Text Book: TI's Teaching ROMs
References: MSP 430 family data sheets and user's guides
http://www.ti.com/ww/eu/university/roms.html
Info

- **Time**: Mon. 6, 7, 8 (EE 203)

- **Evaluation**:
  - Lab 40%
  - Mid term 30% (Take Home)
  - Term Project 30%

- **Website**:
  [http://ares.ee.nchu.edu.tw/course/epr100/](http://ares.ee.nchu.edu.tw/course/epr100/)
Outline

- Introductory Overview
- MSP430 Architecture
- Device Systems and Operating Modes
- General purpose Input/Output
- Timers
- LCD Controller
- Data Acquisition
- Digital-to-Analogue Converter
- Direct Memory Access (DMA)
- Hardware Multiplier
- Flash Programming
- Communications
Analogue and digital signals and systems

Mathematical notations
  - Floating and fixed-point arithmetic

How to read technical datasheets

MSP430 Integrated Development Environments (IDEs)

MSP430 Experimenter’s board
  - eZ430-F2013
  - MSP-EXP430F5438
  - Wireless expansion (Chipcon’s RF transceiver chip)

MSP-FET430 Flash Emulation Tool
In nature, any measurable quantity in time or in space can be considered a signal;
- Example: The velocity of a body, as function of time or as function of position, can be represented by a signal.

- Analogue signal can:
  - Represent every value possible;
  - Swing through a “continuum” of values;
  - Provide information at every instant of time.

- A digital signal assumes only a finite number of values at only specific points in time.
Example: analogue signal:
Example: discrete analogue signal:
- This analogue signal can be discretized in time, using a sampling period, $T$: 
The information contained in the signal only can be used by a computer if an amplitude conversion from the analogue domain to the digital domains is performed;

- **Digital signal:**
  - Rounds off all values to a certain precision or a certain number of digits.

- **Digital signal advantage:**
  - Easiness of data storage and manipulation.
Example: quantized digital signal:

- Result of the analogue signal conversion, performed by a 12-bit Analogue-to-Digital Converter (ADC) with 2.5 V full-scale:
A base number is a notation that through symbols represents a consistent numerical value.

Depending on the numerical basis, the representation 11 may correspond to:

- Eleven for the base decimal (or radix decimal);
- Three for the base binary;
- Or some other number, depending on the base used.
Decimal numerical base representation review:

$$1234_{10} = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

- Each of the symbols is multiplied by a weight of base 10;
- As it moves from right to left the weight is increased.

General form for a number in base $b$:

$$a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \ldots + a_0b^0$$

- $n$ is the length of the numerical representation;
- $\{a_{n-1}, a_{n-2}, \ldots, a_0\}$ is the numerical representation ordered increasingly (natural numbers of the set $\{0, \ldots, b-1\}$).
Mathematical Notations (3/16)

- **Binary Number Base System:**
  - Computers can only take two states, so they use the base binary (base 2);
  - The states allowed are “open” or “closed”, “high” or “low”, “1” or “0”;

- **Example:** $010100_2$

  \[
  = 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 20_{10}
  \]

- **Representation:**

<table>
<thead>
<tr>
<th>Position</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight</td>
<td>$2^5$</td>
<td>$2^4$</td>
<td>$2^3$</td>
<td>$2^2$</td>
<td>$2^1$</td>
<td>$2^0$</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Base 2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Hexadecimal Number Base System:

- A difficulty that arises from the binary system is that it is verbose;
- To represent the value $202_{10}$ in base 2 (binary) requires eight binary digits;
- The decimal version requires only three decimal digits and so represents numbers much more compactly;
- The hexadecimal (base 16) numbering system solves these problems.
  - Hexadecimal numbers are very compact;
  - It is easy to convert from hexadecimal to binary and binary to hexadecimal.
Hexadecimal Number Base System:
- Set: \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\};
- Representation comes directly from the binary representation.

Example: $010100_2$
- Split the value $010100_2$ into groups of 4 bits from right to left.
- Pad out the short group with zeros at the front of the second group:
  $0001\ 0100_2 = 0\times14$ or $(14_{16})$

Example: $0x0ABCD$
- Convert each hexadecimal digit to its binary counterpart:
  $= 0000\ 1010\ 1011\ 1100\ 1101_2$
Decimal to Binary Conversion:

Example 1: Decimal number $721_{10}$ to binary conversion:

$721_{10} = 700 + 20 + 1 = 7 \times 10^2 + 2 \times 10^1 + 1 \times 10^0$

$2^{10} = 1024_{10}$ (Higher than 3rd power decimal value, $10^3$);
$2^9 = 512_{10}$ (MSB: $1 \times 2^9$): $721 - 512 = 209$;
$2^8 = 256_{10}$ (Higher than 209, the 2nd bit: $0 \times 2^8$);
$2^7 = 128_{10}$ (3rd bit is: $1 \times 2^7$): $209 - 128 = 81$;
$2^6 = 64_{10}$ (4th bit is: $1 \times 2^6$): $81 - 64 = 17$;
$2^5 = 32_{10}$ (Higher than 17, the 5th bit is: $0 \times 2^5$);
$2^4 = 16_{10}$ (6th bit is: $1 \times 2^4$): $17 - 16 = 1$;
$2^3 = 8_{10}$ (Higher than 1, the 7th bit: $0 \times 2^3$)
$2^2 = 4_{10}$ (Higher than 1, the 8th bit: $0 \times 2^2$);
$2^1 = 2_{10}$ (Higher than 1, the 9th bit: $0 \times 2^1$);
$2^0 = 1_{10}$ (10th bit - LSB): $1 \times 2^0$;

Result: $721_{10} = 1011010001_2$
This conversion method consists of continuously subtracting the highest power of 2 not higher than the remainder. If it can be subtracted, mark the corresponding position with '1', else with '0'.

- **Decimal to Binary Conversion:**
- Other method: Continuously divide by 2. The division remainder value is the corresponding position binary digit.

\[
103_{10} = 100 + 3 = 1 \times 10^2 + 0 \times 10^1 + 3 \times 10^0 \\
103/2 = 51 \quad \text{Remainder: 1 (LSB)} \\
51/2 = 25 \quad \text{Remainder: 1} \\
25/2 = 12 \quad \text{Remainder: 1} \\
12/2 = 6 \quad \text{Remainder: 0} \\
6/2 = 3 \quad \text{Remainder: 0} \\
3/2 = 1 \quad \text{Remainder: 1} \\
1/2 = 0 \quad \text{Remainder: 1 (MSB)}
\]

Result: \(103_{10} = 1100111_2\)
## Binary to Decimal Conversion:

- **Example 2:** Convert the binary number 1010111₂ to decimal notation.

This method is the inversion of method presented in example 1.

The binary number was 7 bits:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Bit Value</th>
<th>Calculation</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSB</td>
<td>1</td>
<td>1 \times 2^6 = 64</td>
<td></td>
</tr>
<tr>
<td>6th bit</td>
<td>0</td>
<td>0 \times 2^5 = 0</td>
<td></td>
</tr>
<tr>
<td>5th bit</td>
<td>1</td>
<td>1 \times 2^4 = 16</td>
<td></td>
</tr>
<tr>
<td>4th bit</td>
<td>0</td>
<td>0 \times 2^3 = 0</td>
<td></td>
</tr>
<tr>
<td>3rd bit</td>
<td>1</td>
<td>1 \times 2^2 = 4</td>
<td></td>
</tr>
<tr>
<td>2nd bit</td>
<td>1</td>
<td>1 \times 2^1 = 2</td>
<td></td>
</tr>
<tr>
<td>LSB</td>
<td>1</td>
<td>1 \times 2^0 = 1</td>
<td></td>
</tr>
</tbody>
</table>

Result: \[1010111₂ = 87_{10}\]
Binary to Hexadecimal Conversion:

- The conversion from an integer binary number to hexadecimal is accomplished by:
  - Breaking the binary number into 4-bit sections from the LSB to MSB;
  - Converting the 4-bit binary number to its hexadecimal equivalent.

Example 3: Convert the binary value $101011110110010_2$ to hexadecimal notation.

<table>
<thead>
<tr>
<th>1010</th>
<th>1111</th>
<th>1011</th>
<th>0010</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>F</td>
<td>B</td>
<td>2</td>
</tr>
</tbody>
</table>

Result: $101011110110010_2 = 0xAFB2$
Hexadecimal to Binary Conversion:

- The conversion from an integer hexadecimal number to binary is accomplished by:
  - Converting the hexadecimal number to its 4-bit binary equivalent;
  - Combining the 4-bit sections by removing the spaces.

**Example 4:** Convert the hexadecimal value 0x0AFB2 to binary notation.

<table>
<thead>
<tr>
<th>Hexadecimal:</th>
<th>A</th>
<th>F</th>
<th>B</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary:</td>
<td>1010</td>
<td>1111</td>
<td>1011</td>
<td>0010</td>
</tr>
</tbody>
</table>

Result: 0xAFB2 = 1010111110110010₂
Hexadecimal to Decimal Conversion:

- To convert from Hexadecimal to Decimal, multiply the value in each position by its hexadecimal weight and add each value;

- **Example 5:** Convert the hexadecimal number 0x0AFB2 to decimal notation.

\[
\begin{align*}
\text{A} \times 16^3 & \quad \text{F} \times 16^2 & \quad \text{B} \times 16^1 & \quad 2 \times 16^0 \\
10 \times 4096 & \quad 15 \times 256 & \quad 11 \times 16 & \quad 2 \times 1 \\
40960 & \quad + & \quad 3840 & \quad + & \quad 176 & \quad + & \quad 2 & = & 44978_{10}
\end{align*}
\]

Result: 0x0AFB2 = 44978_{10}
Decimal to Hexadecimal Conversion (Repeated Division by 16):

Example 6: Convert the decimal number $44978_{10}$ to hexadecimal notation.

<table>
<thead>
<tr>
<th>Division</th>
<th>Quotient</th>
<th>Remainder</th>
<th>Hexadecimal n.º</th>
</tr>
</thead>
<tbody>
<tr>
<td>44978/16</td>
<td>2811</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2811/16</td>
<td>175</td>
<td>11</td>
<td>B2</td>
</tr>
<tr>
<td>175/16</td>
<td>10</td>
<td>15</td>
<td>FB2</td>
</tr>
<tr>
<td>10/16</td>
<td>0</td>
<td>10</td>
<td>0AFB2</td>
</tr>
</tbody>
</table>

Result: $44978_{10} = 0x0AFB2$
Example:

- Sum of two binary numbers:
  
  \[
  \begin{array}{c}
  01011101_2 \\
  + \quad 00010110_2 \\
  \hline
  01110011_2
  \end{array}
  \]

- Multiplication of two binary numbers:
  
  \[
  \begin{array}{c}
  01011101_2 \\
  \times \quad 0110_2 \\
  \hline
  00000000 \\
  01011101 \\
  01011101 \\
  + \quad 00000000 \\
  \hline
  0100010111_2
  \end{array}
  \]
Negative numbers representation:
- One’s complement notation;
- Two’s complement notation.

One’s complement:
- Complement each bit of the number represented;
- Example: $0001\ 0100_2$;
- One’s complement $\rightarrow 1110\ 1011_2$;
- MSB = 1, means that the representation corresponds to a negative number;
- However, this representation cannot be used directly to carry out mathematical operations – it gives wrong results.
Two’s complement:

- Perform one’s complement and then add 1 to the result;

- The task "subtract two numbers" has been transformed into "add the complement" - so a total different operation 'ADD' instead of 'SUB' is used.

Example: 0001 0100₂;

Two’s complement → 1110 1011 + 1 = 1110 1100₂;

Example: Subtraction operation:

0101 1101₂ - 0001 0100₂

0101 1101₂
+ 1110 1100₂ <- Two’s complement representation
0100 1001₂

(Please note: MSB =0)

Copyright 2009 Texas Instruments
All Rights Reserved
www.msp430.ubi.pt
- **Two’s complement:**
  - Eliminates the double representation of the zero value;
  - Allows addition and subtraction operations to be carried out without need to take the sign of the operands into account;
  - Example: Subtraction operation:

\[
1010 \ 0011_2 \ - \ 0001 \ 0110_2
\]

\[
1010 \ 0011_2 \\
+ \ 1110 \ 1010_2 \\
\underline{1000 \ 1101_2}
\]

<- Two’s complement
(Negative result: MSB = 1)
Representation of fractional numbers:

Two different ways:
- Fixed-point;
- Floating-point.

Fixed-point representation:
- Allows us to represent the fractional part;
- Consumes less processing resources than floating-point;
Representation of fractional numbers (continued):

- **Binary base signed numbers representation:**

  - In the sign and absolute value notation:
    - MSB is reserved to the sign (0: positive; 1: negative);
    - Magnitude of the value is represented by the others bits;
    - Value of zero has two representations.

- **Fixed point representation notation: signed numbers:**

<table>
<thead>
<tr>
<th>Binary value</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign and absolute value</td>
<td>+0</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>-0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>One’s complement</td>
<td>+0</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>-0</td>
</tr>
<tr>
<td>Two’s complement</td>
<td>+0</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>
Representation of fractional numbers (continued):

- **Fixed-point numerical representation:**
  - Easily implemented within a short memory space;
  - Rapid to implement;
  - Suited to real-time applications that do not have a floating point coprocessor.

- A fixed point integer value, \( A \), with \( n \) bits, is given by the notation \( (U_{Q_p}.Q) \):
  - \( U \): unsigned (no sign bit)
  - \( p + q = n \) identifies an unsigned number
  - \( p \) bits: integer component;
  - \( q \) bits: fractional component.

- Limits: \( 0 \leq a_u \leq \frac{2^{p+q} - 1}{2^q} \)
- Resolution: \( r = 2^{-q} \)
Floating and Fixed-point Arithmetic (4/15)

- Representation of fractional numbers (continued):
  - Fixed point numerical representation:

<table>
<thead>
<tr>
<th>Format</th>
<th>Minimum</th>
<th>Maximum</th>
<th>r</th>
<th>n</th>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>(UQ16.)</td>
<td>0</td>
<td>$2^{16}-1$</td>
<td>1</td>
<td>16</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>(UQ.16)</td>
<td>0</td>
<td>$1-2^{-16}$</td>
<td>$2^{-16}$</td>
<td>16</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>(Q15.)</td>
<td>$-2^{15}$</td>
<td>$2^{15}-1$</td>
<td>1</td>
<td>16</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>(Q.15)</td>
<td>-1</td>
<td>$1-2^{-15}$</td>
<td>$2^{-15}$</td>
<td>16</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>(UQ16.16)</td>
<td>0</td>
<td>$2^{16}-1$</td>
<td>$2^{-16}$</td>
<td>32</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>(Q15.16)</td>
<td>$-2^{15}$</td>
<td>$2^{15}-2^{-16}$</td>
<td>$2^{-16}$</td>
<td>32</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>
Representation of fractional numbers: Fixed-point

Example 1:
Convert the decimal number \(123.045_{10}\) into hexadecimal.

- Decimal representation: \(123.045_{10}\)
- Integer component: \(123_{10} = 7B_{16}\)
- Fractional component: \(0.045_{10} = 0.0B8_{16}\)
- Result: \(123.045_{10} = 7B.0B8_{16}\)
Example 2:
Convert the decimal number $8751.135_{10}$ into a hexadecimal representation.

- Integer component: $8751_{10} = 222F_{16}$
- Fractional component: $0.135_{10} = 0.228_{16}$
- Result: $8751.135_{10} = 222F.228_{16}$
Representation of fractional numbers: Floating-point

- In floating-point number representation, the radix point position can float relatively to the significant digits of the number;

- This detail allows the support of a much wider range of values when compared with the fixed point number representation;

- Before 1958 IEEE 754 standard, the floating-point representation of numerical values was made through different formats and word widths;

- Nowadays, the IEEE 754 standard is accepted almost by all kind of computers.
Representation of fractional numbers: Floating-point (cont.)

- From the set of formats supported by IEEE 754 standard, the most widely used are the single-precision (32-bit) and double-precision (64-bit);

- Moreover, the standard also establishes others kinds of formats: "quad" binary and decimal floating-point (128-bit) and double decimal floating-point (64-bit);

- Less used are the extended precision (80-bit) and half-precision (16-bit) formats. The last one appears in the IEEE 754r proposed revision;

- Standard current version is IEEE 754-2008, which was published in August 2008.
Representation of fractional numbers: Floating-point (cont.)

- In the data structure of floating-point numbers, the most significant bit is the sign bit (S), followed by the exponent that is biased;
- The mantissa without the most significant bit is stored after the exponent in the fraction field:

<table>
<thead>
<tr>
<th></th>
<th>Exponent</th>
<th>Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- The exponent is biased by the \((2^e - 1) - 1\), where \(e\) is the exponent number of bits;
- This solution was adopted because exponents must be signed, and the two's complement representation will require extra computational work if it was used.
Representation of fractional numbers: Floating-point (cont.)

- The value of the biased exponent and mantissa determines the data meaning:
  - The number is said to be *normalized* (most significant bit of the mantissa is 1), if exponent ranges between 0 and $2^e - 1$;
  - The number is said to be *de-normalized* (most significant bit of the mantissa is 0), if the exponent is 0 and fraction is not 0. Most important de-normalized numbers:
    - $+0.0$: 0x0000;
    - $-0.0$: 0x8000;
Representation of fractional numbers: Floating-point (cont.)

- The value of the biased exponent and the mantissa determines the data meaning:

  - The number is ±0 (depending on the sign bit), if exponent is 0 and fraction is 0;
  
  - The number is ±infinity (depending on the sign bit), if exponent = \(2^e - 1\) and fraction is 0;
  
  - The number being represented is not a number (NaN), if exponent = \(2^e - 1\) and fraction is not 0.
Representation of fractional numbers: Floating-point (cont.)

- The IEEE 754 binary formats have the following organization:

<table>
<thead>
<tr>
<th>Format</th>
<th>Sign</th>
<th>Exponent</th>
<th>Exponent bias</th>
<th>Mantissa</th>
<th>Total number of bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half</td>
<td>1</td>
<td>5</td>
<td>15</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>Single</td>
<td>1</td>
<td>8</td>
<td>127</td>
<td>23</td>
<td>32</td>
</tr>
<tr>
<td>Double</td>
<td>1</td>
<td>11</td>
<td>1023</td>
<td>52</td>
<td>64</td>
</tr>
<tr>
<td>Quad</td>
<td>1</td>
<td>15</td>
<td>16383</td>
<td>112</td>
<td>128</td>
</tr>
</tbody>
</table>

- A number $X$ has the value:
  $$X = (-1)^S \times 2^{\text{exponent} - \text{exponent bias}} \times \text{mantissa}$$
Representation of fractional numbers: Floating-point (cont.)

- **Example 3:** Consider the decimal number $14.2352_{10}$.

  - The number integer part can be converted to the binary value $1110_2$, while the fractional number part is converted to the binary value $0.00111100001010001111_2$.

  - The number can be represented in the binary base as:

    $1110.00111100001010001111_2$
Representation of fractional numbers: Floating-point (cont.)

- The number normalization requires that the radix is moved to the left:
  \[ 1.110001111000010100011112 \times 2^3 \]

- We can now code the floating-point representation in the single format (32-bit) as:
  - Sign (1-bit): 0 (the number is positive);
  - Exponent (8-bit): \( 127 + 3 = 130 = 10000010 \);
  - Fraction (23-bit): 11000111100001010001111 (fractional part of the mantissa).
Representation of fractional numbers: Floating-point (cont.)

- Finally, the floating-point code:

<table>
<thead>
<tr>
<th>Sign</th>
<th>Exponent</th>
<th>Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10000010</td>
<td>11000111100001010001111</td>
</tr>
</tbody>
</table>

- The floating-point code in hexadecimal: \(0\times4163C28F\)
Manufacturers of electronic components provide datasheets containing the specifications detailing the part/device characteristics;

Datasheets give the electrical characteristics of the device and the pin-out functions, but without detailing the internal operation;

More complex devices are provided with documents that aid the development of applications, such as:
- Application notes;
- User's guides;
- Designer's guides;
- Package drawings, etc...
- **Datasheets include information:**
  - Concerning the part number of the device or device series;

  - Electrical characteristics:
    - Maximum and minimum operating voltages;
    - Operating temperature range e.g. 0 to 70 degrees C;
    - Output drive capacity for each output pin, as well as an overall limit for the entire chip;
    - Clock frequencies;
    - Pin out electrical characteristics (capacitance, inductance, resistance);
    - Noise tolerance or the noise generated by the device itself;
    - Physical tolerances...
MSP430 device datasheet:
- Device has a large number of peripherals;
- Each input/output pin usually has more than one function;
- It has a table with the description of each pin function;
- Example, Pin number 2 = P6.3/A3;
  - Digital Input/Output Port 6 bit 3;
  - 3rd analogue input.
MSP430 User’s Guide:
- Most peripherals are represented by Block Diagrams.

Example: Part of the MSP430F44x clock module block diagram:
MSP430 User’s Guide:

- Example: Part of the MSP430F44x clock module block diagram:
  Detail SELMx;

- Multiplexed block: (4 inputs and 1 output):
  - SELMx = 00: Output routed to the 1st multiplexer output;
  - SELMx = 01: Output routed to the second one and so on;
  - SELMx is a 2-bit mnemonic: SELM1 (MSB), SELM0 (LSB).

- To use the peripheral, it is necessary to find out how the register(s) need to be configured:
  - SELMx is in the FLL_CTL1 register.
### MSP430 User’s Guide:

- **SELMx** are the 3rd and 4th bits of FLL_CTL1 control register.

#### FLL_CTL1, FLL+ control register 1

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>SMCLKOFF</th>
<th>XT2OFF</th>
<th>SELMx</th>
<th>SELS</th>
<th>FLL_DIVx</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Disable the submain clock signal (SMCLK):</td>
<td></td>
<td></td>
<td>SELMx</td>
<td>SELS</td>
<td>FLL_DIVx</td>
</tr>
<tr>
<td>5</td>
<td>Disable the second crystal oscillator (XT2):</td>
<td></td>
<td></td>
<td>SELMx</td>
<td>SELS</td>
<td>FLL_DIVx</td>
</tr>
<tr>
<td>4-3</td>
<td>Select the master clock (MCLK) source:</td>
<td></td>
<td></td>
<td>SELMx</td>
<td>SELS</td>
<td>FLL_DIVx</td>
</tr>
<tr>
<td>2</td>
<td>Select the submain clock (SMCLK) source:</td>
<td></td>
<td></td>
<td>SELMx</td>
<td>SELS</td>
<td>FLL_DIVx</td>
</tr>
<tr>
<td>1-0</td>
<td>Select the auxiliary clock (ACLK) signal divider:</td>
<td></td>
<td></td>
<td>SELMx</td>
<td>SELS</td>
<td>FLL_DIVx</td>
</tr>
</tbody>
</table>

- **SMCLKOFF**
  - Disable the submain clock signal (SMCLK):
    - SMCLKOFF = 0 ➞ SMCLK active
    - SMCLKOFF = 1 ➞ SMCLK inactive

- **XT2OFF**
  - Disable the second crystal oscillator (XT2):
    - XT2OFF = 0 ➞ XT2 active
    - XT2OFF = 1 ➞ XT2 inactive

- **SELMx**
  - 00 ➞ DCO
  - 01 ➞ DCO
  - 10 ➞ XT2
  - 11 ➞ LFXT1

- **SELS**
  - 0 ➞ DCO
  - 1 ➞ XT2

- **FLL_DIVx**
  - 00 ➞ Divider factor: /1
  - 01 ➞ Divider factor: /2
  - 10 ➞ Divider factor: /4
  - 11 ➞ Divider factor: /8
C coding tips:

- Use local variable as much as possible (Local variables use CPU registers whereas global variables use RAM);

- Use unsigned data types where possible;

- Use pointers to access structures and unions;

- Use "static const" class to avoid run-time copying of structures, unions, and arrays;

- Avoid modulo;

- Count down “for” loops.
Good software practices for low power consumption (2/2)

- **Principles for low power applications:**
  - Maximize the time in standby;
  - Use interrupts to control program flow;
  - Replace software functions with peripheral hardware;
  - Manage the power of internal peripherals;
  - Manage the power of external devices;
  - Device choice can make a difference;
  - Effective code is a must. Every unnecessary instruction executed is a portion of the battery wasted that will never return.
Main characteristics of the MSP430 Integrated Development Environments (IDEs);

Available both from TI and third parties;

Special attention will be given to:
- Code Composer Essentials v3;
- IAR Embedded Workbench (EWB) IDE.

Using an IDE:
- Basic functions;
- Step by step project development;
  - Structure and management (source files; compiling, assembling and linking operations) of projects developed both in C and/or Assembly language.
A range of software tools are available for the generation of MSP430 source code:

- Code Composer Essentials (TI)
- IAR EWB - Kickstart ed. (IAR Systems);
- CrossStudio (Rowley Associates);
- MSPGCC (*open-source community*)...
- SwiftX (Forth, Inc.);
- HI-TECH (HI-TECH software);
- ANSI C (ImageCraft);
- Project-430 (Phyton, Inc.);
- AQ430 (Quadravox);
- ...

...
TI’s MSP430 IDE:

- It is available as:
  - A free upgrade for existing v2 users;
  - Professional version ($499), the main features being:
    - Unlimited code size;
    - Can be ordered from the MSP430 web page;
    - Supported by TI Software Support.
  - Evaluation Version (Free):
    - 16 kB Limit on C/ASM code size;
    - Download from MSP430 web page;
    - Supported by TI Software Support.
New features include:

- Free 16 kB code-limited version;
- Support for large memory model (Place data >64k);
- Enhanced Compatibility with IAR C-code:
  \#pragma (ISR declarations), most intrinsics;
- GDB Debugger replaced by TI proprietary debugger that allows faster single stepping;
- Hardware Multiplier libraries (16 bits and 32 bits);
- CCE v2 project support (auto convert);
- Breakpoints:
  - EEM support via unified breakpoint manager;
  - Use of Enhanced Emulation Module (EEM), with predefined Use Cases;
  - Unlimited Breakpoints.
Starter kit: Experimenter’s board

- **Features:**
  - MSP430F2013;
  - MSP430F5438;
  - Compatible with TI’s wireless evaluation modules.

- Combining 2 MCUs provides nearly every MSP430 peripherals available.

- Needs a MSP-FET430 for development
All 14 input/output pins on the MSP430F2013 are accessible on the MSP-EZ430D target board for easy debugging and interfacing to peripherals;

One of these input/output pins is connected to an LED for visual feedback;

Device features and integrated peripherals:
- 16-MIPS performance;
- 16-bit Sigma Delta ADC;
- 16-bit timer;
- Watchdog timer;
- Brownout detector;
- USI module supporting SPI and I2C;
- 5 low power modes (0.5 μA standby).
MSP-EXP430F5438:
- MSP430F5438.

New features:
- Power Management Module (PMM);
- Unified Clock System (UCS);
- System (SYS) modules;
- Expanded memory/peripheral mapping
- Peripheral module enhancements.

Enhanced performance
- 20 bit address capability;
- 32 bit Hardware Multiplier.

This board provides the wide range of F5438 peripherals.
Device features and integrated peripherals

- Device: MSP430F5438:
  - 256 kB + 512 kB flash memory; 16 kB RAM.

- Integrated peripherals:
  - Three 16-bit timers;
  - 12-bit SAR Analogue-to-Digital Converter;
  - Direct Memory Access (DMA);
  - Hardware multiplier (supporting 32-Bit operations);
  - Universal Serial Communication Interfaces (USCI): Enhanced UART Supporting Auto-Baudrate; IrDA Encoder and Decoder; Synchronous SPI; I2C™;
  - Real time clock module with alarm capabilities;
  - Temperature sensor;
  - Up to 87 I/O pins.
The MSP430F5438 supports I2C and SPI protocols using the USCI and the USI peripherals;

This protocol is used for inter-processor communication;

The link can be disconnected in hardware allowing these peripherals to be used for other communication purposes.

- Programming and Debugging:
  - Can be programmed using any MSP430 Flash Emulation Tool (MSP-FET430xIF);
- Wireless expansion:
  - Compatible with TI Wireless CCxxxXEMK Evaluation Modules, such as the CC2500EMK.
  - Compatible with TI eZ-RF2500.
The demo board has various system clock options that support low and high frequencies.

The MSP430F5438 has integrated an Unified Clock System that provides different clock sources:

- Three low-frequency sources:
  - LFXT1;
  - Internal Very Low Power/Low Frequency Oscillator (VLO);
  - Internal Reference Oscillator (REFO).
- Internal Digitally Controlled Oscillator (DCO) / Frequency Locked Loop (FLL) for highspeed operation:
  - FLL reference selectable from LFXT1, REFO, or XT2.
- ACLK/SMCLK/MCLK can all be driven from any source;
- Dedicated MODOSC (internal) used for modules like Flash controller, ADC, among others.
MSP-EXP430F5438 demo board jumper and connectors locations:

- **Board Power Supply Switch:**
  - FET: Power board from FET interface
  - USB: Power board from USB interface
  - BATT: Power board from Battery Pack

- **System Power:**
  - Place to connect Vcc to one of the power sources (FET, USB, BATT). Use also to measure total board current consumption.

- **RF Power:**
  - Place to power the RF connector headers.

- **JTAG:**
  - FET Tool Connector to Program / debug the MSP430F5438

- **E2PROM Connection:**
  - Only used for production programming of the USB E2PROM

- **JP4 – 5438 to USB3410 Connection:**
  - 1-3: UCAITXD / SIN
  - 3-4: UCAIRXD / SOUT

- **CC EMK Headers:**
  - SPI interface to CC EMK modules

- **EZRF Target Header:**
  - SPI interface to EZRF’s CC2500

- **410 PWR:**
  - Place to power MSP430F5438.
  - Use also to measure current

- **Volume:**
  - Connects to microcontroller output voltage level
Wireless expansion (Chipcon’s RF transceiver chip)

- **CC2500EMK Evaluation Module 2.4 GHz:**
  - The CC2500EM evaluation modules are provided with antennas;
  - These evaluation modules are add-on daughter boards that require a CC2500 development kit for evaluation and development;
  - It allows to do range testing (PER testing) and transfer data from one PC to another using the SmartRF®04DK, in order to evaluate how well the SmartRF®04 products fit the intended application;
  - It allows performing RF measurements.
The flash emulation tool (FET) allow the application development on the MSP430 MCU;

There are available two debugging interfaces:
- USB port: MSP-FET430UIF;
- Parallel port: MSP-FET430PIF.

MSP-FET430UIF flash emulation tool:
Are used to program and debug the MSP430 in-system through the:

- 4-wire JTAG interface: MSP-FET430PIF and MSP-FET430UIF;
- 2-wire JTAG interface (Spy Bi-Wire): MSP-FET430UIF.

These debugging tool interface the previously presented MSP430 hardware development tools to the included integrated software environment (CCE or IAR) and includes code to start an application.

Both MSP-FET430 supports development with all MSP430 flash devices.