Bilinear-z Digital Filters Based on SFGs

Kenneth Martin
Granite SemiCom Inc.

Introduction

Digital filter realizations based on applying a bilinear transform to continuous-time transforms is a well used and understood design technique [1][2]. Such filters are often realized using a cascade structure [5]. Applying the bilinear-z transform directly to analog filters designed using integrator-based signal-flow-graph filters (SFGs) is not well understood for the simple reason that this approach results in delay-free loops which can not be easily realized. It is shown that using elementary matrix manipulations the delay-free loops can be eliminated while keeping the state-variables unchanged from the original realization containing the delay-free loops. This approach is predicted to preserve many of the excellent sensitivity properties of the analog SFG filters, especially those based on LC-ladder filters. A limitation of the presented approach is the realizations use many multipliers which can be inefficient depending on hardware realization details. A compensating advantage of the approach is it can be highly automated which helps minimize design errors; this is useful in quickly designing a good prototype filter for quick system evaluation and that later more optimized filters can be compared to. A number of examples are presented including a single-sided complex filter and a high-order real bandpass with unequal stop bands.

Eliminating Delay-Free Loops

A traditional method for realization analog filters was to first realize an LC filter that met specifications, and to then simulate this filter using signal-flow-graph techniques that used integrators, summers, and constant-coefficient gain multiplications [4]. The basic building block of this approach is shown in Fig. 1 (a). Each integrator can have inputs from the filter input and from other integrator outputs. Also, signals from other integrator blocks can be added before the output. Transforming this structure in the signal-flow-graph domain to a digital filter based on the bilinear-z results in the structure shown in Fig. 1 (b). This approach can be applied to all the integrator blocks in an SFG filter resulting in the structure shown Fig. 1 (c), but now the bold lines represent n signals (for an n'th order rea filter, the z^{-1} block represents n registers, and the K_I, K_F, and K_C blocks represent matrix multiplications. Analyzing the SFG of Fig. 1, we have:

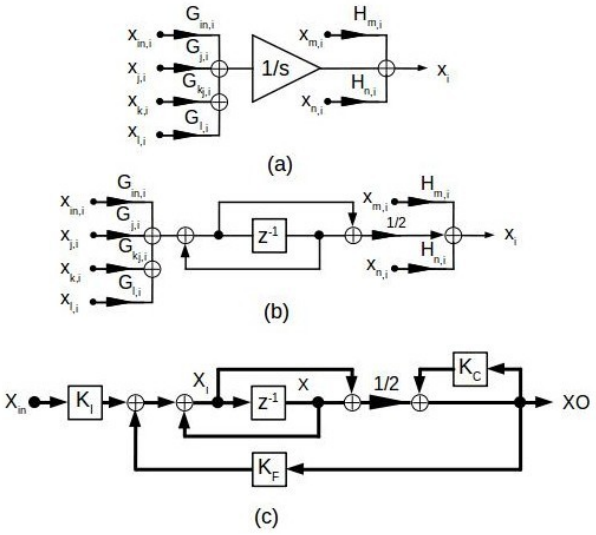

Fig. 1: SFG-based bilinear filters

$$X_O = \frac{(X_I + X)}{2} + K_C \cdot X_O$$
$$\Rightarrow X_O = \frac{[I - K_C]^{-1}}{2} \cdot (X_I + X) \quad (1)$$

Also, we have

$$X_I = K_F \cdot X_O + X + K_I \cdot x_{\text{in}} \quad (2)$$

Substituting (1) into (2) with some simple matrix algebra gives

$$X_I = \left[1 - \tfrac{K_F}{2} \cdot (I - K_C)^{-1}\right]^{-1} \cdot \left[I + \tfrac{K_F}{2} \cdot (I - K_C)^{-1}\right] \cdot X + K_I \cdot x_{\text{in}} \quad (3)$$

Letting $K_X = \frac{1}{2} \cdot (I - K_C)^{-1}$, $M = [I - K_F \cdot K_X]^{-1}$, $N = [I + K_F \cdot K_X]^{-1}$, and $K_{\text{FB}} = M \cdot N$ gives

$$X_I = K_{\text{FB}} \cdot X + K_I \cdot x_{\text{in}} \quad (4)$$

and

$$X_O = K_X \cdot X \quad (5)$$

Equations (4) and (5) are the basis of the simple SFG shown in Fig. 3. Realizing this digital filter straight-forward; furthermore, the state variables are identical to those of the unrealizable SFG of Fig. 1 (c) and therefore the sensitivity and noise properties should also be similar.

The proposed realization involves 2 nxn matrix multiplications (in most realizations, the K_{in} matrix has only a single element); this can be considerably more than a cascade realization of the same transfer function. This is the trade-off involved in return for good sensitivity and noise properties, and also for a design process that is ammenable to being highly automated. In some hardware realizations, for example, those that have a muliplier/accumulator where multiplies are no more

expensive than adds, or for realizations where the sampling rate is not too high, and a single easy to implement compution element, that is then time multiplexed, is desirable, the large number of multiplications may not be that detrimental. Even in applications where the number of multipliers is too expensive, the proposed method may still be useful in quickly realizing a good *prototype* filter during initial system design that will be later replaced by a more optimized filter having fewer multiplications. The proposed realization is also simple to implement in software which can be useful in initial system simulations.

Filter Realization Method

The proposed method for IIR filter realization is:

1. Design a good LC prototype filter that meets specifications.
2. Transform the LC prototype to an analog SFG realization.
3. Amplitude scale the analog SFG filter.
4. Calculate the K_{FB} and K_X matrices; this involves a frequency translation to get the desired passband frequency.
5. Implement the realization based on the SFG of Fig. 3.

Procedures for the steps 1. and 2. are well understood and documented in the literature, for example in [3]. A good tutorial description of steps 1. and 2. can be found at

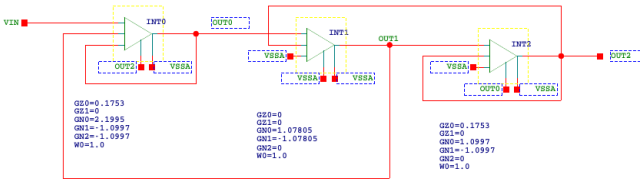

Fig. 2: The schematic of an SFG realization of the filter of Fig. 4 (b).

[8]. In the next section, a few examples will be presented. The author has found that using a Verilog-AMS model for an integrator to be effective in checking and eliminating errors in step 3. Currently steps 4 and 5 are done using python code chosen because it can easily be used to read and write *yaml* files that are useful in desribing the architectures and automating the design process (this will be illustrated shortly). In the future, we hope to also implement step 4 using Matlab. The python code for step 4. will be made available at github after the conference (assuming acceptance – it can't be made available before the conference due to pre-publication restrictions).

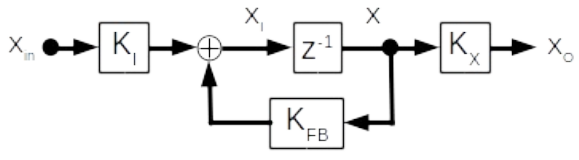

Fig. 3: Delay-free SFG realization.

Examples

```
module INTGR_VAMS(IN0, IN1, IN2, INZ0, INZ1, OUT);
        parameter real GN0=1;
        parameter real GN1=-1;
        parameter real GN2=0;
        parameter real GZ0=0;
        parameter real GZ1=0;
        parameter real wo=`M_TWO_PI*1e6;
        input IN0, IN1, IN2, INZ0, INZ1;
        inout OUT;
        electrical IN0, IN1, IN2, INZ0, INZ1, OUT;
        electrical V1; // internal
        analog
                begin
                V(V1) <+ GN0*V(IN0) + GN1*V(IN1) + GN2*V(IN2);
                V(OUT) <+ wo*idt(V(V1)) + GZ0*V(INZ0) + GZ1*V(INZ1);
                end
        endmodule // INTGR_VAMS
```

Fig. 5: Verilog-AMS code for integrator building block for SFG simulations.

(b)

Fig. 4: LC protoype for 3'rd order elliptic low-pass example.

The first example is a simple 3'rd order elliptic low-pass filter prototype taken from pg. 75 of the tables in Zverev [9]where θ=26. The prototype filter is shown in Fig. 4 (a), along with a modified LC prototype which simplifies the SFG filter design using the procedures described in [8]. The design process for the SFG filter based on Fig. 4 (b) is straight forward. Entering the filter coefficients and structure to enable caculating the matrices of equations (4) and (5) is possible in a number of different ways. One possibility is to use a schematic capture program with the integrator building blocks of Fig. 1 (a). The schematic shown in Fig. 2. The fundament building block for this SFG simulation is realized using the verilog-AMS module given in Fig. 2. This allows for very fast simulation of SFG filters to ensure the schematic has been correctly entered to simplify dynamic scaling of the SFG filter to equalize transfer function peaks.

In order to calulate the matrices needed for (4) and (5), a python program was chosen. The coefficient values and filter structure were input to the python program using *Yaml* input files [10] as these are easy to produce and to manipulate in python. The python program used a yaml format closely akin to a state-space description; it was found that generating this file was error prone. A simpler approach was to use a yaml file more closely related to a net list; this was the approach taken and then a conversion program was written to translated the netlist based yaml file (called *NYaml*) to the state-space based yaml file (simply called *Yaml*). A translation program was also written to take the netlist either produced by the schematic entry program or by an editor and translate it to either desired yaml file.

The python digital filter simulation program took the matrices generated from the yaml files and based on them

realized the filter structure of Fig. 3. In generating the matrices, the pre-warping used in the bilinear transform was based on

$$\mathrm{SclFct} = \tan\left(\frac{\pi \times F_{\mathrm{dig}}}{F_{\mathrm{anlg}}}\right) \qquad (6)$$

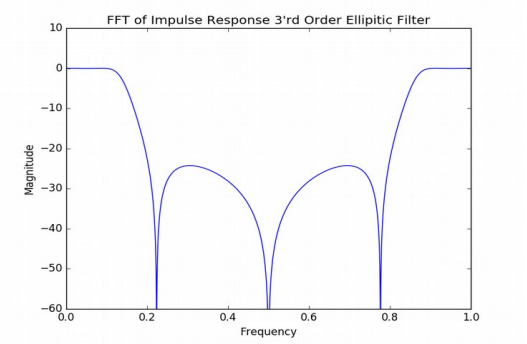

Fig. 6: Magnitude FFT of simulation of impulse respone

Where F_{anlg} is the passband edge frequency ($1\,\mathrm{rad.}/2\times\pi$) and F_{dig} was the desired digital passband edge frequency (0.1 times the sampling frequency in the example). The matices K_I and K_{FB} where multiplied by SclFctr The scaling of the filters was also automated using a python program where another yaml file was used to input the peak voltages of the integrators found using the verilog-AMS simulation of the unscaled SFG to modify the *Yaml* input. The impulse response of the filter was simulated for 8192 sample times and the output was analyzed using an FFT (from numpy) and then plotted using pyplot from Matplotlib. The plot obtained is shown in Fig. 6.

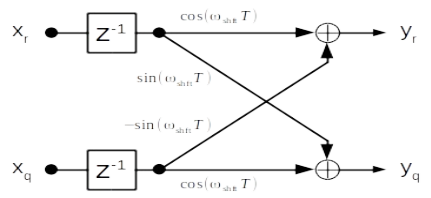

Fig. 7: Complex delay stage with frequency shift to realize positive frequency bandpass.

The filter implemention of Fig. 3 can be easily converted to a frequency shifted complex filter using the cross-coupled structure described in Fig 19(b) of [11] and shown in Fig. 7. A fifth-order complex positive-pass filter based on this approach is shown in Fig. 8. Other examples will be described in the presentation including an 8'th order bandpass filter.

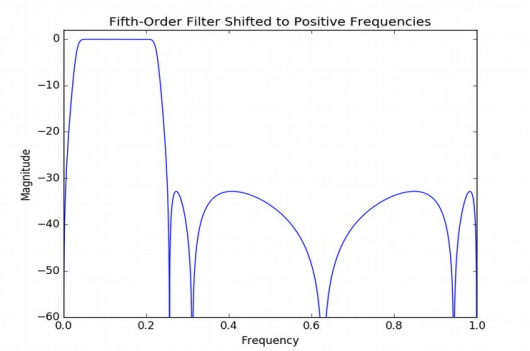

Fig. 8: Complex positive-pass filter

References

[1] L. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall, 1985.

[2] G. Temes and J. LaPatra, *Circuit Synthesis and Design*, McGraw-Hill, 1977

[3] A. Sedra and P. Brackett, *Filter Theory and Design, Active and Passive*, Matrix, 1978.

[4] K. Martin and A. Sedra, "Design of Signal Flow Graph Active Filters," *IEEE Trans. Circuits and Systems*, Vol. CAS-25, No. 4, pp. 184-195, April 1978.

[5] A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing*, Prentice Hall 1989

[6] K. Martin and A. Sedra, "Strays-Insensitive Switched-Capacitor Filters Based on Bilinear z- Transform," *Electronics Letters*, Vol. 15, No. 13, pp. 365-366, June 1979.

[7] K. Martin and A. Sedra, "Exact Design of Switched-Capacitor Bandpass Filters Using Coupled-Biquad Structures," *IEEE Trans. Circuits and Systems*, CAS-27, No. 6, pp. 469478, June 1980.

[8] Haideh Khorramabadi, "EECS 247 Analog-Digital Interface Integrated Circuits Lecture Notes," https://inst.eecs.berkeley.edu/~ee247/fa07/files07/lectures/L4_fo7.pdf, 2007.

[9] A. Zverev, *Handbook of Filter Synthesis*, Wiley, 1967

[10] http://pyyaml.org/

[11] K. Martin, "Complex Signal Processing is Not Complex," *IEEE Transactions on Circuits and Systems-I*, Vol. 51, pp. 1823-1836, Sept. 2004.