Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 36 (2015) 17 – 22

www.elsevier.com/locate/procedia

25th CIRP Design Conference

Genetic Algorithm Software System for Analog Circuit Design

Miri Weiss Cohen*, Michael Aga, Tomer Weinberg

Software Engineering Department, Braude College of Engineering, Karmiel, Israel

* Corresponding author. Tel.:97249901758; fax: +97249901982. *E-mail address:* miri@braude.ac.il

Abstract

A software system sketcher to facilitate analog circuit design is proposed. The system requires solving two sub-problems. First, the issue of placement of the devices is resolved by using genetic algorithms (GAs), followed by activation of a sub-process that combines routing preferences using a search algorithm, A*. An altruism procedure is implemented over the solution output of the A* search to achieve a better design. The sketcher mimics and implements the combination of experience and intuition required from a human designer.

Peer-review under responsibility of the scientific committee of the CIRP 25th Design Conference Innovative Product Creation
Keywords: analog circuit design; design automation; genetic algorithms

1. Introduction

Analog circuit design is a challenging and complex process that requires fulfilling a variety of goals while conforming to constraints. The engineer's aim is to create a circuit diagram that satisfies specified design goals and complies with the drawing rules globally known as IEC (International Electrotechnical Commission) standards [8]. The design comprises three major stages: topology selection, component sizing and layout generation. Both the selected topology and the sizing must ensure that the resulting circuit satisfies the design objectives. Aaserud and Nielsen [1] noted the following: "In contrast to digital design, most of the analog circuits are still handcrafted by the experts and 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 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 science." Over the last three decades researchers have extensively investigated the design, control and planning of analog circuit design. Modern automation of this procedure entails heuristics [2,5,9], genetic algorithms [3,6,10,12,13,17], multi objective optimization [14] and swarm intelligence [16].

Genetic algorithms (GAs) are considered to be a part of evolutionary computation (EC) methods. They belong to a class of non-gradient methods that have grown in popularity following the seminal publications by Holland [11] and Goldberg [7], which extended and helped popularize the idea. GAs are stochastic search methods that mimic natural biological evolution. They operate on populations of potential solutions by applying the principle of "survival of the fittest" to produce better and better solutions. A GA uses a population of individuals (solutions) instead of a single solution to perform a parallel search in the problem space. At each generation, a new set of approximations is created by a nature-inspired process. The natural processes commonly mimicked by GAs are selection, breeding, mutation, migration and survival of the fittest.

The basic structure of GAs is iterative, as follows[4]:

1. Population initialization: A GA is constructed randomly to generate an initial population. Each individual chromosome is defined as a combination of circuit units—i.e., components—provided by the user. For each unit *Ui* we select a position (*x, y*) and orientation (0, 90, 180, 270 degrees), as depicted in Fig. 1. Each chromosome represents a possible circuit design.

Peer-review under responsibility of the scientific committee of the CIRP 25th Design Conference Innovative Product Creation
doi:10.1016/j.procir.2015.01.033

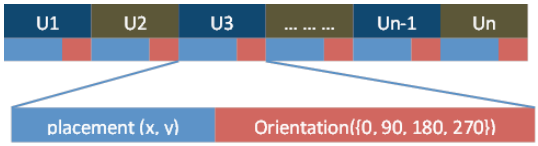


Fig. 1. Unit positioning chromosome structure.

2. Fitness Function: A fitness function is calculated for each individual in the population.
3. Selection: Selection is based on binary tournament selection [4], where two individuals are chosen from the population according to their fitness function.
4. Crossover: Given that population diversity is needed, the GA algorithm provides a crossover procedure that enables reproduction of individuals in the population.
5. Mutation: A schematic procedure is applied to the new chromosome with a prescribed probability.
6. The initial population is replaced by the new population, and Stages 1-5 are repeated recursively until the evaluation function does not develop to a better value.

The aim of this study is to provide the designer with an intuitive tool that mimics the intelligent process of analog system design. The system offers interactive options as well as the choice of a rigid user-defined solution or a totally automatic design solution. The main algorithm of our system is depicted in Fig. 2.

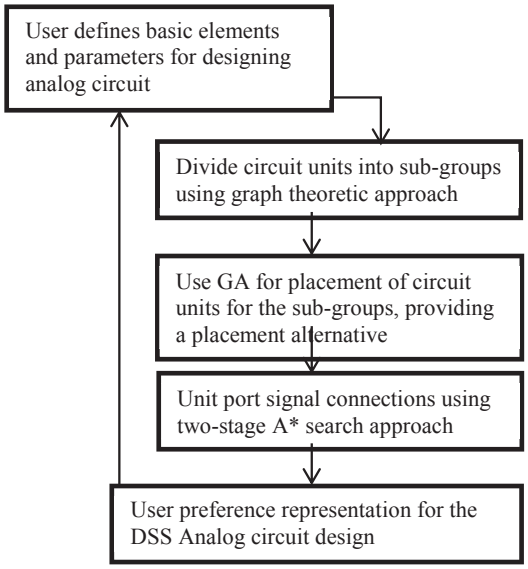


Fig. 2. The flowchart of the DSS of analog circuit design.

2. The Decision Support Sketcher

2.1. Step 1: User defines the basic elements and parameters of the analog circuit design

Each analog circuit design scheme consists of units that will be deployed on the plane to construct a design option for the decision support system (DSS). The problem input is given by the following dataset:

U (designation,$P_u \subseteq P$) – A set of units
P (designation,$S_i \in S$) – A set of pins
S (designation) – A set of signals

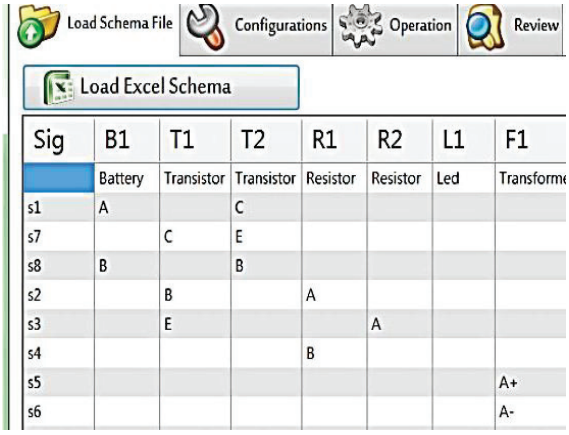
Input for the algorithm will be in the form of the matrix C=[U,S], where each $C_{U,S}$ is assigned a letter and a number representing the pin in the device with which the signal is associated. The process is demonstrated in Fig. 3.

2.2. Step 2: Dividing the circuit units into sub-groups using a graph theory approach

The software implements a partitioning algorithm that tries to obtain groups with the maximum number of connected devices by adding connected devices to the same group. The pseudo code is as follows:

Groups[] (**Devices**[])
Group = new group
  For each **Device** in **Devices**
    For each **Mating_Device** of **Device**
      If **Group** is full
      Group = new group:
      Add **Mating_Device** to **Group**
      Remove **MatingDevice** from **Devices**
    Next **MatingDevice**
  Next **Device**

The algorithm picks a device and puts it in a group. For each device connected to this device, the mating device procedure uses a weighted graph that supplies the edges with weights corresponding to the number of connections to other devices in the group. The groups are formalized according to two criteria: (1) by the number of devices in a group as defined by the user and (2) by the following procedure that chooses the groups according to connectivity weights.



Sig	B1	T1	T2	R1	R2	L1	F1
	Battery	Transistor	Transistor	Resistor	Resistor	Led	Transforme
s1	A		C				
s7		C	E				
s8	B		B				
s2		B		A			
s3		E			A		
s4				B			
s5							A+
s6							A-

Fig. 3. Example signal connection user input (arbitrary)

Fig. 4 shows an example of the influence of user input on group size definition. Both options are in respect to the same input depicted in Figure 3.

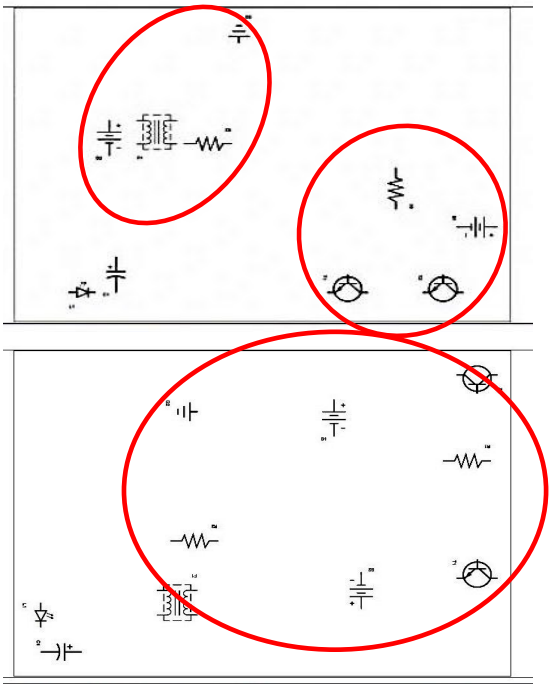


Fig. 4. Temporary results of two different examples: (a) 4 units; (b) 8 units.

Fig. 4 shows (a) that the groups were limited to four units and (b) that the limitation was eight units. The number of units defined by the user influences the group size optimization process used in the GA (step 3) and the mapping design of the circuit.

2.3. Step 3: Using GA for individual placement of each sub-group and combining all sub-groups into previously defined global placement scheme (Fig. 1)

In this step the GA is applied to each group found in the previous step. The iterative procedure is detailed in the following pseudo code.

```
function Solve Circuit(G)
        if(|U| > N) // case circuit is complex //
    G's = Minimum Cut Set(G); // Break it apart. //
                For each (i in Gs)
                        G' = G' UNION
SolveCircuit(Gs_i); // Solve group i //
                G" = Solve Circuit(G'); // Solve the
Groups as a graph. //
                Return G"; // return results
        Else: |U| < N // if Circuit is simple enough
            Decision List DL_1; // create a decision list //
            For Each (u in U) //
                DL: Add Decision (u's Position options);
        // Add Decision of Each units orientation //
                DL: Add Decision (u's Orientation
options); // Add Decision of Each units orientation //
        G' = GA(G, DL); // Use GA for decision on G'.
            Decision List DL_2; // new decision list.
            For Each (u in U)
                DL: Add Decision (u's Pin permutation);
        //Add Decision of Each Pins ordinal position //
        G" = GA (G', DL_2); // Use GA for decision on G'.
            Return G"
End function
```

This step results two sets: a set of unit positions L_U, and a set of pin indexes defining the permutation Lp. where:

L_U *(x,y* ***orientation*** $\in$ *{vertical, horizontal},* $U_i \in U$*)*

L_P *(index* $\in N$, $P_i \in P$*).*

2.4. Calculating the evaluation function

This study was based on basic rules of schematic analog circuit design. After examining the basic features of an accepted design by experts, we grouped all factors into the following attributes: *Device Intersection, Line Intersection*, *Line Lengths*, and *Line Alignment*

Device Intersection: In order to reduce time complexity, we define a two-dimensional grid matrix. The devices are randomly mapped to the grid matrix. While placing each new unit, the system calculates the prices of redundant placement in the grid.

Line Alignment: This is incorporated in the intersection detection; before drawing a line, we check if y1=y2 or x1=x2.

Line Lengths (long/short and standard deviation): This is incorporated into the intersection detection. While drawing the lines we keep a list of line lengths, iterate to find the number of lines exceeding [min, max], and calculate the standard deviation.

The fitness function is, therefore,

$$F= (A+B+C-D) \quad (1)$$

where

A – Number of lines exceeding [min, max] range

B – Standard deviation of line lengths

C – Number of line intersections

D – Number of connected pins that are misaligned.

Each component (A, B, C, and D) is multiplied by a factor for two reasons. The objective is to normalize the units and emphasize or deemphasize each component's importance vis-à-vis the user-defined preferences.

Fig. 5 depicts the configuration input supplied by the user. An attribute's importance influences the alternative solutions generated by the system. The user decides the fitness function criteria by scrolling down and defining the importance of each factor. All parameters are defined by using the scrollbars, as seen in Fig. 5, where the range is between [0,100].

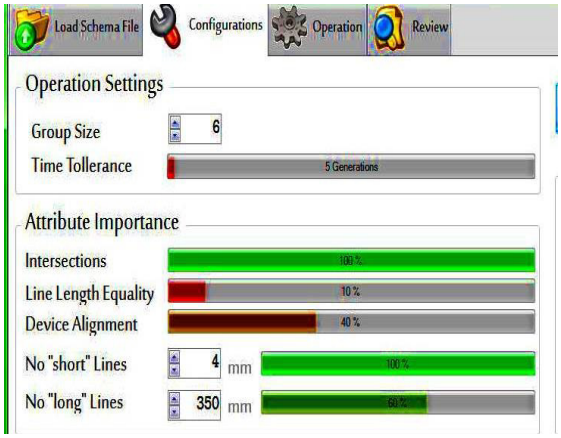


Fig. 5. Configuration input, updated by the user.

The factors are:

- devInterEval – number of device intersections
- lineDiffFactor – value of line length equality
- lineDiffEval – line diff variance
- intersectionFactor – value of the intersections
- lineInterEval – number of intersections between all connections in the chromosome
- short/longLineFactor – value of the short/long lines
- (short)longLinesCnt – number of connections that are shorter/longer than the specified value in the "No Short/Long Lines" numeric box
- alignmentFactor – value of the device alignment
- (short)alignedLinesCnt – number of connections that are aligned on the same X, or the same Y axis. This value has a positive effect on the total eVal.

The algorithm in our proposed method is calculated and assessed in the solution space. The following calculations show that the algorithm converges to a solution in a probable amount of time and provides many alternative designs. (Due to space considerations, only partial time calculations and algorithmic calculations are given.)

Considering each sub-system takes $T(n \le N) \le gN$, where g is the generation per unit limit. The design procedure of the circuit converges to a feasible solution.

$$T(n \le N) = gN \mid g \ge 1$$

$$T(n > N) = \overbrace{\frac{n}{N}T(N)}^{\text{Operating Each Group}} + \overbrace{T\left(\frac{n}{N}\right)}^{\text{Operating on Groups}} + c$$

$$= gn\sum_{j=0}^{i-1}\left(\frac{1}{N^j}\right) + \frac{n}{N^i}T(N) + T\left(\frac{n}{N^i}\right) + ic \tag{2}$$

$$\frac{n}{N^i} \le N \overset{N\ge 1}{\rightleftarrows} n \le N^{i+1} \rightarrow \log_N n \le i+1 \rightarrow \log_N(n) - 1 \le i$$
$$\rightarrow i = \log_N(n) - 1$$

$$gn\sum_{j=0}^{\log_N(n)-1-1}\left(\frac{1}{N^j}\right) + \frac{n}{N^{\log_N(n)-1}}T(N) + T\left(\frac{n}{N^{\log_N(n)-1}}\right) + (\log_N(n) - 1)c == \frac{gn}{N^{\log_N(n)-3}} + gN(N+1) + \log_N(n)\,c = \frac{g}{N^{-3}} + \frac{1}{N^{-4}} + \frac{1}{N^{-5}} + \log_N(n)\,c \le \log_N(n)\,c \Longrightarrow \tag{3}$$

$$O(\log(n))$$

Considering it will take approximately O(n2) operations for a generation to be calculated (fitness, overhead and I/O), we can expect O(n2log(n)) operations per solution.

2.5. Step 4: Unit port signal connections using the two-stage A search approach*

The signal-connecting procedure is based on routing processes and comprises two stages.

(1) *Routing the connection between the pins for optimal signal deployment.* The result of this step is deterministic. The related pins are connected by Manhattan stile lines, while minimal total line length, minimal possible turns and minimum crossover are taken into consideration.

Once the system has found a placement, we are ready to route the connection. Assuming that a good placement was found, we can use a simple yet efficient path-find search algorithm to find the optimal line routing connections in the least expensive way. During routing, we try to avoid intersections, consolidations, wire lengths and curves. Our routing algorithm is built over two layers, where the lower layer is a generic path-finder based on the A* heuristic search algorithm.

A graph *G* is defined to be a set $\{n_i\}$ of vertices called nodes and a set of $\{e_{ij}\}$ of directed edges called arcs. If E_{pq} is an element of the set $\{e_{ij}\}$, then there is an arc from node np to node *nq*, and *nq* is *np*'s successor. A search graph is a collection of nodes and arcs that correspond to states and operators respectively and that have a cost associated with them. Heuristic search theory embodies heuristic information for searching procedures on graphs, and is based on the notion of states and operators. Nilsson [15] defined and detailed the search algorithm A*, a GRAPHSEARCH heuristic procedure for the general structure of exponential-space algorithms. A* can efficiently find a good route to the goal, taking into account limitations such as unreachable positions, turns, intersections and lengths.

The first stage, using A*, works as follows:

- Divide the sheet into a grid (W x H).
- Each device will charge the grid position (*x, y*) C1 for each grid cell it intersects.
- Each line will charge the grid position (*x, y*) C2 for any grid cell through which it passes.

The above algorithm is well defined, but problems arise due to the order of the routes. The example in Fig. 6 describes a situation in which the given pins (2, 2) will be routed before pins (1,1). This behaviour creates two unnecessary intersections, as shown in Fig. 6 (2).

(2) Implementing a second stage is based on altruism.

Overcoming this problem involves implementing a second layer that organizes the order of the connection routing. This second layer is based on the idea of altruism ("selfless concern for the welfare of others").

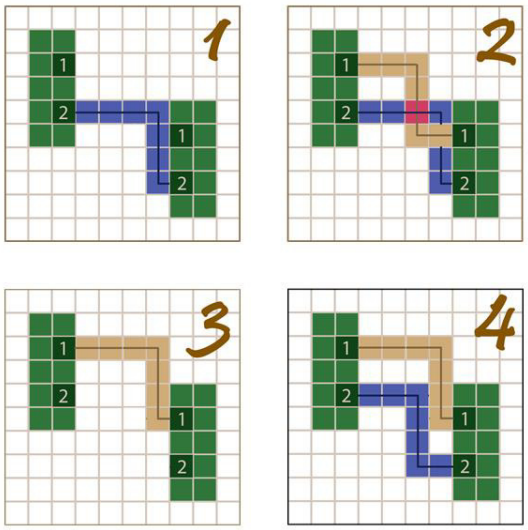


Fig. 6. Altruism algorithm flow

Each line that intersects the currently routed line will "politely" move away and allow the other line to pass through. However, each line will agree to move away from the same connection only once so as to prevent an infinite re-routing of the same connectors. Fig. 6 shows how the algorithm solves a situation in which pins were routed in the wrong order. One can easily see that the second time (2, 2) does not interfere with the first (1, 1) and that both co-exist. Should there be no way to co-exist, (2, 2) "disturbs" (1, 1) and reroutes it to its initial position, but this time when (1, 1) comes across (2, 2), (1,1) will be forced to cross the line.

We use the A* heuristic search algorithm to find the shortest path. The heuristic function *h(x, y)* is defined as the "Manhattan distance" from target x_t, y_t:

$$H(x,y) = \Delta x + \Delta y = |x_t - x| + |y_t - y| \quad (2)$$

The A* cost function *G(x, y)* is the sum of all squares it intersects + the sum of turns. Using this A* level yields a route that has as few turns as possible and is as short as possible with minimal intersections.

3. Examples

The following examples are alternatives offered by the DSS as a single schema formulated by the user. As seen before in Fig. 1, for example, on one side of the screen capture the user has defined the units (register, transistor, etc.), and on the other side the user has defined the signal requirements, Table 1 is an example of a user-input signal connection.

TABLE I. USER-INPUT SIGNAL CONNECTION

Signal \ Unit	Plug1	Plug2(Crossover)	Plug2(Straight)
BI_DA+	1	3	1
BI_DA-	2	6	2
BI_DB+	3	1	3
BI_DC+	4	7	4
BI_DC-	5	8	5
BI_DB-	6	2	6
BI_DD+	7	4	7
BI_DD-	8	5	8

In the next step, the user inputs the parameters. Fig. 7 shows two of many design alternatives when a group size of 4 units was chosen. The user-designer definitions include gene mutation and chromosome rates of 0.08% (a) and 0.03% (b), respectively. The Decision Support Sketcher offers may alternative circuit designs which the designer can use as solutions of designs or partial designs.

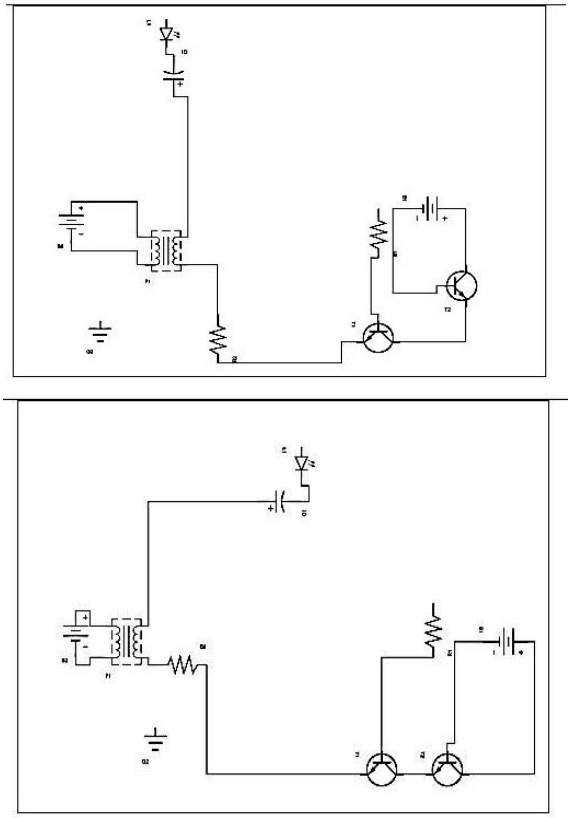


Fig. 7. Two alternative circuit designs for a 4-unit circuit, with gene mutation, and chromosome rates of 0.8% (a) and 0.3% (b)

In a second example, Figs. 8, 9 and 10 show the results of designer definitions. These include a group size of 6 units and gene mutation and chromosome rates of 0.08% (a, b) and 0.03% (c), respectively.

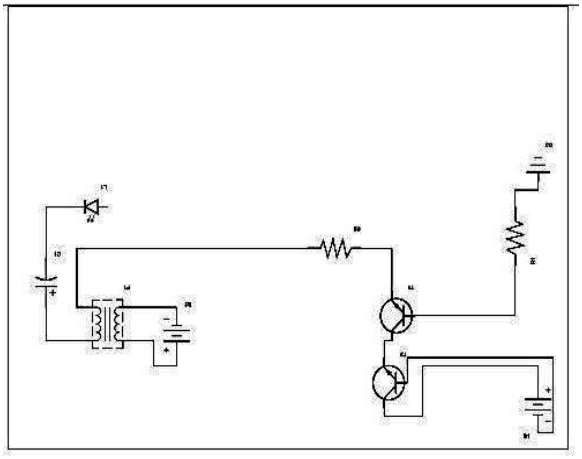


Fig. 8. Circuit design sketch for 6 units in the group size and gene mutation 0.08%.

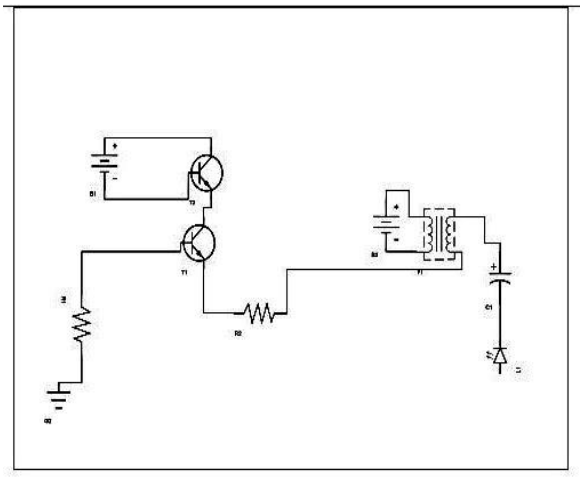


Fig. 9. An alternative circuit design sketch for 6 units in the group size and gene mutation 0.08%.

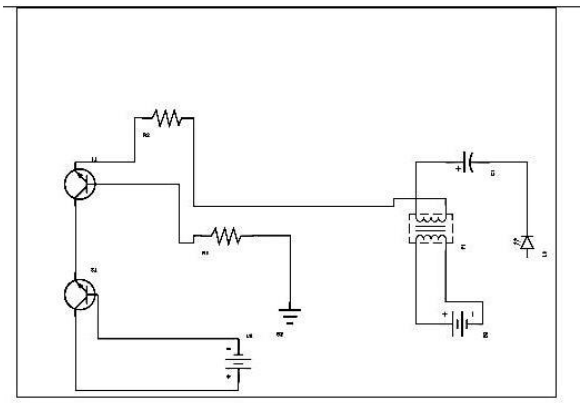


Fig. 10. A third alternative circuit designs for 6 units in the group size and gene mutation rate of 0.3% .

4. Conclusions

The Software system developed is a machine learning tool that offers the designer the ability to automatically mimic some aspect of creativity. Many of the output analog circuit designs obtained by the system represent new possibilities and ideas. Moreover, the presented design system enables to shorten time consuming procedures and results show that robust analog circuits can be achieved at lower design and computational effort. Future work consists embedding the DSS into quality control and circuit simulations procedures for minimizing the power consumption.

References

[1] Aaserud, O.; Nielsen I. R.: Trends in current analog design: A panel debate, Analog Integrated Circuits and Signal Processing, 7(1), 1995, 5-9
[2] Dolatshahia, M.; Hashemipourb, O.; Navib, K.: A new systematic design approach for low-power analog integrated circuits, International Journal of Electronics and Communication, 66(5), 2012, 384-389.
[3] Druiskamp, W.; Leenaerts D.: Analogue circuit synthesis based on genetic algorithms, International Journal of Circuit Theory and Applications, 23, 1995, 285-296.
[4] Eiben, A.E.; Smith, J.E.: Introduction to Evolutionary Computation, Springer 2008. PMCid:3409601
[5] Fakhfakh, M.: A novel alienor-based heuristic for the optimal design of analog circuits, Microelectronics Journal 40, 2009, 141-148.
[6] Feng W.; Yuanxiang L.; Li L.; Kangshun L.: Automated analog circuit design using two-layer genetic programming, Applied Mathematics and Computation, 185, 2007, 1087-1097.
[7] Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Boston, MA, 1989.
[8] Graeb, H.; Zizala, S.; Eckmueller, J.; Antreich, K.: The sizing rules method for analog integrated circuit design, IEEE/ACM International Conference on Computer-Aided Design, Iccad'01, November 4–8, 2001.
[9] Jafari A.; Ehsanbijami, A. N.;, Hasanrekabibana, B.; Saeedsadri, C.: A design automation system for cmos analog integrated circuits using new hybrid shuffled frog leaping algorithm, Microelectronics Journal, 43 ,2012, 908-915.
[10] Jiang Y.; Ju J.; Zhang X.; Yang B.: Automated Analog circuit design using genetic algorithms, IEEE 3rd International Conference on Anti-Counterfeiting, Security, and Identification in Communication, Acid 2009, 223-228..
[11] Holland, J. H.: Adaptation in Natural and Artificial Systems. Univ. of Michigan Press, Ann Arbor, MI, 1975.
[12] Koza J.R.; Bennett F.H.; Andre D.; Keane M.A.: Automated synthesis of analog electrical circuits by means of genetic programming, IEEE Transactions on Evolutionary Computation 1, 1997, 109-128
[13] Kruiskamp W.; Leenaerts, D.: Darwin: Analogue circuit synthesis based on genetic algorithms, International Journal of Circuit Theory and Applications, 23, 1995, 285-296.
[14] Nicosia, G. A.; Rinaudo S.; Sciacca, E.: An evolutionary algorithm-based approach to robust analog circuit design using constrained multi-objective optimization, Knowledge-Based Systems, 21, 2008, 175-183.
[15] Nilsson, N. J.: Principals of Artificial Intelligence, Tioga Pub., Palo Alto, CA 1980
[16] Vural, R.A.; Yildirim, T.: Analog circuit sizing via swarm intelligence, International Journal of Electronics and Communication, 66(11), 2012, 732-740
[17] Wang, F.; Yuanxiang, L. L.; Li, L.; Kangshun, Li.: Automated analog circuit design using two-layer genetic programming, Applied Mathematics and Computation, 185, 2007, 1087–1097