CPE/EE 421 Microcomputers

Instructor: Dr Aleksandar Milenkovic
Lecture Note
S16

*Material used is in part developed by Dr. D. Raskovic and Dr. E. Jovanov

MSP430 Documentation

- MSP430 home page (TI)
  - www.ti.com/msp430
- User's manual
  - http://www.ece.uah.edu/~milenka/cpe421-04S/manuals/slau049c.pdf
- Datasheet
  - http://www.ece.uah.edu/~milenka/cpe421-04S/manuals/slas272c.pdf
- TI Workshop document
  - http://www.ece.uah.edu/~milenka/cpe421-04S/manuals/430_2002_atc_workshop.pdf
- IAR Workbench Tutorial
  - http://www.ece.uah.edu/~milenka/cpe421-04S/manuals/TUTOR.pdf

Basic Clock Systems-control registers

- **Basic Clock Module Control Registers**

The Basic Clock Module is configured using control registers DCOCTL, BCSCTL1, and BCSCTL2, and four bits from the CPU status register: SCG1, SCG0, OscOff, and CPUOFF.

User software can modify these control registers from their default condition at any time. The Basic Clock Module control registers are located in the byte-wide peripheral map and should be accessed with byte (.B) instructions.

Register State	Short Form	Register Type	Address	Initial State
DCO control register	DCOCTL	Read/write	056h	060h
Basic clock system control 1	BCSCTL1	Read/write	057h	084h
Basic clock system control 2	BCSCTL2	Read/write	058h	reset

Basic Clock Systems-control registers

- Direct SW Control
- DCOCLK can be Set - Stabilized
- Stable DCOCLK over Temp/Vcc.

BCSCTL2

058h

SELM.1	SELM.0	DIVM.1	DIVM.0	SELS	DIVS.1	DIVS.0	DCOR
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

BCSCTL1

057h

XT2Off	XTS	DIVA.1	DIVA.0	XT5V	Rsel.2	Rsel.1	Rsel.0
rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-0	rw-1	rw-0	rw-0

Selection of DCO nominal frequency

DCOCTL

056h

DCO.2	DCO.1	DCO.0	MOD.4	MOD.3	MOD.2	MOD.1	MOD.0
rw-0	rw-1	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0

Which of eight discrete DCO frequencies is selected

Define how often frequency f_{DCO+1} within the period of 32 DCOCLK cycles is used. Remaining clock cycles (32-MOD) the frequency f_{DCO} is mixed

RSEL.x Select DCO nominal frequency

DCO.x and MOD.x set exact DCOCLK

… select other clock tree options

Basic Clock Systems-control registers

- **Digitally-Controlled Oscillator (DCO) Clock-Frequency Control**

DCOCTL is loaded with a value of 060h with a valid PUC condition.

	7							0
DCOCTL	DCO.2	DCO.1	DCO.0	MOD.4	MOD.3	MOD.2	MOD.1	MOD.0
056H	0	1	1	0	0	0	0	0

MOD.0 .. MOD.4: The MOD constant defines how often the discrete frequency f_{DCO+1} is used within a period of 32 DCOCLK cycles.

During the remaining clock cycles (32–MOD) the discrete frequency f_{DCO} is used. When the DCO constant is set to seven, no modulation is possible since the highest feasible frequency has then been selected.

DCO.0 .. DCO.2: The DCO constant defines which one of the eight discrete frequencies is selected. The frequency is defined by the current injected into the dc generator.

Basic Clock Systems-control registers

- **Oscillator and Clock Control Register**

BCSCTL1 is affected by a valid PUC or POR condition.

	7							0
BCSCTL1	XT2Off	XTS	DIVA.1	DIVA.0	XT5V	Rsel.0	Rsel.1	Rsel.2
057h	1	0	0	0	0	1	0	0

Bit0 to Bit2: The internal resistor is selected in eight different steps.

Rsel.0 to Rsel.2 The value of the resistor defines the nominal frequency. The lowest nominal frequency is selected by setting Rsel=0.

Bit3, XT5V: XT5V should always be reset.

Bit4 to Bit5: The selected source for ACLK is divided by:

DIVA = 0: 1
DIVA = 1: 2
DIVA = 2: 4
DIVA = 3: 8

Basic Clock Systems-control registers

Bit6, XTS: The LFXT1 oscillator operates with a low-frequency or with a high-frequency crystal:

XTS = 0: The low-frequency oscillator is selected.

XTS = 1: The high-frequency oscillator is selected.

The oscillator selection must meet the external crystal's operating condition.

Bit7, XT2Off: The XT2 oscillator is switched on or off:

XT2Off = 0: the oscillator is on

XT2Off = 1: the oscillator is off if it is not used for MCLK or SMCLK.

Basic Clock Systems-control registers

BCSCTL2 is affected by a valid PUC or POR condition.

	7						0
BCSCTL2	SELM.1	SELM.0	DIVM.1	DIVM.0	SELS	DIVS.1	DIVS.0	DCOR

058h

Bit0, DCOR: The DCOR bit selects the resistor for injecting current into the dc generator. Based on this current, the oscillator operates if activated.

DCOR = 0: Internal resistor on, the oscillator can operate. The fail-safe mode is on.

DCOR = 1: Internal resistor off, the current must be injected externally if the DCO output drives any clock using the DCOCLK.

Bit1, Bit2: The selected source for SMCLK is divided by:

DIVS.1 .. DIVS.0 DIVS = 0:1

DIVS = 1: 2

DIVS = 2: 4

DIVS = 3: 8

Basic Clock Systems-control registers

Bit3, SELS: Selects the source for generating SMCLK:

SELS = 0: Use the DCOCLK

SELS = 1: Use the XT2CLK signal (in three-oscillator systems)

or

LFXT1CLK signal (in two-oscillator systems)

Bit4, Bit5: The selected source for MCLK is divided by DIVM.0 .. DIVM.1

DIVM = 0: 1

DIVM = 1: 2

DIVM = 2: 4

DIVM = 3: 8

Bit6, Bit7: Selects the source for generating MCLK:

SELM.0 .. SELM.1

SELM = 0: Use the DCOCLK

SELM = 1: Use the DCOCLK

SELM = 2: Use the XT2CLK (x13x and x14x devices)

or

Use the LFXT1CLK (x11x(1) devices)

SELM = 3: Use the LFXT1CLK

Initial State

DCOCTL: 60h => DCO = 011b, MOD = 00000b

BCSCTL1: 84h => XT2Off = 1, XTS = 0, DIVA = 00b, Rsel = 000b

BCSCTL2: 00x => SELM = 00b, DIVM = 00b, SELS=0, DIVS=00, DCOR = 0

IE1 (Interrupt Enable 1):
00x (OFIE bit 1: oscillator fault interrupt enable => enables OFIFG interrupt)

IFG1 (Interrupt Flag Register 1):

00x (OFIFG bit 1: oscillator fault interrupt flag)

⇒DCO selected

⇒MCLK = f(Rsel=000, DCO=011b) = 800KHz (min); 1200KHz (nom)

⇒SMCLK = MCLK

⇒ACLK = N/A

msp430x14x.h: STATUS REG. BITS

```
/************************************************************
* STATUS REGISTER BITS
************************************************************/
#define C                   (0x0001)
#define Z                   (0x0002)
#define N                   (0x0004)
#define V                   (0x0100)
#define GIE                 (0x0008)    // GIE: General interrupt enable
#define CPUOFF              (0x0010)    // when set, turns off CPU
#define OSCOFF              (0x0020)    // when set, turns off LFXT1 (if not used for (S)MCLK)
#define SCG0                (0x0040)    // when set, turns off DCO (if not used for (S)MCLK)
#define SCG1                (0x0080)    // when set, turns off SMCLK

/* Low Power Modes coded with Bits 4-7 in SR */
#ifndef __IAR_SYSTEMS_ICC /* Begin #defines for assembler */
#define LPM0                (CPUOFF)
#define LPM1                (SCG0+CPUOFF)
#define LPM2                (SCG1+CPUOFF)
#define LPM3                (SCG1+SCG0+CPUOFF)
#define LPM4                (SCG1+SCG0+OSCOFF+CPUOFF)
/* End #defines for assembler */

#else /* Begin #defines for C */
#define LPM0_bits           (CPUOFF)
#define LPM1_bits           (SCG0+CPUOFF)
#define LPM2_bits           (SCG1+CPUOFF)
#define LPM3_bits           (SCG1+SCG0+CPUOFF)
#define LPM4_bits           (SCG1+SCG0+OSCOFF+CPUOFF)
```


msp430x14x.h: WDT BITS

```
/************************************************************
* WATCHDOG TIMER
************************************************************/

#define WDTCTL_             (0x0120)  /* Watchdog Timer Control */
sfrw    WDTCTL          = WDTCTL_;
/* The bit names have been prefixed with "WDT" */
#define WDTIS0              (0x0001)
#define WDTIS1              (0x0002)
#define WDTSSEL             (0x0004)
#define WDTCNTCL            (0x0008)
#define WDTTMSEL            (0x0010)
#define WDTNMI              (0x0020)
#define WDTNMIES            (0x0040)
#define WDTHOLD             (0x0080)

#define WDTPW               (0x5A00)
```

msp430x14x.h: WDT BITS

```
/* WDT-interval times [1ms] coded with Bits 0-2 */
/* WDT is clocked by fMCLK (assumed 1MHz) */
#define WDT_MDLY_32         (WDTPW+WDTTMSEL+WDTCNTCL)                       /* 32ms interval
(default) */
#define WDT_MDLY_8          (WDTPW+WDTTMSEL+WDTCNTCL+WDTIS0)                /* 8ms     " */
#define WDT_MDLY_0_5        (WDTPW+WDTTMSEL+WDTCNTCL+WDTIS1)                /* 0.5ms   " */
#define WDT_MDLY_0_064      (WDTPW+WDTTMSEL+WDTCNTCL+WDTIS1+WDTIS0)         /* 0.064ms " */
/* WDT is clocked by fACLK (assumed 32KHz) */
#define WDT_ADLY_1000       (WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL)               /* 1000ms  " */
#define WDT_ADLY_250        (WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTIS0)        /* 250ms   " */
#define WDT_ADLY_16         (WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTIS1)        /* 16ms    " */
#define WDT_ADLY_1_9        (WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTIS1+WDTIS0) /* 1.9ms   " */
/* Watchdog mode -> reset after expired time */
/* WDT is clocked by fMCLK (assumed 1MHz) */
#define WDT_MRST_32         (WDTPW+WDTCNTCL)                                /* 32ms interval
(default) */
#define WDT_MRST_8          (WDTPW+WDTCNTCL+WDTIS0)                         /* 8ms     " */
#define WDT_MRST_0_5        (WDTPW+WDTCNTCL+WDTIS1)                         /* 0.5ms   " */
#define WDT_MRST_0_064      (WDTPW+WDTCNTCL+WDTIS1+WDTIS0)                  /* 0.064ms " */
/* WDT is clocked by fACLK (assumed 32KHz) */
#define WDT_ARST_1000       (WDTPW+WDTCNTCL+WDTSSEL)                        /* 1000ms  " */
#define WDT_ARST_250        (WDTPW+WDTCNTCL+WDTSSEL+WDTIS0)                 /* 250ms   " */
#define WDT_ARST_16         (WDTPW+WDTCNTCL+WDTSSEL+WDTIS1)                 /* 16ms    " */
#define WDT_ARST_1_9        (WDTPW+WDTCNTCL+WDTSSEL+WDTIS1+WDTIS0)          /* 1.9ms   " */
```


msp430x14x.h: Basic Clock

```
/************************************************************
* Basic Clock Module
************************************************************/

#define DCOCTL_             (0x0056)  /* DCO Clock Frequency Control */
sfrb    DCOCTL            = DCOCTL_;
#define BCSCTL1_            (0x0057)  /* Basic Clock System Control 1 */
sfrb    BCSCTL1           = BCSCTL1_;
#define BCSCTL2_            (0x0058)  /* Basic Clock System Control 2 */
sfrb    BCSCTL2           = BCSCTL2_;

#define MOD0                (0x01)   /* Modulation Bit 0 */
#define MOD1                (0x02)   /* Modulation Bit 1 */
#define MOD2                (0x04)   /* Modulation Bit 2 */
#define MOD3                (0x08)   /* Modulation Bit 3 */
#define MOD4                (0x10)   /* Modulation Bit 4 */
#define DCO0                (0x20)   /* DCO Select Bit 0 */
#define DCO1                (0x40)   /* DCO Select Bit 1 */
#define DCO2                (0x80)   /* DCO Select Bit 2 */

#define RSEL0               (0x01)   /* Resistor Select Bit 0 */
#define RSEL1               (0x02)   /* Resistor Select Bit 1 */
#define RSEL2               (0x04)   /* Resistor Select Bit 2 */
#define XT5V                (0x08)   /* XT5V should always be reset */
#define DIVA0               (0x10)   /* ACLK Divider 0 */
#define DIVA1               (0x20)   /* ACLK Divider 1 */
#define XTS                 (0x40)   /* LFXTCLK 0:Low Freq. / 1: High Freq. */
#define XT2OFF              (0x80)   /* Enable XT2CLK */
```

msp430x14x.h: Basic Clock

```
#define DIVA_0              (0x00)   /* ACLK Divider 0: /1 */
#define DIVA_1              (0x10)   /* ACLK Divider 1: /2 */
#define DIVA_2              (0x20)   /* ACLK Divider 2: /4 */
#define DIVA_3              (0x30)   /* ACLK Divider 3: /8 */

#define DCOR                (0x01)   /* Enable External Resistor : 1 */
#define DIVS0               (0x02)   /* SMCLK Divider 0 */
#define DIVS1               (0x04)   /* SMCLK Divider 1 */
#define SELS                (0x08)   /* SMCLK Source Select 0:DCOCLK / 1:XT2CLK/LFXTCLK */
#define DIVM0               (0x10)   /* MCLK Divider 0 */
#define DIVM1               (0x20)   /* MCLK Divider 1 */
#define SELM0               (0x40)   /* MCLK Source Select 0 */
#define SELM1               (0x80)   /* MCLK Source Select 1 */

#define DIVS_0              (0x00)   /* SMCLK Divider 0: /1 */
#define DIVS_1              (0x02)   /* SMCLK Divider 1: /2 */
#define DIVS_2              (0x04)   /* SMCLK Divider 2: /4 */
#define DIVS_3              (0x06)   /* SMCLK Divider 3: /8 */

#define DIVM_0              (0x00)   /* MCLK Divider 0: /1 */
#define DIVM_1              (0x10)   /* MCLK Divider 1: /2 */
#define DIVM_2              (0x20)   /* MCLK Divider 2: /4 */
#define DIVM_3              (0x30)   /* MCLK Divider 3: /8 */

#define SELM_0              (0x00)   /* MCLK Source Select 0: DCOCLK */
#define SELM_1              (0x40)   /* MCLK Source Select 1: DCOCLK */
#define SELM_2              (0x80)   /* MCLK Source Select 2: XT2CLK/LFXTCLK */
#define SELM_3              (0xC0)   /* MCLK Source Select 3: LFXTCLK */
```


msp430x14x.h: Basic Clock

```
#define DIVA_0              (0x00)   /* ACLK Divider 0: /1 */
#define DIVA_1              (0x10)   /* ACLK Divider 1: /2 */
#define DIVA_2              (0x20)   /* ACLK Divider 2: /4 */
#define DIVA_3              (0x30)   /* ACLK Divider 3: /8 */

#define DCOR                (0x01)   /* Enable External Resistor : 1 */
#define DIVS0               (0x02)   /* SMCLK Divider 0 */
#define DIVS1               (0x04)   /* SMCLK Divider 1 */
#define SELS                (0x08)   /* SMCLK Source Select 0:DCOCLK / 1:XT2CLK/LFXTCLK */
#define DIVM0               (0x10)   /* MCLK Divider 0 */
#define DIVM1               (0x20)   /* MCLK Divider 1 */
#define SELM0               (0x40)   /* MCLK Source Select 0 */
#define SELM1               (0x80)   /* MCLK Source Select 1 */

#define DIVS_0              (0x00)   /* SMCLK Divider 0: /1 */
#define DIVS_1              (0x02)   /* SMCLK Divider 1: /2 */
#define DIVS_2              (0x04)   /* SMCLK Divider 2: /4 */
#define DIVS_3              (0x06)   /* SMCLK Divider 3: /8 */

#define DIVM_0              (0x00)   /* MCLK Divider 0: /1 */
#define DIVM_1              (0x10)   /* MCLK Divider 1: /2 */
#define DIVM_2              (0x20)   /* MCLK Divider 2: /4 */
#define DIVM_3              (0x30)   /* MCLK Divider 3: /8 */

#define SELM_0              (0x00)   /* MCLK Source Select 0: DCOCLK */
#define SELM_1              (0x40)   /* MCLK Source Select 1: DCOCLK */
#define SELM_2              (0x80)   /* MCLK Source Select 2: XT2CLK/LFXTCLK */
#define SELM_3              (0xC0)   /* MCLK Source Select 3: LFXTCLK */
```

Initialize Basic Clock Module: An Example

```
void InitOsc(void)
{
  WDTCTL = WDTPW | WDTHOLD;             // stop watchdog timer
  BCSCTL1 |= XTS;                       // XT1 as high-frequency
  _BIC_SR(OSCOFF);                      // turn on XT1 oscillator

  do                                    // wait in loop until crystal is stable
    IFG1 &= ~OFIFG;
  while (IFG1 & OFIFG);

  BCSCTL1 |= DIVA0;                     // ACLK = XT1 / 2
  BCSCTL1 &= ~DIVA1;

  IE1 &= ~WDTIE;                        // disable WDT int.
  IFG1 &= ~WDTIFG;                      // clear WDT int. flag

  WDTCTL = WDTPW | WDTTMSEL | WDTCNTCL | WDTSSEL | WDTIS1; // use WDT as timer, flag each
                                                          // 512 pulses from ACLK

  while (!(IFG1 & WDTIFG));                    // count 1024 pulses from XT1 (until XT1's
                                               // amplitude is OK)

  IFG1 &= ~OFIFG;                              // clear osc. fault int. flag
  BCSCTL2 = SELM0 | SELM1;                     // set XT1 as MCLK
}
```


Basic Clock Systems-Examples

➢ **Adjusting the Basic Clock**

The control registers of the Basic Clock are under full software control. If clock requirements other than those of the default from PUC are necessary, the Basic Clock can be configured or reconfigured by software at any time during program execution.

- ACLKGEN from LFXT1 crystal, resonator, or external-clock source and divided by 1, 2, 4, or 8. If no LFXTCLK clock signal is needed in the application, the OscOff bit should be set in the status register.
- SCLKGEN from LFXTCLK, DCOCLK, or XT2CLK (x13x and x14x only) and divided by 1, 2, 4, or 8. The SCG1 bit in the status register enables or disables SMCLK.
- MCLKGEN from LFXTCLK, DCOCLK, or XT2CLK (x13x and x14x only) and divided by 1, 2, 4, or 8. When set, the CPUOff bit in the status register enables or disables MCLK.
- DCOCLK frequency is adjusted using the RSEL, DCO, and MOD bits. The DCOCLK clock source is stopped when not used, and the dc generator can be disabled by the SCG0 bit in the status register (when set).
- The XT2 oscillator sources XT2CLK (x13x and x14x only) by clearing the XT2Off bit.

Watchdog Timer-General

General

The primary function of the watchdog-timer module (WDT) is to perform a controlled-system restart after a software problem occurs. If the selected time interval expires, a system reset is generated. If the watchdog function is not needed in an application, the module can work as an interval timer, to generate an interrupt after the selected time interval.

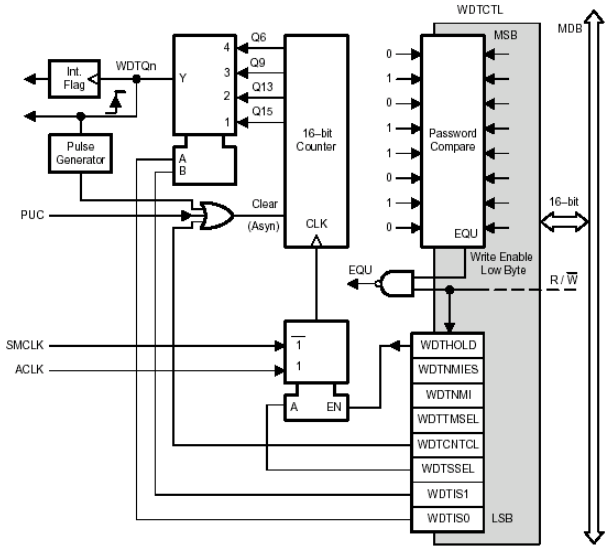
Features of the Watchdog Timer include:

- Eight software-selectable time intervals
- Two operating modes: as watchdog or interval timer
- Expiration of the time interval in watchdog mode, which generates a system reset; or in timer mode, which generates an interrupt request
- Safeguards which ensure that writing to the WDT control register is only possible using a password
- Support of ultralow-power using the hold mode

Watchdog/Timer two functions:

- SW Watchdog Mode
- Interval Timer Mode

Watchdog Timer-Diagram

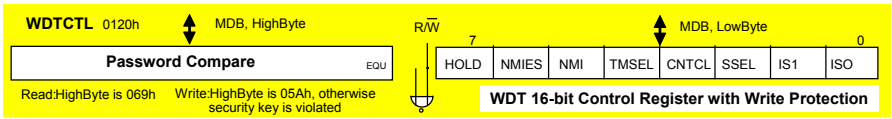


Watchdog Timer-Registers

❑ Watchdog Timer Counter

The watchdog-timer counter (WDTCNT) is a 16-bit up-counter that is not directly accessible by software. The WDTCNT is controlled through the watchdog-timer control register (WDTCTL), which is a 16-bit read/write register located at the low byte of word address 0120h. Any read or write access must be done using word instructions with no suffix or .w suffix. In both operating modes (watchdog or timer), it is only possible to write to WDTCTL using the correct password.

❑ Watchdog Timer Control Register



Bits 0, 1: Bits IS0 and IS1 select one of four taps from the WDTCNT, as described in following table. Assuming $f_{crystal}$ = 32,768 Hz and f_{System} = 1 MHz, the following intervals are possible:

Watchdog Timer-Registers

SSEL	IS1	IS0	Interval [ms]	
0	1	1	0.064	t SMCLK × 2^6
0	1	0	0.5	t SMCLK × 2^9
1	1	1	1.9	t ACLK × 2^6
0	0	1	8	t SMCLK × 2^{13}
1	1	0	16.0	t ACLK × 2^9
0	**0**	**0**	**32**	t SMCLK × 2^{15} <– Value after PUC (reset)
1	0	1	250	t ACLK × 2^{13}
1	0	0	1000	t ACLK × 2^{15}

Table: WDTCNT Taps

Bit 2: The SSEL bit selects the clock source for WDTCNT.

SSEL = 0: WDTCNT is clocked by SMCLK .

SSEL = 1: WDTCNT is clocked by ACLK.

Bit 3: Counter clear bit. In both operating modes, writing a 1 to this bit restarts the WDTCNT at 00000h. The value read is not defined.

Bit 4: The TMSEL bit selects the operating mode: watchdog or timer.

TMSEL = 0: Watchdog mode

TMSEL = 1: Interval-timer mode

Watchdog Timer-Registers

Bit 5: The NMI bit selects the function of the RST/NMI input pin. It is cleared by the PUC signal.

NMI = 0: The RST/NMI input works as reset input.

As long as the RST/NMI pin is held low, the internal signal is active (level sensitive).

NMI = 1: The RST/NMI input works as an edge-sensitive non-maskable interrupt input.

Bit 6: If the NMI function is selected, this bit selects the activating edge of the RST/NMI input. It is cleared by the PUC signal.

NMIES = 0: A rising edge triggers an NMI interrupt.

NMIES = 1: A falling edge triggers an NMI interrupt.

CAUTION: Changing the NMIES bit with software can generate an NMI interrupt.

Bit 7: This bit stops the operation of the watchdog counter. The clock multiplexer is disabled and the counter stops incrementing. It holds the last value until the hold bit is reset and the operation continues. It is cleared by the PUC signal.

HOLD = 0: The WDT is fully active.

HOLD = 1: The clock multiplexer and counter are stopped.

Watchdog Timer-Interrupt Function

- **The Watchdog Timer (WDT) uses two bits in the SFRs for interrupt control.**

The WDT interrupt flag (WDTIFG) (located in IFG1.0, initial state is reset)

The WDT interrupt enable (WDTIE) (located in IE1.0, initial state is reset)

- When using the watchdog mode, the WDTIFG flag is used by the reset interrupt service routine to determine if the watchdog caused the device to reset. If the flag is set, then the Watchdog Timer initiated the reset condition (either by timing out or by a security key violation). If the flag is cleared, then the PUC was caused by a different source. See chapter 3 for more details on the PUC and POR signals.
- When using the Watchdog Timer in interval-timer mode, the WDTIFG flag is set after the selected time interval and a watchdog interval-timer interrupt is requested. The interrupt vector address in interval-timer mode is different from that in watchdog mode. In interval-timer mode, the WDTIFG flag is reset automatically when the interrupt is serviced.
- The WDTIE bit is used to enable or disable the interrupt from the Watchdog Timer when it is being used in interval-timer mode. Also, the GIE bit enables or disables the interrupt from the Watchdog Timer when it is being used in interval-timer mode.

Watchdog Timer-Timer Mode

- Setting WDTCTL register bit TMSEL to 1 selects the timer mode. This mode provides periodic interrupts at the selected time interval. A time interval can also be initiated by writing a 1 to bit CNTCL in the WDTCTL register.

- When the WDT is configured to operate in timer mode, the WDTIFG flag is set after the selected time interval, and it requests a standard interrupt service. The WDT interrupt flag is a single-source interrupt flag and is automatically reset when it is serviced. The enable bit remains unchanged. In interval-timer mode, the WDT interrupt-enable bit and the GIE bit must be set to allow the WDT to request an interrupt. The interrupt vector address in timer mode is different from that in watchdog mode.

Watchdog Timer-Examples

- **How to select timer mode**

```
/* WDT is clocked by fACLK (assumed 32Khz) */
WDTCL=WDT_ADLY_250; // WDT 250MS/4 INTERVAL TIMER
IE1 |=WDTIE;        // ENABLE WDT INTERRUPT
```

- **How to stop watchdog timer**

```
WDTCTL=WDTPW + WDTHOLD ;  // stop watchdog timer
```

- **Assembly programming**

```
WDT_key     .equ    05A00h                ; Key to access WDT
WDTStop     mov    #(WDT_Key+80h),&WDTCTL ; Hold Watchdog
WDT250      mov    #(WDT_Key+1Dh),&WDTCTL ; WDT, 250ms Interval
```

System Reset and Initialization

- Reset
- Interrupt
- Watchdog Timer

Interrupt Service Routines

```
/************************************************************
* Interrupt Vectors (offset from 0xFFE0)
************************************************************/

#define PORT2_VECTOR        1 * 2  /* 0xFFE2 Port 2 */
#define UART1TX_VECTOR      2 * 2  /* 0xFFE4 UART 1 Transmit */
#define UART1RX_VECTOR      3 * 2  /* 0xFFE6 UART 1 Receive */
#define PORT1_VECTOR        4 * 2  /* 0xFFE8 Port 1 */
#define TIMERA1_VECTOR      5 * 2  /* 0xFFEA Timer A CC1-2, TA */
#define TIMERA0_VECTOR      6 * 2  /* 0xFFEC Timer A CC0 */
#define ADC_VECTOR          7 * 2  /* 0xFFEE ADC */
#define UART0TX_VECTOR      8 * 2  /* 0xFFF0 UART 0 Transmit */
#define UART0RX_VECTOR      9 * 2  /* 0xFFF2 UART 0 Receive */
#define WDT_VECTOR          10 * 2 /* 0xFFF4 Watchdog Timer */
#define COMPARATORA_VECTOR  11 * 2 /* 0xFFF6 Comparator A */
#define TIMERB1_VECTOR      12 * 2 /* 0xFFF8 Timer B 1-7 */
#define TIMERB0_VECTOR      13 * 2 /* 0xFFFA Timer B 0 */
#define NMI_VECTOR          14 * 2 /* 0xFFFC Non-maskable */
#define RESET_VECTOR        15 * 2 /* 0xFFFE Reset [Highest Pr.] */
```

Interrupt Service Routines

- **Interrupt Service Routine declaration**

```
// Func. declaration
Interrupt[int_vector] void myISR (Void);

Interrupt[int_vector] void myISR (Void)
{
//  ISR code
}
```


Reset Sources

- Hardware reset (POR – Power On Reset)
  - Internal power on reset
  - Reset line (RST#/NMI) is pulled low
- Software reset (POC – Power on Clear)
  - Watchdog timer expiration
  - Security key violations (Flash, Watchdog)
  - POR (poweron or RST line)

What happens on reset

- POR always calls the reset interrupt vector at 0FFFEh
- POC calls the reset interrupt vector or the interrupt vector of the system that generated it (Flash, WDT)
  - check the data sheet for device you are using

Reset Conditions

- POR
  - Status Register is cleared
  - PC is loaded with the address from FFFE
  - peripheral devices enter their poweron
- POC
  - Status Register is cleared
  - PC is loaded with the address from FFFE or from the interrupt vector of the system that generated interrupt (FFFC for WDT)
  - some peripheral devices are reset, some are not

Software Initialization

- After a system reset, user software must initialize the MSP430 for the application requirements.
  - Initialize the 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.
  - Additionally, the watchdog timer, oscillator fault, and flash memory flags can be evaluated to determine the source of the reset.

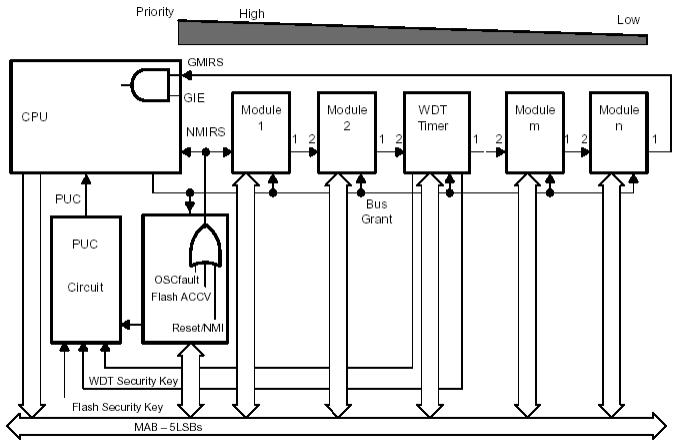
Problem of branched initialization

- Initializations for POR and POC require different setup to be performed
- Procedure
  - Set a Flash memory location to 0x00 after POR initialization
  - Check that location on any Reset
  - If it zero go on POC setup, otherwise it's POR

Interrupts

- Three types
  - System reset
  - (Non)-maskable NMI
  - Maskable
- GIE bit in SR: mask all maskable interrupts
- Enable/disable bit for each interrupt

Interrupt Priorities



Interrupt Sources, Flags, Vectors

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-up, external reset, watchdog, flash password	WDTIFG KEYV	Reset	0FFFEh	15, highest
NMI, oscillator fault, flash memory access violation	NMIIFG OFIFG ACCVIFG	(non)-maskable (non)-maskable (non)-maskable	0FFFCh	14
device-specific			0FFFAh	13
device-specific			0FFF8h	12
device-specific			0FFF6h	11
Watchdog timer	WDTIFG	maskable	0FFF4h	10
device-specific			0FFF2h	9
device-specific			0FFF0h	8
device-specific			0FFEEh	7
device-specific			0FFECh	6
device-specific			0FFEAh	5
device-specific			0FFE8h	4
I/O Port P2	P2IFG.0 to P2IFG.7	maskable	0FFE6h	3
I/O Port P1	P1IFG.0 to P1IFG.7	maskable	0FFE4h	2
device-specific			0FFE2h	1
device-specific			0FFE0h	0, lowest

NMI Interrupts

- Sources
  - An edge on the RST/NMI pin
  - An oscillator fault occurs
  - An access violation to the flash memory
- Cannot be masked by GIE, but controlled by individual bits ACCVIE, NMIIE, OFIE
- PC is initialized from (non)-maskable interrupt vector, 0FFFCh.
- When a NMI is accepted all control bits are reset (ACCVIE, NMIIE, OFIE)

RST/NMI Pin

- RST#/NMI pin is selected in the watchdog control register WDTCTL
- Reset function configuration
  - the CPU is held in the reset state as long as the RST#/NMI pin is held low
  - when in a high state the CPU starts program execution at the address stored in the reset vector, 0FFFEh
- NMI function configuration
  - a signal edge selected by the NMIES bit generates an NMI interrupt if the NMIIE bit is set. The RST/NMI flag NMIFG is also set.

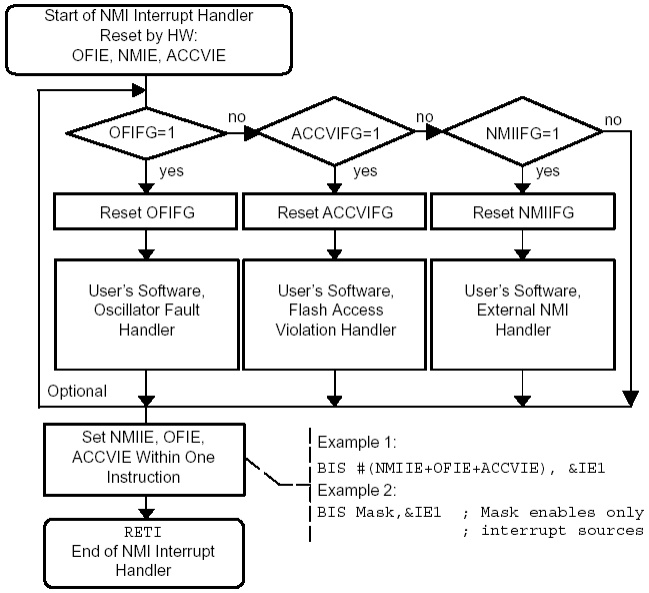
Oscillator Fault

- Warns for a possible error condition with crystal oscillator if OFIF bit is set
- The OFIFG flag can be tested by the interrupt service routine to determine if the NMI was caused by an oscillator fault

Flash Access Violation

- The flash ACCVIFG flag is set when a flash access violation occurs.
- The flash access violation can be enabled to generate an NMI interrupt by setting the ACCVIE bit.
- The ACCVIFG flag can then be tested by NMI the interrupt service routine to determine if the NMI was caused by a flash access violation.

NMI Interrupt Handler



Maskable Interrupts

- Caused by peripherals with interrupt capability + WDT in timer overflow mode
- All can be disabled with GIE (General Interrupt Enable)
- Each can be enabled/disabled individually interrupt enable bit (peripheral specific)

Interrupt Entrance

1. Any currently executing instruction is completed.
2. The PC, which points to the next instruction, is pushed onto the stack.
3. The SR is pushed onto the stack.
4. The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.
5. The interrupt request flag resets automatically on single-source flags. Multiple source flags remain set for servicing by software.
6. The SR is cleared with the exception of SCG0, which is left unchanged. This terminates any low-power mode.
7. The content of the interrupt vector is loaded into the PC: the program continues with the interrupt service routine at that address.

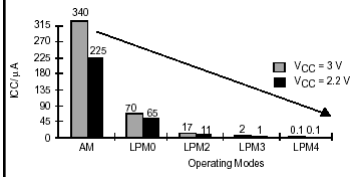
Return from Interrupt

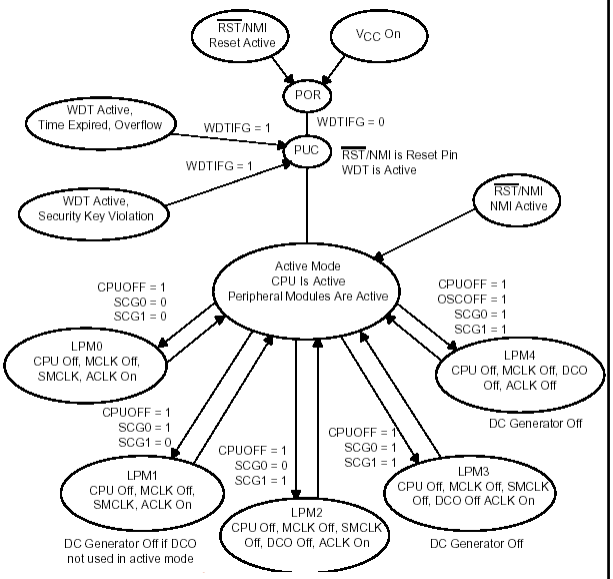
1. The SR with all previous settings pops from the stack. All previous settings of GIE, CPUOFF, etc. are now in effect, regardless of the settings used during the interrupt service routine.
2. The PC pops from the stack and begins execution at the point where it was interrupted.

Operating Modes for Basic Clock System (cont'd)

SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled SMCLK , ACLK are active
0	1	0	1	LPM1	CPU, MCLK, DCO osc. are disabled DC generator is disabled if the DCO is not used for MCLK or SMCLK in active mode SMCLK , ACLK are active
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO osc. are disabled DC generator remains enabled ACLK is active
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO osc. are disabled DC generator disabled ACLK is active
1	1	1	1	LPM4	CPU and all clocks disabled

Operating Modes for Basic Clock System





MSP430x1xx Microcontrollers
Low Power Modes

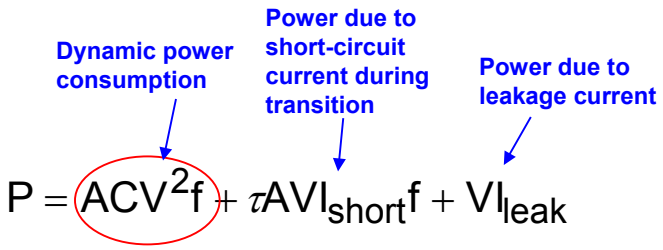
CPE/EE 421/521 Microcomputers

Power as a Design Constraint

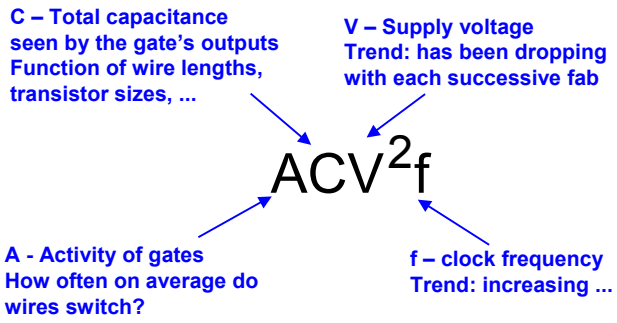
Power becomes a first class architectural design constraint

- Why worry about power?
  - Battery life in portable and mobile platforms
  - Power consumption in desktops, server farms
    - Cooling costs, packaging costs, reliability, timing
    - Power density: 30 W/cm2 in Alpha 21364 (3x of typical hot plate)
  - Environment?
    - IT consumes 10% of energy in the US

Where does power go in CMOS?



Dynamic Power Consumption

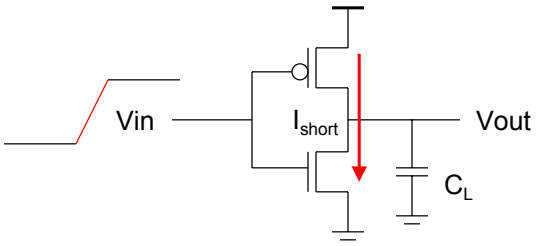


Reducing Dynamic Power

1) **Reducing V has quadratic effect; Limits?**
2) **Lower C - shrink structures, shorten wires**
3) **Reduce switching activity - Turn off unused parts or use design techniques to minimize number of transitions**

Short-circuit Power Consumption

$$\tau AVI_{short}f$$

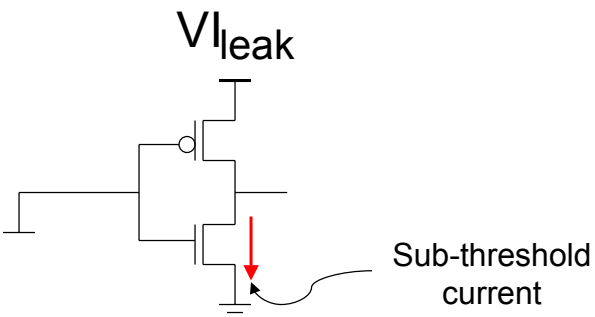


Finite slope of the input signal causes a direct current path between V_{DD} and GND for a short period of time during switching when both the NMOS and PMOS transistors are conducting

Reducing Short-circuit

1) **Lower the supply voltage V**
2) **Slope engineering – match the rise/fall time of the input and output signals**

Leakage Power



Sub-threshold current grows exponentially with increases in temperature and decreases in Vt

CMOS Power Equations

$$P = ACV^2f + \tau AVI_{short}f + VI_{leak}$$

Reduce the supply voltage, V

$$f_{max} \propto \frac{(V - V_t)^2}{V}$$

Reduce threshold V_t

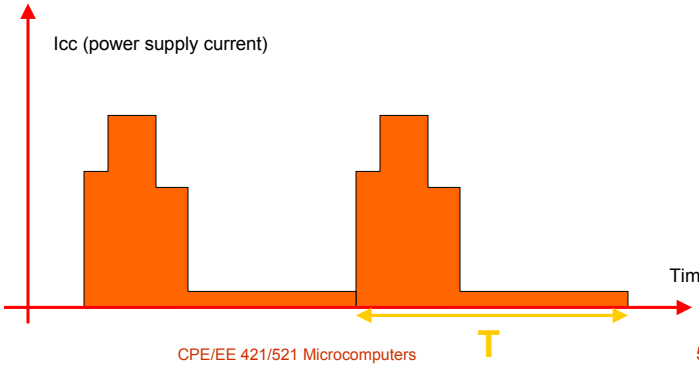
$$I_{leak} \propto \exp(-\frac{qV_t}{kT})$$

How can we reduce power consumption?

- Dynamic power consumption
  - charge/discharge of the capacitive load on each gate's output
  - frequency
- Control activity
  - reduce power supply voltage
  - reduce working frequency
  - turn off unused parts (module enables)
  - use low power modes
  - interrupt driven system
- Minimize the number of transitions
  - instruction formats, coding?

Average power consumption

- Dynamic power supply current
  - Set of modules that are periodically active
  - Typical situation – real time cycle T
  - Iave = $\int$ Icc(t)dt /T
  - In most cases Iave = Σ Ii*ti/T



Low-Power Concept: Basic Conditions for Burst Mode

The example of the heat cost allocator shows that the current of the non-activity periode dominates the current consumption.

		Measure		Process data		Real-Time Clock		LCD Display
I_{AVG}	=	$I_{Measure}$	+	$I_{Calculate}$	+	I_{RTC}	+	$I_{Display}$
	=	$I_{ADC} * t_{Measure}/T$	+	$I_{active} * t_{calc}/T$	+	$I_{active} * t_{RTC}/T$	+	$I_{Display}$
	=	3mA *200µs/60s	+	0.5mA * 10ms/60s	+	0.5mA * 0.5ms/60s	+	2.1µA
	=	10nA	+	83nA	+	4nA	+	2.1µA
I_{AVG}	≅							2.1µA

The ***sleep current dominates*** the current consumption!

The currents are related to the sensor and µC system. Additional current consumption of other system parts should be added for the total system current

Battery Life

- Battery Capacity BC – [mAh]
- Battery Life
  - BL = BC / Iave
- In the previous example, standard 800 mAh batteries will allow battery life of:
  - BL = 750 mAh / 2.1 µA ≈ 44 years !!!
- Conclusion:
  - Power efficient modes
  - Interrupt driven system with processor in idle mode

Low power - features

- Peak power
  - Possible damage
- Dynamic power
  - Non-ideal battery characteristics
  - Ground bounce, di/dt noise
- Energy/operation ratio
  - MIPS/W
  - Energy x Delay

Reducing power consumption

- Logic
  - Clock tree (up to 30% of power)
  - Clock gating (turn off branches that are not used)
  - Half frequency clock (both edges)
  - Half swing clock (half of Vcc)
  - Asynchronous logic
    - completion signals
    - testing
- Architecture
  - Parallelism (increased area and wiring)
  - Speculation (branch prediction)
  - Memory systems
    - Memory access (dynamic)
    - Leakage
    - Memory banks (turn off unused)
  - Buses
    - 32-64 address/data, (15-20% of power)
    - Gray Code, Code compression

Reducing power consumption #2

- Operating System
  - Finish computation "when necessary"
  - Scale the voltage
    - Application driven
    - Automatic
- System Architecture
  - Power efficient and specialized processing cores
  - A "convergent" architecture
  - Trade-off
    - AMD K6 / 400MHz / 64KB cache – 12W
    - XScale with the same cache 450 mW @ 600 MHz (40mW@150MHz)
    - 24 processors? Parallelism?
- Other issues
  - Leakage current – Thermal runaway
  - Voltage clustering (low Vthreshold for high speed paths)

Operating Modes-General

The MSP430 family was developed for ultralow-power applications and uses different levels of operating modes. The MSP430 operating modes, give advanced support to various requirements for ultralow power and ultralow energy consumption. This support is combined with an intelligent management of operations during the different module and CPU states. An interrupt event wakes the system from each of the various operating modes and the RETI instruction returns operation to the mode that was selected before the interrupt event.

The ultra-low power system design which uses complementary metal-oxide semiconductor (CMOS) technology, takes into account three different needs:

- The desire for speed and data throughput despite conflicting needs for ultra-low power
- Minimization of individual current consumption
- Limitation of the activity state to the minimum required by the use of low power modes

Low power mode control

There are four bits that control the CPU and the main parts of the operation of the system clock generator:

- CPUOff,
- OscOff,
- SCG0, and
- SCG1.

These four bits support discontinuous active mode (AM) requests, to limit the time period of the full operating mode, and are located in the status register. The major advantage of including the operating mode bits in the status register is that the present state of the operating condition is saved onto the stack during an interrupt service request. As long as the stored status register information is not altered, the processor continues (after RETI) with the same operating mode as before the interrupt event.

Operating Modes-General

Another program flow may be selected by manipulating the data stored on the stack or the stack pointer. Being able to access the stack and stack pointer with the instruction set allows the program structures to be individually optimized, as illustrated in the following program flow:

- **Enter interrupt routine**

The interrupt routine is entered and processed if an enabled interrupt awakens

the MSP430:

- The SR and PC are stored on the stack, with the content present at the interrupt event.
- Subsequently, the operation mode control bits OscOff, SCG1, and CPUOff are cleared automatically in the status register.

- **Return from interrupt**

Two different modes are available to return from the interrupt service routine and continue the flow of operation:

- Return with low-power mode bits set. When returning from the interrupt, the program counter points to the next instruction. The instruction pointed to is not executed, since the restored low power mode stops CPU activity.
- Return with low-power mode bits reset. When returning from the interrupt, the program continues at the address following the instruction that set the OscOff or CPUOff-bit in the status register. To use this mode, the interrupt service routine must reset the OscOff, CPUOff, SCGO, and SCG1 bits on the stack. Then, when the SR contents are popped from the stack upon RETI, the operating mode will be active mode (AM).

Operating Modes - Software configurable

There are six operating modes that the software can configure:

- Active mode AM; SCG1=0, SCG0=0, OscOff=0, CPUOff=0: CPU clocks are active
- Low power mode 0 (LPM0); SCG1=0, SCG0=0, OscOff=0, CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - SMCLK and ACLK remain active
- Low power mode 1 (LPM1); SCG1=0, SCG0=1, OscOff=0, CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - DCO's dc generator is disabled if the DCO is not used for MCLK or SMCLK when in active mode. Otherwise, it remains enabled.
  - SMCLK and ACLK remain active
- Low power mode 2 (LPM2); SCG1=1, SCG0=0, OscOff=0, CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - SMCLK is disabled
  - DCO oscillator automatically disabled because it is not needed for MCLK or SMCLK
  - DCO's dc-generator remains enabled
  - ACLK remains active

Operating Modes #2

- Low power mode 3 (LPM3); SCG1=1, SCG0=1, OscOff=0, CPUOff=1:
  - CPU is disabled
  - MCLK is disabled
  - SMCLK is disabled
  - DCO oscillator is disabled
  - DCO's dc-generator is disabled
  - ACLK remains active

- Low power mode 4 (LPM4); SCG1=X, SCG0=X, OscOff=1, CPUOff=1:
  - CPU is disabled
  - ACLK is disabled
  - MCLK is disabled
  - SMCLK is disabled
  - DCO oscillator is disabled
  - DCO's dc-generator is disabled
  - Crystal oscillator is stopped

Operating Modes-Low Power Mode in details

- **Low-Power Mode 0 and 1 (LPM0 and LPM1)**

Low power mode 0 or 1 is selected if bit CPUOff in the status register is set. Immediately after the bit is set the CPU stops operation, and the normal operation of the system core stops. The operation of the CPU halts and all internal bus activities stop until an interrupt request or reset occurs. The system clock generator continues operation, and the clock signals MCLK, SMCLK, and ACLK stay active depending on the state of the other three status register bits, SCG0, SCG1, and OscOff.

The peripherals are enabled or disabled with their individual control register settings, and with the module enable registers in the SFRs. All I/O port pins and RAM/registers are unchanged. Wake up is possible through all enabled interrupts.

- **Low-Power Modes 2 and 3 (LPM2 and LPM3)**

Low-power mode 2 or 3 is selected if bits CPUOff and SCG1 in the status register are set. Immediately after the bits are set, CPU, MCLK, and SMCLK operations halt and all internal bus activities stop until an interrupt request or reset occurs.

Peripherals that operate with the MCLK or SMCLK signal are inactive because the clock signals are inactive. Peripherals that operate with the ACLK signal are active or inactive according with the individual control registers and the module enable bits in the SFRs. All I/O port pins and the RAM/registers are unchanged. Wake up is possible by enabled interrupts coming from active peripherals or RST/NMI.

Operating Modes-Low Power Mode in details

- **Low-Power Mode 4 (LPM4)**

System Resets, Interrupts, and Operating Modes In low power mode 4 all activities cease; only the RAM contents, I/O ports, and registers are maintained. Wake up is only possible by enabled external interrupts.

Before activating LPM4, the software should consider the system conditions during the low power mode period . The two most important conditions are environmental (that is, temperature effect on the DCO), and the clocked operation conditions.

The environment defines whether the value of the frequency integrator should be held or corrected. A correction should be made when ambient conditions are anticipated to change drastically enough to increase or decrease the system frequency while the device is in LPM4.

Operating Modes-Examples

- **The following example describes entering into low-power mode 0.**

```
;===Main program flow with switch to CPUOff Mode==============
BIS #18h,SR ;Enter LPM0 + enable general interrupt GIE
            ;(CPUOff=1, GIE=1). The PC is incremented
            ;during execution of this instruction and
            ;points to the consecutive program step.
......      ;The program continues here if the CPUOff
            ;bit is reset during the interrupt service
            ;routine. Otherwise, the PC retains its
            ;value and the processor returns to LPM0.
```

- **The following example describes clearing low-power mode 0.**

```
;===Interrupt service routine=================================
......              ;CPU is active while handling interrupts
BIC #10h,0(SP)      ;Clears the CPUOff bit in the SR contents
                    ;that were stored on the stack.
RETI                ;RETI restores the CPU to the active state
                    ;because the SR values that are stored on
                    ;the stack were manipulated. This occurs
                    ;because the SR is pushed onto the stack
                    ;upon an interrupt, then restored from the
                    ;stack after the RETI instruction.
```


Operating Modes C Examples

- **C – programming msp430x14x.h**

```
/************************
* STATUS REGISTER BITS
************************/

#define C       0x0001
#define Z       0x0002
#define N       0x0004
#define V       0x0100
#define GIE     0x0008
#define CPUOFF  0x0010
#define OSCOFF  0x0020
#define SCG0    0x0040
#define SCG1    0x0080

/* Low Power Modes coded with
     Bits 4-7 in SR */
/* Begin #defines for assembler */
#ifndef __IAR_SYSTEMS_ICC
#define LPM0    CPUOFF
#define LPM1    SCG0+CPUOFF
#define LPM2    SCG1+CPUOFF
#define LPM3    SCG1+SCG0+CPUOFF
#define LPM4    SCG1+SCG0+OSCOFF+CPUOFF
/* End #defines for assembler */

#else /* Begin #defines for C */
#define LPM0_bits   CPUOFF
#define LPM1_bits   SCG0+CPUOFF
#define LPM2_bits   SCG1+CPUOFF
#define LPM3_bits   SCG1+SCG0+CPUOFF
#define LPM4_bits   SCG1+SCG0+OSCOFF+CPUOFF
```

- **…**

```
#include "In430.h"

#define LPM0      _BIS_SR(LPM0_bits) /* Enter LP Mode 0 */
#define LPM0_EXIT _BIC_SR(LPM0_bits) /* Exit LP Mode 0 */
#define LPM1      _BIS_SR(LPM1_bits) /* Enter LP Mode 1 */
#define LPM1_EXIT _BIC_SR(LPM1_bits) /* Exit LP Mode 1 */
#define LPM2      _BIS_SR(LPM2_bits) /* Enter LP Mode 2 */
#define LPM2_EXIT _BIC_SR(LPM2_bits) /* Exit LP Mode 2 */
#define LPM3      _BIS_SR(LPM3_bits) /* Enter LP Mode 3 */
#define LPM3_EXIT _BIC_SR(LPM3_bits) /* Exit LP Mode 3 */
#define LPM4      _BIS_SR(LPM4_bits) /* Enter LP Mode 4 */
#define LPM4_EXIT _BIC_SR(LPM4_bits) /* Exit LP Mode 4 */
#endif /* End #defines for C */
```

```
/* - in430.h -
     Intrinsic functions for the MSP430
*/

unsigned short _BIS_SR(unsigned short);
unsigned short _BIC_SR(unsigned short);
```