Automatic Synthesis, Placement, and Routing of an Amplifier Circuit by Means of Genetic Programming

Forrest H Bennett III
Genetic Programming Inc.
(Currently, FX Palo Alto Laboratory, Palo Alto, California)
forrest@evolute.com

John R. Koza
Stanford University, Stanford, California
koza@stanford.edu

Jessen Yu
Genetic Programming Inc., Los Altos, California
jyu@cs.stanford.edu

William Mydlowec
Genetic Programming Inc., Los Altos, California
myd@cs.stanford.edu

Abstract

The complete design of a circuit typically includes the tasks of creating the circuit's placement and routing as well as creating its topology and component sizing. Design engineers perform these four tasks sequentially. Each of these four tasks is, by itself, either vexatious or computationally intractable. This paper describes an automatic approach in which genetic programming starts with a high-level statement of the requirements for the desired circuit and simultaneously creates the circuit's topology, component sizing, placement, and routing as part of a single integrated design process. The approach is illustrated using the problem of designing a 60 decibel amplifier. The fitness measure considers the gain, bias, and distortion of the candidate circuit as well as the area occupied by the circuit after the automatic placement and routing.

1 Introduction

The *topology* of a circuit involves specification of the gross number of components in the circuit, the identity of each component (e.g., transistor, capacitor), and the connections between each lead of each component. *Sizing* involves the specification of the values (typically numerical) of each component. *Placement* involves the assignment of each of the circuit's components to a particular geographic (physical) location on a printed circuit board or silicon wafer. *Routing* involves the assignment of a particular geographic location to the wires connecting the various components.

Design engineers typically perform the tasks of creating circuit's topology, sizing, placement, and routing as a series of four separate sequential tasks. Each of these tasks is either vexatious or computationally intractable. In particular, the problem of placement and the problem of routing (both analog and digital) are computationally

intractable combinatorial optimization problems that require computing effort that increases exponentially with problem size (Garey and Johnson 1979).

The mandatory requirements for an acceptable scheme for placement and routing are that there must be a wire connecting every lead of all of the circuit's components, that wires must not cross on a particular layer of a silicon chip or on a particular side (or layer) of a printed circuit board, and that minimum clearance distances must be maintained between wires, between components, and between wires and components. Once these mandatory requirements are satisfied, minimization of area typically becomes the next most important consideration.

Genetic programming has recently been shown to be capable of solving the problem of automatically creating the topology and sizing for an analog electrical circuit from a high-level statement of the circuit's desired behavior (Koza, Bennett, Andre, and Keane 1996; Bennett, Koza, Andre, and Keane 1996; Koza, Bennett, Andre, and Keane 1999; Koza, Bennett, Andre, Keane, and Brave 1999). Numerous analog circuits have been designed using genetic programming, including lowpass, highpass, bandpass, bandstop, crossover, multiple bandpass, and asymmetric filters, amplifiers, computational circuits, temperature-sensing circuits, voltage reference circuits, a frequency-measuring circuit, source identification circuits, and analog circuits that perform digital functions. The circuits evolved using genetic programming include eleven previously patented circuits.

However, this previous work did not address the problem of automatically placing and routing of components and wires at particular geographic locations on a printed circuit board or silicon wafer. This paper demonstrates that genetic programming can be used to automatically create the topology, sizing, placement, and routing of analog electrical circuits. Section 2 presents our method. Section 3 describes the preparatory steps required to apply our method to an illustrative problem involving designing a 60 decibel amplifier. Section 4 presents the results.

2 Method

A printed circuit board or silicon wafer has a limited number of layers that are available for wires and a limited number of layers (usually one for a wafer and one or two for a board) that are available for both wires and components. Each wire and component is located at a particular relative geographic (physical) location on the printed circuit board or silicon wafer.

We create electrical circuits using a developmental process in which the component-creating functions, topology-modifying functions, and development-controlling functions of a circuit-constructing program tree are executed. Each of these three types of functions is associated with a modifiable wire or modifiable component in the developing circuit. The starting point of the developmental process is an initial circuit consisting of an embryo and a test fixture. The embryo consists of modifiable wire(s). The embryo is embedded into a test fixture consisting of fixed (hard-wired) components (e.g., the source of the incoming signal) and certain fixed wires that provide connectivity to the circuit's external inputs and outputs. Until the modifiable wires are modified by the developmental process, the circuit produces only trivial output. An electrical circuit is developed by progressively applying the functions in a circuit-constructing program tree (in the population being bred by genetic programming) to the modifiable wires of the original embryo and to the

modifiable components and modifiable wires created during the developmental process. The functions in the program tree are progressively applied (in a breadth-first order) to the initial circuit and its successors until a fully developed circuit emerges.

2.1 The Initial Circuit

Figure 1 shows a one-input, one-output initial circuit (consisting of an embryo and a test fixture) located on one layer of a silicon wafer or printed circuit board. The embryo consists of the three modifiable wires, Z0, Z1, and Z2 (in the middle of the figure). All development originates from these modifiable wires. The test fixture contains two ground points G, an input point V (lower left), an output point O (upper right), nonmodifiable wires (hashed), and four nonmodifiable resistors. There is a fixed 1 kilo-Ohm (kΩ) source resistor R4, a fixed 1 kΩ load resistor R18, a fixed 1 giga-Ohm feedback resistor R14, and a fixed 999 Ω balancing resistor R3.

Each element of this initial circuit (and all successor circuits created by the developmental process) resides at particular geographic location on the circuit's two-dimensional substrate. Each element occupies a particular amount of space. For example, the resistors each occupy a 3 × 3 area; the source point V and the output probe point O each occupy a 1 × 1 area; the nonmodifiable wires each occupy a 1 × *n* or *n* × 1 area; the modifiable wires each occupy a 1 × 1 area.

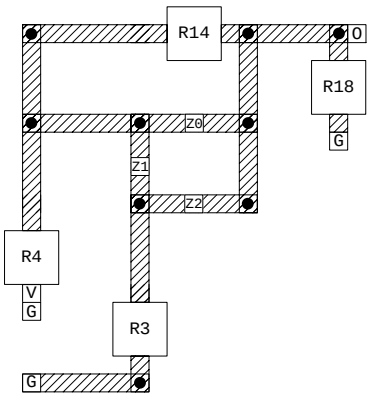
The initial circuit in the developmental process complies with the requirements that wires must not cross on a particular layer of a silicon chip or on a particular side of a printed circuit board, that there must be a wire connecting 100% of the leads of all the circuit's components, and that minimum clearance distances between wires, between components, and between wires and components must be respected. Each of the circuit-constructing functions (described below) preserves compliance with these requirements. Thus, every fully laid-out circuit complies with these requirements.
The component-creating functions insert a component into the developing circuit and assign component value(s) to the new component.

2.2 Circuit-Constructing Functions

Figure 2 shows a partial circuit containing four capacitors (C2, C3, C4, and C5) and a modifiable wire Z0. Each capacitor occupies a 3 × 3 area and is located at a particular geographic location (indicated by an X and Y coordinate). Each piece of wire occupies a 1 × *n* or an *n* × 1 area. The modifiable wire Z0 occupies a 1 × 1 area.

Figure 3 shows the result of applying the one-argument transistor-creating NPN-TRANSISTOR-LAYOUT function to the modifiable wire Z0 of figure 2. The newly created *npn* (q2n3904 BJT) transistor Q6 occupies a 3 × 3 area and is located at (18, 20). The newly created component is larger than that which it replaces. Thus, its insertion affects the locations of preexisting components C2 and C3 in the developing circuit. Specifically, preexisting capacitor C2 is pushed north by one unit thereby relocating it from (18, 23) to (18, 24). Similarly, preexisting capacitor C3 is pushed south by one unit thereby relocating it from (18, 17) to (18, 16). In actual practice, all adjustments in location are made after the completion of the entire developmental process. Details of implementation of this function (and other functions described herein) are found in Koza and Bennett 1999. The newly created transistor is not subject to subsequent modification (and hence there is no

construction-continuing subtree). Similarly, the PNP-TRANSISTOR-LAYOUT



function inserts a *pnp* (q2n3906 BJT) transistor.

Figure 1 Initial circuit consisting of embryo and test fixture.

The two-argument capacitor-creating LAYOUT-C function inserts a capacitor into a developing circuit in lieu of a modifiable wire (or modifiable component). This component-creating function takes an argument that specifies component sizing. Similar functions insert other two-leaded components (e.g., resistors and inductors). The sizing of components are established by a numerical value. In the initial random generation of a run, the numerical value is set, individually and separately, to a random value in a chosen range. In later generations, the numerical value may be perturbed by a mutation operation using a Gaussian probability distribution.

2.3 Topology-Modifying Functions

The topology-modifying functions modify the topology of the developing circuit.

The two-argument SERIES-LAYOUT function creates a series composition consisting of the modifiable wire or modifiable component with which the function is associated and a copy of the modifiable wire or modifiable component.

Each of the two functions in the PARALLEL-LAYOUT family of four-argument functions creates a parallel composition consisting of two new modifiable wires, the preexisting modifiable wire or modifiable component with which the function is associated, and a copy of the modifiable wire or modifiable component.

The one-argument polarity-reversing FLIP function reverses the polarity of the modifiable component or modifiable wire with which the function is associated.

Most practical circuits are not entirely planar. Vias provide a way to connect distant points of a circuit. Each of the four functions in the VIA-TO-GROUND-LAYOUT family of three-argument functions creates a T-shaped composition consisting of the modifiable wire or modifiable component with which the function is associated, a copy of it, two new modifiable wires, and a via to ground. There is a similar VIA-TO-POSITIVE-LAYOUT family of four three-arguments functions to allow direct connection to a positive power supply and a similar VIA-TO-NEGATIVE-LAYOUT family for the negative power supply.

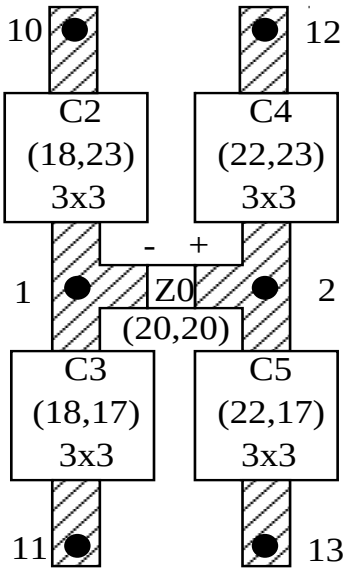


Figure 2 Partial circuit with a 1 × 1 piece of modifiable wire Z0 at location (20, 20).

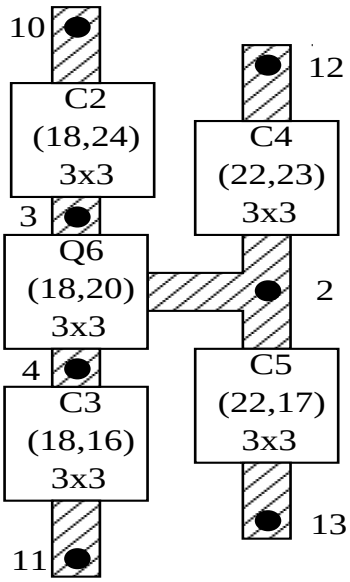


Figure 3 The result of applying the NPN-TRANSISTOR-LAYOUT function.

Similarly, numbered vias provide connectivity between two different layers of a multi-layered silicon wafer or multi-layered printed circuit board. A distinct four-member family of three-argument functions is used for each layer. For example, the VIA-0-LAYOUT and VIA-1-LAYOUT families of functions makes connection with a layers numbered 0 and 1, respectively, of a two-layered substrate.

2.4 Development-Controlling Functions

The zero-argument END function makes the modifiable wire or modifiable component with which it is associated into a non-modifiable wire or component (thereby ending a particular developmental path).

The one-argument NOOP ("No Operation") function has no effect on the modifiable wire or modifiable component with which it is associated; however, it delays the developmental process on the particular path on which it appears.

3 Preparatory Steps

The method will be illustrated on the problem of creating the topology, component sizing, placement, and routing for a 60 dB amplifier with zero distortion and zero bias and with the smallest possible total area for the bounding rectangle of the fully laid-out circuit. (See Bennett, Koza, Andre, and Keane 1996 for a more detailed statement of this problem, without consideration of placement and routing). The circuit is to be constructed on a two-sided printed circuit board with two internal layers. The top side contains discrete components (e.g., transistors, capacitors, and resistors,) that are connected by perpendicularly intersecting metallic wires. The bottom side is devoted to connections to ground. The two internal layers are devoted to via 0 and via 1.

3.1 Initial Circuit

We use the one-input, one-output initial circuit (figure 1) consisting of a test fixture and an embryo with three modifiable wires.

3.2 Program Architecture

There is one result-producing branch in the program tree for each modifiable wire in the embryo. Thus, the architecture of each circuit-constructing program tree has three result-producing branches. Neither automatically defined functions nor architecture-altering operations are used.

3.3 Function and Terminal Sets

The terminal set, $\mathcal{T}_{CCS}$, for each construction-continuing subtree consists of the development-controlling `END` function. The function set, $\mathcal{F}_{CCS}$, for each construction-continuing subtree includes component-creating functions for *npn* transistors, *pnp* transistors, capacitors, resistors, and inductors (a totally extraneous component for this problem); the development-controlling `NOOP` function; and topology-modifying functions for series, parallel, flips and vias to ground, the positive power supply, the negative power supply, and layers 0 and 1 of the printed circuit board.

3.4 Fitness Measure

The fitness measure is based on the area of the bounding rectangle of the laid-out circuit as well as the gain, bias, and distortion of the candidate amplifier circuit.

The evaluation of the fitness of each individual circuit-constructing program tree in the population begins with its execution. This execution progressively applies the functions in the program tree to the embryo of the circuit, thereby creating a fully developed (and fully laid out) circuit. Since the developmental process for creating the fully developed circuit includes the actual geographic placement of components and the actual geographic routing of wires between the components, the area of the bounding rectangle for the fully developed circuit can be easily computed.

A netlist is then 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 is the input to our modified version of the SPICE simulator (Quarles, Newton, Pederson, and Sangiovanni-Vincentelli 1994).

An amplifier can be viewed in terms of 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 of the circuit is not linear.

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 (mv) to +10 mv, with input points at –10 mv, –5 mv, 0 mv, +5 mv, and +10 mv. SPICE then simulated the circuit's behavior for each of these five DC voltages. Four penalties (an amplification penalty, bias penalty, and two non-linearity penalties) are then derived.

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 target (60 dB), 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. There is a penalty equal to the bias times a weight. A weight of 0.1 is used.

Third, the linearity is measured by the deviation between the slope of each of two shorter lines and the overall amplification factor of the circuit. The first shorter line segment connects the output value associated with an input of –10 mv and the output value for –5 mv. The second shorter line segment connects the output value for +5 mv and the output for +10 mv. There is a penalty for each of these shorter line segments equal to the absolute value of the difference in slope between the respective shorter line segment and the overall amplification factor of the circuit.

The fitness measure is multiobjective. Fitness is the sum of (1) the area of the bounding rectangle for the fully developed and laid-out circuit weighted by 10^{-6}, (2) the amplification penalty, (3) the bias penalty, and (4) the two non-linearity penalties; however, if this sum is less than 0.1 (indicating achievement of a very good amplifier), the fitness becomes simply the rectangle's area multiplied by 10^{-6}. Thus, after a good amplifier design is once achieved, fitness is based solely on area minimization.

Circuits that cannot be simulated by SPICE receive a penalty value of fitness (10^8).

3.5 Control Parameters

The population size, M, is 10,000,000. A maximum size of 300 points (functions and terminals) was established for each of the three result-producing branches for each program tree. The other control parameters are those that we have used on many other problems (Koza, Bennett, Andre, and Keane 1999, Appendix D).

3.6 Implementation on Parallel Computer

This problem was run on a home-built Beowulf-style (Sterling, Salmon, Becker, and Savarese 1999) parallel cluster computer system consisting of 1,000 350 MHz Pentium II processors (each accompanied by 64 megabytes of RAM). The system has a 350 MHz Pentium II computer as host. The processing nodes are connected with a 100 megabit-per-second Ethernet. The processing nodes and the host use the Linux operating system. The distributed genetic algorithm with unsynchronized generations and semi-isolated subpopulations was used with a subpopulation size of Q = 10,000 at each of D = 1,000 demes. As each processor (asynchronously) completes a generation, four boatloads of emigrants from each subpopulation are dispatched to each of the four toroidally adjacent processors. The 1,000 processors are hierarchically organized. There are $5 \times 5 = 25$ high-level groups (each containing 40 processors). If the adjacent node belongs to a different group, the migration rate is 2% and emigrants are selected based on fitness. If the adjacent node belongs to the same group, emigrants are selected randomly and the migration rate is 5% (10% if the adjacent node is in the same physical box).

4 Results

The best-of-generation circuit from generation 0 has a fitness of 999.86890.

The first best-of-generation circuit (figure 4) delivering 60 dB of amplification appears in generation 65. This 27-component circuit occupies an area of 8,234 and has an overall fitness of 33.042583.

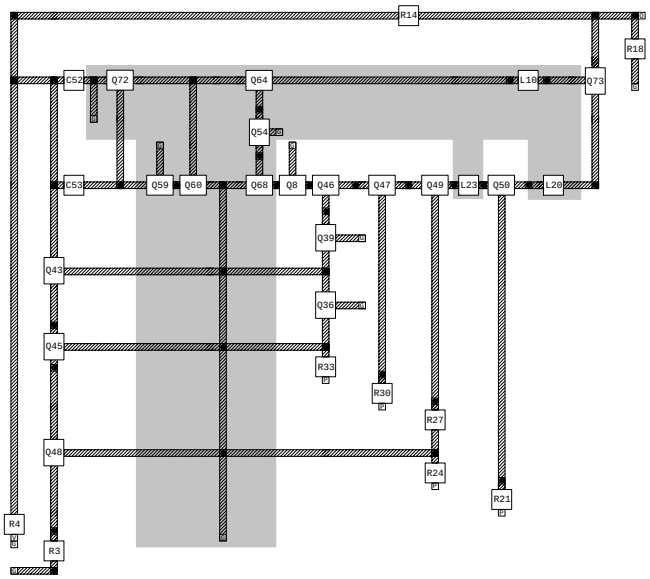


Figure 4 Best-of-run circuit from generation 65

The best-of-run circuit (figure 5) appears in generation 101. This circuit contains 11 transistors, 5 resistors, and 3 capacitors. The four "P" symbols indicate via's to the positive power supply. This 19-component circuit occupies an area of 4,751 and has an overall fitness of 0.004751. It occupies only 58% of the area of the 27-component circuit from generation 65. Note that figures 4 and 5 use different scales. Table 1 shows the number of components, the area, the four penalties comprising the non-area portion of the fitness measure, and the overall fitness for the these two circuits.

Table 1 Comparison of two best-of-generation circuits.

Generation	Components	Area	Four penalties	Fitness
65	27	8,234	33.034348	33.042583
101	19	4,751	0.061965	0.004751

The best-of-generation circuit from generations 65 has 81, 189, and 26 points, respectively, in its three branches. The best-of-run circuit from generation 101 has 65, 85, and 10 points, respectively, in its three branches. That is, the total size of both individuals and the size of each corresponding branch was reduced.

The third branches of these two individuals are both very small (26 and 10 points, respectively). The only effect of these branches are to insert a single transistor (a *pnp* transistor in the generation 65 and an *npn* transistor in generation 101).

The shaded portion of figure 4 shows the portion of the best circuit from generation 65 that is deleted in order to create the best circuit of generation 101.

The first branches of these two individuals are so similar that it is clear that these two branches are genealogically related. These two first branches account for 14 components (nine transistors and five resistors) that are in common with both circuits.

The second branches of these two individuals are almost completely different. The second branches account for the bulk of the reduction in component count (six transistors and three inductors) and the one added component (capacitor C10).

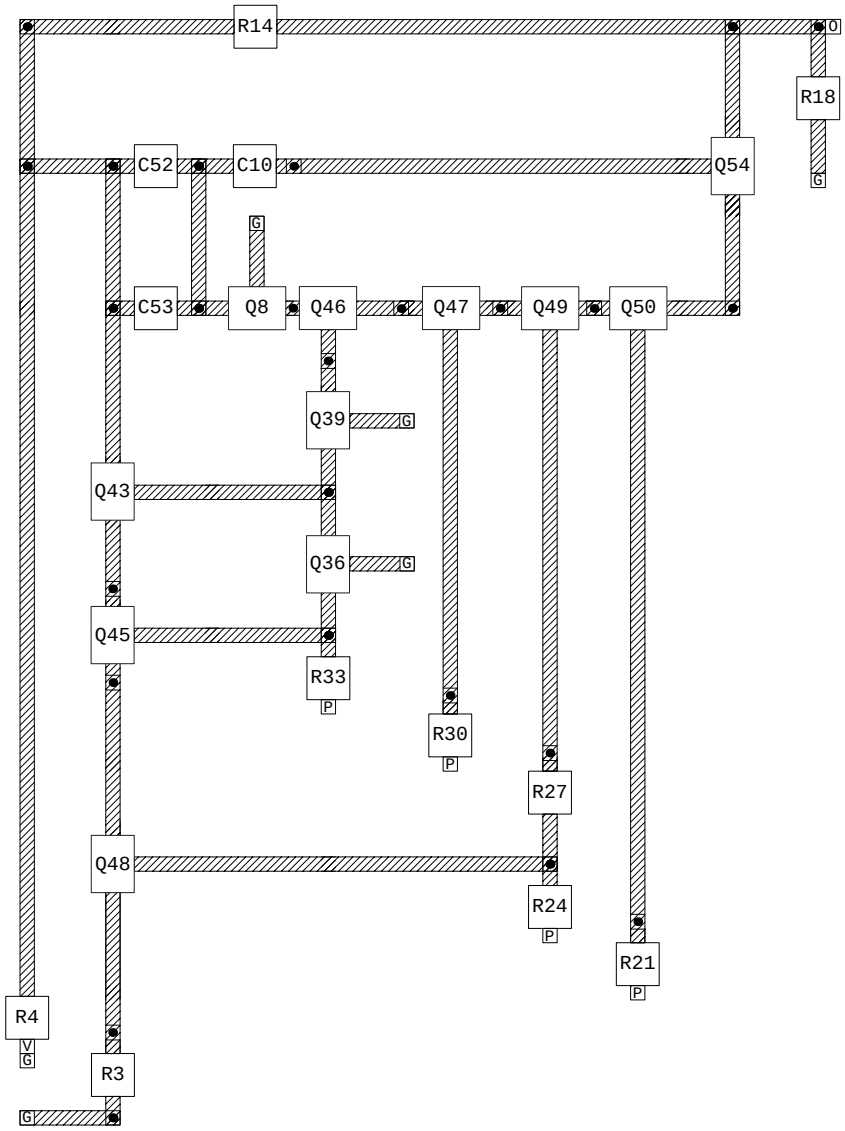


Figure 5 Best-of-run circuit from generation 101.

The difference between generations 65 and 101 caused by the first branches is that two extraneous components that are present in generation 65 are missing from the smaller 19-component circuit from generation 101.

Table 2 shows the *X* and *Y* coordinates for of the 19 components of the best circuit of generation 101 as well as the component value (sizing) for each capacitor and resistor and the type (*npn* q2n3904 or *pnp* q2n3904) for each transistor.

Table 2 Placement of the 19 components of the best circuit of generation 101.

Component	*X* coordinate	*Y* coordinate	Sizing / Type
Q8	-8.398678	21.184582	q2n3906
C10	-8.54565	31.121107	1.01e+02nf
R21	18.245857	-24.687471	1.48e+03k
R24	12.105233	-20.687471	5.78e+03k
R27	12.105233	-12.690355	3.61e+03k
R30	5.128666	-8.690355	8.75e+01k
R33	-3.398678	-4.690355	1.16e+03k
Q36	-3.398678	3.30961	q2n3906
Q39	-3.398678	13.309582	q2n3906
Q43	-18.472164	8.309597	q2n3904
Q45	-18.472164	-1.690355	q2n3904
Q46	-3.398678	21.184582	q2n3904
Q47	5.128666	21.184582	q2n3904
Q48	-18.472164	-17.687471	q2n3904
Q49	12.105233	21.184582	q2n3904
Q50	18.245857	21.184582	q2n3904
C52	-15.472164	31.121107	1.25e-01nf
C53	-15.472164	21.184582	7.78e+03nf
Q54	24.873787	31.121107	q2n3906

References

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.

Garey, Michael R. and Johnson, David S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY: W. H. Freeman.

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

Koza, John R., and Bennett III, Forrest H. 1999. Automatic synthesis, placement, and routing of electrical circuits by means of genetic programming. In Spector, Lee, Langdon, William B., O'Reilly, Una-May, and Angeline, Peter (editors). *Advances in Genetic Programming 3*. Cambridge, MA: MIT Press. Chapter 6. Pages 105 - 134.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1996. 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 Academic. Pages 151-170.

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, Keane, Martin A., and Brave Scott. 1999. *Genetic Programming III Videotape: Human-Competitive Machine Intelligence*. San Francisco, CA: Morgan Kaufmann.

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.

Sterling, Thomas L., Salmon, John, and Becker, Donald J., and Savarese. 1999. *How to Build a Beowulf: A Guide to Implementation and Application of PC Clusters*. Cambridge, MA: The MIT Press.