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

NOTE: This chapter is an excerpt from the *MSP430x5xx and MSP430x6xx Family User's Guide*. The most recent version of the full user's guide is available at http://www.ti.com/lit/pdf/slau208.

The system control module (SYS) is available on all devices. The following list shows the basic feature set of SYS.

- Brownout reset (BOR) and power on reset (POR) handling
- Power up clear (PUC) handling
- (Non)maskable interrupt (SNMI and UNMI) event source selection and management
- Address decoding
- A user data-exchange mechanism using the JTAG mailbox (JMB)
- Bootloader (BSL) entry mechanism
- Configuration management (device descriptors)
- Provides interrupt vector generators for reset and NMIs

Topic **Page**

1.1 System Control Module (SYS) Introduction

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

1.2 System Reset and Initialization

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

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

- Powering up the device
- A low signal on $\overline{\text{RST}}$/NMI pin when configured in the reset mode
- A wake-up event from LPMx.5 (LPM3.5 or LPM4.5) modes
- A software BOR event
- A security violation (access of protected areas in flash such as protected BSL)

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

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

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

- A POR signal
- Watchdog timer expiration when watchdog mode only (see the WDT_A chapter for details)
- Watchdog timer password violation (see the WDT_A chapter for details)
- A Flash memory password violation (see the Flash Controller chapter for details)
- Power Management Module password violation (see the PMM chapter for details)
- Fetch from peripheral area

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

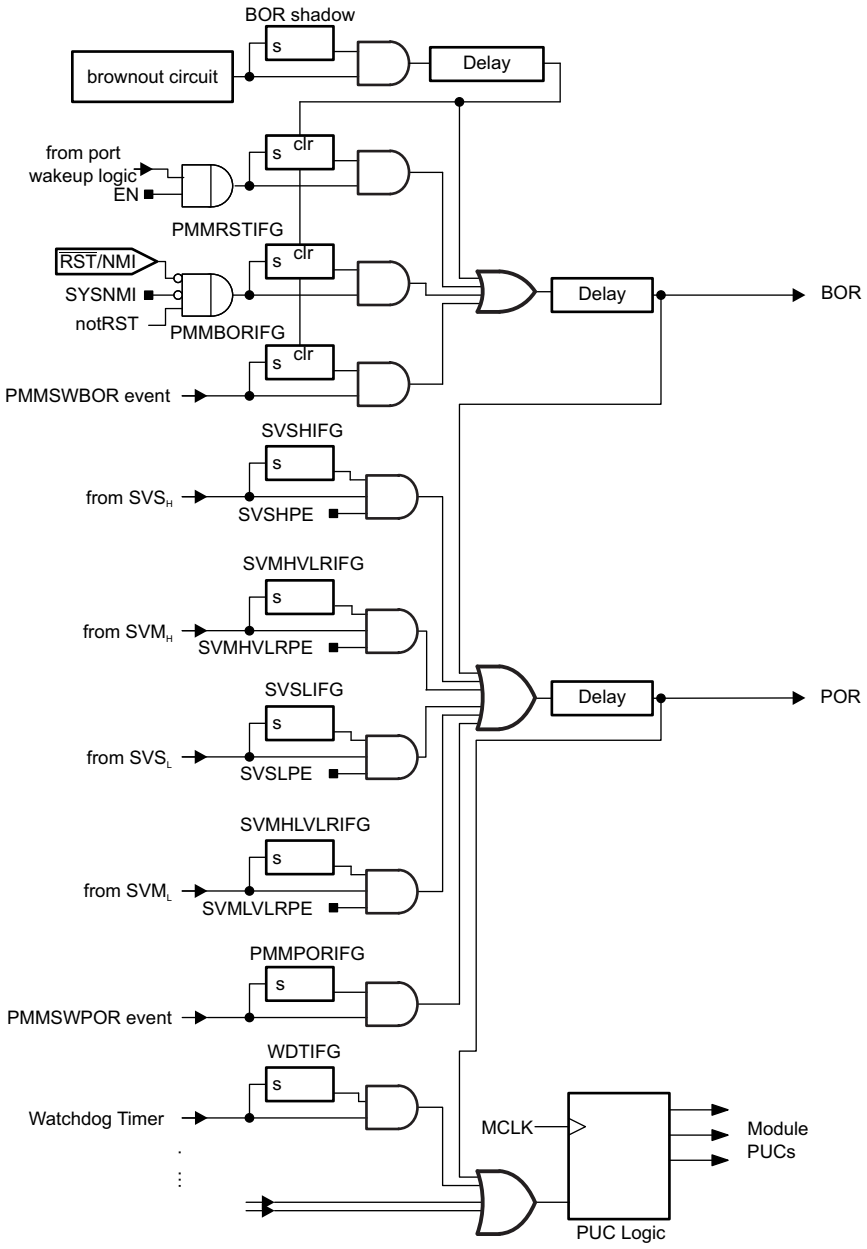

Figure 1-1. BOR/POR/PUC Reset Circuit

1.2.1 Device Initial Conditions After System Reset

After a BOR, the initial device conditions are:

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

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

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

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

NOTE: Some SRAM locations can be modified by the boot code (refer to Section 1.9) after a BOR event. These SRAM locations, when available, are at SRAM locations 01CFAh through 01CFFh and 023FAh through 023FFh.

1.3 Interrupts

The interrupt priorities are fixed and defined by the arrangement of the modules in the connection chain as shown in Figure 1-2. Interrupt priorities determine what interrupt is taken when more than one interrupt is pending simultaneously.

There are three types of interrupts:

- System reset
- (Non)maskable
- Maskable

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

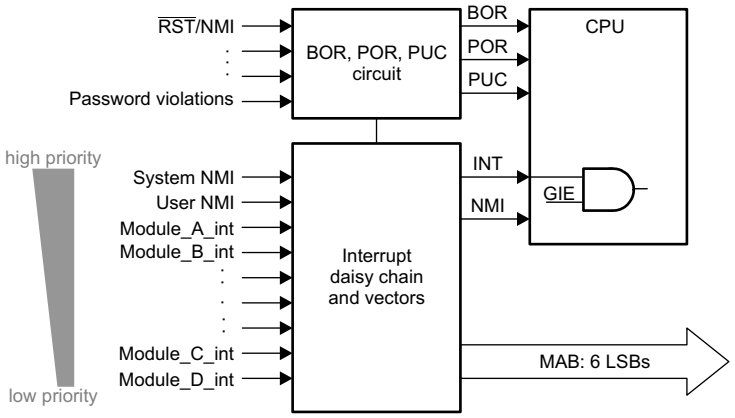

Figure 1-2. Interrupt Priority

1.3.1 (Non)Maskable Interrupts (NMIs)

In general, NMIs are not masked by the general interrupt enable (GIE) bit. The family supports two levels of NMIs — system NMI (SNMI) and user NMI (UNMI). The NMI sources are enabled by individual interrupt enable bits. When an NMI interrupt is accepted, other NMIs of that level are automatically disabled to prevent nesting of consecutive NMIs of the same level. Program execution begins at the address stored in the NMI vector as shown in Table 1-1. To allow software backward compatibility to users of earlier MSP430 families, the software may, but does not need to, reenable NMI sources. The block diagram for NMI sources is shown in Figure 1-3.

A UNMI interrupt can be generated by following sources:

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

A SNMI interrupt can be generated by following sources:

- Power Management Module (PMM) SVM_L/SVM_H supply voltage fault
- PMM high/low side delay expiration
- Vacant memory access
- JTAG mailbox (JMB) event

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

1.3.2 SNMI Timing

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

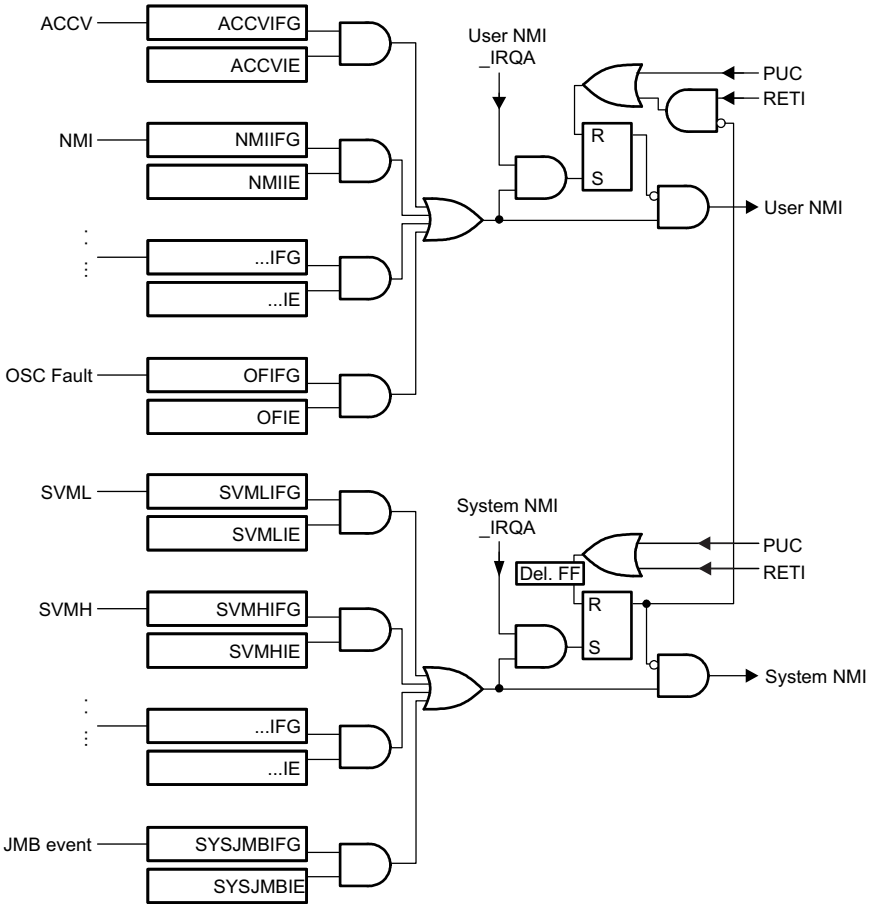

Figure 1-3. NMIs With Reentrance Protection

1.3.3 Maskable Interrupts

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

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

1.3.4 Interrupt Processing

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

1.3.4.1 Interrupt Acceptance

The interrupt latency is six cycles, starting with the acceptance of an interrupt request, and lasting until the start of execution of the first instruction of the interrupt service routine, as shown in Figure 1-4. 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. All bits of SR are cleared except SCG0, thereby terminating any low-power mode. Because the GIE bit is cleared, further interrupts are disabled.
7. The content of the interrupt vector is loaded into the PC; the program continues with the interrupt service routine at that address.

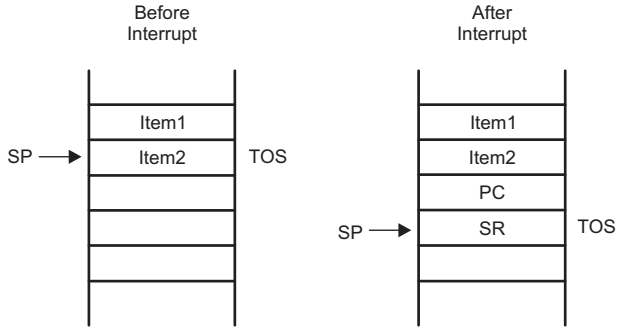

Figure 1-4. Interrupt Processing

NOTE: Enable and Disable Interrupt

Due to the pipelined CPU architecture, setting the general interrupt enable (GIE) requires special care.

- The instruction immediately after the enable interrupts instruction (EINT) is always executed, even if an interrupt service request is pending.
- Include at least one instruction between the clear of an interrupt enable or interrupt flag and the EINT instruction. For example: Insert a NOP instruction in front of the EINT instruction.
- Include at least one instruction between DINT and the start of an code sequence that requires protection from interrupts. For example: Insert a NOP instruction after the DINT.
- Never clear the general interrupt enable (GIE) immediately after setting it. Insert at least one instruction in between such sequence.

The rules above apply to all instructions that set or clear the general interrupt enable bit. Not following these rules might result in unexpected CPU execution.

1.3.4.2 Return From Interrupt

The interrupt handling routine terminates with the instruction:

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

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

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

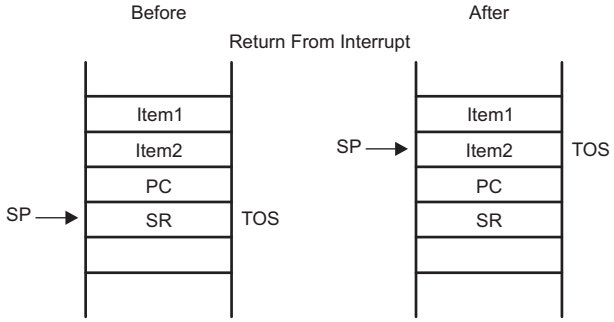

Figure 1-5. Return From Interrupt

1.3.5 Interrupt Nesting

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

1.3.6 Interrupt Vectors

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

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

Interrupt Source	Interrupt Flag	System Interrupt	Word Address	Priority
Reset: power up, external reset watchdog, flash password	... WDTIFG KEYV	... Reset	... 0FFFEh	... Highest
System NMI: PMM		(Non)maskable	0FFFCh	…
User NMI: NMI, oscillator fault, flash memory access violation	... NMIIFG OFIFG ACCVIFG	... (Non)maskable (Non)maskable (Non)maskable	... 0FFFAh	... …
Device specific			0FFF8h	…
...			...	...
Watchdog timer	WDTIFG	Maskable	...	...
...			...	...
Device specific			…	…
Reserved		Maskable	…	Lowest

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

1.3.6.1 Alternate Interrupt Vectors

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

1.3.7 SYS Interrupt Vector Generators

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

1.3.7.1 SYSSNIV Software Example

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

```
SNI_ISR:    ADD      &SYSSNIV,PC ; Add offset to jump table
         RETI             ; Vector 0: No interrupt
         JMP      SVML_ISR       ; Vector 2: SVMLIFG
         JMP      SVMH_ISR       ; Vector 4: SVMHIFG
         JMP      DLYL_ISR       ; Vector 6: SVSMLDLYIFG
         JMP      DLYH_ISR       ; Vector 8: SVSMHDLYIFG
         JMP      VMA_ISR        ; Vector 10: VMAIFG
         JMP      JMBI_ISR       ; Vector 12: JMBINIFG
JMBO_ISR:                        ; Vector 14: JMBOUTIFG
        ...                      ; Task_E starts here
        RETI                     ; Return
SVML_ISR:                        ; Vector 2
        ...                      ; Task_2 starts here
        RETI                     ; Return
SVMH_ISR:                        ; Vector 4
        ...                  ; Task_4 starts here
        RETI                     ; Return
DLYL_ISR:                        ; Vector 6
        ...                      ; Task_6 starts here
        RETI                     ; Return
DLYH_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
```

1.3.7.2 SYSBERRIV Bus Error Interrupt Vector Generator

Some devices, for example those that contain the USB module, include an additional system interrupt vector generator, SYSBERRIV. In general, any type of system related bus error or timeout error is associated with a user NMI event. Upon this event, the SYSUNIV contains an offset value corresponding to a bus error event (BUSIFG). This offset can be added to the PC to automatically jump to the appropriate NMI routine. Similarly, SYSBERRIV also contains an offset value corresponding to which specific event caused the bus error event. The offset value in SYSBERRIV can be added inside the NMI routine to automatically jump to the appropriate routine. In this way, the SYSBERRIV can be thought of as an extension to the user NMI vectors.

1.4 Operating Modes

The MSP430 family is designed for ultra-low-power applications and uses different operating modes shown in Figure 1-6.

The operating modes take into account three different needs:

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

The low-power modes LPM0 through LPM4 are configured with the CPUOFF, OSCOFF, SCG0, and SCG1 bits in the SR. The advantage of including the CPUOFF, OSCOFF, SCG0, and SCG1 mode-control bits in the SR is that the present operating mode is saved onto the stack during an interrupt service routine. Program flow returns to the previous operating mode if the saved SR value is not altered during the interrupt service routine. Program flow can be returned to a different operating mode by manipulating the saved SR value on the stack inside of the interrupt service routine. When setting any of the mode-control bits, the selected operating mode takes effect immediately. Peripherals operating with any disabled clock are disabled until the clock becomes active. Peripherals may also be disabled with their individual control register settings. All I/O port pins and RAM/registers are unchanged. Wakeup from LPM0 through LPM4 is possible through all enabled interrupts.

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

NOTE: LPM3.5 and LPM4.5 low power modes are not available on all devices. See the device specific data sheet to see which LPMx.5 power modes are available.

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

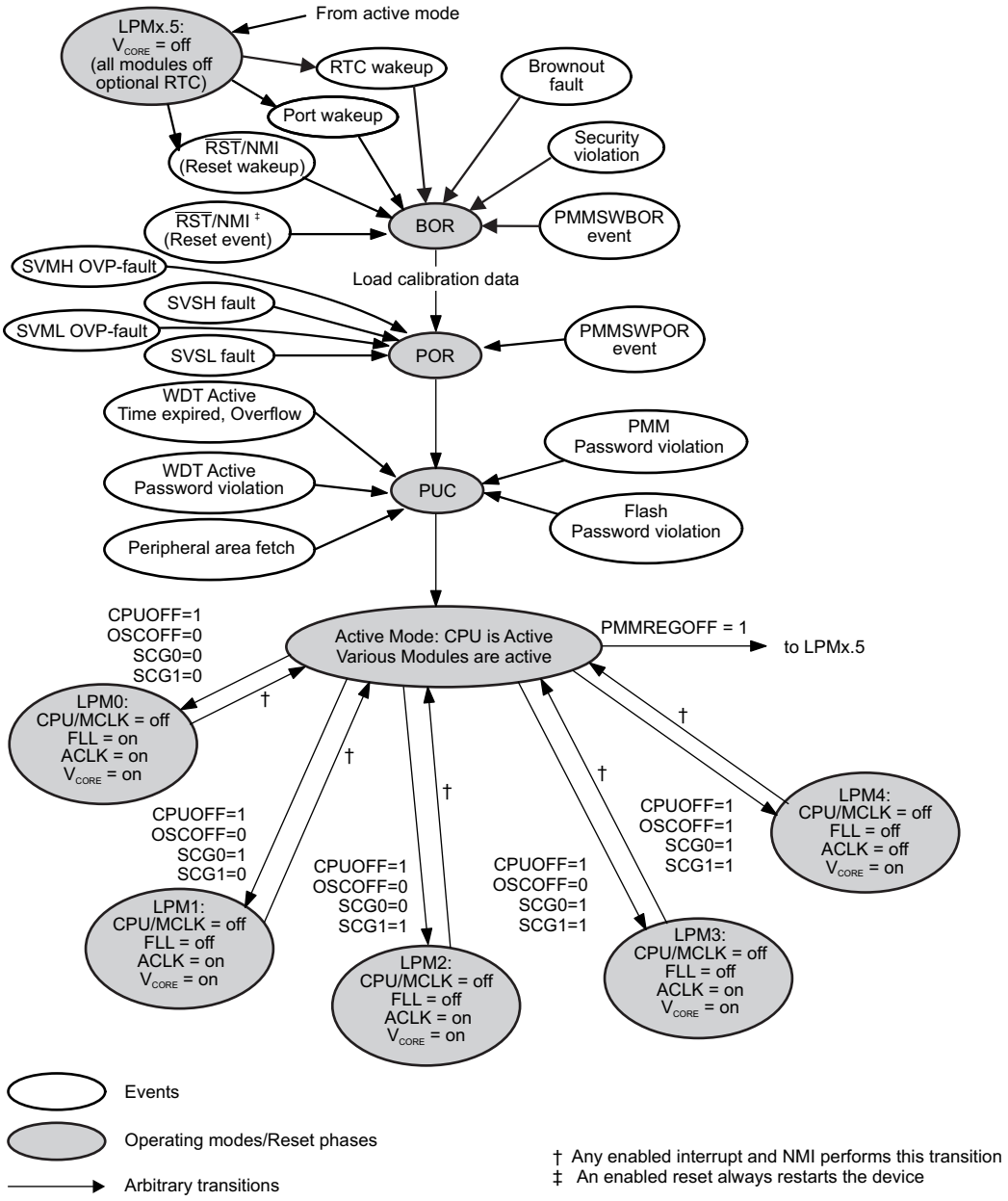

Figure 1-6. Operation Modes

Table 1-2. Operation Modes

SCG1[1]	SCG0	OSCOFF[1]	CPUOFF[1]	Mode	CPU and Clocks Status[2]
0	0	0	0	Active	CPU, MCLK are active. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK, MCLK, or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0). FLL is enabled if DCO is enabled.
0	0	0	1	LPM0	CPU, MCLK are disabled. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0). FLL is enabled if DCO is enabled.
0	1	0	1	LPM1	CPU, MCLK are disabled. ACLK is active. SMCLK optionally active (SMCLKOFF = 0). DCO is enabled if sources ACLK or SMCLK (SMCLKOFF = 0). DCO bias is enabled if DCO is enabled or DCO sources MCLK or SMCLK (SMCLKOFF = 0). FLL is disabled.
1	0	0	1	LPM2	CPU, MCLK are disabled. ACLK is active. SMCLK is disabled. DCO is enabled if sources ACLK. FLL is disabled.
1	1	0	1	LPM3	CPU, MCLK are disabled. ACLK is active. SMCLK is disabled. DCO is enabled if sources ACLK. FLL is disabled.
1	1	1	1	LPM4	CPU and all clocks are disabled.
1	1	1	1	LPM3.5[3]	When PMMREGOFF = 1, regulator is disabled. No memory retention. In this mode, RTC operation is possible when configured properly. See the *RTC* module for further details.
1	1	1	1	LPM4.5[3]	When PMMREGOFF = 1, regulator is disabled. No memory retention. In this mode, all clock sources are disabled; that is, no RTC operation is possible.

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

(2) The low-power modes and, hence, the system clocks can be affected by the clock request system. See the UCS chapter for details.

(3) LPM3.5 and LPM4.5 modes are not available on all devices. See the device-specific data sheet for availability.

1.4.1 Entering and Exiting Low-Power Modes LPM0 Through LPM4

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

- Enter interrupt service routine
  - The PC and SR are stored on the stack.
  - The CPUOFF, SCG1, and OSCOFF bits are automatically reset.
- Options for returning from the interrupt service routine
  - The original SR is popped from the stack, restoring the previous operating mode.
  - The SR bits stored on the stack can be modified within the interrupt service routine returning to a different operating mode when the RETI instruction is executed.

Example 1-1 shows assembly code examples of entering and exiting low-power modes. Example 1-2 shows C code examples of entering and exiting low-power modes.

Example 1-1. Examples of Entering and Exiting LPM in Assembly

```
; 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
```

Example 1-2. Examples of Entering and Exiting LPM in C

```
// Enter LPM0 Example
__bis_SR_register(LPM0_bits + GIE);     // Enter LPM0 with interrupts enabled

// Exit LPM0 Interrupt Service Routine
__bic_SR_register_on_exit (LPM0_bits);  // Exit LPM0

// Enter LPM1 Example
__bis_SR_register(LPM1_bits + GIE);     // Enter LPM1 with interrupts enabled

// Exit LPM1 Interrupt Service Routine
__bic_SR_register_on_exit (LPM1_bits);  // Exit LPM1

// Enter LPM2 Example
__bis_SR_register(LPM2_bits + GIE);     // Enter LPM2 with interrupts enabled

// Exit LPM2 Interrupt Service Routine
__bic_SR_register_on_exit (LPM2_bits);  // Exit LPM2

// Enter LPM3 Example
__bis_SR_register(LPM3_bits + GIE);     // Enter LPM3 with interrupts enabled

// Exit LPM3 Interrupt Service Routine
__bic_SR_register_on_exit (LPM3_bits);  // Exit LPM3

// Enter LPM4 Example
__bis_SR_register(LPM4_bits + GIE);     // Enter LPM4 with interrupts enabled

// Exit LPM4 Interrupt Service Routine
__bic_SR_register_on_exit (LPM4_bits);  // Exit LPM4
```

1.4.2 Entering and Exiting Low-Power Modes LPMx.5

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

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

There are two LPMx.5 power modes, LPM3.5 and LPM4.5. Not all of these are available on all devices. See the device specific data sheet to see which LPMx.5 power modes are available. LPM4.5 allows for the lowest power consumption available. No clock sources are active during LPM4.5. LPM3.5 is similar to LPM4.5, but has the additional capability of having a RTC mode available. In addition to the wake-up events possible in LPM4.5, RTC wake-up events are also possible in LPM3.5.

The program flow for entering LPMx.5 is:

1. Configure I/O appropriately. See the Digital I/O chapter for complete details on configuring I/O for LPMx.5.
   - Set all ports to general purpose I/O. Configure each port to ensure no floating inputs based on the application requirements.
   - If wakeup from I/O is desired, configure input ports with interrupt capability appropriately.
2. If LPM3.5 is available, and desired, enable RTC operation. In addition, configure any RTC interrupts, if desired for LPM3.5 wake-up event. See the RTC Overview chapter for complete details.
3. Ensure clock system settings allow LPMx.5 entry according to in UCS chapter.
4. Enter LPMx.5 by setting PMMREGOFF = 1 and LPM4 status register bits. The following code example shows how to enter LPMx.5 mode. See the PMM chapter for further details.

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

NOTE: It is not possible to wake up from LPMx.5 if its respective interrupt flag is already asserted. TI recommends clearing the respective flag before entering LPMx.5. TI also recommends setting GIE = 1 before entry into LPMx.5. Any pending flags in this case could then be serviced before LPMx.5 entry.

Although TI recommends setting GIE = 1 before entering LPMx.5, it is not required. Device wakeup from LPMx.5 with an enabled wake-up function will still cause the device to wake up from LPMx.5 even with GIE = 0. If GIE = 0 before LPMx.5, additional care may be required. Should the respective interrupt event should occur during LPMx.5 entry, the device may not recognize this or any future interrupt wake-up event on this function.

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

- Enter system reset service routine
  - Reconfigure system as required for the application.
  - Reconfigure I/O as required for the application.

1.4.3 Extended Time in Low-Power Modes

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

```
; Enter LPM4 Example with lowest DCO Setting
   BIC   #SCG0, SR                          ; Disable FLL
   MOV   #0100h, &UCSCTL0                   ; Set DCO tap to first tap, clear
modulation.
   BIC   #DCORSEL2+DCORSEL1+DCORSEL0,&UCSCTL1 ; Lowest DCORSEL
   BIS   #GIE+CPUOFF+OSCOFF+SCG1+SCG0,SR    ; Enter LPM4
;  ...                                      ; Program stops
;
; Interrupt Service Routine
   BIC   #CPUOFF+OSCOFF+SCG1+SCG0,0(SR)     ; Exit LPM4 on RETI
   RETI
```

1.5 Principles for Low-Power Applications

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

- Use interrupts to wake the processor and control program flow.
- Peripherals should be switched on only when needed.
- Use low-power integrated peripheral modules in place of software driven functions. For example, Timer_A and Timer_B can automatically generate PWM and capture external timing with no CPU resources.
- Calculated branching and fast table look-ups should be used in place of flag polling and long software calculations.
- Avoid frequent subroutine and function calls due to overhead.
- For longer software routines, single-cycle CPU registers should be used.
- Overwrite RAM control register RCCTL0 with all not available and unused segments set to powered down (= 1). For information about used RAM segments see the device-specific data sheet.

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

1.6 Connection of Unused Pins

Table 1-3 lists the correct termination of all unused pins.

Table 1-3. Connection of Unused Pins(1)

Pin	Potential	Comment
AVCC	DV_{CC}	
AVSS	DV_{SS}	
CPCAP	Open	For devices where charge pump in not used (no rail-to-rail OA and no rail-to-rail CTSD16).
LCDCAP	DV_{SS}	
LDOI	DV_{SS}	For devices with LDO-PWR module when not being used in the application.
LDOO	Open	For devices with LDO-PWR module when not being used in the application.
PJ.0/TDO PJ.1/TDI PJ.2/TMS PJ.3/TCK	Open	The JTAG pins are shared with general purpose I/O function (PJ.x). If not being used, these should be switched to port function, output direction (PJDIR.n = 1). When used as JTAG pins, these pins should remain open.
PU.0/DP PU.1/DM	Open	For USB devices only when USB module is not being used in the application
PUR(2)	DV_{SS}	For USB devices only when USB module is not being used in the application
Px.y	Open	Switched to port function, output direction (PxDIR.n = 1). Px.y represents port x and bit y of port x (for example, P1.0, P1.1, P2.2, PJ.0, PJ.1)
$\overline{RST}$/NMI	DV_{CC} or V_{CC}	47-kΩ pullup or internal pullup selected with 10-nF (2.2 nF) pulldown(3)
TEST	Open	This pin always has an internal pulldown enabled.
V18	Open	For USB devices only when USB module is not being used in the application
VBAK	Open	For devices where no separate battery backup supply in the system. Set bit BAKDIS = 1.
VBAT	DV_{CC}	For devices where no separate battery backup supply in the system. Set bit BAKDIS = 1.
VBUS, VSSU	DV_{SS}	For USB devices only when USB module is not being used in the application
VUSB	Open	For USB devices only when USB module is not being used in the application
XIN	DV_{SS}	For dedicated XIN pins only. XIN pins with shared GPIO functions should be programmed to GPIO and follow Px.y recommendations.
XOUT	Open	For dedicated XOUT pins only. XOUT pins with shared GPIO functions should be programmed to GPIO and follow Px.y recommendations.
XT2IN	DV_{SS}	For dedicated XT2IN pins only. XT2IN pins with shared GPIO functions should be programmed to GPIO and follow Px.y recommendations.
XT2OUT	Open	For dedicated XT2OUT pins only. XT2OUT pins with shared GPIO functions should be programmed to GPIO and follow Px.y recommendations.

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

(2) The default USB BSL evaluates the state of the PUR pin after a BOR reset. If it is pulled high externally, then the BSL is invoked. Therefore, unless invoking the BSL, it is important to keep PUR pulled low after a BOR reset, even if BSL or USB is never used. A 1-MΩ resistor to ground is recommended.

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

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

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

The $\overline{\text{RST}}$/NMI pin can have either a pullup or pulldown present or not. SYSRSTUP selects either pullup or pulldown and SYSRSTRE causes the pullup or pulldown to be enabled or not. If the $\overline{\text{RST}}$/NMI pin is unused, it is required to have either the internal pullup selected and enabled or an external resistor connected to the $\overline{\text{RST}}$/NMI pin as shown in Table 1-3.

NOTE: All devices except the MSP430F543x (non-A devices) have the internal pullup enabled. In this case, no external pullup resistor is required.

1.8 Configuring JTAG Pins

The JTAG pins are shared with general-purpose I/O pins. After a BOR, the SYSJTAGPIN bit in the SYSCTL register is cleared. With SYSJTAGPIN cleared, the pins with JTAG functionality are configured as general-purpose I/O. In this case, only a special sequences on the TEST and RST/NMI pins enables the JTAG functionality. As long as the TEST pin is pulled to DVCC, the pins remain in their JTAG functionality. If the TEST pin is released to DVSS, the shared JTAG pins revert to general-purpose I/Os.

If SYSJTAGPIN = 1, the JTAG pins are permanently configured to 4-wire JTAG mode and remain in this mode until another BOR condition occurs. Use this feature early in the software if the MSP430 device is part of a JTAG chain. Note that this also disables the Spy-Bi-Wire mode.

The SYSJTAGPIN is a write only once function. Clearing it by software is not possible.

1.9 Boot Code

The boot code is always executed after a BOR. The boot code loads factory stored calibration values of the oscillator and reference voltages. In addition, it checks for the presence of a user-defined boot strap loader (BSL).

1.10 Bootloader (BSL)

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

On devices without USB, a basic BSL program is provided by TI. This supports the commonly used UART protocol with RS232 interfacing, allowing flexible use of both hardware and software. To use the BSL, a specific BSL entry sequence must be applied to specific device pins. The correct entry sequence causes SYSBSLIND to be set. An added sequence of commands initiates the desired function. A boot-loading session can be exited by continuing operation at a defined user program address or by applying the standard reset sequence. Access to the device memory by the BSL is protected against misuse by a user-defined password. Devices with USB have a USB based BSL program provided by TI. For more details, see the *MSP430 Programming With the Bootloader (BSL)*.

The amount of BSL memory that is available is device specific. The BSL memory size is organized into segments and can be set using the SYSBSLSIZE bits. See the device specific data sheet for the number and size of the segments available. It is possible to assign a small amount of RAM to the allocated BSL memory. Setting SYSBSLR allocates the lowest 16 bytes of RAM for the BSL. When the BSL memory is protected, access to these RAM locations is only possible from within the protected BSL memory segments.

It may be desirable in some BSL applications to only allow changing of the Power Management Module settings from the protected BSL segments. This is possible with the SYSPMMPE bit. Normally, this bit is cleared and allows access of the PMM control registers from any memory location. Setting SYSPMMPE, allows access to the PMM control registers only from the protected BSL memory. Once set, SYSPMMPE can only be cleared by a BOR event.

1.11 Memory Map – Uses and Abilities

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

Can generate NMI on read/write/fetch								
Generates PUC on fetch access								
Protectable for read/write accesses								
Always able to access PMM registers from[1]; Mass erase by user possible								
Mass erase by user possible								
Bank erase by user possible								
Segment erase by user possible								
Address Range	**Name and Usage**	Properties						
00000h-00FFFh	Peripherals with gaps							
00000h-000FFh	Reserved for system extension							
00100h-00FEFh	Peripherals						x	
00FF0h-00FF3h	Descriptor type[2]						x	
00FF4h-00FF7h	Start address of descriptor structure						x	
01000h-011FFh	BSL 0	x				x		
01200h-013FFh	BSL 1	x				x		
01400h-015FFh	BSL 2	x				x		
01600h-017FFh	BSL 3	x			x	x		
017FCh-017FFh	BSL Signature Location							
01800h-0187Fh	Info D	x						
01880h-018FFh	Info C	x						
01900h-0197Fh	Info B	x						
01980h-019FFh	Info A	x						
01A00h-01A7Fh	Device Descriptor Table						x	
01C00h-05BFFh	RAM 16KB							
05B80-05BFFh	Alternate Interrupt Vectors							
05C00h-0FFFFh	Program	x	x[1]	x				
0FF80h-0FFFFh	Interrupt Vectors							
10000h-45BFFh	Program	x	x	x				
45C00h-FFFFFh	Vacant							x[3]

(1) Access rights are separately programmable for SYS and PMM.
(2) Fixed ID for all MSP430 devices. See Section 1.13.1 for further details.
(3) On vacant memory space, the value 03FFFh is driven on the data bus.

1.11.1 Vacant Memory Space

Vacant memory is non-existent memory space. Accesses to vacant memory space generate a system (non)maskable interrupt (SNMI) when enabled (VMAIE = 1). Reads from vacant memory results in the value 3FFFh. In the case of a fetch, this is taken as JMP $. Fetch accesses from vacant peripheral space result in a PUC. After the boot code is executed, it behaves like vacant memory space and also causes an NMI on access.

1.11.2 JTAG Lock Mechanism Using the Electronic Fuse

A device can be protected from unauthorized access by disabling the JTAG and SBW interface. This is achieved by programming the electronic fuse. Programming the electronic fuse, completely disables the debug and access capabilities associated with the JTAG and Spy-Bi-Wire interface. The JTAG is locked by programming a certain signature into the device flash memory at dedicated addresses. The JTAG security lock key resides at the end of the bootloader (BSL) memory at addresses 17FCh through 17FFh. Anything other than 0h or FFFFFFFFh programmed to these addresses locks the JTAG interface.

All of the 5xx MSP430 devices come with a preprogrammed BSL (TI-BSL) code that, by default, protects itself from unintended erase and write access. This is done by setting SYSBSLPE in the SYSBSLC register. Since the JTAG security lock key resides in the BSL memory address range, appropriate action must be taken to unprotect the BSL memory area before programming the protection key. For more details on the electronic fuse, see the *MSP430 Programming Via the JTAG Interface User's Guide* (SLAU320).

Some JTAG commands are still possible after the device is secured, including the BYPASS command (see IEEE1149-2001 Standard) and the JMB_EXCHANGE command which allows access to the JTAG Mailbox System (see Section 1.12 for details).

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

1.12 JTAG Mailbox (JMB) System

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

Applications of the JMB are:

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

1.12.1 JMB Configuration

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

1.12.2 JMBOUT0 and JMBOUT1 Outgoing Mailbox

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

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

1.12.3 JMBIN0 and JMBIN1 Incoming Mailbox

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

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

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

1.12.4 JMB NMI Usage

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

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

1.13 Device Descriptor Table

Each device provides a data structure in memory that allows an unambiguous identification of the device, as well as, a more detailed description of the available modules on a given device. SYS provides this information and can be used by device-adaptive SW tools and libraries to clearly identify a particular device and all modules and capabilities contained within it. The validity of the device descriptor can be verified by cyclic redundancy check (CRC). Figure 1-7 shows the logical order and structure of the device descriptor table. The complete device descriptor table and its contents can be found in the device specific data sheet.

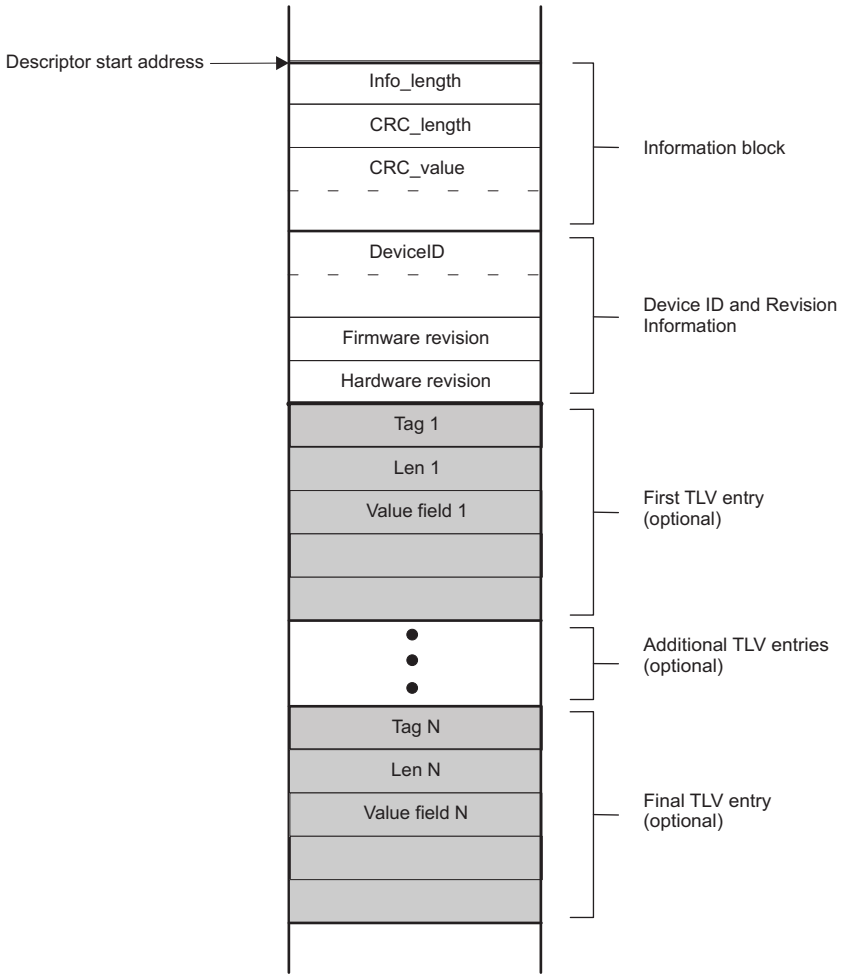

Figure 1-7. Devices Descriptor Table

1.13.1 Identifying Device Type

The value read at address location 00FF0h identifies the family branch of the device. All values starting with 80h indicate a hierarchical structure consisting of the information block and a TLV tag-length-value (TLV) structure containing the various descriptors. Any other value than 80h read at address location 00FF0h indicates the device is of an older family and contains a flat descriptor beginning at location 0FF0h. The information block, shown in Figure 1-7 contains the device ID, die revisions, firmware revisions, and other manufacturer and tool related information. The descriptors contains information about the available peripherals, their subtypes and addresses and provides the information required to build adaptive hardware drivers for operating systems.

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

$$\text{Length} = 2^{\text{Info\_length}} \text{ in 32-bit words} \quad (1)$$

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

1.13.2 TLV Descriptors

The TLV descriptors follow the information block. Because the information block is always a fixed length, the start location of the TLV descriptors is fixed for a given device family. For the MSP430x5xx family, this location is 01A08h. See the device-specific data sheet for the complete TLV structure and what descriptors are available.

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

Each TLV descriptor contains a tag field which identifies the descriptor type. Table 1-4 shows the currently supported tags.

Table 1-4. Tag Values

Short Name	Value	Description
LDTAG	01h	Legacy descriptor (1xx, 2xx, 4xx families)
PDTAG	02h	Peripheral discovery descriptor
Reserved	03h	Future use
Reserved	04h	Future use
BLANK	05h	Blank descriptor
Reserved	06h	Future use
ADC12CAL	11h	ADC12 calibration
REFCAL	12h	REF calibration
ADC10CAL	13h	ADC10 calibration
Reserved	14h-1Ch	Future use
CTSD16CAL	1Dh	CTSD16 calibration
Reserved	1Eh-FDh	Future use
TAGEXT	FEh	Tag extender

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

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

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

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

1.13.3 Peripheral Discovery Descriptor

This descriptor type can describe concatenated or distributed memory or peripheral mappings, as well as, the number of interrupt vectors and their order. The peripheral discovery descriptor has tag value 02h (PDTAG). Table 1-5 shows the structure of the peripheral discovery descriptor.

NOTE: Peripheral Discovery Descriptor is not available in every device. See the Device Descriptors section in the device-specific data sheet for the availability and details on Peripheral Discovery Descriptor.

Table 1-5. Peripheral Discovery Descriptor

Element	Size (bytes)	Comments
Memory entry 1	2	Optional
Memory entry 2	2	Optional
...	2	Optional
Delimiter (00h)	1	Mandatory
Peripheral count	1	Mandatory
Peripheral entry 1	2	Optional
Peripheral entry 2	2	Optional
...	2	Optional
Interrupt priority N-3	1	Optional
Interrupt priority N-4	1	Optional
...	1	Optional
Delimiter (00h)	1	Mandatory

The structures for a memory entry and peripheral entry are shown below. A memory entry consists of two bytes (one word). Table 1-6 shows the individual bit fields of a memory entry word and their respective meanings. Similarly, a peripheral entry consists of two bytes (one word). Table 1-7 shows the individual bit fields of a peripheral entry word and their respective meanings.

Table 1-6. Values for Memory Entry

Bit Fields				
[15:13]	[12:9]	[8]	[7]	[6:0]
Memory Type	Size	More	Unit Size	Address Value
000: None	0000: 0 B	0: End Entry	0: 0200h	0000000
001: RAM	0001: 128 B	1: More Entries	1: 010000h	0000001
010: EEPROM	0010: 256 B			0000010
011: Reserved	0011: 512 B			0000011
100: FLASH	0100: 1KB			0000100
101: ROM	0101: 2KB			0000101
110: MemType appended	0110: 4KB			0000110
111: Undefined	0111: 8KB			0000111
	1000: 16KB			0001000
	1001: 32KB			0001001
	1010: 64KB			0001010
	1011: 128KB			0001011
	1100: 256KB			0001100
	1101: 512KB			...
	1110: Size appended			...
	1111: Undefined			1111111

Table 1-7. Values for Peripheral Entry

Bit Fields		
[15:8]	[7]	[6:0]
Peripheral ID (PID)[(1)]	UnitSize	AdrVal
Any PID	0: 010h	0000000
Any PID	1: 0800h	0000001
Any PID		0000010
Any PID		0000011
Any PID		0000100
Any PID		0000101
Any PID		...
Any PID		...
Any PID		1111111

(1) The Peripheral IDs are listed in Table 1-8. This is not a complete list, but shown as an example.

Table 1-8. Peripheral IDs(1)

Peripheral or Module	PID
No Module	00h
WDT	01h
SFR	02h
UCS	03h
SYS	04h
PMM	05h
Flash Controller	08h
CRC16	09h
Port 1, 2	51h
Port 3, 4	52h
Port 5, 6	53h
Port 7, 8	54h
Port 9, 10	55h
Port J	5Fh
Timer A0	81h
Timer A1	82h
Special info appended	FEh
Undefined module	FFh

(1) This table is not a complete list of all peripheral IDs that might be available on a device, and is shown here for illustrative purposes only.

Table 1-9 shows a simple example for a peripheral discovery descriptor of a hypothetical device:

Table 1-9. Sample Peripheral Discovery Descriptor

Hex	Binary	Entry Type	Description
030h, 0Eh	001_1000_ 0_0_0001110	memory	RAM 16KB, Start address = 01C00h (0Eh × 0200h)(1)
09Bh, 02Eh	100_1011_0_0_0101110	memory	Flash 128KB, Start address = 05C00h (2Eh × 0200h)
00h	0000_0000_0000_0000	delimiter	No more memory entries
0Fh	0000_1111	peripheral count	Peripheral count = 15
02h, 10h	00000010_0_0010000	peripheral	SFR at address = 0100h (10h × 10h)
01h, 01h	00000001_0_0000001	peripheral	WDT at address = 0110h (0100h + 10h)
05h, 01h	00000101_0_0000001	peripheral	PMM at address = 0120h (0110h + 10h)
03h, 01h	00000011_0_0000001	peripheral	UCS at address = 0130h (0120h + 10h)
08h, 01h	00001000_0_0000001	peripheral	FLCTL at address = 0140h (0130h + 10h)
09h, 01h	00001001_0_0000001	peripheral	CRC16 at address = 0150h (0140h + 10h)
04h, 01h	00000100_0_0000001	peripheral	SYS at address = 0160h (0150h + 10h)
51h, 0Ah	01010001_0_0001010	peripheral	Port 1, 2 at address = 0200h (0160h + 10h × 10h)
52h, 02h	01010010_0_0000010	peripheral	Port 3, 4 at address = 0220h (0200h + 02h × 10h)
53h, 02h	01010011_0_0000010	peripheral	Port 5, 6 at address = 0240h (0220h + 02h × 10h)
54h, 02h	01010100_0_0000010	peripheral	Port 7, 8 at address = 0260h (0240h + 02h × 10h)
55h, 02h	01010101_0_0000010	peripheral	Port 9, 10 at address = 0280h (0260h + 02h × 10h)
5Fh, 0Ah	01011111_0_0001010	peripheral	Port J at address = 0320h (0280h + 0Ah × 10h)
81h, 02h	10000001_0_0000010	peripheral	Timer A0 at address = 0340h (0320h + 02h × 10h)
82h, 04h	10000010_0_0000100	peripheral	Timer A1 at address = 0380h (0340h + 04h × 10h)
–			No appended entries
			SYSRSTIV at 0FFFEh (implied)
			SYSSNIV at 0FFFCh (implied)
			SYSUNIV at 0FFFAh (implied)
81h	1000_0001	interrupt	TA0 CCR0 at 0FFF8h
81h	1000_0001	interrupt	TA0 CCR1, CCR1, TA0IFG at 0FFF6h
51h	0101_0001	interrupt	Port 1 at 0FFF4h
82h	1000_0010	interrupt	TA1CCR0 at 0FFF2h
51h	0101_0001	interrupt	Port 2 at 0FFF0h
81h	1000_0010	interrupt	TA1 CCR1, CCR1, TA1IFG at 0FFEEh
00h	0000_0000	delimiter	No more interrupt entries

(1) In this example, the memory type is RAM (bits[15:13] = 001b), the size is 16KB (bits[12:9] = 1000b), and the starting address is 01C00h. The starting address is computed by taking the size field indicated by bit[7] (in this case, 0200h) and multiplying it by the address value (bits[6:0] = 0001110b. In this case, 0200h × 00Eh = 01C00h.

NOTE: The interrupt ordering has some implied rules:

- For timers, CCR0 interrupt has higher priority over all other CCRn interrupts.
- For communication ports, RX has higher priority over TX
- For port pairs, Port 1 has higher priority than Port 2, Port 3 has higher priority than Port 4, and so on.

1.13.4 CRC Computation

The CRC checksum for the TLV structure is stored at memory locations 0x1A02 and 0x1A03. The least significant byte (LSB) and most significant byte (MSB) reside at memory locations 0x1A02 and 0x1A03, respectively. The checksum is computed using data stored at memory locations 0x1A04 through 0x1AFF. The CRC checksum can be easily computed using the CRC16 module. The following simplified C code utilizes the CRC16 module to compute the checksum. See the CRC16 chapter for further details on the CRC algorithm implementation.

NOTE: The CRC module on the MSP430F543x and MSP430F541x non-A versions does not support the bit-wise reverse feature used in this code example. Registers CRCDIRB and CRCRESR, along with their respective functionality, are not available.

```
unsigned int i;
unsigned char CRCRESULT_LSB, CRCRESULT_MSB;

WDTCTL = WDTPW + WDTHOLD;
CRCINIRES = 0xFFFF;
for (i = 0x01A04; i <= 0x01AFF; i++){
  CRCDIRB_L = *(unsigned char*)(i);
}

CRCRESULT_LSB = CRCINIRES_L;  // value stored at 0x1A02
CRCRESULT_MSB = CRCINIRES_H;  // value stored at 0x1A03
```

1.13.5 Calibration Values

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

1.13.5.1 REF Calibration

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

$$CAL\_ADC\_15VREF\_FACTOR = \frac{V_{REF+}}{1.5V} \times 2^{15}$$
$$CAL\_ADC\_20VREF\_FACTOR = \frac{V_{REF+}}{2.0V} \times 2^{15}$$
$$CAL\_ADC\_25VREF\_FACTOR = \frac{V_{REF+}}{2.5V} \times 2^{15} \quad (2)$$

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

$$ADC(corrected) = ADC(raw) \times CAL\_ADC15VREF\_FACTOR \times \frac{1}{2^{15}}$$
$$ADC(corrected) = ADC(raw) \times CAL\_ADC20VREF\_FACTOR \times \frac{1}{2^{15}}$$
$$ADC(corrected) = ADC(raw) \times CAL\_ADC25VREF\_FACTOR \times \frac{1}{2^{15}} \quad (3)$$

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

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

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

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

1.13.5.2 ADC and CTSD16 Offset and Gain Calibration

The offset of the ADC (ADC10, ADC12, or CTSD16) is determined and stored as a twos-complement number in the TLV structure. The offset error correction is done by adding the CAL_ADC_OFFSET to the conversion result.

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

The gain of the ADC is calculated by Equation 5:

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

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

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

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

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

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

1.13.5.3 Temperature Sensor Calibration for Devices With ADCx

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

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

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

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

$$Temp = \left(ADC(raw) - CAL\_ADC\_T30\right) \times \left(\frac{85 - 30}{CAL\_ADC\_T85 - CAL\_ADC\_T30}\right) + 30 \quad (9)$$

1.13.6 Temperature Sensor Calibration for Devices With CTSD16

The temperature sensor is calibrated using the internal V_{REFBG} voltage reference. A value for two temperatures, 30°C ±3°C and 85°C ±3°C, is stored in the TLV structure. The characteristic equation of the temperature sensor voltage, in mV is:

$$V_{SENSE} = TC_{SENSOR} \times Temp + V_{SENSOR} \quad (10)$$

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

The temperature (Temp, °C) can be computed as follows:

$$Temp = \left(ADC(raw) - CAL\_ADC\_T30\right) \times \left(\frac{85 - 30}{CAL\_ADC\_T85 - CAL\_ADC\_T30}\right) + 30 \quad (11)$$

1.14 SFR Registers

The SFRs are listed in Table 1-11. The base address for the SFRs is listed in Table 1-10. Many of the bits inside the SFRs are described in other chapters throughout this user's guide. These bits are marked with a note and a reference. See the specific chapter of the respective module for details.

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

Table 1-10. SFR Base Address

Module	Base Address
SFR	00100h

Table 1-11. SFR Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	SFRIE1	Interrupt Enable	Read/write	Word	0000h	Section 1.14.1
00h	SFRIE1_L (IE1)		Read/write	Byte	00h	
01h	SFRIE1_H (IE2)		Read/write	Byte	00h	
02h	SFRIFG1	Interrupt Flag	Read/write	Word	0082h	Section 1.14.2
02h	SFRIFG1_L (IFG1)		Read/write	Byte	82h	
03h	SFRIFG1_H (IFG2)		Read/write	Byte	00h	
04h	SFRRPCR	Reset Pin Control	Read/write	Word	0000h	Section 1.14.3
04h	SFRRPCR_L		Read/write	Byte	00h	
05h	SFRRPCR_H		Read/write	Byte	00h	

1.14.1 SFRIE1 Register

Interrupt Enable Register

Figure 1-8. SFRIE1 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
JMBOUTIE	JMBINIE	ACCVIE(1)	NMIIE	VMAIE	Reserved	OFIE(2)	WDTIE(3)
rw-0	rw-0	rw-0	rw-0	rw-0	r0	rw-0	rw-0

(1) See the Flash Controller chapter for details.
(2) See the UCS chapter for details.
(3) See the WDT_A chapter for details.

Table 1-12. SFRIE1 Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads as 0.
7	JMBOUTIE	RW	0h	JTAG mailbox output interrupt enable flag 0b = Interrupts disabled 1b = Interrupts enabled
6	JMBINIE	RW	0h	JTAG mailbox input interrupt enable flag 0b = Interrupts disabled 1b = Interrupts enabled
5	ACCVIE	RW	0h	Flash controller access violation interrupt enable flag 0b = Interrupts disabled 1b = Interrupts enabled
4	NMIIE	RW	0h	NMI pin interrupt enable flag 0b = Interrupts disabled 1b = Interrupts enabled
3	VMAIE	RW	0h	Vacant memory access interrupt enable flag 0b = Interrupts disabled 1b = Interrupts enabled
2	Reserved	R	0h	Reserved. Always reads as 0.
1	OFIE	RW	0h	Oscillator fault interrupt enable flag 0b = Interrupts disabled 1b = Interrupts enabled
0	WDTIE	RW	0h	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 0b = Interrupts disabled 1b = Interrupts enabled

1.14.2 SFRIFG1 Register

Interrupt Flag Register

Figure 1-9. SFRIFG1 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
JMBOUTIFG	JMBINIFG	Reserved	NMIIFG	VMAIFG	Reserved	OFIFG[1]	WDTIFG[2]
rw-(1)	rw-(0)	r0	rw-0	rw-0	r0	rw-(1)	rw-0

(1) See the UCS chapter for details.
(2) See the WDT_A chapter for details.

Table 1-13. SFRIFG1 Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads as 0.
7	JMBOUTIFG	RW	1h	JTAG mailbox output interrupt flag 0b = No interrupt pending. When in 16-bit mode (JMBMODE = 0), this bit is cleared automatically when JMBO0 has been written with a new message to the JTAG module by the CPU. When in 32-bit mode (JMBMODE = 1), this bit is cleared automatically when both JMBO0 and JMBO1 have been written with new messages to the JTAG module by the CPU. This bit is also cleared when the associated vector in SYSUNIV has been read. 1b = Interrupt pending, JMBO registers are ready for new messages. In 16-bit mode (JMBMODE = 0), JMBO0 has been received by the JTAG module and is ready for a new message from the CPU. In 32-bit mode (JMBMODE = 1) , JMBO0 and JMBO1 have been received by the JTAG module and are ready for new messages from the CPU.
6	JMBINIFG	RW	0h	JTAG mailbox input interrupt flag 0b = No interrupt pending. When in 16-bit mode (JMBMODE = 0), this bit is cleared automatically when JMBI0 is read by the CPU. When in 32-bit mode (JMBMODE = 1), this bit is cleared automatically when both JMBI0 and JMBI1 have been read by the CPU. This bit is also cleared when the associated vector in SYSUNIV has been read 1b = Interrupt pending, a message is waiting in the JMBIN registers. In 16-bit mode (JMBMODE = 0) when JMBI0 has been written by the JTAG module. In 32-bit mode (JMBMODE = 1) when JMBI0 and JMBI1 have been written by the JTAG module.
5	Reserved	R	0h	Reserved. Always reads as 0.
4	NMIIFG	RW	0h	NMI pin interrupt flag 0b = No interrupt pending 1b = Interrupt pending
3	VMAIFG	RW	0h	Vacant memory access interrupt flag 0b = No interrupt pending 1b = Interrupt pending
2	Reserved	R	0h	Reserved. Always reads as 0.
1	OFIFG	RW	1h	Oscillator fault interrupt flag 0b = No interrupt pending 1b = Interrupt pending
0	WDTIFG	RW	0h	Watchdog timer interrupt flag. In watchdog mode, WDTIFG will self clear upon a watchdog timeout event. The SYSRSTIV can be read to determine if the reset was caused by a watchdog timeout event. In interval mode, WDTIFG is reset automatically by servicing the interrupt, or can be reset by software. Because other bits in ~IFG1 may be used for other modules, it is recommended to set or clear WDTIFG by using BIS.B or BIC.B instructions, rather than MOV.B or CLR.B instructions. 0b = No interrupt pending 1b = Interrupt pending

1.14.3 SFRRPCR Register

Reset Pin Control Register

Figure 1-10. SFRRPCR 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				SYSRSTRE[1]	SYSRSTUP[1]	SYSNMIIES	SYSNMI
r0	r0	r0	r0	rw-1	rw-1	rw-0	rw-0

(1) All devices except the MSP430F5438 (non-A) default to pullup enabled on the reset pin.

Table 1-14. SFRRPCR Register Description

Bit	Field	Type	Reset	Description
15-4	Reserved	R	0h	Reserved. Always reads as 0.
3	SYSRSTRE	RW	1h	Reset pin resistor enable 0b = Pullup/pulldown resistor at the $\overline{\text{RST}}$/NMI pin is disabled 1b = Pullup/pulldown resistor at the $\overline{\text{RST}}$/NMI pin is enabled
2	SYSRSTUP	RW	1h	Reset resistor pin pullup/pulldown 0b = Pulldown is selected 1b = Pullup is selected
1	SYSNMIIES	RW	0h	NMI edge select. This bit selects the interrupt edge for the NMI when SYSNMI = 1. Modifying this bit can trigger an NMI. Modify this bit when SYSNMI = 0 to avoid triggering an accidental NMI. 0b = NMI on rising edge 1b = NMI on falling edge
0	SYSNMI	RW	0h	NMI select. This bit selects the function for the RST/NMI pin. 0b = Reset function 1b = NMI function

1.15 SYS Registers

The SYS configuration registers are listed in Table 1-16 and the base address is listed in Table 1-15. A detailed description of each register and its bits is also provided. Each register starts at a word boundary. Either word or byte data can be written to the SYS configuration registers.

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

Table 1-15. SYS Base Address

Module	Base Address
SYS	00180h

Table 1-16. SYS Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	SYSCTL	System Control	Read/write	Word	0000h	Section 1.15.1
00h	SYSCTL_L		Read/write	Byte	00h	
01h	SYSCTL_H		Read/write	Byte	00h	
02h	SYSBSLC	Bootloader Configuration	Read/write	Word	0003h	Section 1.15.2
02h	SYSBSLC_L		Read/write	Byte	03h	
03h	SYSBSLC_H		Read/write	Byte	00h	
06h	SYSJMBC	JTAG Mailbox Control	Read/write	Word	0000h	Section 1.15.3
06h	SYSJMBC_L		Read/write	Byte	00h	
07h	SYSJMBC_H		Read/write	Byte	00h	
08h	SYSJMBI0	JTAG Mailbox Input 0	Read/write	Word	0000h	Section 1.15.4
08h	SYSJMBI0_L		Read/write	Byte	00h	
09h	SYSJMBI0_H		Read/write	Byte	00h	
0Ah	SYSJMBI1	JTAG Mailbox Input 1	Read/write	Word	0000h	Section 1.15.5
0Ah	SYSJMBI1_L		Read/write	Byte	00h	
0Bh	SYSJMBI1_H		Read/write	Byte	00h	
0Ch	SYSJMBO0	JTAG Mailbox Output 0	Read/write	Word	0000h	Section 1.15.6
0Ch	SYSJMBO0_L		Read/write	Byte	00h	
0Dh	SYSJMBO0_H		Read/write	Byte	00h	
0Eh	SYSJMBO1	JTAG Mailbox Output 1	Read/write	Word	0000h	Section 1.15.7
0Eh	SYSJMBO1_L		Read/write	Byte	00h	
0Fh	SYSJMBO1_H		Read/write	Byte	00h	
18h	SYSBERRIV	Bus Error Vector Generator	Read	Word	0000h	Section 1.15.11
1Ah	SYSUNIV	User NMI Vector Generator	Read	Word	0000h	Section 1.15.8
1Ch	SYSSNIV	System NMI Vector Generator	Read	Word	0000h	Section 1.15.9
1Eh	SYSRSTIV	Reset Vector Generator	Read	Word	0002h	Section 1.15.10

1.15.1 SYSCTL Register

SYS Control Register

Figure 1-11. SYSCTL 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		SYSJTAGPIN	SYSBSLIND	Reserved	SYSPMMPE	Reserved	SYSRIVECT
r0	r0	rw-[0]	r-0	r0	rw-[0]	r0	rw-[0]

Table 1-17. SYSCTL Register Description

Bit	Field	Type	Reset	Description
15-6	Reserved	R	0h	Reserved. Always reads as 0.
5	SYSJTAGPIN	RW	0h	Dedicated JTAG pins enable. Setting this bit disables the shared functionality of the JTAG pins and permanently enables the JTAG function. This bit can be set only once. After it is set, it remains set until a BOR occurs. 0b = Shared JTAG pins (JTAG mode selectable by SBW sequence) 1b = Dedicated JTAG pins (explicit 4-wire JTAG mode selection)
4	SYSBSLIND	RW	0h	BSL entry indication. This bit indicates a BSL entry sequence detected on the Spy-Bi-Wire pins. 0b = No BSL entry sequence detected 1b = BSL entry sequence detected
3	Reserved	R	0h	Reserved. Always reads as 0.
2	SYSPMMPE	RW	0h	PMM access protect. This controls the accessibility of the PMM control registers. Once set to 1, it only can be cleared by a BOR. 0b = Access from anywhere in memory 1b = Access only from the protected BSL segments
1	Reserved	R	0h	Reserved. Always reads as 0.
0	SYSRIVECT	RW	0h	RAM-based interrupt vectors 0b = Interrupt vectors generated with end address TOP of lower 64KB of flash, FFFFh 1b = Interrupt vectors generated with end address TOP of RAM

1.15.2 SYSBSLC Register

Bootloader Configuration Register

Figure 1-12. SYSBSLC Register

15	14	13	12	11	10	9	8
SYSBSLPE	SYSBSLOFF	Reserved					
rw-[0]	rw-[0]	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
Reserved					SYSBSLR	SYSBSLSIZE	
r0	r0	r0	r0	r0	rw-[0]	rw-[1]	rw-[1]

Table 1-18. SYSBSLC Register Description

Bit	Field	Type	Reset	Description
15	SYSBSLPE	RW	0h	BSL memory protection enable for the size covered in SYSBSLSIZE. By default, this bit is cleared by hardware with a BOR event (as indicated above), however the boot code that checks for an available BSL may set this bit by software to protect the BSL. Because devices normally come with a TI BSL preprogrammed and protected, the boot code sets this bit. 0b = Area not protected. Read, program, and erase of BSL memory is possible. 1b = Area protected
14	SYSBSLOFF	RW	0h	BSL memory disable for the size covered in SYSBSLSIZE 0b = BSL memory is addressed when this area is read. 1b = BSL memory behaves like vacant memory. Reads cause 3FFFh to be read. Fetches cause JMP $ to be executed.
13-3	Reserved	R	0h	Reserved. Always reads as 0.
2	SYSBSLR	RW	0h	RAM assigned to BSL 0b = No RAM assigned to BSL area 1b = Lowest 16 bytes of RAM assigned to BSL
1-0	SYSBSLSIZE	RW	03h	BSL size. Defines the space and size of flash memory that is reserved for the BSL. 00b = Size: BSL segment 3 01b = Size: BSL segments 2 and 3 10b = Size: BSL segments 1, 2, and 3 11b = Size: BSL segments 0, 1, 2, and 3

1.15.3 SYSJMBC Register

JTAG Mailbox Control Register

Figure 1-13. SYSJMBC 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
JMBCLR1OFF	JMBCLR0OFF	Reserved	JMBM0DE	JMBOUT1FG	JMBOUT0FG	JMBIN1FG	JMBIN0FG
rw-(0)	rw-(0)	r0	rw-0	r-(1)	r-(1)	rw-(0)	rw-(0)

Table 1-19. SYSJMBC Register Description

Bit	Field	Type	Reset	Description
15-8	Reserved	R	0h	Reserved. Always reads as 0.
7	JMBCLR1OFF	RW	0h	Incoming JTAG Mailbox 1 flag auto-clear disable 0b = JMBIN1FG cleared on read of JMB1IN register 1b = JMBIN1FG cleared by software
6	JMBCLR0OFF	RW	0h	Incoming JTAG Mailbox 0 flag auto-clear disable 0b = JMBIN0FG cleared on read of JMB0IN register 1b = JMBIN0FG cleared by software
5	Reserved	R	0h	Reserved. Always reads as 0.
4	JMBMODE	RW	0h	This bit defines the operation mode of JMB for JMBI0, JMBI1, JMBO0, and JMBO1. Before switching this bit, pad and flush out any partial content to avoid data drops. 0b = 16-bit transfers using JMBO0 and JMBI0 only 1b = 32-bit transfers using JMBI0, JMBI1, JMBO0, and JMBO1
3	JMBOUT1FG	RW	1h	Outgoing JTAG Mailbox 1 flag. This bit is cleared automatically when a message is written to the upper byte of JMBO1 or as word access (by the CPU, DMA,…) and is set after the message was read by JTAG. 0b = JMBO1 is not ready to receive new data. 1b = JMBO1 is ready to receive new data.
2	JMBOUT0FG	RW	1h	Outgoing JTAG Mailbox 0 flag. This bit is cleared automatically when a message is written to the upper byte of JMBO0 or as word access (by the CPU, DMA,…) and is set after the message was read by JTAG. 0b = JMBO0 is not ready to receive new data. 1b = JMBO0 is ready to receive new data.
1	JMBIN1FG	RW	0h	Incoming JTAG Mailbox 1 flag. This bit is set when a new message (provided by JTAG) is available in JMBI1. This flag is cleared automatically on read of JMBI1 when JMBCLR1OFF = 0 (auto clear mode). On JMBCLR1OFF = 1, JMBIN1FG needs to be cleared by software. 0b = JMBI1 has no new data. 1b = JMBI1 has new data available.
0	JMBIN0FG	RW	0h	Incoming JTAG Mailbox 0 flag. This bit is set when a new message (provided by JTAG) is available in JMBI0. This flag is cleared automatically on read of JMBI0 when JMBCLR0OFF = 0 (auto clear mode). On JMBCLR0OFF = 1, JMBIN0FG needs to be cleared by software. 0b = JMBI1 has no new data. 1b = JMBI1 has new data available.

1.15.4 SYSJMBI0 Register

JTAG Mailbox Input 0 Register

Figure 1-14. SYSJMBI0 Register

15	14	13	12	11	10	9	8
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 1-20. SYSJMBI0 Register 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

1.15.5 SYSJMBI1 Register

JTAG Mailbox Input 0 Register

Figure 1-15. SYSJMBI1 Register

15	14	13	12	11	10	9	8
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 1-21. SYSJMBI1 Register 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

1.15.6 SYSJMBO0 Register

JTAG Mailbox Output 0 Register

Figure 1-16. SYSJMBO0 Register

15	14	13	12	11	10	9	8
MSGHI							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
MSGL0							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 1-22. SYSJMBO0 Register 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

1.15.7 SYSJMBO1 Register

JTAG Mailbox Output 1 Register

Figure 1-17. SYSJMBO1 Register

15	14	13	12	11	10	9	8
MSGHI							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
MSGL0							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Table 1-23. SYSJMBO1 Register 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

1.15.8 SYSUNIV Register

User NMI Vector Register

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

Figure 1-18. SYSUNIV Register

15	14	13	12	11	10	9	8
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 1-24. SYSUNIV Register Description

Bit	Field	Type	Reset	Description
15-0	SYSUNIV	R	0h	User NMI vector. Generates a value that can be used as address offset for fast interrupt service routine handling. Writing to this register clears all pending user NMI flags. 00h = No interrupt pending 02h = NMIIFG interrupt pending (highest priority) 04h = OFIFG interrupt pending 06h = ACCVIFG interrupt pending 08h = BUSIFG interrupt pending (Not present on all devices. See device-specific datasheet)

1.15.9 SYSSNIV Register

System NMI Vector Register

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

Figure 1-19. SYSSNIV Register

15	14	13	12	11	10	9	8
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 1-25. SYSSNIV Register Description

Bit	Field	Type	Reset	Description
15-0	SYSSNIV	R	0h	System NMI vector. Generates a value that can be used as address offset for fast interrupt service routine handling. Writing to this register clears all pending system NMI flags. 00h = No interrupt pending 02h = SVMLIFG interrupt pending (highest priority) 04h = SVMHIFG interrupt pending 06h = SVSMLDLYIFG interrupt pending 08h = SVSMHDLYIFG interrupt pending 0Ah = VMAIFG interrupt pending 0Ch = JMBINIFG interrupt pending 0Eh = JMBOUTIFG interrupt pending 10h = SVMLVLRIFG interrupt pending 12h = SVMHVLRIFG interrupt pending 14h = Reserved

1.15.10 SYSRSTIV Register

Reset Interrupt Vector Register

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

Figure 1-20. SYSRSTIV Register

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

(1) Reset value depends on reset source.

Table 1-26. SYSRSTIV Register Description

Bit	Field	Type	Reset	Description
15-0	SYSRSTIV	R	02h-3Eh(1)	Reset interrupt vector. Generates a value that can be used as address offset for fast interrupt service routine handling to identify the last cause of a reset (BOR, POR, PUC) . Writing to this register clears all pending reset source flags. 00h = No interrupt pending 02h = Brownout (BOR) (highest priority) 04h = $\overline{\text{RST}}$/NMI (BOR) 06h = PMMSWBOR (BOR) 08h = Wakeup from LPMx.5 (BOR) 0Ah = Security violation (BOR) 0Ch = SVSL (POR) 0Eh = SVSH (POR) 10h = SVML_OVP (POR) 12h = SVMH_OVP (POR) 14h = PMMSWPOR (POR) 16h = WDT time out (PUC) 18h = WDT password violation (PUC) 1Ah = Flash password violation (PUC) 1Ch = Reserved 1Eh = PERF peripheral/configuration area fetch (PUC) 20h = PMM password violation (PUC) 22h to 3Eh = Reserved

(1) Reset value depends on reset source.

1.15.11 SYSBERRIV Register

System Bus Error Interrupt Vector Register

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

Figure 1-21. SYSBERRIV Register

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

Table 1-27. SYSBERRIV Register Description

Bit	Field	Type	Reset	Description
15-0	SYSBERRIV	R	0h	System bus error interrupt vector. Generates a value that can be used as an address offset for fast interrupt service routine handling. Writing to this register clears all pending flags. 00h = No interrupt pending 02h = USB module timed out. Wait state time out of 8 clock cycles. 16 clock cycles only on the F552x and F551x devices. 04h = Reserved for future extensions 06h = Reserved for future extensions 08h = Reserved for future extensions

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

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