CPE/EE 421 Microcomputers
Instructor: Dr Aleksandar Milenkovic
Lecture Note
S12

Outline
MSP430: An Introduction
- The MSP430 family
- Technology Roadmap
- Typical Applications
- The MSP430 Documentation
- MSP430 Architecture
- MSP430 Devices
- Getting Started with EasyWeb2
- MSP430 RISC core

Review:
Getting Started with EasyWeb2

```c
#include <msp430x14x.h>

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;   // Stop watchdog timer
    P2DIR |= 0x02;  // Set P2.1 to output direction
    for (;;)
    {
        unsigned int i;
        P2OUT ^= 0x02; // Toggle P2.1 using exclusive-OR
        i = 50000;     // Delay
        do { i--; }
            while (i != 0);
    }
}
```

MSP430 16-bit RISC
- Large 16-bit register file eliminates single accumulator bottleneck
- High-bandwidth 16-bit data and address bus with no paging
- RISC architecture with 27 instructions and 7 addressing modes
- Single-cycle register operations with full-access
- Direct memory-memory transfer designed for modern programming
- Compact silicon 30% smaller than an 8051 saves power and cost

Course Administration
- Instructor: Aleksandar Milenkovic
  milenka@ece.uah.edu
  www.ece.uah.edu/~milenka
  EB 217-L
  Mon. 5:30 PM – 6:30 PM,
  Wed. 12:30 – 13:30 PM
- URL: http://www.ece.uah.edu/~milenka/cpe421-05F
- TA: Joel Wilder
- Labs: Lab #3 is on.
- Test 1: Graded. Solutions are in scr/.
- Text: Microprocessor Systems Design: 68000 Hardware, Software, and Interfacing
- Review: Chapter 1; Chapter 2; Chapter 3, MSP430 Basics.
- Today: MSP430 Introduction (cont’d)
Double Data Fetch Technology (DDFT)

CPU Introduction
- RISC architecture with 37 instructions and 7 addressing modes.
- Orthogonal architecture with every instruction usable with every addressing mode.
- Full register access including program counter, status registers, and stack pointer.
- Single-cycle register operations.
- Large 16-bit register file reduces fetches to memory.
- 16-bit address bus allows direct access and branching throughout entire memory range.
- 16-bit data bus allows direct manipulation of word-wide arguments.
- Constant generator provides six most used immediate values and reduces code size.
- Direct memory-to-memory transfers without intermediate register holding.
- Word and byte addressing and instruction formats.

CPU Registers
- R0 = PC Program Counter
  16-bit + no paging
- R1 = SP Stack Pointer
  Addressable by 27 instructions
- R2 = SR Status Register
  Defined UMIx
- R3/R2 = CD Constant Generator
  Automatic generation of common used values reduces code size 30%
- R4 = General Purpose
- R15 = General Purpose

R4 through R15 are single-cycle, general purpose and identical in all respects - used for math, storage, and addressing modes.

Registers: PC (R0)
- Each instruction uses an even number of bytes (2, 4, or 6)
- PC is word aligned (the LSB is 0)

MOV LABEL, PC ; Branch to address LABEL
MOV PC, LABEL ; Branch to address contained in LABEL
MOV [R14,PC] ; Branch indirect, indirect R14

Registers: SP (R1)
- Stack pointer for return addresses of subroutines and interrupts
- SP is word aligned (the LSB is 0)
- Pre-decrement/post-increment scheme

MOV 2(SP), R6 ; Item I2 -> R6
MOV R7, 0(SP) ; Overwrite TOS with R7
PUSH #0123h ; Put 0123h onto TOS
POP R8 ; R8 = 0123h

CPU Registers
- Registers: PC (R0)
- Registers: SP (R1)
- Registers: SR (R2)
Status bits

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Carry bit. This bit is set when the result of an arithmetic operation crosses the signed magnitude range.</td>
</tr>
<tr>
<td>S</td>
<td>Zero bit. Set when the result of an arithmetic operation is zero.</td>
</tr>
<tr>
<td>V</td>
<td>Overflow bit. Set when the result of an arithmetic operation is out of range.</td>
</tr>
<tr>
<td>CPE/EE 421/521 Microcomputers 13</td>
<td></td>
</tr>
</tbody>
</table>

Constant Generators SG1 and SG2

- Instruction field As – source register addressing mode in the instruction word is used as follows:

<table>
<thead>
<tr>
<th>Register</th>
<th>As</th>
<th>Constant</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2</td>
<td>00</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>01</td>
<td>(0)</td>
<td>Absolute address mode</td>
</tr>
<tr>
<td>R2</td>
<td>10</td>
<td>00000b</td>
<td>4, 8 bit processing</td>
</tr>
<tr>
<td>R2</td>
<td>11</td>
<td>00001b</td>
<td>1, 2 bit processing</td>
</tr>
<tr>
<td>R3</td>
<td>00</td>
<td>00001b</td>
<td>0, word processing</td>
</tr>
<tr>
<td>R3</td>
<td>01</td>
<td>00001b</td>
<td>1</td>
</tr>
<tr>
<td>R3</td>
<td>10</td>
<td>00002b</td>
<td>2, 16 bit processing</td>
</tr>
<tr>
<td>R3</td>
<td>11</td>
<td>00002b</td>
<td>1, word processing</td>
</tr>
</tbody>
</table>

CISC / RISC Instruction Set

- Three instruction formats:
  - Source, destination
  - Destination
  - Jumping

- Fifty-two instructions available in assembler
- 27 basic instructions
- 24 emulated instructions

- Seven addressing modes for source, four for destination:
  - Register Mode
  - Indexed Mode
  - Symbolic Mode
  - Absolute Mode
  - Indirect Mode
  - Indirect-arrangement Mode
  - Immediate Modes

- Bit, byte and word processing

CISC / RISC Instruction Set 15

Emulated Instructions

- Simply easier to understand with no code size or speed penalty
- Replaced by assembler with core instructions using CS, PC and SP

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add.w</td>
<td>Add word</td>
</tr>
<tr>
<td>dec.w</td>
<td>Decrement word</td>
</tr>
<tr>
<td>ret</td>
<td>Return word</td>
</tr>
</tbody>
</table>

CPE/EE 421/521 Microcomputers 17

CPE/EE 421/521 Microcomputers 18

27 Core RISC Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add.w</td>
<td>Add word</td>
</tr>
<tr>
<td>dec.w</td>
<td>Decrement word</td>
</tr>
<tr>
<td>ret</td>
<td>Return word</td>
</tr>
</tbody>
</table>

CPE/EE 421/521 Microcomputers 18

51 Total Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add.w</td>
<td>Add word</td>
</tr>
<tr>
<td>dec.w</td>
<td>Decrement word</td>
</tr>
<tr>
<td>ret</td>
<td>Return word</td>
</tr>
</tbody>
</table>

CPE/EE 421/521 Microcomputers 18
Double operand instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>S-Reg</th>
<th>D-Reg</th>
<th>Operation</th>
<th>Status Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD (A)</td>
<td>Acc</td>
<td>Acc</td>
<td>X + Y</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>SUB (A)</td>
<td>Acc</td>
<td>Acc</td>
<td>X - Y</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>MUL (A)</td>
<td>Acc</td>
<td>Acc</td>
<td>X * Y</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>DIV (A)</td>
<td>Acc</td>
<td>Acc</td>
<td>X / Y</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>CMP (A)</td>
<td>Acc</td>
<td>Acc</td>
<td>X = Y</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>AND (A)</td>
<td>Acc</td>
<td>Acc</td>
<td>X &amp; Y</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>OR (A)</td>
<td>Acc</td>
<td>Acc</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>XOR (A)</td>
<td>Acc</td>
<td>Acc</td>
<td>X ^ Y</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>SHL (A)</td>
<td>Acc</td>
<td>Acc</td>
<td>X &lt;&lt; Y</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>SHR (A)</td>
<td>Acc</td>
<td>Acc</td>
<td>X &gt;&gt; Y</td>
<td>V, N, Z, C</td>
</tr>
</tbody>
</table>

Single operand instruction

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>S-Reg</th>
<th>D-Reg</th>
<th>Operation</th>
<th>Status Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOR</td>
<td></td>
<td></td>
<td>X ^ Y</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>MOV</td>
<td></td>
<td></td>
<td>X = Y</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>IN</td>
<td></td>
<td></td>
<td>X = X</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>OUT</td>
<td></td>
<td></td>
<td>X = X</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>ADDI</td>
<td></td>
<td></td>
<td>X + imm</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>SUBI</td>
<td></td>
<td></td>
<td>X - imm</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>MULI</td>
<td></td>
<td></td>
<td>X * imm</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>DIVI</td>
<td></td>
<td></td>
<td>X / imm</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>CMPI</td>
<td></td>
<td></td>
<td>X = imm</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>ANDI</td>
<td></td>
<td></td>
<td>X &amp; imm</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>ORI</td>
<td></td>
<td></td>
<td>X</td>
<td>imm</td>
</tr>
<tr>
<td>XORI</td>
<td></td>
<td></td>
<td>X ^ imm</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>SHLI</td>
<td></td>
<td></td>
<td>X &lt;&lt; imm</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>SHRI</td>
<td></td>
<td></td>
<td>X &gt;&gt; imm</td>
<td>V, N, Z, C</td>
</tr>
</tbody>
</table>

Jump Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>S-Reg</th>
<th>D-Reg</th>
<th>Operation</th>
<th>Status Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>JEQ</td>
<td></td>
<td>Acc</td>
<td>X = 0</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>JNE</td>
<td></td>
<td>Acc</td>
<td>X ≠ 0</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>JCS</td>
<td></td>
<td>Acc</td>
<td>C = 0</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>JCC</td>
<td></td>
<td>Acc</td>
<td>C ≠ 0</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>JZ</td>
<td></td>
<td>Acc</td>
<td>X = 0</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>JNZ</td>
<td></td>
<td>Acc</td>
<td>X ≠ 0</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>JNC</td>
<td></td>
<td>Acc</td>
<td>C = 0</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>JNA</td>
<td></td>
<td>Acc</td>
<td>C ≠ 0</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>JNO</td>
<td></td>
<td>Acc</td>
<td>X ≥ 0</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>JNL</td>
<td></td>
<td>Acc</td>
<td>X &lt; 0</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>JGE</td>
<td></td>
<td>Acc</td>
<td>X ≥ 0</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>JLE</td>
<td></td>
<td>Acc</td>
<td>X ≤ 0</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>JPL</td>
<td></td>
<td>Acc</td>
<td>X &lt; 0</td>
<td>V, N, Z, C</td>
</tr>
<tr>
<td>JPL</td>
<td></td>
<td>Acc</td>
<td>X ≥ 0</td>
<td>V, N, Z, C</td>
</tr>
</tbody>
</table>

Addressing Modes

<table>
<thead>
<tr>
<th>Addr</th>
<th>Addressing Mode</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Register mode</td>
<td>Rs</td>
<td>Register contents are operated</td>
</tr>
<tr>
<td>0001</td>
<td>Indexed mode</td>
<td>R[Rs+X]</td>
<td>(Rn + X) points to the operand, X is stored in the next word.</td>
</tr>
<tr>
<td>0010</td>
<td>Symbolic mode</td>
<td>ADDR</td>
<td>(PC + X) points to the operand, X is stored in the next word. Indexed mode (IPX) is used.</td>
</tr>
<tr>
<td>0011</td>
<td>Absolute mode</td>
<td>ADDR</td>
<td>The word following the instruction contains the absolute address. X is stored in the next word. Indexed mode (IPX) is used.</td>
</tr>
<tr>
<td>0100</td>
<td>Indirect register mode</td>
<td>R[Rn]</td>
<td>Rn is used as a pointer to the operand.</td>
</tr>
<tr>
<td>0101</td>
<td>Indirect autocode</td>
<td>R[Rs+Rn]</td>
<td>Rn is used as a pointer to the operand.</td>
</tr>
<tr>
<td>0110</td>
<td>Immediate mode</td>
<td>imm</td>
<td>X is stored in the next word.</td>
</tr>
</tbody>
</table>

3 Instruction Formats

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Source Register</th>
<th>Destination Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>R1: R2</td>
<td>R3: R4</td>
</tr>
<tr>
<td>0010</td>
<td>R1: R2</td>
<td>R3: R4</td>
</tr>
<tr>
<td>0100</td>
<td>R1: R2</td>
<td>R3: R4</td>
</tr>
</tbody>
</table>

Register Addressing Mode

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Status Register</th>
<th>Rs</th>
<th>Rt</th>
<th>Rs</th>
<th>Destination Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>R0: R1</td>
<td></td>
<td></td>
<td></td>
<td>R4: R5</td>
</tr>
<tr>
<td>0010</td>
<td>R0: R1</td>
<td></td>
<td></td>
<td></td>
<td>R4: R5-me</td>
</tr>
</tbody>
</table>

Valid for Source and destination Rs=00, Rd=01. This is the fastest addressing mode and needs the least memory.
Register-Indexed Addressing Mode

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Op Code</th>
<th>Source Register</th>
<th>Dst Register</th>
<th>As</th>
<th>Destination Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>0100</td>
<td>1 0 01</td>
<td>0101</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

441B00000000
MOV.W 100B(R6),200B(R5) ;

441B00000000
MOV.W 100B(R4),R5 ;

Valid for source and destination As=01, Ad=1
The address of the operand is the sum of the index and the contents of the register.

Register Indirect Autoincrement Addressing Mode

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Op Code</th>
<th>Source Register</th>
<th>Dst Register</th>
<th>As</th>
<th>Destination Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>0100</td>
<td>0 0 11</td>
<td>0101</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4435       MOV.W #R4+,R5 ;

4475       MOV.W #R4+,R5 ;

Source only As=11, Ad=1/2
The registers are used as a pointer to the operand. The registers are incremented afterwards - by 1 in byte mode, by 2 in word mode.

Symbolic Addressing Mode

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Op Code</th>
<th>Source Register</th>
<th>Dst Register</th>
<th>As</th>
<th>Destination Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>0100</td>
<td>1 0 01</td>
<td>0010</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4296f6400000
MOV.W #0E6,PC+1 ;

40156460
MOV.W #0E6,RS ;

Source and destination As=01, Ad=1
The content of the addresses EDE / TONI are used for the operation.

Absolute Addressing Mode

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Op Code</th>
<th>Source Register</th>
<th>Dst Register</th>
<th>As</th>
<th>Destination Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>0101</td>
<td>1 0 01</td>
<td>0101</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

425057000070
MOV.W #0C00,AC0B1 ;

421B00000000
MOV.W #0C00,R5 ;

Source and destination As=01, Ad=1
The contents of the fixed addresses are used for the operation. The SR is used in the indexed mode to create an absolute 0. Use for hardcoded addresses located at an absolute address that can never be relocated.

Register Indirect Addressing Mode

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Op Code</th>
<th>Source Register</th>
<th>Dst Register</th>
<th>As</th>
<th>Destination Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>0100</td>
<td>0 0 10</td>
<td>0101</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4425       MOV.W #R4,R5 ;

4465       MOV.W #R4,R5 ;

Source only As=10, Ad=1/2
The registers are used as a pointer to the operand. The indexed mode with zero index may be used for "indirect register addressing" of the destination operand.

Immediate Addressing Mode

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Op Code</th>
<th>Source Register</th>
<th>Dst Register</th>
<th>As</th>
<th>Destination Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>0000</td>
<td>0 0 11</td>
<td>0101</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

42551214
MOV.W #2241,RS ; ADD 16-Bit Value

Source only As=11, Ad=1/2
Any immediate 8 or 16 bit constant can be used with the instruction. The PC is used in autoincrement mode to emulate this addressing mode.
Code Reduction Effect of Constant Generator

R3/R2 - CG Constant Generator
Automatic generation of commonly used values reduces code size 30%

Machine Cycles for Format II/III Instructions

<table>
<thead>
<tr>
<th>Address Mode</th>
<th>N/A</th>
<th>N/A</th>
<th>Length of Instruction (Clocks)</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x, 0x</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>MOV, ADD</td>
</tr>
<tr>
<td>0x, w</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>CALL, JMP</td>
</tr>
<tr>
<td>18, gh</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>INC, DEC</td>
</tr>
<tr>
<td>18, gh, (h)</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>SINS, SUB</td>
</tr>
</tbody>
</table>

Machine Cycles for Format I Instructions

<table>
<thead>
<tr>
<th>Address Mode</th>
<th>N/A</th>
<th>N/A</th>
<th>Length of Instruction (Clocks)</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x, 0x</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>MOV, ADD</td>
</tr>
<tr>
<td>0x, w</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>CALL, JMP</td>
</tr>
<tr>
<td>18, gh</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>INC, DEC</td>
</tr>
<tr>
<td>18, gh, (h)</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>SINS, SUB</td>
</tr>
</tbody>
</table>

MSP430 Memory Model

- Unified 64kB continuous memory map
- Some instructions for data and peripherals
- Program and data in Flash or RAM with no restrictions
- Easy to understand with no paging
- Designed for modern programming techniques such as pointers and fast lookup tables

Memory Map

- Special function registers
  - memory locations 0000h – 000Fh
  - 0000h, 0001h: interrupt enables
  - 0002h, 0003h: module enable flags
  - 0004h, 0005h: module enable flags

- Peripheral registers
  - byte addressable: 0010h – 00FFh
  - word addressable: 0100h – 01FFh

RAM: 0200h
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 ACLK
- MCLK main system clock MCLK
- SMCLK sub main system clock

---

### Basic Clock Systems-detail

- **DCOCLK** Generated on-chip with 6us 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

---

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

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

- Selection of DCO nominal frequency
  - Which of eight discrete DCO frequencies is selected.

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

<table>
<thead>
<tr>
<th>RSEL.x</th>
<th>Select DCO nominal frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCO.x</td>
<td>and MOD.x set exact DCOCLK</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>select other clock tree options</td>
</tr>
</tbody>
</table>

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 => 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.
  - DCOx 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

Basic Clock Systems-control registers (details)

- Oscillator and Clock Control Register
  - BCSCTL1 is affected by a valid PUC or POR condition.

- 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

- 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(details)

BCSCTL2

<table>
<thead>
<tr>
<th>Bit</th>
<th>SELM.1</th>
<th>SELM.0</th>
<th>DIVM.1</th>
<th>DIVM.0</th>
<th>SELS</th>
<th>DIVS.1</th>
<th>DIVS.0</th>
<th>DCOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</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 DDCCLK.

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)

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 XT2CLK (in two-oscillator systems)
- SELM = 1: Use the DCOCLK

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.

F149 default DCO clock setting

<table>
<thead>
<tr>
<th>Frequency</th>
<th>LMX2</th>
<th>LMX1</th>
<th>LMX0</th>
<th>LMX-1</th>
<th>LMX-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>100kHz</td>
<td>493</td>
<td>493</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>200kHz</td>
<td>493</td>
<td>493</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>500kHz</td>
<td>493</td>
<td>493</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>1MHz</td>
<td>493</td>
<td>493</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>2MHz</td>
<td>493</td>
<td>493</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>5MHz</td>
<td>493</td>
<td>493</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
</tbody>
</table>

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 DDCCLK's
- Initialization or Periodic software set and stabilizes DDCCLK over reference clock
- DDCCLK is programmable 100kHz - 5MHz and stable over voltage and temperature
Basic Clock Systems-software FLL implementation

Example: Set DCCOLK= 1228800, ACLK= 32768
- Comparator2 HW captures SMCLK (1228800Hz) in one ACLK (8192Hz) period
- Target Delta = 1228800/8192 = 150

DCOLK Int... ; Compute Delta
    cmp #150, Delta; ; CeilDelta 1228800/8192
    jlo IncDCO ; JMP to IncDCO

DecDCO dec &DCOCTL; ; Decrease DCOCLK

reti

IncDCO inc &DCOCTL; ; Increase DCOCLK

reti

Basic Clock Systems-DCO TAPS

- DCCOLK frequency control
  - nominal - injected current into DC generator
  - external resistor at Rosc (P2.5/11x)

- Modulation bits MOD0 to MOD4 allow mixing of DCO and DCO+1 for precise frequency generation

Example Selected
- f5: 1000kHz
- f4: 943kHz
- f3: 1042kHz
- f2: 1000 nsec
- f1: 1060 nsec
- f0: 960 nsec

DCCOLK

DCOCLK

DCO +0

DCO +1

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
- MCLK remains high until the next rising edge of the new clock
- The new clock source is selected and continues with a full high period

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 */
    BCSCTL1|=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<SEMS0<SEMS1; /*Then set MCLK same as LFXT1CLK*/
TACTL<TASSEL0+TACLR+ID1; /*USE ACLK/4 AS TIMER_A INPUT CLOCK (1MHz)*/
```

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.

  - ACUGEN from LPXT1 crystal, resonator, or external-clock source and divided by 1, 2, 4, or 8. If no LPXT1CLK clock signal is needed in the application, the DcoOff bit should be set in the status register.
  - SCUGEN from LPXT1CLK, DCCOLK, 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 LPXT1CLK, DCCOLK, 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.
  - DCCOLK frequency is adjusted using the RSEL, DCO, and MOD0 bits. The DCCOLK 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

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

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

EXAMPLE

Interrupt[TIMERA0_VECTOR] void myISR (void);

Interrupt[TIMERA0_VECTOR] void myISR (void)
{
    // ISR code
}

Interrupt Service Routines

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

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

EXAMPLE

Interrupt[TIMERA0_VECTOR] void myISR (Void);

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

MSP430 interrupt vectors (int_vector)

#defines

#define PORT2_VECTOR        1 * 2  /* 0xFFE2 Port 2 */
#define UART1TX_VECTOR      2 * 2  /* UART1 Transmit */
#define UART1RX_VECTOR      3 * 2  /* UART1 Receive */
#define PORT1_VECTOR        4 * 2  /* 0xFFE8 Port 1 */
#define TIMERA1_VECTOR      5 * 2  /* Timer A CC1-2, TA */
#define TIMERA0_VECTOR      6 * 2  /* Timer A CC0 */
#define ADC_VECTOR          7 * 2  /* 0xFFEE ADC */
#define UART0TX_VECTOR      8 * 2  /* UART0 Transmit */
#define UART0RX_VECTOR      9 * 2  /* UART0 Receive */
#define WDT_VECTOR          10 * 2 /* 0xFFF4 Watchdog Timer */
#define COMPARATORA_VECTOR  11 * 2 /* Comparator A */
#define TIMERB1_VECTOR      12 * 2 /* Timer B 1-7 */
#define TIMERB0_VECTOR      13 * 2 /* Timer B 0 */
#define NMI_VECTOR          14 * 2 /* 0xFFF6 Non-maskable */
#define RESET_VECTOR        15 * 2 /* 0xFFFE Reset [Highest Pr.] */

Interrupt Service Routines

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

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