# Automatic Analogue Circuit Synthesis using Genetic Algorithms

**James B. Grimbleby**
**Electronic Engineering Group**
**The University of Reading**
**PO Box 225**
**Reading RG6 6AY**

**Telephone: (0118) 9318586**
**Fax: (0118) 9318583**
**email: j.b.grimbleby@reading.ac.uk**

**pgp:** EC18 B1B1 AACC 52E1 976C BFAC 6A20 27F8 BBFC DC94

December 1, 2000

## Abstract

Most analogue systems are designed manually because automatic circuit synthesis tools are presently available for only a limited range of design problems. This paper presents a new approach to circuit synthesis based on Genetic Algorithms. Using this method it is possible in principle to synthesize circuits to meet any linear or non-linear, frequency-domain or time-domain, specification. When applied to existing filter design problems this circuit synthesis method produces design solutions that are more efficient than those resulting from formal design methods or created manually by an experienced analogue circuit designer.

## 1 Introduction

Digital systems can now be synthesized automatically on a computer but this is not the case for analogue systems. Formal design solutions do exist, but only for limited classes of analogue design problems (for example linear frequency-domain filters) and most analogue circuit design is still performed manually by skilled engineers.

These observations on analogue circuit synthesis do not extend to circuit analysis, and excellent analogue circuit analysis tools such as SPICE have been available for many years. It might be thought that in order to synthesize circuits it is simply necessary to "close the loop" around circuit analysis. Indeed this is exactly what happens when a circuit is subjected to numerical optimization. Unfortunately, although numerical optimization is a true design technique, its application is severely limited by the fact that it operates only on fixed circuit topologies.

If an optimization technique could be found that modified both circuit topology and component values then this could form the basis of an analogue circuit synthesis method. Genetic Algorithms (GAs) are just such an optimization technique. Developed in the 1970s and '80s [1,2], GAs have been successfully applied to a wide range of numerical and non-numerical optimization problems.

## 2 Genetic Algorithms

GAs operate on the principle of "survival of the fittest", generating new design solutions from a population of existing solutions, and discarding those design solutions which have an inferior performance or fitness. Each member of the population has a "chromosome" which consists of a number of "genes"; each gene represents one part of the design solution. New design solutions are created by "breeding" from a pair of existing solutions. Parents are randomly selected from the present population, but with a bias towards the fittest individuals and their chromosomes are merged to produce the chromosome of the offspring. A proportion of offspring is then subjected to random mutation.

A related technique is Genetic Programming (GP) in which the chromosome represents not the design solution directly but a variable-length expression containing terminals and operators which, when evaluated, yields the design solution [3]. In GPs it is this expression, rather than the design solution itself, that is subject to cross-over and mutation.

## 3 Application of Genetic Algorithms to Circuit Synthesis

One approach to circuit synthesis is to allow a GA or GP to determine both the topology and the component values. This approach has been adopted by Koza, Bennett, Andre and Keane [4,5], who use GP with a variable-length expression containing topology modifying operators, component-creating operators and arithmetic-performing operators. In Koza's approach the operators that modify the circuit topology and select component values are inseparable, and all are under the control of the evolutionary processes operating on the expression. Circuit fitness is evaluated using a SPICE simulator, the code for which has been incorporated into the synthesis program.

Circuit synthesis involves both the selection of a suitable topology and choice of component values, and as Koza has shown, these may be optimized simultaneously. However there is no reason why these operations should not be performed separately, with different optimization methods being used. Clearly the circuit topology must be chosen first, and the appropriate algorithm for this is a GA. For each circuit topology generated the component values can then be optimized, and the performance of the circuit returned as the fitness function to the GA.

The component values could also be optimized using a GA but, for problems involving well-behaved objective functions dependent on the values of a fixed number of variables, it is well established that numerical optimization methods converge much faster and involve fewer objective function evaluations. No optimization method guarantees to find the global optimum, but it has been found [6] that numerical optimization of component values achieves a high proportion of results close to the global optimum. This hybrid approach using a GA to select a suitable topology and numerical optimization to choose component values is likely to be more efficient than using a GA or GP to perform both functions.

## 4 The Hybrid Genetic Algorithm

The most important decision in applying GAs to an optimization problem is how to represent the design solution in the chromosome. A circuit topology can be specified by a list of component types together with their terminal nodes and it is natural, therefore, to make each gene of the chromosome represent a single component. Chromosomes are normally of fixed length and contain a fixed number of genes whereas circuits can contain any number of components. This problem can be avoided by having, in addition to the standard component types of resistor, capacitor and inductor, an "empty" component type. Since the GA operates on circuit topology only, the genes do not contain component values. In C++ the data members of the *gene* class are defined:

```
enum component_type {resistor,
capacitor, inductor, empty};
typedef short int node;

class gene {
 private:
    component_type type;
    node n1, n2;
 public:
    ....
};
```

The data member of the *chromosome* class is simply an array of genes:

```
class chromosome {
 private:
    gene genes[16];
 public:
    ....
};
```

The number of genes determines the maximum circuit complexity so that this parameter should be at least equal to the anticipated complexity of the optimum circuit configuration. In fact it should be somewhat larger than this to allow the circuit to evolve towards the optimum by component deletion. Choosing too large a value, however, leads to inefficiency because unrealistically complex circuits will be generated, and will need to be analyzed.

The input of the circuit is always node 1 and the output is node 2; node 0 is the reference (or ground) node. Figure 1 shows a bridged-T filter with numbered nodes. A possible chromosome (containing only 6 genes for clarity) representing this circuit is shown in figure 2.

New individuals are generated by uniform genetic cross-over from two parents (with probability typically 70%) or by mutating a single parent. In either case further mutation (with probability typically 50%) may then be applied. The efficiency of the GA does not depend critically on either of these probabilities. Parents are selected by tournament between two randomly-chosen individuals.

The population size is normally 80-200; a larger population would probably result in an increased robustness of the GA, but would lead to longer synthesis times. Each individual of the initial population is created by repeatedly generating random chromosomes until one is found that corresponds to a viable (connected) circuit. Steady-state replacement of the population is used: a new individual replaces the least-fit member of the existing population provided that it has a superior fitness.

The crudest form of random mutation would be to select randomly a gene in the offspring's chromosome, and replace it with a new gene of random type and random terminal nodes. Unfortunately this process generates a high proportion of lethal mutations in which a viable circuit is transformed into a non-viable circuit. The success rate of mutation can be improved by modifying the circuit only in ways that are likely (but not guaranteed) to lead to a viable result. One of the following four circuit transformations, selected at random, is applied to the offspring's chromosome:

1. Replace an existing component by an open circuit.
2. Replace an existing component by a short circuit.
3. Connect a new random component in parallel with an existing component.
4. Connect a new random component in series with an existing component.

These operations alone would be sufficient to transform any given circuit into any other circuit, and it might be thought that cross-over is therefore superfluous. This is in fact not the case as the use of cross-over greatly improves the efficiency.

Once a circuit topology has been generated its fitness is evaluated after numerically optimizing its component values using a quasi-Newton algorithm based on the Davidon-Fletcher-Powell (DFP) method. This calls an objective function that returns the sum-of-squares of the differences between the circuit's response and the target response at a sequence of logarithmically-spaced frequencies. To reduce the amount of computation involved a symbolic analysis of each new circuit topology is performed before numerical optimization. This involves constructing voltage and current graphs (corresponding to the voltage and current incidence matrices) from the circuit, and after coalescing appropriate pairs of nodes, finding all the common spanning trees of the two graphs [7]. The result is two linked lists of symbolic terms, corresponding to the numerator and denominator of the voltage frequency-response function. Substituting component values into the symbolic form gives the numerical frequency-response function; substituting frequency into the numerical frequency-response function gives the voltage gain.

As the response approaches the target response, the objective function tends to zero and the reciprocal of the objective function for the optimized circuit is returned as the fitness. In the absence of any other constraints the GA will generate successively more complex circuits, because a complex circuit will in general provide a better fit to a target response than a simple circuit. To prevent this the fitness is multiplied by a penalty function $p(n)$ that is unity for simple circuits, but which rapidly becomes smaller as the complexity (measured by the number of nodes $n$) exceeds some predetermined level *n_max*:

$$p(n) = \frac{1}{1 + a^{(n - n\_\max)}} \qquad 1.$$

where a is a constant, typically 8 and *n_max* is set to the anticipated complexity.

This hybrid-GA circuit synthesis method is remarkable for incorporating no design rules or expert knowledge; it simply works towards satisfying the design goals. It is the antithesis of the traditional "expert system" approach to analogue circuit design.

## 5 A Simple Frequency-Domain Filter Benchmark

An obvious way of testing the effectiveness of the hybrid-GA is to synthesize a circuit to a specification for which a formal design method exists. Consider the normalized low-pass filter specification:

| | |
|---|---|
| Pass-band edge: | 1.0 rad/s |
| Stop-band edge: | 1.5 rad/s |
| Maximum pass-band gain: | 0 dB |
| Minimum pass-band gain: | -1 dB |
| Maximum stop-band gain: | -46 dB |

Following the traditional filter design procedure, the first stage is to choose a suitable filter approximation. Provided that pass-band ripples and a non-monotonic stop-band are acceptable, the most efficient filter approximation is the elliptic. The lowest-order elliptic approximation meeting the specification is of 5th order, and the filter can be implemented as an equally-terminated ladder filter as shown in figure 3.

To allow for the fact that the maximum pass-band gain of an equally-terminated filter is -6 dB, all of the gains in the specification must be reduced by 6 dB. The impedance levels in this filter are around 1 Ω and any practical implementation would require them to be scaled to a more realistic level. For example, the resistor and inductor values could be multiplied by 10000, and the capacitor values divided by 10000. Figure 4 shows the corresponding frequency response.

The hybrid-GA circuit synthesis program generated the circuit shown in figure 5. Using a population size of 80 circuits and 100 generations the time taken to synthesize this filter on a 300 MHz PII personal computer was under 3 hours. The response of the hybrid-GA design is shown in figure 6, and it is clear that the design is fully compliant with the specification.

Surprisingly the hybrid-GA design is not simply as good as the elliptic filter, but with only six reactive components is actually a more efficient design. Although the hybrid-GA design meets the specification, nevertheless it is probably sub-optimal with respect to performance factors (such as component value sensitivity) that are not included in the design goals. This difficulty can in principle be overcome simply by modifying the objective function to include these performance factors, but this will certainly adversely affect the speed of operation of the synthesis program.

As far as component value sensitivity is concerned the synthesis program can be modified to include an item of "expert knowledge": equally-terminated ladder configurations are known to have low sensitivity. The synthesis program was therefore constrained to generate only LC circuits between equal-value termination resistances and the result is shown in figure 7. The response of this circuit is shown in figure 8. Again the synthesized circuit meets the specification, while using fewer components than the filter resulting from the traditional formal design process based on an elliptic response.

## 6 Nielsen's Filter Design Problem

In a paper describing a continuous-time filter compiler Nielsen [8] uses as an example a highly asymmetric band-pass filter specification for a modem application:

| | |
|---|---|
| Pass-band: | 31.2 kHz to 45.6 kHz |
| Maximum pass-band ripple: | 0.6 dB |
| Lower stop-band edge: | 20 kHz |
| Lower stop-band gain: | < -38 dB |
| Upper stop-band: | 69.6 kHz to 84.0 kHz |
| Upper stop-band gain: | < -73 dB |
| Gain above stop-band: | < -55 dB |

Unfortunately Nielsen's filter compiler is based on traditional filter design methods and can only generate filter responses that are of the standard types (low-pass, high-pass, symmetrical band-pass and symmetrical band-stop). Although standard filter responses can be found to satisfy asymmetric specifications such as that given above, they are likely to be of unnecessarily high order. For example, to meet Nielsen's specification a 10th-order elliptic filter is required, the response of which is shown in figure 9.

This clearly achieves an unnecessarily high degree of attenuation in the lower stop-band. Nielsen was therefore forced to design by hand a suitable passive prototype filter. His design consists of an 8th-order doubly-terminated filter containing 10 reactive components in addition to the 2 termination resistances.

Koza [4] uses this filter specification as a demonstration of the effectiveness of his GP circuit synthesis program. After 199 generations a circuit emerged that meets the specification, but which contains a total of 38 components and is clearly not a cost-effective design.

The hybrid-GA synthesis program described here uses randomly-chosen component values around unity as a starting point for numerical optimization, and optimization is most likely to succeed if the values corresponding to the global optimum are also around unity. This can be achieved by setting the impedance levels to 1 Ω and by scaling the filter cut-off frequencies to around 1 rad/s. In the case of Nielsen's filter a frequency scaling factor of $h = 2\pi\times45000$ is used, and the scaled specification is used as the target response for the hybrid-GA circuit synthesis program.

Using Nielsen's specification, and constrained to generate an equally-terminated configuration, the hybrid-GA generates the circuit shown in figure 10. It contains one fewer component than Nielsen's manually-designed filter. This 8th-order circuit would not have resulted from any traditional design procedure. Although it superficially resembles a traditional equally-terminated ladder filter, the series combination: Ld/Ch and the series/parallel combination: Lg/Cj/Ll are non-standard.

Of course the response of this circuit is centered on 1 rad/s rather than the 45 kHz of the specification. To convert the filter to a form which meets the original specification the capacitor and inductor values are divided by the normalizing factor $h$. The impedances also need to be brought up to a more practical level. Figure 11 shows the filter response.

It is clear that this filter is fully-compliant with the specification, while at the same time being more efficient (in terms of the number of components) than the traditional elliptic-derived filter, Koza's GP circuit synthesis filter and Nielsen's manually-designed filter.

## 7 A Time-Domain Design Problem

There is no reason why the hybrid-GA synthesis method should be limited to frequency-domain filters. Given a circuit topology and component values it is only slightly more complicated to calculate the impulse or step response than the frequency response.

Consider the following unit-step response specification for an approximation to the ideal averaging filter:

$$g(t) = t \pm 0.02 \quad \text{for} \quad 0 \le t < 1$$
$$g(t) = 1 \pm 0.02 \quad \text{for} \quad 1 \le t$$

This specification is compared with the circuit's response at intervals of 0.05 s from 0 s to 4.0 s and is the only design information supplied to the synthesis program. The actual unit-step response of the circuit under consideration is calculated by the matrix exponential method. Each circuit topology is analyzed to give the symbolic transfer function. To calculate the response the component values are substituted into the transfer function, the state equations derived, and the transition matrix calculated for the required time step. Only a single matrix multiplication is then required to calculate the response at each time point.

Using a population size of 80 circuits and 200 generations the filter shown in figure 12 was synthesized in around 10 hours. Figure 13 shows the unit-step response of this filter and it is clear that the synthesized filter meets the required time-domain specification.

## 8 Conclusion

The design examples presented above indicate that the hybrid-GA method of circuit synthesis is practical on widely-available personal computers, and generates circuits that are efficient and fully meet the design goals. Driven purely by the specification, the circuit synthesis program incorporates no expert knowledge of circuit design (except in some cases where the topology is constrained to be of equally-terminated form).

Circuits generated by the synthesis program are often novel, and would not have been generated by any established design technique. Indeed the synthesis program appears to be creative in a way that is not often associated with computers. In some cases the circuits generated are more efficient than those resulting from traditional design methods (as the example of the sharp cut-off filter demonstrated).

In principle the hybrid-GA approach can be applied to any circuit design problem for which a means of evaluating potential circuits against the design goals is available. Computational effort is the only limiting factor.

Almost all the computational effort goes towards calculating circuit responses. Each new circuit must be optimized numerically to obtain the component values, and this will involve many thousands of response evaluations. Linear circuit analysis is fast and efficient, and synthesizing circuits with up to 16 components is possible on standard personal computers. Non-linear analysis, by contrast, involves the solution of non-linear differential equations and absorbs far more computational effort than linear analysis. At the present time it is impractical to synthesize non-linear circuits of any complexity on PCs using the hybrid-GA method.

**References**

1. HOLLAND, J.H.: 'Adaptation in Natural and Artificial Systems' (University of Michigan Press, Ann Arbor, Mich., 1975).

2. GOLDBERG, D.E.: 'Genetic Algorithms in Search, Optimization, and Machine Learning' (Addison-Wesley Publishing Company, Reading, Mass., 1989).

3. KOZA, J.R.: 'Genetic Programming: On the Programming of Computers by Means of Natural Selection', (MIT Press, 1992).

4. KOZA, J.R., BENNETT III, F.H., ANDRE, D. and KEANE, M.A.: 'Automated WYWIWYG Design for Both Topology and Component Values of Electrical Circuits using Genetic Programming', Genetic Programming 1996: Proceedings of the First Annual Conference, July 1996, Stanford University, Cambridge MA, pp. 123-131.

5. KOZA, J.R., BENNETT III, F.H., ANDRE, D., and KEANE, M.A.: 'Four Problems for which a Computer Program Evolved by Genetic Programming is Competitive with Human Performance', Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, May 1996, Nagoya, Japan, pp. 1-10.

6. GRIMBLEBY, J.B.: 'Hybrid Genetic Algorithms for Analogue Network Synthesis', IEE/IEEE International Congress on Evolutionary Computation (CEC99), July 1999, Washington DC, pp 1781-1787.

7. GRIMBLEBY, J.B.: 'Algorithm for finding the common spanning trees of two graphs', *Electron. Lett.*, 1981, **17**, pp 470-471.

8. NIELSEN, I.R.: 'A C-T Filter Compiler – From Specification to Layout', *Analog Integrated Circuits and Signal Processing*, 1995, **7**, pp. 21-33.

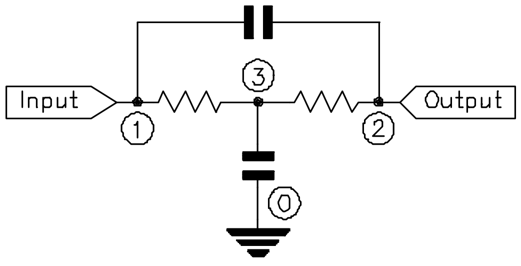


**Figure 1: Bridged-T filter**

| | *Gene 1* | *Gene 2* | *Gene 3* | *Gene 4* | *Gene 5* | *Gene 6* |
|---|---|---|---|---|---|---|
| ***Type*** | **Resistor** | **Empty** | **Capacitor** | **Resistor** | **Capacitor** | **Empty** |
| ***$n_1$*** | **3** | **-** | **3** | **1** | **2** | **-** |
| ***$n_2$*** | **2** | **-** | **0** | **3** | **1** | **-** |

**Figure 2: Chromosome representing bridged-T filter**

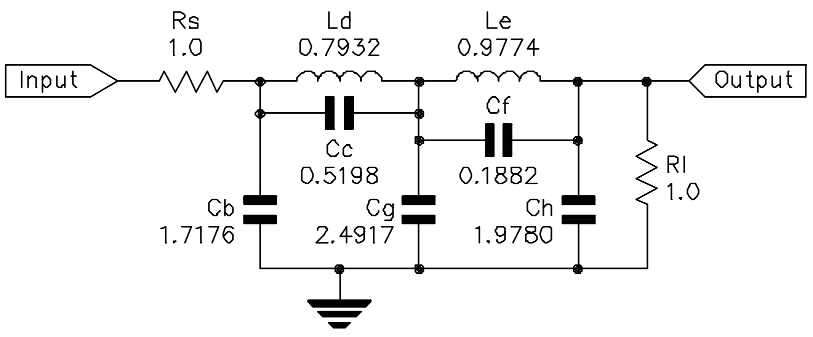
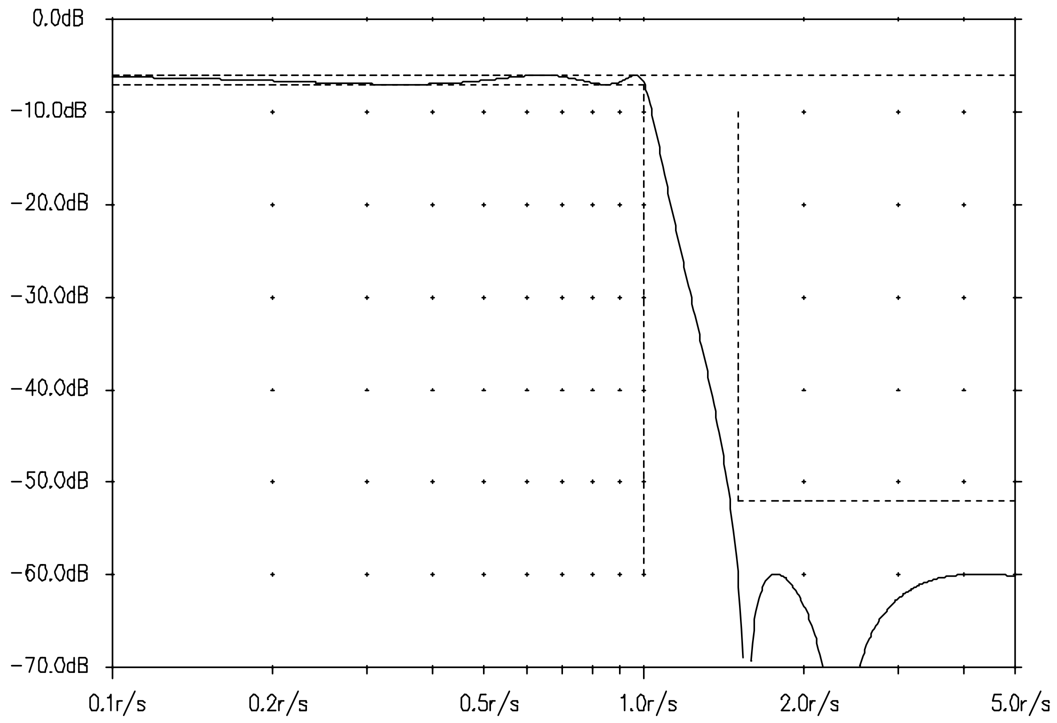


**Figure 3: Equally-terminated elliptic filter**

Gain



Frequency

**Figure 4: Frequency Response of elliptic filter**

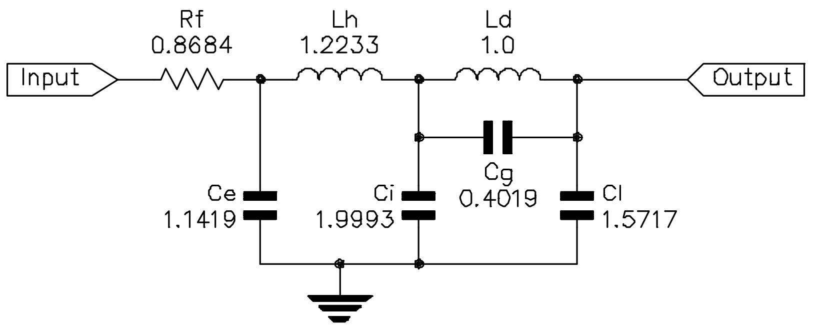
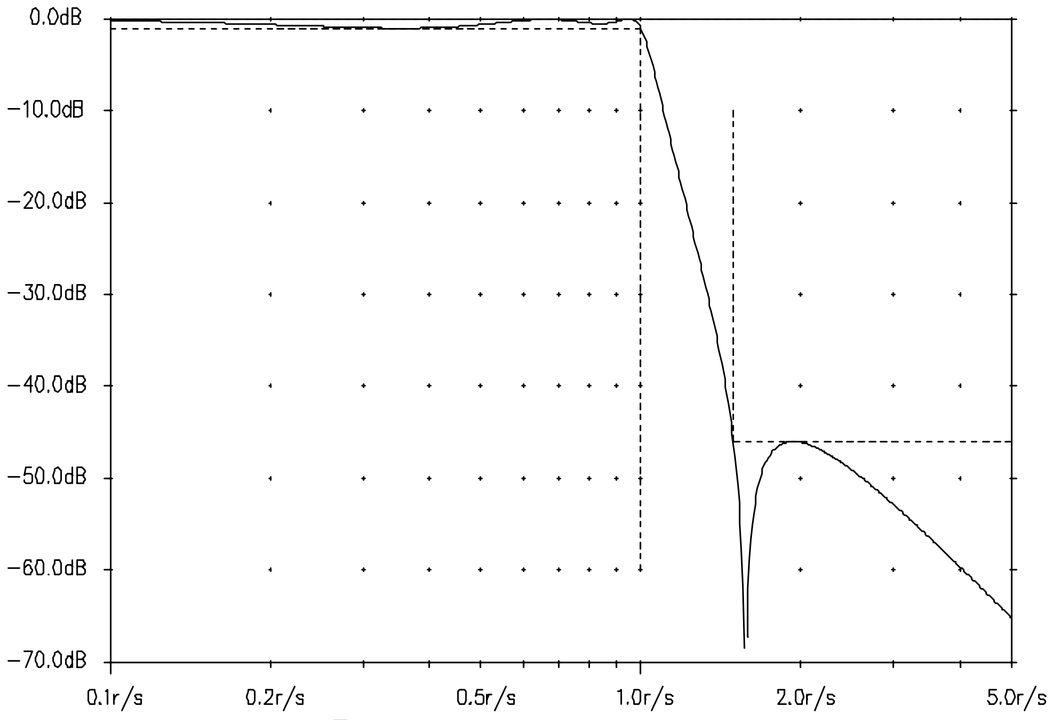


**Figure 5: Hybrid-GA designed filter**

Gain



Frequency

**Figure 6: Frequency response of hybrid-GA designed filter**

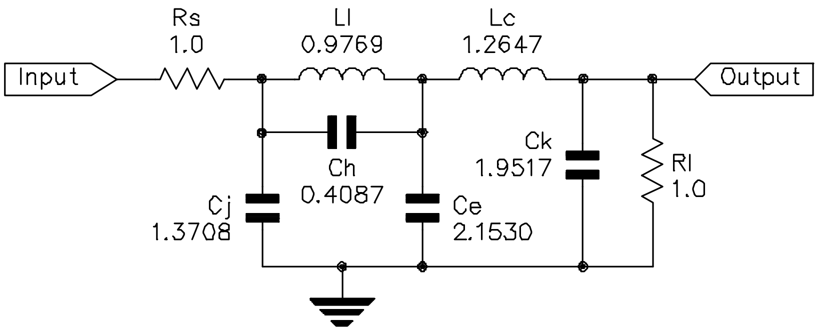


**Figure 7: Hybrid-GA designed equally-terminated filter**

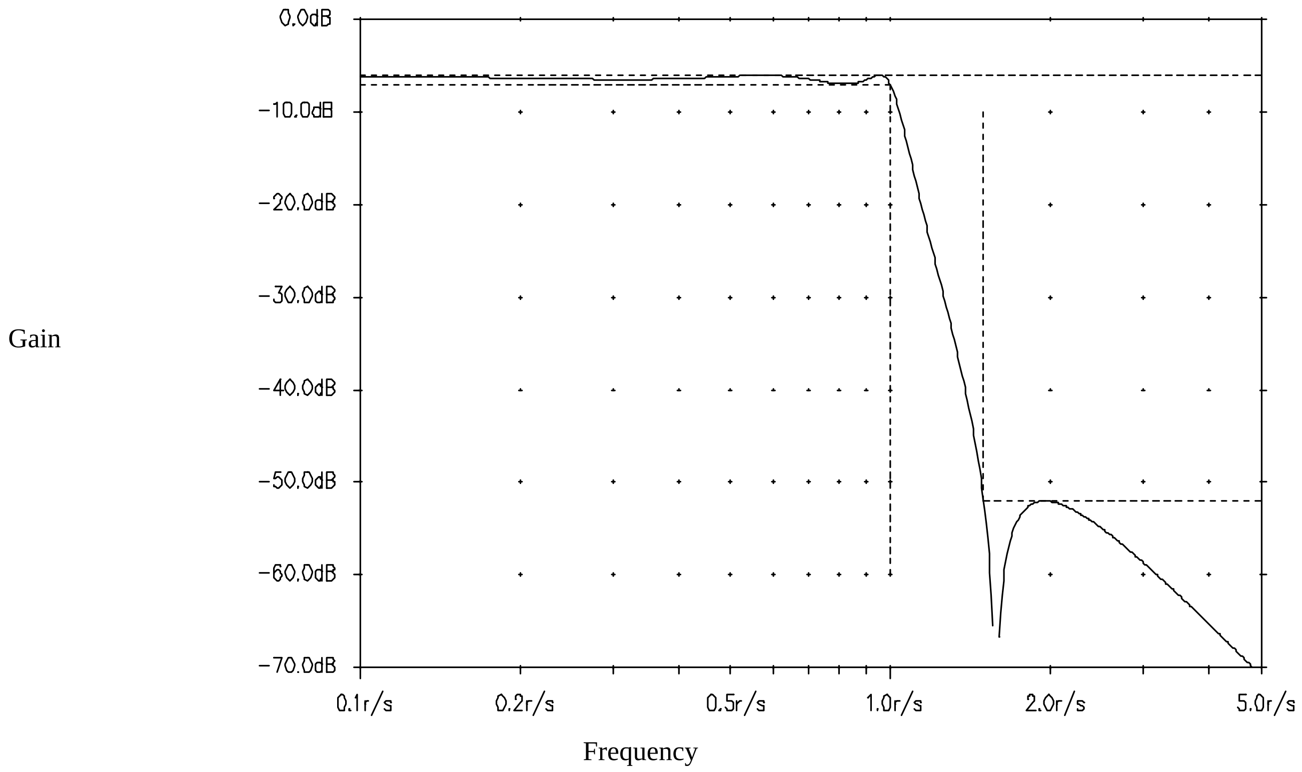


**Figure 8: Frequency response of hybrid-GA designed equally-terminated filter**

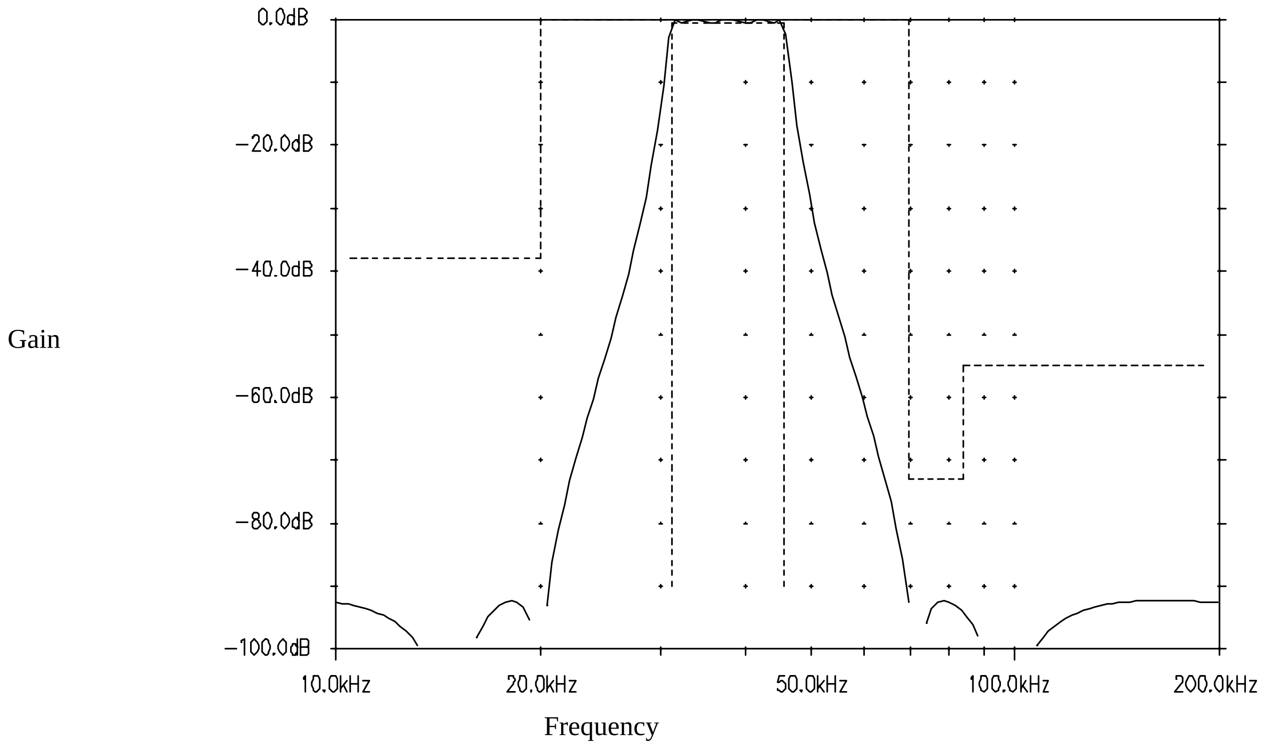


**Figure 9: Frequency response of 10th-order elliptic band-pass filter**

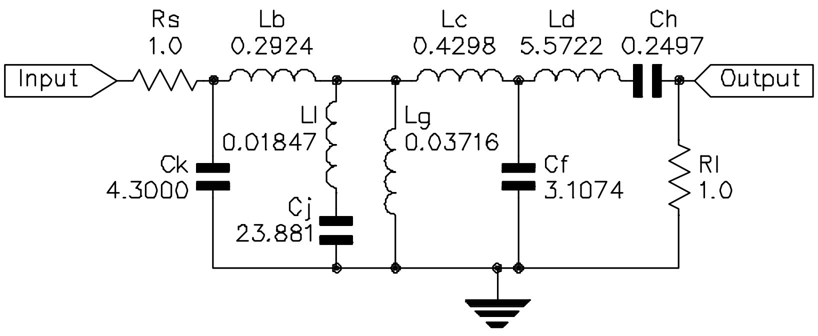


**Figure 10: Hybrid-GA designed Nielsen filter**

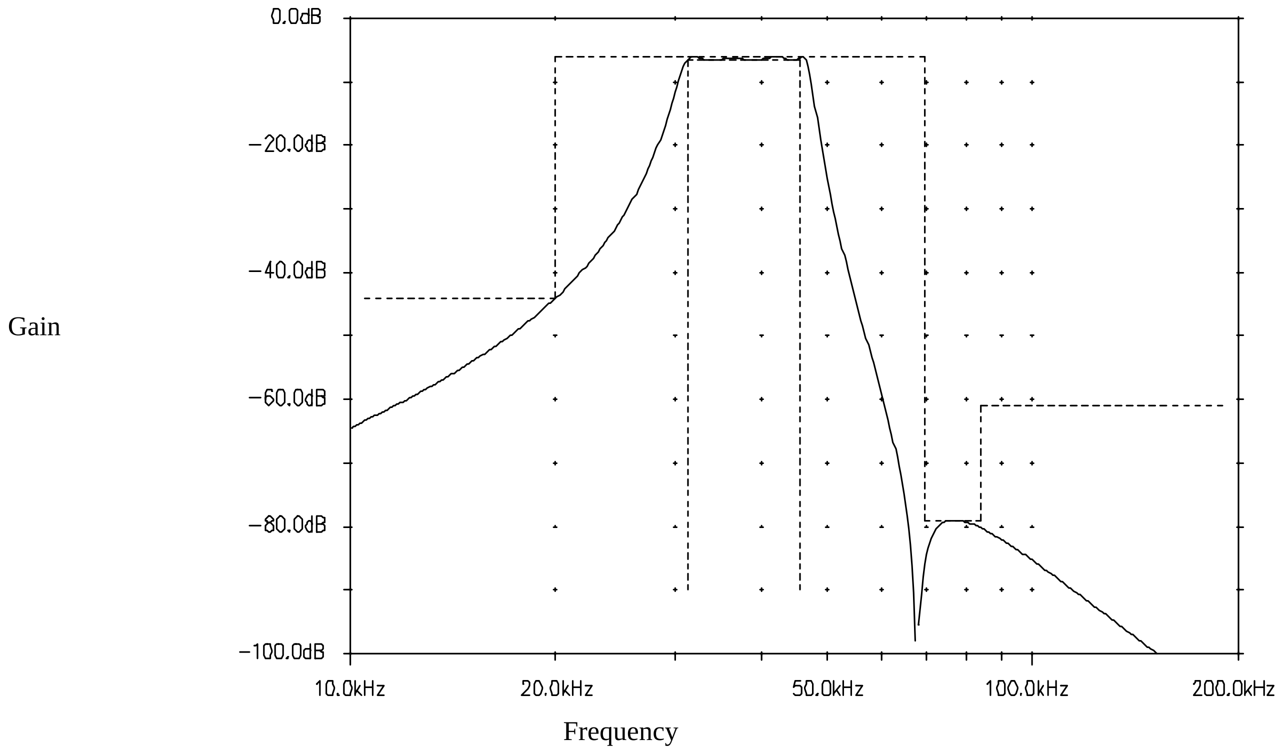


**Figure 11: Frequency response of hybrid-GA designed Nielsen filter**

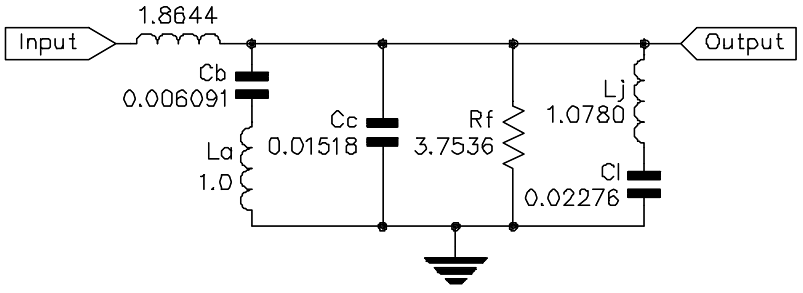


**Figure 12: Hybrid-GA designed averaging filter**

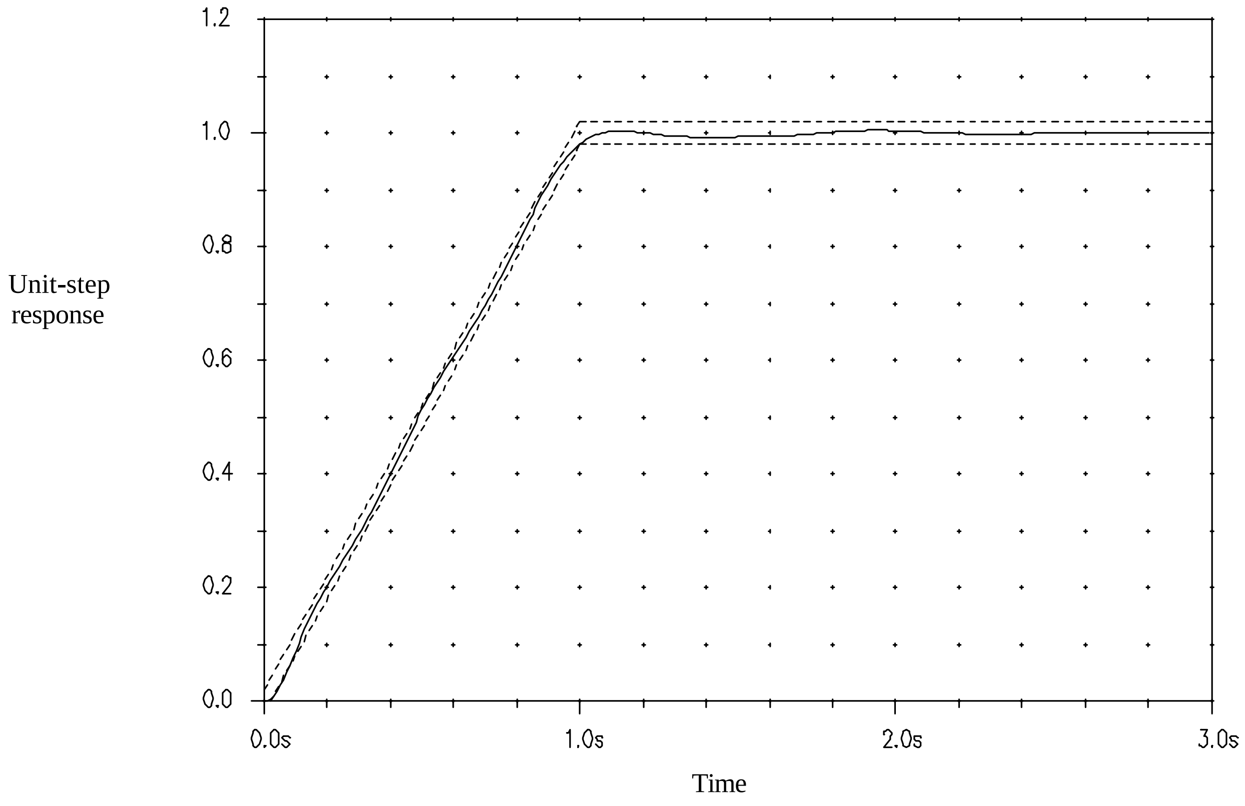


**Figure 13: Unit-step response of hybrid-GA designed averaging filter**