EZ430 chronos 套件源码集合 II（7 个 ADC12，5 个 COMP_B）

ADC12;

cc430x613x_adc12_01.c ADC12, Sample A0, Set P1.0 if A0 > 0.5*AVcc01

```
    //*****************************************************************************
*****
02  //     C430F613x Demo - ADC12_A, Sample A0, Set P1.0 if A0 > 0.5*AVcc
03  //
04  //     Description: A single sample is made on A0 with reference to AVcc.
05  //     Software sets ADC12SC to start sample and conversion - ADC12SC
06  //     automatically cleared at EOC. ADC12 internal oscillator times sample (16x)
07  //     and conversion. In mainloop CC430 waits in LPM0 to save power until ADC12
08  //     conversion complete, ADC12_ISR will force exit from LPM0 in Mainloop on
09  //     reti. If A0 > 0.5*AVcc, P1.0 set, else reset.
10  //
11  //                         CC430F6137
12  //                     -----------------
13  //               /|\|                   |
14  //                | |                   |
15  //                --|RST                |
16  //                  |                   |
17  //       Vin -->|P2.0/A0          P1.0|--> LED
18  //
19  //     M Morales
20  //     Texas Instruments Inc.
21  //     April 2009
22  //     Built with CCE Version: 3.2.2 and IAR Embedded Workbench Version: 4.2
23
    //*****************************************************************************
*****
24
25  #include "cc430x613x.h"
26
27  void main(void)
28  {
29    WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
30    ADC12CTL0 = ADC12SHT02 + ADC12ON;         // Sampling time, ADC12 on
31    ADC12CTL1 = ADC12SHP;                     // Use sampling timer
32    ADC12IE = 0x01;                           // Enable interrupt
33    ADC12CTL0 |= ADC12ENC;
34
35    P2SEL |= BIT0;                            // P2.0 ADC option select
36    P1DIR |= BIT0;                            // P1.0 output
```

```
37
38      while (1)
39      {
40        ADC12CTL0 |= ADC12SC;                       // Start sampling/conversion
41
42        __bis_SR_register(LPM0_bits + GIE);       // LPM0, ADC12_ISR will force exit
43        __no_operation();                           // For debugger
44      }
45    }
46
47    #pragma vector = ADC12_VECTOR
48    __interrupt void ADC12_ISR(void)
49    {
50      switch(__even_in_range(ADC12IV,34))
51      {
52      case  0: break;                               // Vector  0:  No interrupt
53      case  2: break;                               // Vector  2:  ADC overflow
54      case  4: break;                               // Vector  4:  ADC timing overflow
55      case  6:                                      // Vector  6:  ADC12IFG0
56        if (ADC12MEM0 >= 0x7ff)                     // ADC12MEM = A0 > 0.5AVcc?
57          P1OUT |= BIT0;                            // P1.0 = 1
58        else
59          P1OUT &= ~BIT0;                           // P1.0 = 0
60
61        __bic_SR_register_on_exit(LPM0_bits);     // Exit active CPU
62      case  8: break;                               // Vector  8:  ADC12IFG1
63      case 10: break;                               // Vector 10:  ADC12IFG2
64      case 12: break;                               // Vector 12:  ADC12IFG3
65      case 14: break;                               // Vector 14:  ADC12IFG4
66      case 16: break;                               // Vector 16:  ADC12IFG5
67      case 18: break;                               // Vector 18:  ADC12IFG6
68      case 20: break;                               // Vector 20:  ADC12IFG7
69      case 22: break;                               // Vector 22:  ADC12IFG8
70      case 24: break;                               // Vector 24:  ADC12IFG9
71      case 26: break;                               // Vector 26:  ADC12IFG10
72      case 28: break;                               // Vector 28:  ADC12IFG11
73      case 30: break;                               // Vector 30:  ADC12IFG12
74      case 32: break;                               // Vector 32:  ADC12IFG13
75      case 34: break;                               // Vector 34:  ADC12IFG14
76      default: break;
77      }
78    }
```

```
 cc430x613x_adc12_02.c                    ADC12, Using the Internal Reference01
    //*****************************************************************************
*****
02  //  CC430F613x Demo - ADC12_A, Using the Internal Reference
03  //
04  //  Description: This example shows how to use the shared reference for ADC12
05  //  sampling and performs a single conversion on channel A0. The conversion
06  //  results are stored in ADC12MEM0. Test by applying a voltage to channel A0,
07  //  then setting and running to a break point at the "__no_operation()"
08  //  instruction. To view the conversion results, open an ADC12 register window
09  //  in debugger and view the contents of ADC12MEM0
10  //
11  //                        CC430F613x
12  //                    -----------------
13  //               /|\|                  |
14  //                | |                  |
15  //                --|RST               |
16  //                  |                  |
17  //      Vin -->|P2.0/A0                |
18  //                  |                  |
19
20  //
21  //  M. Morales
22  //  Texas Instruments Inc.
23  //  April 2009
24  //  Built with CCE Version: 3.2.2 and IAR Embedded Workbench Version: 4.11B
25
    //*****************************************************************************
*****
26
27  #include "cc430x613x.h"
28
29  void main(void)
30  {
31    WDTCTL = WDTPW+WDTHOLD;                     // Stop watchdog timer
32
33    P2SEL |= BIT0;                            // Enable A/D channel A0
34
35    /* Initialize REF module */
36    // Enable 2.5V shared reference, disable temperature sensor to save power
37    REFCTL0 |= REFMSTR+REFVSEL_2+REFON+REFTCOFF;
38
```

```
39      /* Initialize ADC12 */
40      ADC12CTL0 = ADC12ON+ADC12SHT02;                 // Turn on ADC12, set sampling
time
41      ADC12CTL1 = ADC12SHP;                           // Use sampling timer
42      ADC12MCTL0 = ADC12SREF_1;                       // Vr+=Vref+ and Vr-=AVss
43
44      __delay_cycles(75);                             // 75 us delay @ ~1MHz
45
46      ADC12CTL0 |= ADC12ENC;                          // Enable conversions
47
48      while (1)
49      {
50        ADC12CTL0 |= ADC12SC;                         // Start conversion
51        while (!(ADC12IFG & BIT0));
52        __no_operation();                             // SET BREAKPOINT HERE
53      }
54    }
```

```
 cc430x613x_adc12_05.c                          ADC12, Using an External Reference01
    //*****************************************************************************
*****
02  //  CC430F613x Demo - ADC12_A, Using the Internal Reference
03  //
04  //  Description: This example shows how to use the shared reference for ADC12
05  //  sampling and performs a single conversion on channel A0. The conversion
06  //  results are stored in ADC12MEM0. Test by applying a voltage to channel A0,
07  //  then setting and running to a break point at the "__no_operation()"
08  //  instruction. To view the conversion results, open an ADC12 register window
09  //  in debugger and view the contents of ADC12MEM0
10  //
11  //                        CC430F613x
12  //                    -----------------
13  //               /|\|                 |
14  //                | |                 |
15  //                --|RST              |
16  //                  |                 |
17  //        Vin -->|P2.0/A0              |
18  //                  |                 |
19
20  //
21  //    M. Morales
```

```
22  //     Texas Instruments Inc.
23  //     April 2009
24  //     Built with CCE Version: 3.2.2 and IAR Embedded Workbench Version: 4.11B
25
    //*****************************************************************************
*****
26
27  #include "cc430x613x.h"
28
29  void main(void)
30  {
31     WDTCTL = WDTPW+WDTHOLD;                          // Stop watchdog timer
32
33     P2SEL |= BIT0;                                   // Enable A/D channel A0
34
35     /* Initialize REF module */
36     // Enable 2.5V shared reference, disable temperature sensor to save power
37     REFCTL0 |= REFMSTR+REFVSEL_2+REFON+REFTCOFF;
38
39     /* Initialize ADC12 */
40     ADC12CTL0 = ADC12ON+ADC12SHT02;                  // Turn on ADC12, set sampling
time
41     ADC12CTL1 = ADC12SHP;                            // Use sampling timer
42     ADC12MCTL0 = ADC12SREF_1;                        // Vr+=Vref+ and Vr-=AVss
43
44     __delay_cycles(75);                              // 75 us delay @ ~1MHz
45
46     ADC12CTL0 |= ADC12ENC;                           // Enable conversions
47
48     while (1)
49     {
50        ADC12CTL0 |= ADC12SC;                         // Start conversion
51        while (!(ADC12IFG & BIT0));
52        __no_operation();                             // SET BREAKPOINT HERE
53     }
54  }
```

cc430x613x_adc12_06.c ADC12, Repeated Sequence of Conversions001

```
    //*****************************************************************************
*****
002 //  CC430F613x Demo - ADC12_A, Repeated Sequence of Conversions
```

```
003 //
004 //   Description: This example shows how to perform a repeated sequence of
005 //   conversions using "repeat sequence-of-channels" mode.    AVcc is used for the
006 //   reference and repeated sequence of conversions is performed on Channels A0,
007 //   A1, A2, and A3. Each conversion result is stored in ADC12MEM0, ADC12MEM1,
008 //   ADC12MEM2, and ADC12MEM3 respectively. After each sequence, the 4 conversion
009 //   results are moved to A0results[], A1results[], A2results[], and A3results[].
010 //   Test by applying voltages to channels A0 - A3. Open a watch window in
011 //   debugger and view the results. Set Breakpoint1 in the index increment line
012 //   to see the array values change sequentially and Breakpoint2 to see the entire
013 //   array of conversion results in A0results[], A1results[], A2results[], and
014 //   A3results[]for the specified Num_of_Results.
015 //
016 //   Note that a sequence has no restrictions on which channels are converted.
017 //   For example, a valid sequence could be A0, A3, A2, A4, A2, A1, A0, and A7.
018 //   See the CC430 User's Guide for instructions on using the ADC12_A.
019 //
020 //                        CC430F6137
021 //                    -----------------
022 //               /|\|                   |
023 //                | |                   |
024 //                --|RST                |
025 //                  |                   |
026 //      Vin0 -->|P2.0/A0                |
027 //      Vin1 -->|P2.1/A1                |
028 //      Vin2 -->|P2.2/A2                |
029 //      Vin3 -->|P2.3/A3                |
030 //                  |                   |
031 //
032 //    M. Morales
033 //    Texas Instruments Inc.
034 //    April 2009
035 //    Built with CCE Version: 3.2.2 and IAR Embedded Workbench Version: 4.11B
036
    //*****************************************************************************
*****
037
038 #include "cc430x613x.h"
039
040 #define     Num_of_Results     8
041
042 volatile unsigned int A0results[Num_of_Results];
043 volatile unsigned int A1results[Num_of_Results];
044 volatile unsigned int A2results[Num_of_Results];
```

```
045 volatile unsigned int A3results[Num_of_Results];
046
047 void main(void)
048 {
049   WDTCTL = WDTPW+WDTHOLD;                          // Stop watchdog timer
050
051   P2SEL = 0x0F;                                    // Enable A/D channel inputs
052
053   ADC12CTL0 = ADC12ON+ADC12MSC+ADC12SHT0_8; // Turn on ADC12_A, extend
sampling time
054                                                    // to avoid overflow of results
055   ADC12CTL1 = ADC12SHP+ADC12CONSEQ_3;          // Use sampling timer, repeated
sequence
056   ADC12MCTL0 = ADC12INCH_0;                        // ref+=AVcc, channel = A0
057   ADC12MCTL1 = ADC12INCH_1;                        // ref+=AVcc, channel = A1
058   ADC12MCTL2 = ADC12INCH_2;                        // ref+=AVcc, channel = A2
059   ADC12MCTL3 = ADC12INCH_3+ADC12EOS;           // ref+=AVcc, channel = A3,
end seq.
060   ADC12IE = 0x08;                                  // Enable ADC12IFG.3
061   ADC12CTL0 |= ADC12ENC;                           // Enable conversions
062   ADC12CTL0 |= ADC12SC;                            // Start conversion - software trigger
063
064   __bis_SR_register(LPM0_bits + GIE);          // Enter LPM0, Enable interrupts
065   __no_operation();                                // For debugger
066 }
067
068 #pragma vector=ADC12_VECTOR
069 __interrupt void ADC12ISR (void)
070 {
071   static unsigned int index = 0;
072
073   switch(__even_in_range(ADC12IV,34))
074   {
075   case  0: break;                                  // Vector  0:  No interrupt
076   case  2: break;                                  // Vector  2:  ADC overflow
077   case  4: break;                                  // Vector  4:  ADC timing overflow
078   case  6: break;                                  // Vector  6:  ADC12IFG0
079   case  8: break;                                  // Vector  8:  ADC12IFG1
080   case 10: break;                                 // Vector 10:  ADC12IFG2
081   case 12:                                         // Vector 12:  ADC12IFG3
082     A0results[index] = ADC12MEM0;                  // Move A0 results, IFG is cleared
083     A1results[index] = ADC12MEM1;                  // Move A1 results, IFG is cleared
084     A2results[index] = ADC12MEM2;                  // Move A2 results, IFG is cleared
085     A3results[index] = ADC12MEM3;                  // Move A3 results, IFG is cleared
```

```
086         index++;                                    // Increment results index, modulo; Set
Breakpoint1 here
087
088         if (index == 8)
089             index = 0;                              // Reset index, Set breakpoint 2 here
090
091     case 14: break;                                 // Vector 14:   ADC12IFG4
092     case 16: break;                                 // Vector 16:   ADC12IFG5
093     case 18: break;                                 // Vector 18:   ADC12IFG6
094     case 20: break;                                 // Vector 20:   ADC12IFG7
095     case 22: break;                                 // Vector 22:   ADC12IFG8
096     case 24: break;                                 // Vector 24:   ADC12IFG9
097     case 26: break;                                 // Vector 26:   ADC12IFG10
098     case 28: break;                                 // Vector 28:   ADC12IFG11
099     case 30: break;                                 // Vector 30:   ADC12IFG12
100     case 32: break;                                 // Vector 32:   ADC12IFG13
101     case 34: break;                                 // Vector 34:   ADC12IFG14
102     default: break;
103     }
104 }
```

cc430x613x_adc12_07.c ADC12, Repeated Single Channel Conversions01

```
    //*****************************************************************************
*****
02  //  CC430F613x Demo - ADC12_A, Repeated Single Channel Conversions
03  //
04  //  Description: This example shows how to perform repeated conversions on a
05  //  single channel using "repeat-single-channel" mode.    AVcc is used for the
06  //  reference and repeated conversions are performed on Channel A0. Each
07  //  conversion result is moved to an 8-element array called results[].    Test by
08  //  applying a voltage to channel A0, then running. Open a watch window in
09  //  debugger and view the results. Set Breakpoint1 in the index increment line
10  //  to see the array value change sequentially and Breakpoint to see the entire
11  //  array of conversion results in "results[]" for the specified Num_of_Results.
12  //  This can run even in LPM4 mode as ADC has its own clock (MODOSC)
13  //
14  //                      CC430F6137
15  //                  -----------------
16  //              /|\|                 |
17  //               | |                 |
18  //               --|RST              |
```

```
19  //                  |                       |
20  //        Vin -->|P2.0/A0                |
21  //                  |                       |
22  //
23  //
24  //     M Morales
25  //     Texas Instruments Inc.
26  //     April 2009
27  //     Built with CCE Version: 3.2.2 and IAR Embedded Workbench Version: 4.11B
28
    //*****************************************************************************
*****
29
30  #include "cc430x613x.h"
31
32  #define     Num_of_Results     8
33
34  volatile unsigned int results[Num_of_Results];
35                                                  // Needs to be global in this
36                                                  // example. Otherwise, the
37                                                  // compiler removes it because it
38                                                  // is not used for anything.
39
40  void main(void)
41  {
42    WDTCTL = WDTPW+WDTHOLD;                       // Stop watchdog timer
43
44    P2SEL |= 0x01;                                // Enable A/D channel A0
45
46    /* Initialize ADC12_A */
47    ADC12CTL0 = ADC12ON+ADC12SHT0_8+ADC12MSC; // Turn on ADC12, set
sampling time
48                                                  // set multiple sample conversion
49    ADC12CTL1 = ADC12SHP+ADC12CONSEQ_2;           // Use sampling timer, set
mode
50    ADC12IE = 0x01;                               // Enable ADC12IFG.0
51    ADC12CTL0 |= ADC12ENC;                        // Enable conversions
52    ADC12CTL0 |= ADC12SC;                         // Start conversion
53
54    __bis_SR_register(LPM4_bits + GIE);         // Enter LPM4, Enable interrupts
55    __no_operation();                             // For debugger
56  }
57
58  #pragma vector=ADC12_VECTOR
```

```
59  __interrupt void ADC12ISR (void)
60  {
61    static unsigned char index = 0;
62
63    switch(__even_in_range(ADC12IV,34))
64    {
65    case  0: break;                          // Vector  0:  No interrupt
66    case  2: break;                          // Vector  2:  ADC overflow
67    case  4: break;                          // Vector  4:  ADC timing overflow
68    case  6:                                   // Vector  6:  ADC12IFG0
69      results[index] = ADC12MEM0;              // Move results
70      index++;                                 // Increment results index, modulo; Set
Breakpoint1 here
71
72      if (index == 8)
73        index = 0;                             // Reset the index; Set Breakpoint here
74
75    case  8: break;                          // Vector  8:  ADC12IFG1
76    case 10: break;                          // Vector 10:  ADC12IFG2
77    case 12: break;                          // Vector 12:  ADC12IFG3
78    case 14: break;                          // Vector 14:  ADC12IFG4
79    case 16: break;                          // Vector 16:  ADC12IFG5
80    case 18: break;                          // Vector 18:  ADC12IFG6
81    case 20: break;                          // Vector 20:  ADC12IFG7
82    case 22: break;                          // Vector 22:  ADC12IFG8
83    case 24: break;                          // Vector 24:  ADC12IFG9
84    case 26: break;                          // Vector 26:  ADC12IFG10
85    case 28: break;                          // Vector 28:  ADC12IFG11
86    case 30: break;                          // Vector 30:  ADC12IFG12
87    case 32: break;                          // Vector 32:  ADC12IFG13
88    case 34: break;                          // Vector 34:  ADC12IFG14
89    default: break;
90    }
91  }
```

cc430x613x_adc12_09.c ADC12, Sequence of Conversions (non-repeated)01

```
    //*****************************************************************************
*****
02  //  CC430F6137 Demo - ADC12_A, Sequence of Conversions (non-repeated)
03  //
04  //  Description: This example shows how to perform A/D conversions on a sequence
```

```
05  //  of channels. A single sequence of conversions is performed - one conversion
06  //  each on channels A0, A1, A2, and A3. Each conversion uses AVcc and AVss for
07  //  the references. The conversion results are stored in ADC12MEM0, ADC12MEM1,
08  //  ADC12MEM2, and ADC12MEM3 respectively and are moved to 'results[]' upon
09  //  completion of the sequence. Test by applying voltages to pins A0, A1, A2,
10  //  and A3, then setting and running to a break point at the "_BIC..."
11  //  instruction in the ISR. To view the conversion results, open a watch window
12  //  in debugger and view 'results' or view ADC12MEM0, ADC12MEM1, ADC12MEM2,
and
13  //  ADC12MEM3 in an ADC12 SFR window.
14  //  This can run even in LPM4 mode as ADC has its own clock
15  //  Note that a sequence has no restrictions on which channels are converted.
16  //  For example, a valid sequence could be A0, A3, A2, A4, A2, A1, A0, and A7.
17  //  See the CC430 User's Guide for instructions on using the ADC12.
18  //
19  //                          CC430F6137
20  //                      -----------------
21  //               /|\|                     |
22  //                | |                     |
23  //                --|RST                  |
24  //                  |                     |
25  //      Vin0 -->|P2.0/A0                  |
26  //      Vin1 -->|P2.1/A1                  |
27  //      Vin2 -->|P2.2/A2                  |
28  //      Vin3 -->|P2.3/A3                  |
29  //                  |                     |
30  //
31  //    M Morales
32  //    Texas Instruments Inc.
33  //    April 2009
34  //    Built with CCE Version: 3.2.2 and IAR Embedded Workbench Version: 4.11B
35
    //*****************************************************************************
*****
36
37  #include "cc430x613x.h"
38
39  volatile unsigned int results[4];           // Needs to be global in this example
40                                              // Otherwise, the compiler removes it
41                                              // because it is not used for anything.
42
43  void main(void)
44  {
45     WDTCTL = WDTPW+WDTHOLD;                  // Stop watchdog timer
```

```
46
47      P2SEL = 0x0F;                                   // Enable A/D channel inputs
48
49      /* Initialize ADC12_A */
50      ADC12CTL0 = ADC12ON+ADC12MSC+ADC12SHT0_2; // Turn on ADC12, set
sampling time
51      ADC12CTL1 = ADC12SHP+ADC12CONSEQ_1;          // Use sampling timer, single
sequence
52      ADC12MCTL0 = ADC12INCH_0;                     // ref+=AVcc, channel = A0
53      ADC12MCTL1 = ADC12INCH_1;                     // ref+=AVcc, channel = A1
54      ADC12MCTL2 = ADC12INCH_2;                     // ref+=AVcc, channel = A2
55      ADC12MCTL3 = ADC12INCH_3+ADC12EOS;            // ref+=AVcc, channel = A3,
end seq.
56      ADC12IE = 0x08;                               // Enable ADC12IFG.3
57      ADC12CTL0 |= ADC12ENC;                        // Enable conversions
58
59      while(1)
60      {
61        ADC12CTL0 |= ADC12SC;                       // Start convn - software trigger
62
63        __bis_SR_register(LPM4_bits + GIE);       // Enter LPM4, Enable interrupts
64        __no_operation();                           // For debugger
65      }
66  }
67
68  #pragma vector=ADC12_VECTOR
69  __interrupt void ADC12ISR (void)
70  {
71      switch(__even_in_range(ADC12IV,34))
72      {
73      case  0: break;                               // Vector  0:  No interrupt
74      case  2: break;                               // Vector  2:  ADC overflow
75      case  4: break;                               // Vector  4:  ADC timing overflow
76      case  6: break;                               // Vector  6:  ADC12IFG0
77      case  8: break;                               // Vector  8:  ADC12IFG1
78      case 10: break;                               // Vector 10:  ADC12IFG2
79      case 12:                                      // Vector 12:  ADC12IFG3
80        results[0] = ADC12MEM0;                     // Move results, IFG is cleared
81        results[1] = ADC12MEM1;                     // Move results, IFG is cleared
82        results[2] = ADC12MEM2;                     // Move results, IFG is cleared
83        results[3] = ADC12MEM3;                     // Move results, IFG is cleared
84        __bic_SR_register_on_exit(LPM4_bits);     // Exit active CPU, SET BREAKPOINT
HERE
85      case 14: break;                               // Vector 14:  ADC12IFG4
```

```
    case 16: break;                           // Vector 16:  ADC12IFG5
    case 18: break;                           // Vector 18:  ADC12IFG6
    case 20: break;                           // Vector 20:  ADC12IFG7
    case 22: break;                           // Vector 22:  ADC12IFG8
    case 24: break;                           // Vector 24:  ADC12IFG9
    case 26: break;                           // Vector 26:  ADC12IFG10
    case 28: break;                           // Vector 28:  ADC12IFG11
    case 30: break;                           // Vector 30:  ADC12IFG12
    case 32: break;                           // Vector 32:  ADC12IFG13
    case 34: break;                           // Vector 34:  ADC12IFG14
    default: break;
    }
}
```

cc430x613x_adc12_10.c ADC12, Sample A10 Temp and Convert to oC and oF

```
//******************************************************************************
//  CC430F613x Demo - ADC12_A, Sample A10 Temp and Convert to oC and oF
//
//  Description: A single sample is made on A10 with reference to internal
//  1.5V Vref. Software sets ADC12SC to start sample and conversion - ADC12SC
//  automatically cleared at EOC. ADC12 internal oscillator times sample
//  and conversion. In Mainloop MSP430 waits in LPM4 to save power until
//  ADC10 conversion complete, ADC12_ISR will force exit from any LPMx in
//  Mainloop on reti.
//   ACLK = n/a, MCLK = SMCLK = default DCO ~ 1.045MHz, ADC12CLK = ADC12OSC
//
//  Uncalibrated temperature measured from device to devive will vary due to
//  slope and offset variance from device to device - please see datasheet.
//
//                CC430F6137
//             -----------------
//         /|\|              XIN|-
//          | |                 |
//          --|RST          XOUT|-
//            |                 |
//            |A10              |
//
//   M. Morales
//   Texas Instruments Inc.
```

```
025 //     April 2009
026 //     Built with CCE Version: 3.2.2 and IAR Embedded Workbench Version: 4.11B
027
      //*******************************************************************************
*****
028 #include "cc430x613x.h"
029
030 volatile long temp;
031 volatile long IntDegF;
032 volatile long IntDegC;
033
034 void main(void)
035 {
036     WDTCTL = WDTPW + WDTHOLD;                         // Stop WDT
037
038     /* Initialize the shared reference module */
039     REFCTL0 |= REFMSTR + REFVSEL_0 + REFON;       // Enable internal 1.5V reference
040
041     /* Initialize ADC12_A */
042     ADC12CTL0 = ADC12SHT0_8 + ADC12ON;             // Set sample time
043     ADC12CTL1 = ADC12SHP;                                 // Enable sample timer
044     ADC12MCTL0 = ADC12SREF_1 + ADC12INCH_10;   // ADC input ch A10 => temp
sense
045     ADC12IE = 0x001;                                          // ADC_IFG upon conv
result-ADCMEMO
046
047     __delay_cycles(75);                                 // 35us delay to allow Ref to settle
048                                                                 // based on default DCO frequency.
049                                                                 // See Datasheet for typical settle
050                                                                 // time.
051     ADC12CTL0 |= ADC12ENC;
052
053     while(1)
054     {
055         ADC12CTL0 |= ADC12SC;                         // Sampling and conversion start
056
057         __bis_SR_register(LPM4_bits + GIE);         // LPM4 with interrupts enabled
058         __no_operation();
059
060         // Temperature in Celsius
061         // ((A10/4096*1500mV) - 894mV)*(1/3.66mV) = (A10/4096*410) - 244
062         // = (A10 - 2438) * (410 / 4096)
063         IntDegC = ((temp - 2438) * 410) / 4096;
064
```

```
065         // Temperature in Fahrenheit
066         // ((A10/4096*1500mV) - 829mV)*(1/2.033mV) = (A10/4096*738) - 408
067         // = (A10 - 2264) * (738 / 4096)
068         IntDegF = ((temp - 2264) * 738) / 4096;
069
070         __no_operation();                               // SET BREAKPOINT HERE
071       }
072   }
073
074   #pragma vector=ADC12_VECTOR
075   __interrupt void ADC12ISR (void)
076   {
077     switch(__even_in_range(ADC12IV,34))
078     {
079     case  0: break;                                     // Vector  0:  No interrupt
080     case  2: break;                                     // Vector  2:  ADC overflow
081     case  4: break;                                     // Vector  4:  ADC timing overflow
082     case  6:                                            // Vector  6:  ADC12IFG0
083       temp = ADC12MEM0;                                 // Move results, IFG is cleared
084       __bic_SR_register_on_exit(LPM4_bits);     // Exit active CPU
085       break;
086     case  8: break;                                     // Vector  8:  ADC12IFG1
087     case 10: break;                                     // Vector 10:  ADC12IFG2
088     case 12: break;                                     // Vector 12:  ADC12IFG3
089     case 14: break;                                     // Vector 14:  ADC12IFG4
090     case 16: break;                                     // Vector 16:  ADC12IFG5
091     case 18: break;                                     // Vector 18:  ADC12IFG6
092     case 20: break;                                     // Vector 20:  ADC12IFG7
093     case 22: break;                                     // Vector 22:  ADC12IFG8
094     case 24: break;                                     // Vector 24:  ADC12IFG9
095     case 26: break;                                     // Vector 26:  ADC12IFG10
096     case 28: break;                                     // Vector 28:  ADC12IFG11
097     case 30: break;                                     // Vector 30:  ADC12IFG12
098     case 32: break;                                     // Vector 32:  ADC12IFG13
099     case 34: break;                                     // Vector 34:  ADC12IFG14
100     default: break;
101     }
102   }
```

COMP_B;

```
//*****************************************************************************
// CC430x613x Demo - COMPB output Toggle in LPM4; internal 2.0V reference
//
// Description: Use CompB and internal reference to determine if input'Vcompare'
//      is high of low.    When Vcompare exceeds 2.0V CBOUT goes high and when
//      Vcompare is less than 2.0V then CBOUT goes low.
//      Connect P1.6/CBOUT to P1.0 externally to see the LED toggle accordingly.
//
//                          CC430x613x
//                    ------------------
//              /|\|                      |
//               | |                      |
//               --|RST           P2.0/CB0|<--Vcompare
//                 |                      |
//                 |                      |       P1.6/CBOUT|---->   'high'(Vcompare>2.0V);
'low'(Vcompare<2.0V)
//                 |                      |   |
//                 |                      |    P1.0     |__|  LED   'ON'(Vcompare>2.0V);
'OFF'(Vcompare<2.0V)
//                 |                      |
//
//     M Morales
//     Texas Instruments Inc.
//     April 2009
//     Built with CCE Version: 3.2.2 and IAR Embedded Workbench Version: 4.11B
//*****************************************************************************
#include "cc430x613x.h"

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;                       // Stop WDT

    P1DIR |= BIT6;                                  // P1.6 output direction
    P1SEL |= BIT6;                                  // Select CBOUT function on
P1.6/CBOUT

    PMAPPWD = 0x02D52;                              // Get write-access to port mapping
regs
    P1MAP6 = PM_CBOUT0;                             // Map CBOUT output to P1.6
```

```
36      PMAPPWD = 0;                                    // Lock port mapping registers
37
38      P2SEL |= BIT0;                                  // Select CB0 input for P2.0
39
40      // Setup ComparatorB
41      CBCTL0 |= CBIPEN + CBIPSEL_0;                   // Enable V+, input channel CB0
42      CBCTL1 |= CBPWRMD_1;                            // normal power mode
43      CBCTL2 |= CBRSEL;                               // VREF is applied to -terminal
44      CBCTL2 |= CBRS_3+CBREFL_2;                      // R-ladder off; bandgap ref voltage
(1.2V)
45                                                      // supplied ref amplifier to get
Vcref=2.0V (CBREFL_2)
46      CBCTL3 |= BIT0;                                 // Input Buffer Disable @P6.0/CB0
47      CBCTL1 |= CBON;                                 // Turn On ComparatorB
48
49        __delay_cycles(75);
50
51      __bis_SR_register(LPM4_bits);                   // Enter LPM4
52      __no_operation();                               // For debug
53    }
```

cc430x613x_compB_03.c COMPB interrupts; internal 1.5V reference01

```
    //*****************************************************************************
*****
02  // CC430x613x Demo - COMPB interrupts; internal 1.5V reference
03  //
04  // Description: Use CompB and internal reference to determine if input'Vcompare'
05  //      is high of low.    For the first time, when Vcompare exceeds the 1.5V internal
06  //      reference, CBIFG is set and device enters the CompB ISR. In the ISR CBIES is
07  //      toggled such that when Vcompare is less than 1.5V internal reference, CBIFG is set.
08  //      LED is toggled inside the CompB ISR
09  //
10  //                          CC430x613x
11  //                      ------------------
12  //              /|\|                        |
13  //               | |                        |
14  //               --|RST          P2.0/CB0 |<--Vcompare
15  //                 |                        |
16  //                          |                          P1.0     |--> LED  'ON'(Vcompare<1.5V);
'OFF'(Vcompare>1.5V)
17  //                  |                        |
```

```
18  //
19  //     M Morales
20  //     Texas Instruments Inc.
21  //     April 2009
22  //     Built with CCE Version: 3.2.2 and IAR Embedded Workbench Version: 4.11B
23
    //*******************************************************************************
*****
24  #include "cc430x613x.h"
25
26  void main(void)
27  {
28      WDTCTL = WDTPW + WDTHOLD;                       // Stop WDT
29      P1DIR |= BIT0;                                  // P1.0/LED output direction
30
31      P2SEL |= BIT0;
32
33      /* Setup ComparatorB */
34      CBCTL0 |= CBIPEN + CBIPSEL_0;                   // Enable V+, input channel CB0
35      CBCTL1 |= CBPWRMD_1;                            // normal power mode
36      CBCTL2 |= CBRSEL;                               // VREF is applied to -terminal
37      CBCTL2 |= CBRS_3+CBREFL_1;                      // R-ladder off; bandgap ref voltage
(1.2V)
38                                                      //    amplify    to    get    Vcref=1.5V
(CBREFL_2)
39      CBCTL3 |= BIT0;                                 // Input Buffer Disable @P2.0/CB0
40      CBCTL1 |= CBON;                                 // Turn On ComparatorB
41
42      __delay_cycles(75);                             // Delay for shared ref to stabilize
43
44      CBINT &= ~(CBIFG + CBIIFG);                     // Clear any errant interrupts
45      CBINT    |= CBIE;                               // Enable CompB Interrupt on rising
46                                                      //      edge of CBIFG (CBIES=0)
47
48      __bis_SR_register(LPM4_bits+GIE);               // Enter LPM4 with inetrrupts enabled
49      __no_operation();                               // For debug
50  }
51
52  // Comp_B ISR - LED Toggle
53  #pragma vector=COMP_B_VECTOR
54  __interrupt void Comp_B_ISR (void)
55  {
56      CBCTL1 ^= CBIES;                    // Toggles interrupt edge
57      CBINT &= ~CBIFG;                    // Clear Interrupt flag
```

```
58      P1OUT ^= 0x01;                          // Toggle P1.0
59    }
```

cc430x613x_compB_04.c CBOUT from LPM4; CompB in ultra low power mode; Vref = Vcc*1/201

```
      //*****************************************************************************
*****
02    // CC430x613x Demo - CBOUT from LPM4; CompB in ultra low power mode; Vref =
Vcc*1/2
03    //
04    // Description: Use CompB and shared reference to determine if input 'Vcompare'
05    //       is high of low.    When Vcompare exceeds Vcc*1/2 CBOUT goes high and when
06    //       Vcompare is less than Vcc*1/2 then CBOUT goes low.
07    //       Connect P1.6/CBOUT to P1.0 externally to see the LED toggle accordingly.
08    //
09    //                            CC430x613x
10    //                      ------------------
11    //                /|\|                              |
12    //                 | |                              |
13    //                 --|RST          P2.0/CB0 |<--Vcompare
14    //                   |                              |
15    //                                |                     P1.6/CBOUT|---->   'high'(Vcompare>Vcc*1/2);
'low'(Vcompare<Vcc*1/2)
16    //                   |                              |   |
17    //                               |                        P1.0     |__|   LED   'ON'(Vcompare>Vcc*1/2);
'OFF'(Vcompare<Vcc*1/2)
18    //                   |                              |
19    //
20    //     M Morales
21    //     Texas Instruments Inc.
22    //     April 2009
23    //     Built with CCE Version: 3.2.2 and IAR Embedded Workbench Version: 4.11B
24
      //*****************************************************************************
*****
25    #include "cc430x613x.h"
26
27    void main(void)
28    {
29      WDTCTL = WDTPW + WDTHOLD;          // Stop WDT
30
```

```
31      P1DIR |= BIT6;                                  // P1.6 output direction
32      P1SEL |= BIT6;                                  // Select CBOUT function on
P1.6/CBOUT
33
34      PMAPPWD = 0x02D52;                              // Get write-access to port mapping
regs
35      P1MAP6 = PM_CBOUT0;                             // Map CBOUT output to P1.6
36      PMAPPWD = 0;                                    // Lock port mapping registers
37
38      P2SEL |= BIT0;                                  // Select CB0 input for P2.0
39
40      /* Setup ComparatorB */
41      CBCTL0 |= CBIPEN+CBIPSEL_0;     // Enable V+, input channel CB0
42      CBCTL1 |= CBMRVS;               // CMRVL selects the refV - VREF0
43      CBCTL1 |= CBPWRMD_2;            // Ultra-Low Power mode
44      CBCTL2 |= CBRSEL;               // VREF is applied to -terminal
45      CBCTL2 |= CBRS_1+CBREF04;       // VCC applied to R-ladder; VREF0 is Vcc*1/2
46      CBCTL3 |= BIT0;                 // Input Buffer Disable @P2.0/CB0
47      CBCTL1 |= CBON;                 // Turn On ComparatorB
48
49        __delay_cycles(75);
50
51      __bis_SR_register(LPM4_bits);                   // Enter LPM4
52      __no_operation();                               // For debug
53  }
```

cc430x613x_compB_05.c COMPB Hysteresis, CBOUT Toggle in LPM4; High speed mode01

```
    //*****************************************************************************
*****
02  // CC430x613x Demo - COMPB Hysteresis, CBOUT Toggle in LPM4; High speed mode
03  //
04  // Description: Use CompB and shared reference to determine if input 'Vcompare'
05  //      is high of low.    Shared reference is configured to generate hysteresis.
06  //      When Vcompare exceeds Vcc*3/4 CBOUT goes high and when Vcompare is less
07  //      than Vcc*1/4 then CBOUT goes low.
08  //      Connect P1.6/CBOUT to P1.0 externally to see the LED toggle accordingly.
09  //
10  //                          CC430x613x
11  //                      ------------------
12  //              /|\                         |
```

```
13  //               | |                          |
14  //               --|RST          P2.0/CB0 |<--Vcompare
15  //                 |                          |
16  //                                |                    P1.6/CBOUT|---->   'high'(Vcompare>Vcc*3/4);
'low'(Vcompare<Vcc*1/4)
17  //                 |                          |   |
18  //                            |                          P1.0     |__|  LED   'ON'(Vcompare>Vcc*3/4);
'OFF'(Vcompare<Vcc*1/4)
19  //                 |                          |
20  //
21  //     M Morales
22  //     Texas Instruments Inc.
23  //     April 2009
24  //     Built with CCE Version: 3.2.2 and IAR Embedded Workbench Version: 4.11B
25
    //*******************************************************************************
*****
26  #include "cc430x613x.h"
27
28  void main(void)
29  {
30      WDTCTL = WDTPW + WDTHOLD;         // Stop WDT
31
32      P1DIR |= BIT6;                                    // P1.6 output direction
33      P1SEL  |=  BIT6;                                        // Select CBOUT function on
P1.6/CBOUT
34
35      PMAPPWD = 0x02D52;                                  // Get write-access to port mapping
regs
36      P1MAP6 = PM_CBOUT0;                                  // Map CBOUT output to P1.6
37      PMAPPWD = 0;                                        // Lock port mapping registers
38
39      P2SEL |= BIT0;                                    // Select CB0 input for P2.0
40
41      // Setup ComparatorB
42      CBCTL0 |= CBIPEN + CBIPSEL_0; // Enable V+, input channel CB0
43      CBCTL1  |=  CBPWRMD_0;                 // CBMRVS=0 => select VREF1 as ref when
CBOUT
44                                            // is high and VREF0 when CBOUT is low
45                                            // High-Speed Power mode
46      CBCTL2 |= CBRSEL;                     // VRef is applied to -terminal
47      CBCTL2 |= CBRS_1+CBREF13;          // VREF1 is Vcc*1/4
48      CBCTL2 |= CBREF04+CBREF03;         // VREF0 is Vcc*3/4
49      CBCTL3 |= BIT0;                       // Input Buffer Disable @P6.0/CB0
```

```
50      CBCTL1 |= CBON;                         // Turn On ComparatorB
51
52      __bis_SR_register(LPM4_bits); // Enter LPM4
53      __no_operation();                       // For debug
54  }
```

cc430x613x_compB_06.c COMPB and TIMERAx interaction (TA0.1, TA1.1)
view source

print
?001

```
    //*****************************************************************************
*****
002 // CC430x613x Demo - COMPB and TIMERAx interaction (TA0.1, TA1.1)
003 //
004 // Description: Use CompB to determine if input, Vcompare has a duty cycle
005 //   greater than 50%. When Vcompare exceeds Vcc*3/4 then TimerA0 captures the
006 //   time (TA0CCR1). When Vcompare is less than Vcc*1/4 then TimerA1 captures the
007 //   time (TA1CCR1) and resets the timers for TIMERA0 and TIMERA1. If TA0CCR1 is
008 //   greater than TA1CCR1/2, then turn on the LED. If TA0CCR1 is less than
009 //   TA1CCR1/2, then turn off the LED.
010 //
011 //   Clocks: ACLK = REFO; MCLK = SMCLK = 12MHz
012 //
013 //                          CC430x613x
014 //                      ------------------
015 //               /|\|                       |
016 //                | |                       |
017 //                --|RST                CB0 |<--Vcompare (200Hz<f<1Mhz) (vary dutycycle)
018 //                  |                         |
019 //                  |                         |
020 //                  |                   P1.0|-->LED ('ON' if dutycycle > 50%; 'OFF' if dutycycle
< 50%)
021 //
022 //    M Morales
023 //    Texas Instruments Inc.
024 //    April 2009
025 //    Built with CCE Version: 3.2.2 and IAR Embedded Workbench Version: 4.11B
026
    //*****************************************************************************
*****
```

```
027 #include "cc430x613x.h"
028
029 void main(void)
030 {
031     WDTCTL = WDTPW + WDTHOLD;         // Stop WDT
032
033     // Setup UCS: ACLK = REFO; MCLK = SMCLK = 12MHz
034     UCSCTL4 = SELA__REFOCLK + SELS__DCOCLKDIV + SELM__DCOCLKDIV;
035     UCSCTL3 |= SELREF__REFOCLK;                         // Set DCO FLL reference =
REFO
036
037     __bis_SR_register(SCG0);                          // Disable the FLL control loop
038     UCSCTL0 = 0x0000;                                   // Set lowest possible DCOx, MODx
039     UCSCTL1 = DCORSEL_5;                                // Select DCO range 24MHz
operation
040     UCSCTL2 = FLLD_1 + 374;                             // Set DCO Multiplier for 12MHz
041                                                         // (N + 1) * FLLRef = Fdco
042                                                         // (374 + 1) * 32768 = 12MHz
043                                                         // Set FLL Div = fDCOCLK/2
044     __bic_SR_register(SCG0);                          // Enable the FLL control loop
045
046     // Worst-case settling time for the DCO when the DCO range bits have been
047     // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
048     // UG for optimization.
049     // 32 x 32 x 12 MHz / 32,768 Hz = 375000 = MCLK cycles for DCO to settle
050     __delay_cycles(375000);
051
052     // Loop until XT1,XT2 & DCO fault flag is cleared
053     do
054     {
055       UCSCTL7 &= ~(XT2OFFG + XT1LFOFFG + XT1HFOFFG + DCOFFG);
056                                                         // Clear XT2,XT1,DCO fault flags
057       SFRIFG1 &= ~OFIFG;                                // Clear fault flags
058     }while (SFRIFG1&OFIFG);                             // Test oscillator fault flag
059
060     P2SEL |= BIT0;                                    // Enable CB0 input
061
062 // Setup ComparatorB
063     CBCTL0 |= CBIPEN + CBIPSEL_0; // Enable V+, input channel CB0
064     CBCTL1 |= CBPWRMD_0;                // CBMRVS=0 => select VREF1 as ref when
CBOUT
065                                       // is high and VREF0 when CBOUT is low
066                                       // High-Speed Power mode
067     CBCTL2 |= CBRSEL;                   // Ref is applied to -terminal
```

```
068     CBCTL2 |= CBRS_1+CBREF13;         // VREF1 is Vcc*1/4
069     CBCTL2 |= CBREF04+CBREF03;        // VREF0 is Vcc*3/4
070     CBCTL3 |= BIT0;                  // Input Buffer Disable @P6.0/CB0
071     CBCTL1 |= CBON;                   // Turn On ComparatorB
072
073  // Setup Timer_A0 and Timer_A1
074  // Internally CBOUT --> TA0CCTL1:CCIB (TA0.1)
075  //                  +-> TA1CCTL1:CCIB (TA1.1)
076     TA0CTL = TASSEL_2 + MC_1;                  // SMCLK, upmode
077     TA1CTL = TASSEL_2 + MC_1;                  // SMCLK, upmode
078     TA0CCR0 = 0xFFFF;
079     TA1CCR0 = 0xFFFF;
080     TA0CCTL1 = CM_2+SCS+CAP+CCIS_1;            // Capture Falling Edge
081     TA1CCTL1 = CM_1+SCS+CAP+CCIS_1+CCIE;       // Capture Rising Edge, Enable
Interrupt
082
083     P1DIR |= BIT0;                             // Set P1.0/LED to output
084
085     __bis_SR_register(LPM3_bits+GIE);          // Enter LPM0 with global interrupts
enabled
086     __no_operation();                          // For Debug
087  }
088
089  #pragma vector=TIMER1_A1_VECTOR
090  __interrupt void Timer_A(void)
091  {
092     unsigned int temp;
093     switch( TA1IV )
094     {
095       case  2:
096                 TA0CCR0 = 0;
097                 TA1CCR0 = 0;
098                 TA0CCR0 = 0xFFFF;
099                 TA1CCR0 = 0xFFFF;                // Halt and resart TimerA0 and A1
counters
100                 temp = TA1CCR1>>1;               // Compare On and off time of input
signal
101                 if(TA0CCR1 > temp)
102                   P1OUT |= BIT0;
103                 else
104                   P1OUT &= ~BIT0;
105                 break;
106       case  4: break;                            // CCR2 not used
107       case  6: break;                            // CCR3 not used
```

```
108       case    8: break;                          // CCR4 not used
109       case 10: break;                            // CCR5 not used
110       case 12: break;                            // Reserved not used
111       case 14:                                   // Overflow
112                 __no_operation();
113                 while(1);                        // If input frequency < 200Hz, trap here
114       default: break;
115     }
116 }
```