A PARALLEL GENETIC ALGORITHM FOR AUTOMATED ELECTRONIC CIRCUIT DESIGN

Jason D.Lohn*, Silvano P. Colombano*, Gary L. Haith†, Dimitris Stassinopoulos*
*Computational Sciences Division | †Recom Technologies, Inc.
NASA Ames Research Center
Moffett Field, CA 94035-1000
Email: jlohn@ptolemy.arc.nasa.gov Tel: (650) 604-5138

Abstract— We describe a parallel genetic algorithm (GA) that automatically generates circuit designs using evolutionary search. A circuit-construction programming language is introduced and we show how evolution can generate practical analog circuit designs. Our system allows circuit size (number of devices), circuit topology, and device values to be evolved. We present experimental results as applied to analog filter and amplifier design tasks.

1 Introduction

Although the underlying concepts of using simulated evolution to manipulate hardware are decades old, it is only in recent years that research in this area has attracted significant interest [14, 5, 15, 16]. The nascent field of *evolvable hardware* studies how simulated evolution can reconfigure, adapt, and design hardware structures in an automated manner. The field is almost exclusively concerned with electronic circuitry, but application areas where other types of physical structures are designed or adapted by artificial evolution certainly fall within the purview of evolvable hardware (e.g., designs of trusses, antennas).

Analog circuits are of great importance in electronic system design since the world is fundamentally analog in nature. While the amount of digital design activity far outpaces that of analog design, most digital systems require analog modules for interfacing to the external world. Techniques for analog circuit design automation began appearing about two decades ago (e.g., [18]). Efforts using techniques from evolutionary computation have appeared over the last few years. These include the use of genetic algorithms (GAs) [6] to select filter component sizes [7], to select filter topologies [4], and to design operational amplifiers using a small set of topologies [10]. Various analog filter design problems have been solved using genetic programming, and an overview of these techniques, including eight analog circuit synthesis problems, is found in [9].

The remainder of the paper is as follows. First we discuss the genetic algorithm and the parallelization technique implemented in our software. Then we present the genetic representation of analog circuits and describe the genetic algorithm that is used. The design tasks and the experimental results from our evolutionary design program are then presented. We conclude with a brief discussion of our results.

2 Genetic Algorithms

Genetic algorithms are a type of trial-and-error search technique that are guided by principles of Darwinian evolution. Just as the genetic material of two living organisms can intermix to produce

offspring that are better adapted to their environment, GAs expose genetic material, frequently strings of 1s and 0s, to the forces of artificial evolution: selection, mutation, recombination, etc. GAs start with a pool of randomly-generated candidate solutions which are then tested and scored with respect to their utility. Solutions are then bred by probabilistically selecting high quality parents and recombining their genetic representations to produce offspring solutions. Offspring are typically subjected to a small amount of random mutation. After a pool of offspring is produced, this process iterates until a satisfactory solution is found or an iteration limit is reached. Genetic algorithms have been applied to a wide variety of problems in many fields, including chemistry, biology, and many engineering disciplines.

Parallelized versions of genetic algorithms (GAs) are popular primarily for three reasons: the GA is an inherently parallel algorithm, typical GA applications are very compute intensive, and powerful computing platforms, especially Beowulf-style computing clusters, are becoming more affordable and easier to implement. In addition, the low communication bandwidth required allows the use of inexpensive networking hardware such as standard office ethernet.

There are many styles of parallelism used in implementing parallel GAs [1]. We currently use a master-slave [3] type of parallel genetic algorithm in our work on evolving analog circuits (Fig 1). Also called a "processor farm," this style of parallelism automatically balances the computational load among processors within each generation. The "master" processor maintains the population, performs the genetic operations, and distributes the fitness evaluations to the nodes (the most time consuming part). Each node processor runs a loop consisting of two main functions: ask for a packet of individuals to process, and then process those individuals. Processing individuals requires that each individual is decoded (into a circuit), simulated (using a circuit simulator), and scored using a fitness function.

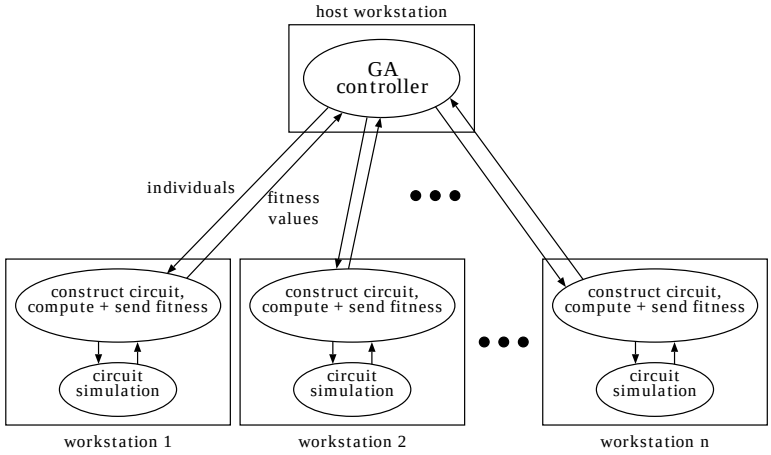


Fig. 1: Parallelization of genetic algorithm showing overall tasks. Running circuit simulations in parallel greatly reduces runtime since nearly all of the processing time is spent simulating circuits.

In Figure 2 we see how load balancing is achieved in a system of seven processing nodes. First, assume all processors are busy doing fitness evaluations. Because of the nondeterministic nature of genetic algorithms, fitness evaluations will take differing amounts of time. So, for example, when node 5 finishes first (Figure 2(a)), it will contact the master node, make a connection, send its results, and then receive a new packet of individuals to process (Figure 2(b)). The

slave nodes will then return to full utilization (Figure 2(c)). This "farm out work on demand" algorithm ensures that all nodes are kept busy. The only time when processors can be left idle is when a synchronization event occurs. For example, at the end of a generation, processors may sit idle if there are no more individuals remaining in the current generation to process. After all nodes have finished evaluating individuals in the current generation, the master processor will create a new population and then begin the process of "farming out the work" again. To alleviate synchronization events at the end of each generation, the GA can be configured to run as a steady-state GA whereby there are no generations. Other parallel GAs exist where synchronization events are not an issue.

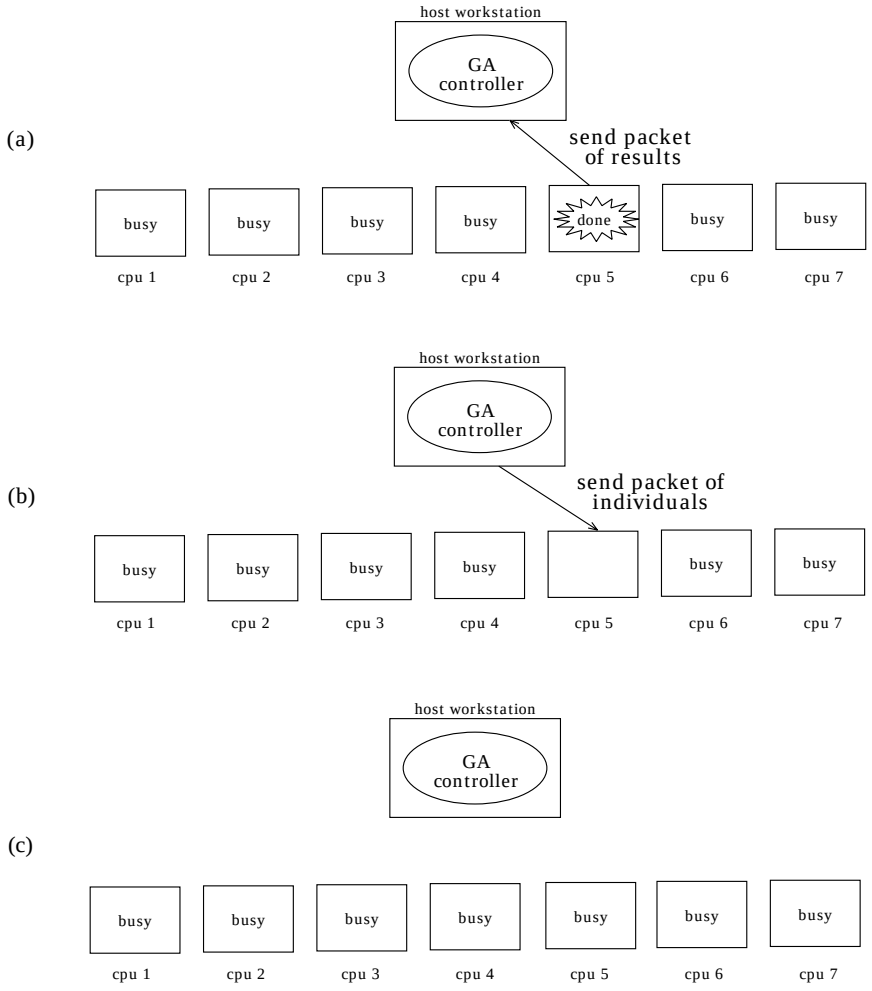


Fig. 2: Farming out work to the node balances the compute load between synchronization points.

Recently our group obtained a Beowulf computing cluster [17] consisting of commodity-class workstations running the Linux operating system. Using 32 Pentium III CPUs our system achieves a SPECfp95 benchmark value of 500 very economically. We make use of dual processor workstations which are becoming a commodity item (previously, only single processor workstations were commodity-priced). Linux is free and inexpensive to maintain, so the software costs are minimal. Because our genetic algorithms evaluation functions typically do not need large datasets, each of our nodes runs diskless. This has several advantages. First, it reduces the cost, power

consumption, and heat output of each node. Second, it alleviates maintaining n copies of the operating system (where n is the number of nodes). Third, with no hard disks, the system is not susceptible to disk failures which would typically crash the operating system and leave the node useless until a new disk was installed. Fourth, there are no backups to make (only the server node requires a backup procedure). For diagnostic messages, and applications where local data storage is needed, each node can write data to the server's disk which is visible via a network mount. Our cluster uses standard office ethernet since the interprocessor communication load is low. It uses its own network switch on a private Class C internet subnet, so that the cluster will still operate even if the building LAN experiences problems. In addition to being diskless, each node workstation is also without a monitor, keyboard, and mouse which also reduces the cost. Since all of the standard Internet utilities are available (e.g., telnet, rlogin), one can remotely login to the nodes in lieu of having a keyboard, mouse, and monitor plugged into each node.

3 Circuit Representation in the GA

In designing an effective circuit representation for use in evolutionary search, the following properties are among the most desirable. First, the representation should permit any circuit or at least a wide range of circuits to be represented. If it is known *a priori* that certain topologies are well suited to a specific design task, topological restrictions inherent in the representation may be beneficial since the search space will be reduced. Conversely, not having this limitation may bring to light novel designs that human designers have never envisioned. Second, the genotype conversion algorithm (the circuit constructing process) should run as fast as possible. Clearly if numerous traversals of the circuit graph structure are required in order to guarantee a valid circuit graph, the performance hit will be commensurate. For an n-component circuit, a reasonable upper bound would be $O(n)$. Third, the representation should be syntactically closed so that genetic operators do not create invalid circuit graphs[1] from those that are valid. The circuit representation we present here was designed to have these properties.

Circuit designs are constructed by an automaton that is programmed via a set of low-level instructions. This automaton is programmed in a small "language" designed for building circuits. In its current incarnation, the language contains only component-placing instructions (e.g., control instructions are not included). This language has the desirable property that virtually all possible sequences of instructions result in a valid electrical circuit. This property is important because it greatly limits the "out-of-bounds" regions of the search space containing invalid circuit graphs. Thus, evolutionary search will spend nearly all its time generating valid circuit graphs. While this is a beneficial, non-trivial achievement, we do lose the ability to generate every possible circuit topology. This is not considered a drawback for the circuit types we investigated since a vast number of topologies and existing circuit designs could be encoded using this approach.

Each instruction places a circuit component and directs the movement of the automaton. The five basic instruction types are: x-move-to-new, x-cast-to-previous, x-cast-to-ground, x-cast-to-input, x-cast-to-output, where x can be replaced by R (resistor), C (capacitor), L (inductor),

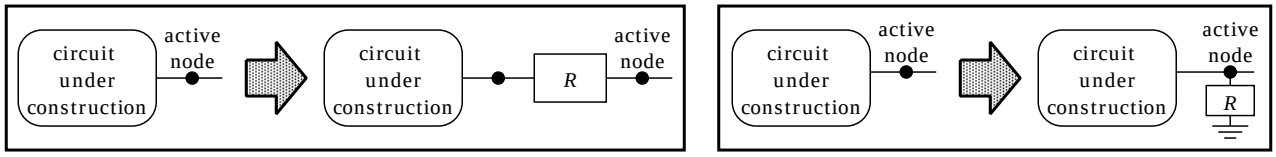
[1]Note that a graph could be a valid circuit graph, yet not make sense as an electrical circuit – for example, dissimilar voltage sources connected in parallel.

or transistor configuration. In a circuit design task involving only inductors and capacitors (an LC circuit), ten opcodes would be available to construct circuits (five for capacitors and five for inductors).

The meanings of each instruction are summarized in Table 1. The move-to-new instruction places one end of a component at the active node and the other at a newly created node (the "active" node is the current location of the automaton). The newly created node then becomes the active node. The cast-to instructions place one end of the component at the active node and the other at either the ground, input, output, or previously-created node. After executing a cast-to instruction, the automaton remains at the active node. The input and output nodes are the overall input and output nodes of the circuit as opposed to the input and output of the placed component. Illustrations of two instructions that place resistors are shown in Fig. 3.

Instruction	Outgoing Node	Active Node
x-move-to-new	new node	becomes new node
x-cast-to-previous	previous node	unchanged
x-cast-to-ground	ground node	unchanged
x-cast-to-input	input node	unchanged
x-cast-to-output	output node	unchanged

Table 1: Summary of opcode types used in current system. x denotes the component type: resistor, capacitor, inductor, or transistor configuration.



result of **move-to-new** instruction

(a)

result of **cast-to-ground** instruction

(b)

Fig. 3: Effect of placing a resistor with (a) move-to-new, and (b) cast-to-ground instructions.

The circuit is constructed by the automaton inside of a template circuit. The design tasks presented here use a template having one input and one output terminal as shown in Fig. 4. An ideal voltage source v_s is connected to ground and to a source resistor R_s. The circuit's output voltage is taken across a load resistor R_l.

The lists of instructions manipulated by the GA are variable-length lists so that the size of the circuit can be evolved. When the automaton reaches the last component to place in the circuit, we arbitrarily chose to have the last active node connected to the output terminal by a wire (accomplished by connection of a $1\mu\Omega$ resistor). By doing so, we eliminate unconnected branches.

As assembly language instructions are mapped to opcodes, our circuit-placing instructions are mapped to bytecodes. Instructions are represented by up to four bytecodes. For instructions that take a component value as an argument, the first byte is the instruction, and the next three

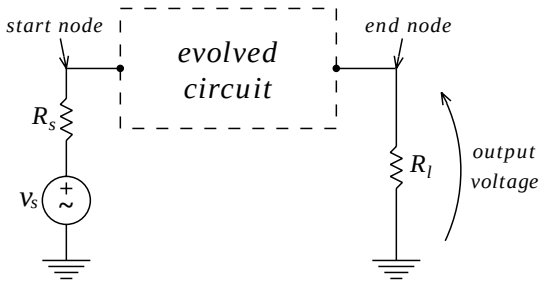


Fig. 4: Template circuit: the evolved circuit is located between fixed input and output terminals. v_s is an ideal voltage source, R_s is the source resistance, R_l is the load resistance.

represent the component value (resistance, capacitance, and inductance values). For transistors, component values are not needed. Using three bytes allows the component values to take on one of 256^3 values, a sufficiently fine-grained resolution. The raw numerical value of these bytes was then scaled into a reasonable range, depending on the type of component. Resistor values were scaled sigmoidally between 1 and 100K ohms using $1/(1+\exp(-1.4(10x-8)))$ so that roughly 75% of the resistor values were biased to be less than 10K ohms. Capacitor values were scaled between approximately 10 pF and 200 μF and inductors between roughly 0.1 mH and 1.5 H.

4 Design Tasks

The design tasks considered in this paper are analog circuits for filtering and amplification applications. A low-pass filter is a circuit that allows low frequencies to pass through it, but stops high frequencies from doing so. In other words, it is frequency selective in that it "filters out" frequencies above a specified frequency. The unshaded area in Fig. 5(a) depicts the region of operation for low-pass filters. Below the frequency f_p the input signal is passed to the output, potentially reduced (attenuated) by K_p decibels (dB). This region is known as the passband. Above the frequency f_s, in the region is called the stopband, the input signal is markedly decreased by K_s decibels. Between the passband and stopband the frequency response curve transitions from low to high attenuation. The parameter located in this region, f_c, is known as the cutoff frequency.

The amplifier design task chosen was the inverting operational amplifier. Such a circuit has found wide application and is considered one of the workhorses of analog circuit design. Figure 5(b) shows the symbol and connections for an ideal inverting amplifier. This circuit generates an output voltage (v_o) that consists of the input voltage (v_i) multiplied by a gain factor, A. Voltage gain is thus equivalent to v_o/v_i. It is common to express gain values in decibels (dB) using $20\log_{10}(A)$. Amplifiers may be either inverting or non-inverting, where an inverted output signal has a 180° phase shift compared to the input. The dc gain of the amplifier refers to the gain when only constant voltage/current sources are applied. The linearity of the gain is the degree to which the gain remains constant across input voltages: ideally the voltage transfer characteristic (v_o vs. v_i) should be linear. The dc component that shifts the entire signal up or down is called the dc bias of the circuit. Power dissipation is the amount of power used by the circuit and is indicative of the amounts of current flowing in the circuit. For simple amplifiers, there

are publications available that catalog many designs. Since there are numerous parameters in amplifier design (e.g., input/output impedance, power dissipation, distortion, common-mode rejection, power supply rejection), the design task can become quite challenging and typically requires an experienced designer. For the amplifier design experiment below, we take into account four objectives: dc gain, linearity of gain, dc bias, and power dissipation.

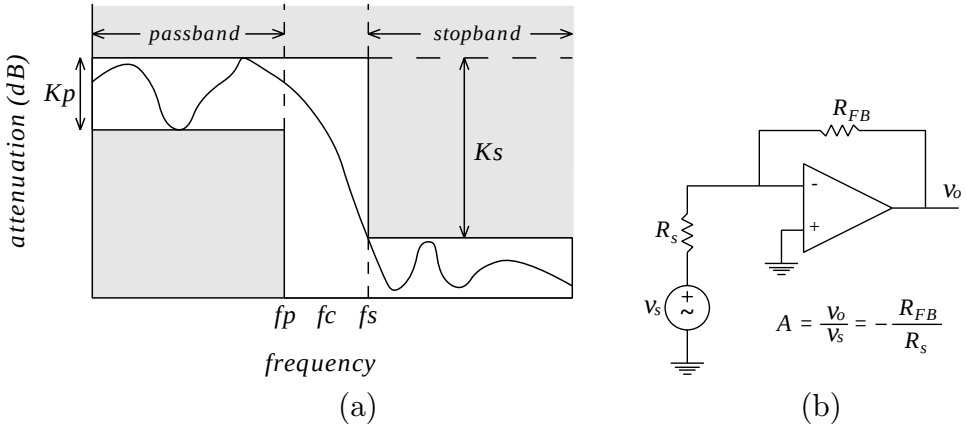


Fig. 5: (a) Low-pass filter terminology and specifications. The shaded regions represent out-of-specification areas. An example frequency response curve that meets specifications is shown. (b) Ideal inverting amplifier showing how gain is set by the ratio of the feedback to source resistor.

5 Experimental Results

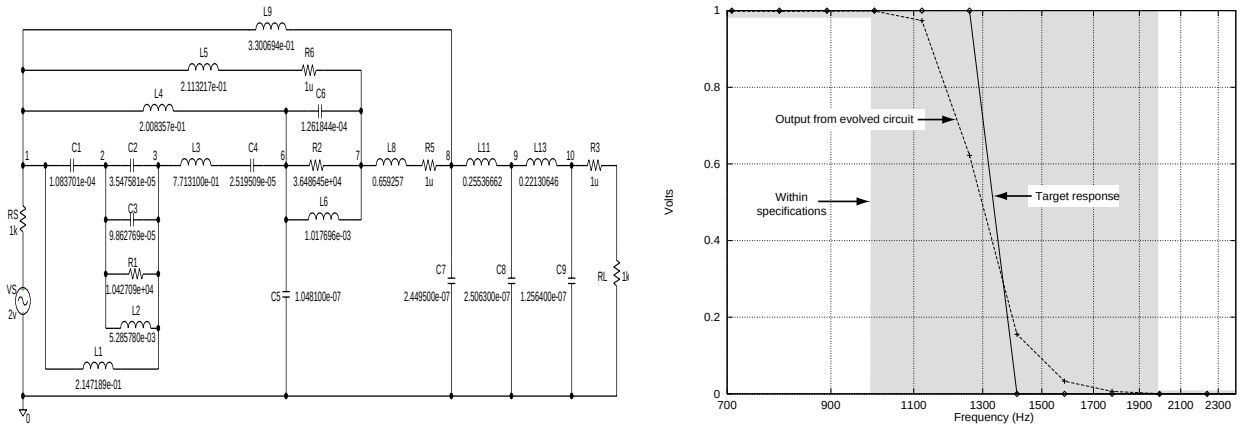
In this section we present experimental results for two design tasks: an analog filter and an analog amplifier. In each case, a set of specifications was defined and a circuit was evolved that satisfied those specifications.

5.1 Filter Design Task

In the filter design experiment, 10 GA runs were performed. Below we present the circuit having the highest fitness value across all runs. Fitness was calculated to promote the regression of the evolved circuit's frequency response toward that of the target. Error values were computed as the absolute value of the difference of the individual's output and the target output. These error values were summed across evaluation points to arrive at a fitness value.

The target specifications for this experiment was: $f_p = 1000$ Hz, $f_s = 2000$ Hz, $K_p = 0.01$ dB, $K_s = 63.50$ dB. These specifications are similar to the fifth-order elliptic filter described in [9]. In that work, the evolved LC circuit satisfies $K_p = 0.3$ dB and $K_s = 60$ dB. Another evolved low-pass filter circuit [19] had the same stopband and passband frequencies, but less demanding attenuation specifications ($K_p = 1.6$ dB and $K_s = 24.8$ dB). The evolved circuit is shown in Fig. 6(a) and its frequency response is seen in Fig. 6(b). Micro-ohm resistors were added as a

convergence aid for the circuit simulator, and can be ignored for analytical purposes. This circuit was found in generation 997 of a run that had a population size of 1000.



(a) (b)

Fig. 6: (a) Evolved circuit satisfying target specifications. (b) Frequency response for evolved analog filter.

5.2 Amplifier Design Task

For the amplifier design task, 10 GA runs were performed and we present the highest performance circuits found across all runs. The goal was to design an inverting amplifier capable of a dc voltage gain up to a maximum of either 100 dB, while minimizing dc bias and maximizing linearity over the dc gain. Population size was set to 1200 individuals, and each run proceeded for 5000 generations, giving a total of 6 million circuit evaluations per run. For an ideal inverting amplifier (as shown in Fig. 5(b)), the magnitude of the gain of the amplifier is simply R_{FB}/R_S, where R_S is the source resistor. Fitness was calculated in a manner similar to the work on amplifiers in [9]. An error value is computed as the sum of the dc gain penalty (the target gain minus the observed gain), the dc bias (zero dc bias is ideal), and the degree to which the dc gain is linear.

The evolved amplifier having the best performance had a dc gain of 85.41 dB $(18,642.33)$. Figure 7 shows the schematic for this circuit. It was found in generation 3635, and had a dc bias of 5.44 volts and a power dissipation of 8.17 watts. The dc current delivered to the load is mostly supplied by the 15 volt battery attached to the collector of transistor Q7. Transistor Q7 is conducting with the sum of its base and collector currents flowing out of its emitter. Q7's base current of 13 mA is supplied by transistor Q6.

Input signal inversion and amplification are seen in Figure 8(a) which shows the time domain response to an ac input of 1 microvolt at 1 kHz. The circuit has a flat-band gain of 85.46 dB and a 3 dB bandwidth of 282.8 kHz (Figure 8(b)). The 3 dB bandwidth is significantly better than the previous amplifier.

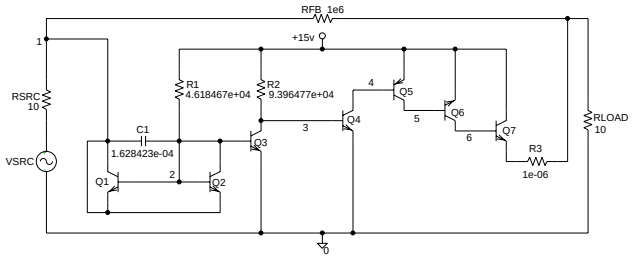
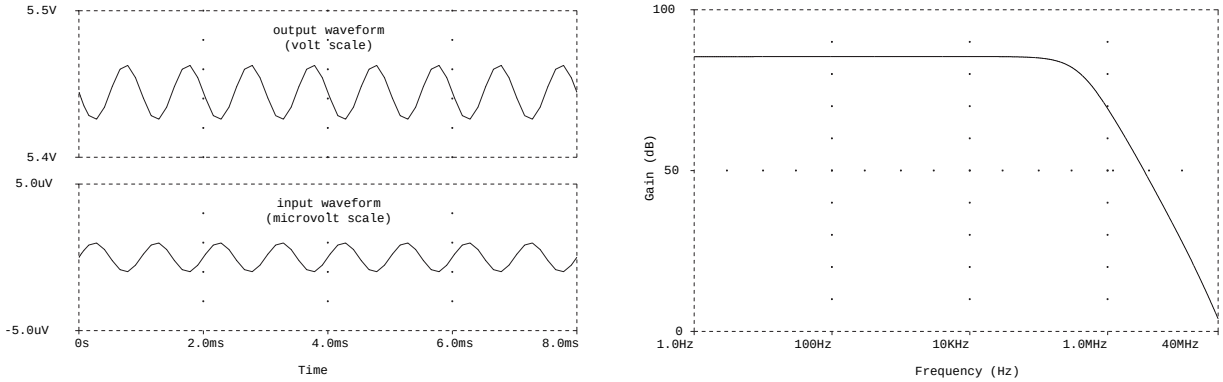


Fig. 7: Circuit schematic of evolved 85 dB amplifier.



(a) (b)

Fig. 8: Small signal behavior of 85 dB evolved amplifier: (a) time domain input waveform is 1 kHz (bottom) which is inverted and amplified (top); (b) frequency response showing a flatband gain of 85.46 dB.

6 Discussion

Through the use of a parallel genetic algorithm, we have presented encouraging results of an evolvable hardware system capable of automatically designing analog circuits. We presented a master-slave genetic algorithm that is able to keep processors fully utilized within generations. We showed that a linear circuit representation and evolutionary search can automatically produce circuit designs of low to medium difficulty in two applications. Detailed simulations of the evolved designs suggest that all are electrically well behaved and thus suitable for physical implementation. The circuit representation method devised permits a wide range of circuits to be constructed, and results in a construction process that is unburdened with repair operations. To gain performance on par with circuits designed by engineers, it will be necessary to place further constraints into the fitness functions. For example, practical amplifiers are typically judged by a dozen or so specifications. To evolve an amplifier that would perform as well would require using a multiobjective fitness function that accounts for each specification. With the recent addition of a Beowulf computing cluster, we are optimistic that adding further design constraints will remain tractable for evolutionary design.

References

[1] E. Cantu-Paz, "A Survey of Parallel Genetic Algorithms," *Calculateurs Paralleles,* vol. 10, no. 2, Paris: Hermes, 1998.

[2] G. Gielen, W. Sansen, *Symbolic Analysis for Automated Design of Analog Integrated Circuits,* Boston, MA: Kluwer, 1991.

[3] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, (Addison-Wesley, Reading, Mass, 1989).

[4] J.B. Grimbleby, "Automatic Analogue Network Synthesis using Genetic Algorithms," *Proc. First Int. Conf. Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA),* 1995, pp. 53-58.

[5] T. Higuchi, M. Iwata, Eds., *Evolvable Systems: From Biology to Hardware*, Proc. of the First International Conference on Evolvable Systems, (Lecture Notes in Computer Science), vol 1259, Berlin: Springer-Verlag, 1997.

[6] J.H. Holland, *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor, 1975.

[7] D.H. Horrocks, Y.M.A. Khalifa, "Genetically Derived Filters using Preferred Value Components," *Proc. IEE Colloq. on Linear Analogue Circuits and Systems,* Oxford, UK, 1994.

[8] L.P. Huelsman, *Active and Passive Analog Filter Design*, New York: McGraw-Hill, 1993.

[9] J.R. Koza, F.H. Bennett, D. Andre, M.A. Keane, F. Dunlap, "Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming," *IEEE Trans. on Evolutionary Computation*, vol. 1, no. 2, July, 1997, pp. 109–128.

[10] M.W. Kruiskamp, *Analog Design Automation using Genetic Algorithms and Polytopes,* Ph.D. Thesis, Dept. of Elect. Engr., Eindhoven University of Technology, Eindhoven, The Netherlands, 1996.

[11] J.D. Lohn, S.P. Colombano, "Automated Analog Circuit Synthesis using a Linear Representation," *Proc. of the Second Int'l Conf on Evolvable Systems: From Biology to Hardware*, Springer-Verlag, Berlin, 1998, pp. 125-133.

[12] J.D. Lohn, S.P. Colombano, "A Circuit Representation Technique for Automated Circuit Design," *IEEE Trans. on Evolutionary Computation*, vol. 3, no. 3, 1999, pp. 205-219.

[13] E.S. Ochotta, R.A. Rutenbar, L.R. Carley, "Synthesis of High-Performance Analog Circuits in ASTRX/OBLX," *IEEE Trans. Computer-Aided Design,* vol. 15, pp. 273–294, 1996.

[14] E. Sanchez and M. Tomassini, Eds., *Toward Evolvable Hardware: The Evolutionary Engineering Approach*, (Lecture Notes in Computer Science), vol 1062, Berlin: Springer-Verlag, 1996.

[15] M. Sipper, D. Mange, A. Perez-Uribe, Eds., *Evolvable Systems: From Biology to Hardware*, Proc. of the Second International Conference on Evolvable Systems, (Lecture Notes in Computer Science), vol 1478, Berlin: Springer-Verlag, 1998.

[16] M. Sipper, D. Mange, Eds., Special Issue on Evolvable Hardware *IEEE Transactions on Evolutionary Computation*, vol 3, no 3, 1999.

[17] T. Sterling, J. Salmon, D. Becker, D. Savarese, *How to Build a Beowulf: A Guide to Implementation and Application of PC Clusters,* Cambridge, MA: MIT Press, 1999.

[18] G.J. Sussman, R.M. Stallman, "Heuristic Techniques in Computer-Aided Circuit Analysis," *IEEE Trans. Circuits and Systems,* vol. 22, 1975.

[19] R.S. Zebulum, M.A. Pacheco, M. Vellasco, "Comparison of Different Evolutionary Methodologies Applied to Electronic Filter Design," *1998 IEEE Int. Conf. on Evolutionary Computation*, Piscataway, NJ: IEEE Press, 1998, pp. 434–439.