The Design of Analog Circuits by Means of Genetic Programming

John R. Koza
Section on Medical Informatics
Department of Medicine
School of Medicine
Stanford University
Stanford, California 94305
koza@smi.stanford.edu

Forrest H Bennett III
Chief Scientist
Genetic Programming Inc.
Los Altos, California 94023
forrest@evolute.com

David Andre
Division of Computer Science
University of California
Berkeley, California 94720
dandre@cs.berkeley.edu

Martin A. Keane
Chief Scientist
Econometrics Inc.
111 E. Wacker Dr.
Chicago, Illinois 60601
makeane@ix.netcom.com

1. Introduction

The design process entails creation of a complex structure to satisfy user-defined requirements. The design of analog electrical circuits is particularly challenging because it is generally viewed as requiring human intelligence and because it is a major activity of practicing analog electrical engineers.

The design process for analog circuits begins with a high-level description of the circuit's desired behavior and entails creation of both the topology and the sizing of a satisfactory circuit. The topology comprises the gross number of components in the circuit, the type of each component (e.g., a resistor), and a list of all connections between the components. The sizing involves specifying the values (typically numerical) of each of the circuit's components.

Considerable progress has been made in automating the design of certain categories of purely digital circuits; however, the design of analog circuits and mixed analog-digital circuits has not proved as amenable to automation (Rutenbar 1993). Describing "the analog dilemma," Aaserud and Nielsen (1995) noted

> "Analog designers are few and far between. In contrast to digital design, most of the analog circuits are still handcrafted by the experts or so-called 'zahs' of analog design. The design process is characterized by a combination of experience and intuition and requires a thorough knowledge of the process characteristics and the detailed specifications of the actual product.

"Analog circuit design is known to be a knowledge-intensive, multiphase, iterative task, which usually stretches over a significant period of time and is performed by designers with a large portfolio of skills. It is therefore considered by many to be a form of art rather than a science."

There has been extensive previous work on the problem of circuit design (synthesis) using simulated annealing, artificial intelligence, and other techniques as outlined in Koza, Bennett, Andre, Keane, and Dunlap 1997, including work using genetic algorithms (Kruiskamp and Leenaerts 1995; Grimbleby 1995; Thompson 1996). However, there has previously been no general automated technique for synthesizing an analog electrical circuit from a high-level statement of the circuit's desired behavior.

This chapter presents a uniform approach to the automatic design of both the topology and sizing of analog electrical circuits. Section 2 presents design problems involving six prototypical analog circuits. Section 3 describes the method. Section 4 details required preparatory steps. Section 5 shows the results for the six problems. Section 6 cites other circuits that have been designed by genetic programming.

2. Six Problems of Analog Design

This chapter applies genetic programming to an illustrative suite of six problems of analog circuit design. The circuits comprise a variety of types of components, including transistors, diodes, resistors, inductors, and capacitors. The circuits have varying numbers of inputs and outputs.

(1) Design a one-input, one-output lowpass filter composed of capacitors and inductors that passes all frequencies below 1,000 Hz and suppresses all frequencies above 2,000 Hz.

(2) Design a one-input, one-output highpass filter composed of capacitors and inductors that suppresses all frequencies below 1,000 Hz and passes all frequencies above 2,000 Hz.

(3) Design a one-input, one-output tri-state frequency discriminator (source identification) circuit that is composed of resistors, capacitors, and inductors and that produces an output of 1/2 volt and 1 volt for incoming signals whose frequencies are within 10% of 256 Hz and within 10% of 2,560 Hz, respectively, but produces an output of 0 volts otherwise.

(4) Design a one-input, one-output computational circuit that is composed of transistors, diodes, resistors, and capacitors and that produces an output voltage equal to the square root of its input voltage.

(5) Design a two-input, one-output time-optimal robot controller circuit that is composed of the above components and that navigates a constant-speed autonomous mobile robot (with nonzero turning radius) to an arbitrary destination in minimal time.

(6) Design a one-input, one-output amplifier composed of the above components and that delivers amplification of 60 dB (i.e., 1,000 to 1) with low distortion and low bias.

The above six prototypical circuits are representative of analog circuits that are in widespread use. Filters extract specified ranges from frequencies from electrical signals and amplifiers enhance the amplitude of signal. Frequency discriminators are used in source identification and signal recognition. Analog computational circuits are used to perform real-time mathematical calculations on signals. Embedded controllers are used to control the operation of numerous automatic devices.

3. Design by Genetic Programming

The circuits are developed using genetic programming (Koza 1992; Koza and Rice 1992), an extension of the genetic algorithm {Holland 1975) in which the population consists of computer programs. Multipart programs consisting of a main program and one or more reusable, parametrized, hierarchically-called subprograms can be evolved using automatically defined functions (Koza 1994a, 1994b). Architecture-altering operations (Koza 1995) automatically determine the number of such subprograms, the number of arguments that each possesses, and the nature of the hierarchical references, if any, among such automatically defined functions. For current research in genetic programming, see Kinnear 1994, Angeline and Kinnear 1996, Koza, Goldberg, Fogel, and Riolo 1996, Koza et al. 1997, and Banzhaf, Nordin, Keller, and Francone 1998.

A computer program is not a circuit design. Genetic programming can be applied to circuits if a mapping is established between the program trees (rooted, point-labeled trees – that is, acyclic graphs – with ordered branches) used in genetic programming and the labeled cyclic graphs germane to electrical circuits. The principles of developmental biology, the creative work of Kitano (1990) on using genetic algorithms to evolve neural networks, and the innovative work of Gruau (1992) on using genetic programming to evolve neural networks provide the motivation for mapping trees into circuits by means of a growth process that begins with an embryo. For circuits, the embryonic circuit typically includes fixed wires that connect the inputs and outputs of the particular circuit being designed and certain fixed components (such as source and load resistors). Until these wires are modified, the circuit does not produce interesting output. An electrical circuit is developed by progressively applying the functions in a circuit-constructing program tree to the modifiable wires of the embryonic circuit (and, during the developmental process, to new components and modifiable wires).

The functions in the circuit-constructing program trees are divided into four categories: (1) topology-modifying functions that alter the circuit topology, (2) component-creating functions that insert components into the circuit, (3) arithmetic-performing functions that appear in subtrees as argument(s) to the component-creating functions and specify the numerical value of the component, and (4) automatically defined functions that appear in the function-defining branches and potentially enable certain substructures of the circuit to be reused (with parameterization).

Each branch of the program tree is created in accordance with a constrained syntactic structure. Branches are composed of construction-continuing subtrees that continue the developmental process and arithmetic-performing subtrees that determine the numerical value of components. Topology-modifying functions have one or more construction-continuing subtrees, but no arithmetic-performing subtree. Component-creating functions have one or more construction-continuing subtrees and typically have one arithmetic-performing subtree. This constrained syntactic structure is preserved using structure-preserving crossover with point typing (see Koza 1994a).

3.1. The Embryonic Circuit

An electrical circuit is created by executing a circuit-constructing program tree that contains various component-creating and topology-modifying functions. Each tree in the population creates one circuit. The specific embryonic circuit used depends on the number of inputs and outputs.

Figure 16.1 shows a one-input, one-output embryonic circuit in which VSOURCE is the input signal and VOUT is the output signal (the probe point). The circuit is driven by an incoming alternating circuit source VSOURCE. There is a fixed load resistor RLOAD and a fixed source resistor RSOURCE in the embryo. In addition to the fixed components, there is a modifiable wire Z0 between nodes 2 and 3. All development originates from this modifiable wire.

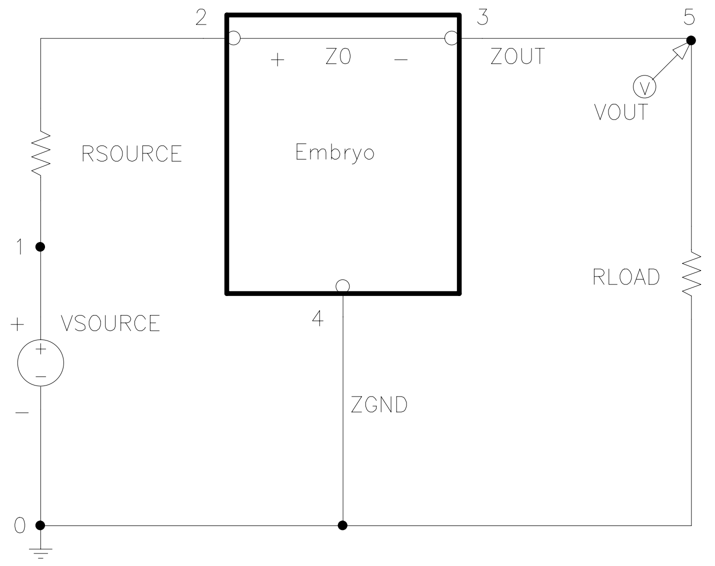

Figure 16.1 One-input, one-output embryonic circuit. 3.2. Component-Creating Functions

Each program tree contains component-creating functions and topology-modifying functions. The component-creating functions insert a component into the developing circuit and assign component value(s) to the component.

Each component-creating functions has a writing head that points to an associated highlighted component in the developing circuit and modifies that component in a specified manner. The construction-continuing subtree of each component-creating functions points to a successor function or terminal in the circuit-constructing program tree.

The arithmetic-performing subtree of a component-creating functions consists of a composition of arithmetic functions (addition and subtraction) and random constants (in the range -1.000 to +1.000). The arithmetic-performing subtree specifies the numerical value of a component by returning a floating-point value that is interpreted on a logarithmic scale as the value for the component in a range of 10 orders of magnitude (using a unit of measure that is appropriate for the particular type of component).

The two-argument resistor-creating R function causes the highlighted component to be changed into a resistor. The value of the resistor in kilo Ohms is specified by its arithmetic-performing subtree.

Figure 16.2 shows a modifiable wire Z0 connecting nodes 1 and 2 of a partial circuit containing four capacitors (C2, C3, C4, and C5). Z0 has a writing head and is subject to subsequent modification. Figure 16.3 shows the result of applying the R function to the modifiable wire Z0 of figure 16.2. The newly created R1 has a writing head so that R1 remains subject to subsequent modification.

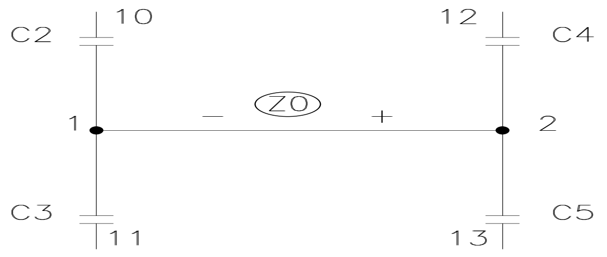

Figure 16.2 Modifiable wire Z0.

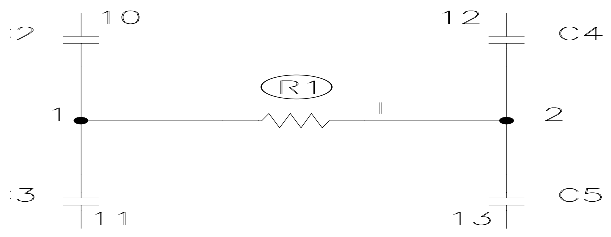

Figure 16.3 Result of applying the R function.

Similarly, the two-argument capacitor-creating C function causes the highlighted component to be changed into a capacitor whose value in micro-Farads is specified by its arithmetic-performing subtree. In addition, the two-argument inductor-creating L function causes the highlighted component to be changed into an inductor whose value in micro-Henrys is specified by its arithmetic-performing subtree.

The one-argument Q_D_PNP diode-creating function causes a diode to be inserted in lieu of the highlighted component. This function has only one argument because there is no numerical value associated with a diode and thus no arithmetic-performing subtree. In practice, the diode is implemented here using a pnp transistor whose collector and base are connected to each other. The Q_D_NPN function inserts a diode using an npn transistor in a similar manner.

There are also six one-argument transistor-creating functions (Q_POS_COLL_NPN, Q_GND_EMIT_NPN, Q_NEG_EMIT_NPN, Q_GND_EMIT_PNP, Q_POS_EMIT_PNP, Q_NEG_COLL_PNP) that insert a bipolar junction transistor in lieu of the highlighted component and that directly connect the collector or emitter of the newly created transistor to a fixed point of the circuit (the positive power supply, ground, or the negative power supply). For example, the Q_POS_COLL_NPN function inserts a bipolar junction transistor whose collector is connected to the positive power supply.

Each of the functions in the family of six different three-argument transistor-creating Q_3_NPN functions causes an npn bipolar junction transistor to be inserted in place of the highlighted component and one of the nodes to which the highlighted component is connected. The Q_3_NPN function creates five new nodes and three modifiable wires. There is no writing head on the new transistor, but there is a writing head on each of the three new modifiable wires. There are 12 members (called `Q_3_NPN0`, ..., `Q_3_NPN11`) in this family of functions because there are two choices of nodes (1 and 2) to be bifurcated and then there are six ways of attaching the transistor's base, collector, and emitter after the bifurcation. Similarly the family of 12 `Q_3_PNP` functions causes a pnp bipolar junction transistor to be inserted.

3.3. Topology-Modifying Functions

Each topology-modifying function in a program tree points to an associated highlighted component and modifies the topology of the developing circuit.

The three-argument `SERIES` division function creates a series composition of the highlighted component (with a writing head), a copy of it (with a writing head), one new modifiable wire (with a writing head), and two new nodes.

The four-argument `PARALLEL0` parallel division function creates a parallel composition consisting of the original highlighted component (with a writing head), a copy of it (with a writing head), two new modifiable wires (each with a writing head), and two new nodes. Figure 16.4 shows the result of applying `PARALLEL0` to the resistor `R1` from figure 16.3.

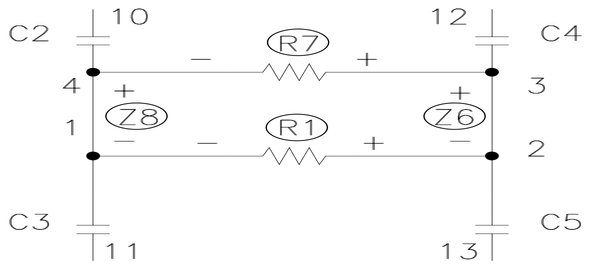

Figure 16.4 Result of the `PARALLEL0` function. The one-argument polarity-reversing `FLIP` function reverses the polarity of the highlighted component.

There are six three-argument functions (`T_GND_0`, `T_GND_1`, `T_POS_0`, `T_POS_1`, `T_NEG_0`, `T_NEG_1`) that insert two new nodes and two new modifiable wires, and then make a connection to ground, positive power supply, or negative power supply, respectively.

There are two three-argument functions (`PAIR_CONNECT_0` and `PAIR_CONNECT_1`) that enable distant parts of a circuit to be connected together. The first `PAIR_CONNECT` to occur in the development of a circuit creates two new wires, two new nodes, and one temporary port. The next `PAIR_CONNECT` creates two new wires and one new node, connects the temporary port to the end of one of these new wires, and then removes the temporary port.

The one-argument NOOP function has no effect on the highlighted component; however, it delays activity on the developmental path on which it appears in relation to other developmental paths in the overall program tree.

The zero-argument END function causes the highlighted component to lose its writing head, thereby ending that particular developmental path.

The zero-argument SAFE_CUT function causes the highlighted component to be removed from the circuit provided that the degree of the nodes at both ends of the highlighted component is three (i.e., no dangling components or wires are created).

An electrical circuit is created by executing the functions in a circuit-constructing program tree. The functions are progressively applied in a developmental process to the embryonic circuit and its successors until all of the functions in the program tree are executed. That is, the functions in the circuit-constructing program tree progressively side-effect the embryonic circuit and its successors until a fully developed circuit eventually emerges. The functions are applied in a breadth-first order.

Figure 16.5 is an illustrative circuit-constructing program tree shown as a rooted, point-labeled tree with ordered branches. The overall program consists of two main result-producing branches joined by a connective LIST function (labeled 1). The first (left) result-producing branch is rooted at the capacitor-creating C function (labeled 2). The second result-producing branch is rooted at the polarity-reversing FLIP function (labeled 3). This figure also contains four occurrences of the inductor-creating L function (at 17, 11, 20, and 12). The figure contains two occurrences of the topology-modifying SERIES function (at 5 and 10). The figure also contains five occurrences of the development-controlling END function (at 15, 25, 27, 31, and 22) and one occurrence of the development-controlling "no operation" NOP function (at 6). There is a seven-point arithmetic-performing subtree at 4 under the capacitor-creating C function at 4. Similarly, there is a three-point arithmetic-performing subtree at 19 under the inductor-creating L function at 11, and one-point arithmetic-performing subtrees (i.e., constants) at 26, 30, and 21. Additional details can be found in Koza, Bennett, Andre, and Keane (1999).

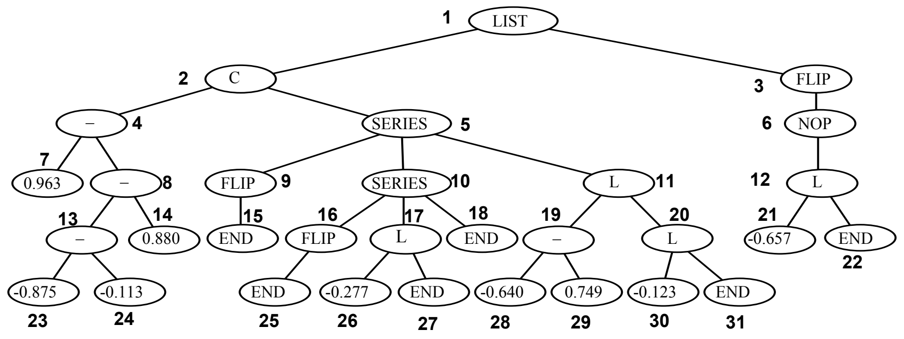

Figure 16.5 Illustrative circuit-constructing program tree.

4. Preparatory Steps

Before applying genetic programming to a problem of circuit design, seven major preparatory steps are required: (1) identify the suitable embryonic circuit, (2) determine the architecture of the overall circuit-constructing program trees, (3) identify the terminals of the program trees, (4) identify the primitive functions of the program trees, (5) create the fitness measure, (6) choose parameters, and (7) determine the termination criterion and method of result designation.

4.1. Embryonic Circuit

The embryonic circuit used on a particular problem depends on the circuit's number of inputs and outputs.

For example, an embryonic circuit with two modifiable wires (Z0 and Z1) was used for the lowpass and highpass filters (which each have one input and one output). However, the robot controller circuit has two inputs (VSOURCE1 and VSOURCE2), not just one. Each input needs its own separate source resistor (RSOURCE1 and RSOURCE2). Thus, the embryonic circuit for the robot controller circuit has three modifiable wires (Z0, Z1, and Z2) in order to provide full connectivity between the two inputs and the one output.

All development originates from the modifiable wires.

In some problems, such as the amplifier, the embryonic circuit contains additional fixed components (as described in Koza, Bennett, Andre, and Keane 1997).

4.2. Program Architecture

Since there is one result-producing branch in the program tree for each modifiable wire in the embryo, the architecture of each circuit-constructing program tree depends on the embryonic circuit. One result-producing branch was used for the frequency discriminator and the computational circuit; two were used for lowpass and highpass filter problems; and three were used for the robot controller and amplifier.

The architecture of each circuit-constructing program tree also depends on the use, if any, of automatically defined functions. Automatically defined functions provide a mechanism enabling certain substructures to be reused and are described in detail in Koza, Bennett, Andre, and Keane 1999. Automatically defined functions and architecture-altering operations were used in the frequency discriminator, robot controller, and amplifier. For these problems, each program in the initial population of programs had a uniform architecture with no automatically defined functions. In later generations, the number of automatically defined functions, if any, emerged as a consequence of the architecture-altering operations (also described in Koza, Bennett, Andre, and Keane 1999).

4.3. Function and Terminal Sets

The function set for each design problem depended on the type of electrical components that were used to construct the circuit. Inductors and capacitors were used for the lowpass and highpass filter problems. Capacitors, diodes, and transistors were used for the computational circuit, the robot controller, and the amplifier. Resistors (in addition to inductors and capacitors) were used for the frequency discriminator. When transistors were used, functions to provide connectivity to the positive and negative power supplies were also included.

For the computational circuit, the robot controller, and the amplifier, the function set, $\mathcal{F}_{\text{ccs-initial}}$, for each construction-continuing subtree was

$\mathcal{F}_{\text{ccs-initial}}$ = {`R`, `C`, `SERIES`, `PARALLEL0`, `PARALLEL1`, `FLIP`, `NOOP`, `T_GND_0`, `T_GND_1`, `T_POS_0`, `T_POS_1`, `T_NEG_0`, `T_NEG_1`, `PAIR_CONNECT_0`, `PAIR_CONNECT_1`, `Q_D_NPN`, `Q_D_PNP`, `Q_3_NPN0`, ..., `Q_3_NPN11`, `Q_3_PNP0`, ..., `Q_3_PNP11`, `Q_POS_COLL_NPN`, `Q_GND_EMIT_NPN`, `Q_NEG_EMIT_NPN`, `Q_GND_EMIT_PNP`, `Q_POS_EMIT_PNP`, `Q_NEG_COLL_PNP`}.

For the *npn* transistors, the `Q2N3904` model was used. For *pnp* transistors, the `Q2N3906` model was used.

The initial terminal set, $\mathcal{T}_{\text{ccs-initial}}$, for each construction-continuing subtree was

$\mathcal{T}_{\text{ccs-initial}}$ = {`END`, `SAFE_CUT`}.

The initial terminal set, $\mathcal{T}_{\text{aps-initial}}$, for each arithmetic-performing subtree consisted of

$\mathcal{T}_{\text{aps-initial}} = \{\Re\}$,

where $\Re$ represents floating-point random constants from –1.0 to +1.0.

The function set, $\mathcal{F}_{\text{aps}}$, for each arithmetic-performing subtree was,

$\mathcal{F}_{\text{aps}} = \{$`+`, `-`$\}$.

The terminal and function sets were identical for all result-producing branches for a particular problem.

For the lowpass filter, highpass filter, and frequency discriminator, there was no need for functions to provide connectivity to the positive and negative power supplies.

For the frequency discriminator, the robot controller, and the amplifier, the architecture-altering operations were used and the set of potential new functions, $\mathcal{F}_{\text{potential}}$, was

$\mathcal{F}_{\text{potential}} = \{$`ADF0`, `ADF1`, ...$\}$.

The set of potential new terminals, $\mathcal{T}_{\text{potential}}$, for the automatically defined functions was

$\mathcal{T}_{\text{potential}} = \{$`ARG0`$\}$.

The architecture-altering operations change the function sets and terminal sets for each construction-continuing subtree of all three result-producing branches and the function-defining branches.

4.4. Fitness Measure

The fitness measure varies for each problem. The high-level statement of desired circuit behavior is translated into a well-defined measurable quantity that can be used by genetic programming to guide the evolutionary process. The evaluation of each individual circuit-constructing program tree in the population begins with its execution. This execution progressively applies the functions in each program tree to an embryonic circuit, thereby creating a fully developed circuit. A netlist is created that identifies each component of the developed circuit, the nodes to which each component is connected, and the value of each component. The netlist becomes the input to our modified version of the 217,000-line SPICE (Simulation Program with Integrated Circuit Emphasis) simulation program (Quarles, Newton, Pederson, and Sangiovanni-Vincentelli 1994). SPICE then determines the behavior of the circuit. It was necessary to make considerable modifications in SPICE so that it could run as a submodule within the genetic programming system.

4.4.1. Fitness Measure for the Lowpass Filter

A simple *filter* is a one-input, one-output electronic circuit that receives a signal as its input and passes the frequency components of the incoming signal that lie in a specified range (called the *passband*) while suppressing the frequency components that lie in all other frequency ranges (the *stopband*).

The desired lowpass LC filter has a passband below 1,000 Hz and a stopband above 2,000 Hz. The circuit is driven by an incoming AC voltage source with a 2 volt amplitude.

The *attenuation* of the filter is defined in terms of the output signal relative to the reference voltage (one volt here). A *decibel* is a unitless measure of relative voltage that is defined as 20 times the common (base 10) logarithm of the ratio between the voltage at a particular probe point and a reference voltage.

In this problem, a voltage in the passband of exactly 1 volt and a voltage in the stopband of exactly 0 volts is regarded as ideal. The (preferably small) variation within the passband is called the *passband ripple*. Similarly, the incoming signal is never fully reduced to zero in the stopband of an actual filer. The (preferably small) variation within the stopband is called the *stopband ripple*. A voltage in the passband of between 970 millivolts and 1 volt (i.e., a passband ripple of 30 millivolts or less) and a voltage in the stopband of between 0 volts and 1 millivolts (i.e., a stopband ripple of 1 millivolts or less) is regarded as acceptable. Any voltage lower than 970 millivolts in the passband and any voltage above 1 millivolts in the stopband is regarded as unacceptable.

A fifth-order *elliptic* (*Cauer*) *filter* with a modular angle Θ of 30 degrees (i.e., the arcsin of the ratio of the boundaries of the passband and stopband) and a reflection coefficient ρ of 24.3% is required to satisfy these design goals (Williams and Taylor 1995).

Since the high-level statement of behavior for the desired circuit is expressed in terms of frequencies, the voltage VOUT is measured in the frequency domain. SPICE performs an AC small signal analysis and report the circuit's behavior over five decades (between 1 Hz and 100,000 Hz) with each decade being divided into 20 parts (using a logarithmic scale), so that there are a total of 101 fitness cases.

Fitness is measured in terms of the sum over these cases of the absolute weighted deviation between the actual value of the voltage that is produced by the circuit at the probe point VOUT and the target value for voltage. The smaller the value of fitness, the better. A fitness of zero represents an (unattainable) ideal filter.

Specifically, the standardized fitness is

$$F(t) = \sum_{i=0}^{100} \left(W(d(f_i), f_i) d(f_i) \right)$$

where f_i is the frequency of fitness case i; $d(x)$ is the absolute value of the difference between the target and observed values at frequency x; and $W(y, x)$ is the weighting for difference y at frequency x.

The fitness measure is designed to not penalize ideal values, to slightly penalize every acceptable deviation, and to heavily penalize every unacceptable deviation. Specifically, the procedure for each of the 61 points in the 3-decade interval between 1 Hz and 1,000 Hz for the intended passband is as follows:

- If the voltage equals the ideal value of 1.0 volt in this interval, the deviation is 0.0.
- If the voltage is between 970 millivolts and 1 volt, the absolute value of the deviation from 1 volt is weighted by a factor of 1.0.
- If the voltage is less than 970 millivolts, the absolute value of the deviation from 1 volt is weighted by a factor of 10.0.

The acceptable and unacceptable deviations for each of the 35 points from 2,000 Hz to 100,000 Hz in the intended stopband are similarly weighed (by 1.0 or 10.0) based on the amount of deviation from the ideal voltage of 0 volts and the acceptable deviation of 1 millivolts.

For each of the five "don't care" points between 1,000 and 2,000 Hz, the deviation is deemed to be zero.

The number of “hits” for this problem (and all other problems herein) is defined as the number of fitness cases for which the voltage is acceptable or ideal or that lie in the "don't care" band (for a filter).

Many of the random initial circuits and many that are created by the crossover and mutation operations in subsequent generations cannot be simulated by SPICE. These circuits receive a high penalty value of fitness (10^8) and become the worst-of-generation programs for each generation.

For details, see Koza, Bennett, Andre, and Keane 1996b.

4.4.2. Fitness Measure for the Highpass Filter

The fitness cases for the highpass filter are the same 101 points in the five decades of frequency between 1 Hz and 100,000 Hz as for the lowpass filter. The fitness measure is substantially the same as that for the lowpass filter problem above, except that the locations of the passband and stopband are reversed.

4.4.3. Fitness Measure for the Tri-state Frequency Discriminator

Fitness is the sum, over 101 fitness cases, of the absolute weighted deviation between the actual value of the voltage that is produced by the circuit and the target value.

The three points that are closest to the band located within 10% of 256 Hz are 229.1 Hz, 251.2 Hz, and 275.4 Hz. The procedure for each of these three points is as follows: If the voltage equals the ideal value of 1/2 volts in this interval, the deviation is 0.0. If the voltage is more than 240 millivolts from 1/2 volts, the absolute value of the deviation from 1/2 volts is weighted by a factor of 20. If the voltage is more than 240 millivolts from 1/2 volts, the absolute value of the deviation from 1/2 volts is weighted by a factor of 200. This arrangement reflects the fact that the ideal output voltage for this range of frequencies is 1/2 volts, the fact that a 240-millivolt discrepancy is acceptable, and the fact that a larger discrepancy is not acceptable.

Similar weighting was used for the three points (2,291 Hz, 2,512 Hz, and 2,754 Hz) that are closest to the band located within 10% of 2,560 ,Hz.

The procedure for each of the remaining 95 points is as follows: If the voltage equals the ideal value of 0 volts, the deviation is 0.0. If the voltage is within 240 millivolts of 0 volts, the absolute value of the deviation from 0 volts is weighted by a factor of 1.0. If the voltage is more than 240 millivolts from 0 volts, the absolute value of the deviation from 0 volts is weighted by a factor of 10. For details, see Koza, Bennett, Lohn, Dunlap, Andre, and Keane 1997b.

4.4.4. Fitness Measure for the Computational Circuit

SPICE is called to perform a DC sweep analysis at 21 equidistant voltages between –250 millivolts and +250 millivolts. Fitness is the sum, over these 21 fitness cases, of the absolute weighted deviation between the actual value of the voltage that is produced by the circuit and the target value for voltage. For details, see Koza, Bennett, Lohn, Dunlap, Andre, and Keane 1997a.

4.4.5. Fitness Measure for the Robot Controller Circuit

The fitness of a robot controller was evaluated using 72 randomly chosen fitness cases each representing a different target point. Fitness is the sum, over the 72 fitness cases, of the travel times. If the robot came within a capture radius of 0.28 meters of its target point before the end of the 80 time steps allowed for a particular fitness case, the contribution to fitness for that fitness case was the actual time. However, if the robot failed to come within the capture radius during the 80 time steps, the contribution to fitness was 0.160 hours (i.e., double the worst possible time).

SPICE performs a nested DC sweep, which provides a way to simulate the DC behavior of a circuit with two inputs. It resembles a nested pair of `FOR` loops in a computer program in that both of the loops have a starting value for the voltage, an increment, and an ending value for the voltage. For each voltage value in the outer loop, the inner loop simulates the behavior of the circuit by stepping through its range of voltages. Specifically, the starting value for voltage is –4 volts, the step size is 0.2 volts, and the ending value is +4 volts. These values correspond to the dimensions of the robot's world of 64 square meters extending 4 meters in each of the four directions from the origin of a coordinate system (i.e., 1 volt equals 1 meter). For details, see Koza, Bennett, Keane, and Andre 1997.

4.4.6. Fitness Measure for the 60 dB Amplifier

SPICE was requested to perform a DC sweep analysis to determine the circuit's response for several different DC input voltages. An ideal inverting amplifier circuit would receive the DC input, invert it, and multiply it by the amplification factor. A circuit is flawed to the extent that it does not achieve the desired amplification, the output signal is not perfectly centered on 0 volts(i.e., it is biased), or the DC response is not linear. Fitness is calculated by summing an amplification penalty, a bias penalty, and two non-linearity penalties – each derived from these five DC outputs. For details, see Bennett, Koza, Andre, and Keane 1996.

4.5. Control Parameters

The population size, *M*, was 640,000 for all problems. Other parameters were substantially the same for each of the six problems and can be found in the references cited above.

4.6. Implementation on Parallel Computer

Each problem was run on a medium-grained parallel Parsytec computer system (Andre and Koza 1996) consisting of 64 80-MHz PowerPC 601 processors arranged in an 8 by 8 toroidal mesh with a host PC Pentium type computer. The distributed genetic algorithm was used with a population size of $Q = 10{,}000$ at each of the $D = 64$ demes (semi-isolated subpopulations). On each generation, four boatloads of emigrants, each consisting of $B = 2\%$ (the migration rate) of the node's subpopulation (selected on the basis of fitness) were dispatched to each of the four adjacent processing nodes.

5. Results

In all six problems, fitness was observed to improve over successive generations. A large majority of the randomly created initial circuits of generation 0 were not able to be simulated by SPICE; however, most were simulatable after only a few generations. Satisfactory results were generated in every case on the first or second trial. When two runs were required, the first produced an almost satisfactory result. This rate of success suggests that the capabilities of the approach and current computing system have not been fully exploited.

5.1. Lowpass Filter

Many of the runs produced lowpass filters having a topology similar to that employed by human engineers. For example, in generation 32 of one run, a circuit (figure 16.6) was evolved with a near-zero fitness of 0.00781. The circuit was 100% compliant with the design requirements in that it scored 101 hits (out of 101). After the evolutionary run, this circuit (and all evolved circuits herein) were simulated anew using the commercially available MicroSim circuit simulator to verify performance. As can be seen, inductors appear in series horizontally across the top of the figure, while capacitors appear vertically as shunts to ground. This circuit had the recognizable ladder topology of a Butterworth or Chebychev lowpass filter (Williams and Taylor 1995).

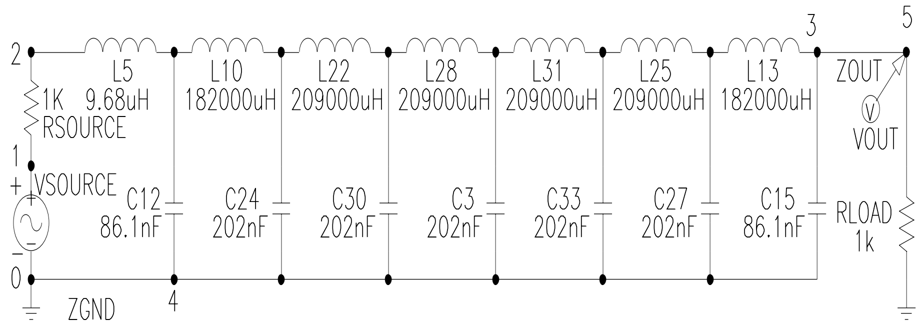

Figure 16.6 Evolved 7-rung ladder lowpass filter. Figure 16.7 shows the behavior in the frequency domain of this evolved lowpass filter. As can be seen, the evolved circuit delivers about 1 volt for all frequencies up to 1,000 Hz and about 0 volts for all frequencies above 2,000 Hz.

In another run, a 100% compliant recognizable "bridged T" arrangement was evolved. In yet another run using automatically defined functions, a 100% compliant circuit emerged with the recognizable elliptic topology that was invented and patented by Cauer. When invented. the Cauer filter was a significant advance (both theoretically and commercially) over the Butterworth and Chebychev filters.

Thus, genetic programming rediscovered the ladder topology of the Butterworth and Chebychev filters, the "bridged T" topology, and the elliptic topology.

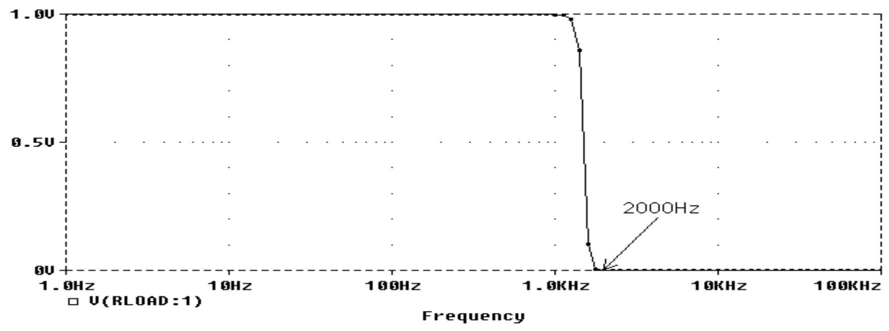

Figure 16.7 Frequency domain behavior of genetically evolved 7-rung ladder lowpass filter.

It is important to note that when we performed the preparatory steps for applying genetic programming to the problem of synthesizing a lowpass filter, we did not employ any significant domain knowledge from the field of electrical engineering. We did not incorporate knowledge of Kirchhoff's laws, integro-differential equations, Laplace transforms, poles, zeroes, or the other mathematical techniques and insights about filters that are known to electrical engineers who design analog filters. In spite of this absence of explicit domain knowledge, genetic programming evolved a 100% compliant circuit for the problem of designing an LC lowpass filter that embodied the ladder topology that is well known in the field of electrical engineering.

The reinvention by genetic programming of the recognizable ladder topology of Butterworth or Chebychev lowpass filters is an instance where genetic programming has produced a result that is competitive with those created by inventive and knowledgeable humans. It satisfies Arthur Samuel's criterion (1983) for artificial intelligence and machine learning, namely

> The aim [is] ... to get machines to exhibit behavior which if done by humans would be assumed to involve the use of intelligence.

5.2. Highpass Filter

In generation 27 of one run, a 100% compliant circuit (figure 16.8) was evolved with a near-zero fitness of 0.213. This circuit has four capacitors and five inductors (in addition to the fixed components of the embryo). As can be seen, capacitors appear in series horizontally across the top of the figure, while inductors appear vertically as shunts to ground.

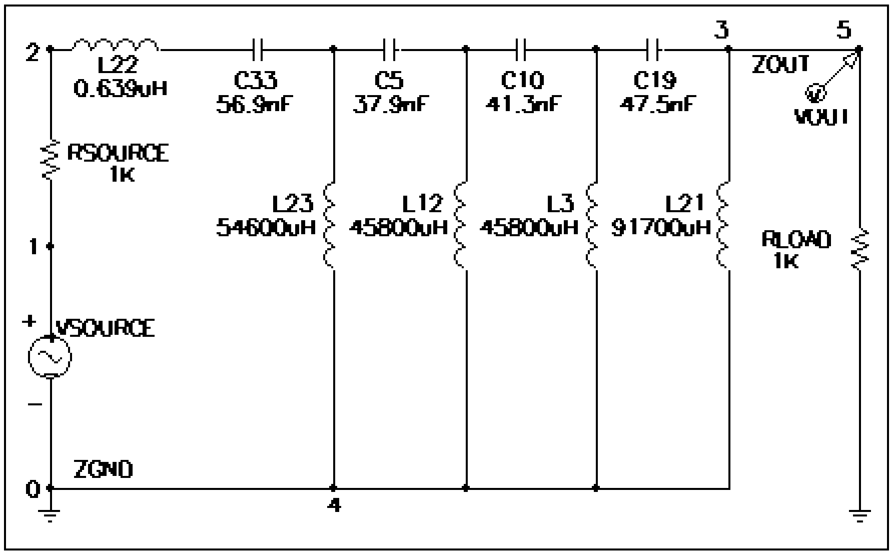

Figure 16.8 Evolved four-rung ladder highpass filter. Figure 16.9 shows the behavior in the frequency domain of this evolved highpass filter. As desired, the evolved highpass delivers about 0 volts for all frequencies up to 1,000 Hz and about 1 volt for all frequencies above 2,000 Hz.

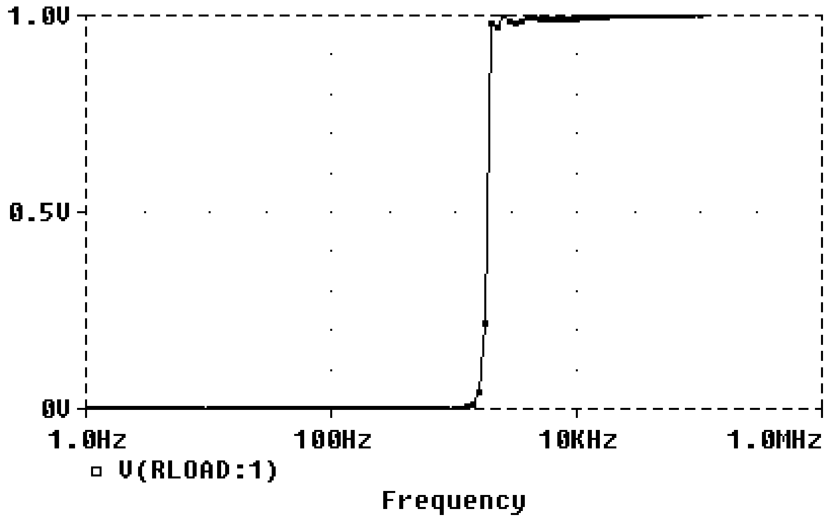

Figure 16.9 Frequency domain behavior of evolved four-rung ladder highpass filter. The reversal of roles for the capacitors and inductors in lowpass and highpass ladder filters is well known to electrical engineers. It arises because of the duality of the single terms (derivatives versus integrals) in the integro-differential equations that represent the voltages and currents of the inductors and capacitors in the loops and nodes of a circuit. However, genetic programming was not given any domain knowledge concerning this duality. In fact, the fitness measure was the only difference in the preparatory steps for the problem of synthesizing the highpass filter versus the problem of synthesizing the lowpass filter. In spite of the absence of explicit domain knowledge about electrical engineering in general or duality in particular,

genetic programming evolved a 100% compliant highpass filter embodying the well-known highpass ladder topology. Using the fitness measure appropriate for highpass filters, genetic programming searched the same space (i.e., the space of circuit-constructing program trees composed of the same component-creating functions and the same topology-modifying functions) and discovered circuit-constructing program tree that yielded a 100%-complaint highpass filter.

The rediscovery by genetic programming of the reversal of roles of capacitors and inductors in lowpass and filters is an instance where genetic programming has produced a result that is competitive with those created by inventive and knowledgeable humans. This result (and all of the succeeding results in this chapter) satisfies Arthur Samuel's criterion (1983) for success in artificial intelligence and machine learning.

5.3. Tri-state Frequency Discriminator

The evolved three-way tri-state frequency discriminator circuit from generation 106 scores 101 hits (out of 101). Figure 16.10 shows this circuit (after expansion of its automatically defined functions). The circuit produces the desired outputs of 1 volt and 1/2 volts (each within the allowable tolerance) for the two specified bands of frequencies and the desired near-zero signal for all other frequencies.

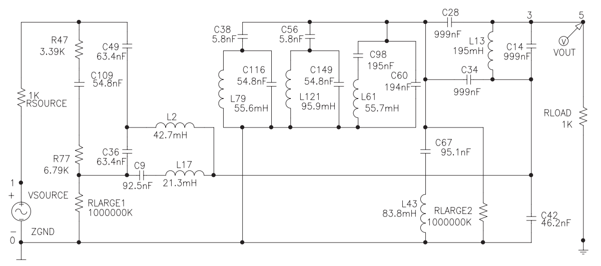

Figure 16.10 Evolved frequency discriminator.

5.4. Computational Circuit

The genetically evolved computational circuit for the square root from generation 60 (figure 16.11), achieves a fitness of 1.68, and has 36 transistors, two diodes, no capacitors, and 12 resistors (in addition to the source and load resistors in the embryo). The output voltages produced by this best-of-run circuit are almost exactly the required values.

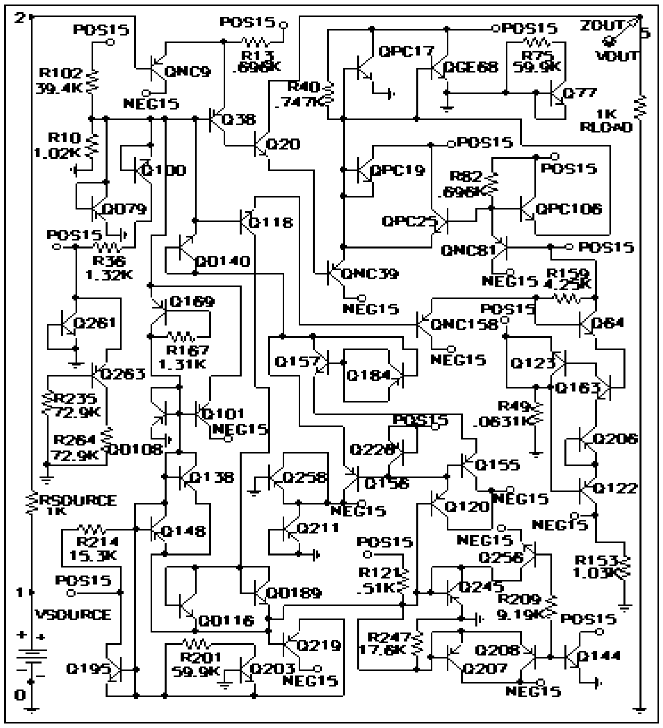

Figure 16.11 Evolved square root circuit.

5.5. Robot Controller Circuit

The best-of-run time-optimal robot controller circuit (figure 16.12) appeared in generation 31, scores 72 hits, and achieves a near-optimal fitness of 1.541 hours. In comparison, the optimal value of fitness for this problem is known to be 1.518 hours. This best-of-run circuit has 10 transistors and 4 resistors. The program has one automatically defined function that is called twice (incorporated into the figure).

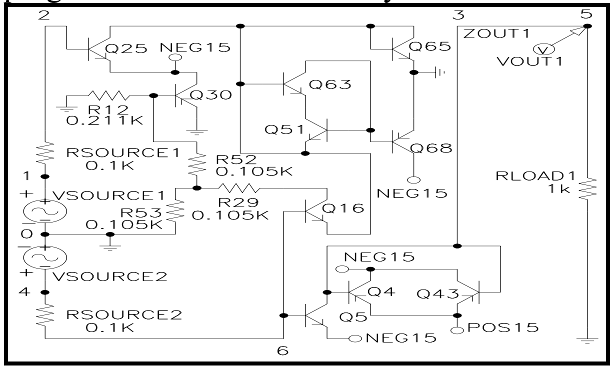

Figure 16.12 Evolved robot controller. This problem entails navigating a robot to a destination in minimum time, so its fitness measure (section 4.4.5) is expressed in terms of elapsed time. The fitness measure is a high-level description of "what needs to be done" – namely, get the robot to the destination in a time-optimal way. However, the fitness measure does not specify "how to do it." In particular, the fitness measure conveys no hint about the critical (and counterintuitive) tactic needed to minimize elapsed

time in time-optimal control problem – namely, that it is sometimes necessary to veer away from the destination in order to reach it in minimal time. Nonetheless, the evolved time-optimal robot controller embodies this counterintuitive tactic. For example, figure 16.13 shows the trajectory for the fitness case where the destination is (0.409, –0.892). Correct time-optimal handling of this difficult destination point requires a trajectory that begins by veering away from the destination (thereby increasing the distance to the destination) followed by a circular trajectory to the destination. The small circle in the figure represents the capture radius of 0.28 meters around the destination point.

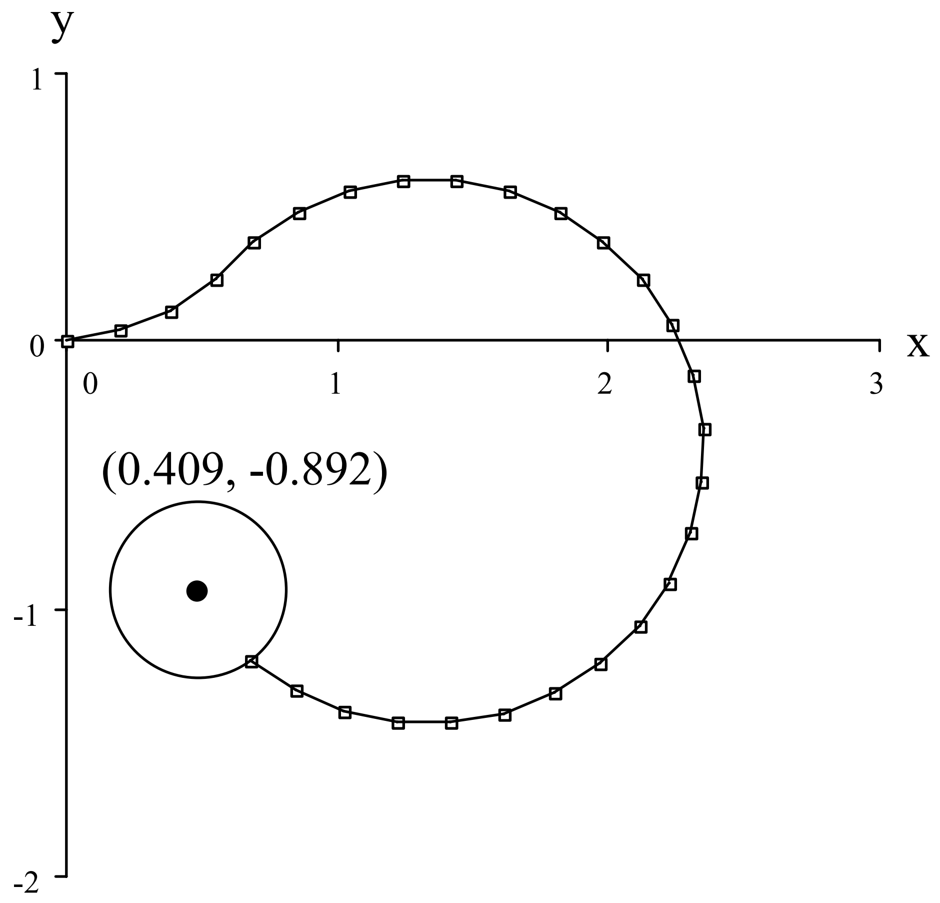

Figure 16.13 Evolved time-optimal trajectory to destination point (0.409, –0.892).

The evolved time-optimal robot controller generalizes so as to correctly handle all other possible destinations in the plane.

5.6. 60 dB Amplifier

The best circuit from generation 109 (figure 16.14) achieves a fitness of 0.178. Based on a DC sweep, the amplification is 60 dB here (i.e., 1,000-to-1 ratio) and the bias is 0.2 volts. Based on a transient analysis at 1,000 Hz, the amplification is 59.7 dB; the bias is 0.18 volts; and the total harmonic distortion is very low

(0.17%). Based on an AC sweep, the amplification at 1,000 Hz is 59.7 dB; the flatband gain is 60 dB; and the 3 dB bandwidth is 79, 333 Hz. Thus, a high-gain amplifier with low distortion and acceptable bias has been evolved.

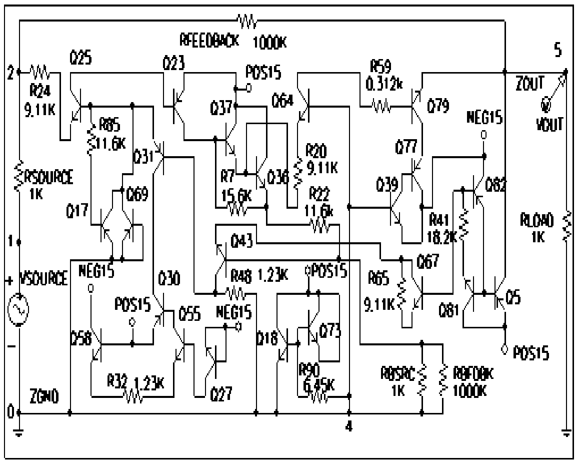

Figure 16.14 Genetically evolved amplifier.

6. Other Circuits

Numerous other circuits have been similarly designed, including asymmetric bandpass filters (Koza, Bennett, Andre, and Keane 1996c), crossover filters (Koza, Bennett, Andre, and Keane 1996a), double passband filters (Koza, Andre, Bennett, and Keane 1996), amplifiers (Koza, Bennett, Andre, and Keane 1997), a temperature-sensing circuit, and a voltage reference circuit (Koza, Bennett, Andre, Keane, and Dunlap 1997).

7. Conclusion

In this chapter, genetic programming succeeded in evolving both the topology and sizing of six different prototypical analog electrical circuits, including a lowpass filter, a highpass filter, a tri-state frequency discriminator circuit, a 60 dB amplifier, a computational circuit for the square root, and a time-optimal robot controller circuit. All six of these genetically evolved circuits constitute instances of an evolutionary computation technique solving a problem that is usually thought to require human intelligence.

There has previously been no general automated technique for synthesizing an analog electrical circuit from a high-level statement of the circuit's desired behavior. The approach using genetic programming to the problem of analog circuit synthesis is general; it can be directly applied to other problems of analog circuit synthesis.

In fact, this same approach is even more general and can be applied to the problem of automatic programming (i.e., the challenge of getting a computer to solve a problem without explicitly programming it). Paraphrasing Arthur Samuel (1959), the challenge of automatic programming concerns

> How can computers be made to do what needs to be done, without being told exactly how to do it?

Each of the problems in this chapter illustrates the automatic creation of a satisfactory way of "how to do it" from a high-level statement of "what needs to be done."

References

Aaserud, O. and Nielsen, I. Ring. 1995. Trends in current analog design: A panel debate. *Analog Integrated Circuits and Signal Processing*. 7(1) 5-9.

Andre, David and Koza, John R. 1996. Parallel genetic programming: A scalable implementation using the transputer architecture. In Angeline, P. J. and Kinnear, K. E. Jr. (editors). 1996. *Advances in Genetic Programming 2*. Cambridge: MIT Press.

Angeline, Peter J. and Kinnear, Kenneth E. Jr. (editors). 1996. *Advances in Genetic Programming 2*. Cambridge, MA: The MIT Press.

Banzhaf, Wolfgang, Nordin, Peter, Keller, Robert E., and Francone, Frank D. 1998. *Genetic Programming – An Introduction*. San Francisco, CA: Morgan Kaufmann and Heidelberg: dpunkt.

Bennett III, Forrest H, Koza, John R., Andre, David, and Keane, Martin A. 1996. Evolution of a 60 Decibel op amp using genetic programming. In Higuchi, Tetsuya, Iwata, Masaya, and Lui, Weixin (editors). *Proceedings of International Conference on Evolvable Systems: From Biology to Hardware (ICES-96)*. Lecture Notes in Computer Science, Volume 1259. Berlin: Springer-Verlag. Pages 455-469.

Grimbleby, J. B. 1995. Automatic analogue network synthesis using genetic algorithms. *Proceedings of the First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications.* London: Institution of Electrical Engineers. Pages 53–58.

Gruau, Frederic. 1992. *Cellular Encoding of Genetic Neural Networks*. Technical report 92-21. Laboratoire de l'Informatique du Parallélisme. Ecole Normale Supérieure de Lyon. May 1992.

Holland, John H. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.

Kinnear, Kenneth E. Jr. (editor). 1994. *Advances in Genetic Programming*. Cambridge, MA: The MIT Press.

Kitano, Hiroaki. 1990. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*. 4(1990) 461–476.

Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: MIT Press.

Koza, John R. 1994a. *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge, MA: MIT Press.

Koza, John R. 1994b. *Genetic Programming II Videotape: The Next Generation*. Cambridge, MA: MIT Press.

Koza, John R. 1995. Evolving the architecture of a multi-part program in genetic programming using architecture-altering operations. In McDonnell, John R., Reynolds, Robert G., and Fogel, David B. (editors). 1995. *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*. Cambridge, MA: The MIT Press. Pages 695–717.

Koza, John R., Andre, David, Bennett III, Forrest H, and Keane, Martin A. 1996. Use of automatically defined functions and architecture-altering operations in automated circuit synthesis using genetic programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). 1996. *Genetic Programming 1996: Proceedings of the First Annual Conference*. Cambridge, MA: The MIT Press.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1999. *Genetic Programming III*: *Darwinian Invention and Problem Solving*. San Francisco, CA: Morgan Kaufmann.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996a. 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*. IEEE Press. Pages 1–10.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996b. Automated design of both the topology and sizing of analog electrical circuits using genetic programming. In Gero, John S. and Sudweeks, Fay (editors). *Artificial Intelligence in Design '96*. Dordrecht: Kluwer. Pages 151-170.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996c. Automated WYWIWYG design of both the topology and component values of analog electrical circuits using genetic programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). 1996. *Genetic Programming 1996: Proceedings of the First Annual Conference*. Cambridge, MA: The MIT Press.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1997. Evolution using genetic programming of a low-distortion 96 Decibel operational amplifier. *Proceedings of the 1997 ACM Symposium on Applied Computing, San Jose, California, February 28 – March 2, 1997*. New York: Association for Computing Machinery. Pages 207 - 216.

Koza, John R., Bennett III, Forrest H, Andre, David, Keane, Martin A, and Dunlap, Frank. 1997. Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Transactions on Evolutionary Computation*. 1(2). Pages 109 – 128.

Koza, John R., Bennett III, Forrest H, Keane, Martin A., and Andre, David. 1997. Automatic programming of a time-optimal robot controller and an analog electrical circuit to implement the robot controller by means of genetic programming. *Proceedings of 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*. Los Alamitos, CA; Computer Society Press. Pages 340 – 346.

Koza, John R., Bennett III, Forrest H, Lohn, Jason, Dunlap, Frank, Andre, David, and Keane, Martin A. 1997. Automated synthesis of computational circuits using genetic programming. *Proceedings of the 1997 IEEE Conference on Evolutionary Computation*. Piscataway, NJ: IEEE Press. 447–452.

Koza, John R., Bennett III, Forrest H, Lohn, Jason, Dunlap, Frank, Andre, David, and Keane, Martin A. 1997b. Use of architecture-altering operations to dynamically adapt a three-way analog source identification circuit to accommodate a new source. In Koza, John R., Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max, Iba, Hitoshi, and Riolo, Rick L. (editors). 1997. *Genetic Programming 1997: Proceedings of the Second Annual Conference* San Francisco, CA: Morgan Kaufmann. 213 – 221.

Koza, John R., Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max, Iba, Hitoshi, and Riolo, Rick L. (editors). 1997. *Genetic Programming 1997: Proceedings of the Second Annual Conference* San Francisco, CA: Morgan Kaufmann.

Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). 1996. *Genetic Programming 1996: Proceedings of the First Annual Conference*. Cambridge, MA: The MIT Press.

Koza, John R., and Rice, James P. 1992. *Genetic Programming: The Movie*. Cambridge, MA: MIT Press.

Kruiskamp Marinum Wilhelmus and Leenaerts, Domine. 1995. DARWIN: CMOS opamp synthesis by means of a genetic algorithm. *Proceedings of the 32nd Design Automation Conference*. New York, NY: Association for Computing Machinery. Pages 433–438.

Quarles, Thomas, Newton, A. R., Pederson, D. O., and Sangiovanni-Vincentelli, A. 1994. *SPICE 3 Version 3F5 User's Manual*. Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA. March 1994.

Rutenbar, R. A. 1993. Analog design automation: Where are we? Where are we going? *Proceedings of the l5th IEEE CICC*. New York: IEEE. 13.1.1-13.1.8.

Samuel, Arthur L. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development.* 3(3): 210–229.

Samuel, Arthur L. 1983. AI: Where it has been and where it is going. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*. Los Altos, CA: Morgan Kaufmann. Pages 1152 – 1157.

Thompson, Adrian. 1996. Silicon evolution. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). 1996. *Genetic Programming 1996: Proceedings of the First Annual Conference*. Cambridge, MA: MIT Press.

Williams, Arthur B. and Taylor, Fred J. 1995. *Electronic Filter Design Handbook*. Third Edition. New York, NY: McGraw-Hill.

Version 4 - Submitted February 6, 1999 for chapter 16 of *Evolutionary Design by Computers* edited by Peter Bentley.

The Design of Analog Circuits by Means of Genetic Programming

John R. Koza
Section on Medical Informatics
Department of Medicine
School of Medicine
Stanford University
Stanford, California 94305
koza@smi.stanford.edu

Forrest H Bennett III
Chief Scientist
Genetic Programming Inc.
Los Altos, California 94023
forrest@evolute.com

David Andre
Division of Computer Science
University of California
Berkeley, California 94720
dandre@cs.berkeley.edu

Martin A. Keane
Chief Scientist
Econometrics Inc.
111 E. Wacker Dr.
Chicago, Illinois 60601
makeane@ix.netcom.com

Send to...

Dr Peter J Bentley
Department of Computer Science
University College London
Gower Street
London WC1E 6BT
United Kingdom