```
*==============================================================================
*
*       TEXAS INSTRUMENTS, INC.
*
*       IIRCAS4
*
*       Revision Data: 05/14/97
*
*       USAGE   This routine is C Callable and can be called as:
*
*               void iircas4(int n, short *c, int *d, int *y)
*
*               n       = the number of cascaded biquads
*               c       = array containing -a1, -a2, b1, b2 biquad coefs
*               d       = array of the delayed states within biquads
*               y       = inputs y[0] and y[1] (also outputs)
*
*       C CODE          This is the C equivalent of the assembly code without
*               restrictions. Note that the assembly code is hand optimized and
*               restrictions may apply.
*
*       void iircas4(int n, short *c, int *d, int *y)
*       {
*       int k0, k1, i;
*        for (i = 0; i < n; i++) {
*         k0   = c[4*i+1]*(d[2*i+1] >> 16) + c[4*i+0]*(d[2*i+0]>> 16) +y[0];
*         y[0] = c[4*i+3]*(d[2*i+1] >> 16) + c[4*i+2]*(d[2*i+0]>> 16) + k0;
*         d[2*i+1] = k0;
*         k1   = c[4*i+1]*(d[2*i+0] >> 16) + c[4*i+0]*(k0 >> 16) + y[1];
*         y[1] = c[4*i+3]*(d[2*i+0] >> 16) + c[4*i+2]*(k0 >> 16) + k1;
*         d[2*i+0] = k1;
*         }
*       }
*
*       DESCRIPTION
*               The iircas4 performs a cascaded biquad iir filter with
*               the direct form II structure (4-multiplies.) It performs two
*               samples at a time. Coefficients are stored in the order -a1,
*               -a2, b1, b2 for each successive biquad located in the c array.
*               Both outputs are stored back to the location of the inputs y[0]
*               and y[1]. The inputs and outputs are 32 bit v       alues while the
*               coefficients are 16 bit values.
*
*       TECHNIQUES
*               The loop is written so that one biquad for each of the two
*               inputs is completed every time through the loop. There is a
*               an extra priming delay for the second input so that the biquads
*               new delayed state k0 is calculated based on the first input
*               (i.e. the second input is being processed by the biquad
*               preceeding the biquad which is processing the first input.)
*
*       MEMORY NOTE
*               The d and c array pointers must be placed on opposite word
*               boundaries to avoid memory hits (i.e. one must start on an even
*               word boundary while the other starts on an odd word boundary.)
*
*       CYCLES          4*N + 16  (Note: the iircas4 cycle count is for two inputs.)
*               for N = 10 -> 56 cycles or 280 nsec
*
*==============================================================================
        .global _iircas4
        .text

_iircas4:
```

```
        STW     .D2     B10,    *B15--          ; push B10 on stack
||      MV      .L1X    B6,     A3              ; copy y pointer

*** BEGIN Benchmark Timing ***
B_START:

        B       .S2     LOOP                    ; for
||      LDW     .D1     *+A3[1],  B10   ; y1 = y[1]
||      MV      .S1X    B11,    A9              ; store B11 in A9

        ADD     .L1     2,      A4,     A1      ; n+2
||      MVK     .S2     3,      B0              ; setup priming count
||      MVK     .S1     2,      A2              ; setup priming count
||      LDW     .D1     *+A3[0],  A7    ; y0 = y[0]

LOOP:

 [!A1]  MVK     .S1     1,      A2
||      MPYH    .M2     B11,    B5,     B9      ; g1 = (d0 >> 16) * b2
||      MV      .L2X    A0,     B8              ;* copy a1, a2
||      MV      .S2     B2,     B11             ;* copy d0
||      ADD     .L1     A5,     A8,     A8      ;* h0 = a1d0 + a2d1
||      MPYH    .M1X    B6,     A4,     A8      ;* b2d1 = (d1 >> 16) * b2
||      LDW     .D1     *A6++,  B2              ;*** d0 = d[2*i+0]
||      LDW     .D2     *B4++,  A0              ;*** a1 = c[4*i], a2 = c[4*i+1]

 [A1]   SUB     .S1     A1,     1,      A1      ; i++
||[!B0] ADD     .S2     B9,     B10,    B10     ; h1 = j1 + y1
||[!A2] ADD     .L1     A8,     A7,     A7      ;* k0 = h0 + y0
||      MV      .L2X    A4,     B5              ;* copy b1, b2
||      MPYH    .M2     B11,    B8,     B1      ;* e1 = (d0 >> 16) * a2
||      MPYHL   .M1X    B2,     A0,     A5      ;** a1d0 = (d0 >> 16) * a1
||      LDW     .D2     *B4++,  A4              ;*** b1 = c[4*i+0],b2 = c[4*i+1]
||      LDW     .D1     *A6++,  B6              ;*** d1 = d[2*i+1]

 [A1]   B       .S1     LOOP                    ; for
||      ADD     .L2     B10,    B1,     B8      ; k1 = h1 + e1
||      ADD     .D2     B7,     B9,     B9      ; m1 = f1 + g1
||      MPYHL   .M2X    A7,     B8,     B9      ;* j1 = (k0 >> 16) * a1
||[!A2] STW     .D1     A7,     *-A6[5]         ;* d[2*i+1] = k0
||      ADD     .L1     A5,     A8,     A8      ;* m0 = b1d0 + b2d1
||      MPYH    .M1X    B6,     A0,     A8      ;** a2d1 = (d1 >> 16) * a2

 [B0]   SUB     .S2     B0,     1,      B0      ; decrement priming count
||[A2]  SUB     .L1     A2,     1,      A2      ; decrement priming count
||[!B0] STW     .D1     B8,     *-A6[8]         ; d[2*i+0] = k1
||[!B0] ADD     .L2     B9,     B8,     B10     ; y1 = m1 + k1
||      MPYHL   .M2X    A7,     B5,     B7      ;* f1 = (k0 >> 16) * b1
||[!A2] ADD     .S1     A8,     A7,     A7      ;* y0 = m0 + k0
||      MPYHL   .M1X    B2,     A4,     A5      ;** b1d0 = (d0 >> 16) * b1
;end of LOOP

        B       .S2     B3                      ; return
||      LDW     .D2     *++B15,  B10            ; pop B10 off stack
||      STW     .D1     A7,     *+A3[0]         ; y[0] = y0

        STW     .D1     B10,    *+A3[1]         ; y[1] = y1
||      MV      .S2X    A9,     B11             ; restore B11

B_END:
*** END Benchmark Timing ***

        NOP     4
```