



Designing fixed point Direct Form II (biquad) IIR filters with the ASN Filter Designer

January, 2012 (Rev 2)

Application note (ASN-AN021)

SYNOPSIS

Infinite impulse response (IIR) filters are useful for a variety of signal processing applications, including noise removal and speech coding. Although several practical implementations for the IIR exist, the direct form II structure is perhaps one of the most established. Thus, by virtue of the simple recursive implementation, a direct form II filtering algorithm is well suited to many fixed point micro-controller architectures, such as Microchip's PIC32.

INTRODUCTION

The IIR filter implementation discussed herein is said to be 'biquad', since it has two poles and two zeros as illustrated below in Figure 1. The biquad implementation is particularly useful for fixed point implementations, as the effects of quantization and numerical stability are of no great concern. However, the overall success of any biquad implementation is dependent upon the available number precision, which must be sufficient enough in order to ensure that the quantized poles are always inside the unit circle¹.

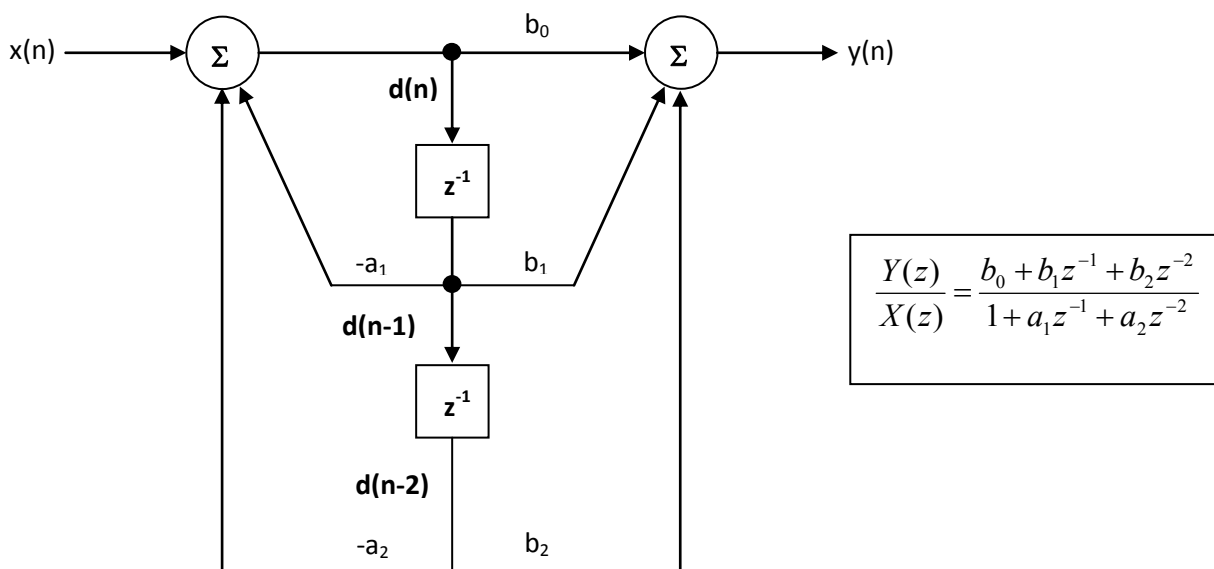


Figure 1 – Direct form II (biquad) IIR filter realization.

¹ The level of this application note assumes that the reader has a firm grasp of digital filtering and z-transform theory.

Analysing Figure 1, it can be seen that the biquad structure is comprised of two feedback paths (scaled by a_1 and a_2) and two feed forward paths (scaled by b_1 and b_2). Thus, the filtering operation of Figure 1 can be summarized by the following set of recursive equations:

$$d(n) = x(n) - a_1d(n-1) - a_2d(n-2) \quad (1)$$

$$y(n) = b_0d(n) + b_1d(n-1) + b_2d(n-2) \quad (2)$$

Analyzing Eqns. 1 and 2 and notice that the biquad implementation only requires four additions and five multiplications, which can be easily accommodated on any modern micro-controller/DSP. Also, implementing the direct form realization into two steps, i.e., Eqns. 1 and 2 significantly reduces the filter's sensitivity to quantization error.

1.1. Data scaling

In order to implement Eqns. 1 and 2 on a fixed point processor, several important data scaling considerations must be taken into account. The following discussion relates to an implementation on a 16-bit word architecture, where the theory is equally valid for any other type of word length.

1.1.1. Quantization

In order to correctly represent the coefficients and input/output numbers, the system word length (16-bit for the purposes of this application note) is first split up into its *number of integers* and *fractional* components. The general format is given by:

$$Q \text{ Num of Integers.Fraction}$$

If we assume that all of data values lie within a maximum/minimum range of ± 1 , we can use Q0.15 format to represent all of the numbers respectively. Notice that Q0.15 format represents a maximum of $1 - 2^{-15} = 0x7FFF$ and a minimum of $-1 = 0x8000$ (two's complement format).

In order to allocate poles anywhere inside the unit circle using Q0.15 quantization, the denominator filter coefficients are halved, and the numerator coefficients are scaled accordingly in order to compensate for the denominator normalization. This approach ensures that all coefficient values lie within the maximum/minimum data range (± 1), and that the filter's overall gain is maintained.

In order to ensure that the feedback path, $d(n)$ will not overflow, the following equation is used to calculate a data scaling factor (norm) when Eqn. 3 is excited by a Kronecker delta impulse function.

$$G = \sum_{n=0}^{\infty} |d(n)| \quad (3)$$

where, Eqn. 3 assumes that the filter is BIBO (bounded input, bounded output) stable.

Therefore, re-writing Eqns. 1 and 2 with the scaling factors, we obtain:

$$d(n) = 2 \times \left(\frac{x(n)}{2G} - \frac{a_1}{2} d(n-1) - \frac{a_2}{2} d(n-2) \right) \quad (4)$$

$$y(n) = 2G \times \left(\frac{b_0}{2} d(n) + \frac{b_1}{2} d(n-1) + \frac{b_2}{2} d(n-2) \right) \quad (5)$$

Eqns. 4 and 5 represent a general solution to the numerator and denominator coefficient representation problem. However, the reader is advised that if $|a_k| \leq 1$ and $|b_k| \leq 1$ then Eqns. 6 and 7 may be used, leading to a less extreme data scaling requirement and lower quantization errors.

$$d(n) = \frac{x(n)}{G} - a_1 d(n-1) - a_2 d(n-2) \quad (6)$$

$$y(n) = G \times [b_0 d(n) + b_1 d(n-1) + b_2 d(n-2)] \quad (7)$$

1.2. Filter design using the ASN Filter Designer

Fortunately, the aforementioned equations are fully implemented within the ASN Filter designer. Therefore, let us now consider the design of a simple 2nd order Elliptic lowpass IIR filter with the specifications shown in Table 1.

Order	=	2
Cut-off frequency	=	325 Hz
Sampling frequency	=	1000 Hz
Type	=	Elliptic
Passband ripple	=	0.1 dB
Stopband gain	=	-40 dB

Table 1 – lowpass filter specifications.

Entering the specifications into the ASN Filter designer, we obtain (double precision):

Filter Coefficients: Biquad 1

Numerator:

$$B1[0] = 0.63895378694532;$$

$$B1[1] = 1.27498563369662;$$

$$B1[2] = 0.63895378694532;$$

Denominator:

$$A1[0] = 1$$

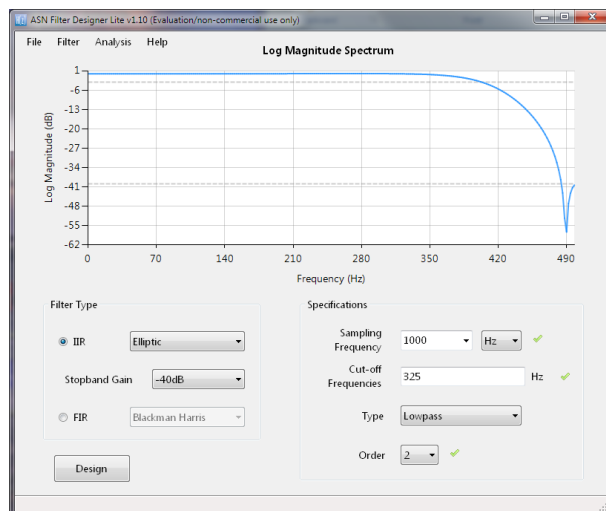
$$A1[1] = 1.14511388070077;$$

$$A1[2] = 0.437307900103239;$$

Scaling:

$$\text{Stability Gain Factor} = 1$$

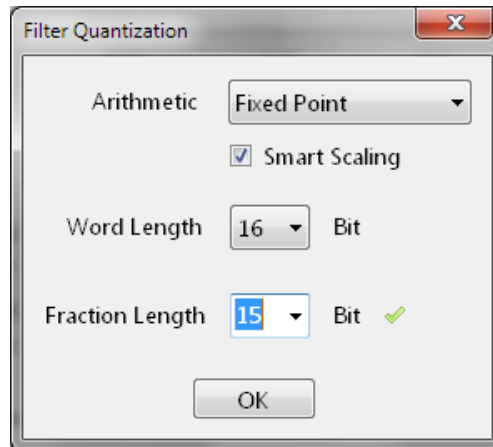
$$\text{Coefficient Scaling Factor} = 1$$



1.2.1. Quantization

Notice that both the second numerator (1.27498563369662) and denominator coefficient (1.14511388070077) are too large to be quantized using Q0.15 quantization. Therefore, as mentioned in section 1.1, the coefficients must be halved for correct implementation. Within the `Filter Summary`, this scaling operation is referred to as the *Coefficient Scaling Factor*, and may take either the value of either 1 or 2.

The desired degree of quantization (Q0.15 in our case) can be selected from the `Filter -> Arithmetic` menu, as shown below:



Selecting `Smart Scaling` will automatically ensure that coefficient and data scaling is applied. Thus, for the example considered herein, the scaled coefficients are shown below (in floating point and Hex format), along with the input/output scaling values, calculated using Eqns. 3 and 4.

Filter Coefficients: Biquad 1

Numerator:

B1[0] = 0.319488525390625; 0x28E5

B1[1] = 0.637481689453125; 0x5199

B1[2] = 0.319488525390625; 0x28E5

Denominator:

A1[0] = 0.5

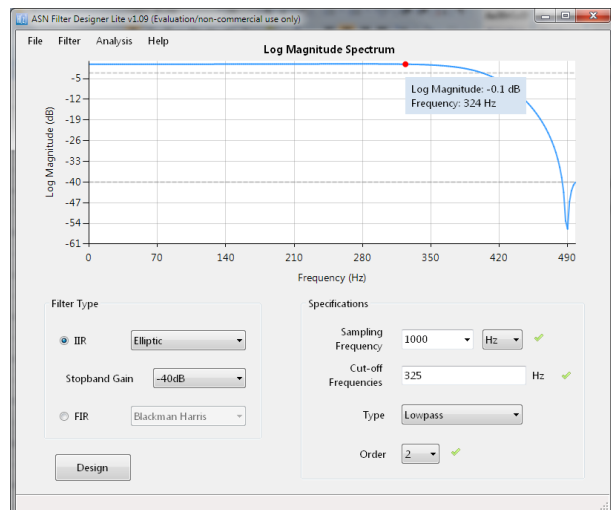
A1[1] = 0.57257080078125; 0x494A

A1[2] = 0.218658447265625; 0x1BFD

Scaling:

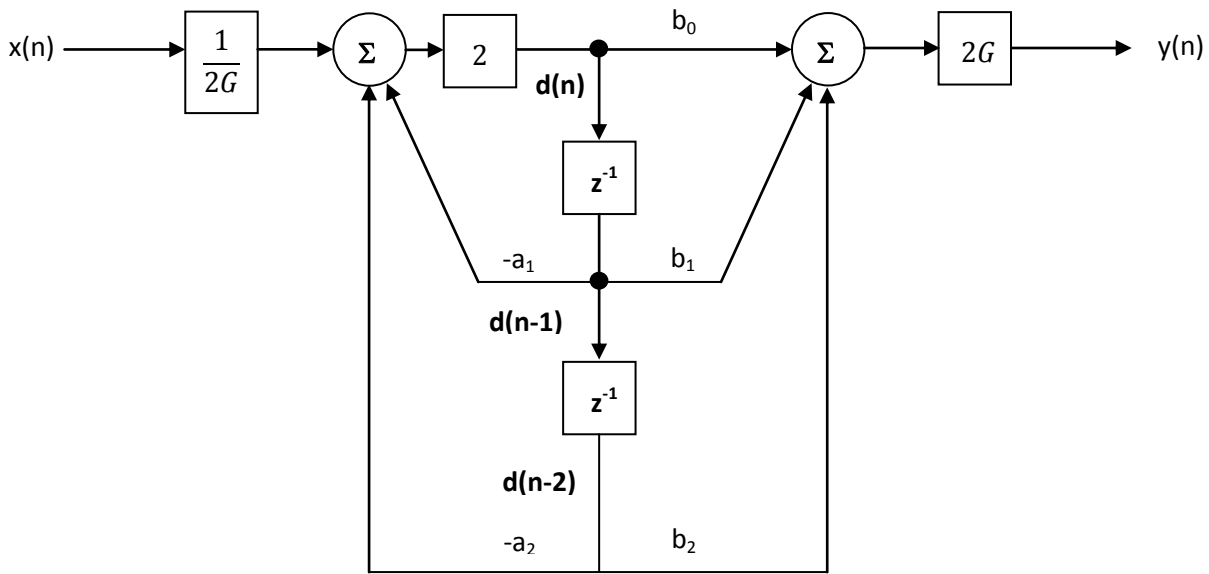
Stability Gain Factor = 8; 0x08

Coefficient Scaling Factor = 2



Note. The Hex format coefficients can now be directly implemented in hardware using Eqns. 4 and 5.

Using the designed gain scaling coefficients and redrawing the realization diagram of Figure 1, we obtain:



Where, G is the stability gain factor. Notice that $d(n)$ and the stability gain factor blocks are multiplied by 2 (the coefficient scaling factor) in order to compensate for the coefficient scaling. Entering the aforementioned coefficients into the diagram, we see:

$$G = 0x08$$

$$a_1 = 0x494A$$

$$a_2 = 0x1BFD$$

$$b_0 = 0x28E5$$

$$b_1 = 0x5199$$

$$b_2 = 0x28E5$$

As a final point, the reader is reminded that the designed denominator coefficients (i.e. a_1, a_2) are positive, and therefore careful attention to the sign should be exercised when implementing Eqn. 4. The following piece of Matlab demonstrates the complete biquad filtering operation:

```
d=zeros(3,1);
yn=zeros(N,1);

for n=1:N
    d=[0;d(1:end-1)];

    d(1) = CoeffScalingFactor*(x(n)/(CoeffScalingFactor*G) - (a(2)*d(2)) - (a(3)*d(3)));
    yn_G = (b(1)*d(1)) + (b(2)*d(2)) + (b(3)*d(3));

    yn(n) = yn_G*G*CoeffScalingFactor;
end
```