Course Administration

- **Instructor:** Aleksandar Milenkovic  
  milenka@ece.uah.edu  
  www.ece.uah.edu/~milenka  
  EB 217-L  
  Mon. 5:30 PM – 6:30 PM,  
  Wen. 12:30 – 13:30 PM

- **URL:** http://www.ece.uah.edu/~milenka/cpe421-05F

- **TA:** Joel Wilder

- **Labs:** Lab #3 is on.

- **Test I:** Graded. Solutions are in scr/.

- **Text:** Microprocessor Systems Design:  
  68000 Hardware, Software, and Interfacing

- **Review:** M68K (Chapter 1; Chapter 2; Chapter 3),  
  MSP430 (Introduction, Arch., Basic Clock System)

- **Today:** MSP430 Basic Clock System
MSP430: Basic Clock System

Basic Clock Systems

MSP430 Clock System
- Low System Cost
- Low Power

- Variety of operating modes driven by application, software selectable
- Support for the Burst Mode – when activated system starts and reacts rapidly
- Stability over voltage and temperature
Basic Clock Systems

- **Basic Clock System-MSP430x1xx**

  - **One DCO, internal digitally controlled oscillator**
    Generated on-chip RC-type frequency controlled by SW + HW

  - **One LF/XT oscillator**
    LF: 32768Hz
    XT: 450kHz .... 8MHz

  - **Second LF/XT2 oscillator**
    Optional XT: 450kHz .... 8MHz

- **Clocks:**
  - ACLK auxiliary clock
  - MCLK main system clock
  - SMCLK sub main system clock

Basic Clock Systems

- **DCOCLK** Generated on-chip with 6µs start-up
- **32KHz Watch Crystal** - or - **High Speed Crystal / Resonator to 8MHz**
  - (our system is 4MHz/8MHz high Speed Crystal)
- **Flexible clock distribution tree for CPU and peripherals**
- **Programmable open-loop DCO Clock with internal and external current source**

- **Digital Controlled Oscillator (DCO)**
  - **LFXT1 oscillator**
  - **LFXT1CLK**
  - **XIN**
  - **XOUT**
  - **100kHz - 5MHz**

- **LFXT2CLK**
  - **Rosc**
  - **Auxiliary Clock to peripherals**
  - **ACLK**
  - **Main System Clock to CPU**
  - **MCLK**
  - **Sub-System Clock to peripherals**
  - **SMCLK**
Basic Clock Systems-detail

The DCO-Generator is connected to pin P2.5/Rosc if DCOR control bit is set.
The port pin P2.5/Rosc is selected if DCOR control bit is reset (initial state).

Basic clock block diagram
(MSP430x13x/14x/15x/16x)
Basic operation

- After POC (Power Up Clear)
  MCLK and SCLK are sourced by DCOCLK (approx. 800KHz) and ACLK is sourced by LFXT1 in LF mode
- Status register control bits SCG0, SCG1, OSCOFF, and CPUOFF configure the MSP430 operating modes and enable or disable portions of the basic clock module
  - SCG1 - when set, turns off the SMCLK
  - SCG0 - when set, turns off the DCO dc generator (if DCOCLK is not used for MCLK or SMCLK)
  - OSCOFF - when set, turns off the LFXT1 crystal oscillator (if LFXT1CLK is not use for MCLK or SMCLK)
  - CPUOFF - when set, turns off the CPU
- DCOCTL, BCSCTL1, and BCSCTL2 registers configure the basic clock module
- The basic clock can be configured or reconfigured by software at any time during program execution

Basic Clock Systems-control registers(detail)

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.

<table>
<thead>
<tr>
<th>Register State</th>
<th>Short Form</th>
<th>Register Type</th>
<th>Address</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCO control register</td>
<td>DCOCTL</td>
<td>Read/write</td>
<td>056h</td>
<td>060h</td>
</tr>
<tr>
<td>Basic clock system control 1</td>
<td>BCSCTL1</td>
<td>Read/write</td>
<td>057h</td>
<td>084h</td>
</tr>
<tr>
<td>Basic clock system control 2</td>
<td>BCSCTL2</td>
<td>Read/write</td>
<td>058h</td>
<td>reset</td>
</tr>
</tbody>
</table>
Basic Clock Systems-control registers

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

BCSCTL1

<table>
<thead>
<tr>
<th>XT2DIR</th>
<th>XT5V</th>
<th>DIVM.1</th>
<th>DIVM.0</th>
<th>SELM.1</th>
<th>SELM.0</th>
<th>MOD.4</th>
<th>MOD.3</th>
<th>MOD.1</th>
<th>MOD.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw-(1)</td>
<td>ne-0</td>
<td>rw-(0)</td>
<td>ne-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>ne-0</td>
<td>ne-0</td>
<td>ne-0</td>
<td>ne-0</td>
</tr>
</tbody>
</table>

DCOCTL

<table>
<thead>
<tr>
<th>DCO.2</th>
<th>DCO.1</th>
<th>DCO.0</th>
<th>MOD.4</th>
<th>MOD.3</th>
<th>MOD.2</th>
<th>MOD.1</th>
<th>MOD.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw-0</td>
<td>rw-0</td>
<td>rw-0</td>
<td>ne-0</td>
<td>ne-0</td>
<td>ne-0</td>
<td>ne-0</td>
<td>ne-0</td>
</tr>
</tbody>
</table>

Selection of DCO nominal frequency
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

Low-power operation: An example

- ACLK can be configured to oscillate with a low-power 32,786-Hz watch crystal
- MCLK can be configured to operate from the on-chip DCO that can be only activated when requested by interrupt-driven events
- SMCLK can be configured to operate from either the watch crystal or the DCO, depending on peripheral requirements
LFXT1 Oscillator

- **LF mode: XTS = 0**
  - 32,768-Hz watch crystal in LF mode. A watch crystal connects to XIN and XOUT without any other external components.

- **HF mode: XTS = 1**
  - The high-speed crystal or resonator connects to XIN and XOUT and requires external capacitors on both terminals. These capacitors should be sized according to the crystal or resonator specifications.

- Software can disable LFXT1 by setting OSCOFF, if this signal does not source SMCLK or MCLK.

---

XT2 Oscillator

- Similar to LFXT1 in HF mode
- **XT2OFF** bit disables the XT2 oscillator if XT2CLK is not used for MCLK or SMCLK
Digitally-Controlled Oscillator DCO

- Integrated ring oscillator with RC-type characteristics
  - Frequency varies with temperature, voltage, and from device to device
- DCO frequency can be adjusted by software using the DCOx, MODx, and RSELx bits.
- The digital control of the oscillator allows frequency stabilization despite its RC-type characteristics
- After a PUC, an internal resistor is selected for the DC generator
  - $RSELx=4$, and $DCOx=3 \Rightarrow$ start at a mid-range frequency
- MCLK and SMCLK are sourced from DCOCLK. Because the CPU executes code from MCLK, which is sourced from the fast-starting DCO, code execution begins from PUC in less than 6us

Adjusting frequency

- DCO frequency is determined by:
  - The current injected into the DC generator by either the internal or external resistor defines the fundamental frequency.
    - $DCOR$ bit selects the internal or external resistor.
  - The three $RSELx$ bits select one of eight nominal frequency ranges for the DCO.
  - The three $DCOx$ bits divide the DCO range into 8 frequency steps, separated by approx. 10%.
  - The five $MODx$ bits, switch between the frequency selected by the $DCOx$ bits and the next higher frequency set by $DCO+1$. 
Disabling DCO

- Software can disable DCO if not used for MCLK and SMCLK

![Diagram of DCO control]

Basic Clock Systems-control registers(details)

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

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

<table>
<thead>
<tr>
<th>MOD.0</th>
<th>MOD.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

During the remaining clock cycles (32–MOD) the discrete frequency \( f_{\text{DCO}} \) 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 (details)

- Oscillator and Clock Control Register

  BCSCTL1 is affected by a valid PUC or POR condition.

<table>
<thead>
<tr>
<th>BCSCTL1</th>
<th>XT2Off</th>
<th>XTS</th>
<th>DIVA.1</th>
<th>DIVA.0</th>
<th>XT5V</th>
<th>Rsel.0</th>
<th>Rsel.1</th>
<th>Rsel.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>057h</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Bit 0 to Bit 2: 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.

Bit 3, XT5V: XT5V should always be reset.

Bit 4 to Bit 5: 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 (details)

- Bit 6, 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.

- Bit 7, 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(details)

**BCSCTL2** is affected by a valid PUC or POR condition.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELM.1</td>
<td>SELM.0</td>
</tr>
<tr>
<td>058h</td>
<td></td>
</tr>
</tbody>
</table>

**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

**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
### Range (RSELx) and Steps (DCOx)

![Diagram showing range and steps of RSEL and DCO](image)

### F149 default DCO clock setting

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TEST CONDITIONS</th>
<th>MIN</th>
<th>NOM</th>
<th>MAX</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>f(DCO0)</strong></td>
<td>Rset = 0, DCO = 3, MOD = 0, DOR = 0, TA = 25°C</td>
<td>0.08</td>
<td>0.12</td>
<td>0.15</td>
<td>MHz</td>
</tr>
<tr>
<td></td>
<td>VCC = 2.3 V</td>
<td>VCC = 3 V</td>
<td>VCC = 3.3 V</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>f(DCO3)</strong></td>
<td>Rset = 1, DCO = 3, MOD = 0, DOR = 0, TA = 25°C</td>
<td>0.14</td>
<td>0.19</td>
<td>0.23</td>
<td>MHz</td>
</tr>
<tr>
<td></td>
<td>VCC = 2.3 V</td>
<td>VCC = 3 V</td>
<td>VCC = 3.3 V</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>f(DCO23)</strong></td>
<td>Rset = 2, DCO = 3, MOD = 0, DOR = 0, TA = 25°C</td>
<td>0.22</td>
<td>0.33</td>
<td>0.38</td>
<td>MHz</td>
</tr>
<tr>
<td></td>
<td>VCC = 2.3 V</td>
<td>VCC = 3 V</td>
<td>VCC = 3.3 V</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>f(DCO30)</strong></td>
<td>Rset = 3, DCO = 3, MOD = 0, DOR = 0, TA = 25°C</td>
<td>0.37</td>
<td>0.49</td>
<td>0.59</td>
<td>MHz</td>
</tr>
<tr>
<td></td>
<td>VCC = 2.3 V</td>
<td>VCC = 3 V</td>
<td>VCC = 3.3 V</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>f(DCO45)</strong></td>
<td>Rset = 4, DCO = 3, MOD = 0, DOR = 0, TA = 25°C</td>
<td>0.61</td>
<td>0.75</td>
<td>0.90</td>
<td>MHz</td>
</tr>
<tr>
<td></td>
<td>VCC = 2.3 V</td>
<td>VCC = 3 V</td>
<td>VCC = 3.3 V</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>f(DCO60)</strong></td>
<td>Rset = 5, DCO = 3, MOD = 0, DOR = 0, TA = 25°C</td>
<td>1.3</td>
<td>1.5</td>
<td>1.5</td>
<td>MHz</td>
</tr>
<tr>
<td></td>
<td>VCC = 2.3 V</td>
<td>VCC = 3 V</td>
<td>VCC = 3.3 V</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>f(DCO75)</strong></td>
<td>Rset = 6, DCO = 3, MOD = 0, DOR = 0, TA = 25°C</td>
<td>2.2</td>
<td>2.2</td>
<td>2.2</td>
<td>MHz</td>
</tr>
<tr>
<td></td>
<td>VCC = 2.3 V</td>
<td>VCC = 3 V</td>
<td>VCC = 3.3 V</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>f(DCO90)</strong></td>
<td>Rset = 7, DCO = 3, MOD = 0, DOR = 0, TA = 25°C</td>
<td>2.7</td>
<td>2.7</td>
<td>2.7</td>
<td>MHz</td>
</tr>
<tr>
<td></td>
<td>VCC = 2.3 V</td>
<td>VCC = 3 V</td>
<td>VCC = 3.3 V</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>f(DCO4F)</strong></td>
<td>Rset = 4, DCO = 7, MOD = 0, DOR = 0, TA = 25°C</td>
<td>2.2</td>
<td>2.2</td>
<td>2.2</td>
<td>MHz</td>
</tr>
<tr>
<td></td>
<td>VCC = 2.3 V</td>
<td>VCC = 3 V</td>
<td>VCC = 3.3 V</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>f(DCO77)</strong></td>
<td>Rset = 7, DCO = 7, MOD = 0, DOR = 0, TA = 25°C</td>
<td>4.4</td>
<td>4.9</td>
<td>4.9</td>
<td>MHz</td>
</tr>
<tr>
<td></td>
<td>VCC = 2.3 V</td>
<td>VCC = 3 V</td>
<td>VCC = 3.3 V</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>S(Rx)</strong></td>
<td>SRx = Rset + 1 / Rend</td>
<td>VCC = 2.2 V</td>
<td>1.30</td>
<td>1.65</td>
<td>2</td>
</tr>
<tr>
<td><strong>S(DCO)</strong></td>
<td>SDCO = 1 / DCO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>C(Rx)</strong></td>
<td>Ctemp (2°C/°C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dv</strong></td>
<td>Drift with VCC variation, Rset = 4, DCO = 3, MOD = 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: All values are approximate and may vary based on specific conditions and specifications.
External Resistor

- The DCO temperature coefficient can be reduced by using an external resistor ROSC to source the current for the DC generator.
- ROSC also allows the DCO to operate at higher frequencies.
  - Internal resistor nominal value is approximately 200 kOhm => DCO to operate up to 5 MHz.
  - External ROSC of approximately 100 kOhm => the DCO can operate up to approximately 10 MHz.

Basic Clock Systems-software FLL idea

- Basic Clock DCO is an open loop - close with SW+HW
  - A reference frequency e.g. ACLK or 50/60Hz can be used to measure DCOCLK’s
  - Initialization or Periodic software set and stabilizes DCOCLK over reference clock
  - DCOCLK is programmable 100kHz - 5MHz and stable over voltage and temperature
Basic Clock Systems-software FLL implementation

Example: Set DCOCLK=1228800, ACLK=32768

- ACLK/4 captured on CCI2B, DCOCLK is clock source for Timer_A
- Comparator2 HW captures SMCLK (1228800Hz) in one ACLK/4 (8192Hz) period
- Target Delta = 1228800/8192 = 150

```assembly
CCI2BI ... ; Compute Delta
    cmp #150, Delta ; Delta = 1228800/8192
    jlo IncDCO ; JMP to IncDCO
DecDCO dec &DCOCTL ; Decrease DCOCLK
reti
IncDCO inc &DCOCTL ; Increase DCOCLK
reti
```

Basic Clock Systems-DCO TAPS

- DCOCLK frequency control
  - nominal - injected current into DC generator
    1) internal resistors Rsel2, Rsel1 and Rsel0
    2) an external resistor at Rosc (P2.5/11x)
- Control bits DCO0 to DCO2 set fDCO tap
- Modulation bits MOD0 to MOD4 allow mixing of fDCO and fDCO+1 for precise frequency generation

<table>
<thead>
<tr>
<th>Example Selected</th>
<th>Frequency</th>
<th>Cycle time</th>
</tr>
</thead>
<tbody>
<tr>
<td>f3: 1000kHz</td>
<td>1000 nsec</td>
<td></td>
</tr>
<tr>
<td>f4: 1042kHz</td>
<td>960 nsec</td>
<td></td>
</tr>
<tr>
<td>MOD=19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To produce an intermediate effective frequency between fDCO and fDCO+1
Cycle time = ((32-MOD)*tDCO+MOD*tDCO+1)/32 = 1000.625 ns, selected frequency = 1 MHz.
Fail Safe Operation

- Basic module incorporates an oscillator-fault detection fail-safe feature.
- The oscillator fault detector is an analog circuit that monitors the LFXT1CLK (in HF mode) and the XT2CLK.
- An oscillator fault is detected when either clock signal is not present for approximately 50 us.
  - When an oscillator fault is detected, and when MCLK is sourced from either LFXT1 in HF mode or XT2, MCLK is automatically switched to the DCO for its clock source.
- When OFIFG is set and OFIE is set, an NMI interrupt is requested. The NMI interrupt service routine can test the OFIFG flag to determine if an oscillator fault occurred. The OFIFG flag must be cleared by software.

Synchronization of clock signals

- When switching MCLK and SMCLK from one clock source to another
  => avoid race conditions
  - The current clock cycle continues until the next rising edge
  - The clock remains high until the next rising edge of the new clock
  - The new clock source is selected and continues with a full high period

![Figure 4-11: Switch MCLK from DCOCLK to LFXT1CLK](image-url)
Basic Clock Systems-Examples

How to select the Crystal Clock

```c
void selectclock(void)
{
    IFG2=0;     /* reset interrupt flag register 1 */
    IFG1=0;     /* reset interrupt flag register 2 */
    BCSC1|=XTS; /* attach HF crystal (4MHz) to XIN/XOUT */
    do {
        /* wait in loop until crystal is stable */
        IFG1&=~OFIFG;
    }while(OFIFG&IFG1);
    Delay();
    IFG1&=~OFIFG; /* Reset osc. fault flag again */
}
```

How to select a clock for MCLK

```c
BCSCTL2=SELM0+SELM1;    /* Then set MCLK same as LFXT1CLK */
TACTL=TASSEL0+TACLR+ID1; /* USE ACLK/4 AS TIMER_A INPUT CLOCK (1MHz) */
```

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.
Interrupt Service Routines

- **Interrupt Service Routine declaration**

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

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

**EXAMPLE**

```c
Interrupt[TIMERA0_VECTOR] void myISR (Void);
Interrupt[TIMERA0_VECTOR] void myISR (Void)
{
    // ISR code
}
```

### MSP430 interrupt vectors (int_vector)

- `PORT2_VECTOR` 1 * 2 /* 0xFFE2 Port 2 */
- `UART1TX_VECTOR` 2 * 2 /* 0xFFE4 UART 1 Transmit */
- `UART1RX_VECTOR` 3 * 2 /* 0xFFE6 UART 1 Receive */
- `PORT1_VECTOR` 4 * 2 /* 0xFFE8 Port 1 */
- `TIMERA1_VECTOR` 5 * 2 /* 0xFFEA Timer A CC1-2, TA */
- `TIMERA0_VECTOR` 6 * 2 /* 0xFFEC Timer A CC0 */
- `ADC_VECTOR` 7 * 2 /* 0xFFEE ADC */
- `UART0TX_VECTOR` 8 * 2 /* 0xFFF0 UART 0 Transmit */
- `UART0RX_VECTOR` 9 * 2 /* 0xFFF2 UART 0 Receive */
- `MDT_VECTOR` 10 * 2 /* 0xFFF4 Watchdog Timer */
- `COMPARATORA_VECTOR` 11 * 2 /* 0xFFF6 Comparator A */
- `TIMERB1_VECTOR` 12 * 2 /* 0xFFF8 Timer B 1-7 */
- `TIMERB0_VECTOR` 13 * 2 /* 0xFFF9 Timer B 0 */
- `IM1_VECTOR` 14 * 2 /* 0xFFFC Non-maskable */
- `RESET_VECTOR` 15 * 2 /* 0xFFFE Reset [Highest Pr.] */
MSP430: Watchdog Timer

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-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 the following table. Assuming f crystal = 32,768 Hz and f System = 1 MHz, the following intervals are possible:
### Watchdog Timer-Registers

<table>
<thead>
<tr>
<th>SSEL</th>
<th>IS1</th>
<th>IS0</th>
<th>Interval [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.064 (t \text{SMCLK} \times 2^6)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.5 (t \text{SMCLK} \times 2^9)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.9 (t \text{ACLK} \times 2^6)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8 (t \text{SMCLK} \times 2^{13})</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>16.0 (t \text{ACLK} \times 2^9)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>32 (t \text{SMCLK} \times 2^{15})</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>250 (t \text{ACLK} \times 2^{13})</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1000 (t \text{ACLK} \times 2^{15})</td>
</tr>
</tbody>
</table>

**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

**Bit 5**: The NMI bit selects the function of the RST/NMI input pin. It is

- 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**
  ```c
  /* 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**
  ```c
  WDTCTL=WDTPM + WDTHOLD ; // stop watchdog timer
  ```

- **Assembly programming**
  ```assembly
  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
  ```

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?

\[ P = ACV^2f + \tau AV_{\text{short}}f + V_{\text{leak}} \]
### Dynamic Power Consumption

- **C** – Total capacitance seen by the gate’s outputs
  - Function of wire lengths, transistor sizes, ...
- **V** – Supply voltage
  - Trend: has been dropping with each successive fab
- **A** – Activity of gates
  - How often on average do wires switch?
- **f** – Clock frequency
  - Trend: increasing ...

\[ ACV^2f \]

**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_{DP} 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

\[ P = ACV^2f + \alpha AV_{\text{short}}f + V_{\text{leak}} \]

- \( f_{\text{max}} \propto \frac{(V - V_t)^2}{V} \)
- \( I_{\text{leak}} \propto \exp\left(-\frac{qV_t}{kT}\right) \)
- Reduce the supply voltage, V
- Reduce threshold \( V_t \)
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 $T$
  - $I_{ave} = \int I_{cc}(t)dt /T$
  - In most cases $I_{ave} = \sum I_i t_i /T$

![Diagram showing power consumption over time](image)
Low-Power Concept: Basic Conditions for Burst Mode

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

\[ I_{AVG} = I_{Measure} + I_{Calculate} + I_{RTC} + I_{Display} \]

\[ I_{AVG} = 3mA \times 200\mu s/60s + 0.5mA \times 10ms/60s + 0.5mA \times 0.5ms/60s + 2.1\mu A \]

\[ I_{AVG} \approx 2.1\mu A \]

The sleep current dominates the current consumption!

The currents are related to the sensor and \( \mu C \) system. Additional current consumption of other system parts should be added for the total system current.

Battery Life

- Battery Capacity BC – [mAh]
- Battery Life
  - \( BL = BC / I_{ave} \)
- In the previous example, standard 800 mAh batteries will allow battery life of:
  - \( BL = 750 \text{ mAh} / 2.1 \mu A \approx 44 \text{ years} !!! \)
- 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, SCG0, 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
- **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.

- **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.

C – programming msp430x14x.h

#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 */

/* - in430.h -

Intrinsic functions for the MSP430 */
unsigned short _BIS_SR(unsigned short);
unsigned short _BIC_SR(unsigned short);