Audio BIQAUD SP/DP

optimized for ARM-Cortex M3/M4

Table of contents

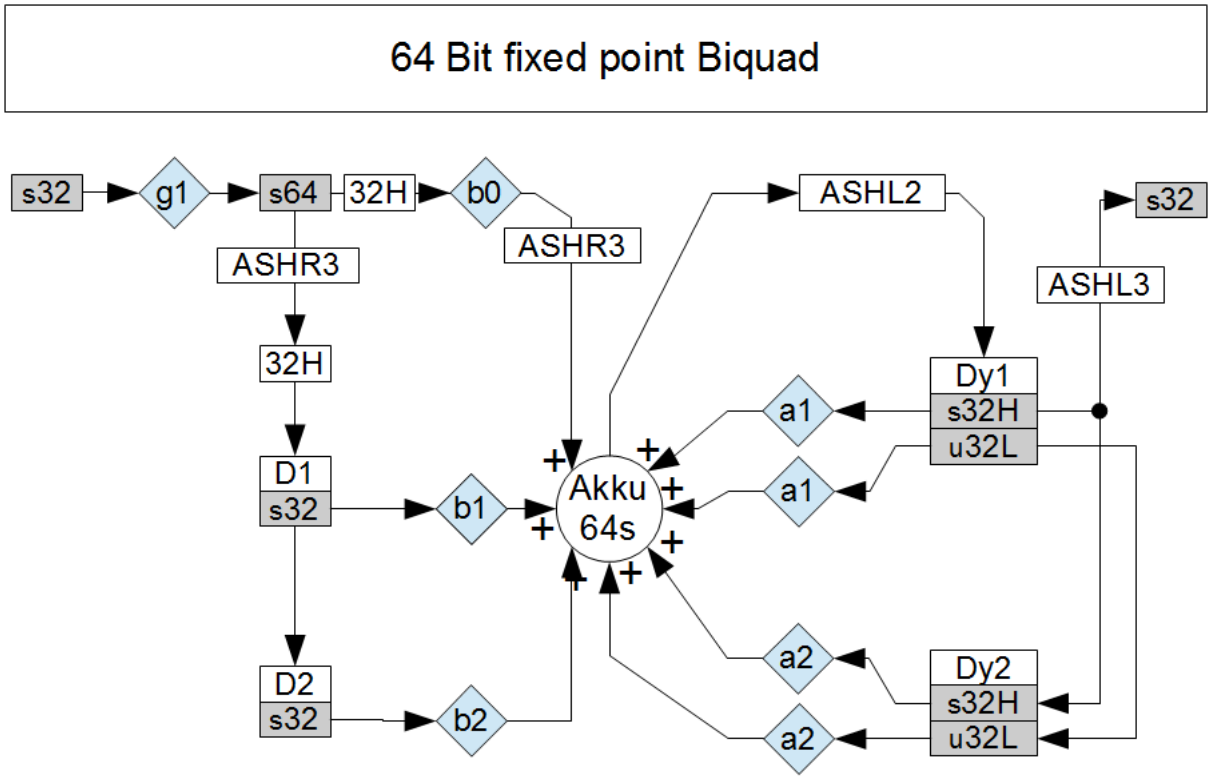

fig 1: general DP/SP Biquad sheme

Annottions:

- b0,b1,b2,a1,a2,g1 are 32Bit signed INTs
- b0,b1,b2,a1,a2 are previously divided by 2 to be able to use coefficients up to 1.9999
- a1,a2 are inverted
- AKKU and AKKU2 are 64Bit signed, multiplication is s32xs32 in s64 or u32xs32 in s64

Calculation steps

(1) g1 is used as pregain (see chapter Filter ordering), for no gain, it should be set to 0.9999999999.

(2) The input value is shifted by 3 bits to the right (div 8) and the output signal is shifted by 3 bits to the left (*8).
This is done to prevent internal Overflows in the filter, even if the input frequency doesn't match the filter.
Because of doing it in the 64Bit domain, no critical precision loss occur.

(3) The coefficients b0,b1,b2 are signed 32Bit and multiplied by the signed 32Bit input/state values (IN,D1,D2) resulting in a 64Bit value wich is accumulated in the signed 64Bit AKKU.
The states D1 and D2 are signed 32Bit, wich is sufficient (see further).

(4) The coefficients a1,a2 are signed 32Bit and multiplied by the signed 64Bit state values (Dy1,Dy2) in Double Precision Mode resulting in a 64Bit value wich is accumulated in the signed 64Bit AKKU.
The states Dy1 and Dy2 are signed 64Bit, wich is sufficient (see further).

(5) The AKKU's output is shifted by 2 bits to the left to equal out the "lost" bit by signed multiplication and the coefficients division by 2.

(6) The by 2 bits shifted AKKU output is further shifted by 3 bits to left to redo the initial shiftdown (see (1))

Filter precision and noise shaping

Normally, the Single precision 32x32 to 64 multiplication is used, but in this type of filter, single precision would loose more information than necessary.

Let's go back to the beginning of the digital audio era, where only a 14Bit DAC was available but 16 Bits precision should be available at the output. To get this work, oversampling of data was introduced, but with oversampling alone, the noise floor rised and the datapath of the 14Bit DAC was not wider than 14Bit, so with running 28Bit datawith , 14Bits would go to the DAC and another 14Bit (the lower ones) were thrown away.....
So, noise shaping was added to keep errors (and noise) down.
Noise shaping, in the simplest form, is to use the lower dataword of a multiplication result (e.g. 14X14 in 14H+14L), store it for one sample and then put it back into the accumulator. This will give good results for low input frequencies and doesn't make much at higher frequencies. The "noise" is shifted all over the spectrum available and therefore get's lower.
Using some mathematics will show, that the best noise shaping occurs, if the error feedback (the lower dataword) is multiplied by a1 for one sample delay and multiplied by a2 for two samples delay.... If you have a look on the diagramm, you will see, that the double precision arithmetic is not more than the best possible noise shaping that can be done :-)

Simulation showed, that the precision of the given architecture is almost the same (< 1 LSB 32BIT) as for a full double precision system.
Even for low filter frequencies (< 50 Hz) , low input frequencies and low signal amplitude, the error compared to double-float will be lower than 12 Bit of 32 Bit, so 20 or more Bits are the same as calculations done in double float while the filters where at extreme settings, like fs=96kHz, Peakfilter f0=35Hz, q=12, g=12. When the filters frequency is increased or the filters gain reduced, the error falls rapidly below the 32BIT LSB mark.

Internal overflow

To prevent internal overflow, the incoming data is shifted by 3 bits to the right in the 64-Bit domain and after the calulations the data is shifted by 3 bits to the left. The loss of precision is neglible, because of the total precision is downgraded from 63-Bit+sign to 60-Bit+sign and an output with of 32 bits .....

Filter ordering

Put the filters with the most positive gain in the end of the chain, the ones with negative gain in front of it and the ones with little or no gain into the beginning. The coefficient g1 can be used to reduce the gain of a stage (e.g. Peakfilter with positive gain). The golden rule of thumb is to keep the input values of a filter as high as possible to don't loss precision.

Biquad chain overflow

As seen in the previous chapter, internal overflow is prevented.
The only thing wich must be observed now is,that the highest (!) gain over frequency must be known. The input of a particular stage must be decreased according that.....

e.g. PeakFilter #1 with f1=100Hz, g1=+12dB and another #2 with f2=5kHz and g2=+3dB
(1) The order must be filter #2, then filter #1
(2) The filter #2 must have a pregain (g1) of -3dB, the filter #1 must have a pregain (g1) of -9dB. In total, no overflow can occur.

e.g. PeakFilter #1 with f1=100Hz, g1=+6dB, #2 with f2=5kHz and g2=-3dB and #3 with f3=15kHz and g3=+5dB
(1) The order must be: filter #2, then filter #3, then filter #1
(2) The filter #2 must have a pregain (g1) of 0dB, the filter #3 of -5dB and filter #1 of -7dB. In total, no overflow can occur.

Keep in mind to let a minimum headroom of ~ 3dB (=0.707) for rounding/frequency errors introduced by the coefficents.

Implementation on Cortex-M3/M4

As it can be seen, the most processor load is multiply s32 x s32 / accumulate and get/store the states D1,D2,Dy1,Dy2.

The cortex M3 can multiply s32 x s32 and accumulate in 5-7 cylces, the M4 in 2 cycles.

The problematic multiplication u32 x s32 must be done as s32 x s32, because no special instruction is available. It's done in total 22 cycles for the M3 and 6 cycles on M4 (for both multiplications, taking shift into account). See fig. 2 for details.

The lower word of Dy1/Dy2 is first (logic !) shift 1 bit to right to get rid of the "sign" bit wich isn't a sign but data. Then, the signed 32*32 multiplication is done and the previous shift is compensated by the (arithmetic !) shift to the left by 1 bit. The compensation for the "lost" bit and the coeff./2 is compensated when generating the final result.

In simulation, the worst error in reference to 64Bit toal Double Precision was lower than 1/1000 of a 32 Bit LSB or -0.00000000000038 = 3.8 E-13 (one 32 Bit LSB has the value of 4.6 E-10) !

Getting/storing data will need the same amount of cycles for both platforms: 1+N , where N is the number of 32 Bit elements to handle.
Both platforms also have 11 generic registers (and 3 special ones).

Putting all informations together,
- cycles are saved when getting/storing a multiple number of words in one run.
- only one pointer for states and coefficients should be used, the other registers can then be used to store local states or coefficients.
- the load/store instructions can handle (simultanously !) pre/post increment and (!) adress shifts (n bytes) wich is very usefull when using pointers.
- C-programs should not use the generic 64Bit datatype but two 32Bit reigisters. This will save unneccessary shiftings

See the instruction set for more information on this topic.

If only one pointer for data, states and coefficients is used, the more complexity by filling in the coefficients and pre-clearing the states is neglible in contrast to the saved cylces.

64Bit DP = 64Bit DoublePrecision for all multiplications

64Bit SPDP = 64Bit SinglePrecision for multiplications with b0,b1,b2 and
64Bit DoublePrecision for multiplications with a1,a2

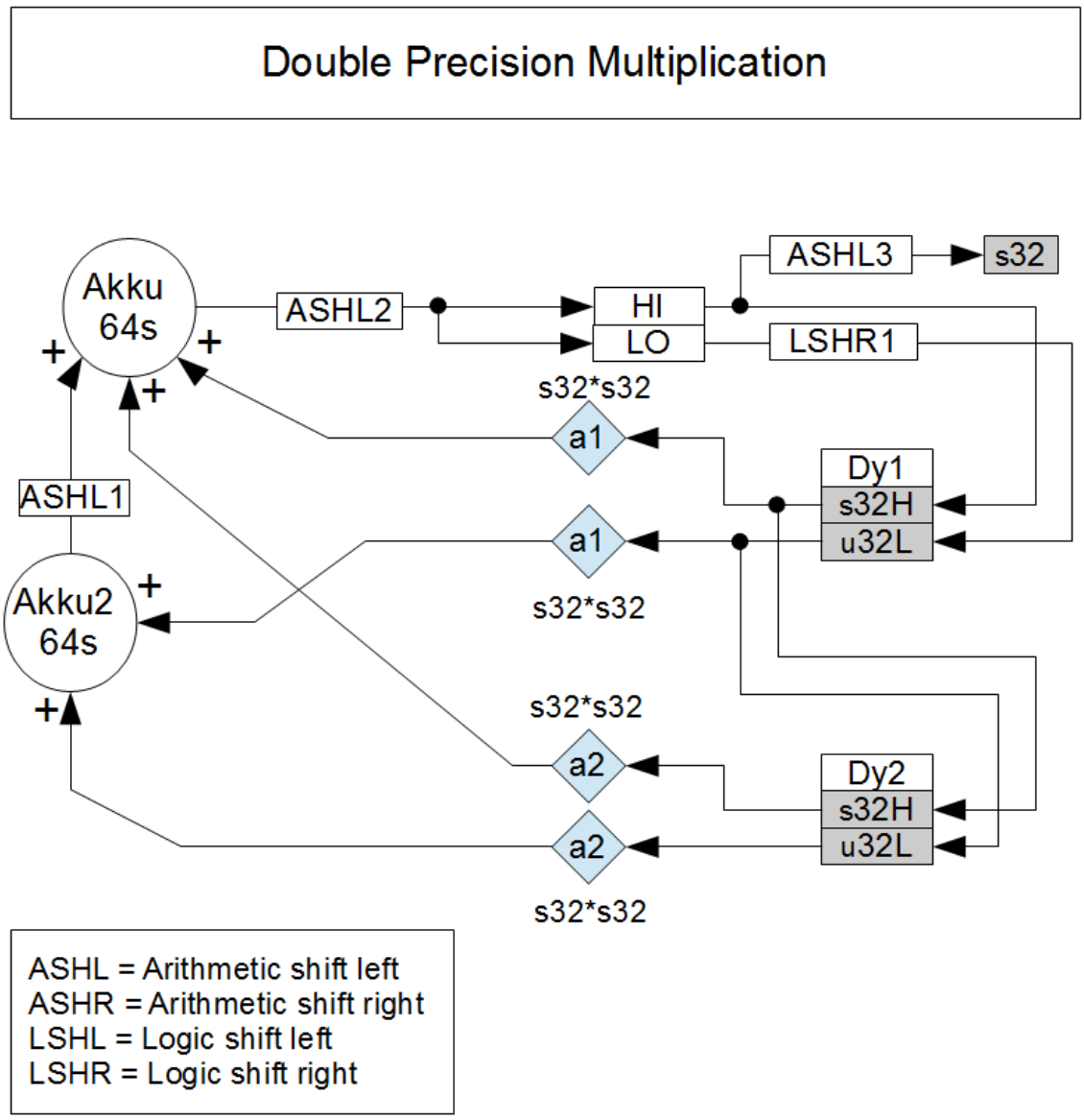

fig 2: Double Precision multiplication

Error Analysis

As mentioned previously, at low frequencies of the filter and the input data, filters behave worse than with higher frequencies.
In the following diagrams, the numeric precision of different dataformats for 3 different filters are presented. The input data has always 24Bit and all values are referenced to the double precision floating point result, wich is not shown here. The values represent the maximum deviation from double precision floating point for 96.000 samples and a samplerate of 96kHz.

The rms value of the deviation will be lower, but the maximum deviation shows much about the problem of choosing the right data format.

The results for 64Bit SPDP and 64Bit DP are almost equal, so both lines are the same.

INT32	is 32x32 --> 64 SinglePrecision
INT32 ErrFeedback	is 32x32 --> 64 SinglePrecision and putting the residium back
INT64 SPDP	is 32x32 --> 64 SinglePrecision for b0,b1,b2 and DoublePrec. for a1,a2
INT64 DP	is 32x32 --> 64 DoublePrecision for all coefficients
float	is 32 Bit floating point with 24Bit Mantissa and 8 Bits Exponent

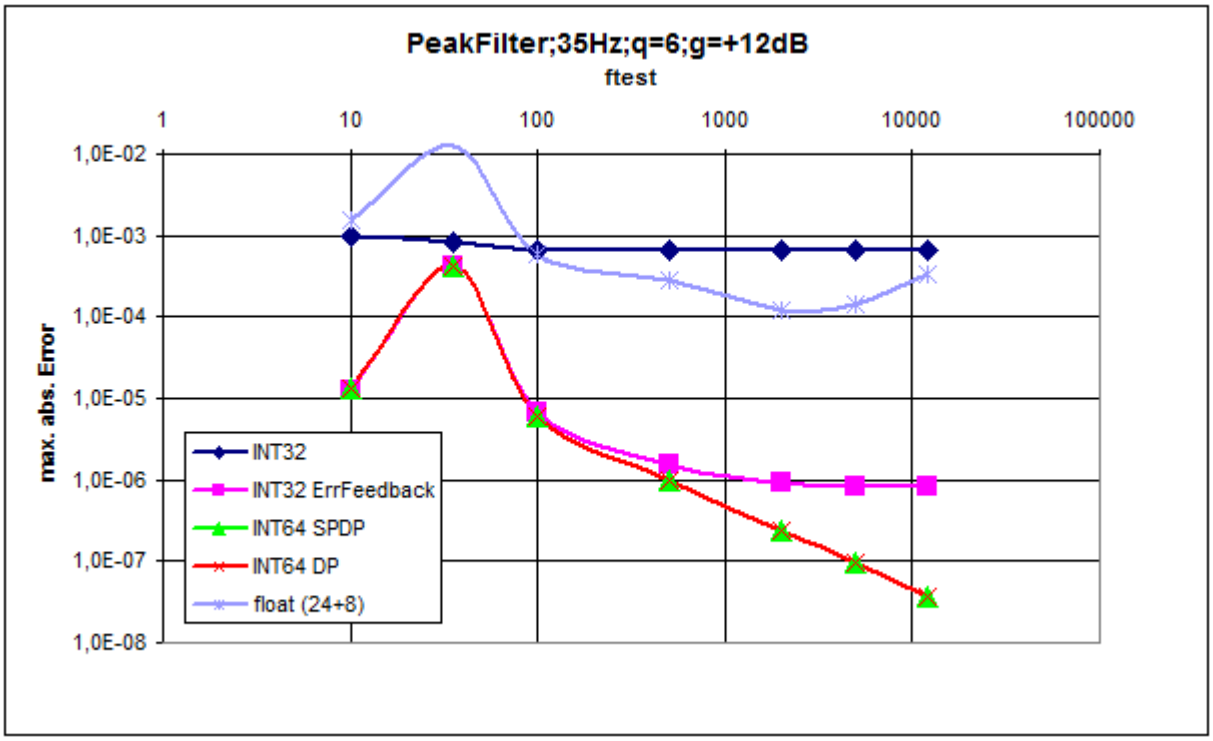

Fig. 3: PEAK PEQ Filter, 35Hz, q=6; gain=+12dB for different precisions

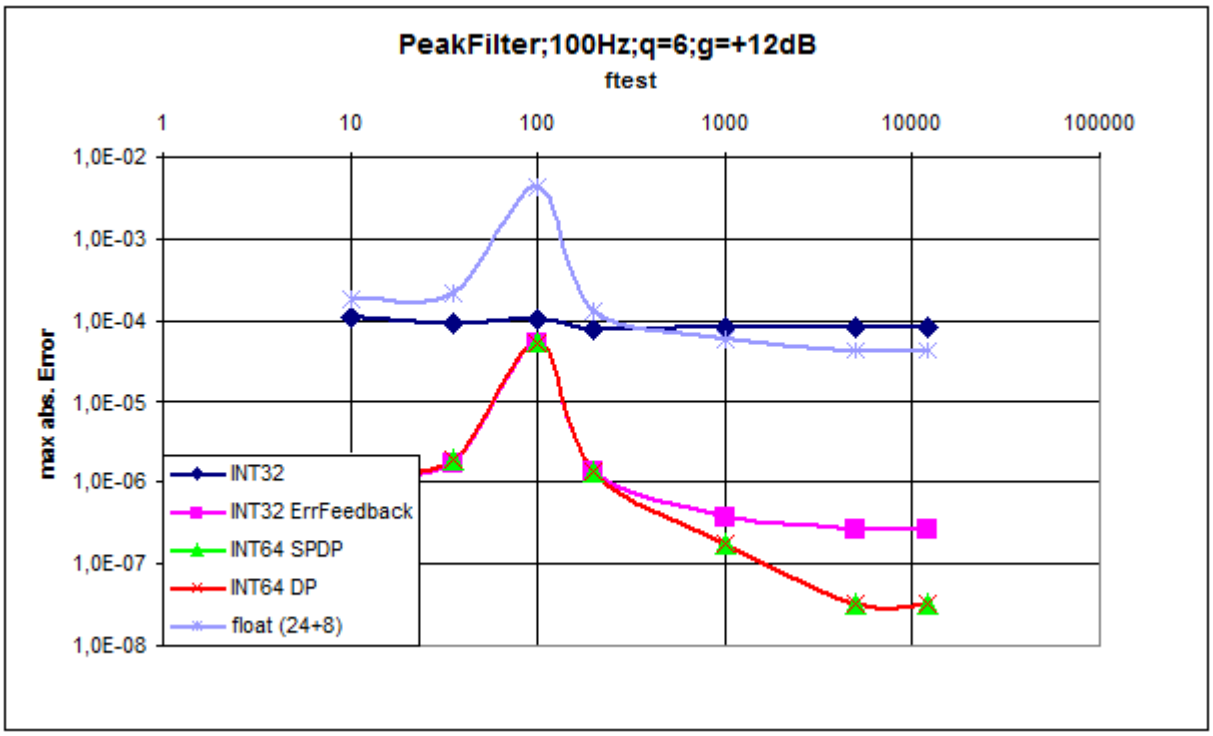

Fig. 4: PEAK PEQ Filter, 100 Hz, q=6; gain=+12dB for different precisions

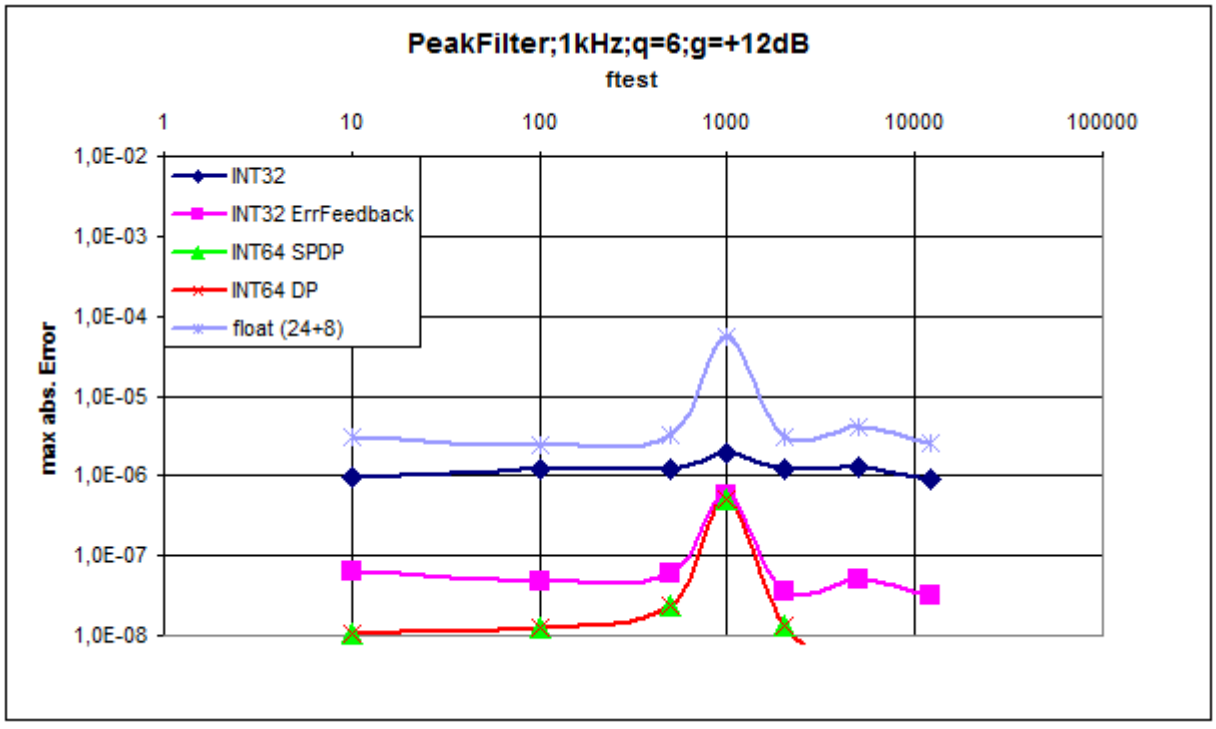

Fig. 5: PEAK PEQ Filter, 1k Hz, q=6; gain=+12dB for different precisions

Conclusion

It can be seen, that the float type should not be used, because it introduces always the most error. Also, the INT32 singlePrecision is not preferable but better than float, especially for the filters f0. The 64Bit DP solution gives the best results, but 64Bit SPDP generates almost the same errors. Single Precision INT32 with feedback is the medium type but with good results at the filters center frequency.

If you have much processing power, the 64Bit SPDP is a good choise, especially for low frequency filters. For higher frequency filters (>100Hz), also 32Bit with feedback is an usable solution, but it generates somewhat more noise if far away from the filters center frequency.

Float and INT32 without error feedback should be avoided under all circumstances.

For further reading and as reference

Philips TDA1540 16-BIT D/A conversion system (1984!)
http://www.dutchaudioclassics.nl/Philips-TDA1540-16bit-da-conversion/

The implementation of Recursive Digital Filters for High-Fidelity Audio (1988!)
Jon Dattorro
AES, VOL 36, No 11, 1988 November