Graph-based evolutionary design of arithmetic circuits

doi: 10.1109/4235.985694

Graph-Based Evolutionary Design of Arithmetic Circuits

Dingjun Chen, Takafumi Aoki, *Member, IEEE*, Naofumi Homma, *Student Member, IEEE*, Toshiki Terasaki, and Tatsuo Higuchi, *Fellow, IEEE*

***Abstract*—In this paper, we present an efficient graph-based evolutionary optimization technique called evolutionary graph generation (EGG) and the proposed approach is applied to the design of combinational and sequential arithmetic circuits based on parallel counter-tree architecture. The fundamental idea of EGG is to employ general circuit graphs as individuals and manipulate the circuit graphs directly using new evolutionary graph operations without encoding the graphs into other indirect representations, such as bit strings used in genetic algorithm (GA) proposed by Holland and trees used in genetic programming (GP) proposed by Koza *et al.* In this paper, the EGG system is applied to the design of constant-coefficient multipliers and the design of bit-serial data-parallel adders. The results demonstrate the potential capability of EGG to solve the practical design problems for arithmetic circuits with limited knowledge of computer arithmetic algorithms. For example, in the design of fast constant-coefficient multipliers consisting of shifters and parallel counters, the results obtained from the EGG are superior to or as good as the known conventional designs using arithmetic algorithms. This means that the proposed EGG system can help to simplify and speed up the process of designing arithmetic circuits and can produce better solutions to the given problem.**

***Index Terms*—Arithmetic circuits, canonic signed-digit (CSD) representation, digital signal processing (DSP), electronic design automation (EDA), evolutionary computation, evolutionary graph generation (EGG), multipliers.**

I. Introduction

DESIGNING electronic circuits is generally a complicated and time-consuming task requiring knowledge of large collections of domain-specific rules. Most electronic systems of any complexity were traditionally created by a designer who had been trained in a particular way to understand the operations of individual electronic components and who could, therefore, use these components to construct larger systems. Clearly, the designers will also work through a number of iterations of testing and debugging the circuits before completing the design. However, the final product will only be as good as the designer's own knowledge and experience allow. In order to simplify and speed up the design process, evolutionary computation methods have been studied and successfully applied to complex design optimization problem. Recent researches [1]–[7] have begun to show that it is possible to design electronic circuits using evolutionary optimization techniques. The reference [1], for example, presented a genetic algorithm (GA) that is capable of evolving 100% functional arithmetic circuits. This new field of research has come to be known as *evolvable hardware* [2]–[4]. Its prominent advantage is the fact that the evolutionary design will inevitably allow the automatic exploration of a much richer set of possibilities in the design space that are beyond the scope of conventional methods.

Currently, arithmetic circuits are becoming more important in today's computing and digital signal processing (DSP) systems. State-of-the-art logic synthesis tools provide only limited capability to create the structural details of arithmetic circuits. Likewise, recent high-level synthesis techniques tend to employ module libraries containing basic arithmetic functional units, which are usually designed in advance as essential resources. In order to address this problem and develop a method for synthesizing arithmetic circuits automatically with limited knowledge of computer arithmetic algorithms, we propose a new approach, called evolutionary graph generation (EGG), to automatically designing circuit (see [8]–[11] for earlier discussions on this topic). The novel idea of EGG is to consider general circuit graphs as individuals and manipulate the circuit graphs directly using new evolutionary graph operations without encoding them into other indirect representations, such as bit strings used in GA proposed by Holland and trees used in genetic programming (GP) presented by Koza *et al.* [5], [6], [12]. This makes it possible to efficiently generate the desired arithmetic structure.

The main contributions of this paper are:

1) graph-based chromosome representation[1] that is capable of handling the structures of complex arithmetic circuits;
2) a symbolic verification technique for checking the functionality and performance of arithmetic circuits quickly by solving a set of mathematical equations, especially, the methodology for evaluating the fitness of fast constant-coefficient combinational multipliers and multiple-operand bit-serial adders;
3) efficient generation-dependent dynamic mechanism of adjusting operator probabilities (the crossover rate and the mutation rate);
4) comparisons of the solution quality of the EGG system in the design of fast constant-coefficient multipliers with the known conventional designs using canonic signed-digit (CSD) number representation.

Manuscript received September 11, 2000; revised January 3, 2001.

The authors are with the Higuchi Laboratory, Department of System Information Sciences, Graduate School of Information Sciences, Tohoku University, Aoba-yama 05, Sendai 980-8579, Japan.

Publisher Item Identifier S 1089-778X(02)02207-5.

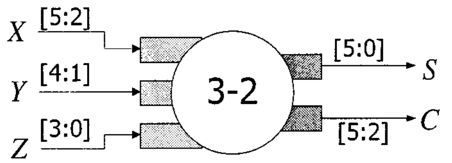
[1]We propose a new graph-based individual representation, called circuit graph, for handling the circuit structures. As for related references, such as [5], [13], we can find some proposals of graph-based evolutionary algorithms in other application domains.

II. Genetic Representation

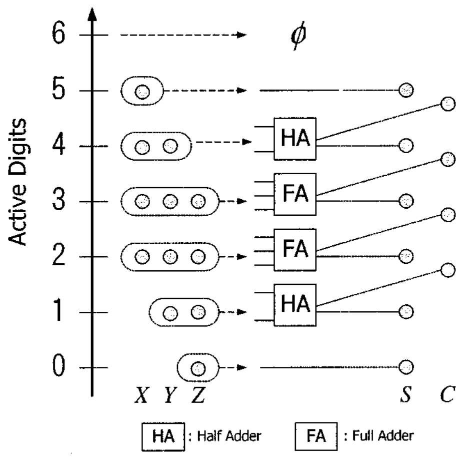
How to encode a potential solution of the problem into a suitable chromosome is a key issue for evolutionary computation. Currently, there are numerous approaches used as the chromosome representation of hardware structures [1]–[7]. In [1], for example, an approach to the evolutionary design of arithmetic circuits is described. The reported design method, however, is based on direct evolution with gate-level primitive components such as logic gates and flip-flops. It seems to be quite difficult for us to apply this method to solve the practical design problems of arithmetic circuits due to its limited capability of modeling arithmetic algorithms. Furthermore, choosing an appropriate representation of candidate solution to the problem at hand is the foundation for applying evolutionary computation to solve real-world problems, which conditions all the subsequent steps of evolutionary computation. GP uses trees to represent individuals. Trees may be viewed as graphs without cycles. This is particularly useful for representing computer programs. Al Globus *et al.*in [13] proposed genetic graphs to map standard GA techniques to molecular design and use cyclic graphs to represent molecules. Circuit design is another field for which genetic graphs should, in principle, be well suited, but the current experimental results show that genetic graphs can evolve only very simple digital logic circuits consisting of logic gates.

In practice, we use a kind of special circuit graphs to represent circuit structure. A circuit graph G is defined by $G = (N(G), D(G))$, where $N(G)$ is the set of nodes and $D(G)$ is the set of directed edges. There are two kinds of different nodes: functional nodes and input–output nodes (I/O). Each node has its own name, the function type and I/O terminals. We assume that each directed edge must connect one output terminal (of a node) and one input terminal (of another node) and each terminal can have one edge connection at most. A circuit graph is said to be *correct* if all the terminals have an edge connection. In order to reduce the search space, the EGG system only generates the correct circuit graphs.

We have implemented a special EGG system, called Arithmetic-EGG, for arithmetic circuits synthesis, which employs a higher level of abstraction for arithmetic algorithms. The Arithmetic-EGG interprets a circuit graph as a data-flow graph representing an arithmetic computation process based on a specific number representation system. A directed edge in the data-flow graph represents the dependence of operands. Two attributes are simultaneously assigned to each edge: 1) the type of number system for operand encoding and 2) the activated operand digits. We suppose the use of a positional number system for operand representation, which is identified by the triplet $\langle D, \Lambda, \Omega\rangle$, where D is the *digit set*, $\Lambda = (\lambda_{k-1}, \ldots, \lambda_i, \ldots, \lambda_0)$ is the *sign vector*, and $\Omega = (\omega_{k-1}, \ldots, \omega_i, \ldots, \omega_0)$ is the *absolute weight vector*, respectively. A node in arithmetic-EGG's data-flow graph represents a specific arithmetic operation. Thus, the node itself has no circuit details at first. It can be transformed into a set of bit-level circuit elements only when the attributes of all input operands are determined. Hence, the actual interpretation of a node depends on the overall structure of the data-flow graph.



(a)



(b)

Fig. 1. 3-2 counter node. (a) Symbol. (b) Actual circuit interpretation of the node in (a).

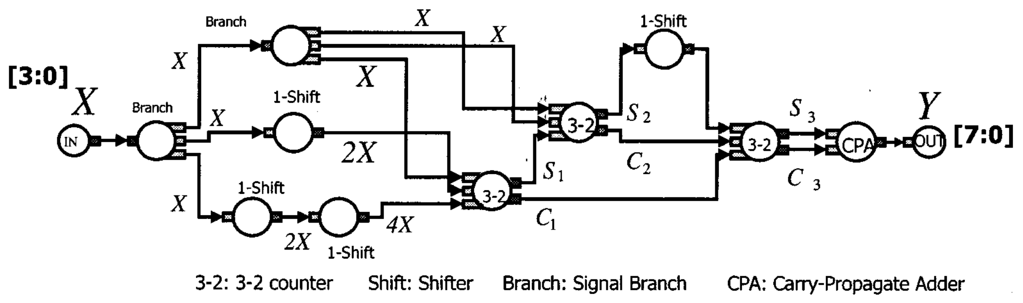
TABLE I
BIT OPERATIONS FOR THE 3-2 COUNTER NODE

Number of inputs	Bit operation
0	ϕ
1	Wire
2	HA (Half Adder)
3	FA (Full Adder)

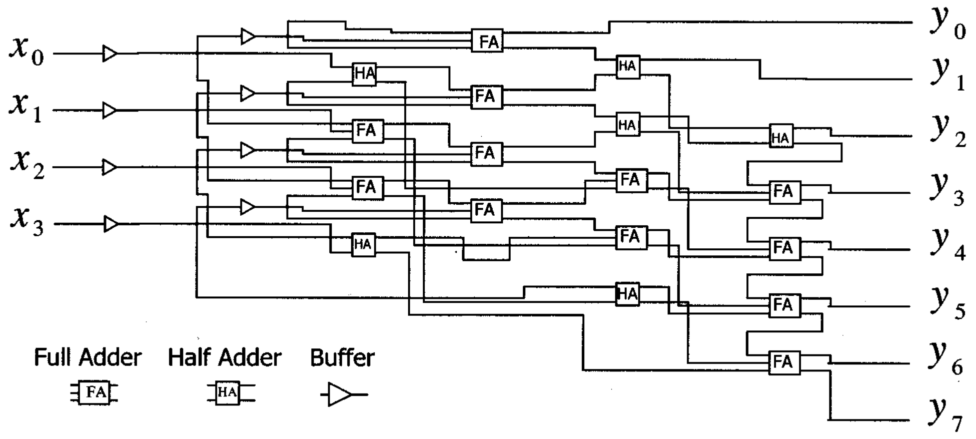
Fig. 1(a) shows an example of a 3-2 counter node, which represents the 3-input 2-output carry-free addition of binary numbers. Note that the symbol [MSD:LSD] represents the range of active digits, where MSD is the most significant active digit and LSD is the least significant active digit. Each node has a rule for generating the corresponding bit-level circuit interpretation. Table I shows the rule for the 3-2 counter node as an example. Using this interpretation rule, the node of Fig. 1(a) can be transformed into a set of circuit elements as illustrated in Fig. 1(b). Fig. 2(a) gives an example of a data-flow graph. By applying 4-bit unsigned binary data to the input node, the graph is translated into the arithmetic circuit, which is illustrated in Fig. 2(b).

III. Methodology for Fitness Evaluation

Each circuit graph generated by the EGG is evaluated by a combination of two different fitness functions: *functionality* and *performance*. The execution time of the EGG system is mainly dominated by the process of evaluating the functionality and performance of the evolved circuit graphs in each generation. As the circuit structures are becoming increasingly complex, it



(a)



(b)

Fig. 2. Circuit graph in the Arithmetic-EGG. (a) An example. (b) Actual circuit interpretation of the graph in (a).

TABLE II
FUNCTIONAL NODES

Name	Symbol Description	Function Description	Output Sign
Signed-Weight 3-2 counter	3-2	3-input 2-output carry-free addition	Variable
		3-input 2-output carry-free addition with 2-way branches	
		3-input 2-output carry-free addition with 3-way branches	
Final stage adders	FSA	Carry-propagate addition with a bias canceling stages	Invariable
1-bit shifter	1-S	1-bit arithmetic shift	Invariable
2-bit shifter	2-S	2-bit arithmetic shift	Invariable
4-bit shifter	4-S	4-bit arithmetic shift	Invariable
Operand input	IN	Input signal	Variable
Operand output	OUT	Output signal	

is not efficient to translate every circuit graph into Verilog hardware description language (HDL) and then simulate it. Thus, we propose a symbolic verification technique for checking the functionality and performance of arithmetic circuits quickly by solving a set of mathematical equations. The computation time of this verification process is $O(m^3)$, where m is the number of nodes. Some experimental results described in our previously published paper [9], have already demonstrated that this approach remarkably reduces the time cost of functional verification and performance evaluation of the evolved circuits, rather than transforming them into Verilog HDL codes.

It is also worth noting that the proposed EGG system can be easily applied to different design specifications by slightly changing the fitness functions. It is well known that there exist some inherent differences in the design of combinational and sequential arithmetic circuits and we need to set up different fitness evaluation functions depending on target circuit specifications (combinational or sequential) accordingly. In this paper, we choose the optimal design of fast constant-coefficient combinational multipliers and multioperand bit-serial adders as typical design problems to observe the feasibility and effectiveness of arithmetic circuit design based on the proposed EGG.

A. Functional Measure

1) Functional Measure of Constant-Coefficient Combinational Multipliers: Consider a constant-coefficient combinational multiplier that is comprised of the primitive components, depicted in Table II. Given a specific constant-coefficient multiplier, the mathematical representation of its function can be derived by performing symbolic model checking on the given structure. For example, the function of the data-flow graph of Fig. 2 can be derived via solving the set of equations as follows:

$$C_1 + S_1 = 4X + 2X + X \tag{1}$$
$$C_2 + S_2 = S_1 + X + X \tag{2}$$
$$C_3 + S_3 = C_1 + C_2 + 2S_2 \tag{3}$$
$$Y = C_3 + S_3 \tag{4}$$

where X is the input vector and Y is the output vector. Using Gauss elimination, we can easily obtain the following:

$$Y = 9X + S_2. \tag{5}$$

We can further simplify the derived equation by considering the specific I/O relationship of every counter node. For example, it is easily proved that $C_2 = 2X$ and $S_2 = S_1$ and, hence, we have

$$Y = 9X + S_1. \tag{6}$$

As a result, the evolving circuit graph can be generally represented as $Y = \hat{R}X + f(X)$, where $\hat{R}$ is an integer constant and $f(X)$ is a nonlinear function of X, which is represented by intermediate variables. To evaluate the function of the desired multiplier, the estimated coefficient $\hat{R}$ is compared with the target coefficient R, where the term $f(X)$ has some adverse effect on the functionality measure. When $f(X) = 0$, the circuit graph can be regarded as an effective constant- coefficient multiplier.

We describe the functionality measure F in detail as follows. Let R be the target coefficient and represented with the following form:

$$R = r_0 2^0 + r_1 2^1 + r_2 2^2 + \cdots = \sum_{j=0}^{||R||-1} r_j 2^j \tag{7}$$

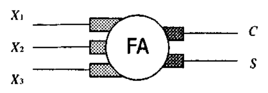
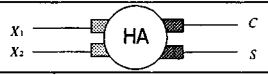
where $||R||$ denotes the length of the representation of the coefficient R and $r_j \in \{-1, 0, 1\}$. As described above, the system examines the functionality of a circuit graph by symbolic verification and obtains the estimated coefficient $\hat{R}$, which may be written as

$$\hat{R} = \hat{r}_0 2^0 + \hat{r}_1 2^1 + \hat{r}_2 2^2 + \cdots = \sum_{j=0}^{||\hat{R}||-1} \hat{r}_j 2^j. \tag{8}$$

The similarity between R and $\hat{R}$ is evaluated by digit-coincidences for all the digit positions of the given two strings. The correlation $M_{\hat{R},R}(s)$ of the two coefficient strings at the shift amount s $(0 \le s \le |\,||\hat{R}|| - ||R||\,|)$ is defined by

$$M_{\hat{R},R}(s) = \begin{cases} \dfrac{1}{\left\|\hat{R}\right\|} \displaystyle\sum_{j=0}^{||\hat{R}||-1} \delta(\hat{r}_j - r_{j-s}), & \text{if } \left\|\hat{R}\right\| \ge ||R|| \\ \dfrac{1}{||R||} \displaystyle\sum_{j=0}^{||R||-1} \delta(\hat{r}_{j-s} - r_j), & \text{if } \left\|\hat{R}\right\| < ||R|| \end{cases} \tag{9}$$

TABLE III
FUNCTIONAL NODES

Name	Symbol / Mathematical representation	Delay
Full adder	 	2τ
	$2C + S = X_1 + X_2 + X_3$	
Half adder	 	τ
	$2C + S = X_1 + X_2$	
1-bit register	 	
	$Y = 2X$	

where $\delta(x)$ is defined as

$$\delta(x) = \begin{cases} 1, & x = 0 \\ 0, & x \ne 0. \end{cases} \tag{10}$$

In the above calculation, we assume the values of the undefined digit positions to be zero for both coefficient strings. Using this correlation function, the functionality measure F is defined as

$$F = \max_{0 \le s \le d} \left[100 M_{\hat{R},R}(s) - C_1 s \right] \tag{11}$$

where $d = |\,||\hat{R}|| - ||R||\,|$ and $C_1 = 2$ in this experiment.

2) Functional Measure of Multioperand Bit-Serial Adders: Consider a multioperand bit-serial adder consisting of the primitive components described in Table III. Fig. 3 is an example of a 2-input bit-serial arithmetic circuit. We can describe the circuit structure as a set of simultaneous equations as follows:

$$W_1 = X_1 \tag{12}$$
$$W_2 = X_1 \tag{13}$$
$$W_3 = 2W_1 \tag{14}$$
$$2W_5 + W_6 = W_2 + W_3 + W_4 \tag{15}$$
$$2W_4 + W_7 = X_2 + W_9 \tag{16}$$
$$W_9 = 2W_8 \tag{17}$$
$$2Y + W_8 = W_5 + W_6 + W_7 \tag{18}$$

where the variables appeared in the above equations are integer variables represented by the corresponding bit-serial signals shown in Fig. 3. Likewise, the I/O relationship of the circuit can be derived via solving these equations. Using Gauss elimination, we have

$$2Y = 3X_1 + X_2 - W_4 - W_5 + W_8. \tag{19}$$

As shown in this example, the function of an n-input 1-output bit-serial arithmetic circuit consisting of the nodes shown in Table III can be generally represented as

$$\hat{K}_0 Y = \sum_{i=1}^{n} \hat{K}_i X_i + f(X_1, \ldots, X_n) \tag{20}$$

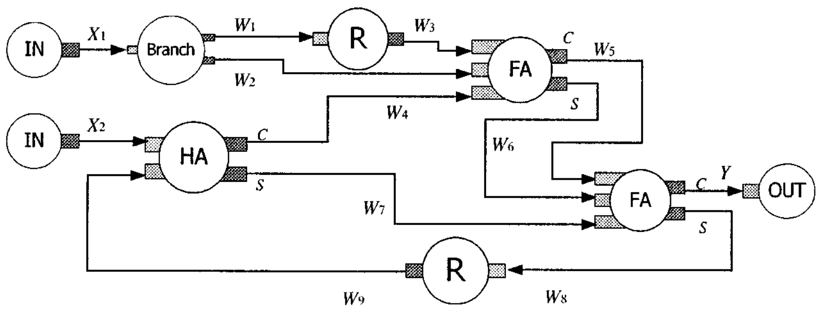


Fig. 3. Example of a 2-input bit-serial sequential arithmetic circuit.

where $X_i (i = 1, \ldots, n)$ represent the bit-serial inputs, Y represents the bit-serial output, $\hat{K}_i (i = 0, \ldots, n)$ are the integer coefficients, and $f(X_1, \ldots, X_n)$ is a nonlinear function of input operands. The term f consists of intermediate variables $W_j (j = 1, 2, \ldots)$ that cannot be eliminated through Gauss elimination.

We assume that the target function is given by

$$K_0 Y = \sum_{i=1}^{n} K_i X_i \tag{21}$$

where $K_i (i = 0, \ldots, n)$ are the nonnegative integer coefficients. Note here that we must set the target coefficients as $K_0 = K_1 = \cdots = K_n = 1$ when we synthesize an n-operand bit-serial adder. The functionality measure F is calculated by evaluating the similarity between the coefficients $\hat{K}_i$ and the target coefficients K_i for $i = 0, 1, \ldots, n$. To do this, we first expand the coefficients into binary strings as

$$\begin{aligned} \hat{K}_i =& \hat{k}_{i,0} 2^0 + \hat{k}_{i,1} 2^1 + \hat{k}_{i,2} 2^2 \\ &+ \cdots + \hat{k}_{i,\, \|\hat{K}_i\|-1} 2^{\|\hat{K}_i\|-1} \end{aligned} \tag{22}$$

$$\begin{aligned} K_i =& k_{i,0} 2^0 + k_{i,1} 2^1 + k_{i,2} 2^2 \\ &+ \cdots + k_{i,\, \|K_i\|-1} 2^{\|K_i\|-1}. \end{aligned} \tag{23}$$

The similarity of these coefficients is evaluated by computing their correlation. The correlation $M_{\hat{K}_i, K_i}(s)$ of the two coefficient strings with the shift amount s is defined by

$$M_{\hat{K}_i, K_i}(s) = \begin{cases} \dfrac{1}{\left\|\hat{K}_i\right\|} \displaystyle\sum_{l=0}^{\|\hat{K}_i\|-1} \delta\left(\hat{k}_{i,l} - k_{i,l-s}\right), & \text{if } \left\|\hat{K}_i\right\| \geq \|K_i\| \\ \dfrac{1}{\|K_i\|} \displaystyle\sum_{l=0}^{\|K_i\|-1} \delta\left(\hat{k}_{i,l-s} - k_{i,l}\right), & \text{if } \left\|\hat{K}_i\right\| < \|K_i\| \end{cases} \tag{24}$$

where $\delta(x)$ is defined by (10).

In the above calculation, we suppose the values of the undefined digit positions to be zero for both coefficient strings. Using this correlation function, the similarity F' between (22) and (23) is defined as

$$F' = \frac{1}{n+1} \sum_{i=0}^{n} \left[\max_{0 \leq s \leq d} \left\{ 100 M_{\hat{K}_i, K_i}(s) - C_1 s \right\} \right] \tag{25}$$

where $d = |\, \|\hat{K}_i\| - \|K_i\| \,|$ and $C_1 = 10$ in our experiments. The term $C_1 s$ represents the adverse effect due to the shift amount s. Using the similarity, we define the functionality measure F as

$$F = F' - C_2 q \tag{26}$$

where q is the number of delay-free loops in the evolved circuit and $C_2 = 5$ in our experiments.

B. Performance Measure

On the other hand, the performance measure P for combinational multiplier and multiple-operand bit-serial adders are both defined as

$$P = \frac{1}{DA} \tag{27}$$

where the delay D is the number of stages of the counters and the area A is the total number of intermodule interconnections when translating a circuit graph into the actual bit-level circuit. The delay time D is measured by using a 2-input XOR gate as an element of the unit delay τ.

During the process of evolution, at first, each individual in population is evolved toward obtaining 100% functionality. Only after the individual has got the 100% functionality, the performance evaluation of the individual begins to be considered. Otherwise, the value of performance measure P is always set to zero. In our experiments, when the function of an individual first reaches the desirable specification, the fitness value will be set to 103. Hereafter, as long as the product $(A * D)$ decreases, the fitness value of the individual will correspondingly increase by degrees. Once the individual has got 100% functionality, the size of the product $(A * D)$ becomes the most crucial factor that assesses the solution quality. The best individual in population evolves toward better performance. In terms of the performance measure P described by (27), the smaller the product $(A * D)$ of the area and delay is, the better

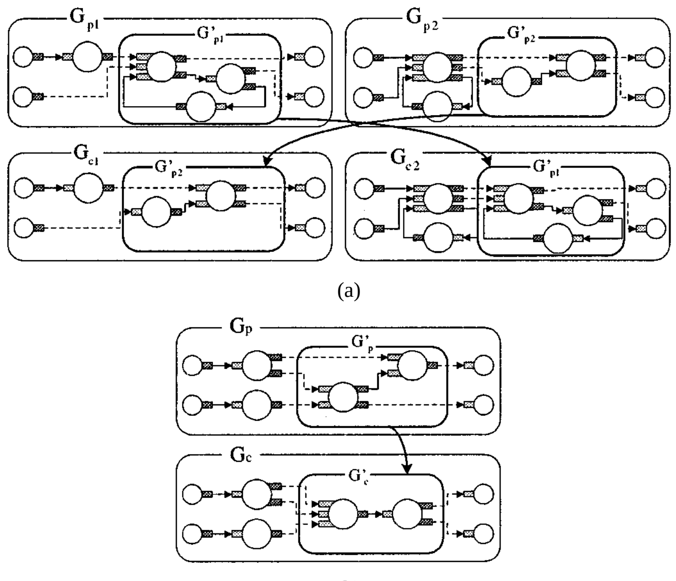


Fig. 4. Examples of genetic operations. (a) Crossover operation. (b) Mutation operation.

the performance of the individual is. We cannot guarantee that, for each run, the evolutionary algorithm will definitely be successful to find the ideal solution to the problem. If the best individual in population does not obtain 100% functionality within the gross number of generations, basically this experiment can be thought of a failing trial. In this case, because the different setting of the seed values can lead to generating the different initial population, we can resume the trial to pursue better solutions after reinitializing the population by means of the random tuning of the seed values.

IV. Genetic Operations

In the EGG, there are two genetic operations crossover and mutation. Both operations transform the structure of a circuit graph preserving its correctness property, that is, if the parents are correct graphs, the transformed graphs also have correct structures. Individuals are generated in the initial population as follows. First, we select functional nodes randomly from the table of primitive functional nodes, then count the number of input terminals and the number of output terminals in all nodes selected, next add specific functional nodes so as to balance the number of input and output terminals, and finally connect the terminals randomly to generate a correct circuit graph. Once the initial population is generated, the subgraphs inside each individual are also randomly generated. Each circuit graph can be allowed to hold 100 subgraphs in our experiments. Generally, the number of the subgraphs is enough for genetic operators to perform genetic operations. When an individual is replicated, the inherent subgraphs are also replicated.

Crossover is the main genetic operator. It recombines two parent graphs into two new graphs. When a pair of parent graphs G_{p1} and G_{p2} are selected from the population, the crossover operation determines a pair of subgraphs G'_{p1} and G'_{p2} to be exchanged between the parent graphs and generates offspring by replacing the subgraph of one parent with that of the other parent as illustrated in Fig. 4(a). In this process, the system randomly selects a subgraph from the mother circuit graph and selects a compatible subgraph from the father circuit graph, where "compatible" means that the cut sets for these subgraphs contain the same number of edges for both negative and positive directions.

Mutation operation, on the other hand, performs random modifications on an individual. Our mutation operator works briefly as follows: it selects a subgraph randomly and replaces it with a randomly generated subgraph that is compatible with the original subgraph. Fig. 4(b) illustrates an example of the mutation operation.

In the following, we provide more details on these variation operations. In the EGG, the data structure for each circuit graph holds the data of a specific number of subgraphs (here it equals 100) randomly selected from the circuit graph. This subgraph data includes the information of the cut set for every subgraph. In the crossover operation, the system first selects a subgraph randomly from the mother circuit graph and then selects a compatible subgraph from the father circuit graph by scanning father's cut set data. Because each circuit graph can be allowed to hold 100 subgraphs, generally, it is unlikely that there will be no appropriate subgraph from the father circuit graph. When there are multiple choices of father's subgraphs, the system takes a random selection. In the case that there is no proper subgraph in the father circuit graph, then we randomly reselect a subgraph from the mother circuit graph and restart this operation. If a subgraph with the appropriate cut set does not appear in the mother, indeed, this crossover operation can be regarded as a failure operation. The mother needs to be reselected. In the mutation operation, on the other hand, the system selects a subgraph randomly from a circuit graph and checks its cut set. Then, to satisfy the cut set compatibility, a compatible subgraph is generated by setting the number of nodes and their attributes at random.

Note that subgraphs to be selected for variation operations are not necessarily connected subgraphs. The EGG system preserves only correctness property during the process of evolution and allows disconnected/isolated subgraph structures. Even loop structures[2] are allowed in basic variation operations. When isolated subgraphs and loops are not necessary, these structures are eliminated at the evaluation stage in the EGG system flow, which depends on applications in general. In other words, the circuit graph with such subgraphs and loops generally has got a lower fitness at the evaluation stage, so it has little chance to be selected and included in the next generation population. Other redundant structures are deleted through evolution process.

V. Selection Mechanism

The selection mechanism has an important responsibility for controlling the diversity of the population. It may maintain or eliminate diversity, depending on its current selection pressure, which represents the degree to which the selection mechanism

[2]During the evolution process, crossover operator will perhaps generate the new loops in the subgraphis. In the case of the design of multipliers, after crossover operation if there appears this case, the corresponding fitness value of the graph will be evaluated and set to zero. The graph with so low fitness will be extremely possible abandoned during the later evolution process. In the case of the design of sequential circuit, this case can be allowed to happen.

favors the better individuals. At one extreme, the search will terminate prematurely, while at the other extreme, progress will be slower than necessary. Typically, low selection pressure is indicated at the start of evolutionary search in favor of a wide exploration of the search space, while high selection pressure is recommended at the end in order to exploit the most promising regions in the search space. During the past few years, many selection methods have been proposed, examined, and compared [5], [14].

Like the other evolutionary algorithms, the EGG system employs two different selections for selecting parents and survivors during the process of evolution. First, some of the individuals are selected as parents by a *rank-based selection method,* which utilizes the indexes of individuals when ordered according to fitness values to calculate the corresponding selection probabilities. The assignment function is linear.

Second, in order to reconstruct the next new population after performing genetic operations, some of individuals or offspring are selected by *mixed sampling selection*. We know, from the characteristic of our genetic operation, that a high-quality parent does not necessarily produce a high-quality descendant, but we still want to use an indeterminate reduction strategy that guarantees the diversity of individuals. In most cases, the number of the offspring generated by genetic operations is not enough to form the next-generation population. Hence, we randomly replicate a number of individuals from the parent population to keep the size of population unchanged. On the other hand, in order to let the current fittest individual survive while it is superior to other individuals at this moment, we had better provide a privilege for it so that it must be added to the next generation population. This is necessary since it is possible that the best chromosome may disappear due to genetic operation. Another advantage is that the fittest individual can be regarded as the final solution found in the entire process of evolution when termination condition is satisfied. It is easy to see that this strategy simultaneously contains both random and a little deterministic features, and tries to reach the balance between exploiting what already works best and exploring possibilities that might eventually evolve into something even better. However, it may have a dangerous effect. The continuous presence of good individuals will be likely to produce an early convergence toward such individuals. Fortunately, bigger size of population probably contributes to the disappearance of this problem.

For better understanding the EGG, the detailed description of algorithm is given as follows.

```
    (Crossover rate: $p_c$, mutation rate: $p_m$, popula-
      tion Size: $N$, generation: $k$.)
Begin
  $k \leftarrow 0$;
  Initialize population $P(k)$;
  Evaluate $P(k)$;
  While (termination condition is not satisfied) do
    Generate offspring $C_c(k)$ from $P(k)$ by crossover
  operation;
    Evaluate $C_c(k)$;
    Select $(1-p_c)N$ individuals from $P(k)$ randomly and
  combine them with $C_c(k)$ to produce the temporary
  population $Temp(k)$;
    Generate offspring $C_m(k)$ from $Temp(k)$ by mutation
  operation;
    Evaluate $C_m(k)$;
    Select $(1-p_m)N$ individuals from $Temp(k)$ randomly
  and combine them with $C_m(k)$ to produce the next
  generation population $P(k+1)$;
    $k \leftarrow k+1$;
  End
End
```

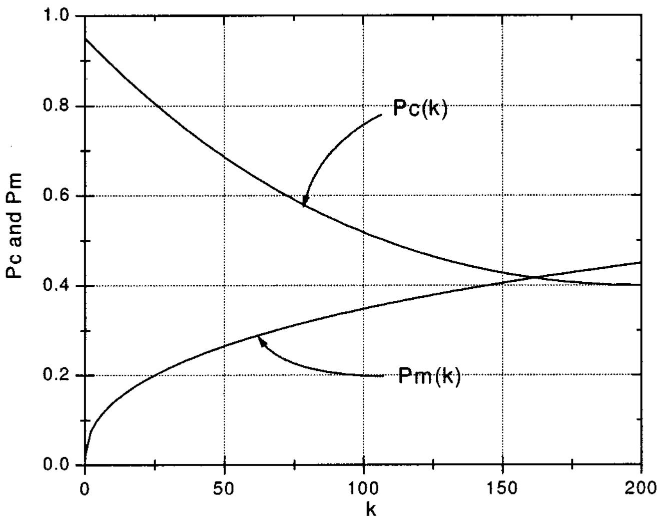


Fig. 5. Variational curves of p_c and p_m.

From above described algorithms, we can see the crossover and mutation operations are performed separately. The new population comes from the parents and offspring generated by crossover and mutation operations. Thus, the probability p_r of random replication from parent is $p_r = 1 - p_c$ and $p_r = 1 - p_m$, respectively. Because operator probabilities p_c and p_m (described in Section VI) both are dynamic, the probability p_r of random replication from parent population is also dynamic.

VI. A Dynamic Mechanism of Operator Probabilities

Finding optimal static parameters for a particular problem for an evolutionary algorithm is a poorly structured ill-defined complex problem. It can be quite difficult and the optimal values may depend on many other factors (such as the applied recombination operator, the selection mechanism, etc.). Some researchers [15]–[18], however, generally think that the use of static parameters itself can lead to inferior algorithm performance. The straightforward way to treat this problem is by using parameters that may change over time, i.e., by replacing a parameter p by a function $p(k)$, where k is the generation counter. Our solution is a new effective generation-dependent dynamic mechanism of operator probabilities. The feasibility and effectiveness of this model are demonstrated later by our experiments.

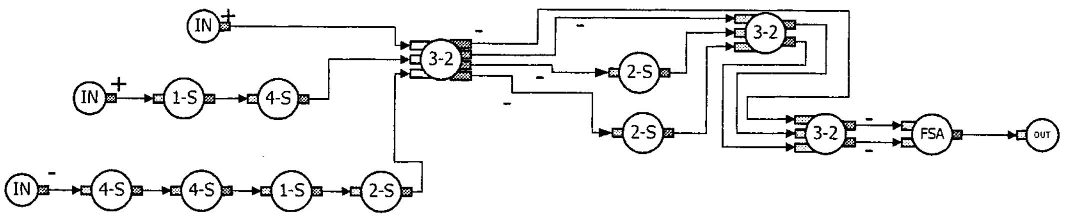


Fig. 6. Circuit graph generated by the arithmetic-EGG with the parameters of Table VI.

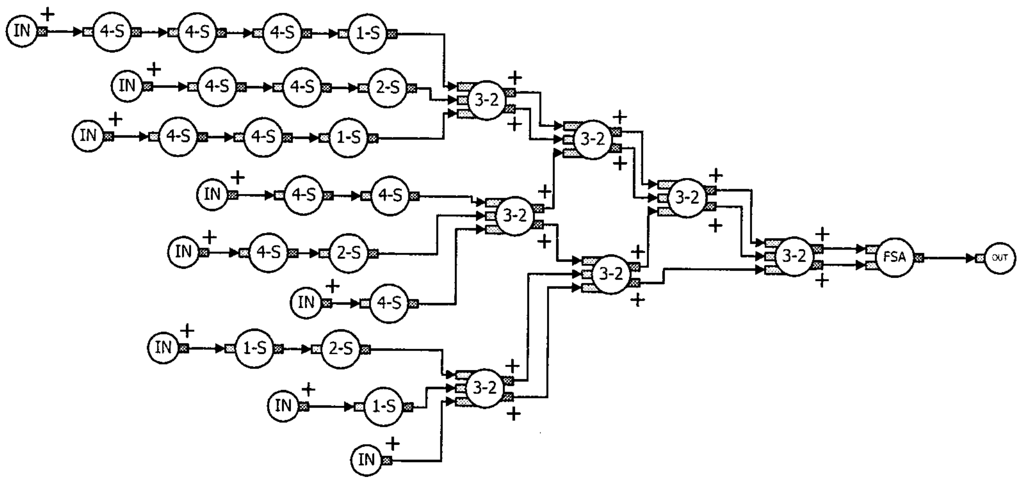


Fig. 7. Multiplier using Wallace-tree architecture consisting of the least number of stages of ordinary unsigned binary 3-2 counters.

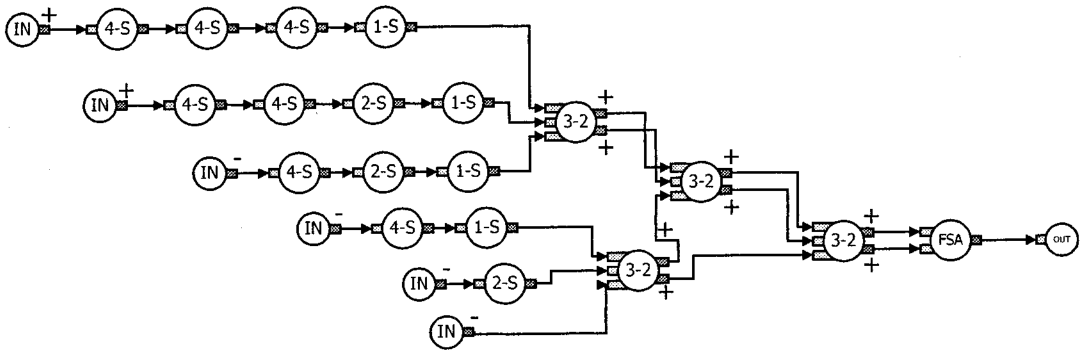


Fig. 8. CSD multiplier using Wallace-tree architecture consisting of the least number of stages of SW 3-2 counters.

A. Model of Crossover Probability p_c

The expression of crossover probability p_c is expressed as

$$p_c(k) = (\alpha_1 + (1-\beta_1))^{g_1(k)} - (1-\beta_1) \tag{28}$$

where

α_1	minimum of p_c;
β_1	maximum of p_c;
k	is the generation ($k = 0, 1, 2, \ldots, \text{MaxGen}$, MaxGen is the maximum generation);
$g_1(k)$	function of generation k, which may be linear or nonlinear.

For example

$$g_1(k) = k/\text{MaxGen} \text{ (linear)}$$

or

$$g_1(k) = \sin\left(\frac{k\pi}{2\,\text{MaxGen}}\right) \text{ (nonlinear).}$$

From (28), we obtain

$$p_c(0) = \beta_1,\ p_c(\text{MaxGen}) = \alpha_1$$

where the range of α_1 and β_1 is [0, 1] and $\alpha_1 \le \beta_1$.

B. Model of Mutation Probability p_m

The expression of mutation probability p_m is given as follows:

$$p_m(k) = (1+\alpha_2) - ((1+\alpha_2) - \beta_2)^{g_2(k)} \tag{29}$$

where

α_2	minimum of p_m;
β_2	maximum of p_m;

k generation;
$g_2(k)$ function of generation, which may be linear or nonlinear as is $g_1(k)$.

We also assume

$$p_m(0) = \alpha_2, \quad p_m(\text{MaxGen}) = \beta_2$$

where $0 \le \alpha_2 \le \beta_2 \le 1$. If $\alpha_2 = \beta_2$, mutation probability becomes a constant, namely

$$p_m(k) = \alpha_2 = \beta_2.$$

In (28) and (29), $g_1(k)$ and $g_2(k)$ are dependent on the actual application. In our experiments, we use the functions

$$\begin{cases} g_1(k) = \sin\left(\dfrac{k\pi}{2\,\text{MaxGen}}\right) \\ g_2(k) = \sqrt{\dfrac{k}{\text{MaxGen}}} \end{cases} . \tag{30}$$

In general, at the beginning of evolution, we expect more individuals to be recombined by crossover operator and fewer individuals to be mutated. After the population has evolved a number of generations, the children chromosomes produced thereafter will possibly be very similar to each other and to their parents, thereby rendering crossover operators largely ineffective. To maintain the population diversity, we expect to increase the number of individuals that are performed by mutation operator and to decrease the number of individuals that are performed by crossover operator. Fig. 5 illustrates p_c and p_m. These formulas just give curves with roughly a desired shape and any other similar formula (such as parabolas) would be just as good. It is easy for us to draw such a conclusion from Fig. 5 that p_c slowly decreases from the maximum β_1 to the minimum α_1, where the speed of descent is considerably fast at first, then gradually stabilize; on the other hand, p_m slowly increases from the minimum α_2 to the maximum β_2, where the speed of ascent is considerably fast at first, then gradually stabilizes.

VII. Experimental Results

A. Constant-Coefficient Multipliers

High-speed digital multipliers are ubiquitous fundamental circuit components in DSP and most of the DSP computations involve the use of multiply-accumulate operations. Multiplication, however, is inherently a slow operation as a large number of partial products are added to produce the product. Especially, in applications such as DSP, this delay is unacceptable. Therefore, the design of high-performance multipliers is imperative and becoming a major concerned issue for modern circuit designers. The problem considered here is to generate efficient structures of combinational multipliers consisting of parallel counters. The target function is the constant-coefficient multiplier given by $y = Rx$, where R is an integer coefficient, x is the integer input, and y is the integer output. In the conventional design of constant-coefficient multipliers, one of the most important techniques is to encode the target coefficient R by the CSD number representation

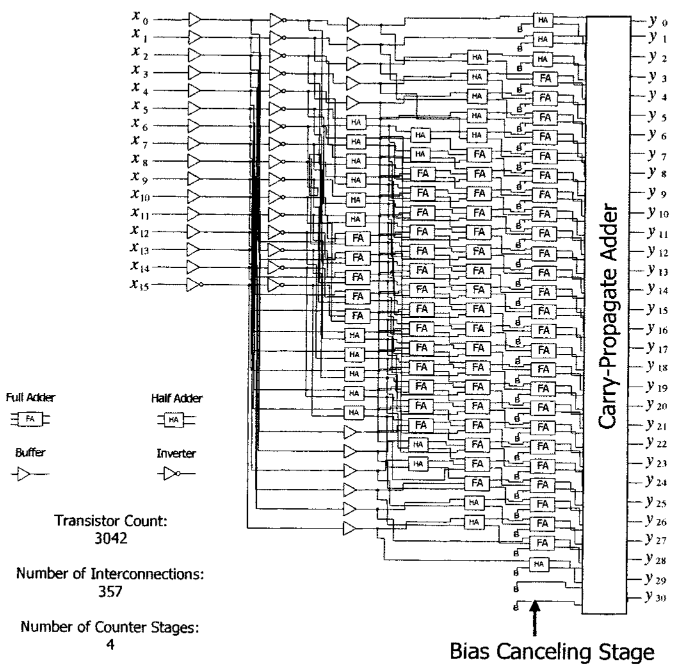


Fig. 9. Actual multiplier structure corresponding to the graph in Fig. 6.

[19], where the CSD number representation is defined as a specific binary signed-digit (SD) number representation that contains the least number of nonzero digits. The CSD encoding combined with the fast partial product accumulation technique using parallel counter trees is widely used in practical DSP applications, such as high-frequency finite-impulse response filter architectures [20]–[22]. As for compact counter-tree design for partial product accumulation, the authors' group has recently proposed the signed-weight (SW) arithmetic [23]. The use of SW arithmetic makes possible the construction of compact counter trees without using irregular arithmetic operations, such as sign extension and twos complementation. This property was confirmed in the design and fabrication of field-programmable digital filter architecture [23]. As a result, the combination of the CSD encoding technique with SW counter trees seems to currently provide the best possible approach to the practical hardware implementation of fast constant-coefficient multipliers. In [1], the authors successfully evolved 2-bit multipliers and 3-bit multipliers, but they did not show more complex multipliers. Thus, in order to evaluate the quality of solution, we decided to compare the 16-bit multipliers generated by the Arithmetic-EGG system with the multipliers manually designed by employing the knowledge of the above techniques (CSD plus SW).

Fig. 6 illustrates the solution generated by EGG employing the functional nodes listed in Table II, where the target coefficient R is $10\,075_{10}$. Fig. 7 shows the multiplier using Wallace-tree architecture consisting of the least number of stages of ordinary unsigned binary 3-2 counters. Fig. 8, on the other hand, illustrates the CSD multiplier using Wallace-tree architecture consisting the least number of stages of SW 3-2 counters. Compared with the designs illustrated in Figs. 7 and 8, the solution obtained from the EGG employs an SW 3-2 counter with 2-way output branches at the first stage and this hardware

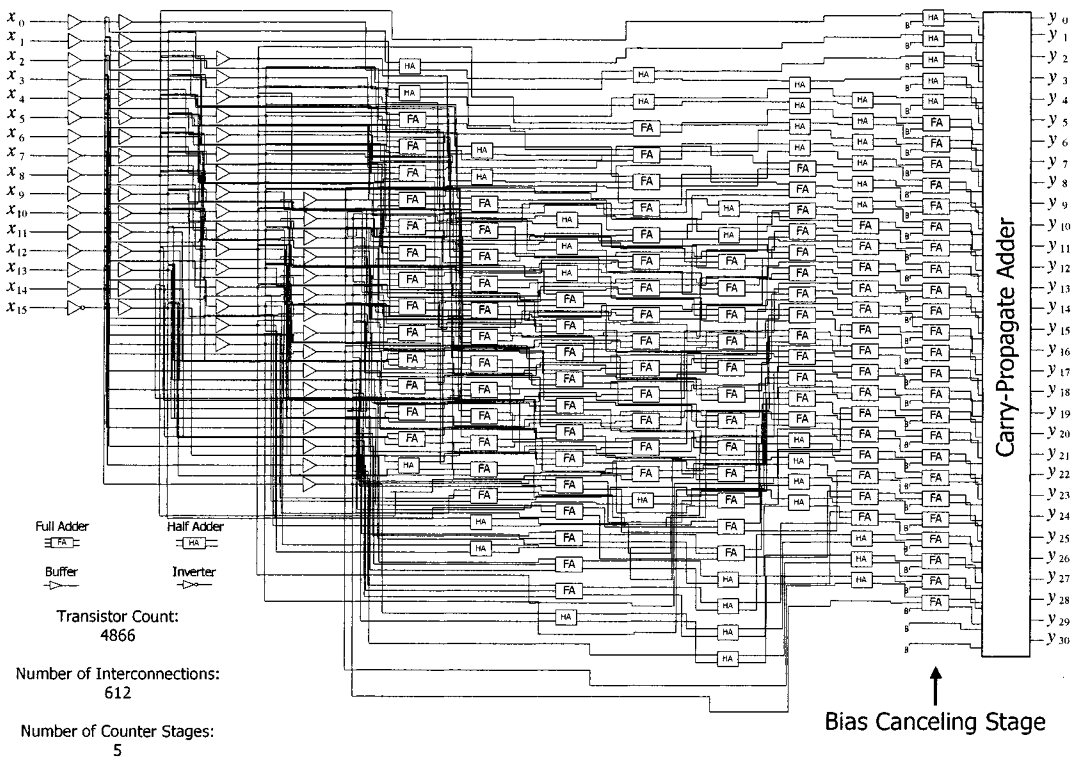


Fig. 10. Actual multiplier structure corresponding to the graph in Fig. 7.

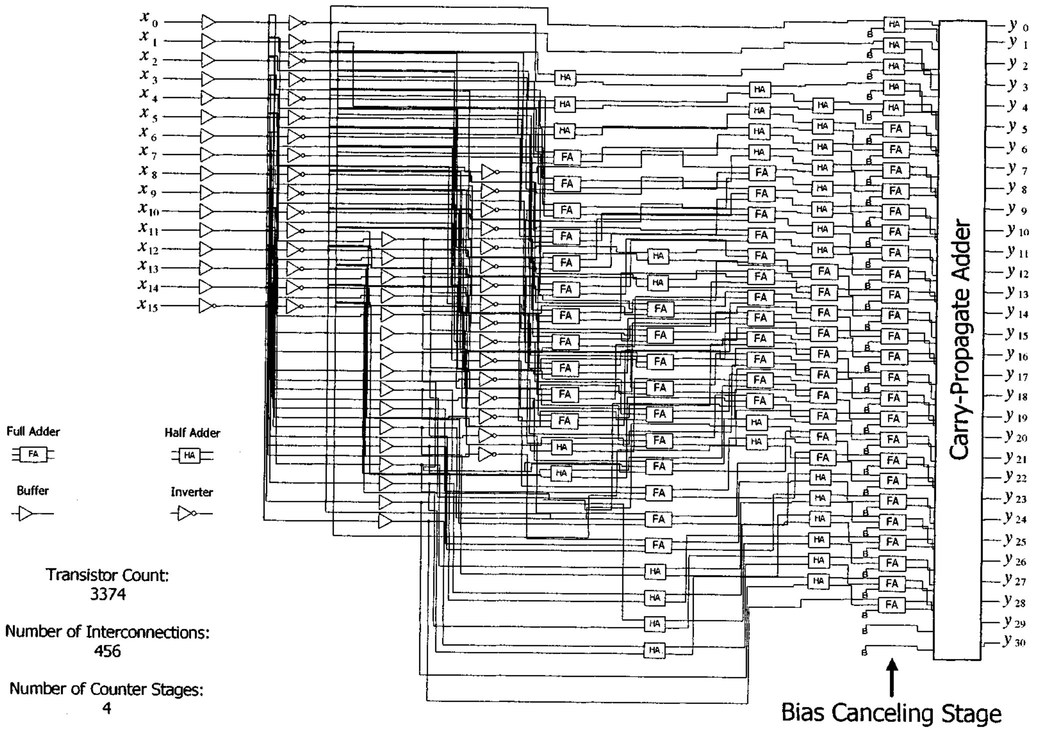


Fig. 11. Actual multiplier structure corresponding to the graph in Fig. 8.

resource is shared by the successive stage. This feature significantly reduces the complexity of the corresponding bit-level circuit configurations as shown in Figs. 9–11, respectively. Although there is no systematic way of synthesizing such sophisticated tree multipliers as shown in Fig. 6, the Arithmetic-EGG can naturally find so good structures through evolution process. This result adequately indicates that the Arithmetic-EGG can generate the efficient multiplier structures whose performance and complexity are comparable with those designed by experienced designers.

In order to further demonstrate the capability of the EGG system and the feasibility of proposed dynamic control scheme

TABLE IV
COMPARISON OF THE BEST SOLUTIONS OBTAINED FROM THE ARITHMETIC-ECG SYSTEM WITH THE KNOWN CONVENTIONAL DESIGNS

Size of Multiplier Coefficient	Arithmetic-EGG system						Known conventional designs	
	Case 1			Case 2			Case 3	Case 4
	Fitness Value	Product (A * D)	Rate of Success	Fitness Value	Product (A* D)	Rate of Success	Product (A* D)	Product (A * D)
971	105	315*4	100%	106	313*4	100%	433*5	317*4
8967	104	336*4	100%	105	336*4	100%	394*4	338*4
12345	104	348*4	100%	104	342*4	100%	411*4	347*4
19444	104	337*4	50%	103	335*4	60%	550*5	337*4
23719	103	526*7	10%	105	468*5	30%	564*5	469*5
27937	105	398*5	20%	106	398*4	10%	456*5	404*4
32168	107	331*4	100%	106	326*4	100%	547*5	330*4
33591	106	404*4	50%	106	396*4	60%	502*5	404*4
41123	103	427*4	60%	105	410*4	100%	427*4	425*4
45995	105	512*5	20%	106	518*5	30%	653*6	527*5
57091	105	359*4	90%	106	359*4	100%	588*5	361*4
59077	106	518*6	20%	107	477*5	30%	580*5	469*5
61073	104	405*4	40%	104	415*4	50%	572*5	413*4

TABLE V
MAIN CONTROL PARAMETER VALUES FOR THE CASE 1

Population size	1000	Crossover rate	0.85
Gross number of generations	1000	Mutation rate	0.4
Maximum number of nodes	50		

TABLE VI
MAIN CONTROL PARAMETER VALUES FOR THE CASE 2

Population size	1000	Gross number of generations	1000
Maximum crossover rate β_1	0.95	Minimum crossover rate α_1	0.75
Maximum mutation rate β_2	0.6	Minimum mutation rate α_2	0.02
Functions of generation g_1, g_2	Expression (30)	Maximum number of nodes	50

for operator probabilities, a group of experimental results with different coefficients are listed in Table IV, where main control parameters are described in Table V for Case 1 (with static operator probabilities) and Table VI for Case 2 (with dynamic control of operator probabilities), respectively. From Table IV, according to (27), it is easy to observe that, in most cases, the final solution quality and the rate of success obtained from the EGG using the dynamic mechanism of operator probabilities (Case 2) outperform those obtained from the EGG without using the dynamic mechanism (Case 1). This implies that the use of dynamic parameters itself can lead to better algorithm performance. The most important conclusion, however, is that the results obtained from the EGG system are quantitatively superior to or as good as the known conventional designs using arithmetic algorithms. The runtime cost of the EGG system is considerably dependent on the settings of control parameters, especially the population size and the maximum number of generation. With the configuration of Table VI and on a Pentium 700-MHz PC with memory size of 1 GB, it took 40 000–50 000 (seconds), namely, about 12 or 13 hours, for the Arithmetic-EGG to reach the termination condition for each run.

B. Bit-Serial Data-Parallel Adders

The bit-serial data-parallel arithmetic circuits are frequently used in the real-time DSP architectures. In order to demonstrate the effectiveness of the Arithmetic-EGG in the design of sequential arithmetic circuits, we choose the synthesis of bit-serial data-parallel adders with multiple operand inputs as an instance. The target function considered here is the n-operand bit-serial adder given by the (21) with $K_0 = K_1 = \cdots = K_n = 1$. In this experiment, we assume main control parameters as the population size is 100, the maximum number of generation is 3000, the maximum number of nodes is 30, the crossover rate is 0.7, and the mutation rate is 0.1. Fig. 12 shows the best individuals

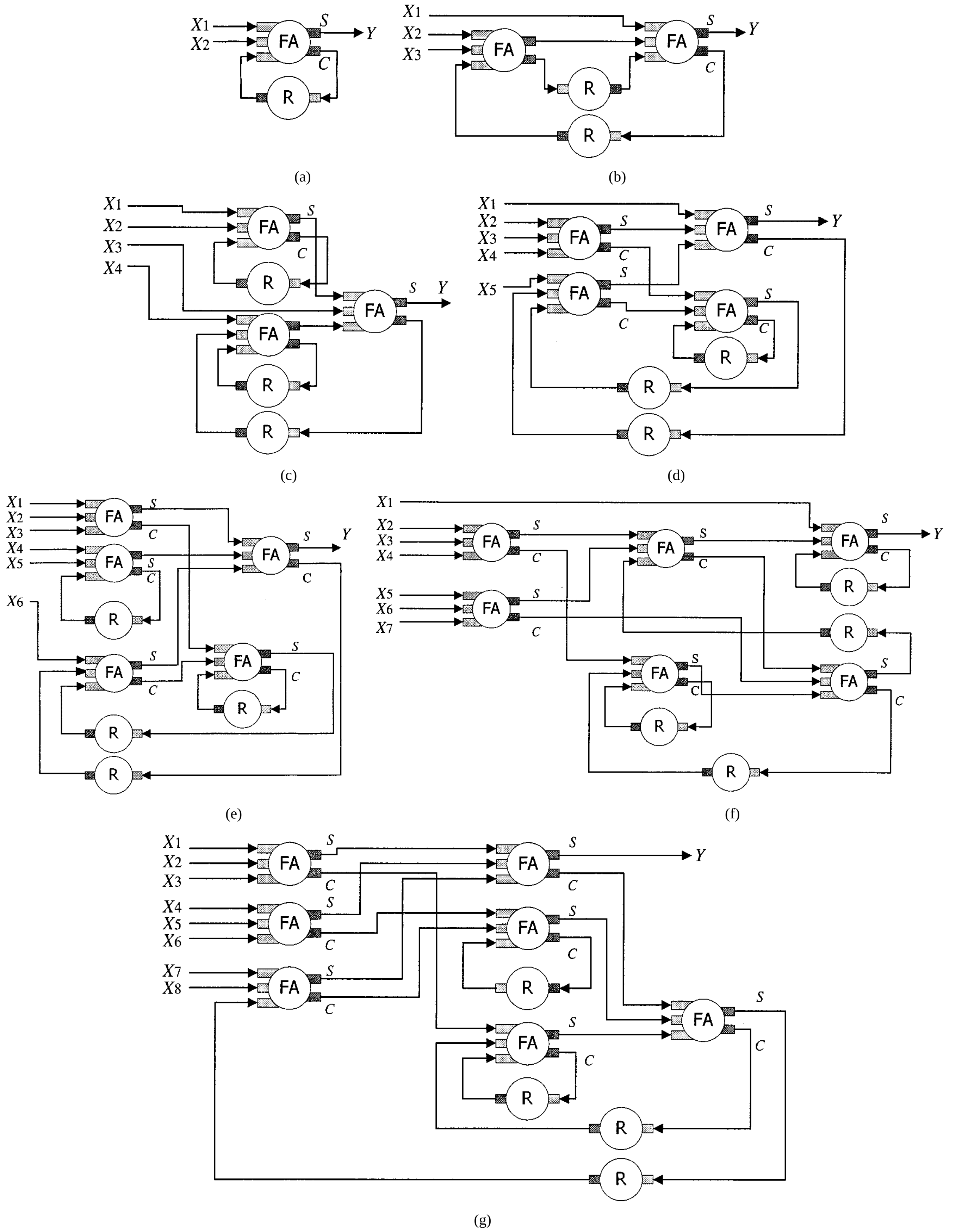
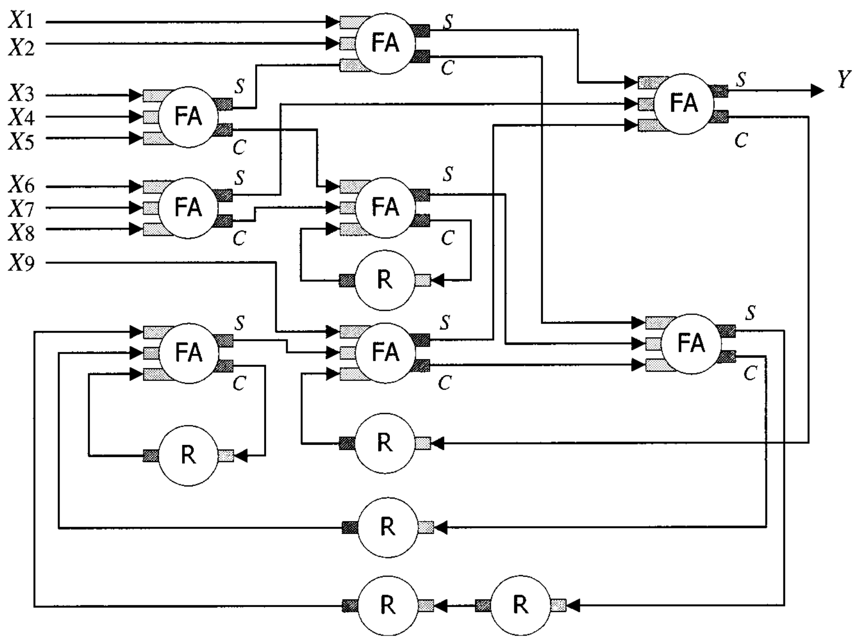
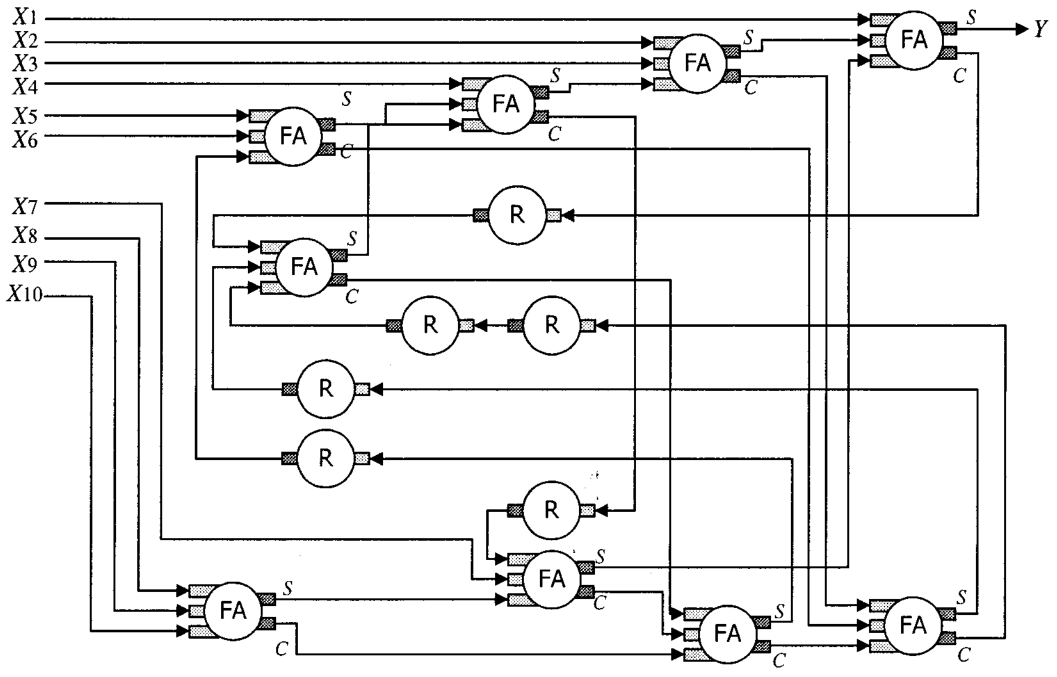


Fig. 12. Best individuals obtained in the 3000th generation, where the number of operands are (a) $n = 2$, (b) $n = 3$, (c) $n = 4$, (d) $n = 5$, (e) $n = 6$, (f) $n = 7$, and (g) $n = 8$.



(h)



(i)

Fig. 12. *(Continued.)* Best individuals obtained in the 3000th generation, where the number of operands are (h) $n = 9$ and (i) $n = 10$.

obtained in five runs for the number of operands ranging from two to ten.

For more detailed discussion, let us examine the evolution process of an 8-operand bit-serial adder as an example. Given the initial random population, the evolution is mainly driven toward better functionality. Each individual shows a tendency to keep a specific level of product $(A * D)$ corresponding to the target function. The first individual achieving 100% functionality appears in the 122nd generation. This individual has the product $(A * D)$ of 290. In the 3000th generation, we eventually yield the best adder configuration shown in Fig. 12(g), where the product $(A * D)$ is reduced to 156. It can be easily proved that this structure consists of the minimum number of stages of adders. Thus, we have demonstrated the capability of the EGG system to create the sequential arithmetic circuits through EGG with limited knowledge of computer arithmetic algorithms.

The proposed approach can be applied not only to the synthesis of multioperand adders but also to various sequential design specifications including multiply-adders by slightly changing the fitness function. For example, if we use the target function in (21) with the different parameters $n = 2$, $K_0 = 1$, $K_1 = 3$, $K_2 = 5$, we can synthesize the multiply-adder given by $Y = 3X + 5Y$. Fig. 13 shows the best solution obtained in

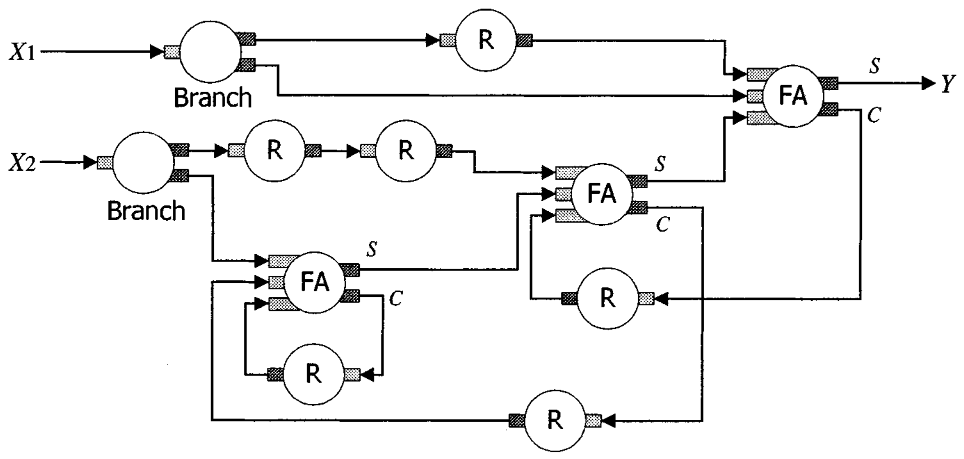


Fig. 13. Evolved multiply-addition structure under the target function $Y = 3X_1 + 5X_2$.

the 3000th generation. Although further investigations will be required, it is practically possible to construct the EGG-based arithmetic synthesis system that can deal with general circuit design problems.

VIII. Conclusion

In this paper, we have proposed an efficient graph-based evolutionary optimization technique called EGG and its application to the design of combinational and sequential arithmetic circuits. Novel chromosome representation and a functional verification technique based on symbolic model checking for arithmetic circuits are presented in order to apply the EGG system to the practical arithmetic design problems for constant-coefficient multipliers and bit-serial data-parallel adders. In fact, the proposed EGG method can easily be applied to the different arithmetic circuit design specifications by changing fitness functions. In addition, the effectiveness and feasibility of the generation-dependent dynamic control of operator probabilities have also been shown by a set of experiments.

The experimental designs of constant-coefficient multipliers and bit-serial data-parallel adders substantially demonstrate the potential capability of the EGG system to solve practical design problems of arithmetic circuits with limited knowledge of computer arithmetic algorithms. In the sample design of some constant-coefficient multipliers, by comparing some results with the optimal CSD multipliers based on Wallace-tree architecture consisting of SW counters, we found that the Arithmetic-EGG consistently performed better than those known conventional designs. This implies that the proposed EGG can help to simplify and speed up the process of designing arithmetic circuit and it is likely for us to obtain better solution to the given problem.

Further investigations are now being conducted to apply the proposed EGG technique to a wide variety of arithmetic design problems including those using unconventional number systems, such as redundant and high-radix number systems [24]–[28] and to implement a distributed EGG system by using message-passing interface techniques on the basis of PC cluster technology for further improvements of the final solution quality.

Acknowledgment

The authors would like to thank the anonymous referees and the associate editor that have proposed many valuable and suggestive comments on this manuscript.

References

[1] J. F. Miller, P. Thomson, and T. Fogarty, "Designing electronic circuits using evolutionary algorithms. Arithmetic circuits: A case study," in *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*. New York: Wiley, 1997, pp. 105–131.
[2] H. De Garis, "Evolvable hardware: Genetic programming of a Darwin machine," in *Artificial Neural Nets and Genetic Algorithms*, R. F. Albrecht, C. R. Reeves, and N. C. Steele, Eds. Heidelberg, Germany: Springer-Verlag, 1993, pp. 441–449.
[3] I. Kajitani, T. Hoshino, M. Iwata, and T. Higuchi, "Variable length genetic algorithms for evolvable hardware," in *Proc. 1996 IEEE Int. Conf. Evolutionary Computation*, May 1996, pp. 443–447.
[4] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Perez-Uribe, and A. Stauffer, "A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 83–97, Apr. 1997.
[5] D. B. Fogel, *Evolutionary Computation: The Fossil Record*, D. B. Fogel, Ed. Piscataway, NJ: IEEE Press, 1998.
[6] J. R. Koza, F. H. Bennett, III, D. Andre, M. A. Keane, and F. Dunlap, "Automated synthesis of analog electrical circuits by means of genetic programming," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 109–128, July 1997.
[7] T. Higuchi, M. Murakawa, M. Iwata, I. Kajitani, L. Weixin, and M. Salami, "Evolvable hardware at function level," in *Proc. 1997 IEEE Int. Conf. Evolutionary Computation*, Apr. 1997, pp. 187–192.
[8] N. Homma, T. Aoki, and T. Higuchi, "A new evolutionary approach for synthesizing circuit structures," in *Proc. 1999 Int. Symp. Nonlinear Theory and its Applications*, vol. 1, Nov. 1999, pp. 239–242.
[9] ——, "Evolutionary graph generation system with symbolic verification for arithmetic circuit design," *Inst. Electr. Eng. Electron. Lett.*, vol. 36, no. 11, pp. 937–939, May 2000.
[10] ——, "Evolutionary synthesis of fast constant-coefficient multipliers," in *IEICE Trans. Fundam.*, vol. E83-a, Japan, Sept. 2000, pp. 1767–1777.
[11] T. Terasaki, T. Aoki, and T. Higuchi, "Evolutionary synthesis of sequential arithmetic circuits," in *Proc. 2000 IEEE Int. Symp. Intelligent Signal Processing and Communication Systems*, Nov. 2000, pp. 1067–1072.
[12] H. Holland, *Adaptation in Natural and Artificial Systems*. Cambridge, MA: MIT Press, 1992.
[13] A. Globus, J. Lawton, and T. Wipke. JavaGenes: Evolving graphs with crossover. [Online]. Available: http://www.nas.nasa.gov/~globus/papers/JavaGenes/paper.html.
[14] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*. New York: Wiley, 1996.
[15] L. Davis, *Handbook of Genetic Algorithms*. New York: Van Nostrand, 1991.
[16] J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-16, pp. 150–157, Jan.–Feb. 1986.

[17] B. A. Julstrom, "What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm," in *Proceedings of the 6th International Conference on Genetic Algorithms*, L. Eshelman, Ed. San Mateo, CA: Morgan Kaufmann, 1995, pp. 81–87.
[18] A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 3, pp. 124–139, July 1999.
[19] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design*. New York: Wiley, 1979.
[20] B. C. Wong and H. Samueli, "A 200-MHz all-digital QAM modulator and demodulator in 1.2-μm CMOS for digital radio application," *IEEE J. Solid-State Circuits*, vol. 26, pp. 1970–1979, Dec. 1991.
[21] K. Khoo, A. Kwentus, and A. N. Willson, "A programmable FIR digital filter using CSD coefficients," *IEEE J. Solid-State Circuits*, vol. 31, pp. 869–874, June 1996.
[22] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II*, vol. 43, pp. 677–688, Oct. 1996.
[23] T. Aoki, Y. Sawada, and T. Higuchi, "Signed-weight arithmetic and its application to a field-programmable digital filter architecture," *IEICE Trans. Electron.*, vol. E82-C, no. 9, pp. 1687–1691, Sept. 1999.
[24] T. Aoki, H. Tokoyo, and T. Higuchi, "High-radix parallel dividers for VLSI signal processing," in *Proc. 1996 IEEE Workshop VLSI Signal Processing*, Oct. 1996, pp. 83–92.
[25] T. Aoki, H. Amada, and T. Higuchi, "Real/complex re-configurable arithmetic using redundant complex number systems," in *Proc. 13th IEEE Symp. Computer Arithmetic*, July 1997, pp. 200–207.
[26] T. Aoki, H. Nogi, and T. Higuchi, "High-radix CORDIC algorithms for VLSI signal processing," in *Proc. 1997 IEEE Workshop Signal Processing Systems*, Nov. 1997, pp. 183–192.
[27] T. Aoki and T. Higuchi, "Beyond-binary arithmetic: Algorithms and VLSI implementations," *Interdisciplinary Info. Sci.*, vol. 6, no. 1, pp. 75–98, 2000.
[28] T. Aoki, I. Kitaori, and T. Higuchi, "Radix-2-4-8 CORDIC for fast vector rotation," *IEICE Trans. Fundam.*, vol. E83-a, no. 6, p. 1106, June 2000.

Dingjun Chen received the Ph.D. degree in specialty of computer application technology from Northeastern University, Shenyang, China, in 1999.

He is currently with the Graduate School of Information Sciences, Tohoku University, Sendai, Japan, as a JSPS Postdoctoral Fellow. His current research interests include evolutionary computation and its applications and parallel and distributed processing in message-passing interface.

Takafumi Aoki (M'91) received the B.E., M.E., and D.E. degrees in electronic engineering from Tohoku University, Sendai, Japan, in 1988, 1990, and 1992, respectively.

He is currently an Associate Professor with the Graduate School of Information Sciences, Tohoku University. From 1997 to 1999, he was with the PRESTO research project, Japan Science and Technology Corporation. His current research interests include theoretical aspects of computation, VLSI computing structures for signal and image processing, multiple-valued logic, and biomolecular computing.

Dr. Aoki received the Outstanding Paper Award at the IEEE International Symposium on Multiple-Valued Logic in 1990, 2000, and 2001, the Outstanding Transactions Paper Award from the Institute of Electronics, Information, and Communication Engineers (IEICE) of Japan in 1989 and 1997, the Institution of Electrical Engineers (IEE) Ambrose Fleming Premium Award in 1994, the IEICE Inose Award in 1997, the IEE Mountbatten Premium Award in 1999, and the Best Paper Award at the 1999 IEEE International Symposium on Intelligent Signal Processing and Communication Systems.

Naofumi Homma (S'00) received the B.E. degree in information engineering and the the M.E. and D.E. degrees in information sciences from Tohoku University, Sendai, Japan, in 1997, 1999, and 2001, respectively. He is currently working toward the Ph.D. degree at the same university.

From 1999 to 2001, he was a Research Fellow of the Japan Society for the Promotion of Science. His current research interests include arithmetic algorithms and their design using evolutionary optimization techniques.

Toshiki Terasaki received the B.E. degree in information engineering and the M.E. degree in information sciences from Tohoku University, Sendai, Japan, in 1999 and 2001, respectively.

His research interests include evolutionary computation and electronic design automation technology.

Tatsuo Higuchi (M'70–SM'82–F'92) received the B.E., M.E., and D.E. degrees in electronic engineering from Tohoku University, Sendai, Japan, in 1962, 1964, and 1969, respectively.

He was a Professor with the Department of Electronic Engineering, Tohoku University, from 1980 to 1993 and was a Dean of the Graduate School of Information Sciences, Tohoku University, from 1994 to 1998, where he is currently a Professor. His general research interests include the design of one-dimensional and multidimensional digital filters, linear time-varying system theory, fractals and chaos in digital signal processing, VLSI computing structures for signal and image processing, multiple-valued integrated circuits (ICs), multiwave opto-electronic ICs, and biomolecular computing.

Dr. Higuchi is a fellow of the Institute of Electronics, Information and Communication Engineers (IEICE) and the Society of Instrument and Control Engineers (SICE). He received the Outstanding Paper Awards at the 1985, 1986, 1988, 1990, 2000, and 2001 IEEE International Symposium on Multiple-Valued Logic, the Outstanding Transactions Paper Award from the SICE of Japan in 1984, the Technically Excellent Award from SICE in 1986, and the Outstanding Book Award from SICE in 1996, the Outstanding Transactions Paper Award from the IEICE of Japan in 1990 and 1997, the Inose Award from IEICE in 1997, the Technically Excellent Award from the Robotics Society of Japan in 1990, the Institution of Electrical Engineers (IEE) Ambrose Fleming Premium Award in 1994, the Outstanding Book Award from the Japanese Society for Engineering Education in 1997, the Award for Persons of scientific and technological merits (Commendation by the minister of state for Science and Technology), the IEE Mountbatten Premium Award in 1999, the Best Paper Award at the 1999 IEEE International Symposium on Intelligent Signal Processing and Communication Systems, and the IEEE Third Millennium Medal in 2000.