The Reaktor Core tutorial to accompany "Preserving the LTI system topology in s- to z-plane transforms"

Vadim Zavalishin*

May 5, 2008

WARNING. This tutorial is not a beginner one. The basic knowledge of DSP theory as well as a good command of Reaktor Core is assumed.

Also, this is a tutorial and not a walkthrough. Be prepared to that a good amount of the material will be available only in the form of hints, with corresponding exercises.

1 Introduction

The article "Preserving the LTI system topology in s- to z-plane transforms", which this tutorial is supposed to accompany, describes a small set of DSP techniques for the design of discrete-time LTI systems. The purpose of this tutorial is to provide examples of how these techniques can be used within the Reaktor Core environment.

2 Bilinear Integrator

The bilinear-transformed integrator structure from Fg. 1[1] can obviously be implemented in Reaktor Core (Fig. 1). The *w/2* input will be accepting the halved prewarped cutoff value. In order to be able to employ such integrator according to Eq. 2 we introduce (Fig. 2) the inputs/outputs s and g which accept the respective values corresponding to the input signal

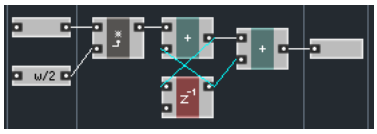


Figure 1: Bilinear integrator.

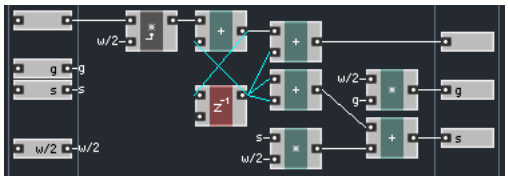


Figure 2: Bilinear integrator with g and s ports.

of the integrator and produce the new values corresponding to the output signal of the integrator. Make sure, that the *Solid* property of the integrator macro is disabled!

Questions and exercises

1. Explain the structure of the integrator in Fig. 2.

2. Don't you think an audio-rate latch somewhere in Fig. 1 and Fig. 2 would be a good practice? At which place? Make a version of Fig. 2 *with* the latch.

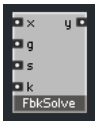
3. Why does the *Solid* property need to be disabled? If you cannot answer this question now, try answering it again after working through the 2-pole filter implementation later in the tutorial.

*Native Instruments GmbH

[1] To tell between the figures from the article and from this tutorial we denote the figures from the article with Fg. and the figures from the tutorial with Fig. The equations from the article will be denoted with Eq.

3 Helper macros

Since it's likely that we'll have to solve Eq. 2 in more than one case, we should build an *FbkSolve* macro which performs the necessary computations:



We also need a macro to compute the halved prewarped cutoff. For that we can simply use the macro *Prewarp BT*, contained inside the *6dB LP/HP EQ* standard macro. The *Prewarp BT* macro accepts the frequency in Hz as the input and produces the halved prewarped cutoff for bilinear-transformed filters in response.

Questions and exercises

1. Explain the possible relative rates of update of the values at the inputs of the *FbkSolve* macro. Which values change at audio rate? Which values change when the system parameters change?

2. Create an implementation of the *FbkSolve* macro. Ignore the case of $1 + gk \leq 0$. Take into account the possible update rates of the input parameters and insert latches and/or modulation macros accordingly.

3. Create another version of the *FbkSolve* macro, which produces two large values (e.g. -10^5 and 10^5) if $1 + gk \leq \varepsilon$ where ε is some small positive number (you have to figure out the appropriate value). This version will be used in the nonlinear case.

   Do you need another input port for this version? Which one?

4 1-pole filter

Using the integrator in Fig. 2 and the *FbkSolve* and *Prewarp BT* macros we can implement the 1-pole lowpass filter from Fg. 2 (Fig. 3).

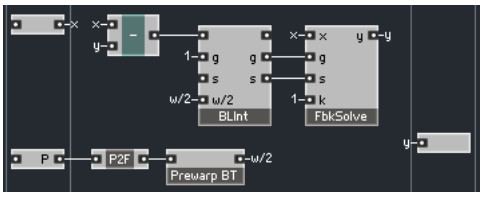


Figure 3: 1-pole state-variable lowpass filter.

Questions and exercises

1. Explain the relationship between Fg. 2 and Fig. 3.

2. Is there a good-practice latch missing in Fig. 3?

3. Can the structure in Fig. 3 get unstable?

4. Why is the integrator output not used? How could it be used?

5. Make a multimode HP/LP version of the filter.

6. Extend the filter implementation with g and s input and output ports, similarly to how it was done for the integrator, so that it can be later used as a building block in larger structures.

7. Create a discrete-time structure diagram version of Fg. 2 consisting only of adders, gain elements, and z^{-1} delays by expanding the integrator icon to a bilinear integrator structure. Now, explicitly incorporate the Eq. 2 into the diagram, so that it does not have delayless feedback loops anymore. Implement this diagram in a Reaktor Core structure. Are there any differences to the one in Fig. 3?

5 2-pole filter

The two pole filter from Fg. 4 is implemented in Fig. 4. The *Res2d* macro is taken from the *2 Pole SV C* standard macro and it converts the resonance $0 \leq Res < 1$ to

$$1/Q = d = 2(1 - Res)$$

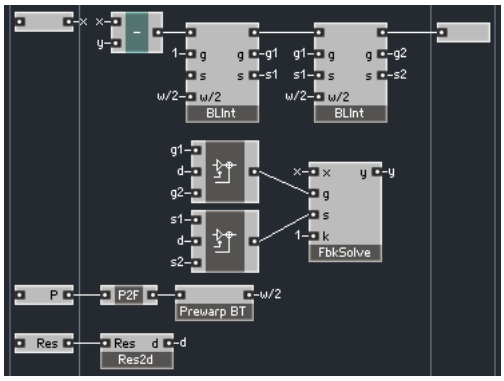


Figure 4: 2-pole state-variable lowpass filter.

so that $1/2 < Q < \infty$. The two modulation multiply-add macros are not 100% semantically correct, but they work.

Questions and exercises

1. Explain the relationship between Fg. 4 and Fig. 4.
2. Don't you think parameter clipping would be in order? Where?
3. Can the structure in Fig. 4 get unstable?
4. Make a multimode HP/LP/BP version of the filter.

6 Ladder filter

Now you should have enough experience with the material to make the ladder filter on your own. Therefore:

Questions and exercises

1. Make a Reaktor Core implementation of the transistor ladder filter model, using the *FbkSolve* macro and the 1-pole filter macros with g and s ports which you created earlier.
2. Analyse the resulting implementation's stability.

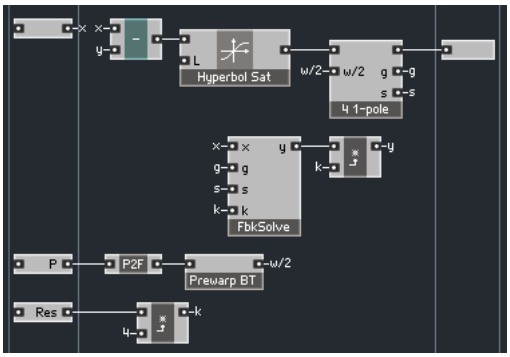


Figure 5: Ladder filter with a cheap and dirty saturator.

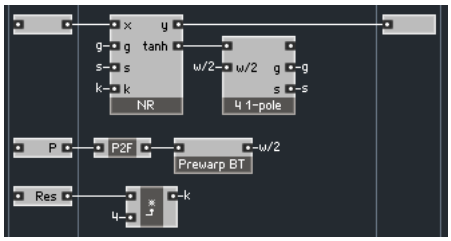


Figure 6: Ladder filter with a Newton-Raphson saturator.

7 Nonlinearities

We begin with the 'cheap and dirty' implementation of a nonlinearity in the ladder filter model in Fg. 5, as explained at the end of the section 4 of the article (Fig. 5). The *4 1-pole* macro contains 4 serially connected 1-pole filters with g and s 'thru' ports, the outputs of the last one available on the outside. Don't forget to disable the *Solid* property at the appropriate places!

The Newton-Raphson based implementation (Fig. 6) is somewhat more challenging. The *NR* macro computes the solution to Eq. 5 by explicitly running 5 iterations of the method (Fig. 7). Don't forget to provide the correct initialization value!

Questions and exercises

1. Implement the missing components for the structures described in this section.

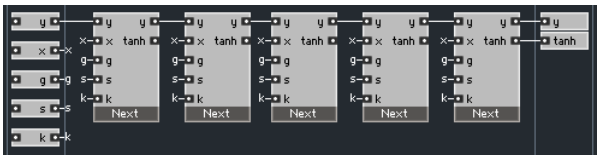


Figure 7: 5 Newton-Raphson iterations.

2. Build a 'cheap and dirty' implementation where the *FbkSolve* macro in the main feedback handles the instable case of Eq. 2. Try the 'unstable' resonance values.

3. Create the implementations for the parabolic and hyperbolic saturator cases, solving Eq. 4 analytically.

4. Analyse the resulting implementation's stability.

8 Homework

Here are some additional implementation challenges for you. Enjoy!

Questions and exercises

1. Implement a phaser and a flanger using the approach described in the article.

2. Design your own filter or effect using the same approach.

Acknowledgements

REAKTOR® and *REAKTOR Core Technology*® are registered trademarks of Native Instruments GmbH.