E91

A/D

(Chapter 9 in text)

Example code at:
http://focus.ti.com/mcu/docs/mcuprodcodeexamples.tsp?sectionId=96&tabId=1468
MSP430FG4618 has 12 bit A/D

- Measures voltages between $V_{R-}$ and $V_{R+}$
- Uses 12 bit successive-approximation

$$N_{ADC} = 4095 \times \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}}$$

- There are 12 inputs (including)
  - $V_{eRef+}$
  - $V_{ref-}$
  - $V_{cc}/2$
  - Temperature

- There is an internal voltage reference.
VR- is either
• Vss (Ground) or
• VREF-/V_eREF-

VR+ is either
• VREF+ (from ref generator),
• V_eREF+ (from off chip), or
• Vcc

**Reference Selection**

**Internal Sources**
- Vcc/2
- Temp Sensor:
  Channel 1010_{binary} = 0x0a = 10_{decimal}

V_{REF+} is turned on with “REFON” and either 1.5 or 2.5 V depending on REF2_5V

**Internal Reference**
Conversion clock selection

Clock for ADC can be on of several inputs. ADC12OSC is an internal clock at about 5 MHz.

Conversion start

Conversion can be started by ADC12SC (Start Conversion) or by one of several timers.
Sample and Hold

SHP=0, Extended Sample Mode

SHP=1, Pulse Sample Mode
E91 I/O

```c
#include "msp430xG46x.h"

volatile unsigned int i;

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog
    P2SEL = 0;
    P2OUT = 0;
    P2DIR = 0x06; //LED's
    P3SEL = 0;
    P3OUT = BIT5;
    P3DIR = BIT5 + BIT6; //Pot
    P6SEL |= BIT5; // Enable A/D channel A5
    ADC12CTL0 = ADC12ON + SHT0_2; // Turn on A/D, set samp time
    ADC12CTL1 = SHP; // Use sampling timer
    ADC12MCTL0 = SREF_0 + INCH_5; // Vr+=Vcc, Vr-=Vss, INCH = 5
    ADC12CTL0 |= ENC; // Enable conversions
    while (1)
    {
        ADC12CTL0 |= ADC12SC; // Start conversions
        while (!(ADC12IFG & 0x0001)); // Conversion done?
        if (ADC12MEM0 > 0x0800) P2OUT = 0x02; // Check result, and light LED's
        else P2OUT = 0x04;
        // P2OUT = (ADC12MEMO > 0x0800 ? 0x02 : 0x04); (condition ? if :else)
    }
}
```

I figured it out!
## ADC12 Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Short Form</th>
<th>Register Type</th>
<th>Address</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC12 control register 0</td>
<td>ADC12CTL0</td>
<td>Read/write</td>
<td>01A0h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>ADC12 control register 1</td>
<td>ADC12CTL1</td>
<td>Read/write</td>
<td>01A2h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>ADC12 interrupt flag register</td>
<td>ADC12IFG</td>
<td>Read/write</td>
<td>01A4h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>ADC12 interrupt enable register</td>
<td>ADC12IE</td>
<td>Read/write</td>
<td>01A6h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>ADC12 interrupt vector word</td>
<td>ADC12IV</td>
<td>Read</td>
<td>01A8h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>ADC12 memory 0</td>
<td>ADC12MEM0</td>
<td>Read/write</td>
<td>0140h</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12 memory 1</td>
<td>ADC12MEM1</td>
<td>Read/write</td>
<td>0142h</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12 memory 2</td>
<td>ADC12MEM2</td>
<td>Read/write</td>
<td>0144h</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12 memory 3</td>
<td>ADC12MEM3</td>
<td>Read/write</td>
<td>0146h</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12 memory 4</td>
<td>ADC12MEM4</td>
<td>Read/write</td>
<td>0148h</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12 memory 5</td>
<td>ADC12MEM5</td>
<td>Read/write</td>
<td>014Ah</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12 memory 6</td>
<td>ADC12MEM6</td>
<td>Read/write</td>
<td>014Ch</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12 memory 7</td>
<td>ADC12MEM7</td>
<td>Read/write</td>
<td>0150h</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12 memory 8</td>
<td>ADC12MEM8</td>
<td>Read/write</td>
<td>0152h</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12 memory 9</td>
<td>ADC12MEM9</td>
<td>Read/write</td>
<td>0154h</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12 memory 10</td>
<td>ADC12MEM10</td>
<td>Read/write</td>
<td>0156h</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12 memory 11</td>
<td>ADC12MEM11</td>
<td>Read/write</td>
<td>0158h</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12 memory 12</td>
<td>ADC12MEM12</td>
<td>Read/write</td>
<td>015Ah</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12 memory 13</td>
<td>ADC12MEM13</td>
<td>Read/write</td>
<td>015Ch</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12 memory 14</td>
<td>ADC12MEM14</td>
<td>Read/write</td>
<td>015Dh</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12 memory 15</td>
<td>ADC12MEM15</td>
<td>Read/write</td>
<td>015Fh</td>
<td>Unchanged</td>
</tr>
<tr>
<td>ADC12CTL0</td>
<td>ADC12MEMx, ADC12 Conversion Memory Registers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC12CTL0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC12CMEMx</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC12MEMMCTLx</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC12IE</td>
<td>ADC12IE15</td>
<td>ADC12IE14</td>
<td>ADC12IE13</td>
<td>ADC12IE12</td>
</tr>
<tr>
<td>--------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADC12IE7</th>
<th>ADC12IE6</th>
<th>ADC12IE5</th>
<th>ADC12IE4</th>
<th>ADC12IE3</th>
<th>ADC12IE2</th>
<th>ADC12IE1</th>
<th>ADC12IE0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADC12IFG</th>
<th>ADC12IFG15</th>
<th>ADC12IFG14</th>
<th>ADC12IFG13</th>
<th>ADC12IFG12</th>
<th>ADC12IFG11</th>
<th>ADC12IFG10</th>
<th>ADC12IFG9</th>
<th>ADC12IFG8</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw-(U)</td>
<td>rw-(U)</td>
<td>rw-(U)</td>
<td>rw-(U)</td>
<td>rw-(U)</td>
<td>rw-(U)</td>
<td>rw-(U)</td>
<td>rw-(U)</td>
<td>rw-(U)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADC12IFG7</th>
<th>ADC12IFG6</th>
<th>ADC12IFG5</th>
<th>ADC12IFG4</th>
<th>ADC12IFG3</th>
<th>ADC12IFG2</th>
<th>ADC12IFG1</th>
<th>ADC12IFG0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADC12IV</th>
<th>ADC12IVx</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r0</td>
<td>r0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r0</th>
<th>r0</th>
<th>r-(0)</th>
<th>r-(0)</th>
<th>r-(0)</th>
<th>r-(0)</th>
<th>r-(0)</th>
<th>r-(0)</th>
<th>r0</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Conversion Memory and Modes

• There are 16 ADC12MEMx memory registers to store conversion results.
• Each is configured with an associated ADC12MCTLx control register.
  • The SREFx bits define the voltage reference,
  • The INCHx bits select the input channel.
  • The EOS bit defines the end of sequence when sequential conversion is used.
• For repeat conversion modes, CSTARTADDx points to the first ADC12MCTLx location.
• A pointers is incremented to the next ADC12MCTLx.
• The sequence continues until an EOS bit in ADC12MCTLx is processed.
• When conversion results are written to a ADC12MEMx, the corresponding flag in the ADC12IFGx register is set.
E91 I/O, with more detail

Channel 5 is input

P6SEL |= BIT5; // Enable A/D channel A5

ADC12CTL0 = ADC12ON + SHT0_2; // Turn on A/D, set samp time
ADC12CTL1 = SHP; // Use sampling timer
ADC12MCTL0 = SREF_0 + INCH_5; // Vr+=Vcc, Vr-=Vss, INCH = 5

ADC12CTL0 |= ENC; // Enable conversions

while (1)
{
    ADC12CTL0 |= ADC12SC; // Start conversions
    while (!(ADC12IFG & 0x0001)); // Conversion done?
    if (ADC12MEM0 > 0x0800) P2OUT = 0x02; // Check result, and light LED's
    else P2OUT = 0x04;
}

ADC12CLK is 5Mhz internal clock by default, Sample time is 16 ADC12CLK cycles

Pulse Sample mode

Only one channel (5)

Enable Conversions

Start Conversions

Interrupt flag is set (but interrupt not enabled)

If result (in memory register) is big, light one LED, else light the other)
void main(void) {
    WDTCTL = WDTPW + WDTHOLD; // Stop WDT
    ADC12CTL0 = SHT0_2 + ADC12ON; // Sampling time, ADC12 on
    ADC12CTL1 = SHP; // Use sampling timer
    ADC12IE = 0x01; // Enable interrupt
    ADC12CTL0 |= ENC; // P6.0 ADC option select
    P6SEL |= 0x01;
    P5DIR |= 0x02; // P5.1 output

    while (1)
    {
        ADC12CTL0 |= ADC12SC; // Start sampling/conversion
        __bis_SR_register(LPM0_bits + GIE); // LPM0, ADC12_ISR will force exit
    }
}

#pragma vector = ADC12_VECTOR
__interrupt void ADC12_ISR(void) {
    if (ADC12MEM0 >= 0x7ff) // ADC12MEM = A0 > 0.5AVcc?
        P5OUT |= 0x02; // P5.1 = 1
    else
        P5OUT &= ~0x02; // P5.1 = 0

    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
## Consecutive conversions

<table>
<thead>
<tr>
<th>CONSEQx</th>
<th>Mode</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Single channel</td>
<td>A single channel is converted once.</td>
</tr>
<tr>
<td></td>
<td>single-conversion</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>Sequence-of-channels</td>
<td>A sequence of channels is converted once.</td>
</tr>
<tr>
<td>10</td>
<td>Repeat-single-channel</td>
<td>A single channel is converted repeatedly.</td>
</tr>
<tr>
<td>11</td>
<td>Repeat-sequence-of-channels</td>
<td>A sequence of channels is converted repeatedly.</td>
</tr>
</tbody>
</table>
ISR and Consecutive Conversions

volatile unsigned int Results[2];

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
    ADC12CTL0 = ADC12ON + MSC + SHT0_15; // Turn on ADC12, set sampling time
    ADC12CTL1 = SHP + CONSEQ_1; // Use sampling timer, single seq
    ADC12MCTL0 = INCH_8; // ref+=AVcc, chan = A8
    ADC12MCTL1 = INCH_9 + EOS; // ref+=AVcc, chan = A9, end seq
    ADC12IE = 0x02; // Enable ADC12IFG.1
    ADC12CTL0 |= ENC; // Enable conversions
    __enable_interrupt(); // Enable interrupts

    while(1)
    {
        ADC12CTL0 |= ADC12SC; // Start conversion
        __bis_SR_register(LPM0_bits); // Enter LPM0
    }
}
#pragma vector=ADC12_VECTOR
__interrupt void ADC12ISR (void)
{
    Results[0] = ADC12MEM0; // Move results, IFG is cleared
    Results[1] = ADC12MEM1; // Move results, IFG is cleared
    __no_operation(); // SET BREAKPOINT HERE
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}
#include "msp430xG46x.h"
#define ADC_DELTA_ON 12 // ~ 2 Deg C delta for LED on
unsigned int ADCResult;

void main(void) {
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog
    ADC12CTL1 = SHS_1 + SHP + CONSEQ_2; // TA trig., rpt conv
    ADC12MCTL0 = SREF_1 + INCH_10; // Channel A10, Vref+
    ADC12IE |= 0x001; // Enable ADC12IFG.0
    ADC12CTL0 = SHT0_8 + REF2_5V + REFIN + ADC12ON; // Config ADC12
    TACCR0 = 13600; // Delay for reference start-up
    TACCTL0 |= CCIE; // Compare-mode interrupt.
    TACTL = TACLR + MC_1 + TASSEL_2; // up mode, SMCLK
    __bis_SR_register(LPM0_bits + GIE); // Enter LPM0, Enable interrupts
    TACCTL0 &= ~CCIE; // Disable timer interrupt
    ADC12CTL0 |= ENC; // Config ADC12
    TACCTL1 = OUTMOD_4; // Toggle on EQU1 (TAR = 0)
    TACTL = TASSEL_2 + MC_2; // SMCLK, cont-mode
    while (!(ADC12IFG & 0x0001)); // First conversion?
    ADCResult = ADC12MEM0; // Read out 1st ADC value
    ADCResult += ADC_DELTA_ON;
    P5OUT = 0; // Clear P5
    P5DIR |= 0x02;
    __bis_SR_register(LPM0_bits + GIE); // LPM0
}

#pragma vector = TIMERA0_VECTOR
__interrupt void TA0_ISR(void) {
    TACCTL = 0; // Clear Timer_A control registers
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}

#pragma vector = ADC12_VECTOR
__interrupt void ADC12_ISR(void) {
    if (ADC12MEM0 >= ADCResult) // ADC12MEM = A0 > ADCResult?
        P5OUT |= 0x02; // P5.1 = 1
    else
        P5OUT &= ~0x02; // P5.1 = 0
}
Efficient Interrupt Handling in ASM

ADC12IV, ADC12 Interrupt Vector Register

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
<td>r0</td>
</tr>
</tbody>
</table>

ADC12IVx Bits 15-0 ADC12 interrupt vector value

<table>
<thead>
<tr>
<th>ADC12IV Contents</th>
<th>Interrupt Source</th>
<th>Interrupt Flag</th>
<th>Interrupt Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>000h</td>
<td>No interrupt pending</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>002h</td>
<td>ADC12MEM0 overflow</td>
<td>-</td>
<td>Highest</td>
</tr>
<tr>
<td>004h</td>
<td>Conversion time overflow</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>006h</td>
<td>ADC12MEM0 interrupt flag</td>
<td>ADC12FG0</td>
<td></td>
</tr>
<tr>
<td>008h</td>
<td>ADC12MEM1 interrupt flag</td>
<td>ADC12FG1</td>
<td></td>
</tr>
<tr>
<td>00Ah</td>
<td>ADC12MEM2 interrupt flag</td>
<td>ADC12FG2</td>
<td></td>
</tr>
<tr>
<td>00Ch</td>
<td>ADC12MEM3 interrupt flag</td>
<td>ADC12FG3</td>
<td></td>
</tr>
<tr>
<td>00Eh</td>
<td>ADC12MEM4 interrupt flag</td>
<td>ADC12FG4</td>
<td></td>
</tr>
<tr>
<td>010h</td>
<td>ADC12MEM5 interrupt flag</td>
<td>ADC12FG5</td>
<td></td>
</tr>
<tr>
<td>012h</td>
<td>ADC12MEM6 interrupt flag</td>
<td>ADC12FG6</td>
<td></td>
</tr>
<tr>
<td>014h</td>
<td>ADC12MEM7 interrupt flag</td>
<td>ADC12FG7</td>
<td></td>
</tr>
<tr>
<td>016h</td>
<td>ADC12MEM8 interrupt flag</td>
<td>ADC12FG8</td>
<td></td>
</tr>
<tr>
<td>018h</td>
<td>ADC12MEM9 interrupt flag</td>
<td>ADC12FG9</td>
<td></td>
</tr>
<tr>
<td>01Ah</td>
<td>ADC12MEM10 interrupt flag</td>
<td>ADC12FG10</td>
<td></td>
</tr>
<tr>
<td>01Ch</td>
<td>ADC12MEM11 interrupt flag</td>
<td>ADC12FG11</td>
<td></td>
</tr>
<tr>
<td>01Eh</td>
<td>ADC12MEM12 interrupt flag</td>
<td>ADC12FG12</td>
<td></td>
</tr>
<tr>
<td>020h</td>
<td>ADC12MEM13 interrupt flag</td>
<td>ADC12FG13</td>
<td></td>
</tr>
<tr>
<td>022h</td>
<td>ADC12MEM14 interrupt flag</td>
<td>ADC12FG14</td>
<td></td>
</tr>
<tr>
<td>024h</td>
<td>ADC12MEM15 interrupt flag</td>
<td>ADC12FG15</td>
<td>Lowest</td>
</tr>
</tbody>
</table>

; Interrupt handler for ADC12.
INT_ADC12 ; Enter Interrupt Service Routine
ADD &ADC12IV,PC; Add offset to PC
RETI ; Vector 0: No interrupt
JMP ADOV ; Vector 2: ADC overflow
JMP ADTOV ; Vector 4: ADC timing overflow
JMP ADM0 ; Vector 6: ADCMEM0 overflow
... ; Vectors 8-32
JMP ADM14 ; Vector 34: ADC12IFG14

; Handler for ADC12IFG15 starts here. No JMP required.
; ADM15 MOV &ADC12MEM15,xxx; Move result, flag is reset
; ... ; Other instruction needed?
; JMP INT_ADC12 ; Check other int pending

; ADC12IFG14-ADC12IFG1 handlers go here
; ADM0 MOV &ADC12MEM0,xxx ; Move result, flag is reset
; ... ; Other instruction needed?
RETI ; Return

; ADTOV ... ; Handle Conv. time overflow
RETI ; Return

; ADOV ... ; Handle ADCMEMx overflow
RETI ; Return
When writing code – don’t start from scratch – check out the Example code link on the “Resources” page of the class web page.