Evolution by Means of Genetic Programming of Analog Circuits that Perform Digital Functions

Forrest H Bennett III
Genetic Programming Inc.
Box 1669
Los Altos, California 94023
forrest@evolute.com
http://www.genetic-programming.com

John R. Koza
Section on Medical Informatics
Department of Medicine
Stanford University
Stanford, California 94305
koza@stanford.edu
http://www.smi.stanford.edu/people/koza

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

Jessen Yu
Genetic Programming Inc.
Box 1669
Los Altos, California 94023
jyu@cs.stanford.edu

William Mydlowec
Genetic Programming Inc.
Box 1669
Los Altos, California 94023
myd@cs.stanford.edu

Oscar Stiffelman
Computer Science Department
Stanford University
Stanford, California 94305
ozzie@cs.stanford.edu

ABSTRACT

This paper demonstrates the ability of genetic programming to evolve analog circuits that perform digital functions and mixed analog-digital circuits. The evolved circuits include two purely digital circuits (a 100 nano-second NAND circuit and a two-instruction arithmetic logic unit circuit) and one mixed-signal circuit, namely a three-input digital-to-analog converter.

1. Introduction

It has been recently demonstrated genetic programming is capable of synthesizing the design of certain analog electrical circuits (Koza, Bennett, Andre, Keane, and Dunlap 1997; Koza, Bennett, Andre, and Keane 1999a, 1999b). In fact, nine of the analog circuits presented in *Genetic Programming: Darwinian Invention and Problem Solving* (Koza, Bennett, Andre, and Keane 1999a) were considered to be creative and inventive at the time they were first discovered and were patented by their inventors. Specifically, the previously patented, genetically evolved circuits include the Darlington emitter-follower transistor circuit (patented by Sidney Darlington of American Telephone and Telegraph in 1952), the circuit that is now known as the "constant *K*" ladder filter (patented by George Campbell in 1917), the "*M*-derived half section" for a filter (patented by Otto Zobel in 1925), the elliptic filter topology (patented by Wilhelm Cauer between 1934 and 1936), and the crossover filter (patented by Otto Zobel in 1925). Other genetically evolved circuits that were covered by one or more patents issued in the past 30 years include an electronic thermometer, a voltage reference circuit, several different high-gain, low-distortion, low-bias, broad-bandwidth amplifiers, and several different computational circuits.

Previous work on circuit synthesis using genetic programming neglected important categories of analog circuits, including analog circuits that perform digital functions and mixed analog-digital circuits. One reason is that the evaluation of the behavior of digital circuits requires time-consuming simulations in the time domain for a number of fitness cases (typically $c2^k$, where k is the number of input bits and c is some constant).

Section 2 provides general background on genetic programming. Sections 3 presents the preparatory steps for evolving a two-input NAND circuit and section 4 presents the results. Sections 5 and 6 describe the evolution of a three-input digital-to-analog converted (DAC) circuit. Sections 7 and 8 describe the evolution of a two-instruction arithmetic logic unit ALU) circuit.

2 Background on Genetic Programming

Genetic programming is an extension of the genetic algorithm (Holland 1975). Genetic programming automatically creates computer programs to solve problems. Genetic programming is described in Koza 1992; Koza and Rice 1992; Koza 1994a, 1994b; Banzhaf, Nordin, Keller, and Francone 1998; Langdon 1998; Kinnear 1994; Angeline and Kinnear 1996; Spector, Langdon, O'Reilly, and Angeline 1999; Koza, Goldberg, Fogel, and Riolo 1996; Koza, Deb, Dorigo, Fogel, Garzon, Iba, and Riolo 1997; Koza, Banzhaf, Chellapilla, Deb, Dorigo, Fogel, Garzon, Goldberg, Iba, and Riolo 1998; and Banzhaf, Poli, Schoenauer, and Fogarty 1998.

3 Preparatory Steps for NAND

NAND circuits are multiple transistor circuits that perform the elementary two-input Boolean NAND function.

Before applying genetic programming to a problem of circuit synthesis, seven major preparatory steps are required: (1) identify the initial circuit (test fixture and embryo) of the developmental process, (2) determine the architecture of the circuit-constructing program trees, (3) identify the primitive functions of the program trees, (4) identify the terminals of the program trees, (5) create the fitness measure, (6) choose control parameters for the run, and (7) determine the termination criterion and method of result designation.

3.1 Initial Circuit for NAND

An electrical circuit can be created by genetic programming by means of a developmental process. This developmental process entails the execution of a circuit-constructing program tree that contains various component-creating, topology-modifying, and development-controlling functions. An initial circuit consisting of an embryo and a test fixture is the starting point of the developmental process for transforming a program tree in the population into a fully developed electrical circuit. The embryo contains at least one modifiable wire. The test fixture is a fixed (hard-wired) substructure composed of nonmodifiable wires and nonmodifiable electrical components. The test fixture provides access to the circuit's external input(s) and permits probing of the circuit's output. A test fixture has one or more ports that enable an embryo to be embedded into it. An embryo has one or more ports that enable it to communicate with the test fixture in which it is embedded. All

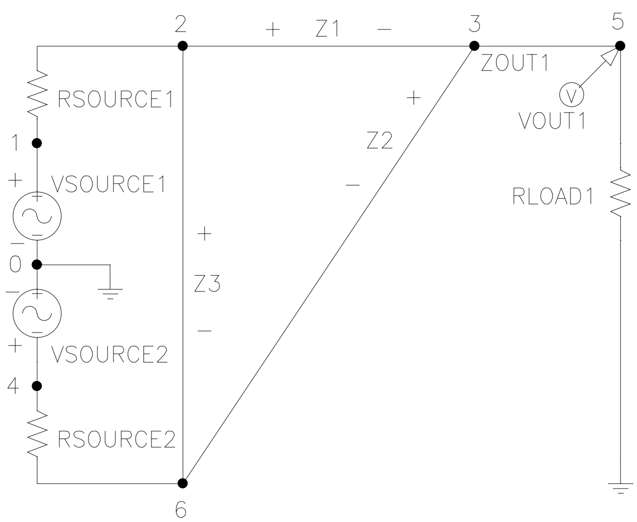

Figure 1 Two-input, one-output initial circuit.

Figure 1 shows a two-input, one-output initial circuit consisting of an embryo embedded in a test fixture. The embryo consists of three modifiable wires Z1, Z2, and Z3. The test fixture receives two incoming signals VSOURCE1 and VSOURCE2, each with two 1Ω source resistors (RSOURCE1 and RSOURCE2), a nonmodifiable wire ZOUT1 between nodes 3 and 5, a probe point VOUT1 (output of the overall circuit) at node 5, and a load resistor RLOAD1 (whose value is 1,000 Ω) between nodes 5 and 0.

3.2 Program Architecture for NAND

Since there is a result-producing branch in the program tree for each modifiable wire in the embryo, the architecture of each program tree has three result-producing branches.

3.3 Function Set for NAND

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

$\mathcal{F}_{ccs}$ = {R, SERIES, PARALLEL0, PARALLEL1, FLIP,
NOP, RETAINING_THREE_GROUND_0,
RETAINING_THREE_GROUND_1,
RETAINING_THREE_POS5V_0,
RETAINING_THREE_POS5V_1,
PAIR_CONNECT_0, PAIR_CONNECT_1,
Q_DIODE_NPN, Q_DIODE_PNP,
Q_THREE_NPN0, ..., Q_THREE_NPN11,
Q_THREE_PNP0, ..., Q_THREE_PNP11,
Q_POS5V_COLL_NPN, Q_POS5V_EMIT_PNP,
Q_GND_EMIT_NPN, Q_GND_EMIT_PNP}

Space does not permit a detailed explanation of all the above functions; however, all the functions in this section are described in detail in Koza, Bennett, Andre, and Keane 1999a. Briefly, the R function is a component-creating function that inserts a resistor into a developing circuit and that establishes the numerical value of the inserted component. The SERIES and the two PARALLEL functions modify the topology of the developing circuit by performing a series or parallel division, respectively. The FLIP function reverses the polarity of a component. The NOP (No operation) function is a development-controlling function.

3.4 Terminal Set for NAND

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

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

Briefly, the development-controlling END function makes the modifiable wire or modifiable component with which it is associated non-modifiable (thereby ending a particular developmental path). The SAFE_CUT function causes the highlighted component to be removed from the circuit in a way that preserves the validity of the circuit.

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

$\mathcal{T}_{aps}$ = {$\Re$}.

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

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

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

3.5 Fitness Measure for NAND

The evaluation of each individual circuit-constructing program tree in the population begins with its execution. The execution progressively applies the functions in the program tree to the embryo of the 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.

The output voltage VOUT is measured in the time domain. SPICE is instructed to perform a transient (time domain) analysis.

Both of the inputs to the initial circuit are presented with 18 100-ns digital signals. Each signal is sampled every 20 ns (i.e., five sample points per 100 ns). Thus, there are 91 fitness cases (figure 2). Both inputs are zero during the first 100 ns. The next 17 input pairs represent the 16 possible transitions between each of the four possible combinations of the two input signals.

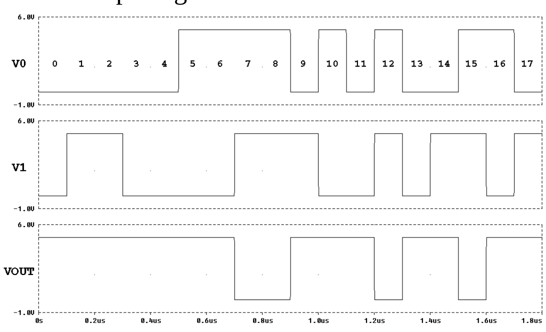

Figure 2 Ninety-one fitness cases in the time domain for the NAND circuit.

The fitness of a circuit is the sum, over the 91 fitness cases, of the weighted absolute value of the difference between the actual output voltage at the probe point VOUT and the desired output voltage.

The fitness measure is designed to not penalize ideal voltage values, to slightly penalize every acceptable voltage, and to heavily penalize every unacceptable voltage.

If the desired digital signal is 1 and the actual voltage is 5.0 volts or the desired digital signal is 0 and the actual voltage is 0, the deviation is 0. If the desired digital signal is 1 and the actual voltage is within 0.3 volts of 5.0 volts or the desired digital signal is 0 and the actual voltage is within 0.4 volts of 0.0 volts, the absolute value of the deviation from the desired output voltage (5 volts or 0 volts, respectively) is weighted by a factor of 1.0. If the actual voltage is outside this range, the absolute value of the deviation is weighted by a factor of 10.0. The smaller the overall value of fitness, the better.

The number of “hits” is defined as the number of fitness cases for which the voltage is acceptable.

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

3.6 Control Parameters for NAND

The population size, *M*, is 132,000. A maximum size of 300 points (functions and terminals) was established for each branch of each circuit-constructing program tree. Other control parameters were the ones that are used previously for the robot controller problem in chapter 48 and appendix D of Koza, Bennett, Andre, and Keane 1999a.

3.7 Termination Criterion for NAND

Since the goal is to generate a variety of 100%-compliant circuits for examination, the run was not automatically terminated upon evolution of the first 100%-compliant individual. Instead, the maximum number of generations, *G*, is set to an arbitrary large number (e.g., 501); numerous 100%-compliant circuits were harvested; and the run was manually monitored and manually terminated.

3.8 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 66 processing nodes (each containing a 533-MHz DEC Alpha microprocessor and 64 megabytes of RAM) arranged in a two-dimensional 6 $\times$ 11 toroidal mesh. This computer was approximately eight time faster than the 64-node parallel computer with 80 MHz PowerPC microprocessors at each note (used for most of the work in Koza, Bennett, Andre, and Keane 1999a). The new 66-node system has a DEC Alpha type 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 was used with a population size of Q = 2,000 at each of the D = 66 demes (semi-isolated subpopulations). Generations are asynchronous on the nodes(Andre and Koza 1996). On each generation, four boatloads of emigrants, each consisting of B = 2% (the migration rate) of the node's subpopulation (selected probabilistically on the basis of fitness) were dispatched to each of the four toroidally adjacent processing nodes. Additional details about both computers are found in Koza, Bennett, Andre, and Keane 1999s and in another GECCO-99 paper (Bennett, Koza, Shipman, and Stiffelman 1999).

4 Results for NAND

The best-of-generation circuit (figure 3) from generation 17 has five transistors, five resistors. This 100% compliant circuit scores 91 (out of 91) hits and has a fitness of 7.85.

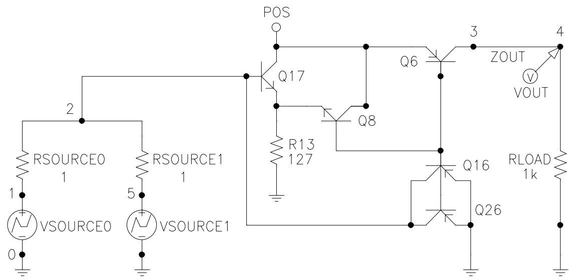

Figure 3 Evolved NAND circuit.

Figure 4 shows the behavior of this best-of-generation circuit from generation 17 in the time domain for the 91 fitness cases.

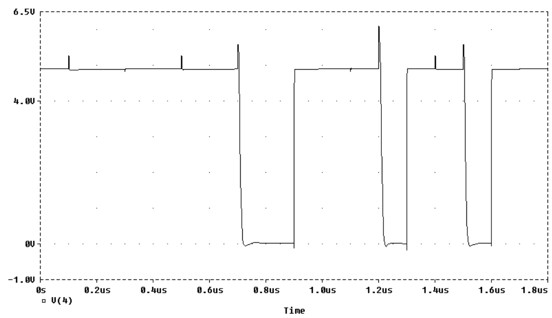

Figure 4 Behavior of evolved NAND circuit.

Figure 5 shows a textbook TTL NAND circuit (from Wakerly 1990) consisting of five transistors, four resistors, and three diodes.

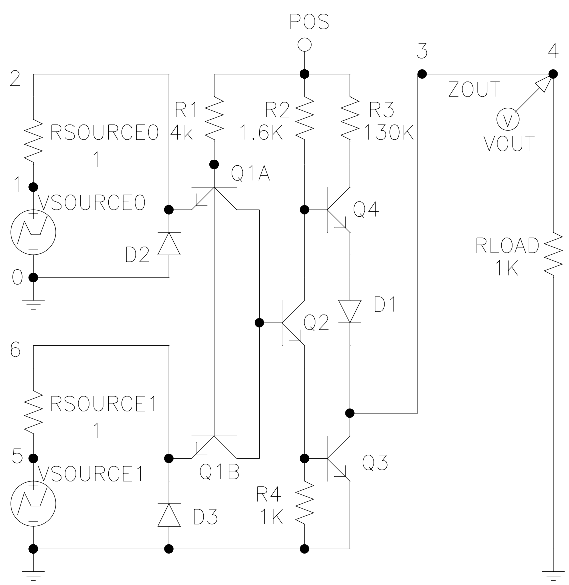

Figure 5 Textbook TTL NAND circuit.

5 Preparatory Steps for DAC Circuit

In this section and section 6, we evolve a digital-to-analog converter (DAC). The basic function of a DAC is the conversion of binary numbers into analog voltages.

The preparatory steps for the DAC are the same as for the NAND circuit, except as mentioned below.

5.1 Initial Circuit for DAC

Figure 6 shows a three-input, one-output initial circuit. The embryo consists of three modifiable wires Z0, Z1, and Z2. The test fixture receives three incoming signals, each with a 1 Ω source resistor. The circuit also has a voltage probe point VOUT and a load resistor RLOAD (whose value is 1,000 Ω).

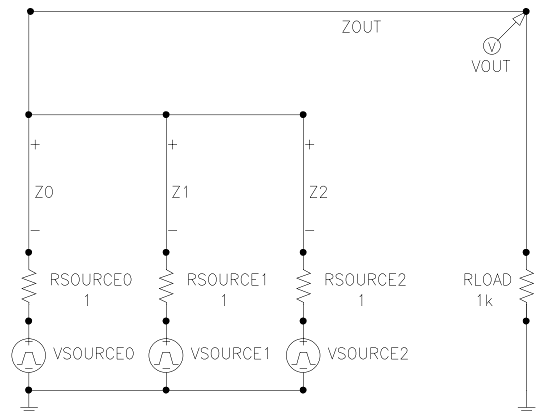

Figure 6 Initial circuit for three-input DAC.

5.2 Fitness for DAC

SPICE is instructed to perform two separate transient (time domain) analyses.

All three of the inputs to the initial circuit are presented with nine 100-µs digital signals. All three inputs are zero during the first 100 µs. The next eight input triples represent all possible combinations of the three input signals. The first group of signals represent a counting from 0 up to 7 while the second group represents a counting from 7 down to 0. Each signal is sampled every 20 µs (i.e., five sample points per 100 µs). This approach yields a total of 92 fitness cases (46 fitness cases in each group). Figure 7 shows the first of the two groups of fitness cases.

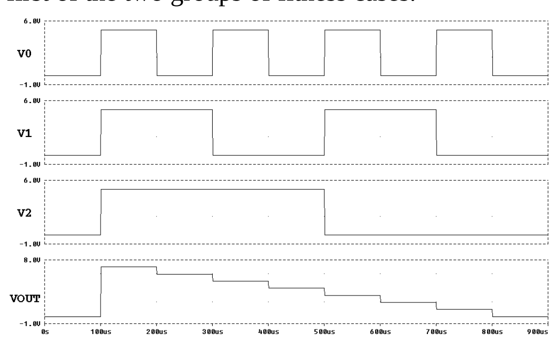

Figure 7 First group of fitness cases in the time domain for the DAC circuit.

5.2.1 Our First Fitness Measure Produced Glitches

We first tried this problem using a fitness measure consisting of the sum, over the 92 fitness cases, of the weighted absolute value of the difference between the actual output voltage at the probe point VOUT and the desired output voltage. The desired voltages ranges from 0 volts (for the 000 input) to 7 volts (for the 111 input). If the voltage exactly equals the desired voltage of VSOURCE0 plus two times VSOURCE1 plus four times VSOURCE2, the deviation is 0. If the voltage is within 0.25 volts of the desired voltage, the absolute value of the deviation from the desired output voltage is weighted by a factor of 1.0. If the voltage is outside this range, the absolute value of the deviation is weighted by a factor of 10.0. The number of hits is the number of fitness cases for which the output is within 0.25 volts of the desired voltage.

Genetic programming successfully evolved a circuit with 92 hits (out of 92) and a very low value of fitness using this first fitness measure. Figure 8 shows the behavior of this circuit using this first fitness measure.

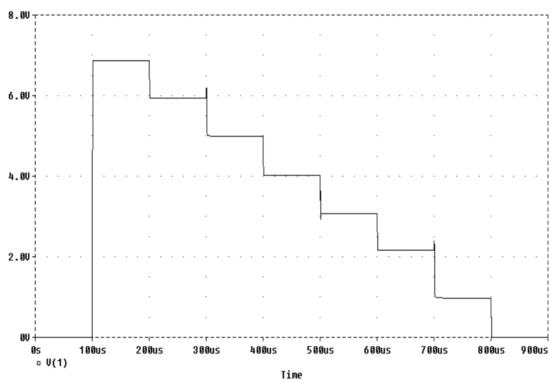

Figure 8 Glitch-ridden behavior of best-of-run evolved DAC circuit using the first fitness measure.

However, as can be seen in figure 8, there were three narrow spikes (of height was between 0.1 and 0.2 volts) at the boundaries between three particular fitness cases, namely 110 to 101, 100 to 011, and 010 to 000. Note that these fitness cases are precisely the ones where two or more of the three bits undergo change. An ideal distortion-free DAC would instantaneously create an output voltage that is proportional to the input voltage. However, in practice, when the digital input changes from one value to another, the output of the DAC reaches its new value after a time delay. The flaw that we observed is called a *glitch* (Song and Harjani 1995). Glitches are evaluated in terms of the size of the undesired voltage spike multiplied by its time duration. Glitches are caused by small time differences that occur when some current sources turn off while others are turn on. Glitches are typically eliminated by adding special *deglitcher* circuitry to DACs. Of course, nothing in our first fitness measure addressed the question of glitches. Genetic programming evolved a circuit that was exactly what we asked for. However, what we asked for was not what we wanted!

5.2.2 Our Second Fitness Measure Produced Even Greater Glitches

In the hope of eliminating the glitches permitted by the first fitness measure, we constructed a second fitness measure.

In simulating circuits, SPICE internally identifies particular points in time where the output voltage "turns." The number of turns, of course, varies from circuit to circuit. Our second fitness measure was the sum, for each of SPICE's internally created turn-defining points, of the areas of the trapezoids between the curve representing the desired output voltage and the curve representing the actual output voltage.

Unfortunately, the magnitude of the glitches in the best-of-run circuit that was evolved using this second fitness measure were as large as 1 volt — far larger that the magnitude of the glitches that we were trying to eliminate. The second fitness measure tolerated these larger glitches because they were very narrow (and hence occupied very little total area). As before, the glitches were at the boundaries where two or more bits changed. Thus, we again got what we asked for, but not what we wanted.

5.2.3 A Crossover of Our First Two Fitness Measure Eliminated the Glitches

Finally, we constructed a third fitness measure by combining (crossing over) some of the features of the first and second fitness measures. (It is the results of using this third fitness measure that we report in detail below).

The third fitness measure was the sum, over SPICE's turn-defining points, of the weighted absolute value of the difference between the actual output voltage at the probe point VOUT and the desired output voltage. As in the first fitness measure, if the voltage exactly equals the desired voltage of VSOURCE0 plus two times VSOURCE1 plus four times VSOURCE2, the deviation is 0. Also, if the voltage is within 0.25 volts of the desired voltage, the absolute value of the deviation from the desired output voltage is weighted by a factor of 1.0. And, as before, if the voltage is outside this range, the absolute value of the deviation is weighted by a factor of 10.0.

5.3 Control Parameters for DAC

The population size, *M*, is 330,000. A maximum size of 300 points (functions and terminals) was established for each branch of each circuit-constructing program tree. Other control parameters were the ones that are used previously for the robot controller problem in chapter 48 and appendix D of Koza, Bennett, Andre, and Keane 1999a.

6 Results for DAC Circuit

The best-of-generation circuit (figure 9) from generation 139 has six transistors, five capacitors, and 10 resistors. This circuit has a fitness of 104.8.

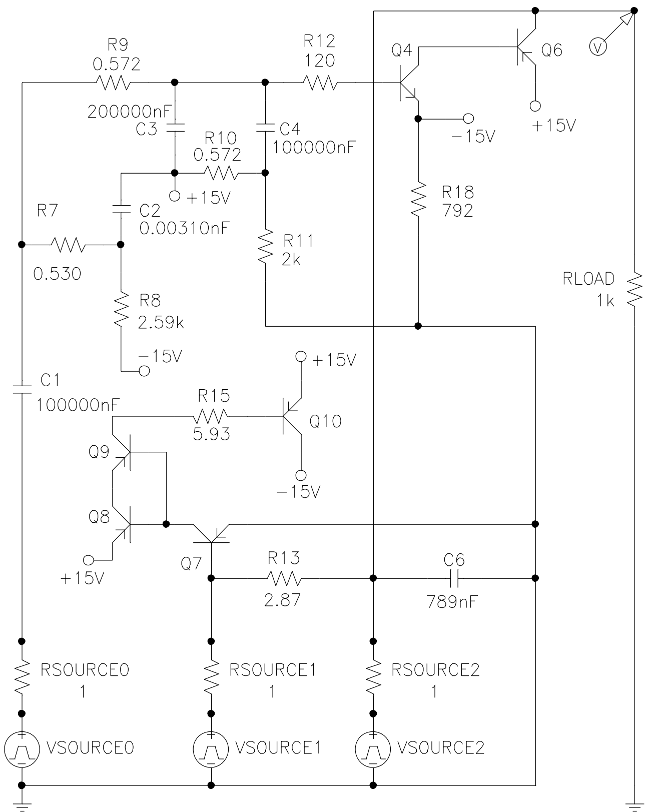

Figure 9 Best-of-run evolved DAC circuit from generation 139.

Figure 10 shows the behavior of this best-of-generation circuit from generation 139 in the time domain for the second group of 46 fitness cases (counting down from 7 to 0).

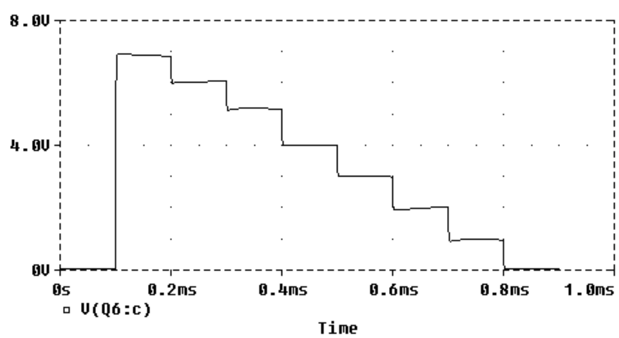

Figure 10 Behavior of best-of-run evolved DAC circuit from generation 139.

7 Preparatory Steps for Two-Instruction ALU Circuit

The preparatory steps are the same as for the DAC circuit, except as mentioned below.

7.1 Fitness for Two-Instruction ALU

SPICE is instructed to perform a transient (time domain) analysis.

All three of the inputs to the initial circuit are presented with 64 10-μs digital signals. Each signal is sampled every 2 μs (i.e., five sample points per 10 μs). Thus, there are 321 fitness cases. The 64 input triples represent all possible transitions between each of the eight possible combinations of the three input signals. The first three parts of figure 11 shows the fitness cases for this problem while the bottom part of the figure shows the correct answer.

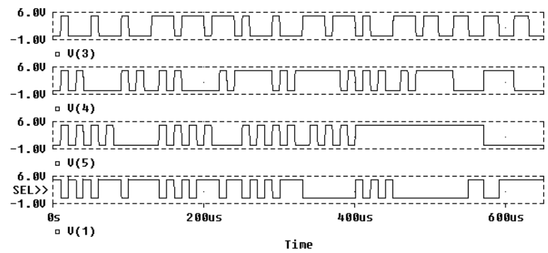

Figure 11 Fitness cases for the two-instruction arithmetic logic unit circuit.

The fitness of a circuit is the sum, over the 321 fitness cases, of the weighted absolute value of the difference between the actual output voltage at the probe point VOUT and the desired output voltage.

If the voltage exactly equals the desired voltage, the deviation is 0. If the voltage is within 0.40 volts of the desired voltage, the absolute value of the deviation from the desired output voltage is weighted by a factor of 1.0. If the voltage is outside this range, the absolute value of the deviation is weighted by a factor of 10.0.

7.2 Control Parameters for Two-Instruction ALU

The population size, *M*, is 1,320,000. A maximum size of 300 points (functions and terminals) was established for each branch of each circuit-constructing program tree.

8 Results for Two-Instruction ALU Circuit

The best-of-generation circuit (figure 12) from generation 33 has three transistors and two resistors. This 100% compliant circuit scores 321 (out of 321) hits and has a fitness of 215.6.

Figure 13 shows the behavior of this best-of-generation circuit from generation 33 in the time domain for the 321 fitness cases.

9 Conclusion

This paper has shown three digital circuits that were evolved by means of genetic programming, namely a five-transistor 100-nanosecond NAND circuit, a three-input digital-to-analog converter (DAC), and a two-instruction arithmetic logic unit (ALU) circuit.

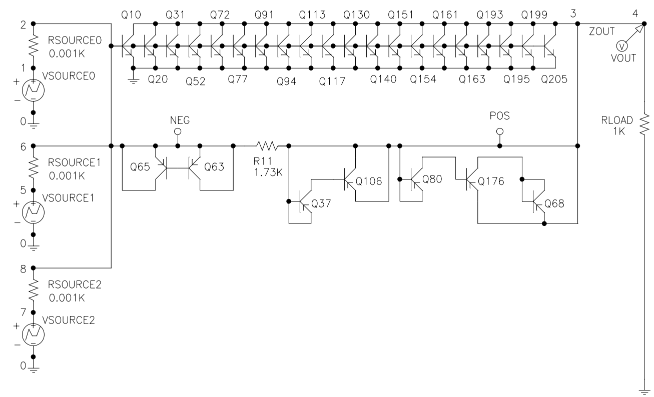

Figure 12 Evolved two-instruction arithmetic logic unit circuit from generation 33.

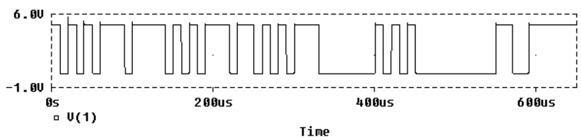

Figure 13 Behavior of evolved two-instruction arithmetic logic unit circuit from generation 33.

References

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.

Banzhaf, Wolfgang, Poli, Riccardo, Schoenauer, Marc, and Fogarty, Terence C. 1998. *Genetic Programming: First European Workshop. EuroGP'98. Paris, France, April 1998 Proceedings. Paris, France. April l998*. Lecture Notes in Computer Science. Volume 1391. Berlin, Germany: Springer-Verlag.

Bennett, Forrest H III, Koza, John R. Shipman, James, and Stiffelman, Oscar. 1999. Building a parallel computer system for $18,000 that performs a half peta-flop per day. Elsewhere in these GECCO-99 conference proceedings.

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., Banzhaf, Wolfgang, Chellapilla, Kumar, Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max H., Goldberg, David E., Iba, Hitoshi, and Riolo, Rick. (editors). 1998. *Genetic Programming 1998: Proceedings of the Third Annual Conference*. San Francisco, CA: Morgan Kaufmann.

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

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

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., 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.

Langdon, William B. 1998. *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!* Amsterdam: Kluwer.

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.

Song, Bang-Sup and Harjani, Ramesh. 1995. In Chen, Wai-Kai (editor). *The Circuits and Filters Handbook*. Boca Raton, Fl: CRC Press. Pages 2072 - 2127.

Spector, Lee, Langdon, William B., O'Reilly, Una-May, and Angeline, Peter (editors). 1999. *Advances in Genetic Programming 3*. Cambridge, MA: The MIT Press.

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

Wakerly, John F. 1990. *Digital Design Principles and Practices*. Englewood Cliffs, NJ: Prentice Hall.