TEXAS INSTRUMENTS, INC.

IIRCAS4

Revision Date: 05/14/97

USAGE  This routine is C Callable and can be called as:

    void iircas4(int n, short c[4], int y[]);

    n = the number of cascaded biquads
    c = array containing a1, a2, b1, b2 biquad coeffs
    d = array of the delayed states within biquad
    y = inputs y[0] and y[1] (also output)

C CODE  This is the C equivalent of the assembly code without
        restrictions. Note that the assembly code is hard optimized and
        restrictions may apply.

    void iircas4(int n, short c[4], int y[])
    {
        int k0, k1;
        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];
        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 values 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 printing delay for the second input so that the biquad
new delayed state k0 is calculated based on the first input
(i.e. the second input is being processed by the biquad
processing 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
4N + 16  Note: the iircas4 cycle count is for two inputs.)
    for N = 10 -> 56 cycles or 200 nsec


17:37:11 16/11/01
STW .D2 B10, *B15-- ; push B10 on stack
MV .L1X B9, A3 ; copy y pointer

*** BEGIN Benchmark Timing ***

B_START:

B .S2 LOOP ; for
LDW .D1 *A3[1], B10 ; y1 = y[1]
MV .L1X B11, A9 ; store B11 in A9
ADD .S2 2, M, A1 ; n+2
MVX .S2 3, B0 ; setup printing count.
MVX .S2 2, A2 ; setup printing count.
LDW .D1 *A3[0], A7 ; y0 = y[0]

LOOP:

[AL] MVK .S1 L, A2
MPYH .M2 B11, B5, B9 ; q1 = (dJ >> 16) * b2
MV .L2X A0, B8 ; * copy a1, a2
MV .S2 R, B11 ; * copy d0
ADD .S1 A5, A8, A8 ; * h0 = a0 + a1
MPYH .M1X B6, A4, A8 ; * k2 = (d1 >> 16) * b2
LDW .D1 *A6++, B2 ; *** d1 = d[2*i+0]
LDW .D1 *B4++, A0 ; *** a1 = c[4*i+1], a2 = c[4*i+1]

[Al] SUB .S1 A1, L, A1 ; i++
[[AS] ADD .S2 B9, B10, B10 ; h2 = h1 + y1
[[AS] ADD .S2 A5, A7, A7 ; * k1 = h0 + y0
MV .L2X A4, B5 ; * copy b4, b2
MPYH .M2 B11, B6, B1 ; * el = (d3 >> 16) * a2
LDW .D1 *A6++, B4 ; ** a2dJ = (dJ »16) * a1
LDW .D1 *B4++, A4 ; *** b1 = c[4*i+0], b2 = c[4*i+1]

[Al] B .S1 LOOP ; for
ADD .S2 B10, B1, B8 ; k1 = b1 + el
ADD .S2 B7, B9, B9 ; el = f1 + g1
MPYH .M2 X A7, B6, B9 ; * g1 = (d0 >> 16) * a1
[[AS] ADD .S1 A5, A6, A8 ; * x0 = b0d0 + b0k1
MPYH .M1X B6, A0, A8 ; ** x2dJ = (dJ >> 16) * a2

[B0] SUB .S2 B0, L, B0 ; decrement printing count.
[[BO] SUB .S1 A2, L, A2 ; decrement printing count.
[[BO] STW .D1 B6, *-A6[8] ; d[2*i+0] = k1
[[BO] ADD .D2 B9, B6, B10 ; y1 = el + k1
MPYH .M2 X A7, B5, B7 ; * k1 = (d0 >> 16) * b1
[[BO] ADD .S1 A6, A7, A7 ; * y0 = x0 + k0
MPYH .M1X B5, A4, A5 ; ** k0dJ = (dJ >> 16) * b1

end of LOOP

B .S2 B3 ; return
LDW .D2 *+B15, B10 ; pop B10 off stack
STW .D1 B10, *A3[0] ; y[0] = y0
MV .S2X A9, B11 ; restore B11

B_END:

*** END Benchmark Timing ***

NOP 4


17:37:11 16/11/01