Analog Devices公司DSP、处理器和开发工具的使用技术指南
联系我们的技术支持中心：dsp.support@analog.com 和 dsptools.support@analog.com
查看我们的在线资源：http://www.analog.com/ee-notes 和 http://www.analog.com/processors

16-bit IIR滤波器在ADSP-TS20x TigerSHARC®处理器上的实现

Rickard Fahlqvist提供 *2003年11月6日*

引言

本文介绍在ADSP-TS20x TigerSHARC®处理器上执行一个2阶16-bit IIR滤波，利用了直接型II。

滤波器结构

2阶滤波器的结构见*图1*。可以通过若干个2阶滤波器的级联或并联达到高于2阶的滤波。这样的滤波器的阶数是2的倍数。做一个奇数阶数的滤波器需要一个1阶滤波器和一个或几个2阶滤波器。*列表3*中的示例代码没有包括这一点。

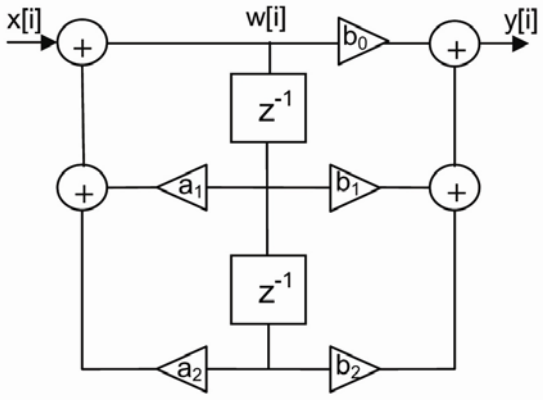

图1 2阶直接型II

下面的*方程式1*描述了这种类型的滤波器结构。

$$w[i] = x[i] + \sum_{k=1}^{2} a_k w[i-k]$$

$$y[i] = \sum_{k=0}^{2} b_k w[i-k]$$

方程式1 描述2阶直接型II IIR

第一个分子系数（b_0）总设为1，这样需要将输入x[i]除以b_0,b_1和b_2再除以b_0。示例中即如是做法，见下面的列表1。

执行

以流水线方式执行一个小的递归运算是个困难的问题。这个滤波器中W[i]的值取决于W[i-1]。这种前后依赖限制了流水线操作的程度。在这个程序中，所有用来产生w[i] 和 y[i]的乘法运算都在单个周期中计算。然而，把部分结果再送入输入，累加得到最终的结果需要5个周期。这期间乘法器处于空闲状态，结果造成程序只利用了一个计算单元。由于用于管理延迟线（还有内存装取）的操作由ALU完成，因此要在每个指令行中插入多个指令槽也是个问题。上面提到，可以通过并连2阶滤波器得到高阶滤波器。这样就可以同时在两个计算区块计算两个滤波输出，最后累加得到这些结果的和。

这2个要素的状态存储在寄存器yR5中，在每次递归运算中，每个新得到的w[i]先被插入状态寄存器，再进行左移位操作。由于延迟线是重叠的，因而可以执行一个如*图2*所示的4-way 16x16-bit乘法指令来计算反馈通道和前馈通道

上的4个乘法运算。

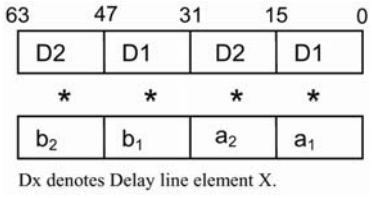

图2 延迟线和系数的乘法

部分结果由Sideways Summation指令得到。如果系数既有分数又有整数，那么这4-way乘法必须分成两个独立的乘法分别运算，同时会降低效率。

因为使用了SDAB（短型字数据队列缓存），每次调用就要载入8个16-bit字，而仅需使用第一个字。因为内存带宽不是瓶颈，所以这种对载入字的看似浪费的处理方法却是合适的。

接口

本例中讲述的C类型的滤波器原型见下面的*列表1.*

```
void iir_16(int2x16 input[],
int output[],
int input_len,
iir_state_t *f_state)
```

列表1 IIR函数原型

定义的“iir_state_t”是一个结构，包含系数，延迟线和输入比例因数。这一结构的定义见*列表2*。

```
typedef struct
{
const int2x16 c;   /* coefficients */
int2x16 d;  /* start of delay line */
int2x16 s;  /* Input value scaling */
} iir_state_t;
```

列表2 滤波器的state结构

附录

附录为示例文件的汇编源代码。

IIR_16.asm

```
/* ********************************************************************************
 *
•   Copyright © 2003 Analog Devices Inc.  All rights reserved.
 *
 * ********************************************************************************/
.section program;
.global _iir_16;


// Local defines
#define Arg0 j4
#define Yout j5
#define Arg2 j6
#define s_p j7
#define Xin j0
#define Xin_tmp j1
#define s_d j2

#define c_offs 0
#define d_offs 1
#define k_offs 2
#define s_offs 3
#define coeff_p j10


_iir_16:
//PROLOGUE
   J26 = J27 - 64;            K26 = K27 - 64;;
   [J27 += -28] = CJMP;K27 = K27 - 20;;
   Q[J27 + 24] = XR27:24;     Q[K27 + 16] = YR27:24;;
   Q[J27 + 20] = XR31:28;     Q[K27 + 12] = YR31:28;;
   Q[J27 + 16] = J19:16;      Q[K27 + 8 ] = K19:16;;
   Q[J27 + 12] = J23:20;      Q[K27 + 4 ] = K23:20;;
//PROLOGUE ENDS
coeff_p = [s_p + c_offs];;
yR24 = [s_p + s_offs]; yR25 = R25 XOR R25;; // Get scale factor
yR21:20 = l[coeff_p += 2];; // Load ai's and bi's
Xin = Arg0 + j31;;
Xin = Xin + Xin; LC0 = Arg2;; // Double it to access shorts; Get # of input samples
yR31 = 0x0010;;              // used for FDEP with length=16, position=0
s_d = [s_p + d_offs]; yR4 = R4 XOR R4;;
yR5 = [s_d += j31];;         // Load delay line

jl0 = jl0 - jl0;;            // Avoid unintended circular buffer
Xin_tmp = Xin;;
yR3:0 = sDAB q[Xin += 8];;
yR4 = R4 OR R5;;             // Replicate the two 16-bit states
yR3:0 = sDAB q[Xin += 8];;
Xin_tmp = Xin_tmp + 1; yR11:10 = R5:4 * R21:20;;// Do a1*D1, a2*D2, b1*D1, b2*D2
// and store in R10:11
Xin = Xin_tmp + j31; yR1:0 = R1:0 * R25:24;; // scale
```

```
yR15 = SUM sR10;;                    // get a1*D1 + a2*D2
yR29:28 = EXPAND sR0(I);;            // LSS(R0) -> R28
yR16 = SUM sR11;;                    // b1*D1 + b2*D2
yR15 = R28 + R15;;                   // w[i] = x[i] + a1*D1 + a2*D2

.align_code 4;
loop_:
yR5 = LSHIFT R5 by 16; yR4 = R4 XOR R4;;        // Shift out oldest element in delay
                                                // line
yR3:0 = sDAB q[Xin += 8]; yR5 += FDEP R15 by R31;;   // Get x[i+1] and store the
                                                     // new w[i] in delay line
yR3:0 = sDAB q[Xin += 8]; yR17 = R16 + R15;;    // y[i] = w[i] + b1*D1 + b2*D2
Xin_tmp = Xin_tmp + 1; yR4 = R4 OR R5;;
yR1:0 = R1:0 * R25:24;;                         // Scale i/p
Xin = Xin_tmp + j31; yR11:10 = R5:4 * R21:20;; // Do a1*D1, a2*D2, b1*D1, b2*D2 and
                                                // store in R10:11
yR29:28 = EXPAND sR0(I);;                       // LSS(R0) -> R28
yR15 = SUM sR10;;                               // get a1*D1 + a2*D2
yR16 = SUM sR11;;                               // b1*D1 + b2*D2
if NLC0E, jump loop_; yR15 = R28 + R15; [Yout += 1] = yR17;; // w[i+1] = x[i+1] +
// a1*D1 + a2*D2
// EPILOGUE STARTS
CJMP = [J26 + 64];;
   YR27:24 = q[K27 + 16];    XR27:24 = q[J27 + 24];;
   YR31:28 = q[K27 + 12];    XR31:28 = q[J27 + 20];;
   K19:16  = q[K27 + 8 ];    J19:16  = q[J27 + 16];;
   K23:20  = q[K27 + 4 ];    J23:20  = q[J27 + 12];;
CJMP (ABS); J27:24=q[J26+68]; K27:24=q[K26+68]; nop;;
// EPILOGUE ENDS
_iir_16.end:
```

列表3　　源文件 iir_16.asm

文档历史

版本	描述
2003年11月6日　　R. Fahlqvist	第一版