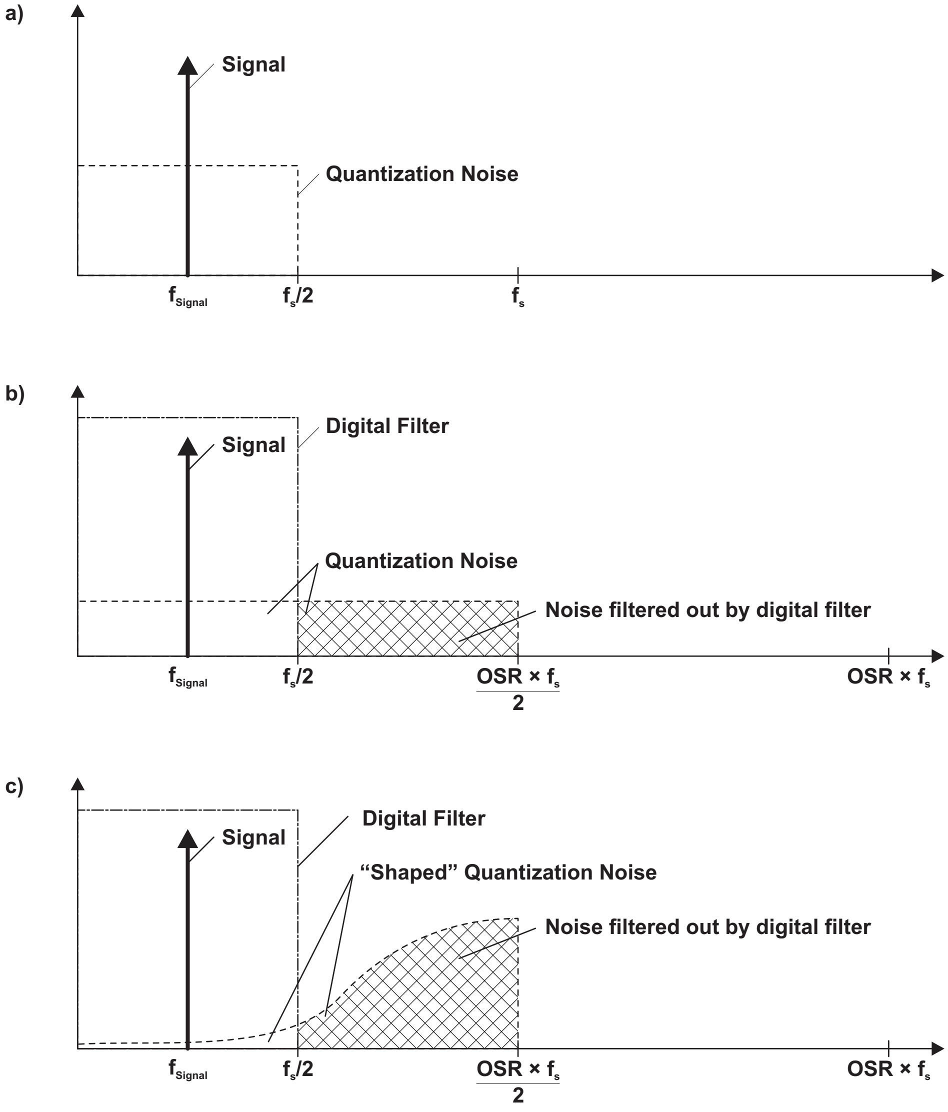


**Figure 10-2. Sigma-Delta Principle**

Within the Nyquist bandwidth from DC to $f_S/2$, an ideal N-Bit ADC has a Signal-to-Quantization Noise Ratio SNQR calculated with Equation 2

$$\text{SNQR} = 6.02 \cdot \text{N} + 1.76 \tag{2}$$

For example, an ideal 14-bit ADC would have a SNQR of about 86.05 dB. Increasing the resolution of an ADC would also increase the SNQR. From a hardware point of view building an 1-bit ADC is much easier than building a 14-bit ADC. To overcome the low SNQR of an 1-bit ADC ($\text{SNQR}_{\text{1-bit ADC}}$ = 7.78 dB) oversampling is used. With oversampling, the input signal $f_{\text{Signal}}$ is sampled at a much higher frequency $f_M$. The oversampling rate OSR is defined in Equation 3.

$$\text{OSR} = \frac{f_M}{f_N} = \frac{f_M}{2 \cdot B} \tag{3}$$

Where,

OSR is the oversampling ratio

B the bandwidth of the input signal

$f_M$ is Modulator (Sample) frequency

$f_N$ is Nyquist frequency

SNQR of an oversampled ideal N-bit ADC is calculated with Equation 4

$$\text{SNQR} = 6.02 \cdot \text{N} + 1.76 + 10\log\left(\frac{f_M}{2 \cdot B}\right) \tag{4}$$

Noise-Shaping as shown in Figure 10-2 c is used by sigma-delta ADCs. Equation 5 shows the formula to calculate the SNQR for a 1st order sigma delta ADC.

$$\text{SNQR}_{\text{SD}} = 6.02 \cdot \text{N} - 3.41 + 30\log\left(\frac{f_M}{2 \cdot B}\right) \tag{5}$$

To achieve the same SNQR of an ideal 14-bit ADC (SNQR = 86.05 dB) the 1st order sigma delta ADC must have an OSR of 1024 ($\text{SNQR}_{\text{SD}}$ = 92.92 dB). Figure 10-3 shows the relation between the $\text{SNQR}_{\text{SD}}$ of a 1st order sigma delta ADC and the Oversampling ratio. On the left axis the SNQR is shown. The right axis depicts the corresponding resolution of an ideal ADC. For example choosing an OSR = 256 for a 1st order sigma delta ADC would give a resolution equivalent to an ideal 12 bit ADC.

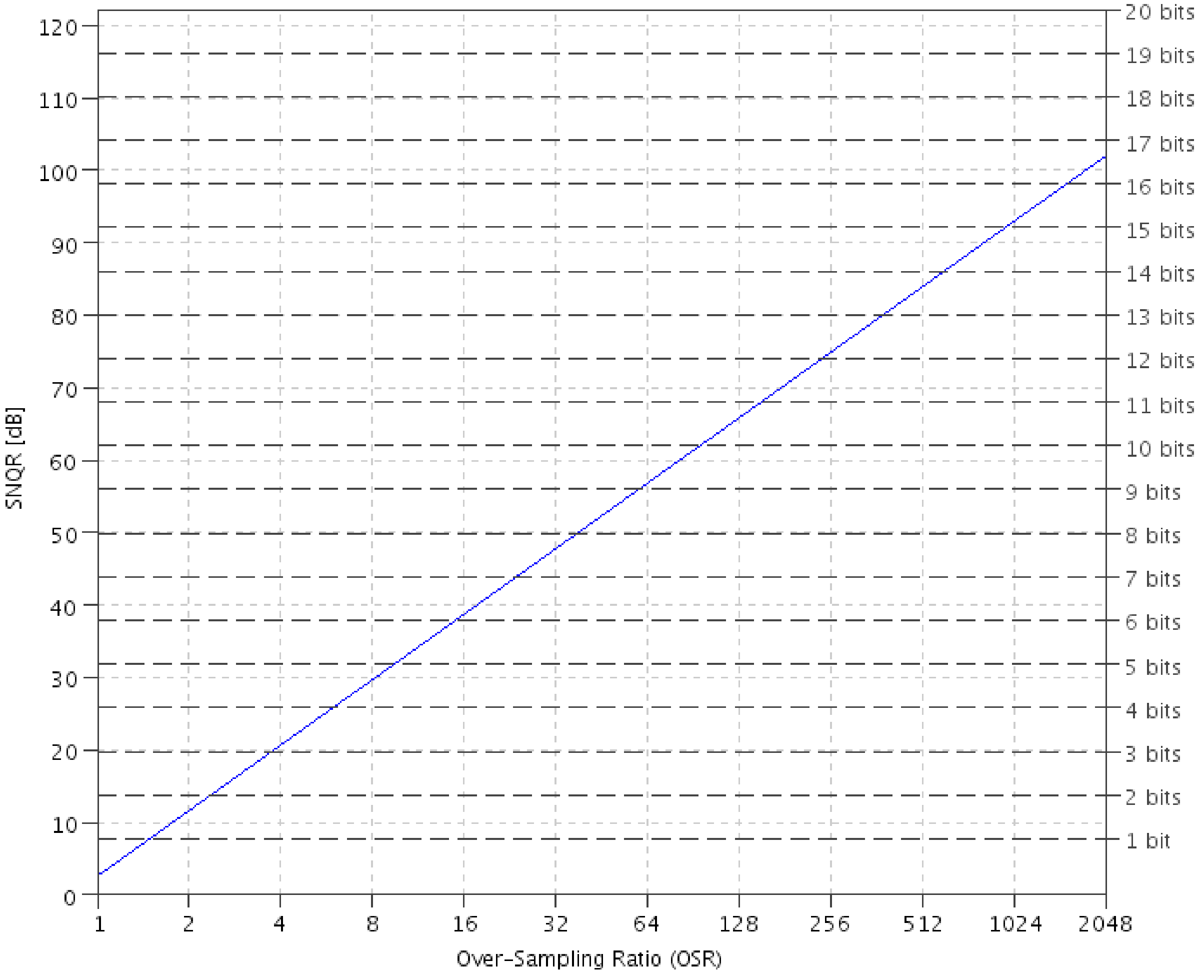


**Figure 10-3. SNQR of First-Order Sigma Delta vs Oversampling Ratio**

### 10.2.2 ADC Core

The analog-to-digital conversion is performed by a 1-bit first-order sigma-delta modulator. A single-bit comparator within the modulator quantizes the input signal and delivers samples with the modulator frequency $f_M$. The resulting bit stream is decimated by the digital filter into a higher precision conversion result.

### 10.2.3 Analog Input Setup

The SD14 can convert up to eight differential input pairs multiplexed into the PGA. The analog input is configured using the SD14INCTL register.

The SD14INCH bits select one of eight differential input pairs of the analog multiplexer. The gain of the PGA is selected with the SD14GAIN bits. A total of four gain settings are available.

During conversion, any modification to the SD14INCH and SD14GAIN bits becomes effective with the next decimation step of the digital filter. After these bits are modified, the next three conversions may be invalid due to the settling time of the digital filter. This can be handled automatically with the SD14INTDLY bits. When SD14INTDLY = 3h, conversion interrupt requests do not begin until the fourth or the eighth conversion after a start condition when using the CIC filter or the Moving average filter, respectively.

### 10.2.4 Resistive Bias Interface

The resistive bias interface consists of a constant current source delivering a constant current to either ADC1/TEMP1 or ADC2/TEMP2. The current generates a voltage over an external resistor which can be measured by the SD14 module. It is possible to connect a thermistor (temperature dependent resistor) or a reference resistor (temperature independent resistor) to pins TEMP1 and TEMP2.

To connect to either TEMP1 or TEMP2 the SD14RBEN bits must be set accordingly, see Table 10-1. To enable the resistive bias interface TEMP1, TEMP2 or both must be connected to the resistive bias interface (SD14RBEN <> 00b). If SD14RBEN = 00b the resistive bias interface is disabled and ADC1 and ADC2 can be used as normal ADC input channels.

**Table 10-1. Resistive Bias Interface Connection**

| SD14RBEN | Resistive Bias Interface Connected to |
|---|---|
| 00b | Disconnected from ADC channels,<br>Resistive Bias Interface disabled |
| 01b | ADC1 / TEMP1 |
| 10b | ADC2 / TEMP2 |
| 11b | TEMP1 and TEMP2 |

### 10.2.5 On-Chip Temperature Sensor

To enable the on-chip temperature sensor, set SD14INCH = 001b in the SD14INCTL register; otherwise, the temperature sensor is disabled.

Equation 6 can be used to calculate the temperature sensor output voltage:

$$V_{SENSOR} = t_c \times T \quad (6)$$

Where,

T = temperature in Kelvin

### 10.2.6 Analog Input Range and PGA

The full-scale differential input voltage range of the analog input pair is dependent on the gain setting of the programmable gain amplifier. The maximal full-scale range of the differential input is from *-VFS* to *+VFS* where *VFS* is shown in Equation 7.

$$V_{FS} = V_{REF} \times \frac{1}{G_{PGA}} \quad (7)$$

For a 0.9-V reference, the maximum full-scale input range for a gain of 1 is shown in Equation 8.

$$\pm V_{FS} = \pm V_{REF} = \pm 0.9\ \text{V} \quad (8)$$

### 10.2.7 Voltage Reference Generator

The SD14 module has a built-in 0.9-V reference. It is enabled or disabled automatically when a conversion is started or stopped, respectively.

### 10.2.8 SVSS Generator

The SVSS generator is used to provide a ground reference for the ADC. ADCs cannot amplify small voltages as shown in Figure 10-4 a). To overcome this behavior, an offset voltage SVSS is added to the input signal and is used as ground reference. With that the coordinate system is shifted to the linear area of the amplifier, see Figure 10-4 b), thus also small voltages can be amplified.

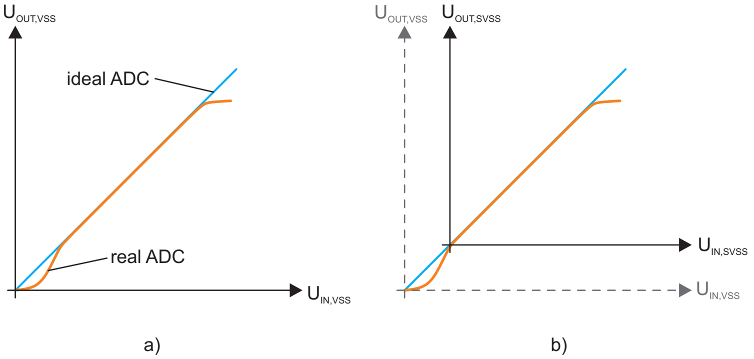


**Figure 10-4. Influence of SVSS to ADC Characteristic**

The SVSS generator output can be selected by setting bit VIRTGND and can be either the chip ground VSS or the internally generated SVSS.

**NOTE:** When turning on the SVSS Generator by setting VIRTGND from 1 to 0, a settling time $t_{Settling}$ must be applied. Please see device related datasheet.

### 10.2.9 Auto Power-Down

The SD14 is designed for low-power applications. When the SD14 is not actively converting, it is automatically disabled, and it is automatically re-enabled when a conversion is started.

## 10.3 Digital Filter

### 10.3.1 Overview

The purpose of the digital filter is to decimate the 1-bit output of the sigma-delta modulator and increase the precision to 14-bits.

Two types of filters are available: a CIC filter and a moving average filter. The filter is selectable using the SD14FILT bit in the SD14CTL1 register.

### 10.3.2 Cascaded Integrator-Comb (CIC) Filter

CIC filters are a simple and hardware-economical implementation of decimation filters. They are well suited for large rate changes where the sampling frequency is much larger than the bandwidth of the input signal. The modulator frequency $f_M$ of the input signal is programmable via CPU using the register SD14CLKDIV. The clock divider SD14CLKDIV must be chosen to match frequencies of $f_M$ = 500 Hz .. 2000 Hz.

#### 10.3.2.1 SINC$^2$ Filter

The structure of a SINC$^2$ filter is shown in Figure 10-5.

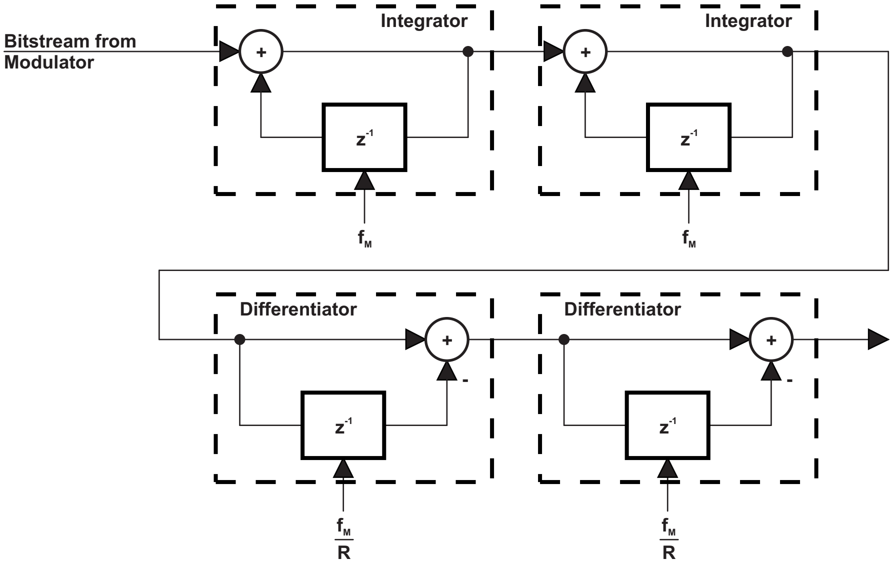


**Figure 10-5. SINC$^2$ Filter Structure**

The transfer function is described in the z-domain by:

$$H(z) = \left( \frac{1}{OSR} \cdot \frac{1 - z^{-OSR}}{1 - z^{-1}} \right)^2 \tag{9}$$

The transfer function is described in the frequency domain by:

$$H(f) = \left( \frac{\text{sinc}\left( OSR \cdot \pi \cdot \frac{f}{f_M} \right)}{\text{sinc}\left( \pi \cdot \frac{f}{f_M} \right)} \right)^2 = \left( \frac{1}{OSR} \times \frac{\sin\left( OSR \cdot \pi \cdot \frac{f}{f_M} \right)}{\sin\left( \pi \cdot \frac{f}{f_M} \right)} \right)^2 \tag{10}$$

where the oversampling rate, OSR, is the ratio of the modulator frequency $f_M$ to the sample frequency $f_S$. Figure 10-6 shows the filter's frequency response for an OSR of 32. The first filter notch is always at $f_S = f_M/OSR$. The notch's frequency can be adjusted by changing the modulator's frequency, $f_M$, using SD24SSELx, SD24PDIVx, and SD24DIVx or by adjusting the oversampling rate using the SD24BOSRx registers.

The digital filter for each enabled ADC converter completes the decimation of the digital bit stream and outputs new conversion results to the corresponding SD24BMEMx register at the sample frequency $f_S$.

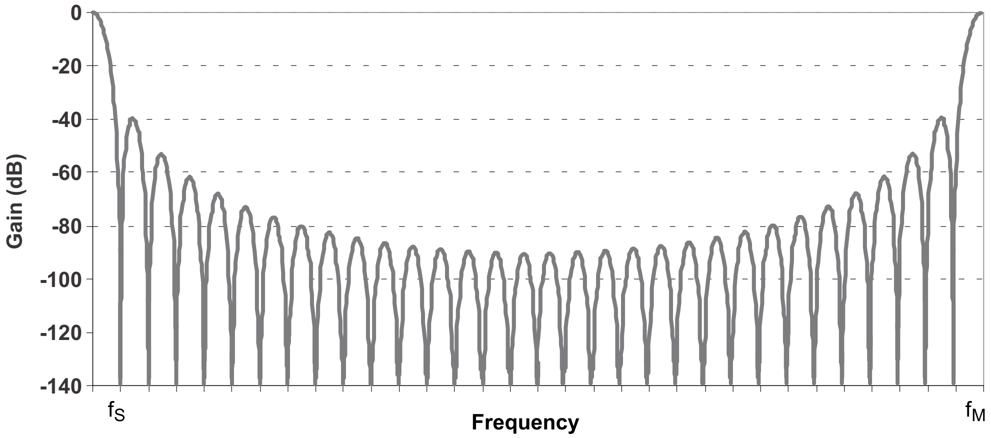


**Figure 10-6. Comb Filter's Frequency Response With OSR = 32**

The CIC filter computes a more precise estimation of the input signal at a lower sampling frequency. The filter output sampling frequency is determined by the decimation rate R and is programmable using the SD14RATE register. To decrease the sampling rate of the modulator bit-stream, the decimation filter takes only every R$^{th}$ sample. For example if R = 64, only every 64$^{th}$ sample is taken from the modulator output stream. The accuracy of the conversion result depends on the decimation rate R, the modulator (sampling) frequency $f_M$ and the bandwidth B of the input signal. The lower the decimation rate R is, the faster is the conversion time, but the lower is the accuracy of the conversion result. Table 10-2 shows the relation between decimation ratio R, the expected accuracy of the conversion result and the measurement time. A maximum input signal bandwidth of B = 1 Hz and a modulator frequency of $f_M$ = 2 kHz is assumed.

**Table 10-2. Relation of Decimation Ratio and Accuracy**

| Decimation Ratio, R | Accuracy of Conversion Result (bits) | Conversion Time ($f_M$ = 2 kHz) (ms) |
|---|---|---|
| 32 | 7 | 16 |
| 64 | 9 | 32 |
| 128 | 10 | 64 |
| 256 | 12 | 128 |
| 512 | 13 | 256 |
| 1024 | 15 | 512 |
| 2048 | 16 | 1024 |

### 10.3.3 Moving Average Filter

The moving average filter is a simple FIR filter that computes the mean of a determined number of data samples.

The response of the filter in Z-domain is shown in Equation 11.

$$H(z) = \sum_{k=0}^{N-1} z^{-k} = \frac{1-z^{-N}}{1-z^{-1}} \tag{11}$$

Where,*N*

is the number of samples

The magnitude of the frequency response of the filter is shown in Equation 12.

$$\left|H(f)\right| = \frac{\sin\left(p \times N \times \frac{f}{F_S}\right)}{\sin\left(p \times \frac{f}{F_S}\right)} \tag{12}$$

Where,

*F S* is the input sampling frequency

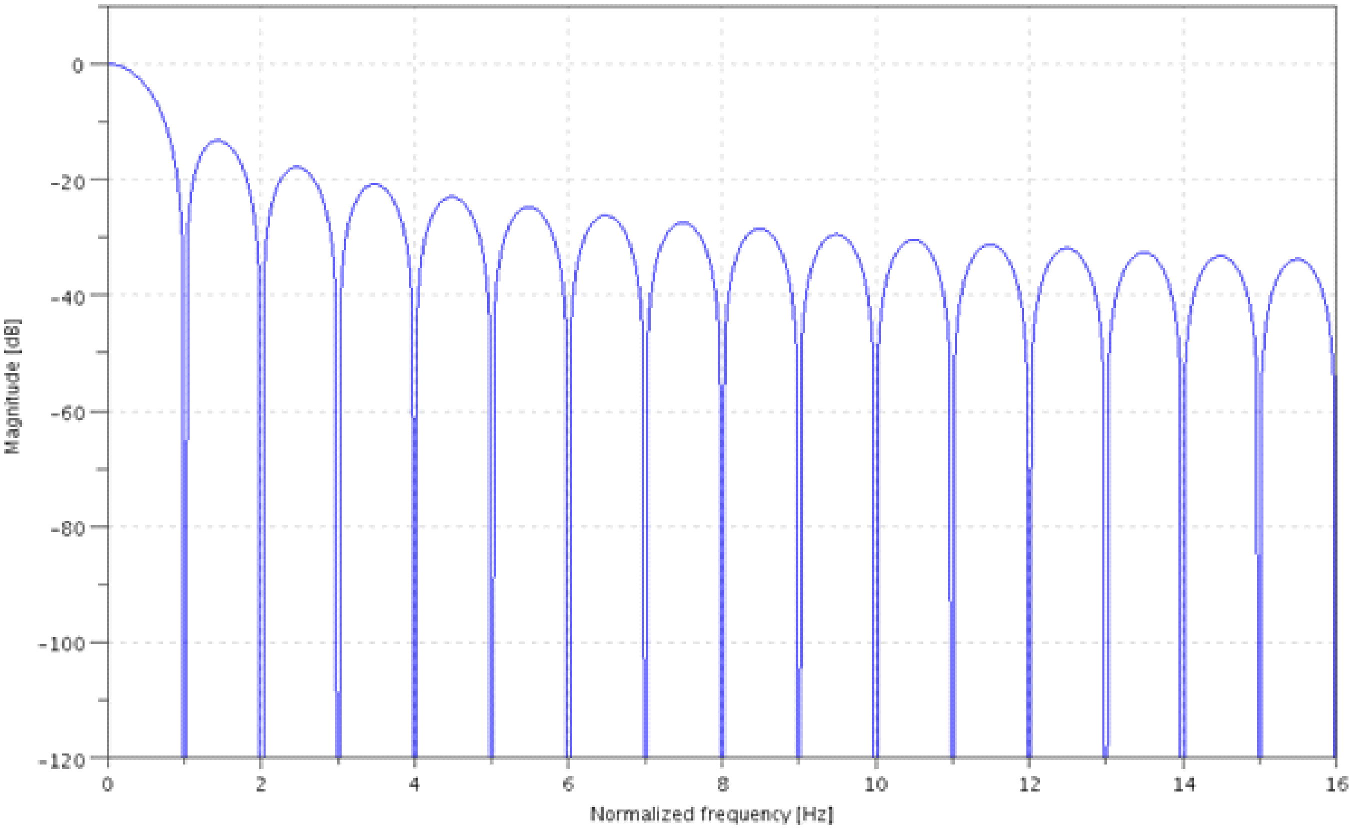


**Figure 10-7. Magnitude of the Frequency Response of a Moving Average Filter**

To ensure a resolution of 14-bits at the output of the filter, the number of samples needed is given by Equation 13.

$$N = 2^{nb\_bits-1} = 2^{13} \quad (13)$$

The number of samples used is programmable via CPU using bits SD14RATE in register SD14CTL.

### 10.3.4 Conversion Memory Register (SD14MEMx)

Conversion results are moved to the SD14MEM0 register with each decimation step of the digital filter. The SD14IFG bit is set when new data is written to SD14MEM0. Last filter output result is always written in SD14MEM0 register and previous results are shifted into successive SD14MEMx registers. So if SD14INTDLY is set to a value different than 00h intermediate results are stored in SD14MEM1 to SD14MEM3 registers (that is, SD14MEM1 contains the last intermediate result). SD14IFG is automatically cleared when SD14MEM0 is read by the CPU or may be cleared with software.

### 10.3.5 Output Data Format

The output data format is configurable in 2s complement, offset binary, or unipolar mode as shown in Table 10-3. The data format is selected by the SD14TC and SD14UNI bits (SD14CTL register). The result in is right justified.

If data format is 2s complement (SD14TC = 1) and the result is negative (13thbit = 1), 14th and 15th bit of SD14MEM0 also become 1 to ease further calculation.

**Table 10-3. Conversion Result Data Format**

| SD14TC | SD14UNI | Format | VAIN+ | VAIN- | SD14MEMx |
|---|---|---|---|---|---|
| 0 | 0 | Bipolar Offset Binary | FS | 0 | 3FFFh |
| | | | 0 | 0 | 2000h |
| | | | 0 | FS | 0000h |
| 0 | 1 | Unipolar | FS | 0 | 3FFFh |
| | | | 0 | 0 | 0000h |
| 1 | x | Bipolar 2s Complement | FS | 0 | 1FFFh |
| | | | 0 | 0 | 0000h |
| | | | 0 | FS | 2000h |

**NOTE: Offset Measurements and Data Format**

Any offset measurement done would be appropriate only when the channel is operating under bipolar mode with SD14UNI = 0.

Figure 10-8 shows the relationship between the full-scale input voltage range from -VFS to +VFS and the conversion result. The data formats are illustrated.

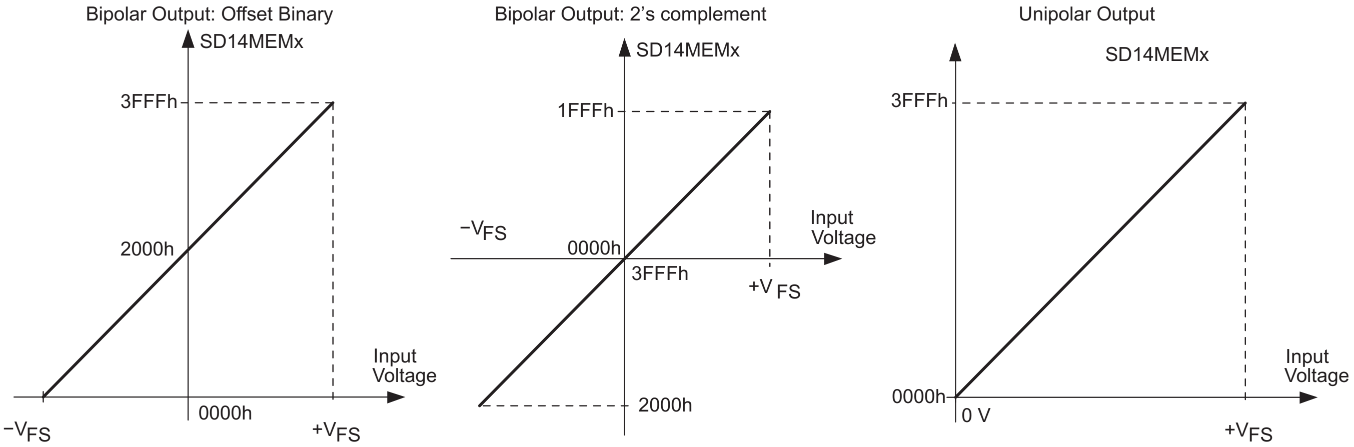


**Figure 10-8. Input Voltage vs Digital Output**

## 10.4 Conversion Modes

The SD14 module can be configured for two modes of operation, listed in Table 10-4. The SD14SGL bit selects the conversion mode.

**Table 10-4. Conversion Mode Summary**

| SD14SGL | Mode | Operation |
|---|---|---|
| 1 | Single Conversion | The channel is converted once. |
| 0 | Continuous conversion | The channel is converted continuously. |

### 10.4.1 Single Conversion

Setting the SD14SC bit of the channel initiates one conversion on that channel when SD14SGL = 1. The SD14SC bit is automatically cleared after conversion completion.

Clearing SD14SC before the conversion is completed immediately stops conversion of the channel, the channel is powered down and the corresponding digital filter is turned off and the filter is reset. The value in SD14MEM0 can change when SD14SC is cleared. It is recommended that the conversion data in SD14MEM0 is read prior to clearing SD14SC to avoid reading an invalid result. Figure 10-9 shows single conversion mode.

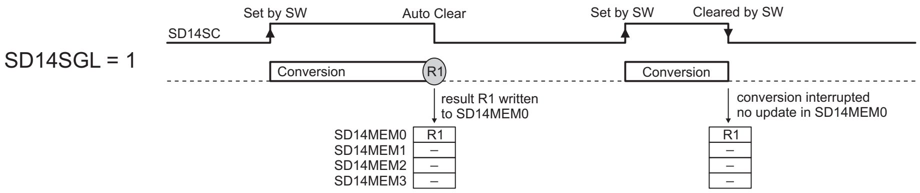


**Figure 10-9. Single Conversion Mode**

### 10.4.2 Continuous Conversion

When SD14SGL = 0 continuous conversion mode is selected. Conversion of the channel begins when SD14SC is set and continues until the SD14SC bit is cleared by software. Clearing SD14SC immediately stops conversion of the selected channel, the channel is powered down and the corresponding digital filter is turned off and the filter is reset. However during continuous conversion mode the digital filter is not reset between conversions. The value in SD14MEM0 can change when SD14SC is cleared. It is recommended that the conversion data in SD14MEM0 is read prior to clearing SD14SC to avoid reading an invalid result. Figure 10-10 shows continuous conversion mode.

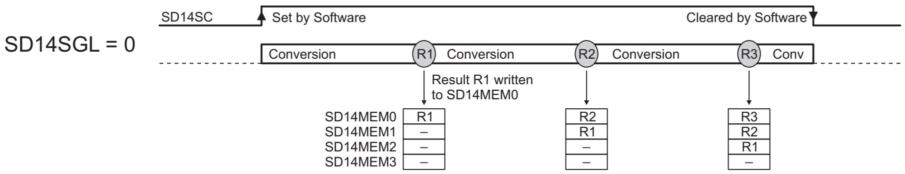


**Figure 10-10. Continuous Conversion Mode**

## 10.5 Interrupt Handling

The SD14 module has two interrupt sources:

- SD14IFG
- SD14OVIFG

The SD14IFG bit is set when the SD14MEM0 memory register is written with a conversion result. An interrupt request is generated if the SD14IE bit and the GIE bit are set. The SD14 overflow condition occurs when a conversion result is written to the SD14MEM location before the previous conversion result was read.

### 10.5.1 SD14IV, Interrupt Vector Generator

All SD14 interrupt sources are prioritized and combined to source a single interrupt vector. SD14IV is used to determine which enabled SD14 interrupt source requested an interrupt. The highest priority SD14 interrupt request that is enabled generates a number in the SD14IV register (see register description). This number can be evaluated or added to the program counter to automatically enter the appropriate software routine. Disabled SD14 interrupts do not affect the SD14IV value.

Any access, read or write, of the SD14IV register has no effect on the SD14OVIFG or SD14IFG flags. The SD14IFG flags are reset by reading the SD14MEM0 register or by clearing the flags in software. SD14OVIFG bits can only be reset with software.

If another interrupt is pending after servicing of an interrupt, another interrupt is generated. For example, if the SD14OVIFG and one or more SD14IFG interrupts are pending when the interrupt service routine accesses the SD14IV register, the SD14OVIFG interrupt condition is serviced first and the corresponding flag must be cleared in software. After the RETI instruction of the interrupt service routine is executed, the highest priority SD14IFG pending generates another interrupt request.

### 10.5.2 Interrupt Delay Operation

The SD14INTDLYx bits control the timing for the first interrupt service request for the corresponding channel. This feature delays the interrupt request for a completed conversion by up to four conversion cycles allowing the digital filter to settle prior to generating an interrupt request. The delay is applied each time the SD14SC bit is set or when the SD14GAIN or SD14INCH bits for the channel are modified. SD14INTDLY disables overflow interrupt generation for the selected number of delay cycles. Interrupt requests for the delayed conversions are not generated during the delay. Figure 10-11 and Figure 10-12 show the single and continuous conversion operations, respectively, and the memory contents depending on the chosen interrupt delay defined by SD14DLY bits.

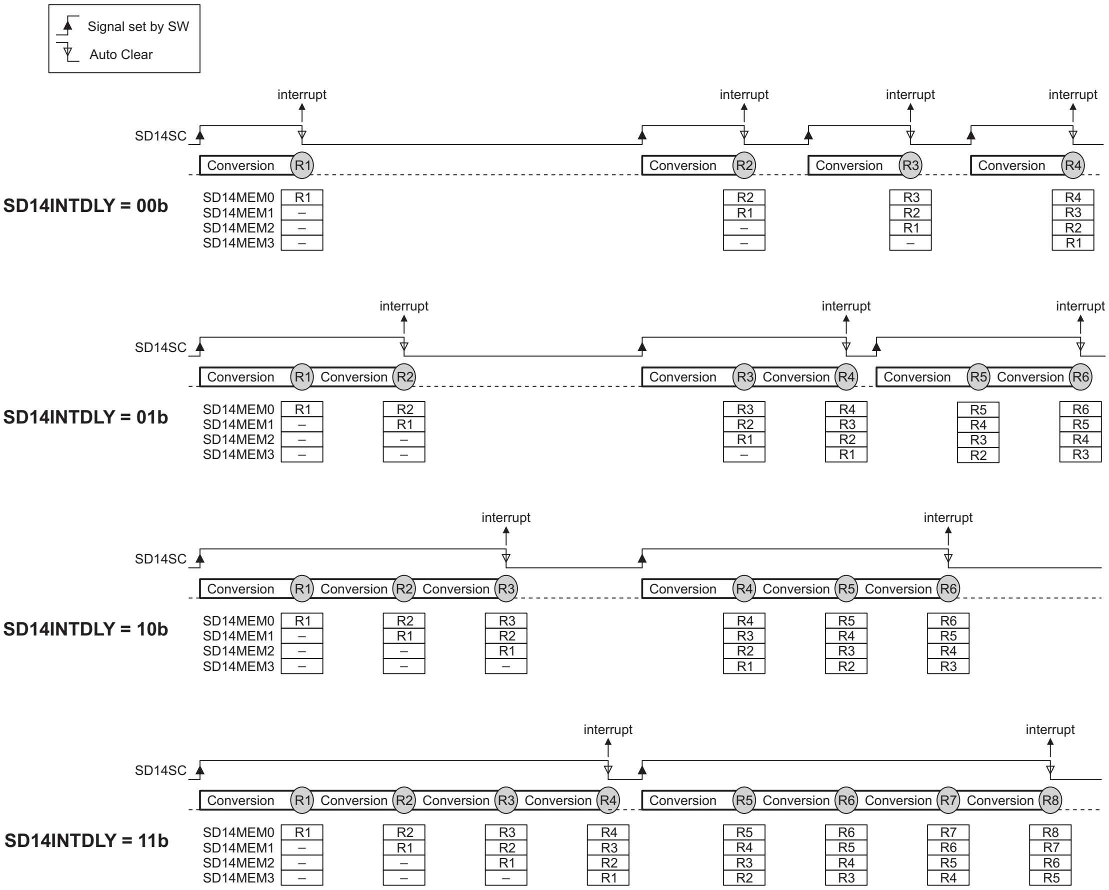


**Figure 10-11. Single Conversion Interrupt Delay**

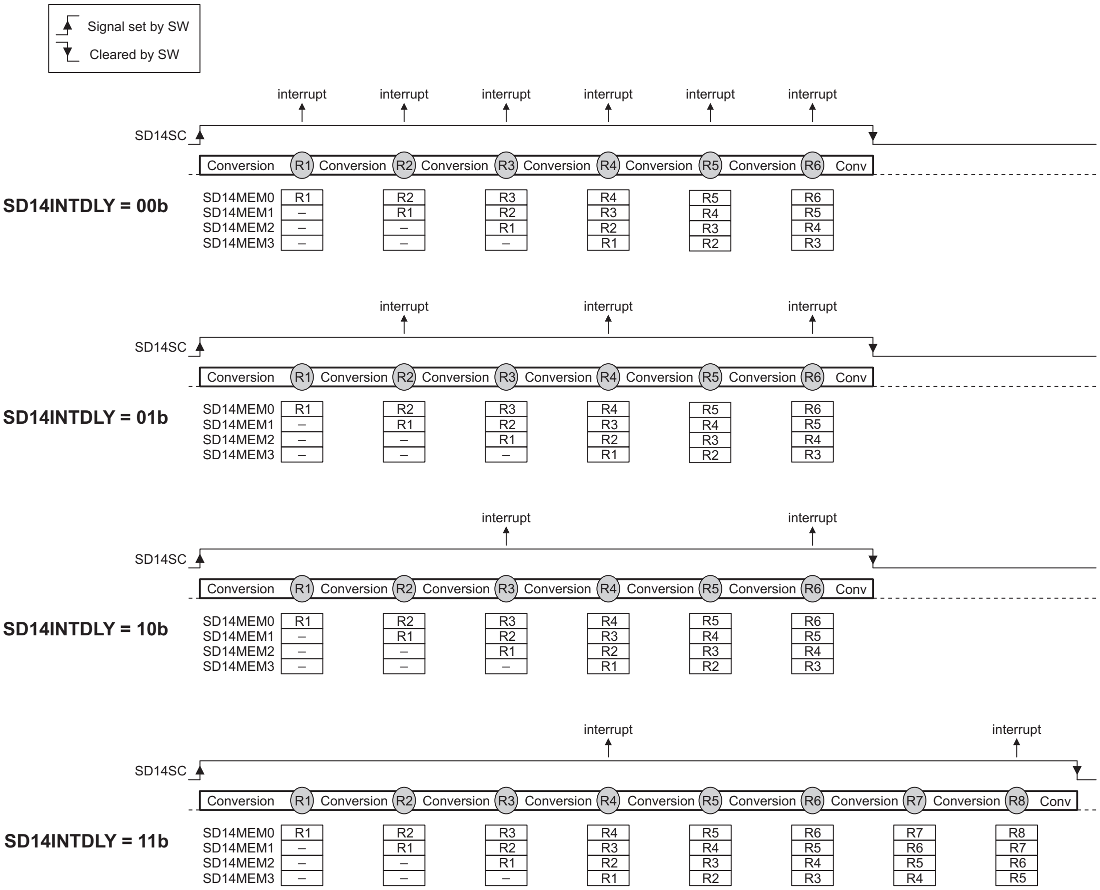


Figure 10-12. Continuous Conversion Interrupt Delay

## 10.6 SD14 Registers

Table 10-5 lists the memory-mapped registers for the SD14. See your device-specific data manual for the memory address of these registers. All other register offset addresses not listed in Table 10-5 should be considered as reserved locations and the register contents should not be modified.

**Table 10-5. SD14 Registers**

| Offset | Acronym | Register Name | Type | Access | Reset | Section |
|---|---|---|---|---|---|---|
| 00h | SD14CTL0 | SD14 Control Register 0 | read/write | Word | 0080h | Section 10.6.1 |
| | SD14CTL0_L | | read/write | Byte | 08h | |
| | SD14CTL0_H | | read/write | Byte | 00h | |
| 02h | SD14CTL1 | SD14 Control Register 1 | read/write | Byte | 00C0h | Section 10.6.2 |
| | SD14CTL1_L | | read/write | Byte | C0h | |
| | SD14CTL1_H | | read/write | Byte | 00h | |
| 04h | SD14MEM0 | SD14 Conversion Result | read/write | Byte | 0000h | Section 10.6.3 |
| | SD14MEM0_L | | read/write | Byte | 00h | |
| | SD14MEM0_H | | read/write | Byte | 00h | |
| 06h | SD14MEM1 | SD14 Intermediate Conversion Result | read/write | Word | 0000h | Section 10.6.4 |
| | SD14MEM1_L | | read/write | Byte | 00h | |
| | SD14MEM1_H | | read/write | Byte | 00h | |
| 08h | SD14MEM2 | SD14 Intermediate Conversion Result | read/write | Word | 0000h | Section 10.6.5 |
| | SD14MEM2_L | | read/write | Byte | 00h | |
| | SD14MEM2_H | | read/write | Byte | 00h | |
| 0Ah | SD14MEM3 | SD14 Intermediate Conversion Result | read/write | Word | 0000h | Section 10.6.6 |
| | SD14MEM3_L | | read/write | Byte | 00h | |
| | SD14MEM3_H | | read/write | Byte | 00h | |
| 0Ch | SD14IV | SD14 Interrupt Vector | read/write | Word | 0000h | Section 10.6.7 |
| | SD14IV_L | | read/write | Byte | 00h | |
| | SD14IV_H | | read/write | Byte | 00h | |

### 10.6.1 SD14CTL0 Register

SD14 Control Register 0

**Figure 10-13. SD14CTL0 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | VIRTGND | SD14OVIG | SD14OVIE | SD14IFG | SD14IE |
| r-0 | r-0 | r-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SD14DIV[1] | | SD14SSEL[1] | | SD14SGL | SD14SC | Reserved | SD14EN |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-1 | rw-0 | r-0 | rw-0 |

[1] Can be modified only if SD14EN = 0.

**Table 10-6. SD14CTL0 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-13 | Reserved | R | 0h | Reserved, read as 0h |
| 12 | VIRTGND | RW | 0h | Defines the virtual ground reference for the ADC<br>0b = SVSS as reference for ADC<br>1b = AVSS as reference for ADC |
| 11 | SD14OVIG | RW | 0h | SD14 overflow interrupt flag<br>0b = No overflow interrupt pending<br>1b = Overflow interrupt pending |
| 10 | SD14OVIE | RW | 0h | SD14 overflow interrupt enable<br>0b = Overflow interrupt disabled<br>1b = Overflow interrupt enabled |
| 9 | SD14IFG | RW | 0h | SD14 interrupt flag. SD14IFG is set when new conversion result is available. SD14IFG is automatically reset when the corresponding SD14MEM register is read, or may be cleared with software.<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 8 | SD14IE | RW | 0h | SD14 interrupt enable. The GIE bit must also be set to enable th interrupt.<br>0b = interrupt disabled<br>1b = interrupt enabled |
| 7-6 | SD14DIV[1] | RW | 0h | SD14 sigma-delta Clock divider<br>00b = Divide by 8<br>01b = Divide by 16<br>10b = Divide by 32<br>11b = Divide by 64 |
| 5-4 | SD14SSEL[1] | RW | 0h | SD14 sigma-delta Clock source select<br>00b = ACLK<br>01b = SMCLK<br>10b = MCLK<br>11b = VLOCLK |
| 3 | SD14SGL | RW | 1h | Conversion mode select<br>0b = Continuous conversion mode<br>1b = Single conversion mode |
| 2 | SD14SC | RW | 0h | Start conversion<br>0b = Stop conversion<br>1b = Start conversion |
| 1 | Reserved | R | 0h | Reserved, read as 0h |

[1] Can be modified only if SD14EN = 0.

### Table 10-6. SD14CTL0 Register Description (continued)

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 0 | SD14EN | RW | 0h | SD14 module enable<br>0b = SD14 module disabled<br>1b = SD14 module enabled |

### 10.6.2 SD14CTL1 Register

SD14 Input Control Register 1

**Figure 10-14. SD14CTL1 Register**

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td></tr>
<tr><td colspan="2">SD14RBEN</td><td>SD14TC</td><td>SD14UNI</td><td colspan="3">SDRATE</td><td>SD14FILT</td></tr>
<tr><td>rw-0</td><td>rw-0</td><td>rw-0</td><td>rw-0</td><td>rw-0</td><td>rw-0</td><td>rw-0</td><td>rw-0</td></tr>
</table>

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="2">SD14INTDLY</td><td>Reserved</td><td colspan="2">SD14GAIN</td><td colspan="3">SD14INCH</td></tr>
<tr><td>rw-1</td><td>rw-1</td><td>r0</td><td>rw-0</td><td>rw-0</td><td>rw-0</td><td>rw-0</td><td>rw-0</td></tr>
</table>

**Table 10-7. SD14CTL1 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-14 | SD14RBEN | RW | 0h | Connects Resistive Bias Interface to TEMP1/TEMP2<br>00b = Not connected to TEMP1 or TEMP2<br>01b = Connected to TEMP1<br>10b = Connected to TEMP2<br>11b = Connected to TEMP1 and TEMP2 |
| 13 | SD14TC | RW | 0h | Twos complement format enable<br>0b = Offset binary format selected<br>1b = Twos complement format selected |
| 12 | SD14UNI | RW | 0h | Offset binary format selection<br>0b = Bipolar format selected<br>1b = Unipolar format selected |
| 11-9 | SDRATE | RW | 0h | SD14 rate change factor. Select decimation ratio R when CIC filter is selected or number of samples when Average filter is selected.<br>000b = CIC filter decimation ratio: 32; Moving average number of samples: 4096<br>001b = CIC filter decimation ratio: 64; Moving average number of samples: 8192<br>010b = CIC filter decimation ratio: 128; Moving average number of samples: 16384<br>011b = CIC filter decimation ratio: 256; Moving average number of samples: 32768<br>100b = CIC filter decimation ratio: 512; Moving average number of samples: -<br>101b = CIC filter decimation ratio: 1024; Moving average number of samples: -<br>110b = CIC filter decimation ratio: 2048; Moving average number of samples: -<br>111b = CIC filter decimation ratio: -; Moving average number of samples: - |
| 8 | SD14FILT | RW | 0h | SD14 digital filter selection.<br>0b = CIC filter selected<br>1b = Moving Average filer selected |
| 7-6 | SD14INTDLY | RW | 3h | Interrupt delay generation after conversion start. These bits select after which sample the first interrupt should be triggered after conversion start.<br>00b = First conversion<br>01b = Second conversion<br>10b = Third conversion<br>11b = Fourth conversion |
| 5 | Reserved | R | 0h | Reserved, read as 0h |
| 4-3 | SD14GAIN | RW | 0h | SD14 preamplifier gain<br>00b = 1x<br>01b = 2x<br>10b = 4x<br>11b = 8x |

### Table 10-7. SD14CTL1 Register Description (continued)

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 2-0 | SD14INCH | RW | 0h | SD14 channel differential pair input selection<br>000b = A0 Voltage input ADC0<br>001b = A1 Voltage Input from On-Chip Temp Sensor; enables On-Chip Temperature Sensor<br>010b = A2 Voltage Input from ADC2/TEMP2 (Thermistor)<br>011b = A3 Voltage Input from ADC1/TEMP1 (Reference Resistor)<br>100b = A4 NC<br>101b = A5 NC<br>110b = A6 NC<br>111b = A7 AVSS/SVSS (depending on bit VIRTGND) |

### 10.6.3 SD14MEM0 Register

SD14 Conversion Result 0

**Figure 10-15. SD14MEM0 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | SD14RES | | | | | |
| r0 | r0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SD14RES | | | | | | | |
| r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 |

**Table 10-8. SD14MEM0 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-14 | Reserved | R | 0h | Reserved, read as 0h<br>If result format is 2s complement and the result is negative, these bits are filled with 1. |
| 13-0 | SD14RES | R | 0h | Conversion result. The 14-bit conversion results are right justified. Bit 13 is the MSB. Writing to the conversion memory register corrupts the results.<br>The contents of this field depend on the value of SD14INTDLY:<br>00b = First sample<br>01b = Second sample<br>10b = Third sample<br>11b = Fourth sample |

### 10.6.4 SD14MEM1 Register

SD14 Intermediate Conversion Result 1

**Figure 10-16. SD14MEM1 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | SD14RES | | | | | |
| r0 | r0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SD14RES | | | | | | | |
| r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 |

**Table 10-9. SD14MEM1 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-14 | Reserved | R | 0h | Reserved, read as 0h |
| 13-0 | SD14RES | R | 0h | First intermediate conversion result.<br>The contents of this field depend on the value of SD14INTDLY:<br>00b = n/a<br>01b = First sample<br>10b = Second sample<br>11b = Third sample |

### 10.6.5 SD14MEM2 Register

SD14 Intermediate Conversion Result 2

**Figure 10-17. SD14MEM2 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | SD14RES | | | | | |
| r0 | r0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SD14RES | | | | | | | |
| r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 |

**Table 10-10. SD14MEM2 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-14 | Reserved | R | 0h | Reserved, read as 0h |
| 13-0 | SD14RES | R | 0h | Second intermediate conversion result.<br>The contents of this field depend on the value of SD14INTDLY:<br>00b = n/a<br>01b = n/a<br>10b = First sample<br>11b = Second sample |

### 10.6.6 SD14MEM3 Register

SD14 Intermediate Conversion Result 3

**Figure 10-18. SD14MEM3 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | SD14RES | | | | | |
| r0 | r0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SD14RES | | | | | | | |
| r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 |

**Table 10-11. SD14MEM3 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-14 | Reserved | R | 0h | Reserved, read as 0h |
| 13-0 | SD14RES | R | 0h | First intermediate conversion result.<br>The contents of this field depend on the value of SD14INTDLY:<br>00b = n/a<br>01b = n/a<br>10b = n/a<br>11b = First sample |

### 10.6.7 SD14IV Register

SD14 Interrupt Vector

**Figure 10-19. SD14IV Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| SD14IV | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SD14IV | | | | | | | |
| r0 | r0 | r0 | r0 | r-0 | r-0 | r-0 | r0 |

**Table 10-12. SD14IV Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | SD14IV | R | 0h | SD14 interrupt vector table<br>00h = no interrupt pending<br>02h = SD14MEM overflow (SD14OVIFG); Priority: Highest<br>04h = SD14MEM new result (SD14IFG); Priority: Lowest |

Chapter 11

SLAU506–December 2014

# Enhanced Universal Serial Communication Interface (eUSCI) – SPI Mode

The enhanced universal serial communication interfaces, eUSCI_A and eUSCI_B, support multiple serial communication modes with one hardware module. This chapter discusses the operation of the synchronous peripheral interface (SPI) mode.

**Topic** **Page**

## 11.1 Enhanced Universal Serial Communication Interfaces (eUSCI_A, eUSCI_B) Overview

Both the eUSCI_A and the eUSCI_B support serial communication in SPI mode.

## 11.2 eUSCI Introduction – SPI Mode

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

SPI mode features include:

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

Figure 11-1 shows the eUSCI when configured for SPI mode.

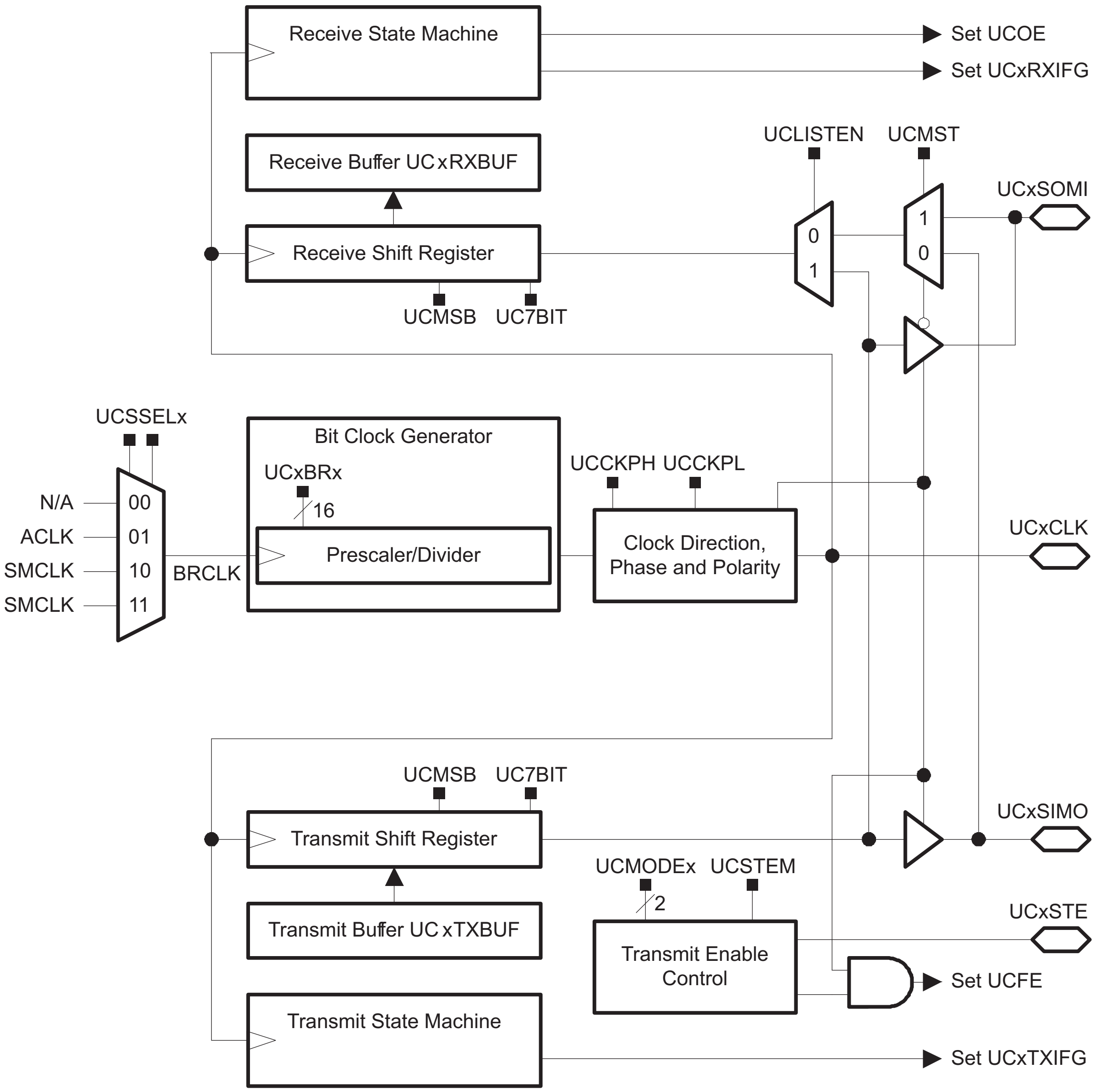


**Figure 11-1. eUSCI Block Diagram – SPI Mode**

## 11.3 eUSCI Operation – SPI Mode

In SPI mode, serial data is transmitted and received by multiple devices using a shared clock provided by the master. An additional pin controlled by the master, UCxSTE, is provided to enable a device to receive and transmit data.

Three or four signals are used for SPI data exchange:

- UCxSIMO – slave in, master out

  Master mode: UCxSIMO is the data output line.
  Slave mode: UCxSIMO is the data input line.
- UCxSOMI – slave out, master in

  Master mode: UCxSOMI is the data input line.
  Slave mode: UCxSOMI is the data output line.
- UCxCLK – eUSCI SPI clock

  Master mode: UCxCLK is an output.
  Slave mode: UCxCLK is an input.
- UCxSTE – slave transmit enable.

  Used in 4-pin mode to allow multiple masters on a single bus. Not used in 3-pin mode. Table 11-1 describes the UCxSTE operation.

**Table 11-1. UCxSTE Operation**

| UCMODEx | UCxSTE Active State | UCxSTE | Slave | Master |
|---|---|---|---|---|
| 01 | High | 0 | Inactive | Active |
| | | 1 | Active | Inactive |
| 10 | Low | 0 | Active | Inactive |
| | | 1 | Inactive | Active |

### 11.3.1 eUSCI Initialization and Reset

The eUSCI is reset by a PUC or by the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the eUSCI in a reset condition. When set, the UCSWRST bit resets the UCRXIE, UCTXIE, UCRXIFG, UCOE, and UCFE bits, and sets the UCTXIFG flag. Clearing UCSWRST releases the eUSCI for operation.

Configuring and reconfiguring the eUSCI module should be done when UCSWRST is set to avoid unpredictable behavior.

**NOTE: Initializing or reconfiguring the eUSCI module**

The recommended eUSCI initialization/reconfiguration process is:

1. Set UCSWRST.

   ```
   BIS.B #UCSWRST,&UCxCTL1
   ```
2. Initialize all eUSCI registers with UCSWRST = 1 (including UCxCTL1).
3. Configure ports.
4. Clear UCSWRST via software.

   ```
   BIC.B #UCSWRST,&UCxCTL1
   ```
5. Enable interrupts (optional) via UCRXIE or UCTXIE.

### 11.3.2 Character Format

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

**NOTE: Default character format**

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

**NOTE: Character format for figures**

Figures throughout this chapter use MSB-first format.

### 11.3.3 Master Mode

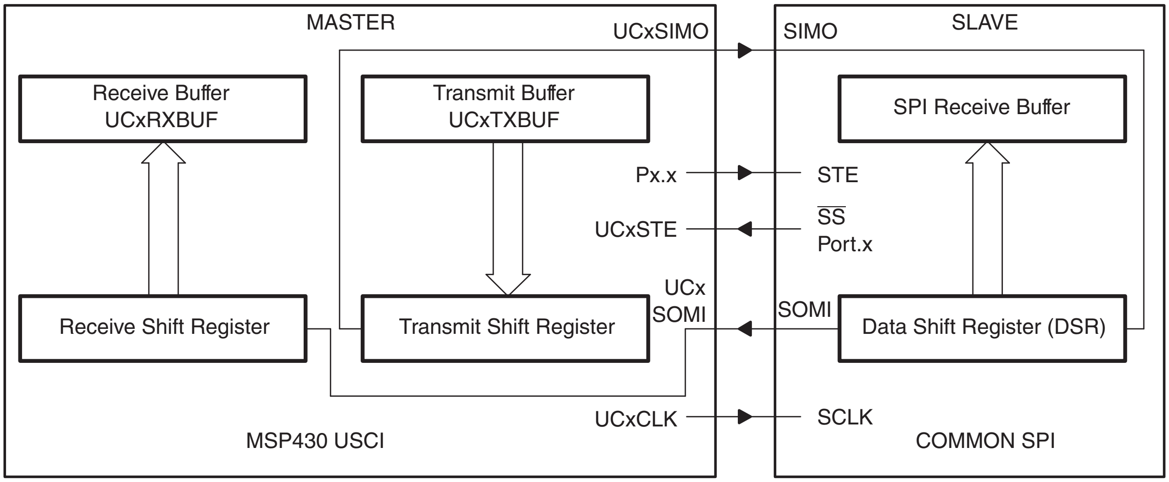


**Figure 11-2. eUSCI Master and External Slave (UCSTEM = 0)**

Figure 11-2 shows the eUSCI as a master in both 3-pin and 4-pin configurations. The eUSCI initiates data transfer when data is moved to the transmit data buffer UCxTXBUF. The UCxTXBUF data is moved to the transmit (TX) shift register when the TX shift register is empty, initiating data transfer on UCxSIMO starting with either the MSB or LSB, depending on the UCMSB setting. Data on UCxSOMI is shifted into the receive shift register on the opposite clock edge. When the character is received, the receive data is moved from the receive (RX) shift register to the received data buffer UCxRXBUF and the receive interrupt flag UCRXIFG is set, indicating the RX/TX operation is complete.

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

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

There two different options for configuring the eUSCI as a 4-pin master, which are described in the next sections:

- The fourth pin is used as input to prevent conflicts with other masters (UCSTEM = 0).
- The fourth pin is used as output to generate a slave enable signal (UCSTEM = 1).

The bit UCSTEM is used to select the corresponding mode.

#### 11.3.3.1 4-Pin SPI Master Mode (UCSTEM = 0)

In 4-pin master mode with UCSTEM = 0, UCxSTE is a digital input that can be used to prevent conflicts with another master and controls the master as described in Table 11-1. When UCxSTE is in the master-inactive state and UCSTEM = 0:

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

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

#### 11.3.3.2 4-Pin SPI Master Mode (UCSTEM = 1)

If UCSTEM = 1 in 4-pin master mode, UCxSTE is a digital output. In this mode the slave enable signal for a single slave is automatically generated on UCxSTE. The corresponding behavior can be seen in Figure 11-4.

If multiple slaves are desired, this feature is not applicable and the software needs to use general purpose I/O pins instead to generate STE signals for each slave individually.

### *11.3.4 Slave Mode*

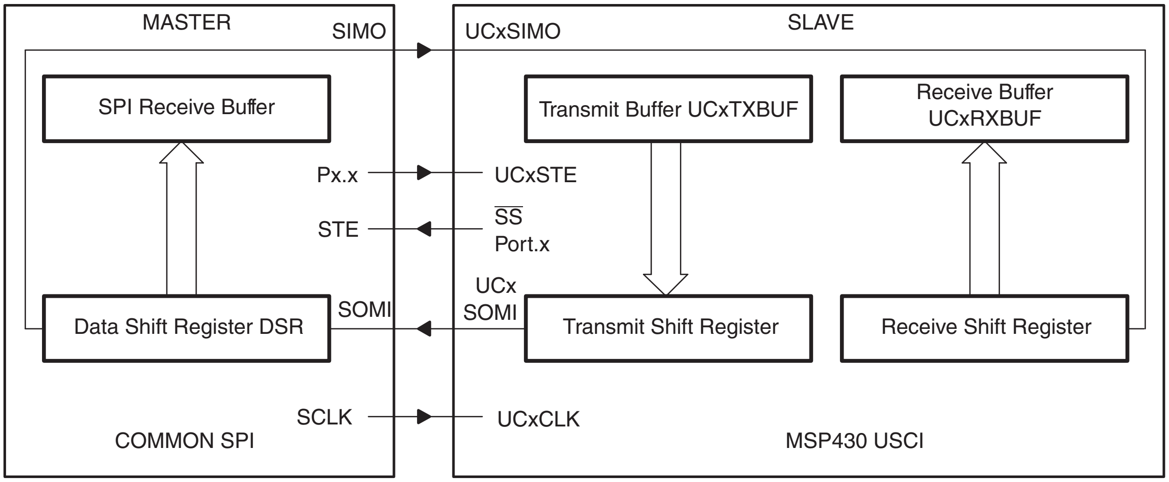


**Figure 11-3. eUSCI Slave and External Master**

Figure 11-3 shows the eUSCI as a slave in both 3-pin and 4-pin configurations. UCxCLK is used as the input for the SPI clock and must be supplied by the external master. The data-transfer rate is determined by this clock and not by the internal bit clock generator. Data written to UCxTXBUF and moved to the TX shift register before the start of UCxCLK is transmitted on UCxSOMI. Data on UCxSIMO is shifted into the receive shift register on the opposite edge of UCxCLK and moved to UCxRXBUF when the set number of bits are received. When data is moved from the RX shift register to UCxRXBUF, the UCRXIFG interrupt flag is set, indicating that data has been received. The overrun error bit UCOE is set when the previously received data is not read from UCxRXBUF before new data is moved to UCxRXBUF.

#### 11.3.4.1 4-Pin SPI Slave Mode

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

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

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

### 11.3.5 SPI Enable

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

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

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

#### 11.3.5.1 Transmit Enable

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

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

#### 11.3.5.2 Receive Enable

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

### 11.3.6 Serial Clock Control

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

The 16-bit value of UCBRx in the bit rate control registers UCxxBRW is the division factor of the eUSCI clock source, BRCLK. With UCBRx = 0 the maximum bit clock that can be generated in master mode is BRCLK. Modulation is not used in SPI mode, and UCAxMCTL should be cleared when using SPI mode for eUSCI_A.

The UCAxCLK or UCBxCLK frequency is given by:

$f_{BitClock} = f_{BRCLK}$ / UCBRx

If UCBRx = 0, $f_{BitClock} = f_{BRCLK}$

Even UCBRx settings result in even divisions and, thus, generate a bit clock with a 50/50 duty cycle.

Odd UCBRx settings result in odd divisions. In this case, the high phase of the bit clock is one BRCLK cycle longer than the low phase.

When UCBRx = 0, no division is applied to BRCLK, and the bit clock equals BRCLK.

#### 11.3.6.1 Serial Clock Polarity and Phase

The polarity and phase of UCxCLK are independently configured via the UCCKPL and UCCKPH control bits of the eUSCI. Timing for each case is shown in Figure 11-4.

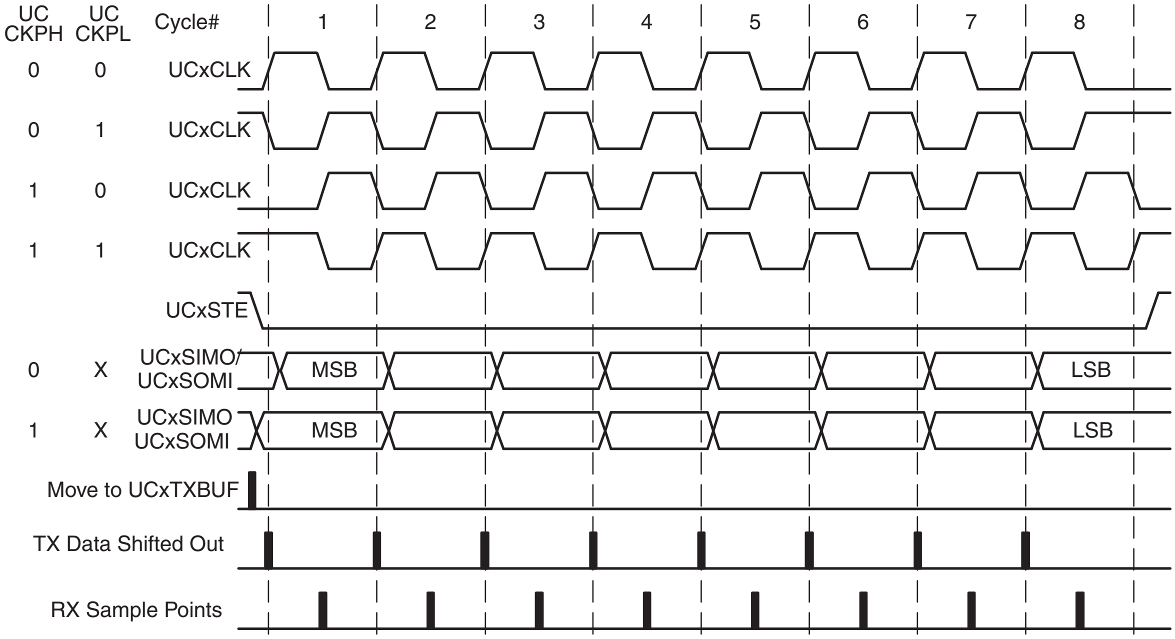


**Figure 11-4. eUSCI SPI Timing With UCMSB = 1**

## 11.3.7 Using the SPI Mode With Low-Power Modes

The eUSCI module provides automatic clock activation for use with low-power modes. When the eUSCI clock source is inactive because the device is in a low-power mode, the eUSCI module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the eUSCI module returns to its idle condition. After the eUSCI module returns to the idle condition, control of the clock source reverts to the settings of its control bits.

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

## 11.3.8 SPI Interrupts

The eUSCI has only one interrupt vector that is shared for transmission and for reception. eUSCI_Ax and eUSCI_Bx do not share the same interrupt vector.

### 11.3.8.1 SPI Transmit Interrupt Operation

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

**NOTE: Writing to UCxTXBUF in SPI mode**

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

### 11.3.8.2 SPI Receive Interrupt Operation

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

### 11.3.8.3 UCxIV, Interrupt Vector Generator

The eUSCI interrupt flags are prioritized and combined to source a single interrupt vector. The interrupt vector register UCxIV is used to determine which flag requested an interrupt. The highest-priority enabled interrupt generates a number in the UCxIV register that can be evaluated or added to the program counter (PC) to automatically enter the appropriate software routine. Disabled interrupts do not affect the UCxIV value.

Any access, read or write, of the UCxIV register automatically resets the highest-pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.

#### *UCxIV Software Example*

The following software example shows the recommended use of UCxIV. The UCxIV value is added to the PC to automatically jump to the appropriate routine. The following example is given for eUSCI_B0.

```
USCI_SPI_ISR
         ADD      &UCB0IV, PC  ; Add offset to jump table
         RETI                  ; Vector 0: No interrupt
         JMP      RXIFG_ISR    ; Vector 2: RXIFG
TXIFG_ISR                      ; Vector 4: TXIFG
         ...                   ; Task starts here
         RETI                  ; Return
RXIFG_ISR                      ; Vector 2
         ...                   ; Task starts here
         RETI                  ; Return
```

## 11.4 eUSCI_A SPI Registers

The eUSCI_A registers applicable in SPI mode and their address offsets are listed in Table 11-2. The base addresses can be found in the device-specific data sheet.

**Table 11-2. eUSCI_A SPI Registers**

| Offset | Acronym | Register Name | Type | Access | Reset | Section |
|---|---|---|---|---|---|---|
| 00h | UCAxCTLW0 | eUSCI_Ax Control Word 0 | Read/write | Word | 0001h | Section 11.4.1 |
| 00h | UCAxCTL1 | eUSCI_Ax Control 1 | Read/write | Byte | 01h | |
| 01h | UCAxCTL0 | eUSCI_Ax Control 0 | Read/write | Byte | 00h | |
| 06h | UCAxBRW | eUSCI_Ax Bit Rate Control Word | Read/write | Word | 0000h | Section 11.4.2 |
| 06h | UCAxBR0 | eUSCI_Ax Bit Rate Control 0 | Read/write | Byte | 00h | |
| 07h | UCAxBR1 | eUSCI_Ax Bit Rate Control 1 | Read/write | Byte | 00h | |
| 0Ah | UCAxSTATW | eUSCI_Ax Status | Read/write | Word | 00h | Section 11.4.3 |
| 0Ch | UCAxRXBUF | eUSCI_Ax Receive Buffer | Read/write | Word | 00h | Section 11.4.4 |
| 0Eh | UCAxTXBUF | eUSCI_Ax Transmit Buffer | Read/write | Word | 00h | Section 11.4.5 |
| 1Ah | UCAxIE | eUSCI_Ax Interrupt Enable | Read/write | Word | 00h | Section 11.4.6 |
| 1Ch | UCAxIFG | eUSCI_Ax Interrupt Flag | Read/write | Word | 02h | Section 11.4.7 |
| 1Eh | UCAxIV | eUSCI_Ax Interrupt Vector | Read | Word | 0000h | Section 11.4.8 |

### 11.4.1 UCAxCTLW0 Register

eUSCI_Ax Control Register 0

**Figure 11-5. UCAxCTLW0 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| UCCKPH | UCCKPL | UCMSB | UC7BIT | UCMST | UCMODEx | | UCSYNC |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCSSELx | | Reserved | | | | UCSTEM | UCSWRST |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-1 |

▨ Modify only when UCSWRST = 1.

**Table 11-3. UCAxCTLW0 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15 | UCCKPH | RW | 0h | Clock phase select<br>0b = Data is changed on the first UCLK edge and captured on the following edge.<br>1b = Data is captured on the first UCLK edge and changed on the following edge. |
| 14 | UCCKPL | RW | 0h | Clock polarity select<br>0b = The inactive state is low.<br>1b = The inactive state is high. |
| 13 | UCMSB | RW | 0h | MSB first select. Controls the direction of the receive and transmit shift register.<br>0b = LSB first<br>1b = MSB first |
| 12 | UC7BIT | RW | 0h | Character length. Selects 7-bit or 8-bit character length.<br>0b = 8-bit data<br>1b = 7-bit data |
| 11 | UCMST | RW | 0h | Master mode select<br>0b = Slave mode<br>1b = Master mode |
| 10-9 | UCMODEx | RW | 0h | eUSCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1.<br>00b = 3-pin SPI<br>01b = 4-pin SPI with UCxSTE active high: Slave enabled when UCxSTE = 1<br>10b = 4-pin SPI with UCxSTE active low: Slave enabled when UCxSTE = 0<br>11b = Reserved |
| 8 | UCSYNC | RW | 0h | Synchronous mode enable<br>0b = Asynchronous mode<br>1b = Synchronous mode |
| 7-6 | UCSSELx | RW | 0h | eUSCI clock source select. These bits select the BRCLK source clock in master mode. UCxCLK is always used in slave mode.<br>00b = Reserved<br>01b = ACLK<br>10b = SMCLK<br>11b = SMCLK |
| 5-2 | Reserved | R | 0h | Reserved |
| 1 | UCSTEM | RW | 0h | STE mode select in master mode. This byte is ignored in slave or 3-wire mode.<br>0b = STE pin is used to prevent conflicts with other masters<br>1b = STE pin is used to generate the enable signal for a 4-wire slave |

**Table 11-3. UCAxCTLW0 Register Description (continued)**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 0 | UCSWRST | RW | 1h | Software reset enable<br>0b = Disabled. eUSCI reset released for operation.<br>1b = Enabled. eUSCI logic held in reset state. |

### 11.4.2 UCAxBRW Register

eUSCI_Ax Bit Rate Control Register 1

**Figure 11-6. UCAxBRW Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| UCBRx | | | | | | | |
| rw | rw | rw | rw | rw | rw | rw | rw |

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

▨ Modify only when UCSWRST = 1.

**Table 11-4. UCAxBRW Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | UCBRx | RW | 0h | Bit clock prescaler setting.<br>$f_{BitClock} = f_{BRCLK}/(UCBRx+1)$ |

### 11.4.3 UCAxSTATW Register

eUSCI_Ax Status Register

**Figure 11-7. UCAxSTATW Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

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

▨ Modify only when UCSWRST = 1.

**Table 11-5. UCAxSTATW Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | Reserved | R | 0h | Reserved |
| 7 | UCLISTEN | RW | 0h | Listen enable. The UCLISTEN bit selects loopback mode.<br>0b = Disabled<br>1b = Enabled. The transmitter output is internally fed back to the receiver. |
| 6 | UCFE | RW | 0h | Framing error flag. This bit indicates a bus conflict in 4-wire master mode. UCFE is not used in 3-wire master or any slave mode.<br>0b = No error<br>1b = Bus conflict occurred |
| 5 | UCOE | RW | 0h | Overrun error flag. This bit is set when a character is transferred into UCxRXBUF before the previous character was read. UCOE is cleared automatically when UCxRXBUF is read, and must not be cleared by software. Otherwise, it does not function correctly.<br>0b = No error<br>1b = Overrun error occurred |
| 4-1 | Reserved | RW | 0h | Reserved |
| 0 | UCBUSY | R | 0h | eUSCI busy. This bit indicates if a transmit or receive operation is in progress.<br>0b = eUSCI inactive<br>1b = eUSCI transmitting or receiving |

### 11.4.4 UCAxRXBUF Register

eUSCI_Ax Receive Buffer Register

**Figure 11-8. UCAxRXBUF Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

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

**Table 11-6. UCAxRXBUF Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | Reserved | R | 0h | Reserved |
| 7-0 | UCRXBUFx | R | 0h | The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCxRXBUF resets the receive-error bits and UCRXIFG. In 7-bit data mode, UCxRXBUF is LSB justified and the MSB is always reset. |

### 11.4.5 UCAxTXBUF Register

eUSCI_Ax Transmit Buffer Register

**Figure 11-9. UCAxTXBUF Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

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

**Table 11-7. UCAxTXBUF Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | Reserved | R | 0h | Reserved |
| 7-0 | UCTXBUFx | RW | 0h | The transmit data buffer is user accessible and holds the data waiting to be moved into the transmit shift register and transmitted. Writing to the transmit data buffer clears UCTXIFG. The MSB of UCxTXBUF is not used for 7-bit data and is reset. |

### 11.4.6 UCAxIE Register

eUSCI_Ax Interrupt Enable Register

**Figure 11-10. UCAxIE Register**

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

**Table 11-8. UCAxIE Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-2 | Reserved | R | 0h | Reserved |
| 1 | UCTXIE | RW | 0h | Transmit interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 0 | UCRXIE | RW | 0h | Receive interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |

### *11.4.7 UCAxIFG Register*

eUSCI_Ax Interrupt Flag Register

**Figure 11-11. UCAxIFG Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | UCTXIFG | UCRXIFG |
| r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | rw-1 | rw-0 |

**Table 11-9. UCAxIFG Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-2 | Reserved | R | 0h | Reserved |
| 1 | UCTXIFG | RW | 1h | Transmit interrupt flag. UCTXIFG is set when UCxxTXBUF empty.<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 0 | UCRXIFG | RW | 0h | Receive interrupt flag. UCRXIFG is set when UCxxRXBUF has received a complete character.<br>0b = No interrupt pending<br>1b = Interrupt pending |

### *11.4.8 UCAxIV Register*

eUSCI_Ax Interrupt Vector Register

**Figure 11-12. UCAxIV Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| UCIVx | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCIVx | | | | | | | |
| r0 | r0 | r0 | r-0 | r-0 | r-0 | r-0 | r0 |

**Table 11-10. UCAxIV Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | UCIVx | R | 0h | eUSCI interrupt vector value<br>000h = No interrupt pending<br>002h = Interrupt Source: Data received; Interrupt Flag: UCRXIFG; Interrupt Priority: Highest<br>004h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG; Interrupt Priority: Lowest |

## 11.5 eUSCI_B SPI Registers

The eUSCI_B registers applicable in SPI mode and their address offsets are listed in Table 11-11. The base addresses can be found in the device-specific data sheet.

**Table 11-11. eUSCI_B SPI Registers**

| Offset | Acronym | Register Name | Type | Access | Reset | Section |
|---|---|---|---|---|---|---|
| 00h | UCBxCTLW0 | eUSCI_Bx Control Word 0 | Read/write | Word | 01C1h | Section 11.5.1 |
| 00h | UCBxCTL1 | eUSCI_Bx Control 1 | Read/write | Byte | C1h | |
| 01h | UCBxCTL0 | eUSCI_Bx Control 0 | Read/write | Byte | 01h | |
| 06h | UCBxBRW | eUSCI_Bx Bit Rate Control Word | Read/write | Word | 0000h | Section 11.5.2 |
| 06h | UCBxBR0 | eUSCI_Bx Bit Rate Control 0 | Read/write | Byte | 00h | |
| 07h | UCBxBR1 | eUSCI_Bx Bit Rate Control 1 | Read/write | Byte | 00h | |
| 08h | UCBxSTATW | eUSCI_Bx Status | Read/write | Word | 00h | Section 11.5.3 |
| 0Ch | UCBxRXBUF | eUSCI_Bx Receive Buffer | Read/write | Word | 00h | Section 11.5.4 |
| 0Eh | UCBxTXBUF | eUSCI_Bx Transmit Buffer | Read/write | Word | 00h | Section 11.5.5 |
| 2Ah | UCBxIE | eUSCI_Bx Interrupt Enable | Read/write | Word | 00h | Section 11.5.6 |
| 2Ch | UCBxIFG | eUSCI_Bx Interrupt Flag | Read/write | Word | 02h | Section 11.5.7 |
| 2Eh | UCBxIV | eUSCI_Bx Interrupt Vector | Read | Word | 0000h | Section 11.5.8 |

### 11.5.1 UCBxCTLW0 Register

eUSCI_Bx Control Register 0

**Figure 11-13. UCBxCTLW0 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| UCCKPH | UCCKPL | UCMSB | UC7BIT | UCMST | UCMODEx | | UCSYNC |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-1 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCSSELx | | Reserved | | | | UCSTEM | UCSWRST |
| rw-1 | rw-1 | r0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-1 |

▨ Modify only when UCSWRST = 1.

**Table 11-12. UCBxCTLW0 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15 | UCCKPH | RW | 0h | Clock phase select<br>0b = Data is changed on the first UCLK edge and captured on the following edge.<br>1b = Data is captured on the first UCLK edge and changed on the following edge. |
| 14 | UCCKPL | RW | 0h | Clock polarity select<br>0b = The inactive state is low.<br>1b = The inactive state is high. |
| 13 | UCMSB | RW | 0h | MSB first select. Controls the direction of the receive and transmit shift register.<br>0b = LSB first<br>1b = MSB first |
| 12 | UC7BIT | RW | 0h | Character length. Selects 7-bit or 8-bit character length.<br>0b = 8-bit data<br>1b = 7-bit data |
| 11 | UCMST | RW | 0h | Master mode select<br>0b = Slave mode<br>1b = Master mode |
| 10-9 | UCMODEx | RW | 0h | eUSCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1.<br>00b = 3-pin SPI<br>01b = 4-pin SPI with UCxSTE active high: Slave enabled when UCxSTE = 1<br>10b = 4-pin SPI with UCxSTE active low: Slave enabled when UCxSTE = 0<br>11b = I2C mode |
| 8 | UCSYNC | RW | 1h | Synchronous mode enable<br>0b = Asynchronous mode<br>1b = Synchronous mode |
| 7-6 | UCSSELx | RW | 3h | eUSCI clock source select. These bits select the BRCLK source clock in master mode. UCxCLK is always used in slave mode.<br>00b = Reserved<br>01b = ACLK<br>10b = SMCLK<br>11b = SMCLK |
| 5-2 | Reserved | R | 0h | Reserved |
| 1 | UCSTEM | RW | 0h | STE mode select in master mode. This byte is ignored in slave or 3-wire mode.<br>0b = STE pin is used to prevent conflicts with other masters<br>1b = STE pin is used to generate the enable signal for a 4-wire slave |

### Table 11-12. UCBxCTLW0 Register Description (continued)

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 0 | UCSWRST | RW | 1h | Software reset enable<br>0b = Disabled. eUSCI reset released for operation.<br>1b = Enabled. eUSCI logic held in reset state. |

### 11.5.2 UCBxBRW Register

eUSCI_Bx Bit Rate Control Register 1

**Figure 11-14. UCBxBRW Register**

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

▨ Modify only when UCSWRST = 1.

**Table 11-13. UCBxBRW Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | UCBRx | RW | 0h | Bit clock prescaler setting.<br>$f_{BitClock} = f_{BRCLK}/(UCBRx+1)$ |

### 11.5.3 UCBxSTATW Register

eUSCI_Bx Status Register

**Figure 11-15. UCBxSTATW Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| UCLISTEN | UCFE | UCOE | Reserved | | | | UCBUSY |
| rw-0 | rw-0 | rw-0 | r0 | r0 | r0 | r0 | r-0 |

▨ Modify only when UCSWRST = 1.

**Table 11-14. UCBxSTATW Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | Reserved | R | 0h | Reserved |
| 7 | UCLISTEN | RW | 0h | Listen enable. The UCLISTEN bit selects loopback mode.<br>0b = Disabled<br>1b = Enabled. The transmitter output is internally fed back to the receiver. |
| 6 | UCFE | RW | 0h | Framing error flag. This bit indicates a bus conflict in 4-wire master mode. UCFE is not used in 3-wire master or any slave mode.<br>0b = No error<br>1b = Bus conflict occurred |
| 5 | UCOE | RW | 0h | Overrun error flag. This bit is set when a character is transferred into UCxRXBUF before the previous character was read. UCOE is cleared automatically when UCxRXBUF is read, and must not be cleared by software. Otherwise, it does not function correctly.<br>0b = No error<br>1b = Overrun error occurred |
| 4-1 | Reserved | R | 0h | Reserved |
| 0 | UCBUSY | R | 0h | eUSCI busy. This bit indicates if a transmit or receive operation is in progress.<br>0b = eUSCI inactive<br>1b = eUSCI transmitting or receiving |

### 11.5.4 UCBxRXBUF Register

eUSCI_Bx Receive Buffer Register

**Figure 11-16. UCBxRXBUF Register**

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

**Table 11-15. UCBxRXBUF Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | Reserved | R | 0h | Reserved |
| 7-0 | UCRXBUFx | R | 0h | The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCxRXBUF resets the receive-error bits and UCRXIFG. In 7-bit data mode, UCxRXBUF is LSB justified and the MSB is always reset. |

### 11.5.5 UCBxTXBUF Register

eUSCI_Bx Transmit Buffer Register

**Figure 11-17. UCBxTXBUF Register**

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

**Table 11-16. UCBxTXBUF Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | Reserved | R | 0h | Reserved |
| 7-0 | UCTXBUFx | RW | 0h | The transmit data buffer is user accessible and holds the data waiting to be moved into the transmit shift register and transmitted. Writing to the transmit data buffer clears UCTXIFG. The MSB of UCxTXBUF is not used for 7-bit data and is reset. |

### 11.5.6 UCBxIE Register

eUSCI_Bx Interrupt Enable Register

**Figure 11-18. UCBxIE Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | UCTXIE | UCRXIE |
| r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | rw-0 | rw-0 |

**Table 11-17. UCBxIE Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-2 | Reserved | R | 0h | Reserved |
| 1 | UCTXIE | RW | 0h | Transmit interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 0 | UCRXIE | RW | 0h | Receive interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |

### 11.5.7 UCBxIFG Register

eUSCI_Bx Interrupt Flag Register

**Figure 11-19. UCBxIFG Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | UCTXIFG | UCRXIFG |
| r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | rw-1 | rw-0 |

**Table 11-18. UCBxIFG Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-2 | Reserved | R | 0h | Reserved |
| 1 | UCTXIFG | RW | 1h | Transmit interrupt flag. UCTXIFG is set when UCxxTXBUF empty.<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 0 | UCRXIFG | RW | 0h | Receive interrupt flag. UCRXIFG is set when UCxxRXBUF has received a complete character.<br>0b = No interrupt pending<br>1b = Interrupt pending |

### 11.5.8 UCBxIV Register

eUSCI_Bx Interrupt Vector Register

**Figure 11-20. UCBxIV Register**

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

**Table 11-19. UCBxIV Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | UCIVx | R | 0h | eUSCI interrupt vector value<br>0000h = No interrupt pending<br>0002h = Interrupt Source: Data received; Interrupt Flag: UCRXIFG; Interrupt Priority: Highest<br>0004h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG; Interrupt Priority: Lowest |

# Enhanced Universal Serial Communication Interface (eUSCI) – $I^2C$ Mode

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

**Topic** **Page**

## 12.1 Enhanced Universal Serial Communication Interface B (eUSCI_B) Overview

The eUSCI_B module supports two serial communication modes:

- $I^2C$ mode
- SPI mode

If more than one eUSCI_B module is implemented on one device, those modules are named with incrementing numbers. For example, if one device has two eUSCI_B modules, they are named eUSCI0_B and eUSCI1_B.

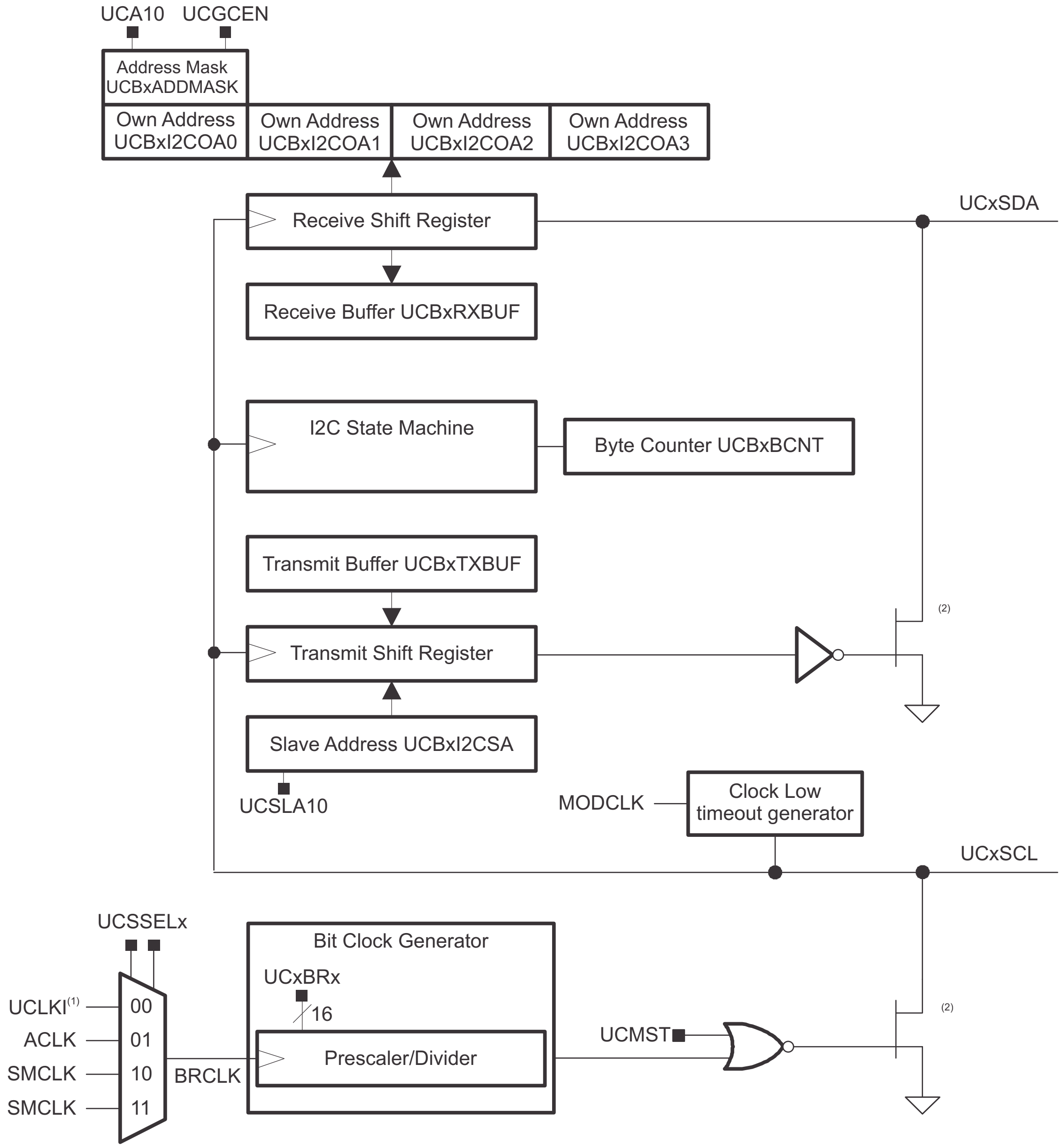
## 12.2 eUSCI_B Introduction – $I^2C$ Mode

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

The eUSCI_B $I^2C$ mode features include:

- 7-bit and 10-bit device addressing modes
- General call
- START, RESTART, STOP
- Multi-master transmitter or receiver mode
- Slave receiver or transmitter mode
- Standard mode up to 100 kbps and fast mode up to 400 kbps support
- Programmable UCxCLK frequency in master mode
- Designed for low power
- 8-bit byte counter with interrupt capability and automatic STOP assertion
- Up to four hardware slave addresses, each having its own interrupt and DMA trigger
- Mask register for slave address and address received interrupt
- Clock low timeout interrupt to avoid bus stalls
- Slave operation in LPM4
- Slave receiver START detection for auto wake-up from LPMx modes (not LPM3.5 and LPM4.5)

Figure 12-1 shows the eUSCI_B when configured in $I^2C$ mode.



[1] Externally provided clock on the eUSCI_B SPI clock input pin
[2] Not the actual implementation (transistor not located in eUSCI_B module)

**Figure 12-1. eUSCI_B Block Diagram – I²C Mode**

## 12.3 eUSCI_B Operation – I²C Mode

The I²C mode supports any slave or master I²C-compatible device. Figure 12-2 shows an example of an I²C bus. Each I²C device is recognized by a unique address and can operate as either a transmitter or a receiver. A device connected to the I²C bus can be considered as the master or the slave when performing data transfers. A master initiates a data transfer and generates the clock signal SCL. Any device addressed by a master is considered a slave.

I²C data is communicated using the serial data (SDA) pin and the serial clock (SCL) pin. Both SDA and SCL are bidirectional and must be connected to a positive supply voltage using a pullup resistor.

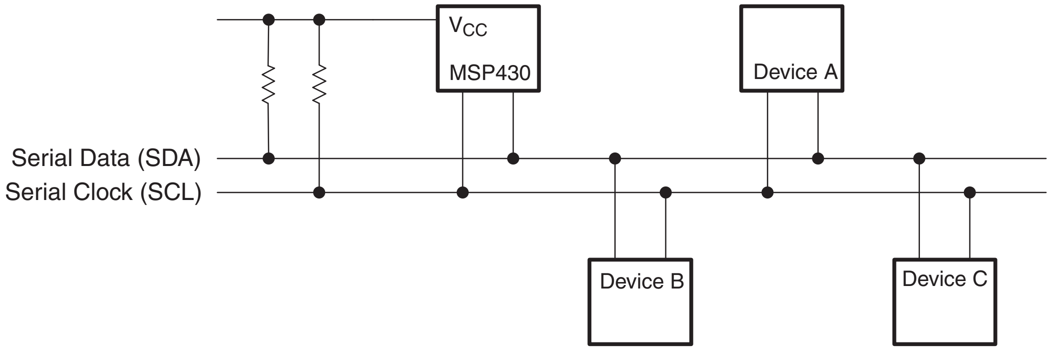


**Figure 12-2. I²C Bus Connection Diagram**

**NOTE: SDA and SCL levels**

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

### 12.3.1 eUSCI_B Initialization and Reset

The eUSCI_B is reset by a PUC or by setting the UCSWRST bit. After a PUC, the UCSWRST bit is automatically set, keeping the eUSCI_B in a reset condition. To select I²C operation, the UCMODEx bits must be set to 11. After module initialization, it is ready for transmit or receive operation. Clearing UCSWRST releases the eUSCI_B for operation.

Configuring and reconfiguring the eUSCI_B module should be done when UCSWRST is set to avoid unpredictable behavior. Setting UCSWRST in I²C mode has the following effects:

- I²C communication stops.
- SDA and SCL are high impedance.
- UCBxSTAT, bits 15-9 and 6-4 are cleared.
- Registers UCBxIE and UCBxIFG are cleared.
- All other bits and registers remain unchanged.

**NOTE: Initializing or re-configuring the eUSCI_B module**

The recommended eUSCI_B initialization/reconfiguration process is:

1. Set UCSWRST (`BIS.B #UCSWRST,&UCxCTL1`).
2. Initialize all eUSCI_B registers with UCSWRST = 1 (including UCxCTL1).
3. Configure ports.
4. Clear UCSWRST via software (`BIC.B #UCSWRST,&UCxCTL1`).
5. Enable interrupts (optional).

### 12.3.2 I²C Serial Data

One clock pulse is generated by the master device for each data bit transferred. The I²C mode operates with byte data. Data is transferred MSB first as shown in Figure 12-3.

The first byte after a START condition consists of a 7-bit slave address and the R/$\overline{W}$ bit. When R/$\overline{W}$ = 0, the master transmits data to a slave. When R/$\overline{W}$ = 1, the master receives data from a slave. The ACK bit is sent from the receiver after each byte on the ninth SCL clock.

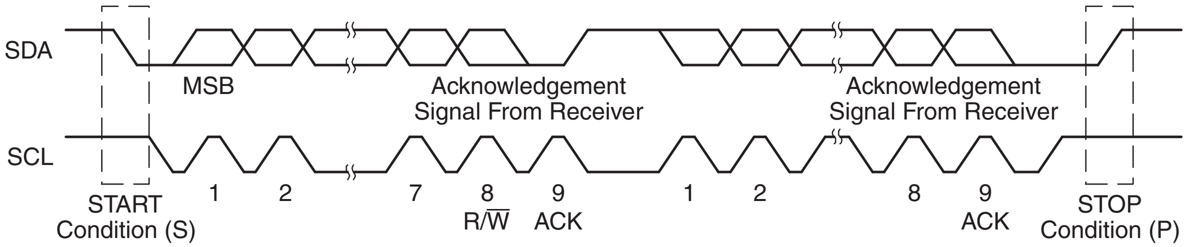


**Figure 12-3. I²C Module Data Transfer**

START and STOP conditions are generated by the master and are shown in Figure 12-3. A START condition is a high-to-low transition on the SDA line while SCL is high. A STOP condition is a low-to-high transition on the SDA line while SCL is high. The bus busy bit, UCBBUSY, is set after a START and cleared after a STOP.

Data on SDA must be stable during the high period of SCL (see Figure 12-4). The high and low state of SDA can change only when SCL is low, otherwise START or STOP conditions are generated.

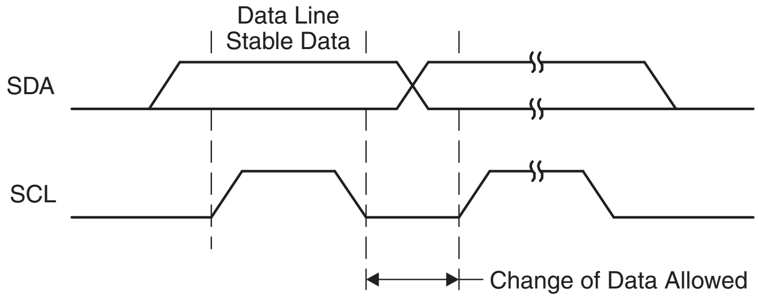


**Figure 12-4. Bit Transfer on I²C Bus**

### 12.3.3 I²C Addressing Modes

The I²C mode supports 7-bit and 10-bit addressing modes.

#### 12.3.3.1 7-Bit Addressing

In the 7-bit addressing format (see Figure 12-5), the first byte is the 7-bit slave address and the R/$\overline{W}$ bit. The ACK bit is sent from the receiver after each byte.

| 1 | 7 | 1 | 1 | 8 | 1 | 8 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| S | Slave Address | R/$\overline{W}$ | ACK | Data | ACK | Data | ACK | P |

**Figure 12-5. I²C Module 7-Bit Addressing Format**

#### 12.3.3.2 10-Bit Addressing

In the 10-bit addressing format (see Figure 12-6), the first byte is made up of 11110b plus the two MSBs of the 10-bit slave address and the R/$\overline{W}$ bit. The ACK bit is sent from the receiver after each byte. The next byte is the remaining eight bits of the 10-bit slave address, followed by the ACK bit and the 8-bit data. See I2C Slave 10-bit Addressing Mode and I2C Master 10-bit Addressing Mode for details how to use the 10-bit addressing mode with the eUSCI_B module.

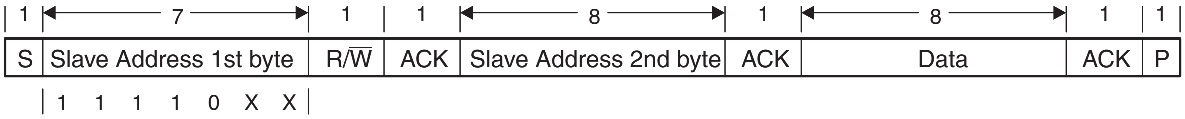


**Figure 12-6. I²C Module 10-Bit Addressing Format**

### 12.3.3.3 Repeated Start Conditions

The direction of data flow on SDA can be changed by the master, without first stopping a transfer, by issuing a repeated START condition. This is called a RESTART. After a RESTART is issued, the slave address is again sent out with the new data direction specified by the R/W̄ bit. The RESTART condition is shown in Figure 12-7.

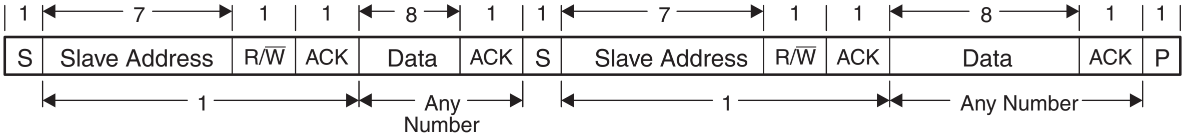


**Figure 12-7. I²C Module Addressing Format With Repeated START Condition**

## *12.3.4 I²C Quick Setup*

This section gives a quick introduction into the operation of the eUSCI_B in I2C mode. The basic steps to start communication are described and shown as a software example. More detailed information about the possible configurations and details can be found in Section 12.3.5.

The latest code examples can be found on the MSP430 web under "Code Examples".

To set up the eUSCI_B as a master transmitter that transmits to a slave with the address 0x12h, only a few steps are needed (see Example 12-1).

***Example 12-1. Master TX With 7-Bit Address***

```
UCBxCTL1 |= UCSWRST;            // put eUSCI_B in reset state
UCBxCTLW0 |= UCMODE_3 + UCMST; // I2C master mode
UCBxBRW = 0x0008;               // baudrate = SMCLK / 8
UCBxCTLW1 = UCASTP_2;           // autom. STOP assertion
UCBxTBCNT = 0x07;               // TX 7 bytes of data
UCBxI2CSA = 0x0012;             // address slave is 12hex
P2SEL |= 0x03;                  // configure I2C pins (device specific)
UCBxCTL1 &= ^UCSWRST;           // eUSCI_B in operational state
UCBxIE |= UCTXIE;               // enable TX-interrupt
GIE;                            // general interrupt enable
...
// inside the eUSCI_B TX interrupt service routine
UCBxTXBUF = 0x77;               // fill TX buffer
```

As shown in the code example, all configurations must be done while UCSWRST is set. To select the I²C operation of the eUSCI_B, UCMODE must be set accordingly. The baudrate of the transmission is set by writing the correct divider in the UCBxBRW register. The default clock selected is SMCLK. How many bytes are transmitted in one frame is controlled by the byte counter threshold register UCBxTBCNT together with the UCASTPx bits.

The slave address to send to is specified in the UCBxI2CSA register. Finally, the ports must be configured. This step is device dependent; see the data sheet for the pins that must be used.

Each byte that is to be transmitted must be written to the UCBxTXBUF inside the interrupt service routine. The recommended structure of the interrupt service routine can be found in Example 12-3.

Example 12-2 shows the steps needed to set up the eUSCI_B as a slave with the address 0x12h that is able to receive and transmit data to the master.

***Example 12-2. Slave RX With 7-Bit Address***

```
UCBxCTL1 |= UCSWRST;        // eUSCI_B in reset state
UCBxCTLW0 |= UCMODE_3;      // I2C slave mode
UCBxI2COA0 = 0x0012;        // own address is 12hex
P2SEL |= 0x03;              // configure I2C pins (device specific)
UCBxCTL1 &= ^UCSWRST;       // eUSCI_B in operational state
UCBxIE |= UCTXIE + UCRXIE;  // enable TX&RX-interrupt
GIE;                        // general interrupt enable
...
// inside the eUSCI_B TX interrupt service routine
UCBxTXBUF = 0x77;           // send 077h
...
// inside the eUSCI_B RX interrupt service routine
data = UCBxRXBUF;           // data is the internal variable
```

As shown in Example 12-2, all configurations must be done while UCSWRST is set. For the slave, $I^2C$ operation is selected by setting UCMODE. The slave address is specified in the UCBxI2COA0 register. To enable the interrupts for receive and transmit requests, the according bits in UCBxIE and, at the end, GIE need to be set. Finally the ports must be configured. This step is device dependent; see the data sheet for the pins that are used.

The RX interrupt service routine is called for every byte received by a master device. The TX interrupt service routine is executed each time the master requests a byte. The recommended structure of the interrupt service routine can be found in Example 12-3.

### 12.3.5 $I^2C$ Module Operating Modes

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

Figure 12-8 shows how to interpret the time-line figures. Data transmitted by the master is represented by grey rectangles; data transmitted by the slave is represented by white rectangles. Data transmitted by the eUSCI_B module, either as master or slave, is shown by rectangles that are taller than the others.

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

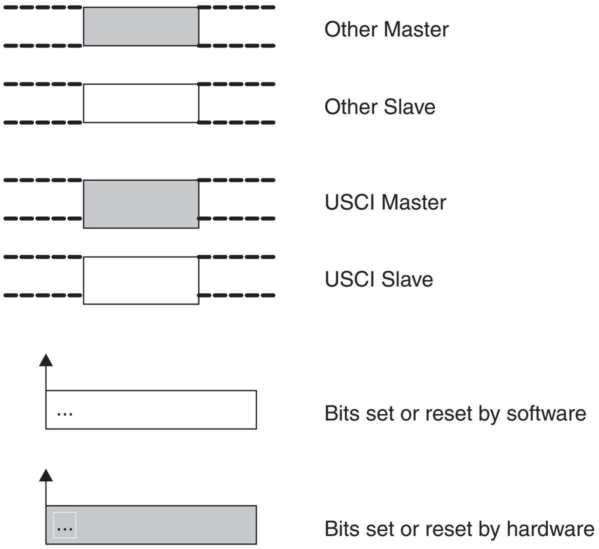


**Figure 12-8. I²C Time-Line Legend**

#### 12.3.5.1 Slave Mode

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

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

The eUSCI_B slave address is programmed with the UCBxI2COA0 register. Support for multiple slave addresses is explained in Section 12.3.9. When UCA10 = 0, 7-bit addressing is selected. When UCA10 = 1, 10-bit addressing is selected. The UCGCEN bit selects if the slave responds to a general call.

When a START condition is detected on the bus, the eUSCI_B module receives the transmitted address and compares it against its own address stored in UCBxI2COA0. The UCSTTIFG flag is set when address received matches the eUSCI_B slave address.

##### *I²C Slave Transmitter Mode*

Slave transmitter mode is entered when the slave address transmitted by the master is identical to its own address with a set R/$\overline{W}$ bit. The slave transmitter shifts the serial data out on SDA with the clock pulses that are generated by the master device. The slave device does not generate the clock, but it does hold SCL low while intervention of the CPU is required after a byte has been transmitted.

If the master requests data from the slave, the eUSCI_B module is automatically configured as a transmitter and UCTR and UCTXIFG0 become set. The SCL line is held low until the first data to be sent is written into the transmit buffer UCBxTXBUF. Then the address is acknowledged and the data is transmitted. As soon as the data is transferred into the shift register, the UCTXIFG0 is set again. After the data is acknowledged by the master, the next data byte written into UCBxTXBUF is transmitted or, if the buffer is empty, the bus is stalled during the acknowledge cycle by holding SCL low until new data is written into UCBxTXBUF. If the master sends a NACK followed by a STOP condition, the UCSTPIFG flag is set. If the NACK is followed by a repeated START condition, the eUSCI_B I²C state machine returns to its address-reception state.

Figure 12-9 shows the slave transmitter operation.

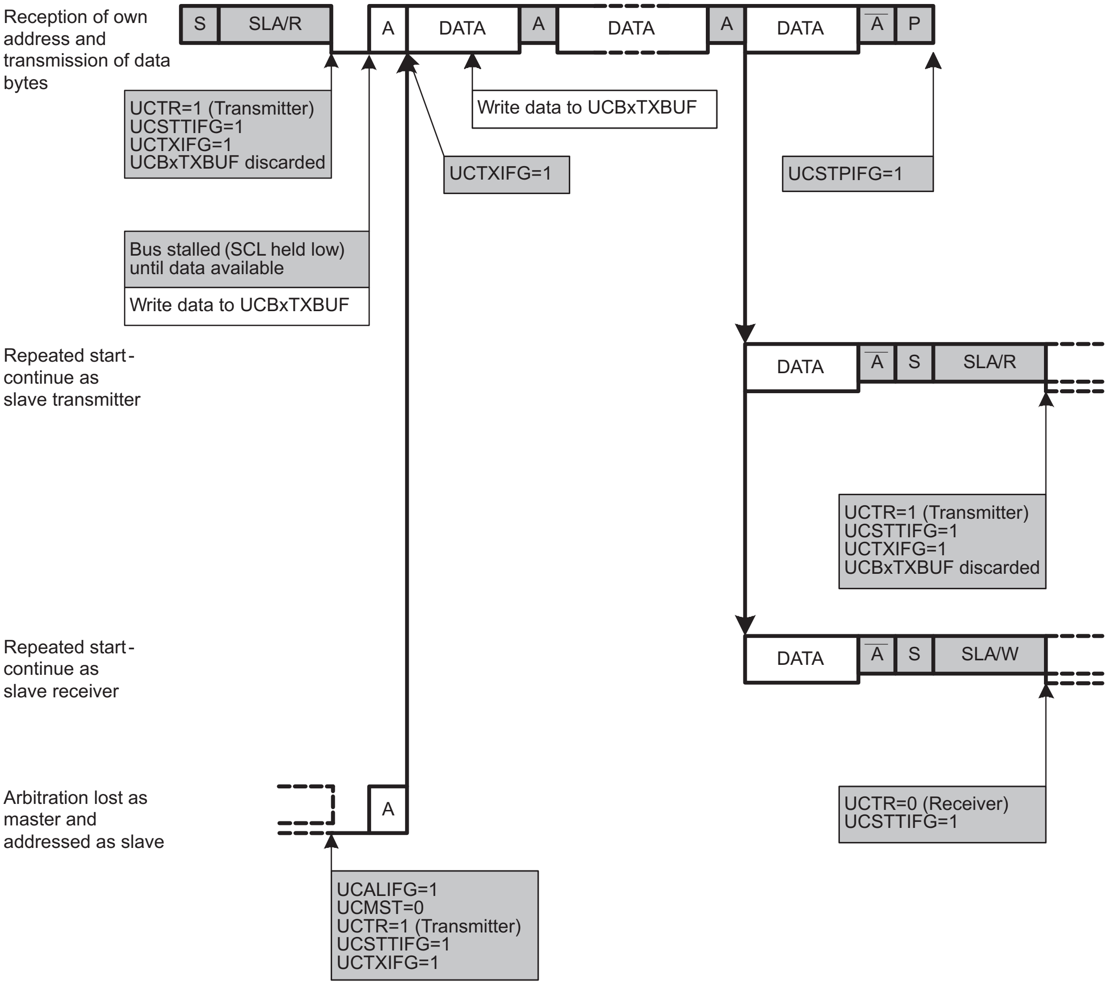


**Figure 12-9. I²C Slave Transmitter Mode**

### *I²C Slave Receiver Mode*

Slave receiver mode is entered when the slave address transmitted by the master is identical to its own address and a cleared R/$\overline{W}$ bit is received. In slave receiver mode, serial data bits received on SDA are shifted in with the clock pulses that are generated by the master device. The slave device does not generate the clock, but it can hold SCL low if intervention of the CPU is required after a byte has been received.

If the slave receives data from the master, the eUSCI_B module is automatically configured as a receiver and UCTR is cleared. After the first data byte is received, the receive interrupt flag UCRXIFG0 is set. The eUSCI_B module automatically acknowledges the received data and can receive the next data byte.

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

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

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

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

Figure 12-10 shows the I$^2$C slave receiver operation.

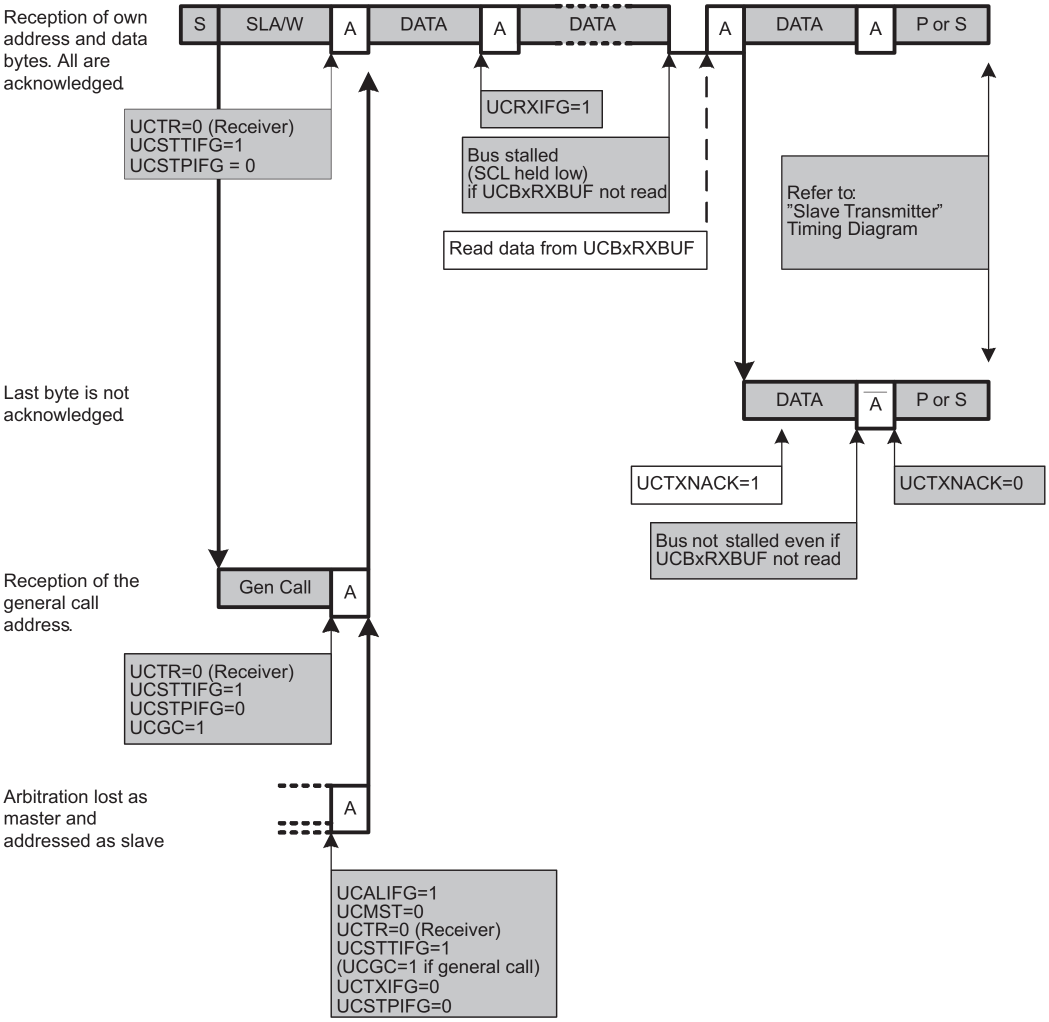


**Figure 12-10. I$^2$C Slave Receiver Mode**

### *I$^2$C Slave 10-Bit Addressing Mode*

The 10-bit addressing mode is selected when UCA10 = 1 and is as shown in Figure 12-11. In 10-bit addressing mode, the slave is in receive mode after the full address is received. The eUSCI_B module indicates this by setting the UCSTTIFG flag while the UCTR bit is cleared. To switch the slave into transmitter mode, the master sends a repeated START condition together with the first byte of the address but with the R/$\overline{W}$ bit set. This sets the UCSTTIFG flag if it was previously cleared by software, and the eUSCI_B modules switches to transmitter mode with UCTR = 1.

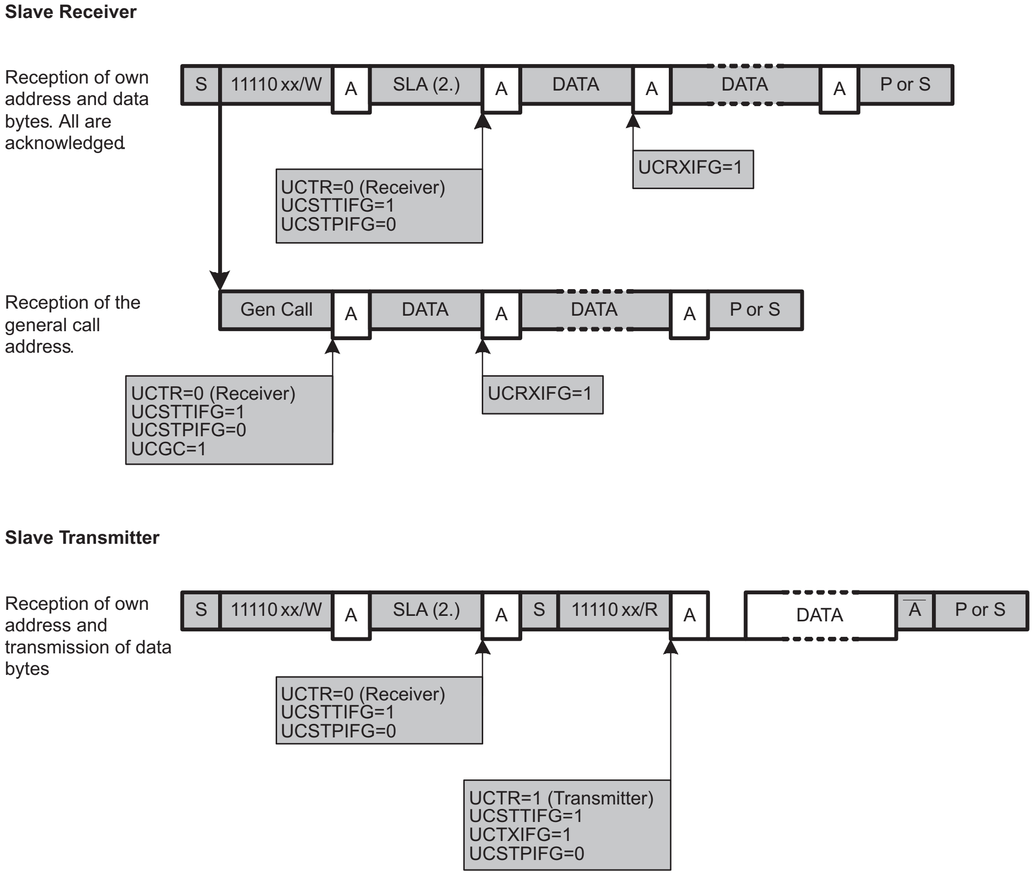


**Figure 12-11. I²C Slave 10-Bit Addressing Mode**

### 12.3.5.2 Master Mode

The eUSCI_B module is configured as an I²C master by selecting the I²C mode with UCMODEx = 11 and UCSYNC = 1 and setting the UCMST bit. When the master is part of a multi-master system, UCMM must be set and its own address must be programmed into the UCBxI2COA0 register. Support for multiple slave addresses is explained in Section 12.3.9. When UCA10 = 0, 7-bit addressing is selected. When UCA10 = 1, 10-bit addressing is selected. The UCGCEN bit selects if the eUSCI_B module responds to a general call.

**NOTE: Addresses and multi-master systems**

In master mode with own-address detection enabled (UCOAEN = 1)—especially in multi-master systems—it is not allowed to specify the same address in the own address and slave address register (UCBxI2CSA = UCBxI2COAx). This would mean that the eUSCI_B addresses itself.

The user software must ensure that this situation does not occur. There is no hardware detection for this case, and the consequence is unpredictable behavior of the eUSCI_B.

### I²C Master Transmitter Mode

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

The eUSCI_B module waits until the bus is available, then generates the START condition, and transmits the slave address. The UCTXIFG0 bit is set when the START condition is generated and the first data to be transmitted can be written into UCBxTXBUF. **The UCTXSTT flag is cleared as soon as the complete address is sent.**

The data written into UCBxTXBUF is transmitted if arbitration is not lost during transmission of the slave address. UCTXIFG0 is set again as soon as the data is transferred from the buffer into the shift register. If there is no data loaded to UCBxTXBUF before the acknowledge cycle, the bus is held during the acknowledge cycle with SCL low until data is written into UCBxTXBUF. Data is transmitted or the bus is held, as long as:

- No automatic STOP is generated
- The UCTXSTP bit is not set
- The UCTXSTT bit is not set

Setting UCTXSTP generates a STOP condition after the next acknowledge from the slave. If UCTXSTP is set during the transmission of the slave address or while the eUSCI_B module waits for data to be written into UCBxTXBUF, a STOP condition is generated, even if no data was transmitted to the slave. **In this case, the UCSTPIFG is set.** When transmitting a single byte of data, the UCTXSTP bit must be set while the byte is being transmitted or any time after transmission begins, without writing new data into UCBxTXBUF. Otherwise, only the address is transmitted. When the data is transferred from the buffer to the shift register, UCTXIFG0 is set, indicating data transmission has begun, and the UCTXSTP bit may be set. When UCASTPx = 10 is set, the byte counter is used for STOP generation and the user does not need to set the UCTXSTP. **This is recommended when transmitting only one byte.**

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

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

Figure 12-12 shows the I²C master transmitter operation.

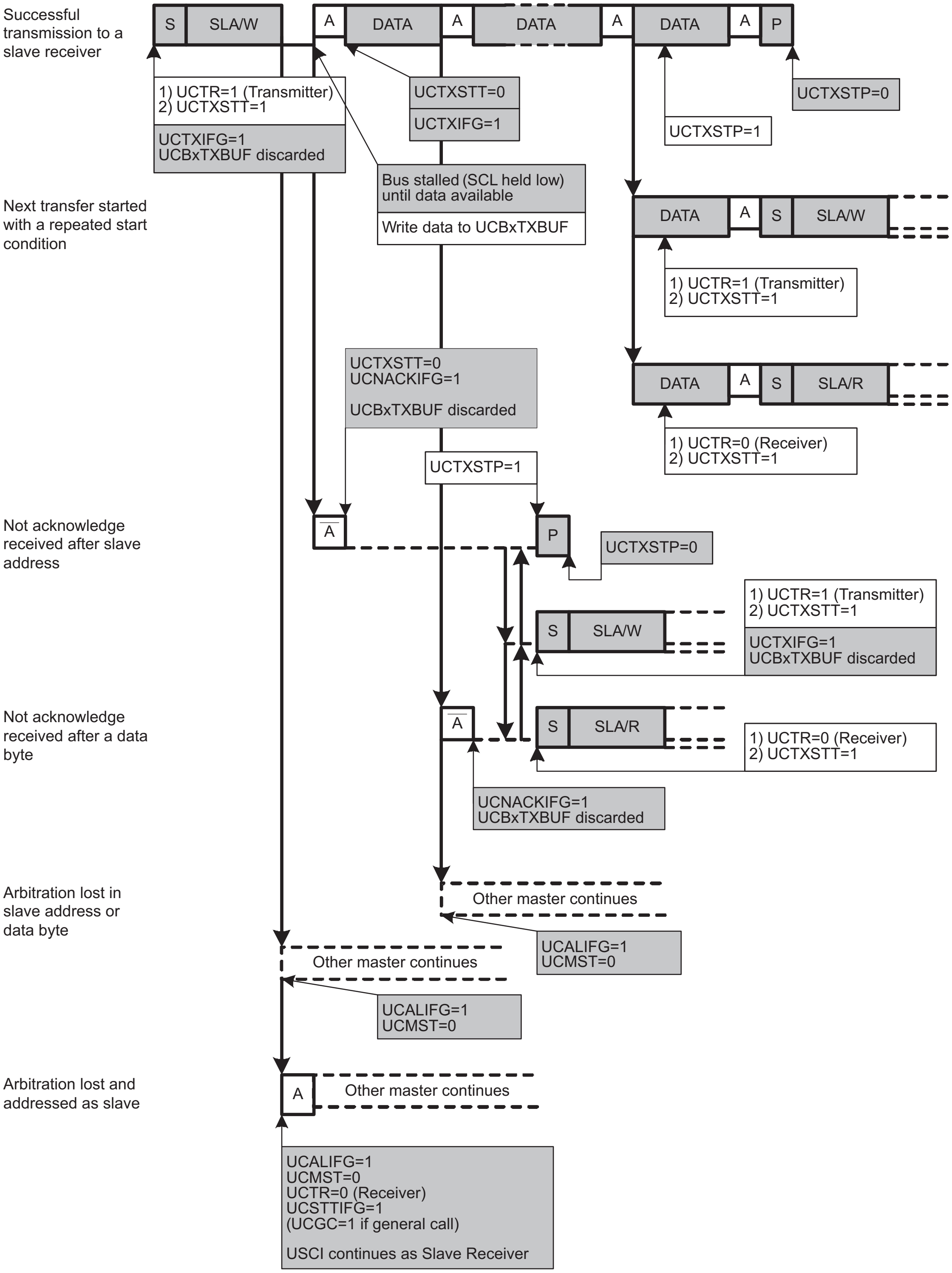


**Figure 12-12. I²C Master Transmitter Mode**

### I$^2$C Master Receiver Mode

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

The eUSCI_B module checks if the bus is available, generates the START condition, and transmits the slave address. The UCTXSTT flag is cleared as soon as the complete address is sent.

After the acknowledge of the address from the slave, the first data byte from the slave is received and acknowledged and the UCRXIFG flag is set. Data is received from the slave, as long as:

- No automatic STOP is generated
- The UCTXSTP bit is not set
- The UCTXSTT bit is not set

If a STOP condition was generated by the eUSCI_B module, the UCSTPIFG is set. If UCBxRXBUF is not read, the master holds the bus during reception of the last data bit and until the UCBxRXBUF is read.

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

A STOP condition is either generated by the automatic STOP generation or by setting the UCTXSTP bit. The next byte received from the slave is followed by a NACK and a STOP condition. This NACK occurs immediately if the eUSCI_B module is currently waiting for UCBxRXBUF to be read.

If a RESTART is sent, UCTR may be set or cleared to configure transmitter or receiver, and a different slave address may be written into UCBxI2CSA if desired.

Figure 12-13 shows the I$^2$C master receiver operation.

**NOTE: Consecutive master transactions without repeated START**

When performing multiple consecutive I$^2$C master transactions without the repeated START feature, the current transaction must be completed before the next one is initiated. This can be done by ensuring that the transmit STOP condition flag UCTXSTP is cleared before the next I$^2$C transaction is initiated with setting UCTXSTT = 1. Otherwise, the current transaction might be affected.

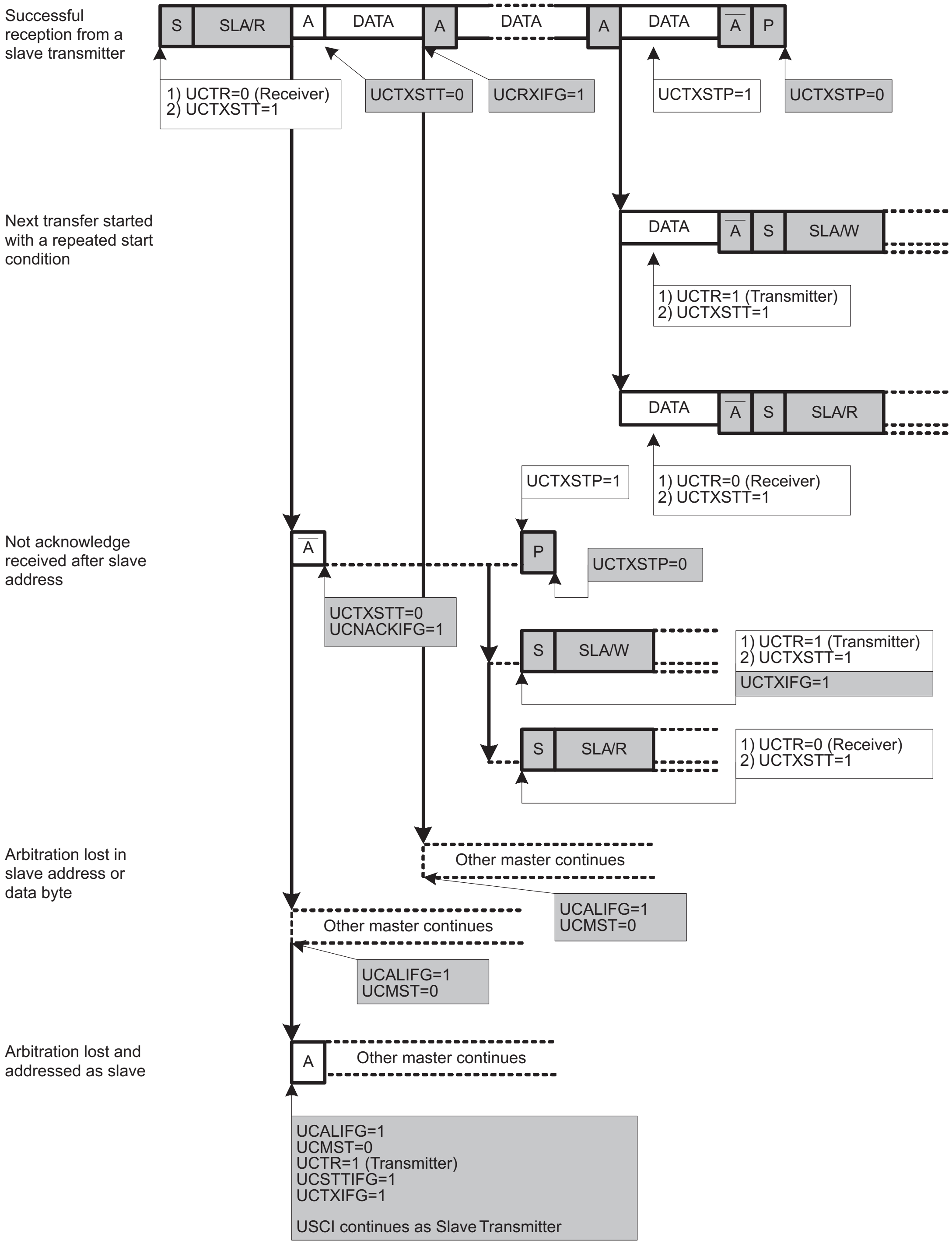


**Figure 12-13. I²C Master Receiver Mode**

### I²C Master 10-Bit Addressing Mode

The 10-bit addressing mode is selected when UCSLA10 = 1 and is shown in Figure 12-14.

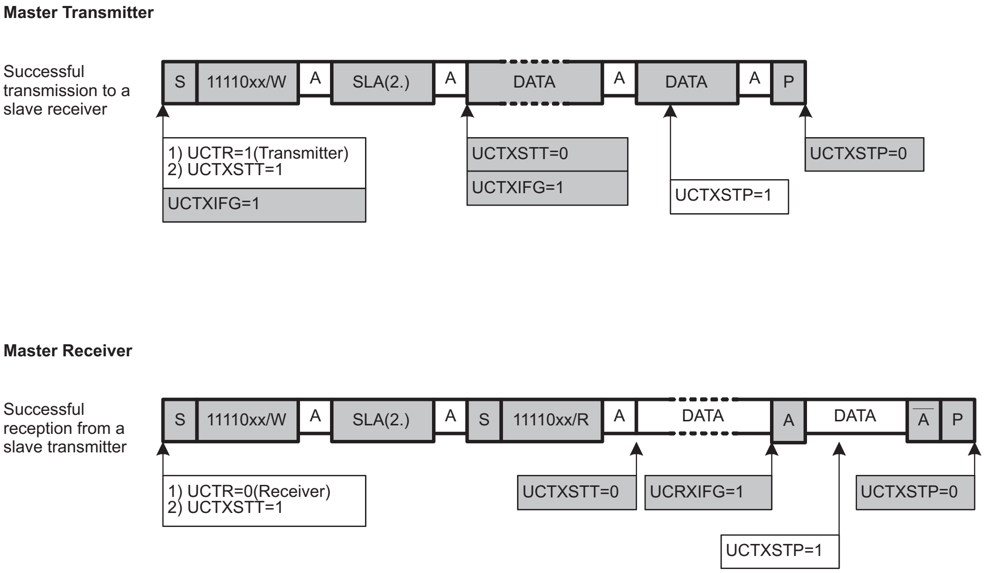


**Figure 12-14. I²C Master 10-Bit Addressing Mode**

#### 12.3.5.3 Arbitration

If two or more master transmitters simultaneously start a transmission on the bus, an arbitration procedure is invoked. Figure 12-15 shows the arbitration procedure between two devices. The arbitration procedure uses the data presented on SDA by the competing transmitters. The first master transmitter that generates a logic high is overruled by the opposing master generating a logic low. The arbitration procedure gives priority to the device that transmits the serial data stream with the lowest binary value. The master transmitter that lost arbitration switches to the slave receiver mode and sets the arbitration lost flag UCALIFG. If two or more devices send identical first bytes, arbitration continues on the subsequent bytes.

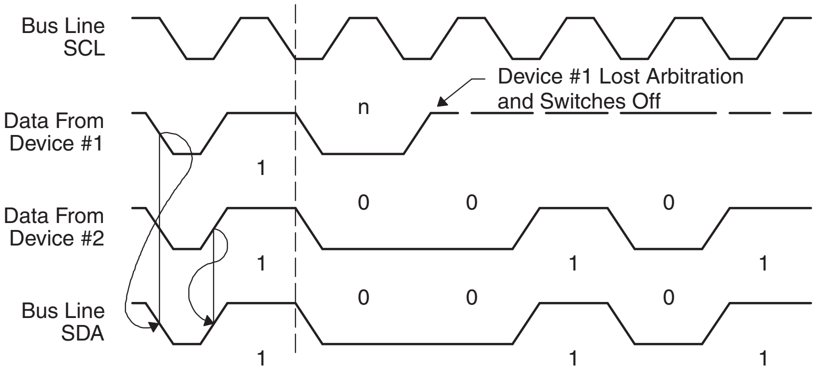


**Figure 12-15. Arbitration Procedure Between Two Master Transmitters**

There is an undefined condition if the arbitration procedure is still in progress when one master sends a repeated START or a STOP condition while the other master is still sending data. In other words, the following combinations result in an undefined condition:

- Master 1 sends a repeated START condition and master 2 sends a data bit.
- Master 1 sends a STOP condition and master 2 sends a data bit.
- Master 1 sends a repeated START condition and master 2 sends a STOP condition.

### 12.3.6 Glitch Filtering

According to the $I^2C$ standard, both the SDA and the SCL line need to be glitch filtered. The eUSCI_B module provides the UCGLITx bits to configure the length of this glitch filter:

**Table 12-1. Glitch Filter Length Selection Bits**

| UCGLITx | Corresponding Glitch Filter Length on SDA and SCL | According to $I^2C$ Standard |
|---|---|---|
| 00 | Pulses of max 50-ns length are filtered | yes |
| 01 | Pulses of max 25-ns length are filtered. | no |
| 10 | Pulses of max 12.5-ns length are filtered. | no |
| 11 | Pulses of max 6.25-ns length are filtered. | no |

### 12.3.7 $I^2C$ Clock Generation and Synchronization

The $I^2C$ clock SCL is provided by the master on the $I^2C$ bus. When the eUSCI_B is in master mode, BITCLK is provided by the eUSCI_B bit clock generator and the clock source is selected with the UCSSELx bits. In slave mode, the bit clock generator is not used and the UCSSELx bits are don't care.

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

$$f_{BitClock} = f_{BRCLK}/UCBRx$$

The minimum high and low periods of the generated SCL are:

$$t_{LOW,MIN} = t_{HIGH,MIN} = (UCBRx/2)/f_{BRCLK} \text{ when UCBRx is even}$$

$$t_{LOW,MIN} = t_{HIGH,MIN} = ((UCBRx - 1)/2)/f_{BRCLK} \text{ when UCBRx is odd}$$

The eUSCI_B clock source frequency and the prescaler setting UCBRx must to be chosen such that the minimum low and high period times of the $I^2C$ specification are met.

During the arbitration procedure the clocks from the different masters must be synchronized. A device that first generates a low period on SCL overrules the other devices, forcing them to start their own low periods. SCL is then held low by the device with the longest low period. The other devices must wait for SCL to be released before starting their high periods. Figure 12-16 shows the clock synchronization. This allows a slow slave to slow down a fast master.

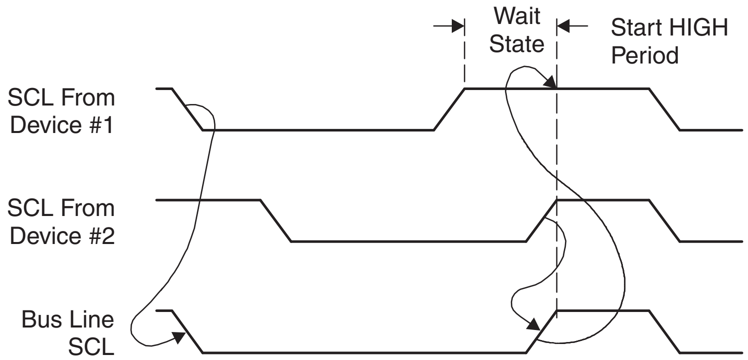


**Figure 12-16. Synchronization of Two $I^2C$ Clock Generators During Arbitration**

#### 12.3.7.1 Clock Stretching

The eUSCI_B module supports clock stretching and also makes use of this feature as described in the Operation Mode sections.

The UCSCLLOW bit can be used to observe if another device pulls SCL low while the eUSCI_B module already released SCL due to the following conditions:

- eUSCI_B is acting as master and a connected slave drives SCL low.

- eUSCI_B is acting as master and another master drives SCL low during arbitration.

The UCSCLLOW bit is also active if the eUSCI_B holds SCL low because it is waiting as transmitter for data being written into UCBxTXBUF or as receiver for the data being read from UCBxRXBUF. The UCSCLLOW bit might be set for a short time with each rising SCL edge because the logic observes the external SCL and compares it to the internally generated SCL.

#### 12.3.7.2 Avoiding Clock Stretching

Even though clock stretching is part of the I2C specification, there are applications in which clock stretching should be avoided.

The clock is stretched by the eUSCI_B under the following conditions:

- The internal shift register is expecting data, but the TXIFG is still pending
- The internal shift register is full, but the RXIFG is still pending
- The arbitration lost interrupt is pending
- UCSWACK is selected and UCBxI2COA0 did cause a match

To avoid clock stretching, all of these situations for clock stretch either need to be avoided or the corresponding interrupt flags need to be processed before the actual clock stretch can occur.

Using the DMA (on devices that contain a DMA) is the most secure way to avoid clock stretching. If no DMA is available, the software must ensure that the corresponding interrupts are serviced in time before the clock is stretched.

In slave transmitter mode, the TXIFG is set only after the reception of the direction bit; therefore, there is only a short amount of time for the software to write the TXBUF before a clock stretch occurs. This situation can be remedied by using the early Transmit Interrupt (see Section 12.3.11.2).

#### 12.3.7.3 Clock Low Timeout

The UCCLTOIFG interrupt allows the software to react if the clock is low longer than a defined time. It is possible to detect the situation, when a clock is stretched by a master or slave for a too long time. The user can then, for example, reset the eUSCI_B module by using the UCSWRST bit.

The clock low timeout feature is enabled using the UCCLTO bits. It is possible to select one of three predefined times for the clock low timeout. If the clock has been low longer than the time defined with the UCCLTO bits and the eUSCI_B was actively receiving or transmitting, the UCCLTOIFG is set and an interrupt request is generated if UCCLTOIE and GIE are set as well. The UCCLTOIFG is set only once, even if the clock is stretched a multiple of the time defined in UCCLTO.

### *12.3.8 Byte Counter*

The eUSCI_B module supports hardware counting of the bytes received or transmitted. The counter is automatically active and counts up for each byte seen on the bus in both master and slave mode.

The byte counter is incremented at the second bit position of each byte independently of the following ACK or NACK. A START or RESTART condition resets the counter value to zero. Address bytes do not increment the counter. The byte counter is also incremented at the second bit position, if an arbitration lost occurs during the first bit of data.

#### 12.3.8.1 Byte Counter Interrupt

If UCASTPx = 01 or 10 the UCBCNTIFG is set when the byte counter threshold value UCBxTBCNT is reached in both master- and slave-mode. Writing zero to UCBxTBCNT does not generate an interrupt.

Because the UCBCNTIFG has a lower interrupt priority than the UCBTXIFG and UCBRXIFG, it is recommended to only use it for protocol control together with the DMA handling the received and transmitted bytes. Otherwise the application must have enough processor bandwidth to ensure that the UCBCNT interrupt routine is executed in time to generate for example a RESTART.

#### 12.3.8.2 Automatic STOP Generation

When the eUSCI_B module is configured as a master, the byte counter can be used for automatic STOP generation by setting the UCASTPx = 10. Before starting the transmission using UCTXSTT, the byte counter threshold UCBxTBCNT must be set to the number of bytes that are to be transmitted or received. After the number of bytes that are configured in UCBxTBCNT have been transmitted, the eUSCI_B automatically generates a STOP condition.

UCBxTBCNT cannot be used if the user wants to transmit the slave address only without any data. In this case, it is recommended to set UCTXSTT and UCTXSTP at the same time.

### *12.3.9 Multiple Slave Addresses*

The eUSCI_B module supports two different ways of implementing multiple slave addresses at the same time:

- Hardware support for up to 4 different slave addresses, each with its own interrupt flag and DMA trigger
- Software support for up to $2^{10}$ different slave addresses all sharing one interrupt

#### 12.3.9.1 Multiple Slave Address Registers

The registers UCBxI2COA0, UCBxI2COA1, UCBxI2COA2, and UCBxI2COA3 contain four slave addresses. Up to four address registers are compared against a received 7- or 10-bit address. Each slave address must be activated by setting the UCAOEN bit in the corresponding UCBxI2COAx register. Register UCBxI2COA3 has the highest priority if the address received on the bus matches more than one of the slave address registers. The priority decreases with the index number of the address register, so that UCBxI2COA0 in combination with the address mask has the lowest priority.

When one of the slave registers matches the 7- or 10-bit address seen on the bus, the address is acknowledged. In the following the corresponding receive- or transmit-interrupt flag (UCTXIFGx or UCRXIFGx) to the received address is updated. The state change interrupt flags are independent of the address comparison result. They are updated according to the bus condition.

#### 12.3.9.2 Address Mask Register

The address mask register can be used when the eUSCI_B is configured in slave or in multiple-master mode. To activate this feature, at least one bit of the address mask in register UCBxADDMASK must be cleared.

If the received address matches the own address in UCBxI2COA0 on all bit positions that are not masked by UCBxADDMASK, the eUSCI_B module considers the received address as its own address. If UCSWACK = 0, the module sends an acknowledge automatically. If UCSWACK = 1, the user software must evaluate the received address in register UCBxADDRX after the UCSTTIFG is set. To acknowledge the received address, the software must set UCTXACK to 1.

The eUSCI_B module also automatically acknowledges a slave address that is seen on the bus if the address matches any of the enabled slave addresses defined in UCBxI2COA1 to UCBxI2COA3.

**NOTE: UCSWACK and slave-transmitter**

If the user selects manual acknowledge of slave addresses, TXIFG is set if the slave is addressed as a transmitter. If the software decides not to acknowledge the address, TXIFG0 must be reset.

### *12.3.10 Using the eUSCI_B Module in I²C Mode With Low-Power Modes*

The eUSCI_B module provides automatic clock activation for use with low-power modes. When the eUSCI_B clock source is inactive because the device is in a low-power mode, the eUSCI_B module automatically activates it when needed, regardless of the control-bit settings for the clock source. The clock remains active until the eUSCI_B module returns to its idle condition. After the eUSCI_B module returns to the idle condition, control of the clock source reverts to the settings of its control bits.

In I$^2$C slave mode, no internal clock source is required because the clock is provided by the external master. It is possible to operate the eUSCI_B in I$^2$C slave mode while the device is in LPM4 and all internal clock sources are disabled. The receive or transmit interrupts can wake up the CPU from any low-power mode.

## 12.3.11 eUSCI_B Interrupts in I$^2$C Mode

The eUSCI_B has only one interrupt vector that is shared for transmission, reception, and the state change.

Each interrupt flag has its own interrupt enable bit. When an interrupt is enabled and the GIE bit is set, the interrupt flag generates an interrupt request. DMA transfers are controlled by the UCTXIFGx and UCRXIFGx flags on devices with a DMA controller. It is possible to react on each slave address with an individual DMA channel.

All interrupt flags are not cleared automatically, but they need to be cleared together by user interactions (for example, reading the UCRXBUF clears UCRXIFGx). If the user wants to use an interrupt flag he needs to ensure that the flag has the correct state before the corresponding interrupt is enabled.

### 12.3.11.1 I$^2$C Transmit Interrupt Operation

The UCTXIFG0 interrupt flag is set whenever the transmitter is able to accept a new byte. When operating as a slave with multiple slave addresses, the UCTXIFGx flags are set corresponding to which address was received before. If, for example, the slave address specified in register UCBxI2COA3 did match the address seen on the bus, the UCTXIFG3 indicates that the UCBxTXBUF is ready to accept a new byte.

When operating in master mode with automatic STOP generation (UCASTPx = 10), the UCTXIFG0 is set as many times as defined in UCBxTBCNT.

An interrupt request is generated if UCTXIEx and GIE are also set. UCTXIFGx is automatically reset if a write to UCBxTXBUF occurs or if the UCALIFG is cleared. UCTXIFGx is set when:

- Master mode: UCTXSTT was set by the user
- Slave mode: own address was received(UCETXINT = 0) or START was received (UCETXINT = 1)

UCTXIEx is reset after a PUC or when UCSWRST = 1.

### 12.3.11.2 Early I$^2$C Transmit Interrupt

Setting the UCETXINT causes UCTXIFG0 to be sent out automatically when a START condition is sent and the eUSCI_B is configured as slave. In this case, it is not allowed to enable the other slave addresses UCBxI2COA1-UCBxI2COA3. This allows the software more time to handle the UCTXIFG0 compared to the normal situation, when UCTXIFG0 is sent out after the slave address match was detected. Situations where the UCTXIFG0 was set and afterward no slave address match occurred need to be handled in software. The use of the byte counter is recommended to handle this.

### 12.3.11.3 I$^2$C Receive Interrupt Operation

The UCRXIFG0 interrupt flag is set when a character is received and loaded into UCBxRXBUF. When operating as a slave with multiple slave addresses, the UCRXIFGx flag is set corresponding to which address was received before.

An interrupt request is generated if UCRXIEx and GIE are also set. UCRXIFGx and UCRXIEx are reset after a PUC signal or when UCSWRST = 1. UCRXIFGx is automatically reset when UCxRXBUF is read.

### 12.3.11.4 I$^2$C State Change Interrupt Operation

Table 12-2 describes the I$^2$C state change interrupt flags.

### Table 12-2. I²C State Change Interrupt Flags

| Interrupt Flag | Interrupt Condition |
|---|---|
| UCALIFG | Arbitration lost interrupt. Arbitration can be lost when two or more transmitters start a transmission simultaneously, or when the eUSCI_B operates as master but is addressed as a slave by another master in the system. The UCALIFG flag is set when arbitration is lost. When UCALIFG is set, the UCMST bit is cleared and the I²C controller becomes a slave. |
| UCNACKIFG | Not acknowledge interrupt. This flag is set when an acknowledge is expected but is not received. UCNACKIFG is used in master mode only. |
| UCCLTOIFG | Clock low timeout. This interrupt flag is set, if the clock is held low longer than defined by the UCCLTO bits. |
| UCBIT9IFG | This interrupt flag is generated each time the eUSCI_B is transferring the nineth clock cycle of a byte of data. This gives the user the ability to follow the I²C communication in software if wanted. UCBIT9IFG is not set for address information. |
| UCBCNTIFG | Byte counter interrupt. This flag is set when the byte counter value reaches the value defined in UCBxTBCNT and UCASTPx = 01 or 10. This bit allows to organize following communications, especially if a RESTART will be issued. |
| UCSTTIFG | START condition detected interrupt. This flag is set when the I²C module detects a START condition together with its own address[1]. UCSTTIFG is used in slave mode only. |
| UCSTPIFG | STOP condition detected interrupt. This flag is set when the I²C module detects a STOP condition on the bus. UCSTPIFG is used in slave and master mode. |

[1] The address evaluation includes the address mask register if it is used.

#### 12.3.11.5 UCBxIV, Interrupt Vector Generator

The eUSCI_B interrupt flags are prioritized and combined to source a single interrupt vector. The interrupt vector register UCBxIV is used to determine which flag requested an interrupt. The highest-priority enabled interrupt generates a number in the UCBxIV register that can be evaluated or added to the PC to automatically enter the appropriate software routine. Disabled interrupts do not affect the UCBxIV value.

Read access of the UCBxIV register automatically resets the highest-pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.

Write access of the UCBxIV register clears all pending Interrupt conditions and flags.

Example 12-3 shows the recommended use of UCBxIV. The UCBxIV value is added to the PC to automatically jump to the appropriate routine. The example is given for eUSCI0_B.

### Example 12-3. UCBxIV Software Example

```
#pragma vector = USCI_B0_VECTOR __interrupt void USCI_B0_ISR(void) {
    switch(__even_in_range(UCB0IV,0x1e))    {
        case 0x00:     // Vector 0: No interrupts
                   break;
        case 0x02: ... // Vector 2: ALIFG
                   break;
        case 0x04: ... // Vector 4: NACKIFG
                   break;
        case 0x06: ... // Vector 6: STTIFG
                   break;
        case 0x08: ... // Vector 8: STPIFG
                   break;
        case 0x0a: ... // Vector 10: RXIFG3
                   break;
        case 0x0c: ... // Vector 12: TXIFG3
                   break;
        case 0x0e: ... // Vector 14: RXIFG2
                   break;
        case 0x10: ... // Vector 16: TXIFG2
                   break;
        case 0x12: ... // Vector 18: RXIFG1
                   break;
        case 0x14: ... // Vector 20: TXIFG1
                   break;
        case 0x16: ... // Vector 22: RXIFG0
                   break;
        case 0x18: ... // Vector 24: TXIFG0
                   break;
        case 0x1a: ... // Vector 26: BCNTIFG
                   break;
        case 0x1c: ... // Vector 28: clock low timeout
                   break;
        case 0x1e: ... // Vector 30: 9th bit
                   break;
        default:   break;
    }
}
```

## 12.4 eUSCI_B I2C Registers

The eUSCI_B registers applicable in I2C mode and their address offsets are listed in Table 12-3. The base address can be found in the device-specific data sheet.

**Table 12-3. eUSCI_B Registers**

| Offset | Acronym | Register Name | Type | Access | Reset | Section |
|---|---|---|---|---|---|---|
| 00h | UCBxCTLW0 | eUSCI_Bx Control Word 0 | Read/write | Word | 01C1h | Section 12.4.1 |
| 00h | UCBxCTL1 | eUSCI_Bx Control 1 | Read/write | Byte | C1h | |
| 01h | UCBxCTL0 | eUSCI_Bx Control 0 | Read/write | Byte | 01h | |
| 02h | UCBxCTLW1 | eUSCI_Bx Control Word 1 | Read/write | Word | 0000h | Section 12.4.2 |
| 06h | UCBxBRW | eUSCI_Bx Bit Rate Control Word | Read/write | Word | 0000h | Section 12.4.3 |
| 06h | UCBxBR0 | eUSCI_Bx Bit Rate Control 0 | Read/write | Byte | 00h | |
| 07h | UCBxBR1 | eUSCI_Bx Bit Rate Control 1 | Read/write | Byte | 00h | |
| 08h | UCBxSTATW | eUSCI_Bx Status Word | Read | Word | 0000h | Section 12.4.4 |
| 08h | UCBxSTAT | eUSCI_Bx Status | Read | Byte | 00h | |
| 09h | UCBxBCNT | eUSCI_Bx Byte Counter Register | Read | Byte | 00h | |
| 0Ah | UCBxTBCNT | eUSCI_Bx Byte Counter Threshold Register | Read/Write | Word | 00h | Section 12.4.5 |
| 0Ch | UCBxRXBUF | eUSCI_Bx Receive Buffer | Read/write | Word | 00h | Section 12.4.6 |
| 0Eh | UCBxTXBUF | eUSCI_Bx Transmit Buffer | Read/write | Word | 00h | Section 12.4.7 |
| 14h | UCBxI2COA0 | eUSCI_Bx I2C Own Address 0 | Read/write | Word | 0000h | Section 12.4.8 |
| 16h | UCBxI2COA1 | eUSCI_Bx I2C Own Address 1 | Read/write | Word | 0000h | Section 12.4.9 |
| 18h | UCBxI2COA2 | eUSCI_Bx I2C Own Address 2 | Read/write | Word | 0000h | Section 12.4.10 |
| 1Ah | UCBxI2COA3 | eUSCI_Bx I2C Own Address 3 | Read/write | Word | 0000h | Section 12.4.11 |
| 1Ch | UCBxADDRX | eUSCI_Bx Received Address Register | Read | Word | | Section 12.4.12 |
| 1Eh | UCBxADDMASK | eUSCI_Bx Address Mask Register | Read/write | Word | 03FFh | Section 12.4.13 |
| 20h | UCBxI2CSA | eUSCI_Bx I2C Slave Address | Read/write | Word | 0000h | Section 12.4.14 |
| 2Ah | UCBxIE | eUSCI_Bx Interrupt Enable | Read/write | Word | 0000h | Section 12.4.15 |
| 2Ch | UCBxIFG | eUSCI_Bx Interrupt Flag | Read/write | Word | 0002h | Section 12.4.16 |
| 2Eh | UCBxIV | eUSCI_Bx Interrupt Vector | Read | Word | 0000h | Section 12.4.17 |

### 12.4.1 UCBxCTLW0 Register

eUSCI_Bx Control Word Register 0

**Figure 12-17. UCBxCTLW0 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| UCA10 | UCSLA10 | UCMM | Reserved | UCMST | UCMODEx | | UCSYNC |
| rw-0 | rw-0 | rw-0 | r0 | rw-0 | rw-0 | rw-0 | r1 |

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

▨ Modify only when UCSWRST = 1.

**Table 12-4. UCBxCTLW0 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15 | UCA10 | RW | 0h | Own addressing mode select.<br>Modify only when UCSWRST = 1.<br>0b = Own address is a 7-bit address.<br>1b = Own address is a 10-bit address. |
| 14 | UCSLA10 | RW | 0h | Slave addressing mode select<br>0b = Address slave with 7-bit address<br>1b = Address slave with 10-bit address |
| 13 | UCMM | RW | 0h | Multi-master environment select.<br>Modify only when UCSWRST = 1.<br>0b = Single master environment. There is no other master in the system. The address compare unit is disabled.<br>1b = Multi-master environment |
| 12 | Reserved | R | 0h | Reserved |
| 11 | UCMST | RW | 0h | Master mode select. When a master loses arbitration in a multi-master environment (UCMM = 1), the UCMST bit is automatically cleared and the module acts as slave.<br>0b = Slave mode<br>1b = Master mode |
| 10-9 | UCMODEx | RW | 0h | eUSCI_B mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1.<br>Modify only when UCSWRST = 1.<br>00b = 3-pin SPI<br>01b = 4-pin SPI (master or slave enabled if STE = 1)<br>10b = 4-pin SPI (master or slave enabled if STE = 0)<br>11b = I2C mode |
| 8 | UCSYNC | RW | 1h | Synchronous mode enable. For eUSCI_B always read and write as 1. |
| 7-6 | UCSSELx | RW | 3h | eUSCI_B clock source select. These bits select the BRCLK source clock. These bits are ignored in slave mode.<br>Modify only when UCSWRST = 1.<br>00b = UCLKI<br>01b = ACLK<br>10b = SMCLK<br>11b = SMCLK |
| 5 | UCTXACK | RW | 0h | Transmit ACK condition in slave mode with enabled address mask register. After the UCSTTIFG has been set, the user needs to set or reset the UCTXACK flag to continue with the I2C protocol. The clock is stretched until the UCBxCTL1 register has been written. This bit is cleared automatically after the ACK has been send.<br>0b = Do not acknowledge the slave address<br>1b = Acknowledge the slave address |

### Table 12-4. UCBxCTLW0 Register Description (continued)

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 4 | UCTR | RW | 0h | Transmitter/receiver<br>0b = Receiver<br>1b = Transmitter |
| 3 | UCTXNACK | RW | 0h | Transmit a NACK. UCTXNACK is automatically cleared after a NACK is transmitted. Only for slave receiver mode.<br>0b = Acknowledge normally<br>1b = Generate NACK |
| 2 | UCTXSTP | RW | 0h | Transmit STOP condition in master mode. Ignored in slave mode. In master receiver mode, the STOP condition is preceded by a NACK. UCTXSTP is automatically cleared after STOP is generated. This bit is a don't care, if automatic UCASTPx is different from 01 or 10.<br>0b = No STOP generated<br>1b = Generate STOP |
| 1 | UCTXSTT | RW | 0h | Transmit START condition in master mode. Ignored in slave mode. In master receiver mode, a repeated START condition is preceded by a NACK. UCTXSTT is automatically cleared after START condition and address information is transmitted. Ignored in slave mode.<br>0b = Do not generate START condition<br>1b = Generate START condition |
| 0 | UCSWRST | RW | 1h | Software reset enable.<br>Modify only when UCSWRST = 1.<br>0b = Disabled. eUSCI_B released for operation.<br>1b = Enabled. eUSCI_B logic held in reset state. |

### 12.4.2 UCBxCTLW1 Register

eUSCI_Bx Control Word Register 1

**Figure 12-18. UCBxCTLW1 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | UCETXINT |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | rw-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCCLTO | | UCSTPNACK | UCSWACK | UCASTPx | | UCGLITx | |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |

▨ Modify only when UCSWRST = 1.

**Table 12-5. UCBxCTLW1 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-9 | Reserved | R | 0h | Reserved |
| 8 | UCETXINT | RW | 0h | Early UCTXIFG0. Only in slave mode. When this bit is set, the slave addresses defined in UCxI2COA1 to UCxI2COA3 must be disabled.<br>Modify only when UCSWRST = 1.<br>0b = UCTXIFGx is set after an address match with UCxI2COAx and the direction bit indicating slave transmit<br>1b = UCTXIFG0 is set for each START condition |
| 7-6 | UCCLTO | RW | 0h | Clock low timeout select.<br>Modify only when UCSWRST = 1.<br>00b = Disable clock low timeout counter<br>01b = 135 000 MODCLK cycles (approximately 28 ms)<br>10b = 150 000 MODCLK cycles (approximately 31 ms)<br>11b = 165 000 MODCLK cycles (approximately 34 ms) |
| 5 | UCSTPNACK | RW | 0h | The UCSTPNACK bit allows to make the eUSCI_B master acknowledge the last byte in master receiver mode as well. This is not conform to the I2C specification and should only be used for slaves, which automatically release the SDA after a fixed packet length.<br>Modify only when UCSWRST = 1.<br>0b = Send a non-acknowledge before the STOP condition as a master receiver (conform to I2C standard)<br>1b = All bytes are acknowledged by the eUSCI_B when configured as master receiver |
| 4 | UCSWACK | RW | 0h | Using this bit it is possible to select, whether the eUSCI_B module triggers the sending of the ACK of the address or if it is controlled by software.<br>0b = The address acknowledge of the slave is controlled by the eUSCI_B module<br>1b = The user needs to trigger the sending of the address ACK by issuing UCTXACK |
| 3-2 | UCASTPx | RW | 0h | Automatic STOP condition generation. In slave mode only UCBCNTIFG is available.<br>Modify only when UCSWRST = 1.<br>00b = No automatic STOP generation. The STOP condition is generated after the user sets the UCTXSTP bit. The value in UCBxTBCNT is a don't care.<br>01b = UCBCNTIFG is set with the byte counter reaches the threshold defined in UCBxTBCNT<br>10b = A STOP condition is generated automatically after the byte counter value reached UCBxTBCNT. UCBCNTIFG is set with the byte counter reaching the threshold.<br>11b = Reserved |

### Table 12-5. UCBxCTLW1 Register Description (continued)

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 1-0 | UCGLITx | RW | 0h | Deglitch time<br>00b = 50 ns<br>01b = 25 ns<br>10b = 12.5 ns<br>11b = 6.25 ns |

### 12.4.3 UCBxBRW Register

eUSCI_Bx Bit Rate Control Word Register

**Figure 12-19. UCBxBRW Register**

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

▨ Modify only when UCSWRST = 1.

**Table 12-6. UCBxBRW Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | UCBRx | RW | 0h | Bit clock prescaler.<br>Modify only when UCSWRST = 1. |

### 12.4.4 UCBxSTATW

eUSCI_Bx Status Word Register

**Figure 12-20. UCBxSTATW Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| UCBCNTx | | | | | | | |
| r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Reserved | UCSCLLOW | UCGC | UCBBUSY | Reserved | | | |
| r0 | r-0 | r-0 | r-0 | r-0 | r0 | r0 | r0 |

**Table 12-7. UCBxSTATW Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | UCBCNTx | R | 0h | Hardware byte counter value. Reading this register returns the number of bytes received or transmitted on the I2C-Bus since the last START or RESTART. There is no synchronization of this register done. When reading UCBxBCNT during the first bit position, a faulty readback can occur. |
| 7 | Reserved | R | 0h | Reserved |
| 6 | UCSCLLOW | R | 0h | SCL low<br>0b = SCL is not held low<br>1b = SCL is held low |
| 5 | UCGC | R | 0h | General call address received. UCGC is automatically cleared when a START condition is received.<br>0b = No general call address received<br>1b = General call address received |
| 4 | UCBBUSY | R | 0h | Bus busy<br>0b = Bus inactive<br>1b = Bus busy |
| 3-0 | Reserved | R | 0h | Reserved |

### 12.4.5 UCBxTBCNT Register

eUSCI_Bx Byte Counter Threshold Register

**Figure 12-21. UCBxTBCNT Register**

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

▨ Modify only when UCSWRST = 1.

**Table 12-8. UCBxTBCNT Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | Reserved | R | 0h | Reserved |
| 7-0 | UCTBCNTx | RW | 0h | The byte counter threshold value is used to set the number of I2C data bytes after which the automatic STOP or the UCSTPIFG should occur. This value is evaluated only if UCASTPx is different from 00.<br>Modify only when UCSWRST = 1. |

## 12.4.6 UCBxRXBUF Register

eUSCI_Bx Receive Buffer Register

**Figure 12-22. UCBxRXBUF Register**

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

**Table 12-9. UCBxRXBUF Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | Reserved | R | 0h | Reserved |
| 7-0 | UCRXBUFx | R | 0h | The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading UCBxRXBUF resets the UCRXIFGx flags. |

## 12.4.7 UCBxTXBUF

eUSCI_Bx Transmit Buffer Register

**Figure 12-23. UCBxTXBUF Register**

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

**Table 12-10. UCBxTXBUF Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | Reserved | R | 0h | Reserved |
| 7-0 | UCTXBUFx | RW | 0h | The transmit data buffer is user accessible and holds the data waiting to be moved into the transmit shift register and transmitted. Writing to the transmit data buffer clears the UCTXIFGx flags. |

### 12.4.8 UCBxI2COA0 Register

eUSCI_Bx I2C Own Address 0 Register

**Figure 12-24. UCBxI2COA0 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| UCGCEN | Reserved | | | | UCOAEN | I2COA0 | |
| rw-0 | r0 | r0 | r0 | r0 | rw-0 | rw-0 | rw-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| I2COA0 | | | | | | | |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |

▨ Modify only when UCSWRST = 1.

**Table 12-11. UCBxI2COA0 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15 | UCGCEN | RW | 0h | General call response enable. This bit is only available in UCBxI2COA0.<br>Modify only when UCSWRST = 1.<br>0b = Do not respond to a general call<br>1b = Respond to a general call |
| 14-11 | Reserved | R | 0h | Reserved |
| 10 | UCOAEN | RW | 0h | Own Address enable register. With this register it can be selected if the I2C slave-address related to this register UCBxI2COA0 is evaluated or not.<br>Modify only when UCSWRST = 1.<br>0b = The slave address defined in I2COA0 is disabled<br>1b = The slave address defined in I2COA0 is enabled |
| 9-0 | I2COAx | RW | 0h | I2C own address. The I2COA0 bits contain the local address of the eUSCIx_B I2C controller. The address is right justified. In 7-bit addressing mode, bit 6 is the MSB and bits 9-7 are ignored. In 10-bit addressing mode, bit 9 is the MSB.<br>Modify only when UCSWRST = 1. |

### 12.4.9 UCBxI2COA1 Register

eUSCI_Bx I2C Own Address 1 Register

**Figure 12-25. UCBxI2COA1 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | UCOAEN | I2COA1 | |
| rw-0 | r0 | r0 | r0 | r0 | rw-0 | rw-0 | rw-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| I2COA1 | | | | | | | |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |

▨ Modify only when UCSWRST = 1.

**Table 12-12. UCBxI2COA1 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-11 | Reserved | R | 0h | Reserved |
| 10 | UCOAEN | RW | 0h | Own Address enable register. With this register it can be selected if the I2C slave-address related to this register UCBxI2COA1 is evaluated or not.<br>Modify only when UCSWRST = 1.<br>0b = The slave address defined in I2COA1 is disabled<br>1b = The slave address defined in I2COA1 is enabled |
| 9-0 | I2COA1 | RW | 0h | I2C own address. The I2COAx bits contain the local address of the eUSCIx_B I2C controller. The address is right justified. In 7-bit addressing mode, bit 6 is the MSB and bits 9-7 are ignored. In 10-bit addressing mode, bit 9 is the MSB.<br>Modify only when UCSWRST = 1. |

### 12.4.10 UCBxI2COA2 Register

eUSCI_Bx I2C Own Address 2 Register

**Figure 12-26. UCBxI2COA2 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | UCOAEN | I2COA2 | |
| rw-0 | r0 | r0 | r0 | r0 | rw-0 | rw-0 | rw-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| I2COA2 | | | | | | | |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |

▨ Modify only when UCSWRST = 1.

**Table 12-13. UCBxI2COA2 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-11 | Reserved | R | 0h | Reserved |
| 10 | UCOAEN | RW | 0h | Own Address enable register. With this register it can be selected if the I2C slave-address related to this register UCBxI2COA2 is evaluated or not.<br>Modify only when UCSWRST = 1.<br>0b = The slave address defined in I2COA2 is disabled<br>1b = The slave address defined in I2COA2 is enabled |
| 9-0 | I2COA2 | RW | 0h | I2C own address. The I2COAx bits contain the local address of the eUSCIx_B I2C controller. The address is right justified. In 7-bit addressing mode, bit 6 is the MSB and bits 9-7 are ignored. In 10-bit addressing mode, bit 9 is the MSB.<br>Modify only when UCSWRST = 1. |

### 12.4.11 UCBxI2COA3 Register

eUSCI_Bx I2C Own Address 3 Register

**Figure 12-27. UCBxI2COA3 Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | UCOAEN | I2COA3 | |
| rw-0 | r0 | r0 | r0 | r0 | rw-0 | rw-0 | rw-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| I2COA3 | | | | | | | |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |

▨ Modify only when UCSWRST = 1.

**Table 12-14. UCBxI2COA3 Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-11 | Reserved | R | 0h | Reserved |
| 10 | UCOAEN | RW | 0h | Own Address enable register. With this register it can be selected if the I2C slave-address related to this register UCBxI2COA3 is evaluated or not.<br>Modify only when UCSWRST = 1.<br>0b = The slave address defined in I2COA3 is disabled<br>1b = The slave address defined in I2COA3 is enabled |
| 9-0 | I2COA3 | RW | 0h | I2C own address. The I2COA3 bits contain the local address of the eUSCIx_B I2C controller. The address is right justified. In 7-bit addressing mode, bit 6 is the MSB and bits 9-7 are ignored. In 10-bit addressing mode, bit 9 is the MSB.<br>Modify only when UCSWRST = 1. |

### 12.4.12 UCBxADDRX Register

eUSCI_Bx I2C Received Address Register

**Figure 12-28. UCBxADDRX Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | ADDRXx | |
| r-0 | r0 | r0 | r0 | r0 | r0 | r-0 | r-0 |

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

**Table 12-15. UCBxADDRX Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-10 | Reserved | R | 0h | Reserved |
| 9-0 | ADDRXx | R | 0h | Received Address Register. This register contains the last received slave address on the bus. Using this register and the address mask register it is possible to react on more than one slave address using one eUSCI_B module. |

### 12.4.13 UCBxADDMASK Register

eUSCI_Bx I2C Address Mask Register

**Figure 12-29. UCBxADDMASK Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | ADDMASKx | |
| r-0 | r0 | r0 | r0 | r0 | r0 | rw-1 | rw-1 |

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

▨ Modify only when UCSWRST = 1.

**Table 12-16. UCBxADDMASK Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-10 | Reserved | R | 0h | Reserved |
| 9-0 | ADDMASKx | RW | 3FFh | Address Mask Register. By clearing the corresponding bit of the own address, this bit is a don't care when comparing the address on the bus to the own address. Using this method, it is possible to react on more than one slave address. When all bits of ADDMASKx are set, the address mask feature is deactivated.<br>Modify only when UCSWRST = 1. |

### 12.4.14 UCBxI2CSA Register

eUSCI_Bx I2C Slave Address Register

**Figure 12-30. UCBxI2CSA Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | I2CSAx | |
| r-0 | r0 | r0 | r0 | r0 | r0 | rw-0 | rw-0 |

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

**Table 12-17. UCBxI2CSA Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-10 | Reserved | R | 0h | Reserved |
| 9-0 | I2CSAx | RW | 0h | I2C slave address. The I2CSAx bits contain the slave address of the external device to be addressed by the eUSCIx_B module. It is only used in master mode. The address is right justified. In 7-bit slave addressing mode, bit 6 is the MSB and bits 9-7 are ignored. In 10-bit slave addressing mode, bit 9 is the MSB. |

### 12.4.15 UCBxIE Register

eUSCI_Bx I2C Interrupt Enable Register

**Figure 12-31. UCBxIE Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | UCBIT9IE | UCTXIE3 | UCRXIE3 | UCTXIE2 | UCRXIE2 | UCTXIE1 | UCRXIE1 |
| r0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCCLTOIE | UCBCNTIE | UCNACKIE | UCALIE | UCSTPIE | UCSTTIE | UCTXIE0 | UCRXIE0 |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |

**Table 12-18. UCBxIE Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15 | Reserved | R | 0h | Reserved |
| 14 | UCBIT9IE | RW | 0h | Bit position 9 interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 13 | UCTXIE3 | RW | 0h | Transmit interrupt enable 3<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 12 | UCRXIE3 | RW | 0h | Receive interrupt enable 3<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 11 | UCTXIE2 | RW | 0h | Transmit interrupt enable 2<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 10 | UCRXIE2 | RW | 0h | Receive interrupt enable 2<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 9 | UCTXIE1 | RW | 0h | Transmit interrupt enable 1<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 8 | UCRXIE1 | RW | 0h | Receive interrupt enable 1<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 7 | UCCLTOIE | RW | 0h | Clock low timeout interrupt enable.<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 6 | UCBCNTIE | RW | 0h | Byte counter interrupt enable.<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 5 | UCNACKIE | RW | 0h | Not-acknowledge interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 4 | UCALIE | RW | 0h | Arbitration lost interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 3 | UCSTPIE | RW | 0h | STOP condition interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |

### Table 12-18. UCBxIE Register Description (continued)

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 2 | UCSTTIE | RW | 0h | START condition interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 1 | UCTXIE0 | RW | 0h | Transmit interrupt enable 0<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 0 | UCRXIE0 | RW | 0h | Receive interrupt enable 0<br>0b = Interrupt disabled<br>1b = Interrupt enabled |

### 12.4.16 UCBxIFG Register

eUSCI_Bx I2C Interrupt Flag Register

**Figure 12-32. UCBxIFG Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | UCBIT9IFG | UCTXIFG3 | UCRXIFG3 | UCTXIFG2 | UCRXIFG2 | UCTXIFG1 | UCRXIFG1 |
| r0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCCLTOIFG | UCBCNTIFG | UCNACKIFG | UCALIFG | UCSTPIFG | UCSTTIFG | UCTXIFG0 | UCRXIFG0 |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-1 | rw-0 |

**Table 12-19. UCBxIFG Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15 | Reserved | R | 0h | Reserved |
| 14 | UCBIT9IFG | RW | 0h | Bit position 9 interrupt flag<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 13 | UCTXIFG3 | RW | 0h | eUSCI_B transmit interrupt flag 3. UCTXIFG3 is set when UCBxTXBUF is empty in slave mode, if the slave address defined in UCBxI2COA3 was on the bus in the same frame.<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 12 | UCRXIFG3 | RW | 0h | Receive interrupt flag 2. UCRXIFG2 is set when UCBxRXBUF has received a complete byte in slave mode and if the slave address defined in UCBxI2COA2 was on the bus in the same frame.<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 11 | UCTXIFG2 | RW | 0h | eUSCI_B transmit interrupt flag 2. UCTXIFG2 is set when UCBxTXBUF is empty in slave mode, if the slave address defined in UCBxI2COA2 was on the bus in the same frame.<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 10 | UCRXIFG2 | RW | 0h | Receive interrupt flag 2. UCRXIFG2 is set when UCBxRXBUF has received a complete byte in slave mode and if the slave address defined in UCBxI2COA2 was on the bus in the same frame.<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 9 | UCTXIFG1 | RW | 0h | eUSCI_B transmit interrupt flag 1. UCTXIFG1 is set when UCBxTXBUF is empty in slave mode, if the slave address defined in UCBxI2COA1 was on the bus in the same frame.<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 8 | UCRXIFG1 | RW | 0h | Receive interrupt flag 1. UCRXIFG1 is set when UCBxRXBUF has received a complete byte in slave mode and if the slave address defined in UCBxI2COA1 was on the bus in the same frame.<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 7 | UCCLTOIFG | RW | 0h | Clock low timeout interrupt flag<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 6 | UCBCNTIFG | RW | 0h | Byte counter interrupt flag. When using this interrupt the user needs to ensure enough processing bandwidth (see the Byte Counter Interrupt section).<br>0b = No interrupt pending<br>1b = Interrupt pending |

Table 12-19. UCBxIFG Register Description (continued)

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 5 | UCNACKIFG | RW | 0h | Not-acknowledge received interrupt flag. This flag only is updated when operating in master mode.<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 4 | UCALIFG | RW | 0h | Arbitration lost interrupt flag<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 3 | UCSTPIFG | RW | 0h | STOP condition interrupt flag<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 2 | UCSTTIFG | RW | 0h | START condition interrupt flag<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 1 | UCTXIFG0 | RW | 0h | eUSCI_B transmit interrupt flag 0. UCTXIFG0 is set when UCBxTXBUF is empty in master mode or in slave mode, if the slave address defined in UCBxI2COA0 was on the bus in the same frame.<br>0b = No interrupt pending<br>1b = Interrupt pending |
| 0 | UCRXIFG0 | RW | 0h | eUSCI_B receive interrupt flag 0. UCRXIFG0 is set when UCBxRXBUF has received a complete character in master mode or in slave mode, if the slave address defined in UCBxI2COA0 was on the bus in the same frame.<br>0b = No interrupt pending<br>1b = Interrupt pending |

### 12.4.17 UCBxIV Register

eUSCI_Bx Interrupt Vector Register

**Figure 12-33. UCBxIV Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| UCIVx | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| UCIVx | | | | | | | |
| r0 | r0 | r0 | r0 | r-0 | r-0 | r-0 | r0 |

**Table 12-20. UCBxIV Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | UCIVx | R | 0h | eUSCI_B interrupt vector value. It generates an value that can be used as address offset for fast interrupt service routine handling. Writing to this register clears all pending interrupt flags.<br>00h = No interrupt pending<br>02h = Interrupt Source: Arbitration lost; Interrupt Flag: UCALIFG; Interrupt Priority: Highest<br>04h = Interrupt Source: Not acknowledgment; Interrupt Flag: UCNACKIFG<br>06h = Interrupt Source: Start condition received; Interrupt Flag: UCSTTIFG<br>08h = Interrupt Source: Stop condition received; Interrupt Flag: UCSTPIFG<br>0Ah = Interrupt Source: Slave 3 Data received; Interrupt Flag: UCRXIFG3<br>0Ch = Interrupt Source: Slave 3 Transmit buffer empty; Interrupt Flag: UCTXIFG3<br>0Eh = Interrupt Source: Slave 2 Data received; Interrupt Flag: UCRXIFG2<br>10h = Interrupt Source: Slave 2 Transmit buffer empty; Interrupt Flag: UCTXIFG2<br>12h = Interrupt Source: Slave 1 Data received; Interrupt Flag: UCRXIFG1<br>14h = Interrupt Source: Slave 1 Transmit buffer empty; Interrupt Flag: UCTXIFG1<br>16h = Interrupt Source: Data received; Interrupt Flag: UCRXIFG0<br>18h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG0<br>1Ah = Interrupt Source: Byte counter zero; Interrupt Flag: UCBCNTIFG<br>1Ch = Interrupt Source: Clock low timeout; Interrupt Flag: UCCLTOIFG<br>1Eh = Interrupt Source: Nineth bit position; Interrupt Flag: UCBIT9IFG; Priority: Lowest |

Chapter 13

SLAU506–December 2014

# 13.56-MHz RF Communication Module (RF13M)

The RF13M module is an NFC / RFIDcommunication module that supports ISO 15693 communication protocols. This chapter describes the module.

**Topic** **Page**

## 13.1 RF13M Module Introduction

The RF13M module consists of a 13.56-MHz passive ISO 15693 compliant analog front end. It also comprises an encoding and decoding finite state machine (FSM) with separate 32-byte receive and transmit buffers. In addition to the demodulator and modulator functions, the analog front end can provide supply power for the device. The receive and transmit buffers can be operated in either buffer mode or FIFO mode, which allows the application to send and receive packets greater than the physical size of the buffers.

Features of the RF13M include:

- Power supply generation for the device
- Full ISO 15693 compliance
- Separate 32-byte receive and transmit data buffers
- FIFO mode allows data packets greater than 32 bytes
- Automatic CRC generation and verification

Figure 13-1 shows a block diagram of the RF13M module.

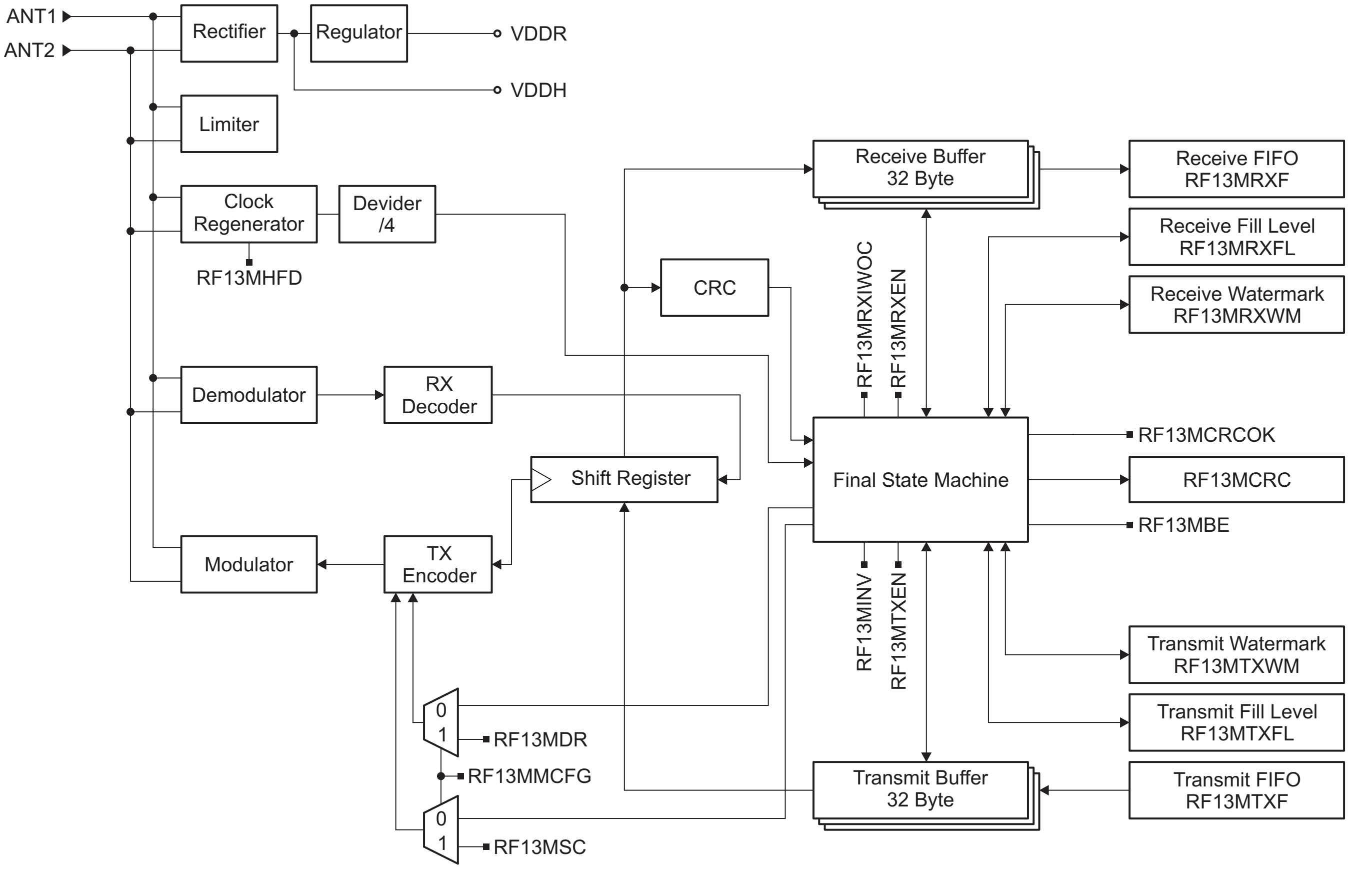


**Figure 13-1. RF13M Module Block Diagram**

## 13.2 RF13M Operation

The RF13M module is configured with software. The setup and operation of the module is discussed in the following sections.

### 13.2.1 Principle of Operation

The RF13M module is a slave in ISO 15693 NFC / RFID systems. Such a slave is often called a tag or a transponder. The reader always initiates the communication, and the transponder sends a response back to the reader. The communication path from the reader to the transponder is called the downlink, and the response is called the uplink. The reader sends a downlink to the transponder by 10% or 100% ASK modulation of the 13.56-MHz carrier. The transponder uplink is implemented by using a modulated subcarrier which is itself added to the 13.56-MHz carrier by ASK load modulation. Figure 13-2 shows the downlink and uplink principle. Each downlink and uplink is framed by special symbols called Start of Frame (SOF) and End of Frame (EOF). See the ISO 15693 specifications for further details.

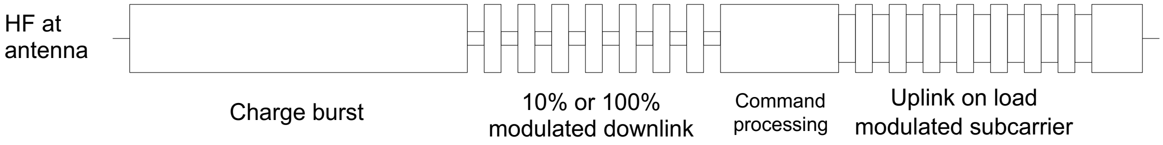

HF at antenna
Charge burst
10% or 100% modulated downlink
Command processing
Uplink on load modulated subcarrier

**Figure 13-2. Schematic of Downlink Followed by Uplink**

### 13.2.2 Receive Operation

When the module is enabled and detects a start of downlink, the RX and TX FSM waits for a valid SOF and then automatically decodes the received data and writes it into the RX buffer. When an EOF is detected, the reception stops, and an interrupt is optionally asserted. A CRC value is calculated for the received downlink, and the interrupt assertion can optionally be blocked if the CRC value is not as expected.

A high watermark interrupt can be asserted when the RX FIFO fill level has reached a configurable value. This interrupt is designed for use in FIFO mode.

When more data is received than fits in the RX buffer or when the RX FIFO fills up in FIFO mode, an overflow interrupt is asserted.

### 13.2.3 Reception of Inventory Slot Markers

In an ISO 15693 system, the reader may send slot marker symbols after an inventory command. The detection of these slot markers by the module is only possible when inventory mode is active. Inventory mode is automatically activated if the inventory bit of the flags byte (the first byte of the downlink) is set or the application has manually overwritten the automatic configuration by setting RF13MMCFG = 1 and RF13MINV = 1. When a slot marker is detected, the module asserts the slot marker interrupt by setting the RF13MSLIFG interrupt flag.

### 13.2.4 Transmit Operation

After a valid downlink has been received, the module is ready to send an uplink. The uplink is started when RF13MTXEN = 1 and the transmit size is non-zero. The transmit size can be configured by writing directly to the RF13MTXFL register, which is also automatically incremented when data is written to the TX FIFO using the RF13MTXF register. The module starts the uplink with a SOF symbol and continues to send an uplink as long as the TX FIFO fill level has not reached zero. When there is no more data to be sent (fill level = 0), the FSM finishes the uplink by transmitting an EOF symbol, and an interrupt is optionally asserted.

As with the receive side, an additional low watermark interrupt can be asserted when the TX FIFO fill level has reached a configurable value.

If no uplink should be sent, the RF13MTXEN bit can be set to 0, and the module is ready to receive the next downlink.

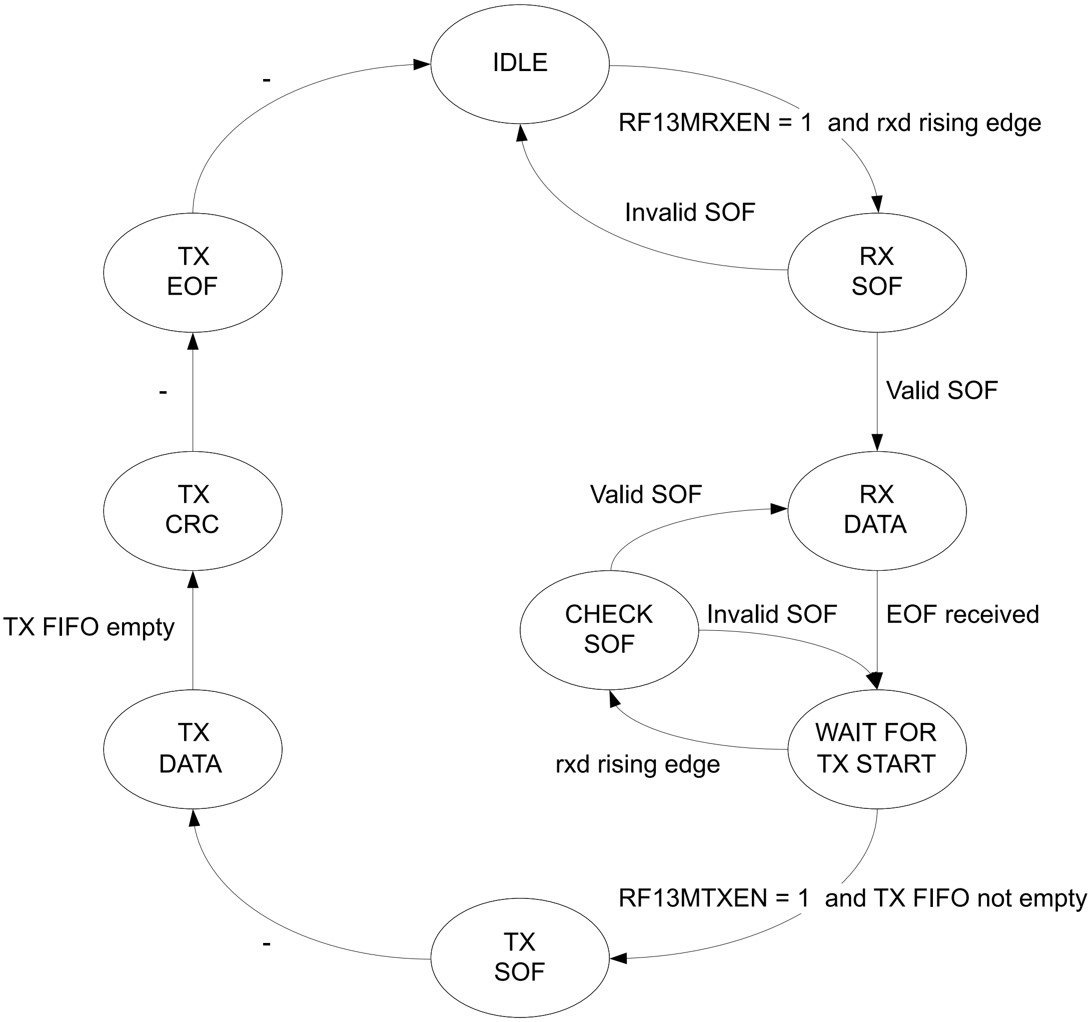


**Figure 13-3. Flow Diagram of RX and TX Finite State Machine**

### 13.2.5 Read and Write Data to the RX and TX Buffers or FIFOs

The data buffers for both RX and TX can be used in directly memory mapped buffer mode or in FIFO mode. The two RX and TX data buffers are accessible at memory locations starting from RF13MRXFBASE and RF13MTXFBASE. The FIFO read and write registers (RF13MRXF and RF13MTXF) are located in the RF13M register bank.

**NOTE:** Both modes are active at the same time, and the application can choose which mode is used.

#### 13.2.5.1 FIFO Mode

In FIFO mode data is read and written to the buffers using the RF13MRXF and RF13MTXF registers. In this mode, data frames greater than 32 bytes can be received and transmitted if the software reads or writes data before the FIFOs fill up or become empty, respectively. The module supports both byte and word mode accesses to the FIFO register, and the endianness of these accesses can be configured by the RF13MBE control bit.

The FIFOs are implemented as ring buffers using two separate read and write indices as shown in Figure 13-4. The indices wrap around to zero when incremented past 31. The formulas for incrementing the indices and calculating the fill level are:

new index = (old index + 1) mod 32

fill level = (write index – read index) mod 32

The indices point to the next address to be written or to be read. When both indices are equal, the FIFO is empty as shown in Figure 13-4a. Figure 13-4b shows a FIFO with fill level of 5 where the write pointer has wrapped around.

The indices for the RX FIFO are automatically reset to 0 by the RX and TX FSM at the start of each downlink. The indices for the TX FIFO are automatically reset to 0 at the end of each uplink.

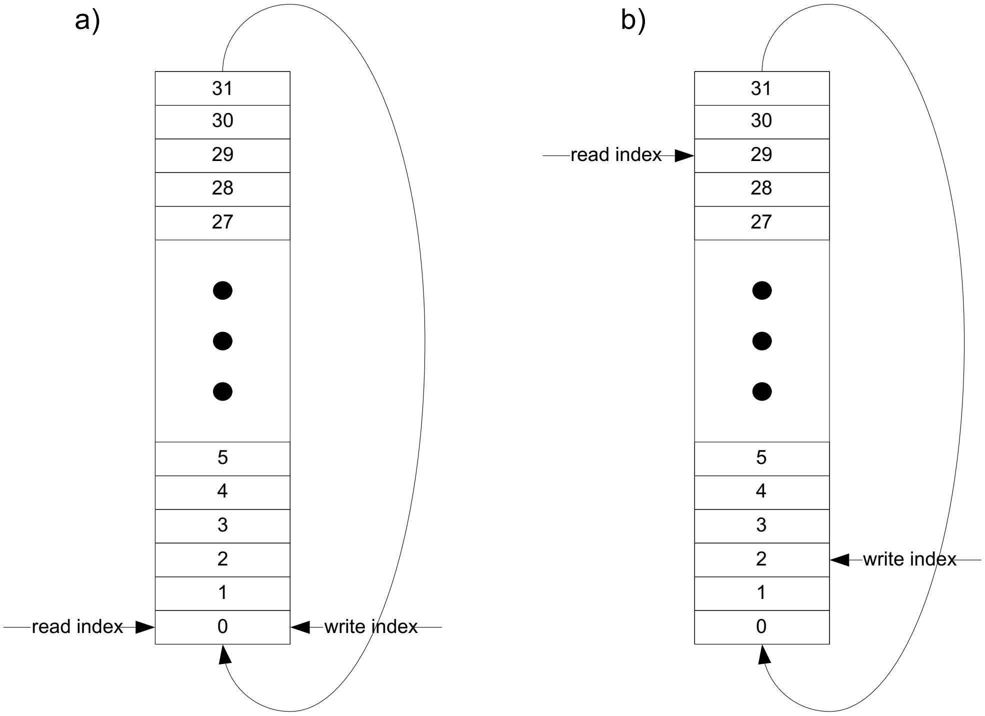
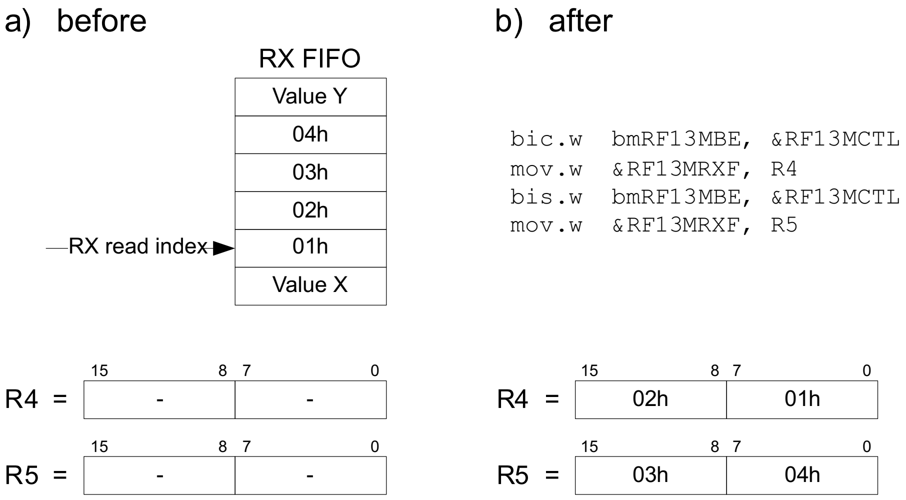


**Figure 13-4. RX and TX FIFO Buffers as Ring Buffers**

### 13.2.5.2 Byte Order in Word Mode

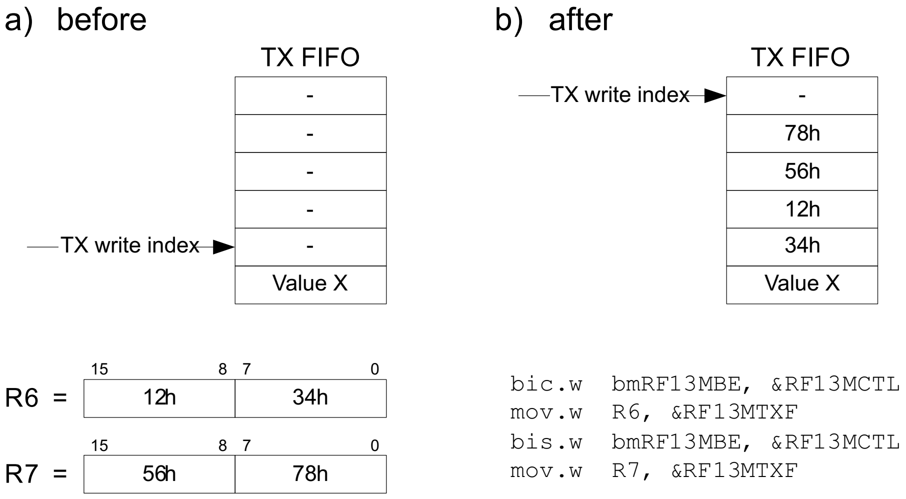
The FIFOs can be read from and written to in both byte and word mode. When the FIFOs are accessed in word mode, the order of the two bytes is determined by the RF13MBE bit in the RF13MCTL register. When RF13MBE = 0, little endian mode is used and the LSB of the data read or written is the first byte to be processed, and the MSB is the second. When RF13MBE = 1, the MSB is the first byte to be processed and the LSB is the second.

Figure 13-5 and Figure 13-6 show the effect of the RF13MBE bit on word mode read and write accesses to the FIFOs.



```
bic.w  bmRF13MBE, &RF13MCTL
mov.w  &RF13MRXF, R4
bis.w  bmRF13MBE, &RF13MCTL
mov.w  &RF13MRXF, R5
```

**Figure 13-5. Word Mode Read From RX FIFO With Little and Big Endian Mode**



```
bic.w  bmRF13MBE, &RF13MCTL
mov.w  R6, &RF13MTXF
bis.w  bmRF13MBE, &RF13MCTL
mov.w  R7, &RF13MTXF
```

**Figure 13-6. Word Mode Write to TX FIFO With Little and Big Endian Mode**

### 13.2.5.3 Buffer Mode

In buffer mode, all 32 bytes of each buffer are read and written like normal memory. After reception of a downlink is finished, the RF13MRXFL register contains the number of received bytes that are stored in the RX buffer. The first byte is always at offset 0 in the RX buffer.

After the data to be transmitted has been written to the TX buffer, the RF13MTXFL register must configured to the number of bytes to transmit. The first byte to transmit is always at offset 0 in the TX buffer.

**NOTE:** The RX and TX FSM uses the FIFO mechanism even when data is directly written to the TX buffer. Writing to the RF13TXFL register updates the TX FIFO write index, resulting in a TX FIFO fill level equal to the written value, because the TX FIFO read index is 0 before transmit is started. Similarly, RF13MRXFL contains the number of received bytes, because the RX FIFO read index is still 0 (if the application did not access the RF13MRXF register during downlink).

### 13.2.6 Receive High and Transmit Low Watermarks

When using the FIFO mode, the RX high watermark and TX low watermark interrupts can be used to read and refill the FIFOs before they fill up or become empty.

The RX FIFO high watermark interrupt is asserted when the RX FIFO fill level reaches the value configured by the RF13MRXWM register. The interrupt is only asserted when this level is reached from below; that is, when the fill level is increased by a reception of a byte. It is not asserted when the fill levels reaches this value because the FIFO was read. The application should set this watermark level to a value that allows it to read the FIFO before it is completely full.

A similar concept applies to the TX FIFO low watermark interrupt, which is asserted when the TX FIFO fill level drops during transmit to the value configured by the RF13MTXWM register. It is not asserted when the fill level reaches this value because it was written to by the application.

### 13.2.7 Uplink Parameters and Inventory Mode for ISO 15693

The ISO 15693 standard specifies four types of uplink modes:

- Low data rate using single subcarrier frequency
- High data rate using single subcarrier frequency
- Low data rate using two subcarrier frequencies
- High data rate using two subcarrier frequencies

The mode in use is defined by two bits in the flags byte, which is the first byte of the downlink. The RX and TX FSM automatically extracts these bits from the received downlink and configures the uplink mode accordingly. The user can optionally override the received mode flag by setting the RF13MMCFG bit and then configuring the mode using the RF13MDR and RF13MSC bits in the RF13MCTL register.

The user can also override the automatic detection of inventory mode, which is also activated by a bit in the flags byte of the downlink. The RX and TX FSM only asserts the slot marker interrupt when the inventory mode is set and a valid downlink is received.

### 13.2.8 FIFO Overflow and Underflow Interrupt

When the module is used in FIFO mode, an overflow or underflow condition can occur. The module asserts the RF13MOUFLIFG interrupt flag when such an condition is detected.

When the RX FIFO is full and an additional byte is received, an overflow condition occurs. An underflow also occurs when the buffer mode is used and the reader sends more bytes than fit in the RX buffer. A RX FIFO underflow condition occurs when the RF13MRXF register is read if the FIFO is empty or if it contains only one byte but is read using word mode.

A TX FIFO overflow conditions occurs when the RF13MTXF register is written when it is full or when it is written in word mode but there is space for only one byte. A TX FIFO underflow condition cannot occur, because the RX and TX FSM treats the TX FIFO fill level = 0 condition as a end of uplink condition.

**NOTE:** RF13MRXIFG is asserted at the end of the downlink even if an RX overflow condition is detected. The application must verify that no overflow occurred during the downlink when processing the downlink data.

### 13.2.9 RX Error Interrupt

If the RX and TX FSM detects an error during reception of a downlink, an RX error interrupt is asserted by setting the RF13MRXEIFG bit. This flag is asserted only if the error is detected after a valid SOF was received. An invalid SOF does not assert the interrupt.

### 13.2.10 Cyclic Redundancy Check (CRC)

The module automatically calculates a CRC for the received downlink and the transmitted uplink using the polynomial specified by the ISO 15693 standard. If RF13MRXIWOC = 0, the RF13MRXIFG interrupt flag is only asserted if the received CRC bytes match the calculated ones. If RF13MRXIWOC = 1, the module asserts the RF13MRXIFG interrupt independent of the correctness of the received CRC.

The user can read the resulting CRC accumulator through the RF13MCRC register.

### 13.2.11 HF Detection Flag

For very sensitive measurement applications, the presence of an HF field can influence the measurement. The module therefore asserts the RF13MHFD flag whenever a HF field is detected. This flag is reset on read and can be polled after each measurement cycle to verify that no HF field was present since the measurement cycle was started.

## 13.3 RF13M Registers

Table 13-1 shows the module registers. All register can be accessed both byte- and word-wise.

**Table 13-1. RF13M Registers**

| Offset | Acronym | Register Name | Type | Reset | Section |
|---|---|---|---|---|---|
| 00h | RF13MCTL | RF13M Control register | Read/write | 0000h | Section 13.3.1 |
| 02h | RF13MINT | RF13M Interrupt register | Read/write | 0000h | Section 13.3.2 |
| 04h | RF13MIV | RF13M Interrupt Vector register | Read only | 0000h | Section 13.3.3 |
| 06h | RF13MRXF | RF13M Receive Data FIFO register | Read only | 0000h | Section 13.3.4 |
| 08h | RF13MTXF | RF13M Transmit Data FIFO register | Write only | 0000h | Section 13.3.5 |
| 0Ah | RF13MCRC | RF13M CRC accumulator register | Read only | FFFFh | Section 13.3.6 |
| 0Ch | RF13MFIFOFL | RF13M RX/TX FIFO Fill Level register | Read only | 0000h | Section 13.3.7 |
| 0Eh | RF13MWMCFG | RF13M RX/TX High/Low Watermark configuration register | Read/write | 0303h | Section 13.3.8 |

### 13.3.1 RF13MCTL Register

RF13M Module Control Register

**Figure 13-7. RF13MCTL Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | RF13MBE | RF13M<br>RXIWOC | Reserved | | | RF13MHFD | RF13MCRCOK |
| r-0 | rw-0 | rw-0 | r0 | r0 | r0 | r-0 | r-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RF13MMCFG | RF13MINV | RF13MDR | RF13MSC | Reserved | RFTOEN | RF13MTXEN | RF13MRXEN |
| rw-0 | rw-0 | rw-0 | rw-0 | r0 | rw-0 | rw-0 | rw-0 |

**Table 13-2. RF13MCTL Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15 | Reserved | R | 0h | Reserved. Must be written to 0. |
| 14 | RF13MBE | RW | 0h | Select between big or little endian mode when reading and writing RX and TX FIFOs in word mode<br>0b = Little endian mode selected<br>1b = Big endian mode selected |
| 13 | RF13MRXIWOC | RW | 0h | Gate RX interrupt with CRC OK<br>0b = RX interrupt only generated when CRC is OK<br>1b = RX interrupt generated independent of CRC OK |
| 12-10 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 9 | RF13MHFD | R | 0h | Indicates if a HF field was detected since last readout. Register is reset on read.<br>0b = No HF field detected<br>1b = HF field detected |
| 8 | RF13MCRCOK | R | 0h | CRC valued correct or not flag<br>0b = CRC is not correct<br>1b = CRC is correct |
| 7 | RF13MMCFG | RW | 0h | Enable manual configuration of ISO 15693 uplink parameters and inventory mode settings using bits 6-4.<br>0b = Automatically configure uplink parameters using flags in first received data byte<br>1b = Manually configure uplink parameters using bits 6-4 |
| 6 | RF13MINV | RW | 0h | Manually enable inventory mode and thereby enable detection of slot markers<br>0b = Inventory mode not active<br>1b = Inventory mode active |
| 5 | RF13MDR | RW | 0h | Manually select uplink data rate<br>0b = Low data rate selected<br>1b = High data rate selected |
| 4 | RF13MSC | RW | 0h | Manually select uplink subcarrier setting<br>0b = Single subcarrier frequency<br>1b = Two subcarrier frequencies |
| 3 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 2 | RFTOEN | RW | 0h | Enable RF Timeout detection<br>0b = RF Timeout detection is disabled<br>1b = RF Timeout detection is enabled |
| 1 | RF13MTXEN | RW | 0h | Enable transmission of uplink<br>0b = Uplink disabled<br>1b = Uplink enabled |

Table 13-2. RF13MCTL Register Description (continued)

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 0 | RF13MRXEN | RW | 0h | Enable reception of downlink<br>0b = Downlink reception disabled<br>1b = Downlink reception enabled |

### 13.3.2 RF13MINT Register

RF13M Module Interrupt Register

**Figure 13-8. RF13MINT Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| RF13MRFTOIE | RF13MRXEIE | RF13MOUFLIE | RF13MSLIE | RF13MTXWMIE | RF13MRXWMIE | RF13MTXIE | RF13MRXIE |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RF13MRFTOIFG | RF13MRXEIFG | RF13MOUFLIFG | RF13MSLIFG | RF13MTXWMIFG | RF13MRXWMIFG | RF13MTXIFG | RF13MRXIFG |
| rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |

**Table 13-3. RF13MINT Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15 | RX13MRFTOIE | RW | 0h | RF timeout interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 14 | RX13MRXEIE | RW | 0h | RX error interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 13 | RX13MOUFLIE | RW | 0h | Overflow or underflow interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 12 | RF13MSLIE | RW | 0h | ISO 15693 slot marker interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 11 | RF13MTXWMIE | RW | 0h | TX FIFO low watermark interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 10 | RF13MRXWMIE | RW | 0h | RX FIFO high watermark interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 9 | RF13MTXIE | RW | 0h | TX done interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 8 | RF13MRXIE | RW | 0h | RX done interrupt enable<br>0b = Interrupt disabled<br>1b = Interrupt enabled |
| 7 | RF13MRFTOIFG | RW | 0h | RF timeout detected<br>0b = No RF timeout has been detected<br>1b = RF timeout has been detected |
| 6 | RF13MRXEIFG | RW | 0h | RX error detected<br>0b = No error has been detected<br>1b = Error has been detected during reception of downlink |
| 5 | RF13MOUFLIFG | RW | 0h | Overflow or underflow condition detected<br>0b = No overflow or underflow has been detected<br>1b = Overflow or underflow has been detected |
| 4 | RF13MSLIFG | RW | 0h | RX slot marker interrupt flag<br>0b = No slot marker has been detected<br>1b = ISO 15693 Slot marker detected |

### Table 13-3. RF13MINT Register Description (continued)

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 3 | RF13MTXWMIFG | RW | 0h | TX FIFO low watermark reached interrupt flag<br>0b = TX FIFO fill level still above configured level<br>1b = TX FIFO fill level has reached configured level |
| 2 | RF13MRXWMIFG | RW | 0h | RX FIFO high watermark reached interrupt flag<br>0b = RX FIFO fill level still below configured level<br>1b = RX FIFO fill level has reached configured level |
| 1 | RF13MTXIFG | RW | 0h | Uplink transmission done interrupt flag<br>0b = Uplink transmission not done<br>1b = Uplink transmission done |
| 0 | RF13MRXIFG | RW | 0h | Receive downlink done interrupt flag<br>0b = No or invalid downlink received<br>1b = Valid downlink received |

### 13.3.3 RF13MIV Register

RF13M Module Interrupt Vector Register

**Figure 13-9. RF13MIV Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| RF13MIVx | | | | | | | |
| r0 | r0 | r0 | r0 | r0 | r0 | r0 | r0 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RF13MIVx | | | | | | | |
| r0 | r0 | r0 | r-0 | r-0 | r-0 | r-0 | r0 |

**Table 13-4. RF13MIV Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-0 | RF13MIVx | R | 0h | 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.<br>00h = No interrupt pending<br>02h = Interrupt source: Receive done; Interrupt flag: RF13MRXIFG; Priority: Highest<br>04h = Interrupt source: Transmit done; Interrupt flag: RF13MTXIFG<br>06h = Interrupt source: Receive high watermark; Interrupt flag: RF13MRXWMIFG<br>08h = Interrupt source: Transmit low watermark; Interrupt flag: RF13MTXWMIFG<br>0Ah = Interrupt source: ISO 15693 slot marker detected; Interrupt flag: RF13MSLIFG<br>0Ch = Interrupt source: Overflow or underflow detected; Interrupt flag: RF13MOUFLIFG<br>0Eh = Interrupt source: RX error detected; Interrupt flag: RF13MRXEIFG<br>10h = Interrupt source: RF timeout detected; Interrupt flag: RF13MRFTOIFG; Priority: Lowest |

### 13.3.4 RF13MRXF Register

RF13M Module Receive FIFO Register

The RX FIFO register is a read only register. Reading this register in byte mode returns the next byte from the receive FIFO. Reading this register in word mode returns the next 2 bytes from the FIFO. The order of the bytes read depends on the RF13MBE configuration bit the RF13MCTL register. When RF13MBE is 0, the next byte in the buffer is returned on the LSByte and the second byte in the buffer on the MSByte. When RF13MBE is 1, the first byte is return on MSByte and the second on the LSByte.

Note: Underflow condition: Reading data from the RX FIFO while its empty (or reading two bytes when only one is available) can result in data loss.

**Figure 13-10. RF13MRXF Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| RF13MRXFH | | | | | | | |
| r | r | r | r | r | r | r | r |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RF13MRXFL | | | | | | | |
| r | r | r | r | r | r | r | r |

**Table 13-5. RF13MRXF Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | RF13MRXFH | R | 0h | Returns next byte in RX FIFO when read in byte mode. |
| 7-0 | RF13MRXFL | R | 0h | Returns next byte in RX FIFO when read in byte mode. |

### 13.3.5 RF13MTXF Register

RF13M Module Transmit FIFO Register

The TX FIFO register is a write only register. Writing to the register in byte mode adds a single byte to the transmit FIFO. Writing to the register in word mode adds 2 bytes to the transmit FIFO. The order with which the 2 bytes are added to the FIFO depends on the RF13MBE configuration bit the RF13MCTL register. When RF13MBE is 0 the LSByte is written first to the FIFO followed by the MSByte. When RF13MBE is 1 the MSByte is written first, followed by the LSByte.

Note: Overflow condition: Writing data to the TX FIFO when it is full (or adding 2 bytes when only 1 byte is free) results in data loss.

**Figure 13-11. RF13MTXF Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| RF13MTXFH | | | | | | | |
| w | w | w | w | w | w | w | w |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RF13MTXFL | | | | | | | |
| w | w | w | w | w | w | w | w |

**Table 13-6. RF13MTXF Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | RF13MTXFH | W | 0h | Byte to add to TX FIFO (word mode only) |
| 7-0 | RF13MTXFL | W | 0h | Byte to add to TX FIFO |

### 13.3.6 RF13MCRC Register

RF13M Module CRC Accumulator Register

**Figure 13-12. RF13MCRC Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| RF13MCRCH | | | | | | | |
| r-1 | r-1 | r-1 | r-1 | r-1 | r-1 | r-1 | r-1 |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| RF13MCRCL | | | | | | | |
| r-1 | r-1 | r-1 | r-1 | r-1 | r-1 | r-1 | r-1 |

**Table 13-7. RF13MCRC Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-8 | RF13MCRCH | RW | FFh | CRC accumulator high byte |
| 7-0 | RF13MCRCL | RW | FFh | CRC accumulator low byte |

### 13.3.7 RF13MFIFOFL Register

RF13M Module FIFO Fill Level Register

**Figure 13-13. RF13MFIFOFL Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | RF13MTXFL | | | | | |
| r0 | r0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 | rw-0 |
| **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Reserved | | RF13MRXFL | | | | | |
| r0 | r0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 |

**Table 13-8. RF13MFIFOFL Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-14 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 13-8 | RF13MTXFL | RW | 0h | Indicates the number of bytes to be transmitted in the TX data buffer. |
| 7-6 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 5-0 | RF13MRXFL | R | 0h | Indicates the number of received bytes available in the RX data buffer or FIFO. |

### 13.3.8 RF13MWMCFG Register

RF13M Module RX High/TX Low Watermark Configuration Register

**Figure 13-14. RF13MWMCFG Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | | | RF13MTXWM | | | | |
| r0 | r0 | r0 | rw-0 | rw-0 | rw-0 | rw-1 | rw-1 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | | | RF13MRXWM | | | | |
| r0 | r0 | r0 | rw-0 | rw-0 | rw-0 | rw-1 | rw-1 |

**Table 13-9. RF13MWMCFG Register Description**

| Bit | Field | Type | Reset | Description |
|---|---|---|---|---|
| 15-13 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 12-8 | RF13MTXWM | RW | 3h | TX low watermark fill level at which the TX FIFO low watermark interrupt is asserted. |
| 7-5 | Reserved | R | 0h | Reserved. Always reads as 0. |
| 4-0 | RF13MRXWM | RW | 3h | RX high watermark fill level at which the RX FIFO high watermark interrupt is asserted. |

# IMPORTANT NOTICE

| **Products** | | **Applications** | |
|---|---|---|---|
| Audio | www.ti.com/audio | Automotive and Transportation | www.ti.com/automotive |
| Amplifiers | amplifier.ti.com | Communications and Telecom | www.ti.com/communications |
| Data Converters | dataconverter.ti.com | Computers and Peripherals | www.ti.com/computers |
| DLP® Products | www.dlp.com | Consumer Electronics | www.ti.com/consumer-apps |
| DSP | dsp.ti.com | Energy and Lighting | www.ti.com/energy |
| Clocks and Timers | www.ti.com/clocks | Industrial | www.ti.com/industrial |
| Interface | interface.ti.com | Medical | www.ti.com/medical |
| Logic | logic.ti.com | Security | www.ti.com/security |
| Power Mgmt | power.ti.com | Space, Avionics and Defense | www.ti.com/space-avionics-defense |
| Microcontrollers | microcontroller.ti.com | Video and Imaging | www.ti.com/video |
| RFID | www.ti-rfid.com | | |
| OMAP Applications Processors | www.ti.com/omap | **TI E2E Community** | e2e.ti.com |
| Wireless Connectivity | www.ti.com/wirelessconnectivity | | |

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