Extended Precision IIR Filter Design on the TMS320C54x DSP

Application on an Audio Equalizer

Literature Number: SPRA454
Texas Instruments Europe
June 1998

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products 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, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

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

TI assumes no liability for applications assistance or customer product design. TI does not 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. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Contents

List of Figures

Extended Precision IIR Filter Design on the TMS320C54x DSP

ABSTRACT

This paper presents methods of achieving a good compromise in accuracy when carrying out extended-precision multiplications for the implementation of IIR (Infinite Impulse Response) filters.

The ‘C54x devices are 16-bit fixed-point processors and have several features that help to perform extended-precision computation efficiently.

1. Introduction

An useful convention for fixed-point digital signal processing is to interpret signal samples as integers, i.e. in Q0 format, and to represent coefficients in sub-unitary format, that is to allow only coefficient values less than unity, thus reducing the overflow problems [3]. Generally all coefficients are represented in binary format with the implied binary point to the left of the MSB (signed coefficients from -1 to $(1-2^{-15})$ using Q15 format).

Firstly, extended-precision 16x32-bit (or 32x16-bit) and 32x32-bit multiplications are explained and an implementation on the ‘C54x is shown. Then, after a brief theory of IIR filters, implementations of the following different forms of IIR filters are described:

- Direct Form I in extended precision 32x16-bit.
- Direct Form II in extended precision 32x16-bit.
- Cascade Form in extended precision 32x16-bit.
- Direct Form I in extended precision 32x32-bit.
- Direct Form II in extended precision 32x32-bit.
- Cascade Form in extended precision 32x32-bit.
- Direct Form I in extended precision 16x32-bit.

2. Extended Precision Multiplication

The ‘C54x CPU has a 17x17-bit hardware multiplier coupled to a 40-bit dedicated adder. The advantage of this multiplier is that it can multiply two unsigned numbers or two signed numbers as well as signed/unsigned numbers [4]. The ‘C54x has two accumulators called A and B which can be configured as the destination registers for the multiplier/adder unit.

Each accumulator is split into three parts, as shown in Figure 1.

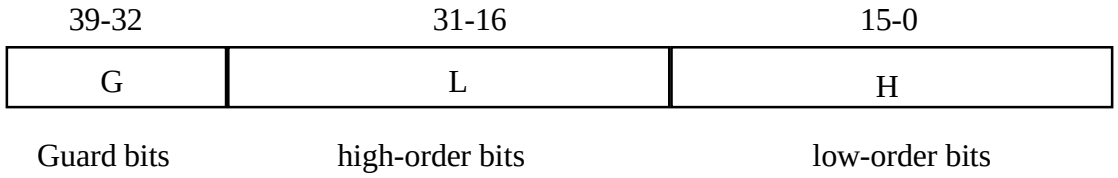

Figure 1: The 'C54x accumulators

The guard bits are used as headmargin for computations allowing some overflow in iterative calculations.

The fractional mode of the 'C54x allows the multiplier to shift left by one bit to compensate for the extra sign bit generated by multiplying two signed 16-bit numbers in fractional mode (when the FRCT bit of the ST1 register is set the fractional mode is selected).

- For signed multiplication, the 16-bit memory operands are assumed to be 17-bit words with sign extension.
- For unsigned multiplication, a zero is added to the MSB (the 17th bit) in each operand.
- For signed/unsigned multiplication, one operand is signed extended, while the other is extended with a zero in the MSB.

The SXM bit of the register ST1 controls the signed/unsigned extension of the data operands (when the SXM bit is set the signed extension mode is selected) [4].

The 'C54x architecture is built around eight major 16-bit buses (four program/data buses and four address buses). The 'C54x can generate up to two data memory addresses per cycle.

Other advantages of the C54x multiplier/adder unit are its several input sources and its dedicated zero detector, its rounder and its overflow/saturation logic.

The possible input sources to the multiplier for the first operand are:

- the T register,
- a data-memory operand from data bus DB,
- accumulator A (bits 32-16).

The possible input sources to the multiplier for the second operand are:

- a data-memory operand from data bus DB,
- a data-memory operand from data bus CB,
- a program-memory operand from program bus PB,
- accumulator A (bits 32-16).

2.1 The 32x32-bit multiplication

The following figure shows how two 32-bit numbers can be multiplied to obtain a 64-bit result.

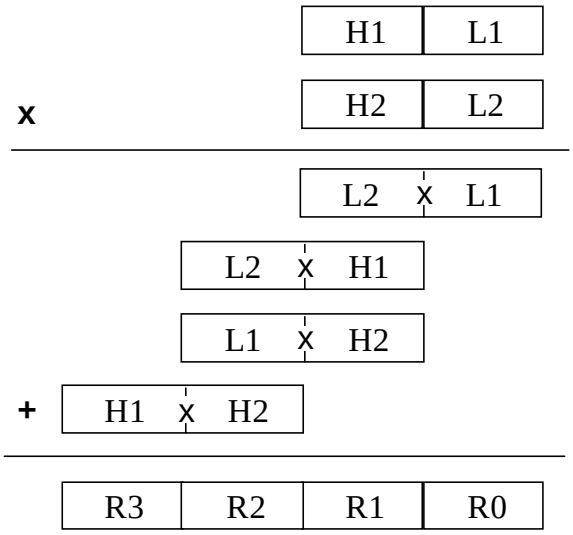

Figure 2: Multiplication 32x32-bit

The first multiplication L1 x L2 is an unsigned/unsigned multiplication. The second and third ones, L2 x H1 and L1 x H2, are signed/unsigned multiplications. The last, H1xH2, is a signed/signed multiplication.

2.2 The 32x16-bit multiplication

The following figure shows how a 32-bit number and a 16-bit number can be multiplied to obtain a 48-bit result.

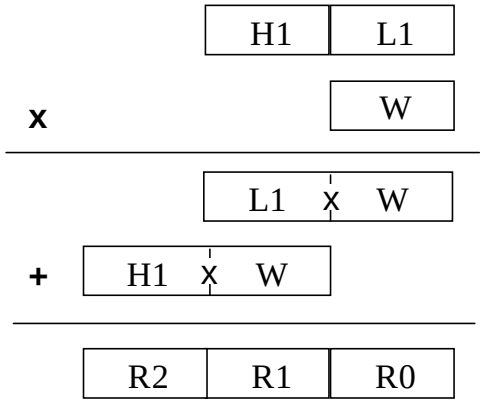

Figure 3: Multiplication 32x16-bit

The first multiplication L1xW is a signed/unsigned multiplication, whereas the second one H1xW is a signed/signed multiplication.

2.3 'C54x instruction set to handle extended precision computation

The following instructions lead to extended precision multiplications being performed efficiently.

- MACSU
- MAC
- MPYU
- MPY
- LD
- DLD
- DST
- Rounding mode

The MPYU instruction multiplies two unsigned 16-bit numbers and places the 32-bit result in one of the two accumulators in a single cycle whereas the MACSU instruction permits multiplication of a signed 16-bit number by an unsigned 16-bit number and adds the result to the source accumulator in a single cycle [5]. The LD instruction may be used to perform a right-shift of the accumulator by 16 bits in a single cycle in order to obtain a 16-bit result [5].

The two internal data buses, CB and DB, allow some instructions to handle 32-bit operands in a single cycle like DLD and DST instructions. The DLD instruction enable loading of a long-word into a specified accumulator. With the DST instruction a specified accumulator can be stored in memory as a long-word.

Alternatively, the rounding method consists of adding 8000h to the 32-bit result of the accumulator before shifting the lower 16 bits [3]. When the suffix R is included with the special multiply instructions MAC (multiplication/accumulation), or MAS (multiply/subtract), or MPY, or load operation LD, the rounding operation is executed [5].

3. IIR Filters

The goal is to use extended-precision multiplications to implement IIR filters.

A filter is called IIR if its z-transform has poles. The underlying problem is that it can be unstable. The IIR filters have an impulse response that bends towards zero if the filter is stable and towards a different value if not. The general expression for its z-transform is:

$$G(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{N} a_i z^{-i}}{1 + \sum_{i=1}^{M} b_i z^{-i}}$$

and the corresponding difference equation is:

$$y(n) = -\sum_{i=1}^{M} b_i y(n-i) + \sum_{i=0}^{N} a_i x(n-i) \quad (1)$$

3.1 Direct Form I

The direct graphic representation of the difference equation (1) is called *the Direct Form I* [1]. This structure is depicted in Figure 4.

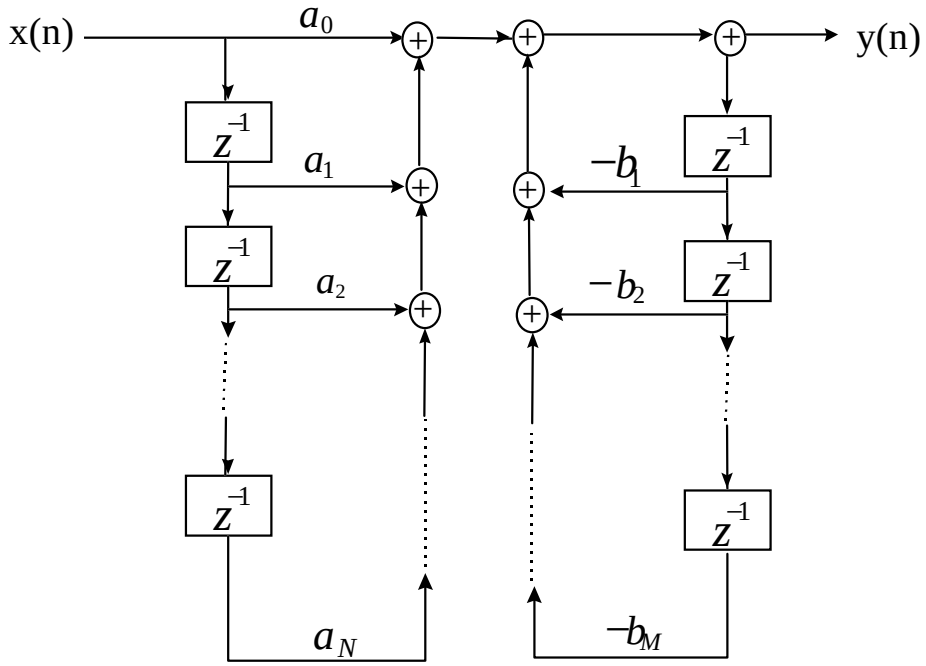

Figure 4: Direct Form I

Note that the left half of the figure implements the numerator (zeros) of $G(z)$, while the right half implements the denominator (poles) of $G(z)$.

3.2 Direct Form II

When the order of the numerator and the order of the denominator are the same *(N=M)*, the delay lines can be combined in a single one. This structure, called the *Direct Form II* [1], is shown in Figure 5.

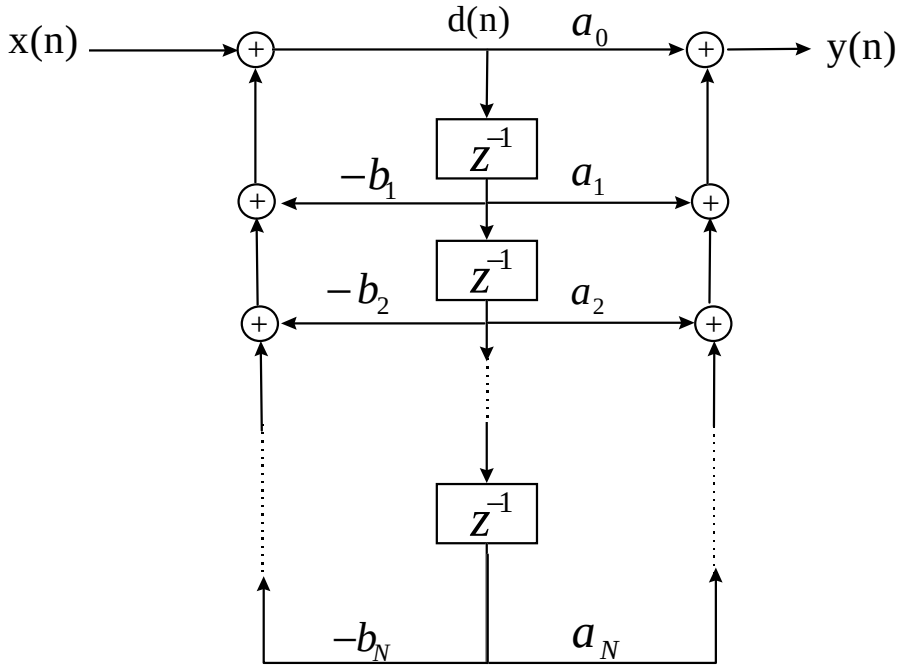

Figure 5: Direct Form II

The two forms require the same number of arithmetic operations, but the form II can require as few as half the number of memory registers for storing the past values of the inputs and outputs. Although Direct Form I and Direct Form II have the same z-transform *G(z)*, the corresponding difference equations are not the same. The difference equation of the Form II structure is:

$$d(n) = -\sum_{k=1}^{N} b_k d(n-k) + x(n)$$

$$y(n) = \sum_{k=0}^{M} a_k d(n-k)$$

Other structures or other sets of equations can be found for the implementation of IIR filters by manipulating the z-transform *G(z)*.

3.3 Cascade Form

The cascade form is obtained by factoring the numerator and denominator of the z-transform into second order polynomials to give the following z-transform:

$$G(z) = C.\prod_{i=1}^{(N+1)/2} \frac{a_{i0} + a_{i1}z^{-1} + a_{i2}z^{-2}}{1 + b_{i1}z^{-1} + b_{i2}z^{-2}} \qquad (2)$$

When N is odd or when $N \neq M$, some of the coefficients in (2) will be equal to zero. This form is a serial structure which can be built as a cascade of second order filters implemented in any desirable form. An example with $N=M=4$ where the second-order filters are implemented in Direct Form II is shown in Figure 6.

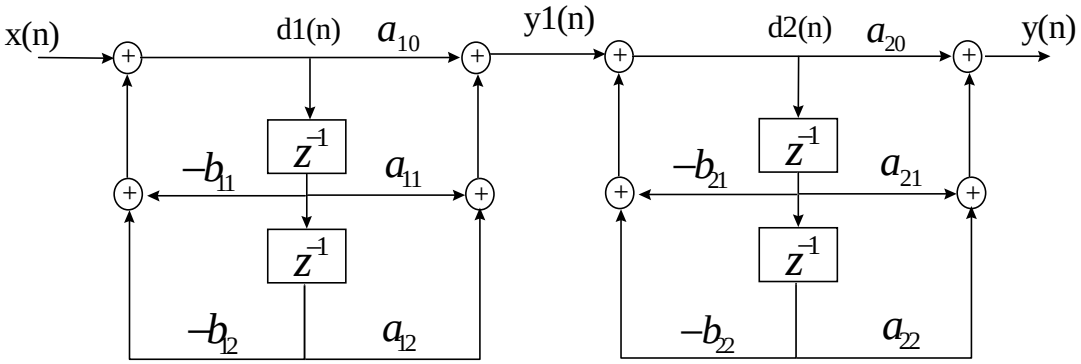

Figure 6: Cascade Form

Another form can be used to implement the second order filters involved in the cascade form [1]. It will be seen in paragraph 5.3 as a cascade form using a modified Direct Form I to implement the second order filters.

4. Implementation of Extended Precision multiplication on 'C54x

Our aim is to use the extended precision multiplication for the implementation of IIR filters. Thus, recursive calculations have to be performed. If two n-bit numbers are multiplied, 2n bits are required to store the result. It is possible to use two instructions to store both halves of the result in consecutive memory locations, but this doubles the time taken to store the output and the amount of memory required. It also doubles the time required to retrieve the value for use in subsequent calculations. Because of this overhead it is usual to store only the most significant bits and truncate the result. There is also an option to round the result before truncating it. In this paragraph the implementation of different multiplications in extended precision is described and the error due to the truncation is evaluated.

4.1 Extended Precision 32x32-bit multiplication

In order to have full accuracy it is necessary to preserve the 64-bit result. However, if we want to use the result recursively as an input in later calculations, it is useful to keep only the most significant 32 bits. For example the 32 LSB corresponding to L1 x L2 can be neglected and the result of the addition L1xH2 + L2xH1 can be truncated to keep only the 16 MSB (bits 16-31) before being added to H1xH2.

The implementation of this extended precision multiplication on a 'C54x can be performed by the following assembly instructions:

```
MACSU   *AR1-,*AR2+,A                   ; A = A+(H2*L1)<<1
MACSU   *AR2-,*AR1,A                    ; A = A+(L2*H1)<<1
LD      A,-16,A                         ; A = A>>16
MAC     *AR1-,*AR2,A                    ; A = A+(H1*H2)
```

Initially, the auxiliary register AR1 points to the low 16-bit of the first 32-bit word and the auxiliary register AR2 points to the high 16-bit of the second 32-bit word. Note the use of MACSU and LD instructions. The final 32-bit result is in the accumulator A.

Truncation noise

The accuracy error introduced in such a multiplication can be evaluated. We introduce an error of 1 LSB for the loss of the intermediate product L1xL2 and another 1 LSB by truncating the 16 LSB of the addition L1xH2 + L2xH1 in the worst case [6] [7]. On average, we introduce an error of 1 LSB with a mean square $\sigma = \frac{q}{\sqrt{6}}$ where q is the step of quantification or a variance $V = \sigma^2 = \frac{q^2}{6}$ where $q = 2^{-31}$. This variance of $\frac{q^2}{6}$ corresponds to the addition of the variance $V_1 = \sigma_1^2$ for the loss of the intermediate product L1xL2 and the variance $V_2 = \sigma_2^2$ for the truncation of the 16 LSB of the addition L1xH2 + L2xH. Each of them corresponds to an error of one LSB in the worst case and we have $V_1 = V_2 = \frac{q^2}{2}\int_0^1 x^2 dx - \frac{q^2}{2}[\int_0^1 x dx]^2 = \frac{q^2}{12}$. Therefore we have $V = V_1 + V_2 = \frac{q^2}{6}$. In

order to reduce the error of accuracy it is possible to do a rounding instead of discarding the 16 LSB of the intermediate product L1 x L2. In this case, an error of half a LSB is introduced in the worst case, rather than one LSB, with an average of zero [6] [7]. However, this would take one operation more.

4.2 Extended Precision 32x16-bit or 16x32-bit multiplication

The result of the multiplication L1xW is truncated to keep only the 16 MSB. The implementation of this extended-precision multiplication on a 'C54x can be performed as follow:

```
MACSU   *AR1+,*AR2,A                    ; A = A+(W*L1)<<1
LD      A,-16,A                         ; A = A>>16
MAC     *AR1-,*AR2,A                    ; A = A+(W*H1)
```

Initially, AR1 points to the LSB of the 32-bit word. Note the use of `MACSU` and `LD` instructions. At the end, the final 32-bit result is present in accumulator A.

Truncation noise

The error E due to the truncation of the L1xW1 result is equal to one LSB ($q = 2^{-31}$) in the worst case. We have $0 < E < q$. On average we introduce an error of $\frac{1}{2}$ LSB ($\frac{q}{2} = 2^{-32}$) with a standard deviation $\sigma = \frac{q}{\sqrt{12}}$ [6].

5. Implementations of IIR filters in extended precision on 'C54x

Efficient implementations of the Direct Form I, Direct Form II and Cascade Form in extended precision 32x16-bit and 32x32-bit are explained in this section.

Several buffers have to be used and must be multiplied in the correct order. Additionally, although the coefficient values are static, the input data changes every sample period, e.g. the x(n) value for one sampling period becomes *x(n-1)* in the next one, then *x(n-2)*, ..., until it simply drops off the end of the delay line.

The most efficient method for handling these data tables is to load all the input values into a circular buffer [2].

5.1 Implementation of the 32x16-bit Direct Form I

In this paragraph, an example of a second order IIR filter in a Direct Form I is given, with coefficients in a Q15 format and one circular buffer for handling the data samples. The AR0 auxiliary register is used to carry out an indexed circular addressing mode.

5.1.1 Circular buffer

A circular buffer requires:

- the location of the first (lowest) address. A circular buffer which contains R 16-bit words must start on an address whose N LSB are 0, where N is the smallest integer that satisfies $2^N > R$ [4],
- an initialization of the circular-buffer size register BK to specify the size of the circular buffer. The value R must be loaded into BK [4].

5.1.2 Memory organization

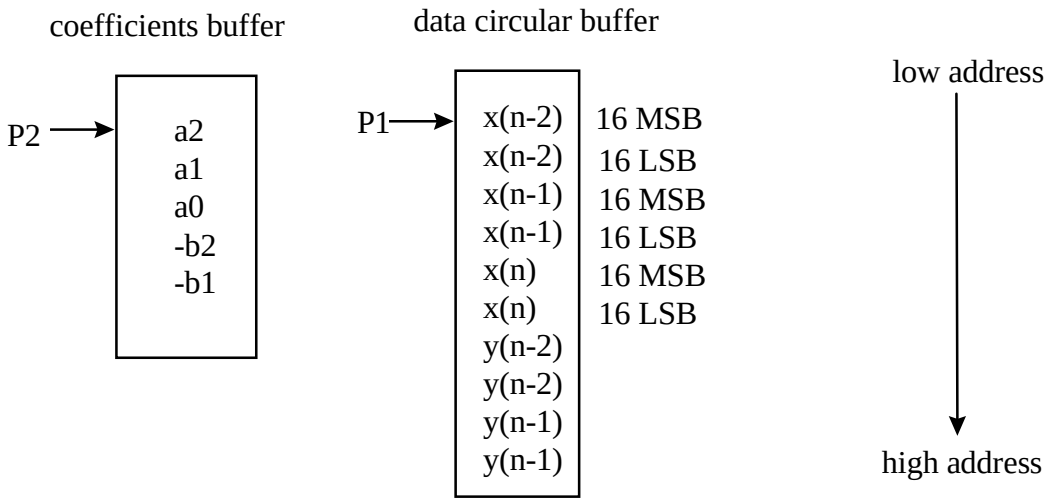

Figure 7: Memory organization for the Direct Form I implementation

The 32-bit values x(i) are stored in memory using two 16-bit words.

When the end of the buffer is reached (here, for example, when the pointer is on *x(n)*), the pointer automatically goes back onto the beginning of the buffer (here *x(n-2)*) when it is incremented.

Data circular buffer update is performed as follows:

- Data pointers need to be on *x(n-2)* before computation and then they are incremented automatically to do the multiplication with the correct coefficients. The new sample is placed at the 32-bit position x(n). At the end of the output calculation $y(n) = a_2 x(n-2) + a_1 x(n-1) + a_0 x(n) - b_1 y(n-1) - b_2 y(n-2)$, the pointer P1 is set on x(n-2) MSB and the 32 bits of y(n) are stored instead of the old *x(n-2)*.
- The pointers are incremented to point to the old x(n-1) which becomes x(n-2) and the *x(n)* becomes *x(n-1)*.
- A new *x(n)* will be stored instead of the old *y(n-2)* value, the old *y(n-1)* values becoming *y(n-2)*.

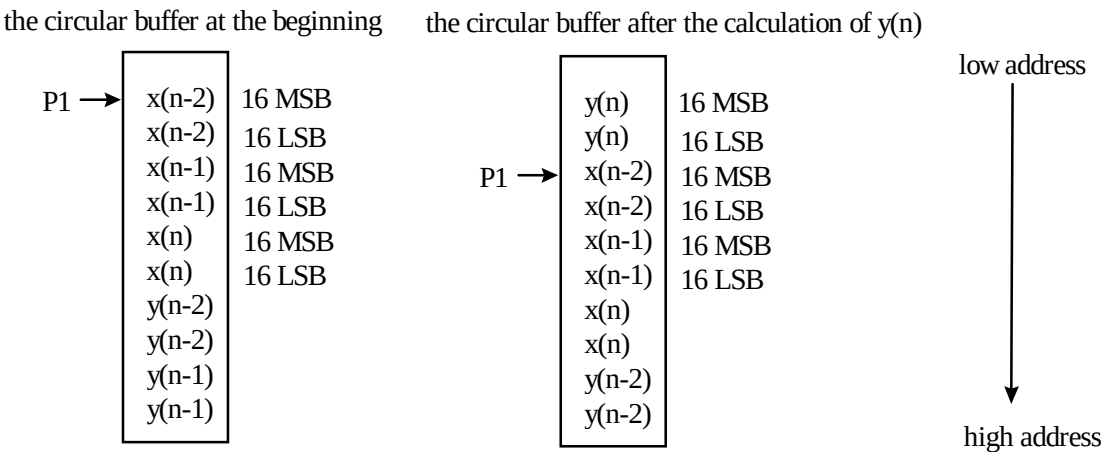

Figure 8: Data circular buffer update

where P1 is the pointer on the current data high.

The INDEX value, that is the content of the auxiliary register AR0 (index of the pointers P1), is set to 1 in order to jump in the data buffer from one 16-bit MSB to the LSB of the same data.

5.1.3 Program organization

When processing a 32x16-bit multiplication, signed/unsigned multiplications are used with a 16-bit right shift followed by a signed multiplication. In order to decrease the number of cycles, we have grouped the signed/unsigned multiplications on one side before performing the truncation (the right shift) and the signed multiplications on the other side. This gives a better accuracy.

The coefficients have been calculated so that the filter does not have a gain greater than 1. Nevertheless, an internal overflow may occur because of the accumulation of multiplications. Therefore overflow is allowed (OVM = 0) in the guard bits of the accumulator.

Six auxiliary registers are needed:

- AR2 (dual access) to point to the low internal values (16 bits). This is declared as IIR_DATA_P_L.
- AR3 (dual access) to point to the high internal values (16 bits). This is declared as IIR_DATA_P_H.
- AR4 (dual access) to point to the coefficients declared as pIIR_COEFF.
- AR6 to point to the inputs. This is declared as INBUF_P.
- AR7 to point to the outputs. This is declared as OUTBUF_P.
- and AR0 to index the circular buffer. This is declared as IIR_INDEX.

The 32-bit output result is in Accumulator B before being stored.

The entry and output conditions are SXM=1 (sign extended mode), OVM=0 (to allow overflow mode) and FRCT=1 (fractional mode) in the status register ST1 [4].

5.1.4 Program explanations

The full program is given in the appendix but some explanations are given here of the main instructions that make up the IIR filter. These instructions are:

```
IirFilterBegin

     STM    #(K_FRAME_SIZE-1),BRC
     RPTBD  IirFilterLoopEnd-1
     STM    #(DataFilout),pOUTBUF          ; Load output address memory

     LD     #0,A
     MPY    *pIIR_DATA+0%,*pIIR_COEFF,B    ; A2*x(n-2)high
     MACSU  *pIIR_DATA+0%,*pIIR_COEFF+,A   ; A2*x(n-2)low
     MAC    *pIIR_DATA+0%,*pIIR_COEFF,B    ; A1*x(n-1)high
     MACSU  *pIIR_DATA+0%,*pIIR_COEFF+,A   ; A1*x(n-1)low
     MVDD   *pINBUF+,*pIIR_DATA+           ; Load x(n)  (32 bits)
     MVDD   *pINBUF+,*pIIR_DATA-           ; in the data buffer
     MAC    *pIIR_DATA+0%,*pIIR_COEFF,B    ; A0*x(n)high
     MACSU  *pIIR_DATA+0%,*pIIR_COEFF+,A   ; A0*x(n)low
     MAC    *pIIR_DATA+0%,*pIIR_COEFF,B    ; -B2*y(n-2)high
     MACSU  *pIIR_DATA+0%,*pIIR_COEFF+,A   ; -B2*y(n-2)low
     MAC    *pIIR_DATA+0%,*pIIR_COEFF,B    ; -B1*y(n-1)high
     MACSU  *pIIR_DATA+0%,*pIIR_COEFF,A    ; -B1*y(n-1)low
     ADD    A,-16,B
     MAR    *+pIIR_COEFF(-4)               ; pIIR_COEFF points on A2
     STH    B,*pIIR_DATA+0%                ; Store y(n) in the buffer
     STL    B,*pIIR_DATA+0%
     DST    B,*pOUTBUF+                    ; Store y(n) in the output

IirFilterLoopEnd
```

We use the RPTBD instruction that permits the repetition of a block of instructions by the number of times specified by the memory-mapped block-repeat counter (BRC). BRC must be loaded before the execution of this instruction and, if the loaded value is equal to N, the block of instructions will be executed N+1 times. We load the counter BRC with the total number of input samples. The RPTB instruction allows the parallel execution of one 2-word instruction or two 1-word instructions following the RPTBD instead of flushing the pipeline. These instructions cannot belong to the repeated block of instructions. The RPTBD instructions effectively execute in 2 cycles. To have a better accuracy, all the multiplications and additions on the low data are realized before doing the 16-bit right shift to keep only the 16 MSB.

In order to use the double load instruction DLD to load a 32-bit input sample in one of the two accumulator and then two instructions STL and STH to store the input data into the correct location in the data buffer, we have used two instructions MVDD. These instructions make it possible to move the 32-bit input data directly into the data buffer location (a dual data-memory operand is needed). We can use either a double store instruction DST or the two instructions STL and STH to store the 32-bit result.

One instruction MAR *+pIIR_COEFF(-4) is used to restart the pointer AR4 at the beginning of the coefficients vector stored in memory.

5.1.5 Performances

The performance of the loop implementing the equations of a second order IIR filter in a 32x16-bit Direct Form I is, for each input sample:

Number of cycles	RAM	ROM	Number of registers
19 cycles	15 words	18 words	5

5.2 Implementation of the 32x16-bit Direct Form II

For the implementation of this form, a circular buffer can be used for the coefficients.

In this paragraph an example of a second order IIR filter in a Direct Form II is described with coefficients in a Q15 format and using two circular buffers, one for handling the internal data and the other for handling the coefficients. The AR0 auxiliary register is used for an indexed addressing mode.

For this form, in contrast to the Direct Form I, a 2-bit right shift is used to reduce the input level thus preventing an eventual overflow of the internal data d(n) that have to be stored. Before storing the output of the filter, a 2-bit left shift is applied to restore the output level.

For the implementation of the Direct Form I, the intermediate calculations are stored in an accumulator. So, an eventual internal overflow due to the numerator computations of the filter z-transform is allowed (8 guard bits in an accumulator). The calculations of the denominator then bring down the output value. As the intermediate calculations are not

stored in memory but are kept in an accumulator, there is no problem in terms of internal overflow.

5.2.1 Memory organization

The memory organization for the coefficients circular buffer and the internal data circular buffer is as follows:

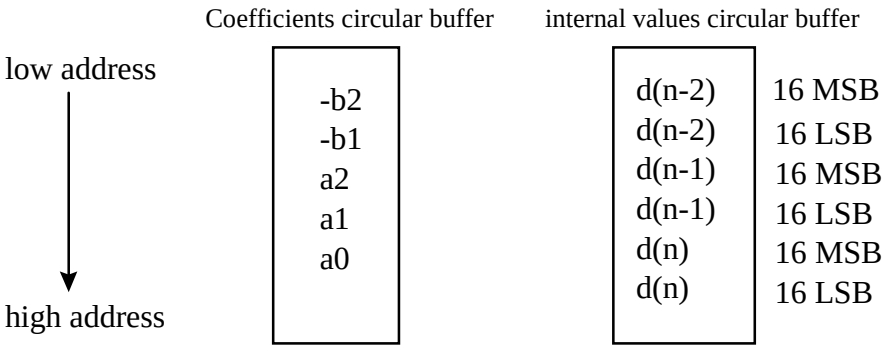

Figure 9: Memory organization

The 32-bit values d(i) are stored in memory using two 16-bit words for each.

Data circular buffer update is performed as follows:

- Data pointer points to d(n-2) before computation. Then $d(n) = x(n) - b_2 d(n-2) - b_1 d(n-1)$ is calculated and stored in the data buffer after the values of *d(n-1)*.
- The data pointer is restarted on the d(n-2) and the output is calculated: $y(n) = a_2 d(n-2) + a_1 d(n-1) + a_0 d(n)$.
- Finally the pointer is set on *d(n-1)* which becomes *d(n-2)*, *d(n)* becomes *d(n-1)* and the new *d(n)* calculated is stored instead the old *d(n-2)* value.

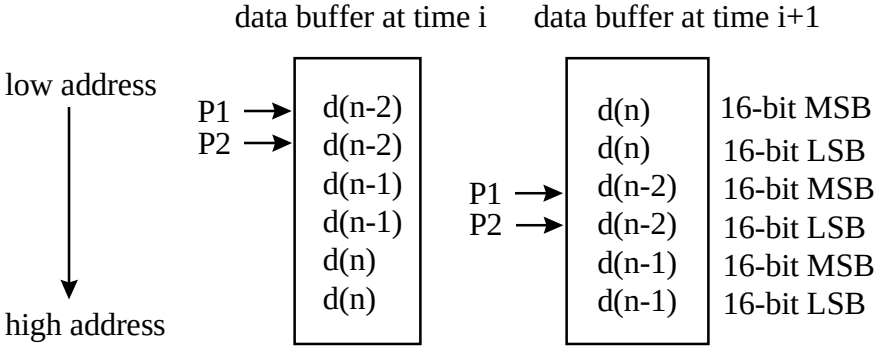

Figure 10: Data circular buffer update

where P1 is the pointer on the MSB of the current data and P2 is the pointer on the LSB of the current data.

The size of the data buffer is one 16-bit word larger than the coefficients buffer. One word is not used at the end of the coefficients buffer so to jump over this word and go back to the beginning of the buffer, the pointer is incremented by the INDEX value (index of the circular buffers). This is set to 2 (AR0 = 2) in order to jump within the data buffer from one 16 MSB word (or LSB) to the next one.

5.2.2 Program organization

As in the previous Direct Form I, to implement the Direct Form II, seven auxiliary registers are required as followed :

- AR2 (dual access) points to the low internal values (16 bits). This is declared as IIR_DATA_P_L .
- AR3 (dual access) points to the high internal values (16 bits). This is declared as IIR_DATA_P_H.
- AR4 and AR5 (dual access) point to the coefficients. We need two registers, one for the multiplication with the low 16-bit of the internal values and another one for the multiplication with the high 16-bit of the internal values. These are declared as IIR_COFF_P_1 and IIR_COFF_2.
- AR6 points to the inputs. This is declared as INBUF_P.
- AR7 points to the outputs. This is declared as OUTBUF_P.
- and AR0 indexes the two circular buffers. This is declared as IIR_INDEX.

The 32-bit output result is in the Accumulator B before being stored.

The entry and output conditions are SXM=1 (sign extended mode), OVM=0 (to allow overflow mode) and FRCT=1 (fractional mode) in the status register ST1.

5.2.3 Program explanation

The full program is given in the appendix. Nevertheless some explanation is given here of the main instructions, which are:

```
     RPTBD  iir_filter_loop_end-1
     STM    #(d_filout),OUTBUF_P           ; Load output address memory
     DLD    *INBUF_P+,A                    ; Load in A the new Q31 sample
     LD     A,-2,A                         ; pre scaling
; feedback_path
     LD     #0,B
     MACSU  *IIR_DATA_P_L+0%,*IIR_COFF_P_1+,B    ; -b2*d(n-2)low
     MACSU  *IIR_DATA_P_L+0%,*IIR_COFF_P_1+,B    ; -b1*d(n-1)low
     ADD    B,-16,A                ;x(n)+(-b2*d(n-2)low-b1*d(n-1)low)>>16

     MAC    *IIR_COFF_P_2+,*IIR_DATA_P_H+0%,A    ;-b2*d(n-2)high
     MAC    *IIR_COFF_P_2+,*IIR_DATA_P_H+0%,A    ;-b1*d(n-1)high
     STL    A,*IIR_DATA_P_L+0%             ; Store the 32-bit result d(n)
     STH    A,*IIR_DATA_P_H+0%
;forward_path
     LD     #0,B
     MACSU  *IIR_DATA_P_L+0%,*IIR_COFF_P_1+,B    ;a2*d(n-2)low
     MACSU  *IIR_DATA_P_L+0%,*IIR_COFF_P_1+,B    ;a1*d(n-1)low
```

```
      MACSU   *IIR_DATA_P_L+0%,*IIR_COFF_P_1+0%,B  ,a0*d(n)low
      LD      B,-16,B               ;a0*d(n)low+a1*d(n-1)low+a2*d(n-2)low>>16

      MAC     *IIR_COFF_P_2+,*IIR_DATA_P_H+0%,B    ;a2*d(n-2)high
      MAC     *IIR_COFF_P_2+,*IIR_DATA_P_H+0%,B    ;a1*d(n-1)high
      MAC     *IIR_COFF_P_2+0%,*IIR_DATA_P_H+0%,B  ;a0*d(n)high

      MAR     *IIR_DATA_P_L+0%                     ; Update d(n-2)=d(n-1)
      MAR     *IIR_DATA_P_H+0%                     ; Update d(n-1)=d(n)
      LD      B,2,B                                ; post scaling
      STH     B,*OUTBUF_P+                         ; Store the low and high
      STL     B,*OUTBUF_P+                         ; part of the Q31 result
iir_filter_loop_end
```

As in the previous Direct Form I, the RPTBD instruction is used. The double load instruction DLD is used to load the input data in one cycle. In this form, only one circular buffer is used for handling coefficients. In this case, to restart the pointer on the beginning of the coefficients buffer, in the last MAC and MACSU instructions the addresses in the auxiliary registers AR4 and AR5 are incremented by the content of AR0 using circular addressing (*pIIR_COFF_P_1+0% et *pIIR_COFF_P_2+0%). This enables again of two instructions STM to restart the pointers on the beginning of the coefficients buffer.

5.2.4 Performances

The performance of the loop implementing the equations of a second order IIR filter in a 32x16-bit Direct Form II is,for each input sample:

Number of cycles	RAM	ROM	Number of registers
23 cycles	11 words	23 words	7

5.3 Implementation of the 32x16-bit Cascade Form

The advantage of an IIR cascade form of 2nd order structure is its modularity. This permits an efficient implementation which limits overflow problems. The optimal cascade form uses second-order filters implemented in a Direct Form I as illustrated in the following figure:

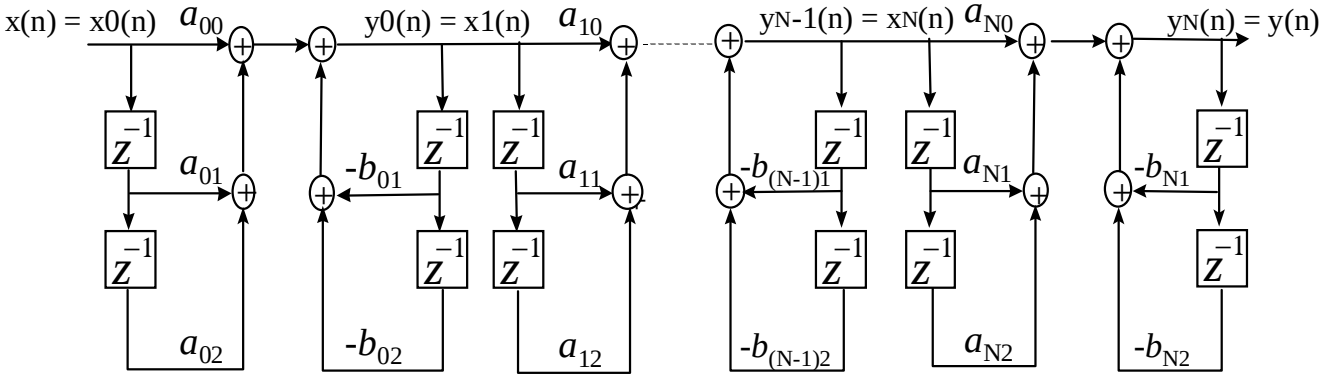

Figure 11: Optimized IIR Cascade Form

The alternating calculation of zeros and poles limits the overflow problem on the stored data (i.e. *xi(n-j)*). Each second order IIR uses Direct Form I. The number of taps is reduced by combining the poles of a second order IIR with the zeros of the next second order IIR. This reduction gives us the following canonical direct form.

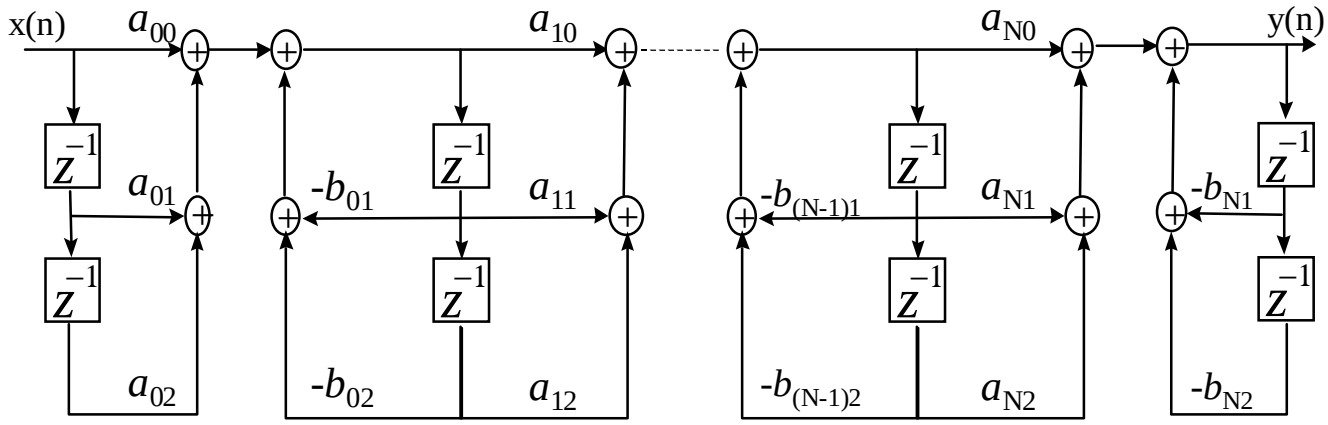

Figure 12: Canonical Direct Form

The implementation of this form has low cost in terms of memory (because we use the same delay line for the calculation of the poles of a filter and the zeroes of the next one) and also in terms of number of cycles.

5.3.1 Memory organization

One circular buffer is used for handling the input data. The coefficients buffer can be a circular one but the size of the input data buffer is larger than the coefficients buffer. To restart the auxiliary register which points to the coefficients buffer, it will take us more than the two instructions STM we have previously taken.

The AR0 auxiliary register is used to adopt an indexed addressing mode. It is set to 2 in order to jump within the data buffer from one 16 MSB word (or LSB) to another.

The 32-bit values $x_j(i)$ are stored in memory using two 16-bit words. The size of the coefficients buffer is equal to $5 \times Nbf$ where *Nbf* is the number of filters, each filter needing 5 coefficients (*a0, a1, a2, -b1, and -b2*). The size of the circular buffer of input data is equal to:

$$3 \times 2 \times Nbf + 4$$

the three data *x(n)* , *x(n-1)* and *x(n-2)* are required (the output of a filter is the input of the next one $y_{i-1}(n)=x_i(n)$)

every sample is stored in two 16-bit words.

for the 32-bit taps of the last filter $y_N(n-1)$ and $y_N(n-2)$ and the output $y_N(n)$ can be stored instead of the old $x_0(n-2)$ value.

The memory organization of the cascade form is given in Figure 13.

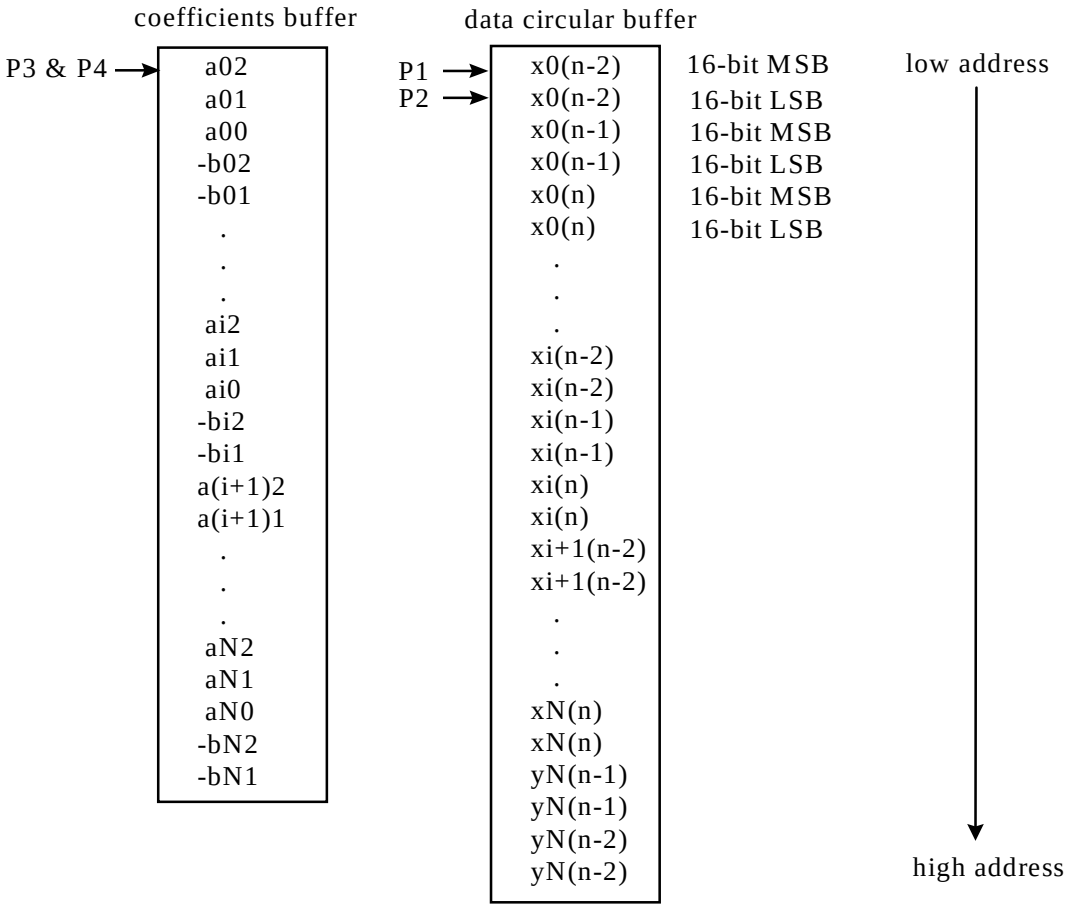

Figure 13: Memory organization of the Cascade Form

Data circular buffer update is implemented as follows:

- Data pointer needs to be on x0(n-2) before computation. Then the output $y(n) = a_2 x(n-2) + a_1 x(n-1) + a_0 x(n) - b_1 y(n-1) - b_2 y(n-2)$ (for one input sample) is calculated, and the pointers P1 and P2 are set on $x_0(n-1)$ which becomes $x_0(n-2)$ and the $x_0(n)$ becomes $x_0(n-1)$ and so on $x_i(n)$ becomes $x_i(n-1)$, ...
- the output $y_N(n)$ is stored instead of the old $x_0(n-2)$,

- and a new sample $x_0(n)$ will be stored instead of the old $x_1(n-2)$ value.

This update is shown in Figure 14.

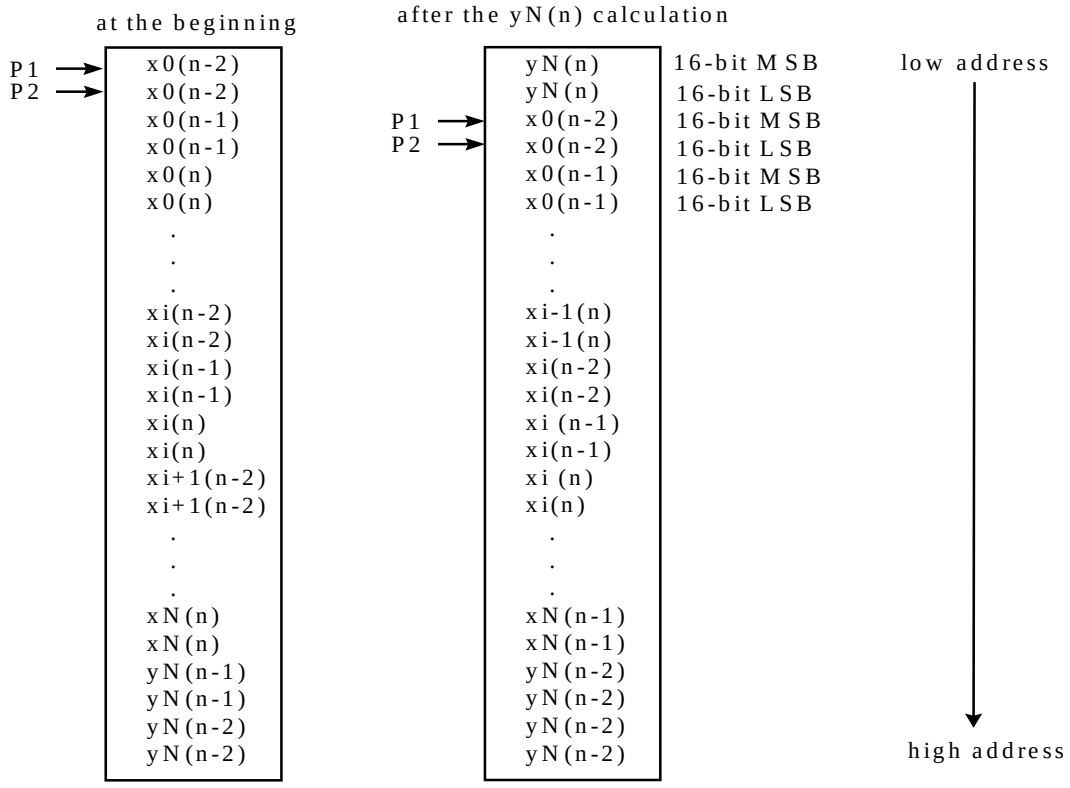

Figure 14: Data circular buffer update

where P1 is the pointer on the current data high and P2 is the pointer on the current data low.

5.3.2 Program organization

In order to decrease the number of cycles and to increase the accuracy, we have grouped the signed/unsigned multiplications in one side before carrying out the truncation (the right 16-bit shift) and the signed multiplications in the other side.

This implies that there can be an overflow. The coefficients have been calculated such that the filter does not have a gain greater than 1. However, if an overflow occurs before d(n) has been stored, then the value that is updated in the data buffer is wrong. To prevent an internal overflow, it is usually necessary to scale down the input signal. We have scaled the input value by a 2-bit left shift. And then, the output value is scaled by a 2-bit right shift.

We need seven auxiliary registers as follows :

- AR2 (dual data-memory addressing) to point to the low internal values (16 bits). This is declared as pEQZ_DATA_L .

- AR3 (dual data-memory addressing) to point to the high internal values (16 bits). This is declared as pEQZ_DATA_H.
- AR4 and AR5 (dual data-memory addressing) to point to the coefficients. We need two registers, one for the multiplication with the low 16-bit of the internal values and another one for the multiplication with the high 16-bit of the internal values. These are declared as pEQZ_COFF_1 and pEQZ_COFF_2.
- AR6 to point to the inputs. This is declared as pINBUF.
- AR7 to point to the outputs. This is declared as pOUTBUF.
- AR1 to handle the number of input samples.
- And AR0 to index the two circular buffers. This is declared asIIR_INDEX.

The 32-bit output result is in the Accumulator A before being stored.

The entry and output conditions are SXM=1 (sign extended mode), OVM=0 (to allow overflow mode) and FRCT=1 (fractional mode) in the status register ST1.

AR2 and AR3 are the previous pointers P1 and P2.

5.3.3 A typical application: an equalizer

An equalizer can be implemented as a cascade form of 2^{nd} order IIR filters.

The equalizer program given in the appendix uses a cascade form of five 2^{nd} order Direct Form I IIR filters using one circular buffer (for data). The coefficients are calculated to give an equalization between -12dB and 12dB. To prevent an eventual overflow, a 2-bit right shift is applied (a division by 4) to bring down the level between -24dB and 0dB (because 20log(1/4) = -12dB) in order to allow a maximum gain of 4. Each filter is a pass-band filter centered on one of the frequencies of interest. For the coefficients given in the example in the appendix, these are: 100 Hz, 330 Hz, 1 kHz, 3.3 kHz and 10 kHz.

The coefficients of the filters may be greater than 1 in absolute value. To represent them in a Q15 format, they must be divided by the correct power of two in order to bring them back into the range $[-1, (1-2^{-15})]$. The right shifts of the coefficients can be applied before storing them in memory. Therefore, a left shift corresponding to the same power of two must be performed before storing the output of each filter (we have used the instruction: LD B,1,B).

We do all the multiplications and additions with the low data such that:

$a2i*Xi_{low}(n-2) + a1i*Xi_{low}(n-1) + a0i*Xi_{low}(n) - b2i*Yi_{low}(n-2) - b1i*Yi(n-1)$ before making the 16-bit right shift to keep only the 16 MSB.

Figure 11 shows a periodic structure which is implemented with a conditional loop. This loop is improved thanks to the BANZD instruction.

```
Loop:
     MACSU  *pEQZ_DATA_L+0%,*pEQZ_COFF_1+,A      ;-b2/2*ylow(n-2)
     MACSU  *pEQZ_DATA_L,*pEQZ_COFF_1+,A         ;-b1/2*ylow(n-1)
     ADD    A,-16,B
     MAC    *pEQZ_DATA_H+0%,*pEQZ_COFF_2+,B      ;-b2/2*yhigh(n-2)
```

```
      MAC     *pEQZ_DATA_H,*pEQZ_COFF_2+,B        ;-b1/2*yhigh(n-1)
      MAR     *pEQZ_DATA_L-0%
      MAR     *pEQZ_DATA_H-0%
      MPY     *pEQZ_DATA_H+0%,*pEQZ_COFF_2+,A     ;a2/2*xhigh(n-2)next
      MAC     *pEQZ_DATA_H+0%,*pEQZ_COFF_2+,A     ;a1/2*xhigh(n-1)next
      LD      B,1,B
      DST     B,*pEQZ_DATA_H
      MAC     *pEQZ_DATA_H+0%,*pEQZ_COFF_2+,A     ;a0/2*xhigh(n)next
      LD      A,B
      LD      #0,A
      MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF_1+,A     ;a2/2*xlow(n-2)next

      BANZD   Loop,*EQZ_NB-
      MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF_1+,A     ;a1/2*xlow(n-1)next
      MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF_1+,A     ;a0/2*xlow(n)next
EndLoop
```

The first part of the first filter and the last part of the last filter are realized respectively before and after this conditional loop. The complete cascade structure is computed for each input sample using a RPTBD instruction.

To restart the pointers at the beginning of the coefficients memory location we use the two instructions MAR *+pEQZ_COFF_1(-24) and MAR *+pEQZ_COFF_2(-24).

5.3.4 Performances

The performance of the loop implementing the equations of an IIR filter in a 32x16-bit Cascade Form of N 2nd order is, for each input sample:

Number of cycles	RAM	ROM	Number of registers
25+18*N cycles	5*N + 6*N+4 words	24+17*N words	8

5.4 Implementation of the 32x32-bit Direct Form I

The complete code for the implementation of the 32x32-bit Direct Form I is given in the appendix. This code is similar to the one in the 32x16-bit implementation. Nevertheless, a comparison between the two Direct Form I implementations 32x16-bit and 32x32-bit in terms of memory organization and resources can be given in this paragraph.

5.4.1 Memory Organization

Both coefficients buffer and data buffer are circular because they have the same size.

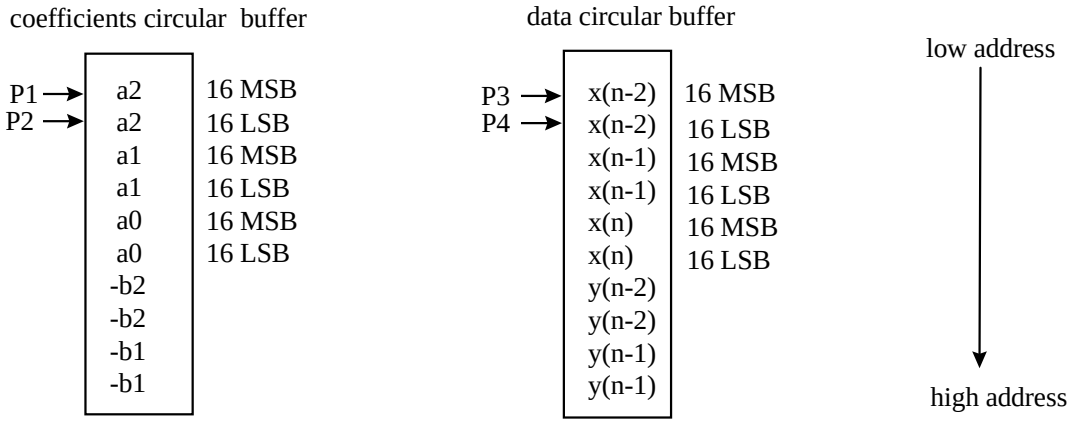

Figure 15: Memory organization of the 32x32-bit Direct Form

As for the 32x16-bit implementation, each pointer is used to address a dual data-memory operand because of the MACSU instruction which needs two dual data-memory operands. But here, two pointers are used to handle the 32-bit coefficients. As there are only four auxiliary registers that permit a dual data-memory addressing, the MVDD instruction cannot be used to acquire the 32-bit input value x(n). This instruction needs two dual data-memory operands. Thus, the instruction DLD is used to permit this acquisition and then a DST instruction is used to store this data at the right location in the memory buffer.

5.4.2 Resources

For this implementation, four pointers are required in order to improve the multiplications-additions L1xH2 + H1xL2 for each coefficient and data that have to be multiplied together. L1 is the 16-bit word corresponding to the coefficients LSB and H1 is the 16-bit word corresponding to the coefficients MSB. Apart from the data, L2 and H2 are handled the same way. All these multiplications-additions have to be performed before discarding the 16 LSB of the result (with a 16-bit shift) to have better accuracy. The result of these products with a 16-bit shift is stored in the accumulator B. Then, before starting the first multiplication H1xH2, the pointers are returned to the beginning of the circular buffers. These multiplications-accumulations H1xH2 are improved for each coefficient and data with the pointers P1 and P3 which are indexed by the content of AR0. They are stored in the accumulator A. The final result is A+B.

5.4.3 Performances

The performance of the loop implementing the equations of a 2^{nd} order IIR filter in a 32x32-bit Direct Form I is, for each input sample:

Number of cycles	RAM	ROM	Number of registers
24 cycles	20 words	22 words	7

5.5 Implementation of the 32x32-bit Direct Form II

The complete code for the implementation of the 32x32-bit Direct Form II is given in the appendix. A comparison between the two Direct Form II implementations 32x16-bit and 32x32-bit in terms of memory organization and resources is given in this paragraph.

5.5.1 Memory Organization

The two buffers, one for data one and one for coefficients, cannot be circular because they do not have the same size. Thus, only the data buffer is circular.

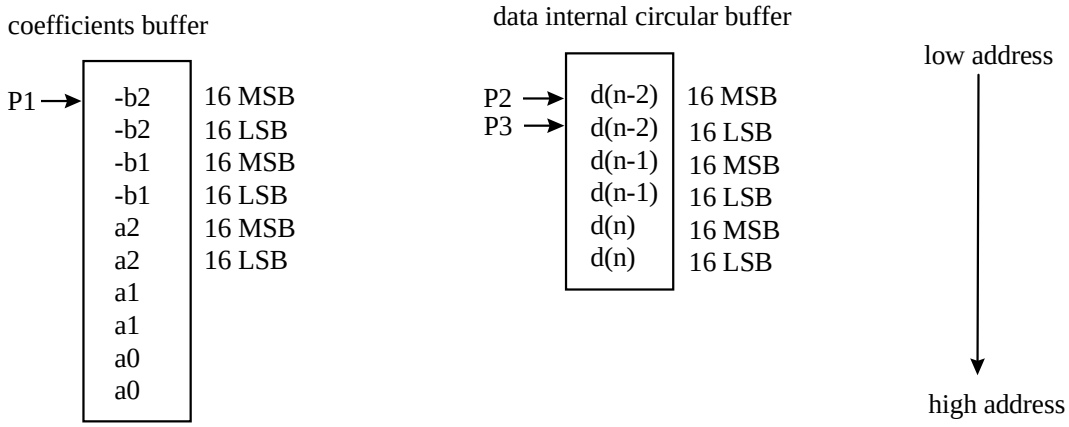

Figure 16: Memory organization of the 32x32-bit Direct Form II

5.5.2 Resources

Three pointers are required for this implementation. The input is stored in the accumulator A to be added to -b2xd(n-2)-b1xd(n-1) to calculate the value d(n). Thus, for this form no special resource is required for handling the input value.

As with the 32x16-bit implementation, a 2-bit right shift on the input value is used to bring down the input level, thus preventing an eventual overflow of the internal data d(n) that would have to be stored. Before storing the output of the filter, a 2-bit left shift is applied to restore the output level.

5.5.3 Performances

The performance of the loop implementing the equations of a 2nd order IIR filter in a 32x32-bit Direct Form II is, for each input sample:

Number of cycles	RAM	ROM	Number of registers
29 cycles	16 words	29 words	6

5.6 Implementation of the 32x32-bit Cascade Form

The complete code for the implementation of the 32x32-bit Direct Form II is given in the appendix. A comparison between the two Cascade Form implementations 32x16-bit and 32x32-bit in terms of memory organization and resources is given in this paragraph.

5.6.1 Memory Organization

The coefficients buffer and the data buffer have a different sizes and only the data buffer is circular.

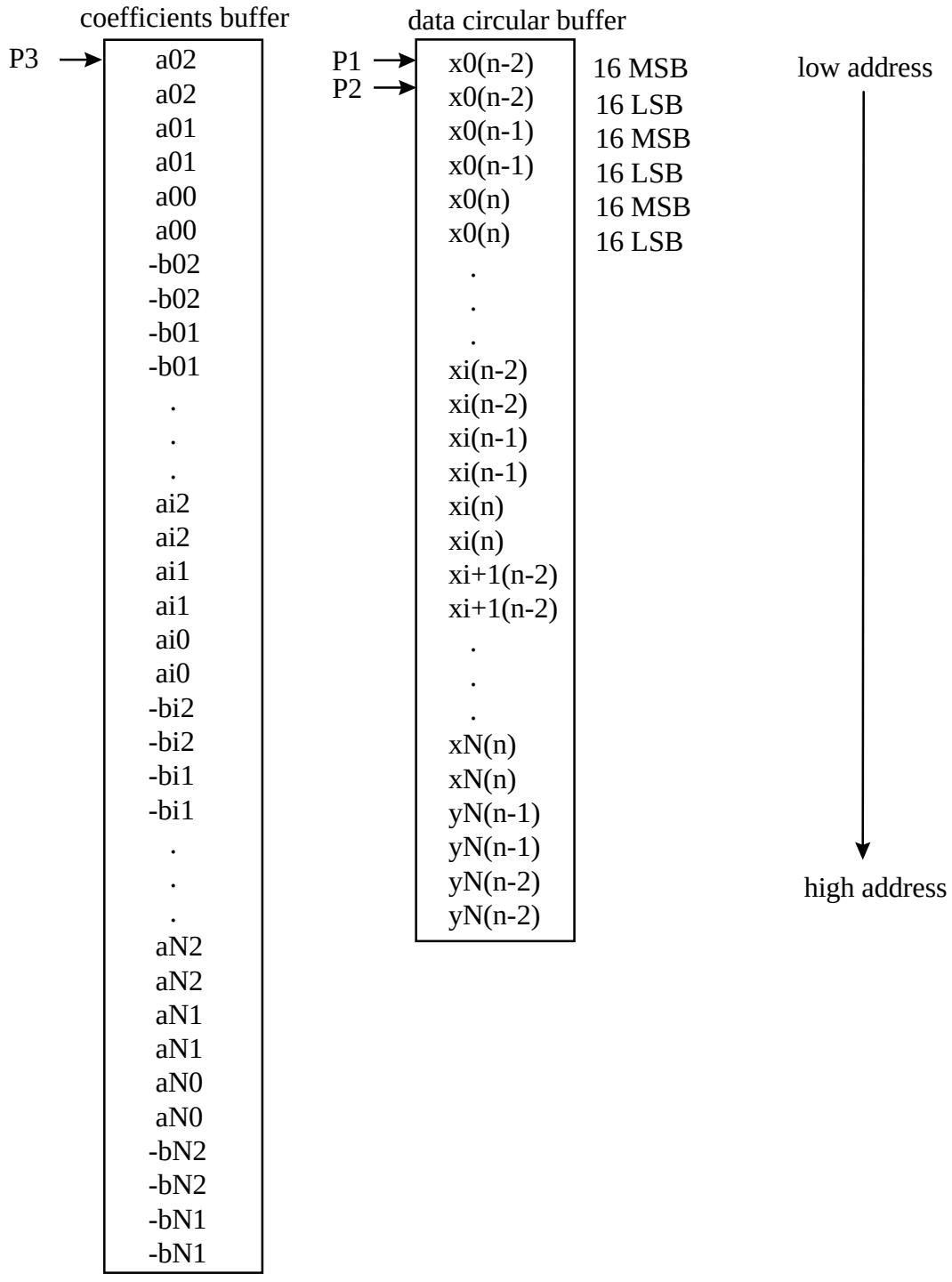

Figure 17: Memory organization of the 32x32-bit Cascade Form

5.6.2 Resources

In this implementation, unlike the 32x16-bit one, only three auxiliary registers are needed. But, to ensure the correct multiplications-additions for each 32-bit coefficient and 32-bit data, the two accumulators are used. For the input, a temporary buffer is then used. When the two pointers P1 and P2 are in the correct, location the temporary buffer is stored in this memory location with two instructions MVDD.

In contrast to the 32x16-bit implementation in the 32x32-bit implementation, the MAC instructions have to be located after each second instruction MACSU to improve the H1xH2 multiplications. Thus, for each set of data and coefficients, there are in the program two instructions MACSU permitting the L1xH2 and the H1xL2 multiplications, followed by a MAC instruction allowing the H1xH2 multiplication. Otherwise, three pointers on the data buffer can be used. In this case several MAR instructions have to be added in the code to store the input data in memory and to handle the coefficients pointer to improve the H1xH2 multiplications-additions. This is because the coefficients buffer is not circular and a dual data-memory operand can be incremented only with an index using circular addressing.

5.6.3 Performances

The performance of the loop implementing the equations of an IIR filter in a 32x32-bit Cascade Form of N 2nd order is, for each input sample:

Number of cycles	RAM	ROM	Number of registers
31+22*N cycles	10*N + 6*N+4 words	29+22*N words	8

5.7 Implementation of the 16x32-bit Direct Form I

For the implementations 16x32-bit, only the Direct Form I structure is described in this note because they are very similar to the 32x16-bit implementations.

The complete code for the implementation of the 16x32-bit Direct Form I is given in the appendix. A comparison between the two Direct Form I implementations 32x16-bit and 16x32-bit in terms of memory organization and resources is given in this paragraph.

5.7.1 Memory Organization and resources

The memory organization is the inverse of the one used for the 32x16-bit implementation. Three pointers are still used in this implementation.

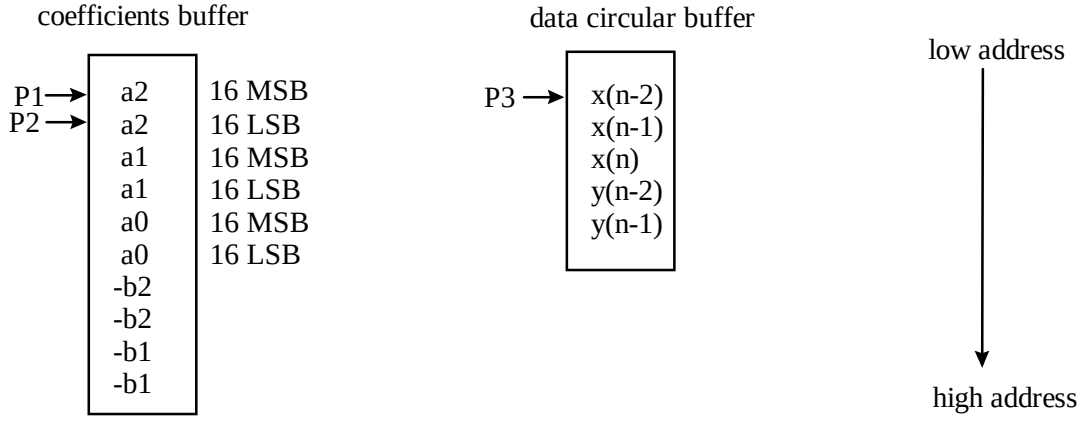

Figure 18: Memory organization of the 16x32-bit Direct Form

But here, two pointers are used to handle the 32-bit coefficients.

5.7.2 Program explanation

In this implementation only one instruction MVDD is needed to store the input value x(n) in memory. To store the y(n) value in the data buffer only one instruction STH is required instead of two instructions STH and STL. And finally, to store the y(n) value in the output value, one instruction STH (1 cycle) replaces the DST instruction (2 cycles).

```
     RPTBD  IirFilterLoopEnd-1
     STM    #(DataFilout),pOUTBUF          ; Load output address memory

     LD     #0,A
     MPY    *pIIR_COEFF+,*pIIR_DATA,B      ; A2high*x(n-2)
     MACSU  *pIIR_COEFF+,*pIIR_DATA+0%,A   ; A2low*x(n-2)
     MAC    *pIIR_COEFF+,*pIIR_DATA,B      ; A1high*x(n-1)
     MACSU  *pIIR_COEFF+,*pIIR_DATA+0%,A   ; A1low*x(n-1)
     MVDD   *pINBUF+,*pIIR_DATA            ; load x(n) in the buffer
     MAC    *pIIR_COEFF+,*pIIR_DATA,B      ; A0high*x(n)
     MACSU  *pIIR_COEFF+,*pIIR_DATA+0%,A   ; A0low*x(n)
     MAC    *pIIR_COEFF+,*pIIR_DATA,B      ; -B2high*y(n-2)
     MACSU  *pIIR_COEFF+,*pIIR_DATA+0%,A   ; -B2low*y(n-2)
     MAC    *pIIR_COEFF+,*pIIR_DATA,B      ; -B1high*y(n-1)
     MACSU  *pIIR_COEFF,*pIIR_DATA+0%,A    ; -B1low*y(n-1)
     ADD    A,-16,B
     MAR    *+pIIR_COEFF(-9)               ; pIIR_COEFF points on A2high
     STH    B,*pIIR_DATA+0%                ; Store y(n) in the buffer
     STH    B,*pOUTBUF+   ; Store y(n) in the output
IirFilterLoopEnd
```

5.7.3 Performances

The performance of the loop implementing the equations of an IIR filter in a 16x32-bit Direct Form I is, for each input sample:

Number of cycles	RAM	ROM	Number of registers
16 cycles	15 words	16 words	5

References

[1] J. G. Proakis & D. G. Manolakis, Digital Signal Processing - Principles, Algorithms, and Applications, Prentice-Hall, Third Edition 1996.

[2] G. Marven & G. Ewers, Digital Signal Processing, Texas Instruments, 1993.

[3] A. Bateman & W. Yates, Digital Signal Processing Design, Pitman Computer, Systems Series, 1988.

[4] TMS320C54x DSP CPU and Peripherals, Texas Instruments, Reference Set Volume 1, 1996.

[5] TMS320C54x DSP Mnemonic Instruction Set, Texas Instruments, Reference Set Volume 2, 1996.

[6] M. Bellanger, Traitement Numérique du Signal - Théorie et Pratique, Masson et CNET-ENST, Paris, 1980-1996.

[7] R. Boite & H. Leich, Les Filtres Numériques - Analyse et Synthèse des filtres unidimensionnels, Masson 1982.

Appendix A: The implementation of an IIR 32x16-bit Direct Form I on ‘C54x

```
; TEXAS INSTRUMENTS FRANCE
;
; AUTHOR        MESSINA  Nathalie
;               CAVALIER Philippe
;
; (C) Copyright 1997. Texas Instruments France. All rights reserved
;

*********************************************************
*                  macro definitions                    *
*********************************************************
        .mmregs
        .include  "init54x.inc"

*********************************************************
*                    reset/interrupt/trap vectors       *
*********************************************************
*
* Always start from Reset.
*

        .global Start


        .sect "vecs"
Start
        BD      Init                            ; Branch to MainLine.
        NOP
        NOP

*********************************************************
*         Set up constant and filter coeff              *
*********************************************************

K_FRAME_SIZE          .set 128                  ; Number of samples
K_NB_FILTER           .set 1                    ; Number of 2nd-order IIR filter
K_BUFFER_INDEX        .set 1                    ; Circ buff index
K_BUFFER_SIZE         .set (6*(K_NB_FILTER+1)-2) ; Circ buff size

BPFilterCoffTable     .sect"iir_coff"
        .word   0100Ah                          ; A2
        .word   02014h                          ; A1
        .word   0100Ah                          ; A0
        .word   0DC01h                          ; -B2
        .word   063D6h                          ; -B1


*********************************************************
*                 Set up in/out buffer                  *
*********************************************************
DataFilin       .usect "iir_vars",256           ; for 128 samples
DataFilout      .usect "iir_vars",256           ; for 128 samples
DataInternal    .usect "iir_bfr",K_BUFFER_SIZE
DataTempBuff    .usect "iir_bfr",1
```

```
*********************************************************
*                     Init section                      *
*********************************************************
        .text
Init
        LD      #0,A                            ; acc = >00000000.
        STLM    A,SWWSR                         ; 0 Wait States.
        STLM    A,BSCR                          ; Bank shift.
        STM     #K_ST0,ST0
        STM     #K_ST1,ST1
        STM     #K_PMST,PMST
*********************************************************
*                                                       *
*                     Main program                      *
*                                                       *
*********************************************************

Main
        CALL    IirFilterInit
Continue
        NOP
        NOP
        CALL    IirFilterTask
        NOP
        NOP
        B       Continue

*********************************************************
*                                                       *
*                      Sub routines                     *
*                                                       *
*********************************************************

        .asg    AR0,IIR_INDEX                   ; Circ buff index AR0=1
        .asg    AR2,pIIR_DATA                   ; data circular buffer pointer
        .asg    AR3,pIIR_COEFF                  ; Coefficient pointer
        .asg    AR4,pINBUF                      ; Input data
        .asg    AR5,pOUTBUF                     ; Output data
        .sect   "iir_prog"

IirFilterInit
        STM     #DataInternal,pIIR_DATA
        RPTZ    A,#(K_BUFFER_SIZE-1)            ; Clear the internal data
        STL     A,*pIIR_DATA+                   ; data buffer
        STM     #DataInternal,pIIR_DATA         ; pIIR_DATA points on the
                                                ; first high word of the
                                                ; data buffer
        STM     #BPFilterCoffTable,pIIR_COEFF
        STM     #K_BUFFER_SIZE,BK               ; Load circ buff size
        RETD
        STM     #K_BUFFER_INDEX,IIR_INDEX       ; Load circ buff index


;---------------------------------------------------------------------

IirFilterTask
        STM     #(DataFilin),pINBUF             ; Load input add mem

IirFilterBegin
```

```
        STM     #(K_FRAME_SIZE-1),BRC
        RPTBD   IirFilterLoopEnd-1
        STM     #(DataFilout),pOUTBUF             ; Load output add mem

        LD      #0,A
        MPY     *pIIR_DATA+0%,*pIIR_COEFF,B       ; A2*x(n-2)high
        MACSU   *pIIR_DATA+0%,*pIIR_COEFF+,A      ; A2*x(n-2)low
        MAC     *pIIR_DATA+0%,*pIIR_COEFF,B       ; A1*x(n-1)high
        MACSU   *pIIR_DATA+0%,*pIIR_COEFF+,A      ; A1*x(n-1)low
        MVDD    *pINBUF+,*pIIR_DATA+              ; Load x(n) (32 bits)
        MVDD    *pINBUF+,*pIIR_DATA-              ; in the buffer
        MAC     *pIIR_DATA+0%,*pIIR_COEFF,B       ; A0*x(n)high
        MACSU   *pIIR_DATA+0%,*pIIR_COEFF+,A      ; A0*x(n)low
        MAC     *pIIR_DATA+0%,*pIIR_COEFF,B       ; -B2*y(n-2)high
        MACSU   *pIIR_DATA+0%,*pIIR_COEFF+,A      ; -B2*y(n-2)low
        MAC     *pIIR_DATA+0%,*pIIR_COEFF,B       ; -B1*y(n-1)high
        MACSU   *pIIR_DATA+0%,*pIIR_COEFF,A       ; -B1*y(n-1)low
        ADD     A,-16,B
        MAR     *+pIIR_COEFF(-4)                  ; pIIR_COEFF points on A2
        STH     B,*pIIR_DATA+0%                   ; Store y(n) in the buffer
        STL     B,*pIIR_DATA+0%
        DST     B,*pOUTBUF+                       ; Store y(n) in the output

IirFilterLoopEnd
        RET
        .end
```

Appendix B: The implementation of an IIR 32x16-bit Direct Form II on 'C54x

```
; TEXAS INSTRUMENTS FRANCE


; AUTHORS       CAVALIER Philippe
;               ERCOLE   Damien
;
; (C) Copyright 1997. Texas Instruments France. All rights reserved
;
;
*****************************************************************
*     Macro definitions                                         *
*****************************************************************
        .mmregs
        .include "init54x.inc"

*****************************************************************
*        Reset/interrupt/trap vectors                           *
*****************************************************************
*
* Always start from Reset.
*

        .global start
        .sect   "vecs"
start
BD      init                                    ; Branch to MainLine.
        NOP
        NOP


*****************************************************************
*        Set up constant and filter coeff                       *
*****************************************************************

K_FRAME_SIZE    .set   128                      ; Number of samples
K_IIR_INDEX     .set   2                        ; Circ buff index
K_IIR_BFFR      .set   6                        ; Circ buff size
iir_coff_table  .sect  "iir_coff"
        .word   0DC01h      ; -B2
        .word   063D6h      ; -B1
        .word   0100Ah      ; A2
        .word   02014h      ; A1
        .word   0100Ah      ; A0

*****************************************************************
*        Set up in/out buffer                                   *
*****************************************************************
d_filin         .usect "iir_vars",256           ; for 32 samples
d_filout        .usect "iir_vars",256           ; for 32 samples
d_iir_dn        .usect "iir_bfr",6              ; to store d(n),d(n-1),d(n-2)

*****************************************************************
*        Init section                                           *
*****************************************************************
        .text
init
```

```
LD      #0,A                            ; acc = >00000000.
STLM    A,SWWSR                         ; 0 Wait States.
STLM    A,BSCR      ; Bank shift.
STM     #K_ST0,ST0
STM     #K_ST1,ST1
STM     #K_PMST,PMST
```

```
*****************************************************************
*                                                               *
*            Main program                                       *
*                                                               *
*****************************************************************

Main
        CALL    iir_init
Continue
        NOP
        NOP
        CALL    iir_task
        NOP
        NOP
        B       Continue
*****************************************************************
*                                                               *
*       Sub routines                                            *
*                                                               *
*****************************************************************

        .asg    AR0,IIR_INDEX                   ; Circ buff index
        .asg    AR2,IIR_DATA_P_L                ; Low word of internal value
        .asg    AR3,IIR_DATA_P_H                ; High word of internal value
        .asg    AR4,IIR_COFF_P_1                ; Coefficient pointer 1
        .asg    AR5,IIR_COFF_P_2                ; Coefficient pointer 2
        .asg    AR6,INBUF_P                     ; Input data
        .asg    AR7,OUTBUF_P                    ; Output data
        .sect   "iir_prog"

iir_init
        STM     #d_iir_dn,IIR_DATA_P_L
        RPTZ    A,#(K_IIR_BFFR-1)             ; Clear the internal
        STL     A,*IIR_DATA_P_L+              ; value buffer
        STM     #d_iir_dn+1,IIR_DATA_P_L
                                              ; IIR_DATA_P_L points on the
                                              ; first low word of the internal value
        STM     #d_iir_dn,IIR_DATA_P_H
                                              ; IIR_DATA_P_H points on the
                                              ; first high word of the internal value
        STM     #iir_coff_table,IIR_COFF_P_1
                                              ; IIR_COFF_P_1 points on the
                                              ; first coefficient -B2
        STM     #iir_coff_table,IIR_COFF_P_2
                                              ; IIR_COFF_P_2 points on the
                                              ; first coefficient -B2
        STM     #K_IIR_BFFR,BK                ; Load circ buff size
        RETD
        STM     #K_IIR_INDEX,IIR_INDEX        ; Load circ buff index

;---------------------------------------------------------------
iir_task
        STM     #(d_filin),INBUF_P              ; Load input add mem
        STM     #K_FRAME_SIZE-1,BRC

RPTBD iir_filter_loop_end-1
        STM     #(d_filout),OUTBUF_P            ; Load output address memory
        DLD     *INBUF_P+,A                     ; Load in A the new Q31 sample
        LD      A,-2,A      ; pre scaling
; feedback_path
LD      #0,B
```

```
        MACSU *IIR_DATA_P_L+0%,*IIR_COFF_P_1+,B   ; -b2*d(n-2)low
        MACSU *IIR_DATA_P_L+0%,*IIR_COFF_P_1+,B   ; -b1*d(n-1)low
        ADD     B,-16,A                        ;x(n)+(-b2*d(n-2)low-b1*d(n-1)low)>>16

        MAC     *IIR_COFF_P_2+,*IIR_DATA_P_H+0%,A       ;-b2*d(n-2)high
        MAC     *IIR_COFF_P_2+,*IIR_DATA_P_H+0%,A       ;-b1*d(n-1)high
        STL     A,*IIR_DATA_P_L+0%                ; Store the 32-bit result d(n)
        STH     A,*IIR_DATA_P_H+0%
        ;forward_path
        LD      #0,B
        MACSU *IIR_DATA_P_L+0%,*IIR_COFF_P_1+,B         ;a2*d(n-2)low
        MACSU *IIR_DATA_P_L+0%,*IIR_COFF_P_1+,B         ;a1*d(n-1)low
        MACSU *IIR_DATA_P_L+0%,*IIR_COFF_P_1+0%,B       ,a0*d(n)low
        LD      B,-16,B                      ;a0*d(n)low+a1*d(n-1)low+a2*d(n-2)low>>16

        MAC     *IIR_COFF_P_2+,*IIR_DATA_P_H+0%,B       ;a2*d(n-2)high
        MAC     *IIR_COFF_P_2+,*IIR_DATA_P_H+0%,B       ;a1*d(n-1)high
        MAC     *IIR_COFF_P_2+0%,*IIR_DATA_P_H+0%,B     ;a0*d(n)high

        MAR     *IIR_DATA_P_L+0%                  ; Update d(n-2)=d(n-1)
        MAR     *IIR_DATA_P_H+0%                  ; Update d(n-1)=d(n)
        LD      B,2,B                             ; post scaling
        STH     B,*OUTBUF_P+                      ; Store the low and high
        STL     B,*OUTBUF_P+                      ; part of the Q31 result
iir_filter_loop_end

        RET
        .end
```

Appendix C: Implementation of an 32x16-bit Equalizer on 'C54x

```
; TEXAS INSTRUMENTS FRANCE
; Audio DSP Development

;       ASM Code Module of the eqz32x16.asm

; AUTHOR        CAVALIER Philippe
;               MESSINA  Nathalie
;
; (C) Copyright 1997. Texas Instruments France. All rights reserved
;

*********************************************************
*     macro definitions                                 *
*********************************************************
        .mmregs
        .include      "init54x.inc"

*********************************************************
*     reset/interrupt/trap vectors                      *
*********************************************************
*
* Always start from Reset.
*

        .global Start
        .sect   "vecs"
Start
        BD      Init   ; Branch to MainLine.
        NOP
        NOP

*********************************************************
*      Set up constant and filter coeff                 *
*********************************************************
K_FRAME_SIZE    .set   128                      ; Number of samples
K_NB_FILTER     .set   5                        ; Number of 2nd-order IIR filter
K_BUFFER_INDEX  .set   2                        ; Circ buff index
K_BUFFER_SIZE   .set   2*3*(K_NB_FILTER+1)-2    ; Circ buff size

EqzCoffTable    .sect  "eqz_coff"

        .word   03fd4h      ; A2/2    first IIR filter
        .word   0802dh      ; A1/2
        .word   03fffh      ; A0/2    (0dB/100hz)
        .word   0c02ah      ; -B2/2
        .word   07fd2h      ; -B1/2

        .word   03ef4h      ; A2/2
        .word   081a8h      ; A1/2
        .word   03f80h      ; A0/2    (-9dB/330hz)
        .word   0c189h      ; -B2/2
        .word   07e57h      ; -B1/2

        .word   03d54h      ; A2/2
        .word   082b9h      ; A1/2
        .word   04105h      ; A0/2    (7dB/1khz)
        .word   0c1a5h      ; -B2/2
        .word   07d46h      ; -B1/2
```

```
        .word   039f9h      ; A2/2
        .word   091d5h      ; A1/2
        .word   03ef8h      ; A0/2    (-3dB/3k3hz)
        .word   0c70dh      ; -B2/2
        .word   06e2ah      ; -B1/2

        .word   032c2h      ; A2/2    Last IIR filter
        .word   0d345h      ; A1/2
        .word   03d01h      ; A0/2    (-4dB/10khz)
        .word   0d03bh      ; -B2/2
        .word   02cbah      ; -B1/2


*********************************************************
*        Set up in/out buffer                           *
*********************************************************

DataFilin       .usect "eqz_vars",256             ; for 32 samples
DataFilout      .usect "eqz_vars",256             ; for 32 samples
DataInternal    .usect "eqz_bfr",K_BUFFER_SIZE    ; to store d(n),d(n-1),d(n-2)
DataTempBuff    .usect "eqz_bfr",1

*********************************************************
*       Init section                                    *
*********************************************************
        .text
Init
        LD      #0,A                              ; acc = >00000000.
        STLM    A,SWWSR                           ; 0 Wait States.
        STLM    A,BSCR      ; Bank shift.
        STM     #K_ST0,ST0
        STM     #K_ST1,ST1
        STM     #K_PMST,PMST

*****************************************************************
*                                                               *
*       Main program                                            *
*                                                               *
*****************************************************************

Main
        CALL    EqzInit
Continue
        NOP
        NOP
        CALL    EqzTask
        NOP
        NOP
        B       Continue

*****************************************************************
*                                                               *
*        Sub routines                                           *
*                                                               *
*****************************************************************

        .asg    AR0,EQZ_INDEX                     ; Circ buff index
        .asg    AR1,EQZ_NB                        ; Number of filters
        .asg    AR2,pEQZ_DATA_L                   ; Low word of data buffer
        .asg    AR3,pEQZ_DATA_H                   ; High word of data buffer
```

```
        .asg    AR4,pEQZ_COFF_1                 ; Coefficient pointer 1
        .asg    AR5,pEQZ_COFF_2                 ; Coefficient pointer 2
        .asg    AR6,pINBUF                      ; Input data
        .asg    AR7,pOUTBUF                     ; Output data
        .sect   "eqz_prog"

EqzInit
        STM     #DataInternal,pEQZ_DATA_L
        RPTZ    A,#(K_BUFFER_SIZE-1)            ; Clear the data
        STL     A,*pEQZ_DATA_L+                 ; buffer
        STM     #DataInternal+1,pEQZ_DATA_L
                                                ; pEQZ_DATA_L points on the
                                                ; first low word of the data buffer
        STM     #DataInternal,pEQZ_DATA_H
                                                ; pEQZ_DATA_H points on the
                                                ; first high word of the data buffer
        STM     #K_BUFFER_SIZE,BK               ; Load circ buff size
        STM     #K_BUFFER_INDEX,EQZ_INDEX       ; Load circ buff index
        STM     #EqzCoffTable,pEQZ_COFF_1       ; pEQZ_COFF_1 points on the
                                                ; first coefficient
        RETD
        STM     #EqzCoffTable,pEQZ_COFF_2
                                                ; pEQZ_COFF_2 points also on
                                                ; the first coefficient

;---------------------------------------------------------------

EqzTask
        STM     #(DataFilin),pINBUF             ; Load input add mem
        STM     #(K_FRAME_SIZE-1),BRC           ; Load loop counter

EqzFilterBegin
        RPTBD   EqzFilterEnd-1
        STM     #(DataFilout),pOUTBUF           ; Load output add mem
        STM     #(K_NB_FILTER-2),EQZ_NB
        DLD     *pINBUF+,B
        LD            B,-2,B
        LD      #0,A
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF_1+,A   ;a2/2*x0low(n-2)
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF_1+,A   ;a1/2*x0low(n-1)
        STL     B,*pEQZ_DATA_L-%                  ;storage x0high(n)
        STH     B,*pEQZ_DATA_L+%                  ;storage x0low(n)
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF_1+,A   ;a0/2*x0low(n)
        MPY     *pEQZ_DATA_H+0%,*pEQZ_COFF_2+,B   ;a2/2*x0high(n-2)
        MAC     *pEQZ_DATA_H+0%,*pEQZ_COFF_2+,B   ;a1/2*x0high(n-1)
        MAC     *pEQZ_DATA_H+0%,*pEQZ_COFF_2+,B   ;a0/2*x0high(n)
Loop:
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF_1+,A   ;-b2/2*ylow(n-2)
        MACSU   *pEQZ_DATA_L,*pEQZ_COFF_1+,A      ;-b1/2*ylow(n-1)
        ADD     A,-16,B
        MAC     *pEQZ_DATA_H+0%,*pEQZ_COFF_2+,B   ;-b2/2*yhigh(n-2)
        MAC     *pEQZ_DATA_H,*pEQZ_COFF_2+,B      ;-b1/2*yhigh(n-1)
        MAR     *pEQZ_DATA_L-0%
        MAR     *pEQZ_DATA_H-0%
        MPY     *pEQZ_DATA_H+0%,*pEQZ_COFF_2+,A   ;a2/2*xhigh(n-2)next
        MAC     *pEQZ_DATA_H+0%,*pEQZ_COFF_2+,A   ;a1/2*xhigh(n-1)next
        LD      B,1,B
        DST     B,*pEQZ_DATA_H
        MAC     *pEQZ_DATA_H+0%,*pEQZ_COFF_2+,A   ;a0/2*xhigh(n)next
        LD      A,B
        LD      #0,A
```

```
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF_1+,A   ;a2/2*xlow(n-2)next

        BANZD   Loop,*EQZ_NB-
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF_1+,A   ;a1/2*xlow(n-1)next
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF_1+,A   ;a0/2*xlow(n)next
EndLoop
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF_1+,A   ;-b2/2*ylow(n-2)
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF_1,A    ;-b1/2*ylow(n-2)
        ADD     A,-16,B
        MAC     *pEQZ_DATA_H+0%,*pEQZ_COFF_2+,B   ;-b2/2*yhigh(n-2)
        MAC     *pEQZ_DATA_H+0%,*pEQZ_COFF_2,B    ;-b1/2*yhigh(n-1)
        LD            B,1,B
        STH     B,*pEQZ_DATA_H+0%
        STL     B,*pEQZ_DATA_L+0%
        MAR     *+pEQZ_COFF_1(-24)
        MAR     *+pEQZ_COFF_2(-24)
        DST     B,*pOUTBUF+
EqzFilterEnd

        RET
        .end
```

Appendix D: Implementation of an IIR 32x32-bit Direct Form I on 'C54x

```
; TEXAS INSTRUMENTS FRANCE
;

; AUTHOR        MESSINA  Nathalie
;               CAVALIER Philippe
;

; (C) Copyright 1997. Texas Instruments France. All rights reserved

;-----------------------------------------------------------

*********************************************************
*        macro definitions                              *
*********************************************************
        .mmregs
        .include      "init54x.inc"

*********************************************************
*        reset/interrupt/trap vectors                   *
*********************************************************
*
* Always start from Reset.
*

        .global start
        .sect   "vecs"

start
        BD      init                            ; Branch to MainLine.
        NOP
        NOP

*********************************************************
*       Set up constant and filter coeff                *
*********************************************************

K_FRAME_SIZE    .set   128                        ; Number of samples
K_BUFFER_INDEX  .set   2                          ; Circ buff index
K_BUFFER_SIZE   .set   2*(3*2-1)                  ; Circ buff size
iir_coff_table  .sect  "iir_coff"
                .word  0100Ah                     ;A2 high
                .word  0h                         ;A2 low
                .word  02014h                     ;A1 high
                .word  0h                         ;A1 low
                .word  0100Ah                     ;A0 high
                .word  0h                         ;A0 low
                .word  0DC01h                     ;-B2 high
                .word  0h                         ;-B2 low
                .word  063D6h                     ;-B1 high
                .word  0h                         ;-B1 low


*********************************************************
*                Set up in/out buffer                   *
*********************************************************
d_filin         .usect "iir_vars",256             ; for 32 samples
d_filout        .usect "iir_vars",256             ; for 32 samples
d_internal      .usect "iir_bfr",K_BUFFER_SIZE
```

```
d_temp_buff     .usect "iir_bfr",1


*********************************************************
*       Init section                                    *
*********************************************************
        .text
init
        LD      #0,A                            ; acc = >00000000.
        STLM    A,SWWSR                         ; 0 Wait States.
        STLM    A,BSCR                          ; Bank shift.
        STM     #K_ST0,ST0
        STM     #K_ST1,ST1
        STM     #K_PMST,PMST
*************************************************************
*                                                           *
*       Main program                                        *
*                                                           *
*************************************************************

Main
        CALL    IirFilterInit
Continue
        NOP
        NOP
        CALL    IirFilterTask
        NOP
        NOP
        B       Continue


*************************************************************
*                                                           *
*        Sub routines                                       *
*                                                           *
*************************************************************

        .asg    AR0,IIR_INDEX                   ; Circ buff index
        .asg    AR2,pIIR_DATA_L                 ; Low word of internal value
        .asg    AR3,pIIR_DATA_H                 ; High word of internal value
        .asg    AR4,pIIR_COFF_L                 ; Low Part Coefficient pointer
        .asg    AR5,pIIR_COFF_H                 ; High Part Coefficient pointer
        .asg    AR6,pINBUF                      ; Input data
        .asg    AR7,pOUTBUF                     ; Output data

        .sect   "iir_prog"

IirFilterInit
        STM     #d_internal,pIIR_DATA_L
        RPTZ    A,#K_BUFFER_SIZE-1              ; Clear the internal
        STL     A,*pIIR_DATA_L+                 ; value buffer
        STM     #d_internal+1,pIIR_DATA_L
                                              ; IIR_DATA_P_L points on the first low
                                              ; word of the internal value
        STM     #d_internal,pIIR_DATA_H
                                              ; IIR_DATA_P_H points on the first
                                              ; high word of the internal value
        STM     #iir_coff_table,pIIR_COFF_H
                                              ; IIR_COFF_P_H points on the high part
                                              ; of the first coefficient
        STM     #iir_coff_table+1,pIIR_COFF_L
```

```
                                                ; IIR_COFF_P_L points on the low part
                                                ; of the first coefficient
        STM     #K_BUFFER_SIZE,BK               ; Load circ buff size
        RETD
        STM     #K_BUFFER_INDEX,IIR_INDEX       ; Load circ buff index

;-----------------------------------------------------------------
        .sect   "iir_prog"

IirFilterTask
        STM     #(d_filin),pINBUF                  ; Load input add mem
        STM     #K_FRAME_SIZE-1,BRC

        RPTBD   IirFilterLoopEnd-1
        STM     #(d_filout),pOUTBUF                ; Load output add mem

IirFilterBegin
        DLD     *pINBUF+,A  ; Load in B the new Q31
        LD      #0,B
        MACSU   *pIIR_COFF_L+0%,*pIIR_DATA_H+0%,B ;a2low*x(n-2)high
        MACSU   *pIIR_DATA_L+0%,*pIIR_COFF_H+0%,B ;a2high*x(n-2)low
        MACSU   *pIIR_COFF_L+0%,*pIIR_DATA_H+0%,B ;a1low*x(n-1)high
        MACSU   *pIIR_DATA_L+0%,*pIIR_COFF_H+0%,B ;a1high*x(n-1)low
        DST     A,*pIIR_DATA_H                    ;store x(n)
        MACSU   *pIIR_COFF_L+0%,*pIIR_DATA_H+0%,B ;a0low*x(n)high
        MACSU   *pIIR_DATA_L+0%,*pIIR_COFF_H+0%,B ;a0high*x(n)low
        MACSU   *pIIR_COFF_L+0%,*pIIR_DATA_H+0%,B ;-b2low*y(n-2)high
        MACSU   *pIIR_DATA_L+0%,*pIIR_COFF_H+0%,B ;-b2high*y(n-2)low
        MACSU   *pIIR_COFF_L+0%,*pIIR_DATA_H+0%,B ;-b1low*y(n-1)high
        MACSU   *pIIR_DATA_L+0%,*pIIR_COFF_H+0%,B ;-b1high*y(n-1)low
        MPY     *pIIR_COFF_H+0%,*pIIR_DATA_H+0%,A ;a2high*x(n-2)high
        MAC     *pIIR_COFF_H+0%,*pIIR_DATA_H+0%,A ;a1high*x(n-1)high
        MAC     *pIIR_COFF_H+0%,*pIIR_DATA_H+0%,A ;a0high*x(n)high
        MAC     *pIIR_COFF_H+0%,*pIIR_DATA_H+0%,A ;-b2high*y(n-2)high
        MAC     *pIIR_COFF_H+0%,*pIIR_DATA_H+0%,A ;-b1high*x(n-1)high
        ADD     B,-16,A     ; y(n)
        STH     A,*pIIR_DATA_H+0%
        STL     A,*pIIR_DATA_L+0%
        DST     A,*pOUTBUF+

IirFilterLoopEnd
        RET

        .end
```

Appendix E: Implementation of an 32x32-bit IIR Direct Form II on ‘C54x

```
; TEXAS INSTRUMENTS FRANCE
; Audio DSP Development

;          ASM Code Module of the iir32x32.asm

; AUTHOR        CAVALIER Philippe
;               MESSINA  Nathalie
;
; (C) Copyright 1997. Texas Instruments France. All rights reserved

;
;---------------------------------------------------------------
*********************************************************
*        macro definitions                              *
*********************************************************
        .mmregs
        .include      "init54x.inc"

*********************************************************
*        reset/interrupt/trap vectors                   *
*********************************************************
*
* Always start from Reset.
*

        .global start


        .sect   "vecs"
start
        BD      init                            ; Branch to MainLine.
        NOP
        NOP



*********************************************************
*        Set up constant and filter coeff               *
*********************************************************
K_FRAME_SIZE    .set   128                      ; Number of samples
K_IIR_INDEX     .set   2                        ; Circ buff index
K_IIR_BFFR      .set   6                        ; Circ buff size
iir_coff_table  .sect  "iir_coff"
                .word  00000h                   ;-B2 low
                .word  0DC01h                   ;-B2 high
                .word  00000h                   ;-B1 low
                .word  063D6h                   ;-B1 high
                .word  00000h                   ;A2 low
                .word  0100Ah                   ;A2 high
                .word  00000h                   ;A1 low
                .word  02014h                   ;A1 high
                .word  00000h                   ;A0 low
                .word  0100Ah                   ;A0 high


*********************************************************
*           Set up in/out buffer                        *
*********************************************************
```

```
d_filin         .usect "iir_vars",256             ; for 128 samples
d_filout        .usect "iir_vars",256             ; for 128 samples
d_iir_dn        .usect "iir_bfr",6                ; to store d(n),d(n-1),d(n-2)
d_iir_y         .usect "iir_bfr",1
```

```
*************************************************************
*        Init section                                       *
*************************************************************
        .text
init
        LD      #0,A                            ; acc = >00000000.
        STLM    A,SWWSR                         ; 0 Wait States.
        STLM    A,BSCR                          ; Bank shift.
        STM     #K_ST0,ST0
        STM     #K_ST1,ST1
        STM     #K_PMST,PMST
*****************************************************************
*                                                               *
*        Main program                                           *
*                                                               *
*****************************************************************

Main
        CALL    iir_init
Continue
        NOP
        NOP
        CALL    iir_task
        NOP
        NOP
        B       Continue

*****************************************************************
*                                                               *
*        Sub routines                                           *
*                                                               *
*****************************************************************

        .asg    AR0,pIIR_INDEX                  ; Circ buff index
        .asg    AR2,pIIR_DATA_L                 ; Low word of internal value
        .asg    AR3,pIIR_DATA_H                 ; High word of internal value
        .asg    AR4,pIIR_COFF                   ; Coefficient pointer
        .asg    AR6,pINBUF                      ; Input data
        .asg    AR7,pOUTBUF                     ; Output data
        .sect   "iir_prog"

iir_init
        STM     #d_iir_dn,pIIR_DATA_L
        RPTZ    A,#K_IIR_BFFR-1                 ; Clear the internal
        STL     A,*pIIR_DATA_L+                 ; value buffer
        STM     #d_iir_dn+1,pIIR_DATA_L
                                                ; pIIR_DATA_L points on the first low
                                                ; word of the internal value
        STM     #d_iir_dn,pIIR_DATA_H
                                                ; pIIR_DATA_H points on the first high
                                                ; word of the internal value
        STM     #iir_coff_table,pIIR_COFF
                                                ; pIIR_COFF points on the low word of
                                                ; the first coefficient A2
        STM     #K_IIR_BFFR,BK                  ; Load circ buff size
        RETD
        STM     #K_IIR_INDEX,pIIR_INDEX         ; Load circ buff index

;---------------------------------------------------------------
        .sect   "iir_prog"
```

```
iir_task
        STM     #(d_filin),pINBUF               ; Load input add mem
        STM     #K_FRAME_SIZE-1,BRC
        RPTBD   iir_filter_loop_end-1
        STM     #(d_filout),pOUTBUF             ; Load output add mem

        DLD     *pINBUF+,A                      ; Load in A the new Q31
        LD      A,-2,A                          ; sample that will be treated
iir_filter

        ;feedback_path
        LD      #0,B
        MACSU   *pIIR_COFF+,*pIIR_DATA_H,B      ; B=b2Low*d(n-2)High
        MACSU   *pIIR_DATA_L+0%,*pIIR_COFF,B    ; B=B+b2High*d(n-2)Low
        MAC     *pIIR_COFF+,*pIIR_DATA_H+0%,A   ; A=A+b2High*d(n-2)High
        MACSU   *pIIR_COFF+,*pIIR_DATA_H,B      ; B=b1Low*d(n-1)High
        MACSU   *pIIR_DATA_L+0%,*pIIR_COFF,B    ; B=B+b1High*d(n-1)Low
        MAC     *pIIR_COFF+,*pIIR_DATA_H+0%,A   ; A=A+b1High*d(n-1)High
        ADD     B,-16,A                         ; 16 Right shift \ add
        STL     A,*pIIR_DATA_L+0%               ; Store d(n) the result
        STH     A,*pIIR_DATA_H+0%               ; of the add and mult.
        ; forward_path
        LD      #0,B
        MACSU   *pIIR_COFF+,*pIIR_DATA_H,B      ; B=a2Low*d(n-2)High
        MACSU   *pIIR_DATA_L+0%,*pIIR_COFF,B    ; B=B+a2High*d(n-)Low
        MPY     *pIIR_COFF+,*pIIR_DATA_H+0%,A   ; A=A+a2High*d(n-2)High
        MACSU   *pIIR_COFF+,*pIIR_DATA_H,B      ; B=a1Low*d(n-1)High
        MACSU   *pIIR_DATA_L+0%,*pIIR_COFF,B    ; B=a1Low*d(n-1)High
        MAC     *pIIR_COFF+,*pIIR_DATA_H+0%,A   ; A=A+a1High*d(n-1)High
        MACSU   *pIIR_COFF+,*pIIR_DATA_H,B      ; B=a0Low*d(n)High
        MACSU   *pIIR_DATA_L+0%,*pIIR_COFF,B    ; B=a0Low*d(n)High
        MAC     *pIIR_COFF,*pIIR_DATA_H+0%,A    ; A=A+a0High*d(n)High
        ADD     B,-16,A
        MAR     *pIIR_DATA_L+0%                 ; Update the circular
        MAR     *pIIR_DATA_H+0%                 ; buffer d(n-2)=d(n-1)
                                                ; and d(n-1)=d(n)
        MAR     *+pIIR_COFF(-9)
        LD      A,2,A
        STH     A,*pOUTBUF+                     ; Store the low and high
        STL     A,*pOUTBUF+                     ; part of the Q31 result
iir_filter_loop_end

        RET
        .end
```

Appendix F: Implementation of an 32x32-bit Equalizer on 'C54x

```
; TEXAS INSTRUMENTS FRANCE
;

; AUTHOR        CAVALIER Philippe
;               MESSINA  Nathalie
;
; (C) Copyright 1997. Texas Instruments France. All rights reserved

*********************************************************
*                     macro definitions                 *
*********************************************************
        .mmregs
        .include  "init54x.inc"

*********************************************************
*                     reset/interrupt/trap vector       *
*********************************************************
*
* Always start from Reset.
*

        .global Start


        .sect   "vecs"
Start
        BD      Init                            ; Branch to MainLine.
        NOP
        NOP


*********************************************************
*                 Set up constant and filter coeff      *
*********************************************************
K_FRAME_SIZE    .set   128                      ; Number of samples
K_NB_FILTER     .set   5                        ; Number of 2nd-order IIR filter
K_BUFFER_INDEX  .set   2                        ; Circ buff index
K_BUFFER_SIZE   .set   2*3*(K_NB_FILTER+1)-2    ; Circ buff size

EqzCoffTable    .sect  "eqz_coff"

        .word   0000h                           ;A2/2 low
        .word   03fd4h                          ;A2/2 high
        .word   0000h                           ;A1/2 low
        .word   0802dh                          ;A1/2 high
        .word   0000h                           ;A0/2 low
        .word   03fffh                          ;A0/2 high
        .word   0000h                           ;-B2/2 low
        .word   0c02ah                          ;-B2/2 high
        .word   0000h                           ;-B1/2 low
        .word   07fd2h                          ;-B1/2 high

        .word   0000h                           ;A2/2 low
        .word   03ef4h                          ;A2/2 high
        .word   0000h                           ;A1/2 low
        .word   081a8h                          ;A1/2 high
        .word   0000h                           ;A0/2 low
        .word   03f80h                          ;A0/2 high
```

```
        .word   0000h                           ;-B2/2 low
        .word   0c189h                          ;-B2/2 high
        .word   0000h                           ;-B1/2 low
        .word   07e57h                          ;-B1/2 high

        .word   0000h                           ;A2/2 low
        .word   03d54h                          ;A2/2 high
        .word   0000h                           ;A1/2 low
        .word   082b9h                          ;A1/2 high
        .word   0000h                           ;A0/2 low
        .word   04105h                          ;A0/2 high
        .word   0000h                           ;-B2/2 low
        .word   0c1a5h                          ;-B2/2 high
        .word   0000h                           ;-B1/2 low
        .word   07d46h                          ;-B1/2 high

        .word   0000h                           ;A2/2 low
        .word   039f9h                          ;A2/2 high
        .word   0000h                           ;A1/2 low
        .word   091d5h                          ;A1/2 high
        .word   0000h                           ;A0/2 low
        .word   03ef8h                          ;A0/2 high
        .word   0000h                           ;-B2/2 low
        .word   0c70dh                          ;-B2/2 high
        .word   0000h                           ;-B1/2 low
        .word   06e2ah                          ;-B1/2 high

        .word   0000h                           ;A2/2 low
        .word   032c2h                          ;A2/2 high
        .word   0000h                           ;A1/2 low
        .word   0d345h                          ;A1/2 high
        .word   0000h                           ;A0/2 low
        .word   03d01h                          ;A0/2 high
        .word   0000h                           ;-B2/2 low
        .word   0d03bh                          ;-B2/2 high
        .word   0000h                           ;-B1/2 low
        .word   02cbah                          ;-B1/2 high


*********************************************************
*                Set up in/out buffer                   *
*********************************************************

DataFilin       .usect "eqz_vars",256           ; for 32 samples
DataFilout      .usect "eqz_vars",256           ; for 32 samples
DataInternal    .usect "eqz_bfr",K_BUFFER_SIZE
DataTempBuff    .usect "eqz_bfr",1


*********************************************************
*        Init section                                   *
*********************************************************
        .text

Init
        LD      #0,A                            ; acc = >00000000.
        STLM    A,SWWSR                         ; 0 Wait States.
        STLM    A,BSCR                          ; Bank shift.
        STM     #K_ST0,ST0
        STM     #K_ST1,ST1
```

```
        STM     #K_PMST,PMST

*****************************************************************
*                                                               *
*       Main program                                            *
*                                                               *
*****************************************************************

Main
        CALL    EqzInit
Continue
        NOP
        NOP
        CALL    EqzTask
        NOP
        NOP
        B       Continue
```

```
*****************************************************************
*                                                               *
*       Sub routines                                            *
*                                                               *
*****************************************************************

        .asg    AR0,EQZ_INDEX                   ; Circular buffer index
        .asg    AR1,EQZ_NB
        .asg    AR2,pEQZ_DATA_L                 ; Low word of internal value
        .asg    AR3,pEQZ_DATA_H                 ; High word of internal value
        .asg    AR4,pEQZ_COFF                   ; Coefficients pointer
        .asg    AR5,pTEMP_BUFF
        .asg    AR6,pINBUF                      ; Input data
        .asg    AR7,pOUTBUF                     ; Output data
        .sect   "eqz_prog"

EqzInit
        STM     #DataInternal,pEQZ_DATA_L
        RPTZ    A,#K_BUFFER_SIZE-1              ; Clear the internal
        STL     A,*pEQZ_DATA_L+                 ; value buffer
        STM     #(DataTempBuff),pTEMP_BUFF
        STM     #DataInternal+1,pEQZ_DATA_L
                                                ; pEQZ_DATA_L points on the
                                        ; first low word of the internal value
        STM     #DataInternal,pEQZ_DATA_H
                                                ; pEQZ_DATA_H points on the
                                        ; first high word of the internal value
        STM     #EqzCoffTable,pEQZ_COFF
                                                ; pEQZ_COFF points on the first
                                                ; coefficient
        STM     #K_BUFFER_SIZE,BK               ; Load circ buff size
        RETD
        STM     #K_BUFFER_INDEX,EQZ_INDEX       ; Load circ buff index

;---------------------------------------------------------------
        .sect   "eqz_prog"
        .bss    InputHigh,1
        .bss    InputLow,1

EqzTask
        STM     #(DataFilin),pINBUF             ; Load input add mem
        STM     #(K_FRAME_SIZE-1),BRC           ; Load loop counter

EqzFilterBegin
        RPTBD   EqzFilterEnd-1
        STM     #(DataFilout),pOUTBUF           ; Load output add mem

        STM     #(K_NB_FILTER-2),EQZ_NB

        DLD     *pINBUF+,B
        LD      B,-2,B
        DST     B,*pTEMP_BUFF
        LD      #0,B
        MACSU   *pEQZ_COFF+,*pEQZ_DATA_H,B      ;a2/2low*x0(n-2)high
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF,B    ;a2/2high*x0(n-2)low
        MPY     *pEQZ_COFF+,*pEQZ_DATA_H+0%,A   ;a2/2high*x0(n-2)high
        MACSU   *pEQZ_COFF+,*pEQZ_DATA_H,B      ;a1/2low*x1(n-1)high
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF,B    ;a1/2high*x1(n-1)low
        MAC     *pEQZ_COFF+,*pEQZ_DATA_H+0%,A   ;a1/2high*x1(n-1)high
        MVDD    *pTEMP_BUFF+,*pEQZ_DATA_H       ;x(n)high storage
        MVDD    *pTEMP_BUFF-,*pEQZ_DATA_L       ;x(n)low storage
```

```
        MACSU   *pEQZ_COFF+,*pEQZ_DATA_H,B        ;a0/2low*x(n)high
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF,B      ;a0/2high*x(n)low
        MAC     *pEQZ_COFF+,*pEQZ_DATA_H+0%,A     ;a0/2high*x(n)high

Loop:
        MACSU   *pEQZ_COFF+,*pEQZ_DATA_H,B        ;b2/2low*y(n-2)high
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF,B      ;b2/2high*y(n-2)low
        MAC     *pEQZ_COFF+,*pEQZ_DATA_H+0%,A     ;b2 /2high*y(n-2)high
        MACSU   *pEQZ_COFF+,*pEQZ_DATA_H,B        ;b1/2low*y(n-1)high
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF,B      ;b1/2high*y(n-1)high
        MAC     *pEQZ_COFF+,*pEQZ_DATA_H+0%,A     ;b1/2high*y(n-1)high
        ADD     B,-16,A
        LD      A,1,A
        STH     A,*pEQZ_DATA_H-0%
        STL     A,*pEQZ_DATA_L-0%
        LD      #0,B
        MAR     *pEQZ_DATA_H-0%
        MAR     *pEQZ_DATA_L-0%
        MACSU   *pEQZ_COFF+,*pEQZ_DATA_H,B        ;a2/2low*x(n-2)high next
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF,B      ;a2/2high*x(n-2)low next
        MPY     *pEQZ_COFF+,*pEQZ_DATA_H+0%,A     ;a2/2high*x(n-2)high next
        MACSU   *pEQZ_COFF+,*pEQZ_DATA_H,B        ;a1/2low*x(n-1)high next
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF,B      ;a1/2high*x(n-1)low next
        MAC     *pEQZ_COFF+,*pEQZ_DATA_H+0%,A     ;a1/2high*x(n-1)high next
        MACSU   *pEQZ_COFF+,*pEQZ_DATA_H,B        ;a0/2low*x(n)high next
        BANZD   Loop,*EQZ_NB-
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF,B      ;a0/2high*x(n)low next
        MAC     *pEQZ_COFF+,*pEQZ_DATA_H+0%,A     ;a0/2high*x(n)high next
EndLoop
        MACSU   *pEQZ_COFF+,*pEQZ_DATA_H,B        ;b2/2low*y(n-2)high next
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF,B      ;b2/2high*y(n-2)low next
        MAC     *pEQZ_COFF+,*pEQZ_DATA_H+0%,A     ;b2/2high*y(n-2)high next
        MACSU   *pEQZ_COFF+,*pEQZ_DATA_H,B        ;b1/2low*y(n-1)high next
        MACSU   *pEQZ_DATA_L+0%,*pEQZ_COFF,B      ;b1/2high*y(n-1)low next
        MAC     *pEQZ_COFF+,*pEQZ_DATA_H+0%,A     ;b1/2high*y(n-1)high next
        ADD     B,-16,A
        LD      A,1,A
        STH     A,*pEQZ_DATA_H+0%
        STL     A,*pEQZ_DATA_L+0%
        MAR     *+pEQZ_COFF(-24)
        DST     A,*pOUTBUF+
EqzFilterEnd
        RET
        .end
```

Appendix G: Implementation of the 16x32-bit Direct Form I on 'C54x

```
; TEXAS INSTRUMENTS FRANCE
;
; AUTHOR        MESSINA  Nathalie
;               CAVALIER Philippe
;
; (C) Copyright 1997. Texas Instruments France. All rights reserved
;
;
*********************************************************
*                  macro definitions                    *
*********************************************************
        .mmregs
        .include      "init54x.inc"

*********************************************************
*             reset/interrupt/trap vectors              *
*********************************************************
*
* Always start from Reset.
*

        .global Start


        .sect   "vecs"
Start
        BD      Init                            ; Branch to MainLine.
        NOP
        NOP

*********************************************************
*       Set up constant and filter coeff                *
*********************************************************
K_FRAME_SIZE    .set   128                        ; Number of samples
K_NB_FILTER     .set   1                          ; Number of 2nd-order IIR filter
K_BUFFER_INDEX  .set   1                          ; Circ buff index
K_BUFFER_SIZE   .set   (3*(K_NB_FILTER+1)-1)      ; Circ buff size

BPFilterCoffTable     .sect"iir_coff"
        .word   0100Ah                            ; A2 high
        .word   00000h                            ; A2 low
        .word   02014h                            ; A1 high
        .word   00000h                            ; A1 low
        .word   0100Ah                            ; A0 high
        .word   00000h                            ; A0 low
        .word   0DC01h                            ; -B2 high
        .word   00000h                            ; -B2 low
        .word   063D6h                            ; -B1 high
        .word   00000h                            ; -B1 low


*********************************************************
*        Set up in/out buffer                           *
*********************************************************
DataFilin       .usect "iir_vars",128             ; for 128 samples
DataFilout      .usect "iir_vars",128             ; for 128 samples
DataInternal    .usect "iir_bfr",K_BUFFER_SIZE
DataTempBuff    .usect "iir_bfr",1
```

```
*********************************************************
*        Init section                                   *
*********************************************************
        .text
Init
        LD      #0,A                            ; acc = >00000000.
        STLM    A,SWWSR                         ; 0 Wait States.
        STLM    A,BSCR                          ; Bank shift.
        STM     #K_ST0,ST0
        STM     #K_ST1,ST1
        STM     #K_PMST,PMST
*********************************************************
*       *
*       Main program   *
*       *
*********************************************************

Main
        CALL    IirFilterInit
Continue
        NOP
        NOP
        CALL    IirFilterTask
        NOP
        NOP
        B       Continue
*********************************************************
*                                                       *
*       Sub routines                                    *
*                                                       *
*********************************************************

        .asg    AR0,IIR_INDEX                   ; Circ buff index
        .asg    AR2,pIIR_DATA                   ; High word of data buffer
        .asg    AR3,pIIR_COEFF                  ; Coefficient pointer 1
        .asg    AR4,pINBUF                      ; Input data
        .asg    AR5,pOUTBUF                     ; Output data

        .sect   "iir_prog"

IirFilterInit
        STM     #DataInternal,pIIR_DATA
        RPTZ    A,#(K_BUFFER_SIZE-1)            ; Clear the internaldata
        STL     A,*pIIR_DATA+                   ; data buffer
        STM     #DataInternal,pIIR_DATA
                                                ; pIIR_DATA points on the first high
                                                ; word of the data buffer
        STM     #BPFilterCoffTable,pIIR_COEFF
        STM     #K_BUFFER_SIZE,BK               ; Load circ buff size
        RETD
        STM     #K_BUFFER_INDEX,IIR_INDEX       ; Load circ buff index


;---------------------------------------------------------------------

IirFilterTask
        STM     #(DataFilin),pINBUF             ; Load input add mem

IirFilterBegin

        STM     #(K_FRAME_SIZE-1),BRC
```

```
        RPTBD   IirFilterLoopEnd-1
        STM     #(DataFilout),pOUTBUF           ; Load output add mem

        LD      #0,A
        MPY     *pIIR_COEFF+,*pIIR_DATA,B       ; A2high*x(n-2)
        MACSU   *pIIR_COEFF+,*pIIR_DATA+0%,A    ; A2low*x(n-2)
        MAC     *pIIR_COEFF+,*pIIR_DATA,B       ; A1high*x(n-1)
        MACSU   *pIIR_COEFF+,*pIIR_DATA+0%,A    ; A1low*x(n-1)
        MVDD    *pINBUF+,*pIIR_DATA             ; load x(n) in the buffer
        MAC     *pIIR_COEFF+,*pIIR_DATA,B       ; A0high*x(n)
        MACSU   *pIIR_COEFF+,*pIIR_DATA+0%,A    ; A0low*x(n)
        MAC     *pIIR_COEFF+,*pIIR_DATA,B       ; -B2high*y(n-2)
        MACSU   *pIIR_COEFF+,*pIIR_DATA+0%,A    ; -B2low*y(n-2)
        MAC     *pIIR_COEFF+,*pIIR_DATA,B       ; -B1high*y(n-1)
        MACSU   *pIIR_COEFF,*pIIR_DATA+0%,A     ; -B1low*y(n-1)
        ADD     A,-16,B
        MAR     *+pIIR_COEFF(-9)                ; pIIR_COEFF points on A2high
        STH     B,*pIIR_DATA+0%                 ; Store y(n) in the buffer
        STH     B,*pOUTBUF+                     ; Store y(n) in the output
IirFilterLoopEnd
        RET
        .end
```