EVOLUTION USING GENETIC PROGRAMMING OF A LOW-DISTORTION 96 DECIBEL OPERATIONAL AMPLIFIER

John R. Koza
Computer Science Dept.
258 Gates Building
Stanford University
Stanford, California 94305
koza@cs.stanford.edu

Forrest H Bennett III
Visiting Scholar
Stanford University
Stanford, California 94305
fhb3@slip.net

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

Martin A. Keane
Martin Keane Inc.
5733 West Grover
Chicago, Illinois 60630
makeane@ix.netcom.com

KEYWORDS

Genetic programming, Automated circuit design, Analog circuit synthesis, operational amplifier

ABSTRACT

There is no known general technique for automatically designing an analog electrical circuit that satisfies design specifications. Genetic programming was used to evolve *both* the topology and the sizing (numerical values) for each component of a low-distortion 96 decibel (64,860 -to-1) amplifier circuit.

1. THE ANALOG DILEMMA

The field of engineering design offers a practical yardstick for evaluating automated techniques because the design process is usually viewed as requiring human intelligence and because design is a major activity of practicing engineers. In the design process, the design requirements specify "what needs to be done." A satisfactory design tells us "how to do it."

In the field of electrical engineering, the design process typically involves the creation of an electrical circuit that satisfies user-specified design goals.

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 circuitshas not proved to be as amenable to automation (Rutenbar 1993). In discussing "the analog dilemma," O. Aaserud and I. Ring Nielsen (1995) (not to be confused with Ivan Riis Nielsen cited later) observe,

> "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."

Of course, engineers employ human reasoning abilities and intelligence in designing complex structures. In contrast, nature employs an entirely different approach to design. In nature, complex structures are designed by means of evolution and natural selection. This suggests the possibility of applying the techniques of evolutionary computation in order to automate the design of complex structures.

Genetic algorithms have been applied to the problem of circuit synthesis. A CMOS operational amplifier (op amp) circuit was designed using a modified version of the genetic algorithm (Kruiskamp and Leenaerts 1995); however, the topology of each op amp was one of 24 pre-selected topologies based on the conventional human-designed stages of an op amp. Thompson (1996) used a genetic algorithm to evolve a frequency discriminator on a Xilinx 6216 reconfigurable digital gate array operating in analog mode.

Holland (1975) described how an analog of the naturally-occurring evolutionary process can be applied to solving scientific and engineering problems using what is now called the *genetic algorithm* (GA).

Koza (1992) described an extension of Holland's genetic algorithm in which the population consists of computer programs. See also Koza and Rice 1992. Koza (1994a, 1994b) described a way to evolve multi-part programs consisting of a main program and one or

more reusable, parameterized, hierarchically-called subprograms (called *automatically defined functions*). Architecture-altering operations provide a way to 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 subprograms (Koza 1995). Recent research papers on genetic programming can be found in Kinnear (1994), Angeline and Kinnear (1996), and Koza, Goldberg, Fogel, and Riolo (1996).

Gruau's *cellular encoding* (1996) is an innovative technique in which genetic programming is used to concurrently evolve the architecture, weights, thresholds, and biases of neurons in a neural network.

This paper demonstrates that a design for a low-distortion 96 decibel (dB) op amp (including both the circuit topology and component sizing) can be evolved using genetic programming. The problem-specific information that the user must supply in order to apply genetic programming to a particular new problem of analog circuit synthesis is minimal; it primarily consists of a fitness measure for the operating characteristics of the desired circuit. The user must also specify certain additional basic information such as the number of inputs and outputs of the desired circuit, the set of parts that are to be available to the circuit (e.g., transistors, resistors, and capacitors), and the repertoire of circuit-constructing functions (which generally does not vary from problem to problem).

Additional evidence of the ability of genetic programming to evolve the design for analog electrical circuits was presented by showing genetically evolved designs for other types of circuits, including a lowpass filter, an asymmetric bandpass filter, and a crossover (woofer and tweeter) filter.

2. Circuit-Constructing Program Trees

Genetic programming can be applied to circuits if a mapping is established between the kind of rooted, point-labeled trees with ordered branches found in genetic programming and the line-labeled cyclic graphs germane to circuits.

The principles of developmental biology suggest a way to map program trees into circuits. The starting point of the growth process used herein is a very simple embryonic electrical circuit. The embryonic circuit contains certain fixed parts appropriate to the problem at hand and certain wires that are capable of subsequent modification. An electrical circuit is progressively developed by applying the functions in a circuit-constructing program tree to the modifiable wires of the embryonic circuit (and, later, to both the modifiable wires and other components of the successor circuits).

The functions in the circuit-constructing program trees are divided into four categories:

(1) connection-modifying functions that modify the topology of the circuit,

(2) component-creating functions that insert components into the circuit,

(3) arithmetic-performing functions that appear in arithmetic-performing subtrees as argument(s) to the component-creating functions and that specify the numerical value of the component, and

(4) automatically defined functions that appear in function-defining branches and potentially enable certain substructures to be reused.

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

2.1. The Embryonic Circuit

The developmental process for converting a program tree into an electrical circuit begins with an embryonic circuit.

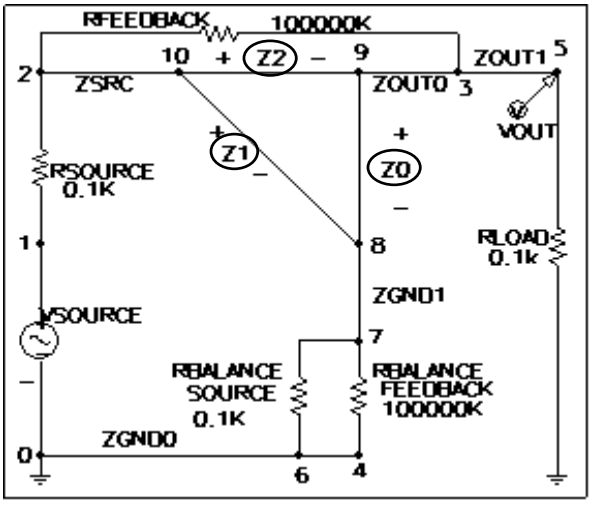

Figure 1 Feedback embryo for an amplifier. corrected 10-96 version of N:\fhb\tojohn\r708826e.doc

Figure 1 shows a one-input, one-output embryonic circuit that serves as a test harness for evolving op amp circuits. VSOURCE is the input signal. VOUT is the output signal. There is a fixed 100 Ohm load resistor RLOAD and a fixed 100 Ohm source resistor RSOURCE.

At the beginning of the developmental process, there is a writing head pointing to (highlighting) each of the three modifiable wires (Z0, Z1, and Z2). All development occurs at wires or components to which a writing head points. The three modifiable wires provide connectivity between the three distinct

elements of a circuit (i.e., the input, the output, and the ground).

The domain knowledge embodied in this embryonic circuit consists of the facts that (1) the embryo has one input and one output, (2) the embryo is a circuit, (3) there are modifiable connections between the output and the source and between the output and ground, and (4) the circuit is to be an amplifier.

Because we are evolving an amplifier, there is also a fixed 100,000,000 Ohm feedback resistor RFEEDBACK, a fixed 100 Ohm balancing source resistor RBALANCE_SOURCE, and a fixed 100,000,000 Ohm balancing feedback resistor RBALANCE_FEEDBACK. This arrangement limits the possible amplification of the evolving circuit to a 1,000,000-to-1 ratio (120 dB).

2.2. Component-Creating Functions

Each circuit-constructing program tree in the population contains component-creating functions and connection-modifying functions.

Component-creating functions insert a component into the developing circuit and assigns component value(s) to the component. Each component-creating function has a writing head that points to an associated highlighted component in the developing circuit and modifies the highlighted component in a specified way. The construction-continuing subtree of each component-creating function points to a successor function or terminal in the circuit-constructing program tree.

The arithmetic-performing subtree of a component-creating function 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, in turn, interpreted, in a logarithmic way, 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 involved) as described in detail in Koza, Andre, Bennett, and Keane (1996).

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 2 shows a modifiable wire Z0 connecting nodes 1 and 2 of a partial circuit containing four capacitors.

Figure 3 shows the result of applying the R function to the modifiable wire Z0 of figure 2.

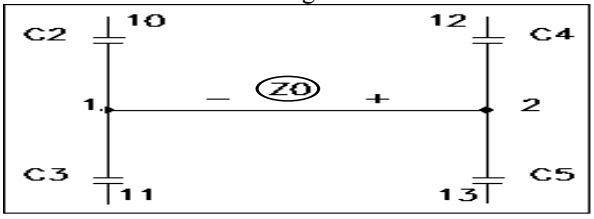

Figure 2 Modifiable wire Z0.

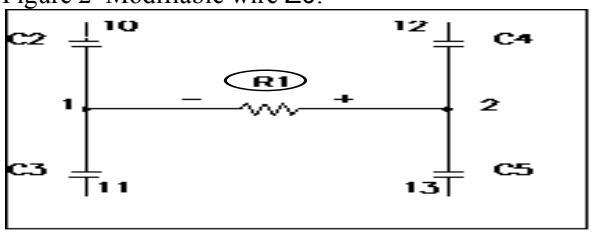

Figure 3 Result of applying the R function.

Similarly, the two-argument capacitor-creating C function causes the highlighted component to be changed into a capacitor. The value of the capacitor in nano-Farads is specified by its arithmetic-performing subtree.

Space does not permit a detailed description of each function herein. See Koza, Andre, Bennett, and Keane (1996), and Koza, Bennett, Andre, and Keane (1996a, 1996b, 1996c, 1996d) for details.

The one-argument Q_D_PNP diode-creating function causes a diode to be inserted in lieu of the highlighted component, where the diode is implemented 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 (called 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 transistor in lieu of the highlighted component. For example, the Q_POS_COLL_NPN function inserts a npn transistor whose collector is connected to the positive power supply.

The three-argument transistor-creating Q_3_NPN function causes an npn bipolar junction transistor (model Q2N3904) 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 new modifiable wires. There is no writing head on the new transistor. Similarly, the three-argument transistor-creating Q_3_PNP function causes a pnp bipolar junction transistor (model Q2N3906) to be inserted.

Figure 4 shows the result of applying the Q_3_NPN0 function, thereby creating transistor Q6 in lieu of modifiable wire Z0 of figure 2.

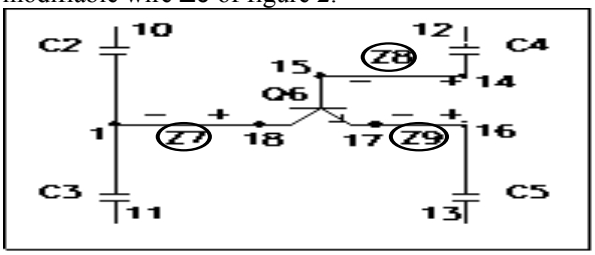

Figure 4 Result of applying Q_3_NPN0 function.

2.3. Connection-Modifying Functions

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

The one-argument polarity-reversing FLIP function attaches the positive end of the highlighted component to the node to which its negative end is currently attached and vice versa. After execution of the FLIP function, there is one writing head pointing to the component.

The three-argument SERIES division function creates a series composition consisting 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.

Figure 5 illustrates the result of applying the SERIES division function to resistor R1 from figure 3.

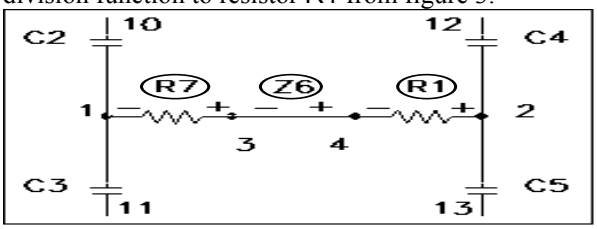

Figure 5 Result after applying the SERIES function.

The four-argument PSS and PSL parallel division functions create 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 6 shows the result of applying PSS to the resistor R1 from figure 3.

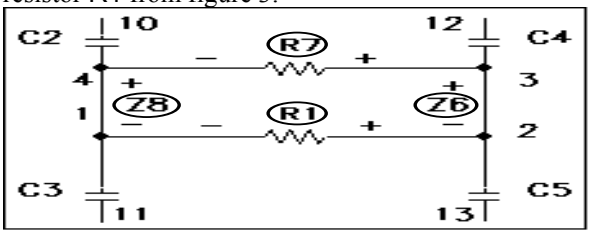

Figure 6 Result of the PSS parallel division function.

There are six three-argument functions (called 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. Figure 7 shows the T_GND_0 function connecting resistor R1 of figure 3 to ground.

The three-argument PAIR_CONNECT_0 and PAIR_CONNECT_1 functions 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 to occur (whether PAIR_CONNECT_0 or PAIR_CONNECT_1) 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 NOP 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. The END function causes its writing head to be lost – 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).

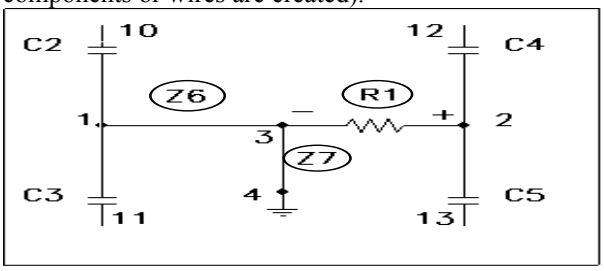

Figure 7 Result of applying the T_GND_0 function.

3. Preparatory Steps

Our goal in this research is to evolve the design of a high-gain amplifier. Before applying genetic programming to circuit synthesis, the user must perform seven major preparatory steps, namely

(1) identifying the embryonic circuit that is suitable for the problem,

(2) determining the architecture of the overall circuit-constructing program trees,

(3) identifying the terminals of the to-be-evolved programs,

(4) identifying the primitive functions contained in the to-be-evolved programs,

(5) creating the fitness measure,

(6) choosing certain control parameters (notably population size and the maximum number of generations to be run), and

(7) determining the termination criterion and method of result designation.

The feedback embryo for the one-input, one-output amplifier circuit of figure 1 is suitable for this problem.

The embryonic circuit has a writing head associated with each of the three result-producing branches and there are three result-producing branches (called RPB0, RPB1, and RPB2) in each program tree. The number of automatically defined functions, if any, will emerge as a consequence of the evolutionary process using the architecture-altering operations. Each program in the initial population of programs has a uniform

architecture with no automatically defined functions (i.e., three result-producing branches).

The terminal sets are identical for all three result-producing branches of the program trees for this problem. The function sets are identical for all three result-producing branches.

The initial function set, $\mathcal{F}_{\text{ccs-initial}}$, for each construction-continuing subtree is

$\mathcal{F}_{\text{ccs-initial}}$ = {R, C, SERIES, PSS, PSL, FLIP, NOP, NEW_T_GND_0, NEW_T_GND_1, NEW_T_POS_0, NEW_T_POS_1, NEW_T_NEG_0, NEW_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 is

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

The set of potential new functions, $\mathcal{F}_{\text{potential}}$, is

$\mathcal{F}_{\text{potential}}$ = {ADF0, ADF1, ADF2, ADF3}.

The set of potential new terminals, $\mathcal{T}_{\text{potential}}$, is

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

The architecture-altering operations change the function set, $\mathcal{F}_{\text{ccs}}$ for each construction-continuing subtree of all three result-producing branches and the function-defining branches, so

$\mathcal{F}_{\text{ccs}} = \mathcal{F}_{\text{ccs-initial}} \approx \mathcal{F}_{\text{potential}}$.

The architecture-altering operations change the terminal set, $\mathcal{T}_{\text{aps-adf}}$, for each arithmetic-performing subtree, so

$\mathcal{T}_{\text{aps-adf}} = \mathcal{T}_{\text{aps-initial}} \approx \mathcal{T}_{\text{potential}}$.

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

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

where ← represents floating-point random constants from –1.0 to +1.0.

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

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

The evaluation of fitness for each individual circuit-constructing program tree in the population begins with its execution. This execution applies the functions in the program tree to the embryonic circuit, thereby developing the embryonic circuit into a fully developed circuit. A netlist describing the fully developed circuit is then created. The netlist identifies each component of the circuit, the nodes to which that component is connected, and the value of that component. Each circuit is then simulated to determine its behavior. The 217,000-line SPICE (Simulation Program with Integrated Circuit Emphasis) simulation program (Quarles et al. 1994) was modified to run as a submodule within the genetic programming system.

Figure 8 provides additional detail on the calculation of fitness. The calculation starts by initializing the current CIRCUIT to the embryonic circuit. The individual circuit-constructing program tree from the population is then executed. This execution causes the component-creating and connection-modifying functions in the program tree to be applied to the current CIRCUIT (i.e., these functions side effect the current CIRCUIT). When this execution is completed, the current CIRCUIT is translated into a NETLIST.

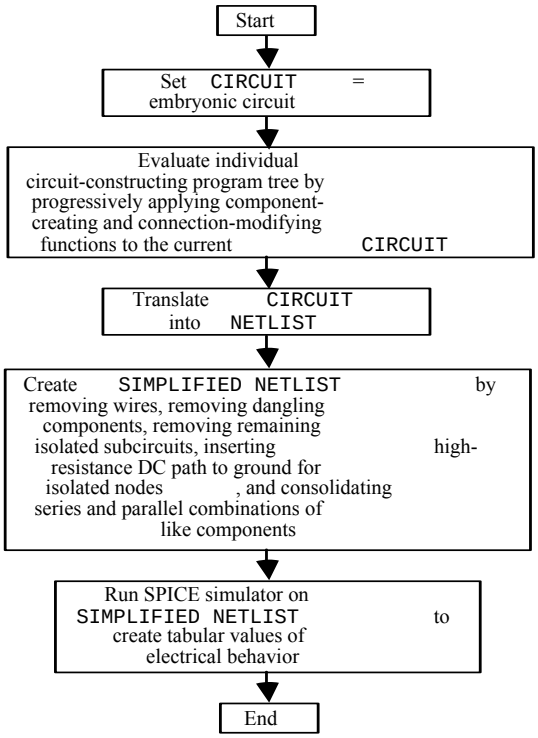

Figure 8 More detailed flowchart for calculation of fitness.

The NETLIST is then simplified in five ways to create a SIMPLIFIED NETLIST. First, all wires are removed. Second, any dangling components are removed. Third, any isolated substructures are removed. Fourth, a very large resistance (a 1 giga-Ohm resistor RHUGE) is inserted between ground and any node for which there is no DC path to ground. For example, if two capacitors join at a certain node, there is no DC path to ground from that node. The introduction of a very large resistance between that node and ground has no significant electrical effect; however, it is essential for enabling the SPICE simulator to simulate the circuit. The fifth simplification of the netlist is done for reasons of accelerating the SPICE simulation. The time required for a SPICE simulation generally increases nonlinearly as a function of the number of nodes in the netlist (in an approximately sub-quadratic to quartic way). Thus, it

is advantageous to shorten the netlist provided to SPICE. All series and parallel compositions of like passive components are replaced, for purposes of the simulation only, by a single component of appropriate value. For example, two resistors (or inductors) in series are replaced by a single resistor (or inductors) whose value if the sum of the two resistances. Two resistors in parallel are reaplced by a single resistor whose value is the reciprocal of the sum of the reciprocal of the two resistances. Two capacitors in parallel are consolidated in the same manner as two resistors in series and two capacitors in series are consolidated in the same manner as two resistors in parallel.

The starting point for evaluating the fitness of a circuit is its response to a DC input. An ideal inverting amplifier circuit would receive a 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; to the extent that the output signal is not centered on 0 Volts (i.e., it has a bias); and to the extent that the DC response is not linear.

Thus, for this problem, we used a fitness measure based on SPICE's DC sweep. The DC sweep analysis measures the DC response of the circuit at several different DC input voltages. The circuits were analyzed with a 5 point DC sweep ranging from –10 millvolts to +10 mV, with input points at –10 mV, –5 mV, 0 mV, +5 mV, and +10 mV. SPICE then produced the circuit's output for each of these five DC voltages.

Fitness is then calculated from four penalties derived from these five DC output values. Fitness is the sum of an amplification penalty, a bias penalty, and two non-linearity penalties.

First, the amplification factor of the circuit is measured by the slope of the straight line between the output for –10 mV and the output for +10 mV (i.e., between the outputs for the endpoints of the DC sweep). If the amplification factor is less than the maximum allowed by the feedback resistor (120 dB for this problem), there is a penalty equal to the shortfall in amplification.

Second, the bias is computed using the DC output associated with a DC input of 0 Volts. The penalty is equal to the bias times a weight. For this problem, a weight of 0.1 is used.

Third, the linearity is measured by the deviation between the slope of each of two line segments and the overall amplification factor of the circuit. The first line segment spans the output values associated with inputs of –10 mv through –5 mv. The second line segment spans the output values associated with inputs of +5 mv and through +10 mv. The penalty for each of these line segments is equal to the absolute value of the difference in slope between the respective line segment and amplification factor of the circuit.

Many of the circuits that are created in the initial random population and many that are created by the crossover and mutation operations cannot be simulated by SPICE. Such circuits are assigned a high penalty value of fitness (10^8).

The population size, M, was 640,000.

The architecture-altering operations are used sparingly on each generation. The percentage of operations on each generation after generation 5 was 86.5% one-offspring crossovers; 10% reproductions; 1% mutations; 1% branch duplications; 0% argument duplications; 0.5% branch deletions; 0.0% argument deletions; 1% branch creations; and 0% argument creations. Since we do not want to waste large amounts of computer time in early generations where only a few programs have any automatically functions at all, the percentage of operations on each generation before generation 6 was 78.0% one-offspring crossovers; 10% reproductions; 1% mutations; 5.0% branch duplications; 0% argument duplications; 1% branch deletions; 0.0% argument deletions; 5.0% branch creations; and 0% argument creations.

The maximum size, H_{rpb}, for each of the three result-producing branches in each overall program is 300 points.

The maximum number of automatically defined functions is 4.

The number of arguments for each automatically defined function is one.

The maximum size, H_{adf}, for each of the automatically defined functions, if any, is 200 points.

The other parameters for controlling the runs of genetic programming were the default values specified in Koza 1994 (appendix D).

This problem was run on a medium-grained parallel Parsytec computer system consisting of 64 80 MHz Power PC 601 processors arranged in a 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. 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 toroidally adjacent processing nodes. See Andre and Koza 1996 for details.

4. Results

We made three identical runs of this problem. The worst of the runs produced an op amp that delivered 92 dB of amplification. We describe the best run here.

About 41% of the circuits of generation 0 cannot be simulated by SPICE; however, the percentage of unsimulatable circuits drops to between 2% and 4% between generations 1 and 10 and never exceeds 8% thereafter.

The fitness of the best-of-generation individual tends to improve from generation to generation.

The best circuit (figure 9) from generation 50 has 33 transistors, no diodes, eight capacitors, and five resistors (in addition to the five resistors of the feedback embryo). It achieves a fitness of 971,076.4. No automatically defined functions are present in this particular circuit. The DC sweep shows that the circuit has an amplification of 89.7 dB (30,545-to-1) and a bias of 9.77 Volts.

Based on the time domain behavior for a 20 microvolt sinusoidal 1,000 Hz input signal, the amplification is 89.7 dB (30,500-to-1) for the best circuit from generation 50; the bias is 9.76 Volts; and the distortion is 6.29%.

Based on the AC sweep for the best circuit of generation 50, the 3 dB bandwidth is 2,300 Hz. The circuit has a flatband gain of 89.7 dB.

The best-of-run circuit (figure 10) appeared in generation 86 and achieves a fitness of 938,427.3. The program tree has two automatically defined functions. `ADF0` is called once; `ADF1` is not called. The circuit (without `ADF0`) has 25 transistors, no diodes, two capacitors, and two resistors (in addition to the five resistors of the feedback embryo).

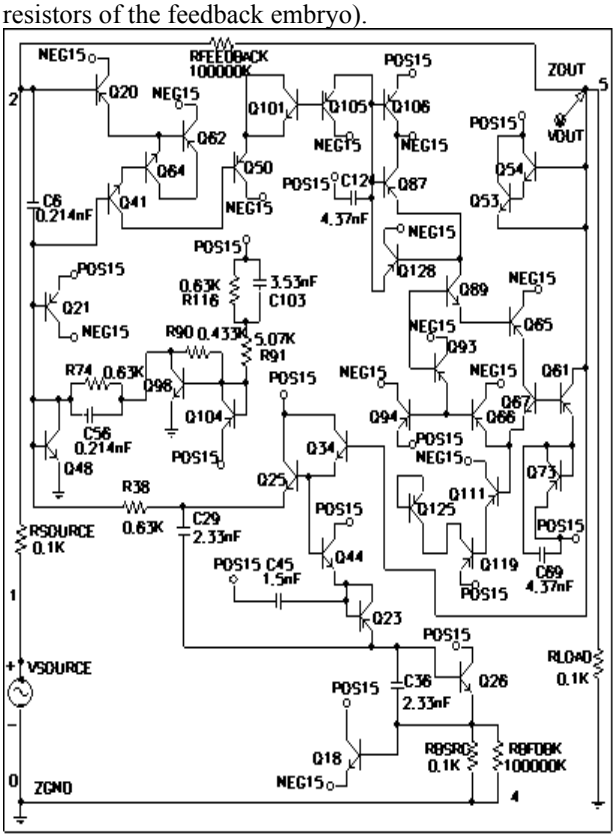

Figure 9 Best circuit from generation 50.

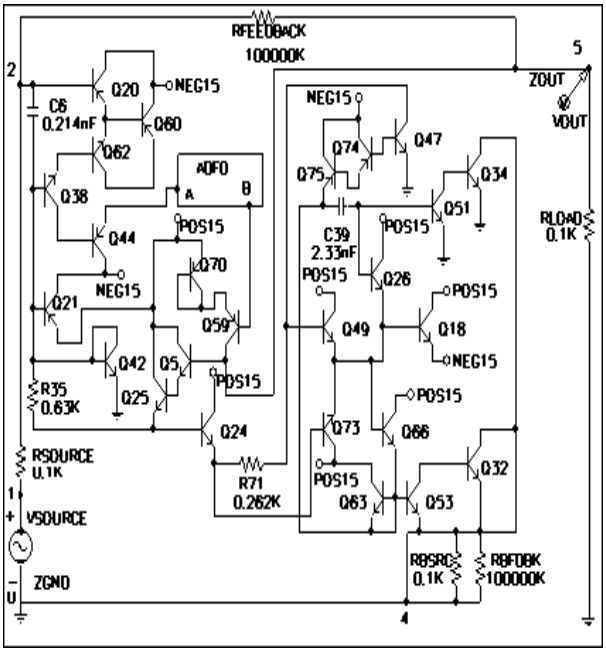

Figure 10 Best circuit from generation 86. d:\fhb\tojohn\r708826a.doc

Figure 11 shows automatically defined function `ADF0` of the best circuit from generation 86 (which has 12 transistors, no diodes, one capacitor, and two resistors).

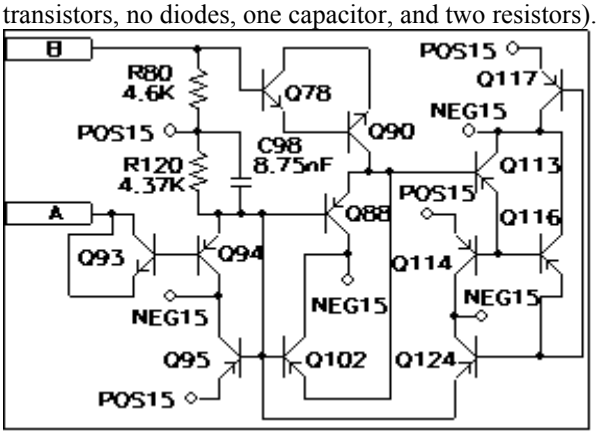

Figure 11 `ADF0` for best circuit from generation 86. d:\fhb\tojohn\r708826a.doc

The DC sweep for this best of generation circuit from generation 86 shows that the circuit has an amplification of 96.2 dB (64,860 -to-1) and a bias of 7.44 Volts.

Figure 12 shows the time domain behavior of the best circuit from generation 86. The vertical axis is voltage from –20 volts to +20 volts. The input is the 20 microvolt sinusoidal signal; however, it appears here as a nearly straight line because of the scale necessary to show the high amplification of the output signal. Based on this transient analysis, the amplification is 94.1 dB; the bias is 7.46 volts; and the distortion is 7.07%.

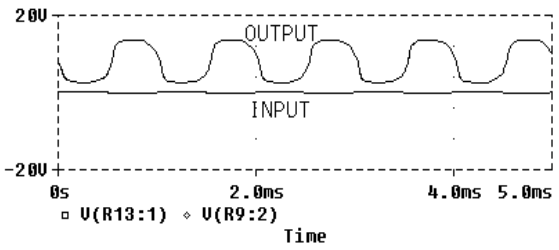

Figure 12 Time domain behavior of best of generation 86.

Figure 13 shows the frequency response of this circuit as shown by an AC sweep. The horizontal axis shows frequency on a logarithmic scale from 1 Hz to 1,000,000 Hz. The vertical axis shows gain and ranges from 0 to 100 dB. The 3 dB bandwidth is 1078.4 Hz. The circuit has a flatband gain of 96.3 dB.

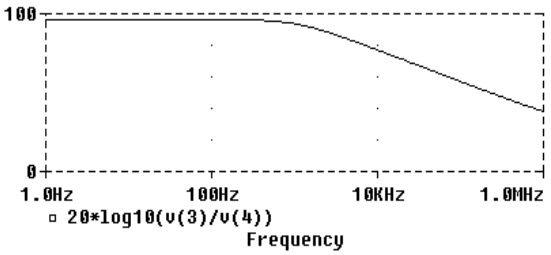

Figure 13 AC sweep for the best circuit from generation 86. d:\fhb\tojohn\r708826a.doc

We do not claim that the genetically evolved amplifier satisfies all the additional requirements that a human design engineer might want to incorporate into a practical design. We do, however, claim that genetic programming successfully created a 96 dB op amp circuit based on the fitness measure that it was given. We also claim that the genetically evolved circuit demonstrates the principle that both the topology and component sizing of a complex analog circuit can be evolved using genetic programming – that is, that the synthesis of analog circuits can be automated.

5. Other

Genetic programming has also been successfully applied to a variety of other problems of analog circuit design.

The problem-specific information that the user must supply in order to apply genetic programming to different problems of analog circuit synthesis is minimal. It primarily consists of a fitness measure for the operating characteristics of the desired circuit. That is, structure arises from fitness. In addition, the user must also specify information such as the number of inputs and outputs of the desired circuit, the set of parts that are to be available to the circuit (e.g., transistors, resistors, and capacitors), and the repertoire of circuit-constructing functions (which generally does not vary from problem to problem).

In this section, we describe several additional circuit synthesis problems in order to demonstrate how to modify the basic technique described above for other problems.

5.1. Lowpass Filter

Genetic programming has successfully evolved a design for a lowpass filter with passband below 1,000 Hz and a stopband above 2,000 Hz with requirements equivalent to that of a fifth order elliptic filter (Koza, Bennett, Andre, and Keane 1996a, 1996c).

For the amplifier described above, the fitness measure was based on gain. For a filter, the fitness measure is based on the amount of voltage that the circuit passes at various frequencies. Specifically, fitness is measured in terms of the sum, over fitness cases representing various frequencies, of the weighted absolute value of the deviation between the actual value of the voltage that is produced by the circuit at the probe point VOUT at node 5 and the target value for voltage. The smaller the value of fitness, the better (with zero being best). A second difference is that the embryo for a filter circuit does not need a balancing or feedback resistor. A third difference is that the desired filter is a passive circuit created from inductors and capacitors (but without transistors or power supplies).

Numerous runs produced lowpass filters having a topology that is similar to that employed by human engineers. For example, in one run, a 100% compliant evolved circuit (figure 14) had the recognizable ladder topology of a Butterworth or Chebychev filter (i.e., a composition of series inductors horizontally with capacitors as vertical shunts).

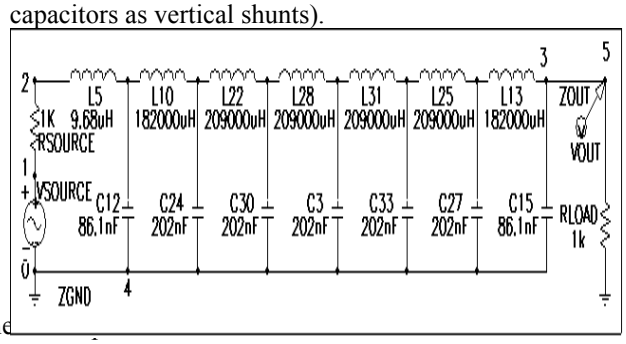

Figure 14 Genetically evolved ladder filter circuit.

5.2. A Crossover Filte

A design for a crossover (woofer and tweeter) filter with a crossover frequency of 2,512 Hz was reported in Koza, Bennett, Andre, and Keane 1996b.

This problem requires a one-input, two-output embryonic circuit and requires that the fitness be measured at two probe points.

The lowpass part of the genetically evolved best-of-run circuit (figure 15) has the Butterworth topology. Except for additional capacitor C38, the highpass part of this circuit also has the Butterworth topology. This circuit is slightly better than the combination of lowpass and highpass Butterworth filters of order 7.

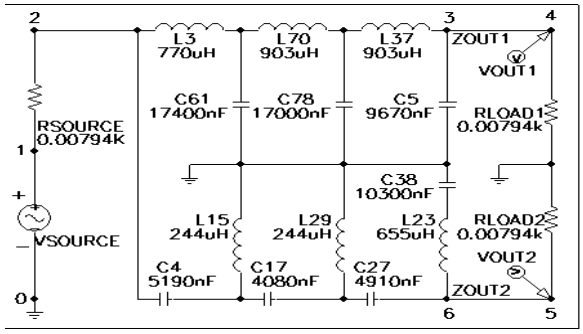

Figure 15 Genetically evolved crossover filter.

5.3. Asymmetric Bandpass Filter

A design for an asymmetric bandpass filter with requirements equivalent to a tenth-order elliptic filter was successfully evolved (Koza, Bennett, Andre, and Keane 1996d).

The preparatory steps required to solve this problem were almost identical to that of the two lowpass filter mentioned above, except that the fitness measure reflected the more complicated asymmetric requirements of the problem.

Figure 16 shows a 100% compliant evolved asymmetric bandpass filter.

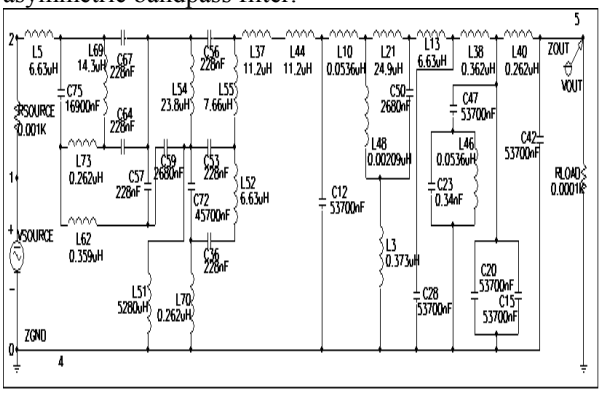

Figure 16 Genetically evolved asymmetric bandpass filter.

5.4. Cube Root Circuit

Analog electrical circuits that perform mathematical functions (e.g., logarithm, square, cube root) are called *computational circuits*. Computational circuits are of special practical importance when the small number of required mathematical functions does not warrant converting an analog signal into a digital signal, performing the mathematical function in the digital domain, and then converting the result to the analog domain. The design of computational circuits is notoriously difficult even for mundane mathematical functions and often relies on the clever exploitation of some aspect of the underlying device physics of the components. Moreover, implementation of each mathematical function typically requires an entirely different clever insight.

A design for a computational circuit for the cube root function has been successfully evolved. Figure 17 shows a genetically evolved cube root circuit with 36 transistors, two diodes, no capacitors, and 12 resistors (in addition to the source and load resistors in the embryo).

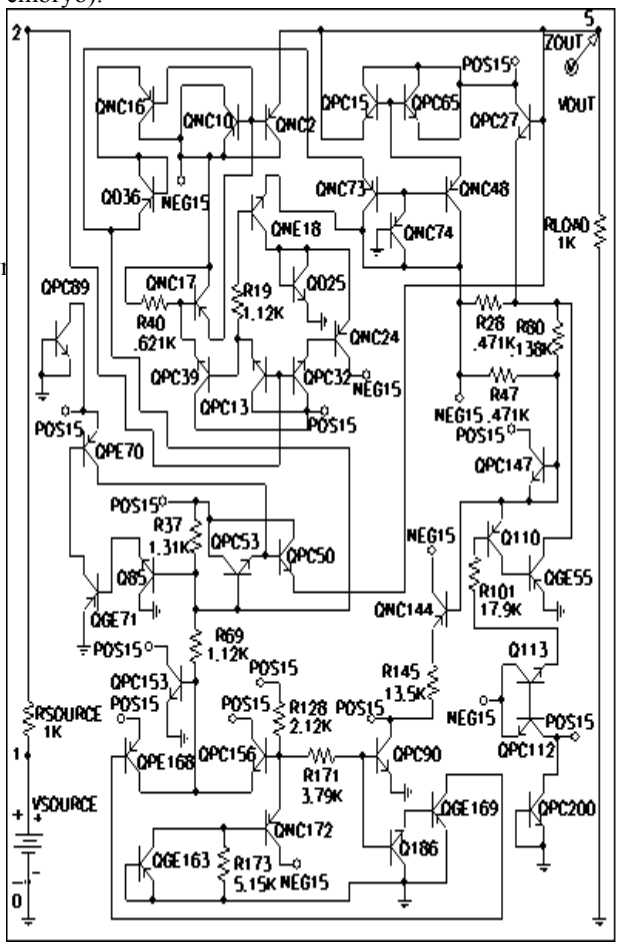

Figure 17 Genetically evolved cube root circuit. d:\users\fhb\tojohn\r708994c.doc

6 CONCLUSION

Genetic programming successfully evolved a 37-transistor amplifier that delivers a DC gain of 96 dB (64,860 -to-1).

ACKNOWLEDGMENTS

Jason Lohn and Simon Handley made helpful comments on various drafts of this paper.

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.

Gruau, Frederic. 1996. Artificial cellular development in optimization and compilation. In Sanchez, Eduardo and Tomassini, Marco (editors). 1996. *Towards Evolvable Hardware*. Lecture Notes in Computer Science, Volume 1062. Berlin: Springer-Verlag. Pages 48 – 75.

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.

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, July 28-31, 1996, Stanford University*. Cambridge, MA: The MIT Press.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996a. Toward evolution of electronic animals using genetic programming. *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*. Cambridge, MA: The MIT Press.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996b. 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. 1996c. 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. 1996d. 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, July 28-31, 1996, Stanford University*. Cambridge, MA: The MIT Press.

Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). 1996. *Genetic Programming 1996: Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University*. 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.

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, July 28-31, 1996, Stanford University*. Cambridge, MA: MIT Press.

BIOGRAPHIES

John R. Koza is Consulting Professor in the Computer Science Department at Stanford University.

Forrest H Bennett III is visiting scholar in the Computer Science Department at Stanford University.

David Andre is a researcher in the Computer Science Department at the University of California at Berkeley.

Martin A. Keane received his PhD from Northwestern University in Mathematics in 1969.

VERSION 2 - CAMERA-READY - SUBMITTED NOVEMBER ---, 1996 TO ACM SYMPOSIUM ON APPLIED COMPUTING (SAC-97) TO BE HELD ON FEBRUARY 28 – MARCH 2, 1997 IN SAN JOSE, CALIFORNIA.

EVOLUTION USING GENETIC PROGRAMMING OF A LOW-DISTORTION 96 DECIBEL OPERATIONAL AMPLIFIER

John R. Koza
Computer Science Dept.
258 Gates Building
Stanford University
Stanford, California 94305
koza@cs.stanford.edu

Forrest H Bennett III
Visiting Scholar
Stanford University
Stanford, California 94305
fhb3@slip.net

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

Martin A. Keane
Martin Keane Inc.
5733 West Grover
Chicago, Illinois 60630
makeane@ix.netcom.com

10 pages = 5 x $20 = $100 page charge