Analog to Digital Conversion

EE3376
The ADC10 module is a high-performance 10-bit analog-to-digital converter. This chapter describes the operation of the ADC10 module of the 2xx family in general. There are device with less than eight external input channels.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.1 ADC10 Introduction</td>
<td>548</td>
</tr>
<tr>
<td>22.2 ADC10 Operation</td>
<td>550</td>
</tr>
<tr>
<td>22.3 ADC10 Registers</td>
<td>556</td>
</tr>
</tbody>
</table>
Analog to Digital Conversion

Sampling period:

Analog Input

5V = 65

-5V = -65

Digital version:

| 56 | -40 | -65 | -41 | 60 | 55 | -45 | -62 | -22 | 50 |

This is the essence of how CDs or MP3s store music. Stream of digital values represents an audio (analog) waveform.

Nyquist Frequency = twice the frequency of the data and is required for sampling to correctly represent an analog signal. MP3s typically sampled at 128kps (period of ~ 8 microseconds)
Sampling Theory – Nyquist Freq

- **a.** Analog frequency = 0.0 (i.e., DC)
- **b.** Analog frequency = 0.09 of sampling rate
- **c.** Analog frequency = 0.31 of sampling rate
- **d.** Analog frequency = 0.95 of sampling rate
Quantization Noise
DACs are a little less complicated and often used in ADCs. The simplest DAC is a PWM with a low pass filter. Here is an R-2R DAC – fast but high power.
<table>
<thead>
<tr>
<th>Pick This Architecture if you want:</th>
<th>FLASH (Parallel)</th>
<th>SAR</th>
<th>DUAL SLOPE (Integrating ADC)</th>
<th>PIPELINE</th>
<th>SIGMA DELTA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Conversion Method</strong></td>
<td>N bits - 2^N-1 Comparators</td>
<td>Medium to high resolution (8 to 16bit), 5MSPs and under, low power, small size.</td>
<td>Monitoring DC signals, high resolution, low power consumption, good noise performance ICL7106.</td>
<td>High speeds, few Msps to 100+ Msps, 8 bits to 16 bits, lower power consumption than flash.</td>
<td>High resolution, low to medium speed, no precision external components, simultaneous 50/60Hz rejection, digital filter reduces anti-aliasing requirements.</td>
</tr>
<tr>
<td><strong>Encoding Method</strong></td>
<td>Thermometer Code Encoding</td>
<td>Binary search algorithm, internal circuitry runs higher speed.</td>
<td>Unknown input voltage is integrated and value compared against known reference value.</td>
<td>Small parallel structure, each stage works on one to a few bits.</td>
<td>Oversampling ADC, 5-Hz - 60Hz rejection programmable data output.</td>
</tr>
<tr>
<td><strong>Disadvantages</strong></td>
<td>Sparkle codes / metastability, high power consumption, large size, expensive.</td>
<td>Speed limited to ~5MSPs. May require anti-aliasing filter.</td>
<td>Slow Conversion rate. High precision external components required to achieve accuracy.</td>
<td>Parallelism increases throughput at the expense of power and latency.</td>
<td>Higher order (4th order or higher) - multibit ADC and multibit feedback DAC.</td>
</tr>
<tr>
<td><strong>Conversion Time</strong></td>
<td>Conversion Time does not change with increased resolution.</td>
<td>Increases linearly with increased resolution.</td>
<td>Conversion time doubles with every bit increase in resolution.</td>
<td>Increases linearly with increased resolution.</td>
<td>Tradeoff between data output rate and noise free resolution.</td>
</tr>
<tr>
<td><strong>Resolution</strong></td>
<td>Component matching typically limits resolution to 8 bits.</td>
<td>Component matching requirements double with every bit increase in resolution.</td>
<td>Component matching requirements double with every bit increase in resolution.</td>
<td>Component matching requirements double with every bit increase in resolution.</td>
<td>Component matching requirements double with every bit increase in resolution.</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>2^N-1 comparators, Die size and power increases exponentially with resolution.</td>
<td>Die increases linearly with increase in resolution.</td>
<td>Core die size will not materially change with increase in resolution.</td>
<td>Die increases linearly with increase in resolution.</td>
<td>Core die size will not materially change with increase in resolution.</td>
</tr>
</tbody>
</table>
A/D circuit – simplified flash a/d

V_{\text{ref}}

8 to 3 Encoder

3

%100 = 4
digital output

Index of highest true input is the output

analog Input

11/16 V_{\text{ref}}

thermometer code

0

1

1

0

1

1

1
Ramp ADC

- Analog input
- Comparator
- DAC output
- Start pulse
- Sample time
- EOC
- Comparator output
- Digital to Analog Conv.
- Counter
- Clock
- Reset
- After Tocci, Digital Systems

Explanation:
- Start pulse resets the counter and blocks clock during reset. At end of start pulse, the counter starts.
- When the digital ramp output of the DAC reaches the signal voltage $V_s$, the comparator goes low, stopping the count.
- Counter increases voltage out of DAC until it reaches the input sample voltage $V_s$. 
Successive Approximation ADC

Number of bits $\approx$ number of cycles for conversion
The ADC10 module is a high-performance 10-bit analog-to-digital converter. This chapter describes the operation of the ADC10 module of the 2xx family in general. There are device with less than eight external input channels.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.1 ADC10 Introduction</td>
<td>548</td>
</tr>
<tr>
<td>22.2 ADC10 Operation</td>
<td>550</td>
</tr>
<tr>
<td>22.3 ADC10 Registers</td>
<td>566</td>
</tr>
</tbody>
</table>
ADC10 Overview

Figure 22-1. ADC10 Block Diagram

†Channels A12-A15 are available in MSP430F22xx devices only. Channels A12-A15 tied to channel A11 in other devices. Not all channels are available in all devices.
‡TA1 on MSP430F20x2, MSP430G2x31, and MSP430G2x30 devices
22.2.5 Sample and Conversion Timing

An analog-to-digital conversion is initiated with a rising edge of sample input signal SHI. The source for SHI is selected with the SHSx bits and includes the following:

- The ADC10SC bit
- The Timer_A Output Unit 1
- The Timer_A Output Unit 0
- The Timer_A Output Unit 2

Max sample period = $13 \text{ ADC10CLK} + T_{\text{sample}} + T_{\text{sync}}$

The polarity of the SHI signal source can be inverted with the ISSH bit. The SHTx bits select the sample period $t_{\text{sample}}$ to be 4, 8, 16, or 64 ADC10CLK cycles. The sampling timer sets SAMPCON high for the selected sample period after synchronization with ADC10CLK. Total sampling time is $t_{\text{sample}}$ plus $t_{\text{sync}}$. The high-to-low SAMPCON transition starts the analog-to-digital conversion, which requires 13 ADC10CLK cycles as shown in Figure 22-3.

![Sample Timing Diagram](image-url)
## ADC10 Control Register 0

### 22.3.1 ADC10CTL0, ADC10 Control Register 0

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>SREFx</td>
<td>ADC10SHTx</td>
<td>ADC10SR</td>
<td>REFOUT</td>
<td>REFBURST</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>
<td>rw-(0)</td>
<td></td>
<td></td>
</tr>
<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>
</tr>
<tr>
<td>MSC</td>
<td>REF2,5V</td>
<td>REFON</td>
<td>ADC10ON</td>
<td>ADC10IE</td>
<td>ADC10IFG</td>
<td>ENC</td>
<td>ADC10SC</td>
</tr>
<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>

Can be modified only when ENC = 0

**SREFx** Bits 15-13 Select reference.
- 000 $V_{RX} = V_{CC}$ and $V_{RX} = V_{SS}$
- 001 $V_{RX} = V_{REF+}$ and $V_{RX} = V_{SS}$
- 010 $V_{RX} = V_{REF+}$ and $V_{RX} = V_{REF-}$. Devices with $V_{REF+}$ only.
- 011 $V_{RX} = \text{Buffered } V_{\text{REF+}}$, and $V_{RX} = V_{SS}$. Devices with $V_{\text{REF+}}$, pin only.
- 100 $V_{RX} = V_{CC}$ and $V_{RX} = V_{REF+}/V_{REF-}$. Devices with $V_{\text{REF+}}$, pin only.
- 101 $V_{RX} = V_{REF+}$ and $V_{RX} = V_{REF+}/V_{REF-}$. Devices with $V_{\text{REF+}}$, pins only.
- 110 $V_{RX} = V_{REF+}$ and $V_{RX} = V_{REF+}/V_{REF-}$. Devices with $V_{\text{REF+}}$, pins only.
- 111 $V_{RX} = \text{Buffered } V_{\text{REF+}}$, and $V_{RX} = V_{REF+}/V_{REF-}$. Devices with $V_{\text{REF+}}$, pins only.

**ADC10SHTx** Bits 12-11 ADC10 sample-and-hold time
- 00 4 × ADC10CLKs
- 01 8 × ADC10CLKs
- 10 16 × ADC10CLKs
- 11 64 × ADC10CLKs

**ADC10SR** Bit 10 ADC10 sampling rate. This bit selects the reference buffer drive capability for the maximum sampling rate. Setting ADC10SR reduces the current consumption of the reference buffer.
- 0 Reference buffer supports up to ~200 kspks
- 1 Reference buffer supports up to ~50 kspks

**REFOUT** Bit 9 Reference output
- 0 Reference output off
- 1 Reference output on. Devices with $V_{\text{REF+}}/V_{\text{REF-}}$, pin only.
<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 8</td>
<td>Reference burst. Whether reference buffer is on continuously.</td>
</tr>
<tr>
<td>0</td>
<td>Reference buffer on continuously.</td>
</tr>
<tr>
<td>1</td>
<td>Reference buffer on only during sample-and-conversion.</td>
</tr>
<tr>
<td>Bit 7</td>
<td>Multiple sample and conversion. Valid only for sequence or repeated modes.</td>
</tr>
<tr>
<td>0</td>
<td>The sampling requires a rising edge of the SHI signal to trigger each sample-and-conversion.</td>
</tr>
<tr>
<td>1</td>
<td>The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior conversion is completed.</td>
</tr>
<tr>
<td>Bit 6</td>
<td>Reference-generator voltage. REFON must also be set.</td>
</tr>
<tr>
<td>0</td>
<td>1.5 V</td>
</tr>
<tr>
<td>1</td>
<td>2.5 V</td>
</tr>
<tr>
<td>Bit 5</td>
<td>Reference generator on.</td>
</tr>
<tr>
<td>0</td>
<td>Reference off.</td>
</tr>
<tr>
<td>1</td>
<td>Reference on.</td>
</tr>
<tr>
<td>Bit 4</td>
<td>ADC10 on.</td>
</tr>
<tr>
<td>0</td>
<td>ADC10 off.</td>
</tr>
<tr>
<td>1</td>
<td>ADC10 on.</td>
</tr>
<tr>
<td>Bit 3</td>
<td>ADC10 interrupt enable.</td>
</tr>
<tr>
<td>0</td>
<td>Interrupt disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Interrupt enabled.</td>
</tr>
<tr>
<td>Bit 2</td>
<td>ADC10 interrupt flag. This bit is set if ADC10MEM is loaded with a conversion result. It is automatically reset when the interrupt request is accepted, or it may be reset by software. When using the DTC this flag is set when a block of transfers is completed.</td>
</tr>
<tr>
<td>0</td>
<td>No interrupt pending.</td>
</tr>
<tr>
<td>1</td>
<td>Interrupt pending.</td>
</tr>
<tr>
<td>Bit 1</td>
<td>Enable conversion.</td>
</tr>
<tr>
<td>0</td>
<td>ADC10 disabled.</td>
</tr>
<tr>
<td>1</td>
<td>ADC10 enabled.</td>
</tr>
<tr>
<td>Bit 0</td>
<td>Start conversion. Software-controlled sample-and-conversion start. ADC10SC and ENC may be set together with one instruction. ADC10SC is reset automatically.</td>
</tr>
<tr>
<td>0</td>
<td>No sample-and-conversion start.</td>
</tr>
<tr>
<td>1</td>
<td>Start sample-and-conversion.</td>
</tr>
</tbody>
</table>
### ADC10 Control Register 1

#### 22.3.2 ADC10CTL1, ADC10 Control Register 1

<table>
<thead>
<tr>
<th></th>
<th>INCHx</th>
<th>SHSx</th>
<th>ADC10DF</th>
<th>ISSH</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
</tr>
<tr>
<td>14</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
</tr>
<tr>
<td>13</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
</tr>
<tr>
<td>12</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
</tr>
<tr>
<td>11</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
</tr>
<tr>
<td>10</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
</tr>
<tr>
<td>9</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>r-0</td>
</tr>
<tr>
<td>8</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
</tr>
</tbody>
</table>

#### ADC10DIVx

<table>
<thead>
<tr>
<th></th>
<th>ADC10SSSELx</th>
<th>CONSEQx</th>
<th>ADC10BUSY</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
</tr>
<tr>
<td>6</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
</tr>
<tr>
<td>5</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
</tr>
<tr>
<td>4</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
</tr>
<tr>
<td>3</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
<td>rw-(0)</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**INCHx**

Bits 15-12

Input channel select. These bits select the channel for a single-conversion or the highest channel for a sequence of conversions. Only available ADC channels should be selected. See device specific datasheet.

- 0000 A0
- 0001 A1
- 0010 A2
- 0011 A3
- 0100 A4
- 0101 A5
- 0110 A6
- 0111 A7
- 1000 $V_{\text{REF+}}$
- 1001 $V_{\text{REF-}}$
- 1010 Temperature sensor
- 1011 $(V_{\text{CC}} - V_{\text{SS}}) / 2$
- 1100 $(V_{\text{CC}} - V_{\text{SS}}) / 2$, A12 on MSP430F22xx devices
- 1101 $(V_{\text{CC}} - V_{\text{SS}}) / 2$, A13 on MSP430F22xx devices
- 1110 $(V_{\text{CC}} - V_{\text{SS}}) / 2$, A14 on MSP430F22xx devices
- 1111 $(V_{\text{CC}} - V_{\text{SS}}) / 2$, A15 on MSP430F22xx devices

**SHSx**

Bits 11-10

Sample-and-hold source select.

- 00 ADC10SC bit
- 01 Timer_A.OUT1
- 10 Timer_A.OUT0
- 11 Timer_A.OUT2 (Timer_A.OUT1 on MSP430F20x0, MSP430G2x31, and MSP430G2x30 devices)

**ADC10DF**

Bit 9

ADC10 data format

- 0 Straight binary
- 1 2s complement

**ISSH**

Bit 8

Invert signal sample-and-hold

- 0 The sample-input signal is not inverted.
- 1 The sample-input signal is inverted.
### ADC10 Control Register 1 (cont)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC10DIVx</td>
<td>Bits 7-5</td>
<td>ADC10 clock divider</td>
</tr>
<tr>
<td>000</td>
<td>/1</td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>/2</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>/3</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>/4</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>/5</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>/6</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>/7</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>/8</td>
<td></td>
</tr>
<tr>
<td>ADC10SSELx</td>
<td>Bits 4-3</td>
<td>ADC10 clock source select</td>
</tr>
<tr>
<td>00</td>
<td>ADC10OSC</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>ACLK</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>MCLK</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>SMCLK</td>
<td></td>
</tr>
<tr>
<td>CONSEQx</td>
<td>Bits 2-1</td>
<td>Conversion sequence mode select</td>
</tr>
<tr>
<td>00</td>
<td>Single-channel-single-conversion</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>Sequence-of-channels</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Repeat-single-channel</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Repeat-sequence-of-channels</td>
<td></td>
</tr>
<tr>
<td>ADC10BUSY</td>
<td>Bit 0</td>
<td>ADC10 busy. This bit indicates an active sample or conversion operation</td>
</tr>
<tr>
<td>0</td>
<td>No operation is active</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>A sequence, sample, or conversion is active</td>
<td></td>
</tr>
</tbody>
</table>
ADC Lab

\[ X_\text{\textunderscore AXIS} = 0, \ Y_\text{\textunderscore AXIS} = 1, \ Z_\text{\textunderscore AXIS} = 2 \]

```c
void adc_init()
// Initialize ADC
{
    PINDIR &= ~(1 << X_AXIS) + (1 << Y_AXIS) + (1 << Z_AXIS);  // Set Pins as inputs

    ADC10CTL0 = ADC10ON + ADC10SHT_0 + SREF_0;       // configure ADC module
    ADC10CTL1 = CONSEQ_0 + ADC10SSEL_0 + ADC10DIV_0 + SHS_0 + INCH_0;
    ADC10AE0 = (1 << X_AXIS) + (1 << Y_AXIS) + (1 << Z_AXIS);   // Enable analog function in pins
}

int adc_measureAll(void)
// Return the sum of all ADC channels
{
    return (adc_measure(X_AXIS) + adc_measure(Y_AXIS) + adc_measure(Z_AXIS));
}

int adc_measure(int channel)
// measure selected ADC Channel: X_AXIS, Y_AXIS, Z_AXIS
{
    ADC10CTL0 &= ~ENC;       // Disable ADC
    ADC10CTL1 = CONSEQ_0 + ADC10SSEL_0 + ADC10DIV_0 + SHS_0 + (channel << 12);  // Configure the MUX channel to sample
    ADC10CTL0 |= ENC + ADC10SC;  // Enable ADC and start conversion

    while ((ADC10CTL0 & ADC10IFG) == 0);  // Wait until the conversion is finished

    ADC10CTL0 &= ~ENC;       // Disable ADC before retrieving the conversion from memory

    return ADC10MEM;        // Return measurement value
}
```
msp430x22x4_adc10_01

; *********************************************************************************************************************
;  MSP430F22x4 Demo - ADC10, Sample A0, AVcc Ref, Set P1.0 if A0 > 0.5*AVcc
;  
; Description: A single sample is made on A0 with reference to AVcc.
; Software sets ADC10SC to start sample and conversion – ADC10SC
; automatically cleared at EOC. ADC10 internal oscillator times sample (16x) and conversion. In Mainloop MSP430 waits in LPM0 to save power until ADC10
; conversion complete, ADC10_ISR will force exit from LPM0 in Mainloop on reti. If A0 > 0.5*AVcc, P1.0 set, else reset.
; 
; 
;    MSP430F22x4
;    ______________________
;   
;   /\    XIN
;  |   |
;  | RST
; | XOUT
; >---|P2.0/A0  P1.0--- LED
#include "msp430x22x4.h"

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;       // Stop WDT
    ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; // ADC10ON, interrupt enabled
    ADC10AE0 |= 0x01;               // P2.0 ADC option select
    P1DIR |= 0x01;                  // Set P1.0 to output direction

    for (; ;)
    {
        ADC10CTL0 |= ENC + ADC10SC; // Sampling and conversion start
        __bis_SR_register(CPUOFF + GIE);
        if (ADC10MEM < 0x1FF)
            P1OUT &= ~0x01;
        else
            P1OUT |= 0x01;          // Set P1.0 LED on
    }
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
    __bic_SR_register_on_exit(CPUOFF); // Clear CPUOFF bit from 0(SR)
;--------------------------------------------------------------------------
 .text
 ;--------------------------------------------------------------------------

RESET       mov.w #300h,SP    ; Initialize stack pointer
StopWDT     mov.w #WDTPW+WDTHOLD,&WDTCTL   ; Stop WDT
SetupADC10  mov.w #ADC10SHT_2+ADC10ON+ADC10IE,&ADC10CTL0 ; 16x, enable int.
             bis.b #01h,&ADC10AE0    ; P2.0 ADC10 option select
SetupP1     bis.b #001h,&P1DIR    ; P1.0 output
Mainloop    bis.w #ENC+ADC10SC,&ADC10CTL0 ; Start sampling/conversion
             bis.w #CPUOFF+GIE,SR    ; LPM0, ADC10_ISR will force exit
             bic.b #01h,&P1OUT        ; P1.0 = 0
             cmp.w #01FFh,&ADC10MEM   ; ADC10MEM = A0 > 0.5Vcc?
             jlo     Mainloop         ; Again
             bis.b #01h,&P1OUT        ; P1.0 = 1
             jmp     Mainloop         ; Again

ADC10_ISR;  Exit LPM0 on reti

ADC10_ISR; Exit LPM0 on reti

bic.w #CPUOFF,0(SP)         ; Exit LPM0 on reti
reti

;--------------------------------------------------------------------------
| Interrupt Vectors |
;--------------------------------------------------------------------------

.sect   "reset"                ; MSP430 RESET Vector
.short  RESET
.sect   "int05"                ; ADC10 Vector
.short  int05
.end    ADC10_ISR
msp430x22x4_adc10_10

;**************************************************************************************
; MSP430F22x4 Demo – ADC10, DTC Sample A2–0, AVcc, Single Sequence, DCO
;
; Description: Sample A2/A1/A0 as single sequence with reference to AVcc.
; Software sets ADC10SC to trigger sample sequence. In Mainloop MSP430 waits
; in LPM0 to save power until ADC10 conversion complete, ADC10_ISR(DTC) will
; force exit from any LPMx in Mainloop on reti. ADC10_ISR will force LPM0
; exit. ADC10 internal oscillator times sample period (16x) and conversion
; (13x). DTC transfers conversion code to RAM 200h – 206h. P1.0 set at start
; of conversion burst, reset on completion.
;
; MSP430F22x4
; -----------------
; /\|\| XIN|-
; | | |
; --|RST XOUT|-
; | |
; --->|P2.2/A2 P1.0-->LED
; --->|P2.1/A1 |
; --->|P2.0/A0 |
```assembly
.RESET   mov.w #300h,SP       ; Initialize stack pointer
StopWDT mov.w #WDTPW+WDTHOLD,&WDTCR ; Stop WDT
SetupADC10 mov.w #INCH_2+CONSEQ_1,&ADC10CTL1 ; A2/A1/A0, single sequence
          mov.w #ADC10SHT_2+MSC+ADC10ON+ADC10IE,&ADC10CTL0 ;
          bis.b #07h,&ADC10AE0       ; P2.0,1,2 ADC10 option selects
          mov.b #03h,&ADC10DTC1      ; 3 conversions
SetupPI1  mov.b #001h,&P1DIR ; P1.0 output
Mainloop  bic.w #ENC,&ADC10CTL0 ;
busy_test bit #BUSY,&ADC10CTL1 ; ADC10 core inactive?
jnz busy_test ;
mov.w #0200h,&ADC10SA       ; Data buffer start
bis.b #001h,&P1OUT          ; P1.0 = 0
bis.w #ENC+ADC10SC,&ADC10CTL0 ; Start sampling
bis.w #CPUOFF+GIE,SR       ; LPM0, ADC10_ISR will force exit
bic.b #001h,&P1OUT          ; P1.0 = 1
jmp Mainloop ; Again

;-----------------------------------------------
ADC10_ISR; Exit LPM0 on reti
;-----------------------------------------------
    bic.w #CPUOFF,0(SP) ; Exit LPM0 on reti
    reti
;
;-----------------------------------------------
; Interrupt Vectors
;-----------------------------------------------
.sect "reset" ; MSP430 RESET Vector
.short RESET
.sect "int05" ; ADC10 Vector
.short ADC10_ISR
.end
```
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop WDT
    ADC10CTL1 = INCH_2 + CONSEQ_1; // A2/A1/A0, single sequence
    ADC10CTL0 = ADC10SHT_2 + MSC + ADC10ON + ADC10IE;
    ADC10DTC1 = 0x03; // 3 conversions
    ADC10AE0 |= 0x07; // P2.2,1,0 ADC10 option select
    P1DIR |= 0x01; // Set P1.0 output

    for (;;)
    {
        ADC10CTL0 &= ~ENC; // Wait if ADC10 core is active
        while (ADC10CTL1 & BUSY);
        ADC10SA = 0x200; // Data buffer start
        P1OUT |= 0x01; // P1.0 = 1
        ADC10CTL0 |= ENC + ADC10SC; // Sampling and conversion start
        __bis_SR_register(CPUOFF + GIE);
        P1OUT &~ 0x01;
    }
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
    __bic_SR_register_on_exit(CPUOFF); // Clear CPUOFF bit from 0(SR)
msp430x22x4_adc10_03

;******************************************************************************;
; MSP430F22x4 Demo - ADC10, Sample A10 Temp, Set P1.0 if Temp ++ ~2C
;
; Description: Set ADC10 and the integrated temperature sensor to detect
; temperature gradients. The temperature sensor output voltage is sampled
; ~ every 120ms and compared with the defined delta values using an ISR.
; (ADC10OSC/4)/64 determines sample time which needs to be greater than
; 30us for temperature sensor.
; ADC10 is operated in repeat-single channel mode with the sample and
; convert trigger sourced from Timer_A CCR1. The ADC10IFG at the end
; of each conversion will trigger an ISR.
; ACLK = n/a, MCLK = SMCLK = default DCO ~1.2MHz, ADC10CLK = ADC10OSC
;
;---------------------------
;                     XIN|--
;                     |   |
; --| RST  XOUT|--
; |
; | A10 (Temp)  P1.0|-->LED
#include "msp430x22x4.h"

static unsigned int FirstADCVal;  // holds 1st ADC result
#define ADCDeltaOn 3  // ~ 2 Deg C delta for LED on

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;  // Stop watchdog
    ADC10CTL1 = ADC10DIV_3 + INCH_10 + SMS_1 + CONSEQ_2;  // TA trig., rpt, A10
    ADC10CTL0 = SREF_1 + ADC10SHT_3 + REF2_5V + ADC10IE + REFON + ADC10ON;
    TACCR0 = 30;  // Delay to allow Ref to settle
    TACCTL0 |= CCIE;  // Compare-mode interrupt
    TACCTL = TASSEL_2 + MC_1;
    __bis_SR_register(CPUOFF + GIE);
    TACCTL0 &= ~CCIE;
    ADC10CTL0 |= ENC;
    TACCTL1 = OUTMOD_4;
    TACCTL = TASSEL_2 + MC_2;
    while (!(ADC10IFG & ADC10CTL0));
    FirstADCVal = ADC10MEM;
    P1OUT = 0x00;
    P1DIR = 0x00;
    __bis_SR_register(LPM0_bits + GIE);
}

#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
    if (ADC10MEM >= FirstADCVal + ADCDeltaOn)  // LED on
        P1OUT |= 0x01;
    else
        P1OUT &= ~0x01;  // LED off
}

#pragma vector=TIMERA0_VECTOR
__interrupt void TA0_ISR(void)
{
    TACCTL = 0;
    LPM0_EXIT;  // Exit LPM0 on return
ADC Delta On

.text

; Program reset

RESET

mov.w #300h,SP ; Initialize stack pointer
mov.w #WDPW+WDTHOLD,&WDTCR ; Stop WDT
mov.w #ADC10DIV_3+INCH_10+SHS_1+CONSEQ_2,&ADC10CTL
mov.w #SREF_1+ADC10SHT_3+REF2_5V+ADC10IE+REFON+ADC10ON,&ADC10CTL0
mov.w #30,&TACCR0 ; Delay to allow Ref to settle
bis.w #CCIE,&TACCTL0 ; Compare-mode interrupt.
mov.w #TACLR+MC_1+TASSEL_2,&TACCTL; up mode, SMCLK
bis.w #LPM0+GIE,SR ; Enter LPM0, enable interrupts
bic.w #CCIE,&TACCTL0 ; Disable timer interrupt
bis.w #ENC,&ADC10CTL0 ;
mov.w #OUTMOD_4,&TACCTL1 ; Toggle on EQU1 (TAR = 0)
mov.w #TASSEL_2+MC_2,&TACCTL ; SMCLK, cont-mode

ADC Wait

bit.w #ADC10IFG,&ADC10CTL0 ; First conversion?

jnc ADC_Wait

mov.w &ADC10MEM,R4 ; Read out 1st ADC value
add.w &ADCDeltaOn,R4 ;
clr.b &P1OUT ; Clear P1
bis.b #01h,&P1DIR ; P1.0 as output

Mainloop

bis.w #LPM0+GIE,SR ; Enter LPM0, enable interrupts
nop ; Required only for debugger

ISR for TACCR0

clr.w &TACCTL ; Clear Timer_A control registers
bic.w #LPM0,0(SP) ; Exit LPM0 on reti

ADC10_ISR;

cmp.w R4,&ADC10MEM ; ADC10MEM = A10 > R4
jlo ADC_ISR_1 ; Again
bis.b #01h,&P1OUT ; P1.0 = 1

ADC_ISR_1

bic.b #01h,&P1OUT ; P1.0 = 0

Interrupt Vectors

.sect ".reset" ; MSP430 RESET Vector
.short RESET ;
.sect ".int05" ; ADC10 Vector
.short ADC10_ISR ;
.sect ".int09" ; Timer_A0 Vector
.sect TAC_ISR
msp430x22x4_adc10_13

;******************************
; MSP430F22x4 Demo – ADC10, DTC Sample A1 32x, AVcc, TA0 Trig, DCO
;
; Description; A1 is sampled in 32x burst using DTC 16 times per second
; (ACLK/2048) with reference to AVcc. Activity is interrupt driven.
; Timer_A in upmode uses TA0 toggle to drive ADC10 conversion. Sample burst
; is automatically triggered by TA0 rising edge every 2048 ACLK cycles.
; ADC10_ISR will exit from LPM3 mode and return CPU active. Internal ADC10OSC
; times sample (16x) and conversion (13x). DTC transfers conversion code to
; RAM 200h – 240h. In the Mainloop P1.0 is toggled. Normal Mode is LPM3.
; /* An external watch crystal on XIN XOUT is required for ACLK */
;
;-----------------------------
; /\\
; | \\
; | RST
; >-- P2.1/A1  P1.0  --> LED
; | 32kHz
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;   // Stop WDT
    ADC10CTL1 = INCH_1 + SHS_2 + CONSEQ_2; // TA0 trigger
    ADC10CTL0 = ADC10SHT_2 + MSC + ADC10ON + ADC10IE;
    ADC10DTC1 = 0x20;            // 32 conversions
    P1DIR |= 0x01;               // Set P1.0 output
    ADC10AE0 |= 0x02;            // P2.1 ADC10 option select
    TACCR0 = 1024-1;            // PWM Period
    TACCTL0 = OUTMOD_4;        // TACCR0 toggle
    TACTL = TASSEL_1 + MC_1;   // ACLK, up mode

    for (;;)
    {
        ADC10CTL0 &= ~ENC;        // Wait if ADC10 core is active
        while (ADC10CTL1 & BUSY); // Data buffer start
        ADC10SA = 0x200;          // Sampling and conversion ready
        ADC10CTL0 |= ENC;
        __bis_SR_register(LPM3_bits + GIE);
        P1OUT ^= 0x01;            // Enter LPM3, enable interrupts
        __cleardsr(LPM3_bits);   // Toggle P1.0 using exclusive-OR
    }
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
    __bic_SR_register_on_exit(LPM3_bits);       // Clear LPM3 bits from 0(SR)
```assembly
; Initialize stack pointer
mov.w  #300h,SP

; Stop WDT
mov.w  #WDT_PW+WDTHOLD,&WDTCTL

; TA0 trigger
mov.w  #INCH_1+SHS_2+CONSEQ_2,&ADC10CTL1

; TA0 trigger
mov.w  #ADC10SHT_2+MSC+ADC10ON+ADC10IE,&ADC10CTL0;

; 32 conversions
mov.b  #020h,&ADC10DTC1

; P1.0 output
bis.b  #001h,&P1DIR

; P2.1 ADC10 option select
bis.b  #002h,&ADC10AE0

; PWM Period
mov.w  #1024-1,&TACCR0

; TACCR0 toggle
mov.w  #OUTMOD_4,&TACCTL0

; ACLK, up mode
mov.w  #TASSEL_1+MC_1,TACCTL

; ADC10 core inactive?
bic.w  #ENC,&ADC10CTL0

; ADC10 core inactive?
bisz  BUSY,&ADC10CTL1

; Data buffer start
mov.w  #0200h,&ADC10SA

; Sampling and conversion ready
bic.w  #ENC,&ADC10CTL0

; LPM3, ADC10_ISR will force exit
bis.w  #LPM3+GIE,SR

; P1.0 = toggle
xor.b  #001h,&P1OUT

; Again
jmp    Mainloop

; Exit LPM3 on reti
bic.w  #LPM3,0(SP)

; Exit LPM3 on reti
reti

; Interrupt Vectors

; MSP430 RESET Vector
.sect  "reset"
.short  RESET

; ADC10 Vector
.sect  "int05"
.short  ADC10_ISR
.end
```

```bash
~
```
msp430x22x4_adc10_14

MSP430F22x4 Demo - ADC10, DTC Sample A1-A0 32x, AVcc, Repeat Seq, DCO

Description: Use DTC to sample A1/A0 repeat sequence 32x(64 total samples)
with reference to AVcc. Software sets ADC10SC to trigger sample burst.
In Mainloop MSP430 waits in LPM0 to save power until ADC10 conversion
complete, ADC10_ISR will force exit from LPM0 in Mainloop on reti.
ADC10 internal oscillator times sample period (16x) and conversion (13x).
DTC transfers conversion code to RAM 200h – 280h. ADC10(DTC) interrupt
will return system active. P1.0 set at start of conversion burst, reset
on completion.

MSP430F22x4
--------------
/\ | XIN -
| | |
-- RST XOUT -
| | |
>--- P2.1/A1 P1.0 -->LED
>--- P2.0/A0
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;       // Stop WDT
    ADC10CTL1 = INCH_1 + CONSEQ_3;  // A1/A0, repeat multi channel
    ADC10CTL0 = ADC10SHT_2 + MSC + ADC10ON + ADC10IE;
    ADC10AE0 = 0x03;                // P2.0,1 ADC option select
    ADC10DTC1 = 0x40;               // 64 conversions
    P1DIR |= 0x01;                   // Set P1.0 output

    for (;;);
    {
        P1OUT |= 0x01;                // Set P1.0 LED on
        ADC10CTL0 &= ~ENC;
        while (ADC10CTL1 & BUSY);
        ADC10SA = 0x200;              // Wait if ADC10 core is active
        ADC10CTL0 |= ENC + ADC10SC;   // Data buffer start
        __bis_SR_register(CPUOFF + GIE);
        P1OUT &= ~0x01;               // Sampling and conversion ready
        __bic_SR_register_on_exit(CPUOFF); // LPM0, ADC10_ISR will force exit
        // Clear P1.0 LED off
    }
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
    __bic_SR_register_on_exit(CPUOFF);       // Clear CPUOFF bit from 0(SR)
```assembly
RESET    mov.w  #300h,SP        ; Initialize stack pointer
StopWDT  mov.w  #WDTPW+WDTHOLD,&WDTCTL ; Stop WDT
SetupADC10 mov.w #INCH_1+CONSEQ_3,&ADC10CTL1 ; AI/A0, repeat multi channel
         mov.w  #ADC10SHT_2+MSC+ADC10ON+ADC10IE,&ADC10CTL0 ;
bis.b   #03h,&ADC10AE0 ; P2.0,1 ADC option select
         mov.b  #040h,&ADC10DTC1 ; 64 conversions
SetupP1  bis.b  #001h,&P1DIR ; P1.0 output
Mainloop bis.b  #001h,&P1OUT ; P1.0 = 1
busy_test bit    #BUSY,&ADC10CTL1 ; ADC10 core inactive?
jnz     busy_test
         mov.w  #0200h,&ADC10SA ; Data buffer start
         mov.w  #ENC+ADC10SC,&ADC10CTL0 ; Start sampling
         mov.w  #CPUOFF+GIE,SR ; CPU off, Enable interrupts
         bic.b  #001h,&P1OUT ; P1.0 = 0
         jmp    Mainloop ; Again

ADC10_ISR;

bic.w   #ENC,&ADC10CTL0 ; ADC10 disabled
bic.w   #LPM0,0(SP) ; Exit LPM0 on reti
reti

Interrupt Vectors

.sect    ".reset"       ; MSP430 RESET Vector
.short   RESET
.sect    ".int05"       ; ADC10 Vector
.short   ADC10_ISR
.end
```