Interrupts, Low Power Modes
Status Register

Figure 3-6. Status Register Bits

Table 3-1 describes the status register bits.

Table 3-1. Description of Status Register Bits

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>Overflow bit. This bit is set when the result of an arithmetic operation overflows the signed-variable range.</td>
</tr>
<tr>
<td>SCG1</td>
<td>System clock generator 1. This bit, when set, turns off the SMCLK.</td>
</tr>
<tr>
<td>SCG0</td>
<td>System clock generator 0. This bit, when set, turns off the DCOOCC generator, if DCOOCC is not used for MCLK or SMCLK.</td>
</tr>
<tr>
<td>OSCOFF</td>
<td>Oscillator Off. This bit, when set, turns off the LFXT1 crystal oscillator, when LFXT1CLK is not use for MCLK or SMCLK.</td>
</tr>
<tr>
<td>CPUOFF</td>
<td>CPU off. This bit, when set, turns off the CPU.</td>
</tr>
<tr>
<td>GIE</td>
<td>General interrupt enable. This bit, when set, enables maskable interrupts. When reset, all maskable interrupts are disabled.</td>
</tr>
<tr>
<td>N</td>
<td>Negative bit. This bit is set when the result of a byte or word operation is negative and cleared when the result is not negative.</td>
</tr>
<tr>
<td>Word operation:</td>
<td>( N ) is set to the value of bit 15 of the result</td>
</tr>
<tr>
<td>Byte operation:</td>
<td>( N ) is set to the value of bit 7 of the result</td>
</tr>
<tr>
<td>Z</td>
<td>Zero bit. This bit is set when the result of a byte or word operation is 0 and cleared when the result is not 0.</td>
</tr>
<tr>
<td>C</td>
<td>Carry bit. This bit is set when the result of a byte or word operation produced a carry and cleared when no carry occurred.</td>
</tr>
</tbody>
</table>
Interrupts
(Chapter 6 in text)

A computer has 2 basic ways to react to inputs:

1) polling: The processor regularly looks at the input and reacts as appropriate.
   + easy to implement and debug
   - processor intensive
     • if event is rare, you waste a lot of time checking
     • processor can’t go into low power (slow or stopped) modes

2) interrupts: The processor is “interrupted” by an event.
   + very efficient time-wise: no time wasted looking for an event that hasn’t occurred.
   + very efficient energy-wise: processor can be asleep most of the time.
   - can be hard to debug
Polling vs Interrupt

This program sets P1.0 based on state of P1.4.

```c
#include <msp430x20x3.h>

void main(void) {
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog
    P1DIR = 0x01;             // P1.0 output
    P1OUT = 0x10;            // P1.4 hi (pullup)
    P1REN |= 0x10;            // P1.4 pullup
    P1IE |= 0x10;             // P1.4 IRQ enabled
    P1IES |= 0x10;            // P1.4 Hi/lo edge
    P1IFG &= ~0x10;           // P1.4 IFG cleared
    _BIS_SR(LPM4_bits + GIE); // Enter LPM4

    while (1) {
        // Test P1.4
        if (0x10 & P1IN) P1OUT |= 0x01;
        else P1OUT &= ~0x01;
    }
}
```

This program toggles P1.0 on each push of P1.4.

```c
#include <msp430x20x3.h>

void main(void) {
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog
    P1DIR = 0x01;             // P1.0 output
    P1OUT = 0x10;            // P1.4 hi (pullup)
    P1REN |= 0x10;            // P1.4 pullup
    P1IE |= 0x10;             // P1.4 IRQ enabled
    P1IES |= 0x10;            // P1.4 Hi/lo edge
    P1IFG &= ~0x10;           // P1.4 IFG cleared

    _BIS_SR(LPM4_bits + GIE); // Enter LPM4

    // Port 1 interrupt service routine
    #pragma vector=PORT1_VECTOR
    __interrupt void Port_1(void) {
        P1OUT ^= 0x01;            // P1.0 = toggle
        P1IFG &= ~0x10;           // P1.4 IFG cleared
    }
}
```

The details are not important now, we will come back to the interrupt version later and go over it line-by-line, bit-by-bit.
What happens on interrupt?

Interrupt Acceptance
The interrupt latency is 6 cycles (CPU), from the acceptance of an interrupt request to the start of execution of the interrupt-service routine. 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 (e.g., I/O ports) remain set for servicing by software.
6. The SR is cleared. This terminates any low-power mode (next slide). 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.
8. This is different from a typical function call because SR is saved.

Return From Interrupt
The interrupt handling routine terminates with instruction: RETI (return from ISR)

The return from the interrupt takes 5 cycles (CPU) or 3 cycles (CPUx) to execute the following actions.
1. The SR with all previous settings pops from the stack. All previous settings of GIE, CPUOFF, etc. are now in effect, regardless of the settings used during the interrupt service routine.
2. The PC pops from the stack and begins execution at the point where it was interrupted.
Low power modes

**Figure 2-8. Typical Current Consumption of 21x1 Devices vs Operating Modes**

<table>
<thead>
<tr>
<th>SCG1</th>
<th>SCG0</th>
<th>OSCOFF</th>
<th>CPUOFF</th>
<th>Mode</th>
<th>CPU and Clocks Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Active</td>
<td>CPU is active, all enabled clocks are active</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>LPM0</td>
<td>CPU, MCLK are disabled, SMCLK, ACLK are active</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>LPM1</td>
<td>CPU, MCLK are disabled, DCO and DC generator are disabled if DCO is not used for SMCLK. ACLK is active</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>LPM2</td>
<td>CPU, MCLK, SMCLK, DCO are disabled, DC generator remains enabled, ACLK is active</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>LPM3</td>
<td>CPU, MCLK, SMCLK, DCO are disabled, DC generator disabled, ACLK is active</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>LPM4</td>
<td>CPU and all clocks disabled</td>
</tr>
</tbody>
</table>

**Figure 5-1. Basic Clock Module + Block Diagram – MSP430F2xx**

AA battery has capacity of about 2 Amp-hours.

This is about 2 million hours (228 years) at 0.1 μA.
Getting into and out of LPM

An enabled interrupt event wakes the MSP430 from any of the low-power operating modes. The program flow 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.
Getting into and out of LPM

Using C

```c
__bis_SR_register(CPUOFF + GIE);  // LPM0, ADC10_ISR will force exit

// ...

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void) {
  __bic_SR_register_on_exit(CPUOFF);  // Clear CPUOFF bit from 0(SR)
}

__bis_SR_register(unsigned short mask);  //BIS mask, SR

__bic_SR_register_on_exit(unsigned short mask);  //BIC mask, saved_SR
```

In main routine (enter LPM mode)

In ISR (exit LPM when returning to main program).
Registers that effect interrupts on P1

<table>
<thead>
<tr>
<th>Interrupt Flag</th>
<th>P1IFG</th>
<th>023h</th>
<th>Read/write</th>
<th>Reset with PUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt Edge Select</td>
<td>P1IES</td>
<td>024h</td>
<td>Read/write</td>
<td>Unchanged</td>
</tr>
<tr>
<td>Interrupt Enable</td>
<td>P1IE</td>
<td>025h</td>
<td>Read/write</td>
<td>Reset with PUC</td>
</tr>
</tbody>
</table>

If a bit in PIES=0, the corresponding bit in P1IFG is set on rising edge on corresponding input pin (P1IN).

If PIES=1, P1IFG is set on falling edge of P1IN.

If interrupt enable bit is set in (P1IE), and Global Interrupts are enabled (GIE in Status Register), an interrupt is requested when the corresponding interrupt flag is set (P1IFG).

### Note: Writing to P1IESx

Writing to P1IES, or P2IES can result in setting the corresponding interrupt flags.

<table>
<thead>
<tr>
<th>P1IESx</th>
<th>P1INx</th>
<th>P1IFGx</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 → 1</td>
<td>0</td>
<td>May be set</td>
</tr>
<tr>
<td>0 → 1</td>
<td>1</td>
<td>Unchanged</td>
</tr>
<tr>
<td>1 → 0</td>
<td>0</td>
<td>Unchanged</td>
</tr>
<tr>
<td>1 → 0</td>
<td>1</td>
<td>May be set</td>
</tr>
</tbody>
</table>
# Using interrupts on Port 1

Toggles P1.0 on each push of P1.4.

```c
void main(void) {
    WDTCTL = WDTPW + WDTHOLD;  // Stop watchdog
    P1DIR = 0x01;              // P1.0 output
    P1OUT = 0x10;             // P1.4 hi (pullup)
    P1REN |= 0x10;             // P1.4 pullup
    P1IE |= 0x10;              // P1.4 IRQ enabled
    P1IES |= 0x10;             // P1.4 Hi/lo edge
    P1IFG &= ~0x10;            // P1.4 IFG cleared
    _BIS_SR(LPM4_bits + GIE);  // Enter LPM4
}
```

// Port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void) {
    P1OUT ^= 0x01;             // P1.0 = toggle
    P1IFG &= ~0x10;            // P1.4 IFG cleared
}
```

- P1.0 is output
- P1.4 resistor is enabled
- P1.4 resistor is connected to logic 1
- Enable interrupt on P1.4 (GIE is still cleared)
- Set sensitivity to falling edge.
- Clear Interrupt flag (just in case).
- Enter LPM4 and enable interrupts

Tell compiler to fill in interrupt vector with address of this function
Tell compiler to return from function with "iret" (as opposed to "ret")
Toggle P1.0
Clear interrupt flag. (Some interrupts do this automatically, check manual, or example code)
Keep ISR’s short!

It is important to keep interrupt service routines short. Since interrupts are disable globally during an ISR, you might miss something important.

If you need to do a lot of processing, have the ISR set a flag, and have main routine act on it.

Figure 1. MSP430 Top-Level Code Flow
Interrupt Vector Addresses

The interrupt vectors and the power-up starting address are located in the address range 0FFFFh to 0FFC0h. The vector contains the 16-bit address of the appropriate interrupt handler instruction sequence.

If the reset vector (located at address 0FFFEh) contains 0FFFFh (for example, flash is not programmed), the CPU goes into LPM4 immediately after power-up.

<table>
<thead>
<tr>
<th>INTERRUPT SOURCE</th>
<th>INTERRUPT FLAG</th>
<th>SYSTEM INTERRUPT</th>
<th>WORD ADDRESS</th>
<th>PRIORITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power-Up External Reset</td>
<td>PORIFG, RSTIFG</td>
<td>Reset</td>
<td>0FFFEh</td>
<td>31, highest</td>
</tr>
<tr>
<td>Watchdog Timer+</td>
<td>WDTIFG, KEYV</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flash key violation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC out-of-range(1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMI</td>
<td>NMIIFG, OFIFG</td>
<td>(non)-maskable</td>
<td>0FFFC0h</td>
<td>30</td>
</tr>
<tr>
<td>Oscillator fault</td>
<td>ACCVIFG(2)(3)</td>
<td>(non)-maskable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flash memory access violation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comparator_A+</td>
<td>TA1CCR0 CCIFG(4)</td>
<td>maskable</td>
<td>0FFFAh</td>
<td>29</td>
</tr>
<tr>
<td>Watchdog Timer+</td>
<td>TA1CCR2 TA1CCR1 CCIFG, TA1IFG(2)(4)</td>
<td>maskable</td>
<td>0FFF8h</td>
<td>28</td>
</tr>
<tr>
<td>Timer1_A3</td>
<td></td>
<td></td>
<td>0FFFC0h</td>
<td>27</td>
</tr>
<tr>
<td>Timer0_A3</td>
<td></td>
<td></td>
<td>0FFFEh</td>
<td>26</td>
</tr>
<tr>
<td>USCI_A0/USCI_B0 receive</td>
<td>UCA0RXIFG, UCB0RXIFG(2)(5)</td>
<td>maskable</td>
<td>0FFFCh</td>
<td>24</td>
</tr>
<tr>
<td>USCI_B0 I2C status</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USCI_A0/USCI_B0 transmit</td>
<td>UCA0TXIFG, UCB0TXIFG(2)(6)</td>
<td>maskable</td>
<td>0FFFEh</td>
<td>22</td>
</tr>
<tr>
<td>USCI_B0 I2C receive/transmit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC10 (MSP430G2x53 only)</td>
<td>ADC10IFG(4)</td>
<td></td>
<td>0FFFEAh</td>
<td>21</td>
</tr>
<tr>
<td>I/O Port P2 (up to eight flags)</td>
<td>P2IFG.0 to P2IFG.7(2)(4)</td>
<td>maskable</td>
<td>0FFFEh</td>
<td>20</td>
</tr>
<tr>
<td>I/O Port P1 (up to eight flags)</td>
<td>P1IFG.0 to P1IFG.7(2)(4)</td>
<td>maskable</td>
<td>0FFFE4h</td>
<td>19</td>
</tr>
</tbody>
</table>

(1) A reset is generated if the CPU tries to fetch instructions from within the module register memory address range (0h to 01FFh) or from addresses within the unused address space.

(2) Multiple interrupt-enable bits can disable an interrupt event, but the general interrupt enable cannot.

(3) Interrupt flags are located in the module.

(4) This location is used as a boot program security key (BLSKEY). This key disables the erase of the flash if an invalid password is supplied.

(5) In UART or SPI mode, USB0TXIFG in I2C mode: USB0RXIFG, USB0IFG, USB0TXIFG, USB0RXIFG.

(6) The interrupt vectors at addresses 0FF00h to 0FFC0h are not used in this device and can be used for regular program code if necessary.

(7) Interrupt flags are located in the module.

(8) See (7)