IIR Filter Design on the TMS320C54x DSP

Mandy Tsai

Field Application Engineer Customer Application Center
Texas Instruments Taiwan Limited

SPRA079
May 1996

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Contents

List of Illustrations

ABSTRACT

Digital filter design plays an important role in the digital signal processor (DSP) field. The two methods of designing a digital filter are infinite-impulse response (IIR) and finite-impulse response (FIR). If an IIR filter satisfies the same specifications as a FIR filter, the IIR filter is usually faster and requires less memory. IIR filters are not easy to implement with a fixed-point DSP. The feedback path of the IIR filter causes a calculation overflow. Scaling the input data corrects this overflow problem. The scaling results in a small output signal. To recover the output signal level, the last stage of the hardware design uses an operational amplifier to boost the output signal level. This application report shows how to eliminate the operational amplifier and input calculation by combining the cascaded second-order IIR filters.

IIR FILTER DESIGN

A high-order IIR filter design simplifies to several second-order filters. The equation for a high-order filter is:

$$H(z) = H_1(z) \times H_2(z) \times H_3(z) \times H_4(z) \times \cdots \times H_n(z) \quad (1)$$

where *n* is the order of the filter.

Dividing equation (1) by several second-order filters gives:

$$H_k(z) = \frac{1 + b_{1k}z^{-1} + b_{2k}z^{-1}}{1 - a_{1k}z^{-1} - a_{2k}z^{-1}} \quad (2)$$

where $H_k(z)$ is a second-order filter and k =1, 2, 3, 4, ...

Figure 1 shows the structure of the second-order IIR filter. The IIR filter provides separate delay buffers for the input and output sequences. This filter structure is Direct Form I. The output from the filter is:

$$y(n) = a_{11}y(n-1) + a_{12}y(n-2) + x(n) + b_{11}x(n-1) + b_{12}x(n-2) \quad (3)$$

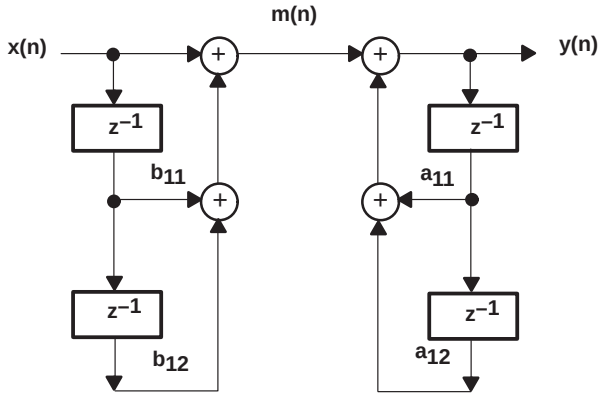

Figure 1. Direct Form I of a Second-Order IIR Filter

The second-order IIR filter is two cascaded sections: the zeros and the poles. Since IIR filters are linear systems, the two sections are interchangeable. The sections generate the same output in either order (see Figure 2).

Since the output of the delays in Figure 2 are identical, one set of delays is eliminated from the filter structure. The structure of Figure 3 is equivalent to that of Figure 2, while using fewer delay elements to define the system. This filter structure is Direct Form I and is known as a canonical structure.

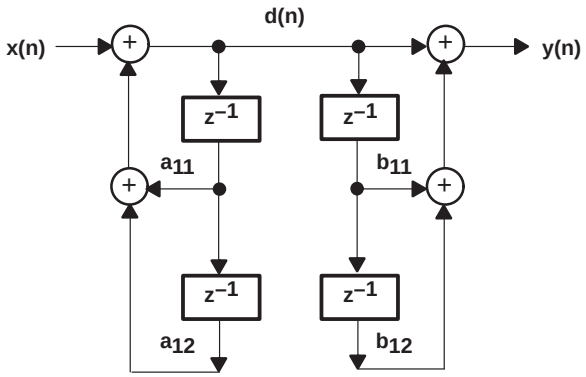

Figure 2. IIR Filter Poles and Zeros Interchange

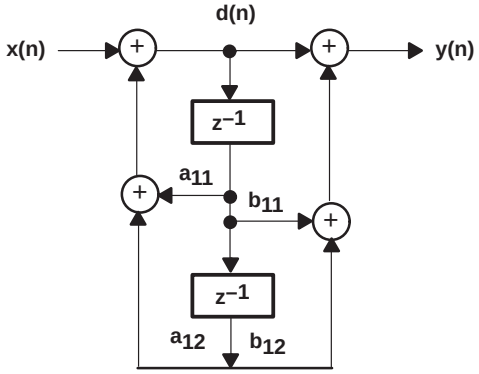

Figure 3. Canonical Structure of Second-Order IIR Filter

The concept of eliminating hardware (the delay buffers) from a design is easy to understand, but how does this translate to the software? A digital IIR filter uses a fixed-point DSP, either ‘C5x or ‘C54x. Data overflow in the DSP is a problem due to the limitations of its 32-bit architecture. The feedback path or poles of the IIR filter are the cause of the overflow. The filter gain (from the pole section) increases and generates an output data overflow. This data value does not fit within the 32 bits of the CPU.

To prevent the overflow problem, the input data value is scaled down before entering the filter system. In a filter application the input data value is very small, resulting in an output without overflow. The scaling of the input causes a small output value.

If the programmer follows the canonical structure of Figure 3 to design the code, two problems emerge: the programmer wastes time adjusting the input values to avoid output overflow and the system requires additional hardware, an operational amplifier, to recover the level on the output signal. A modified Direct Form I structure solves these problems. Figure 4 shows the canonical structure of Direct Form I.

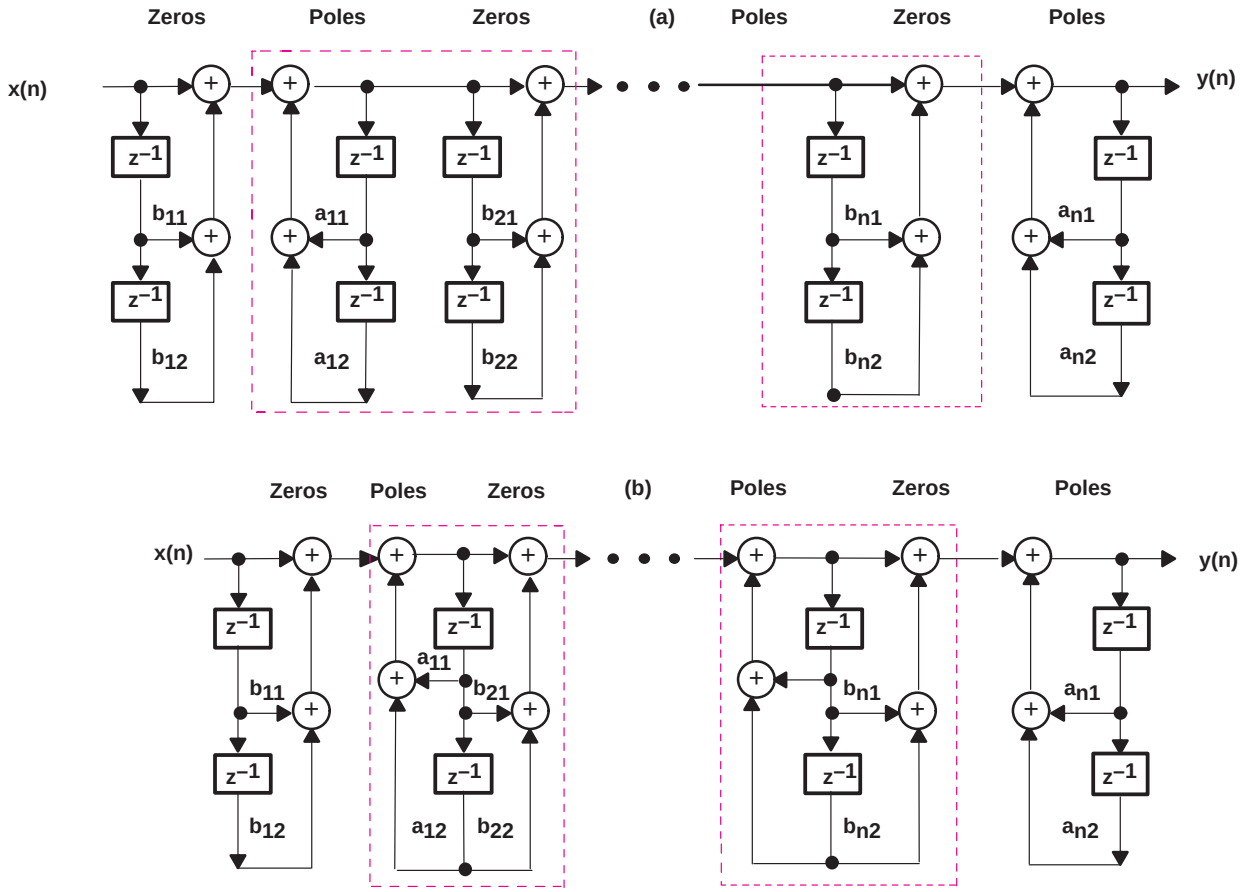

Figure 4. Canonical Direct Form of a High-Order IIR Filter

Efficient IIR Filter Design

The IIR filter using Direct Form I (see Figure 1) calculates the zeros in the forward path, then the poles in the feedback path. The intermittent result m(n) is a small value due to the zeroes calculation. The pole calculation, d(n) of Figure 2 provides a value larger than m(n). The output data, y(n) assumes a proper level by passing the interim value through the pole feedback path. This method for filter design eliminates overflow concerns.

Figure 4 shows the high-order IIR filter as a number of cascaded second-order filters. Each second-order IIR uses Direct Form I. The number of delay buffers is reduced by combining the poles (a_{11}, a_{12}) of a second-order IIR filter with the zeros (b_{21}, b_{22}) of the next IIR filter section. This reduction produces the canonical direct form.

The software for the cascaded sections uses the repeat block instruction. The overhead software comes from the calculations for the first zero section (b_{11},b_{12}) and the last pole portion (a_{n1},a_{n2}). The output data, y(n), is at a normal level, thus this form eliminates the operational amplifier of the original IIR filter.

‘C54x CODE AND EXPERIMENT

Two software programs are in the appendixes of this report. The program in Appendix A represents the original IIR filter; the one in Appendix B represents the efficient filter design. Both programs use the ‘C54x code.

Filter Specifications

A low-pass filter meets the following frequency standards: a cut-off frequency for the pass band of 200 Hz and a cut-off frequency for the stop band of 500 Hz.

Figure 5 shows how the power level of the input signal converts to integers ranging from –16000 to 16000.

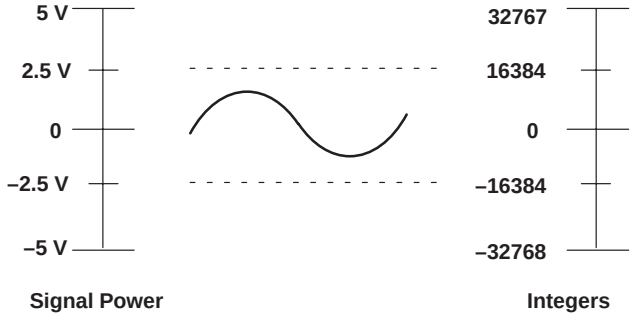

Figure 5. Power to Integer Conversion

Experiment Results

Table 1 compares the IIR filters and a high-order FIR filter. The modified IIR filter achieves the same system functionality as the high-order FIR with a second-order design. The original IIR filter output signal reduces the input signal by 100. This results in the addition of an operational amplifier to the hardware of the system. The modified IIR eliminates the op amp. The modified IIR needs to scale the input signal, but not to the extent required in the original IIR filter design.

Table 1. Comparison of IIR and FIR Filters

Filter	IIR	Modified IIR	FIR	Symmetric FIR
Algorithm	Elliptic	Elliptic	Kaiser	Kaiser
Order of Filter	4	4	32	32
Input Scaling	9-Bit Left Shift (1/512)	5-Bit Left Shift (1/32)	None	None
Output Level	–1000 ~ 1000 (small)	–16000 ~ 16000 (normal)	–16000 ~ 16000 (normal)	–16000 ~ 16000 (normal)
Program Size (words)	56	61	65	53
Data Size (words)	8	8	32	32
Cycles/Output	27	22	36	28

SUMMARY

An IIR filter design using a modified structure, canonical Direct Form I, improves upon the original IIR filter design in two ways: the scaling of the input signal eliminates overflow in the CPU and is no longer necessary, and the operational amplifier is eliminated. The modified IIR filter requires scaling of the input signal to provide a proper output signal without the additional hardware. This input scaling is not as great as in the original IIR filter.

APPENDIX A — IIR FILTER PROGRAM IN ‘C54X CODE

```
;***************************************************************
; Original IIR Low pass filter design
; Language : C54x
; Filter type : Elliptic Filter
; Filter order : 4 order (cascade: 2nd order + 2nd order)
; cut freq. of pass band : 200 Hz
; cut freq. of stop band : 500 Hz
; Designer : Mandy Tsai
; Date : Feb,20,1996
;                                                B0
;           ... -----> + ------> d(n) ---- x- -> + -----....
;                        |              |                    |
;                        |    A1      |          B1      |
;                        + <- x -- d(n-1) -- x -> +
;                        |              |                    |
;                        |    A2     |           B2      |
;                        + <- x -- d(n-2) -- x -> +
;
;                          second order IIR
; ***************************************************************
        .mmregs
       .def     begin, N
N       .set    2                   ; number of cascade 2nd order section
        .bss    d,3*2                 ;delay buffer in 2nd order
        .bss    X,1                   ;input buffer
        .bss    Y,1                   ;output buffer
        .data
table
*
*  second-order section # 01
*
        .word    -26778               ;A2
        .word    29529                ;A1/2
        .word    19381                ;B2
        .word    -23184               ;B1
        .word    19381                ;B0
*
*  second-order section # 02
*
        .word    -30497               ;A2
        .word    31131                ;A1/2
        .word    11363                ;B2
        .word    -20735               ;B1
```

```
        .word    11363                  ;B0
        .sect   ”vectors”               ;define a reset vector
        B     begin
begin  .text                            ;the start of the program
        STM   #1111111110100000b,PMST   ;initial PMST
        STM   #0010001100000000b,ST1    ;initial ST1
        STM    #0,SWWSR                 ;zero wait state
        SSBX   OVM                      ;OVM=1
        SSBX   FRCT                     ;FRCT=1 : output of multiply will left
                                        shift 1 bit automatically
        SSBX    SXM                     ;SXM=1
        STM    #X,AR1
        STM    #Y,AR2
        STM   #d,AR3                    ;AR3:d(n),d(n–1),d(n–2)
        RPT   A,#5                      ;initial d(n),d(n–1),d(n–2)=0
        STL    A,*AR3+
        STM   #2,AR0                    ;initial constant offset of ARn addressing
INLOOP:
        STM   #d+5,AR3                  ;AR3:d(n),d(n–1),d(n–2)
        STM   #table,AR4                ;AR4:coeff of IIR filter A2,A1,B2,B1,B0
        PORTR 100H, *AR1                ;Read data from the port and save
                                        ;in data array.
                                        ;the connection between file and I/O
                                        ;port is defined in siminit.cmd
        LD    *AR1,7,A                  ;scaling the input
        STM   #N–1,BRC                  ;calculating N sections of 2nd order IIR
        RPT    ELOOP–1
LOOP:
* Feedback path
        MAC   *AR4+,*AR3–,A             ;input+d(n–2)*A2
        MAC   *AR4,*AR3,A               ;input+d(n–2)*A2+d(n–1)*A1
        MAC    *AR4+,*AR3–,A
        STH   A,*AR3+0                  ;d(n) = input+d(n–2)*A2+d(n–1)*A1
* Forward path
        MPY   *AR4+,*AR3–,A             ;d(n–2)*B2
        MAC   *AR4+,*AR3,A              ;d(n–2)*B2+d(n–1)*B1
        DELAY *AR3–                     ;d(n–2)=d(n–1)
        MAC   *AR4+,*AR3,A              ;d(n–2)*B2+d(n–1)*B1+d(n)*B0
        DELAY  *AR3–                    ;d(n–1)=d(n)
ELOOP:
        STH    A,*AR2                   ;output=d(n–2)*B2+d(n–1)*B1+d(n)*B0
        PORTW  *AR2, 200h               ;write the result to a file
        B      INLOOP                   ;calculating next output
```

APPENDIX B — EFFICIENT IIR FILTER PROGRAM IN ‘C54X CODE

```
;***************************************************************
; New IIR Low pass filter design
; Language : C54x
; Filter type : Elliptic Filter
; Filter order : 4 order (cascade: 1 order + 2nd order + 1 order)
;                canonical direct from I
; cut freq. of pass band : 200 Hz
; cut freq. of stop band : 500 Hz
; Designer : Mandy Tsai
; Date : Feb,20,1996
; ***************************************************************
        .mmregs
        .def     begin, N
N       .set    2                       ; length of IIR filter
        Q_FACT  .set    32768
        .bss    d,3*2
        .bss    X,1
        .bss    Y,1
        .data
*Q31 format
table
;*
;*  SECOND–ORDER SECTION # 01
;*
        .word    19381                   ;B2
        .word    –23184                  ;B1
        .word    19381                   ;B0
        .word    –26778                  ;A2
        .word    29529                   ;A1/2
;*
;*  SECOND-ORDER SECTION # 02
;*
        .word    11363                   ;B2
        .word    –20735                  ;B1
        .word    11363                   ;B0
        .word    –30497                  ;A2
        .word    31131                   ;A1/2
        .sect  ”vectors”                 ;define a reset vector
        B     begin
begin   .text                           ;the start of the program
        STM     #1111111110100000b,PMST   ;initial PMST
        STM     #0010001100000000b,ST1    ;initial ST1
        STM     #0,SWWSR                  ;zero wait state
```

```
        SSBX    OVM
        SSBX    FRCT                    ;FRCT=1 : output of multiply will left shift
        1 bit automatically
        SSBX    SXM
        STM     #d,AR3                  ;AR3:d(n),d(n–1),d(n–2)
        RPTZ    A,#7                    ;initial d(n),d(n–1),d(n–2)=0
        STL     A,*AR3+
        STM     #2,AR0
INLOOP:
        STM     #d+7,AR3                ;AR1:d(n),d(n–1),d(n–2)
        STM     #table,AR4              ;AR2:coeff of IIR filter –a2,–a1,b2,b1,b0
        MPY     *AR4+,*AR3–,A           ;A=d(n–2)*b2
        MAC     *AR4+,*AR3,A            ;A=A+d(n–1)*b1
        DELAY   *AR3–
        MAC     *AR4+,*AR3,A            ;A=A+d(n)*b0
        DELAY   *AR3
        PORTR   100H,*AR3
        LD      *AR3,B
        STH     B,11,*AR3–              ;left shift by 5 to scale the input
        STM     #N–2,BRC
        RPTB    ELOOP–1
LOOP:
        MAC     *AR4+,*AR3–,A           ;A = A+d(n–2)*(–a2)
        MAC     *AR4,*AR3,A             ;A = A+d(n–1)*(–a1)
        MAC     *AR4+,*AR3–,A
        STH     A,*AR3+0                ;save d(n)
        MPY     *AR4+,*AR3–,A           ;A=d(n–2)*b2
        MAC     *AR4+,*AR3,A            ;A=A+d(n–1)*b1
        DELAY   *AR3–
        MAC     *AR4+,*AR3,A            ;A=A+d(n)*b0
        DELAY   *AR3–
ELOOP:
        MAC     *AR4+,*AR3–,A
        MAC     *AR4,*AR3,A
        MAC     *AR4+,*AR3,A
        DELAY   *AR3
        STH    A,*AR3
        PORTW   *AR3, 200h              ;write the result to a file, two word
                                        instruction
        B       INLOOP
```