Study on Multi-objective Optimization of Circuit Design by Evolutionary Computation Technologies

Bao, Zhiguo

April 2011

Waseda University Doctoral Dissertation

Study on Multi-objective Optimization of Circuit Design by Evolutionary Computation Technologies

Bao, Zhiguo

Graduate School of Information, Production and Systems
Waseda University

April 2011

Abstract

Study on Multi-objective Optimization of Circuit Design
by Evolutionary Computation Technologies

BAO, Zhiguo

Evolvable Hardware (EHW) has been researched for hardware design since early 1990s. It is classified into two categories: evolutionary circuit design and adaptive systems. Evolutionary circuit design uses evolutionary algorithms (EAs) to design a system that meets a predefined specification, and adaptive systems reconfigure an existing design to counteract faults or to adapt to a variable operational environment. EHW can be used as an alternative to conventional hardware design methodology. This research field has been actively studied by many researchers since Field Programmable Gate Array (FPGA) appeared as a hardware device. EHW techniques combined with FPGAs seems to be successful and promising in various applications, because it could automatically generate digital circuits by using EAs. EHW has been used to design digital circuits, such as a multiplier, a neural network, a robot controller, a traffic signal classifier, a digital image filter and so on. However, there still remain critical issues such as scalability, maintainability and generalization to apply EHW for practical design problems. One of them is a circuit optimization with mixed design constraints.

Therefore, in this research, the circuit design for mixed constraints is discussed, and a new design methodology using several evolutionary computation technologies is proposed. As a result, it is shown that our proposed design methodology can efficiently produce good circuits to meet the mixed design constraints.

The organization of the thesis is as follows:

In chapter 1, the evolutionary computation technologies are introduced, such as

Genetic algorithm (GA) or Particle Swarm Optimization (PSO). Evolutionary computation involves combinatorial optimization problems. GA is a search heuristic that mimics the process of natural evolution. This heuristic is routinely used to generate good solutions to optimization and search problems. PSO is a computational method that optimizes a problem by iteratively improving a candidate solution, and it is a method for performing numerical optimization without explicit knowledge of the gradient of the problem. EHW has an interesting application of evolutionary computation technologies. We focus on mixed constrained circuit design problems where evolutionary computation technologies are used to efficiently optimize circuit design and also to deal with the fault tolerance circuit design.

In chapter 2, GA is applied to gate level circuit design optimization. We propose optimal circuit design by using GA with parameterized uniform crossover (GApuc) and with fitness function composed of circuit complexity, power and signal delay. Parameterized uniform crossover is much more likely to distribute its disruptive trials in an unbiased manner over larger portions of the space. It has higher exploratory power than one- and two-point crossover and we have more chances to find better solutions.

Its effectiveness is verified by experiments. From the results, we can see that the best elite fitness, the average value of fitness of the correct circuits and the number of the correct circuits of GApuc are better than that of GA with one- or two-point crossover. The best circuits generated by GApuc are 10.18% and 6.08% better in evaluating value compared with those by GA with one- and two-point crossover, respectively.

Chapter 3 proposes GA with different structure selection (GAdss) and its application to autonomous design optimization for combinatorial circuits.

In traditional GA, the tournament selection for crossover and mutation is based

on fitness of individuals. It can make convergence easily, but maybe lose some useful genes. In selection, besides on fitness, we consider the different structure from individuals comparing to the elite one. First, some individuals are selected using more different structures, then crossover and mutation are performed for them to generate new individuals. By this way, GA can increase diversification to searching spaces, so that it can find better solution. By evolution, GAdss can find optimized circuits with less complexity, less power and less signal delay than traditional GA. From the results, we can see that the best elite fitness, the average value of fitness of correct circuits and the number of correct circuits of GAdss are better than traditional GA. The best case of optimal circuits generated by GAdss is 8.1% better in evaluating value than that by traditional GA.

Chapter 4 describes mixed constrained image filter design for noise reduction using a GApuc (Genetic Algorithm with parameterized uniform crossover) on a reconfigurable processing array. The complexity, power and signal delay in Configurable Logic Blocks (CLBs) and wires are considered. An image filter for noise reduction is experimentally synthesized to verify the validity of the proposed method. By evolution, quality of the optimized image filter on reducing salt-and-pepper noise is better than that of other papers. Consequently it is shown that the proposed design method is effective in mixed constrained image filter design for salt-and-pepper noise reduction.

Chapter 5 proposes the use of PSO to design a mixed constrained image filter. In order to further reduce the processing time, PSO is used instead of GA. An image filter is experimentally synthesized using PSO to verify the effectiveness of our proposed method. By using evolution process, the quality of an optimized image filter by PSO is almost same as that of GA, but the running time by PSO is 10% shorter than that of GA.

Chapter 6 describes mixed constrained image filter design with fault tolerance for

noise reduction using GA on a reconfigurable processing array. Some CLBs (Configurable Logic Blocks) in a reconfigurable processing array are set fault at random. The proposed method with GA autonomously synthesizes a filter fitted to the reconfigurable device with some faults, evaluating the complexity, power and signal delay. An image filter for noise reduction is experimentally synthesized to verify our method. By evolution, the quality of an optimized image filter on a reconfigurable processing array with faults is almost same as that on a reconfigurable processing array with no fault. Consequently our proposed design method is also effective for fault-tolerant optimization.

Finally, chapter 7 concludes that the proposed method by GApuc can produce better circuits compared with other researches, and that PSO can get the almost same circuit in shorter processing time. Furthermore, the quality of an optimized image filter on a reconfigurable processing array with some faults is almost same as that on a reconfigurable processing array with no faults.

We will also apply evolutionary computation technologies to autonomous design circuits for more complex functional requirements, and enhance more practical information about circuit to fitness function. The future subject is to develop an adaptive system which can reconfigure an existing design by evolutionary computation technologies in order to meet a variable operational environment.

Keywords: *Genetic Algorithm, Particle Swarm Optimization, Circuit optimization, Multi-objective optimization of circuit design, Image filter design for noise reduction, fault tolerance circuit design*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Evolutionary Computation

In computer science, evolutionary computation [1–3] is a subfield of artificial intelligence (more particularly computational intelligence) that involves combinatorial optimization problems.

The use of Darwinian principles for automated problem solving originated in the fifties. It was not until the sixties that three distinct interpretations of this idea started to be developed in three different places. Evolutionary programming (EP) was introduced by Lawrence J. Fogel in the USA, while John Henry Holland called his method a genetic algorithm (GA). In Germany, Ingo Rechenberg and Hans-Paul Schwefel introduced evolution strategies (ES). These areas developed separately for about 15 years. From the early nineties, they are unified as different representatives of one technology, called evolutionary computing. Also in the early nineties, a fourth stream following the general ideas had emerged - genetic programming (GP). These terminologies denote the field of evolutionary computing and consider evolutionary programming, evolution strategies, genetic algorithms, and genetic programming as sub-areas.

Evolutionary computation uses iterative progress, such as growth or development in a population. This population is then selected in a guided random search using parallel processing to achieve the desired end. Such processes are often inspired by biological mechanisms of evolution.

1.1.1 Evolutionary Algorithm

In artificial intelligence, an evolutionary algorithm (EA) [4–6] is a subset of evolutionary computation, a generic population-based meta heuristic optimization algorithm. An EA uses some mechanisms inspired by biological evolution: reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the environment within which the solutions "live". Evolution of the population then takes place after the repeated application of the above operators.

EAs often perform well approximating solutions to all types of problems because they ideally do not make any assumption about the underlying fitness landscape; this generality is shown by successes in fields as diverse as engineering, art, biology, economics, marketing, genetics, operations research, robotics, social sciences, physics, politics and so on.

Genetic algorithm (GA) [7, 8] is the most popular type of EA. The GA is a search heuristic that mimics the process of natural evolution. This heuristic is routinely used to generate useful solutions to optimization and search problems. GAs belong to the larger class of EA, which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

In GA, a population of strings (called chromosomes or the genotype of the genome), which encode candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem, evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly

randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

1.1.2 Swarm Intelligence

Swarm intelligence (SI) describes the collective behaviour of decentralized, self-organized systems, natural or artificial. The concept is employed in work on artificial intelligence. The expression was introduced by Gerardo Beni and Jing Wang in 1989, in the context of cellular robotic systems [9]. SI systems are typically made up of a population of simple agents interacting locally with one another and with their environment. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents lead to the emergence of "intelligent" global behavior, unknown to the individual agents.

Particle swarm optimization (PSO) [10–12] is a method for performing numerical optimization without explicit knowledge of the gradient of the problem to be optimized.

PSO optimizes a problem by maintaining a population of candidate solutions called particles and moving these particles around in the search-space according to simple formulae. The movements of the particles are guided by the best found positions in the search-space, which are continually updated as better positions are found by the particles.

1.2 Evolvable Hardware

Evolvable hardware (EHW) [13–19] is a new field about the use of EAs to create specialized electronics without manual engineering, as shown in Figure 1.1. It brings to-

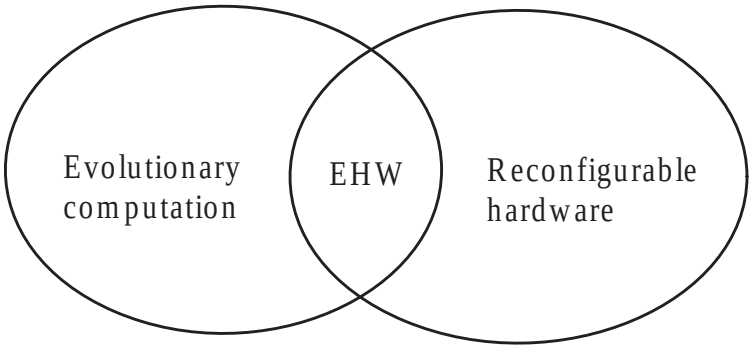


Figure 1.1: EHW combines evolutionary techniques and a reconfigurable hardware.

gether reconfigurable hardware, artificial intelligence, fault tolerance and autonomous systems. EHW refers to hardware that can change its architecture and behavior dynamically and autonomously by interacting with its environment.

In its most fundamental form, an EA manipulates a population of individuals where each individual describes how to construct a candidate circuit. Each circuit is assigned a fitness, which indicates how well a candidate circuit satisfies the design specification. The EA uses stochastic operators to evolve new circuit configurations from existing ones. Done properly, over time the EA will evolve a circuit configuration that exhibits desirable behavior.

Each candidate circuit can be either simulated or physically implemented in a reconfigurable device. Typical reconfigurable devices are field-programmable gate arrays (FPGA, for digital designs) or field-programmable analog arrays (FPAA, for analog designs). At the lower level of abstraction, they are field-programmable transistor arrays (FPTA) that can implement either digital or analog designs.

The concept was pioneered by Adrian Thompson at the University of Sussex, England, who evolved a tone discriminator using fewer than 40 programmable logic gates and no clock signal in a FPGA in 1996. This is a remarkably small design for such a device and relied on exploiting peculiarities of the hardware that engineers normally avoid. For example, one group of gates has no logical connection to the rest

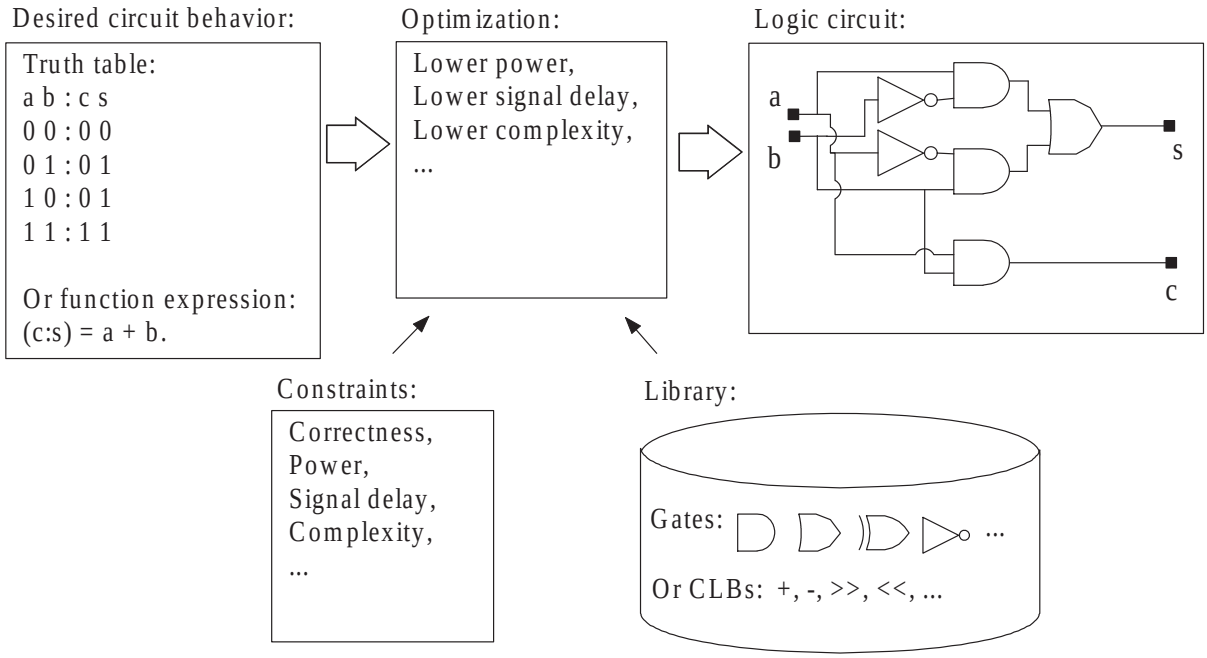


Figure 1.2: An example of mixed constrained circuit design optimization.

of the circuit, yet is crucial to its function.

EHW problems fall into two categories: original design and adaptive systems. Original design uses EAs to design a system that meets a predefined specification. Adaptive systems reconfigure an existing design to counteract faults or a changed operational environment.

1.3 Mixed Constrained Circuit Design

Circuit design flow in our proposed system is shown in Fig. 1.2, where an abstract form of desired circuit behavior (truth table or function expression) is finally turned into a design implementation in terms of logic gates or CLBs (Configurable Logic Blocks of FPGA). Circuit optimization is the process of finding an equivalent representation of the specified logic circuit under one or more specified constraints. Generally the circuit is constrained to correctness, less complexity, less power, and less signal delay.

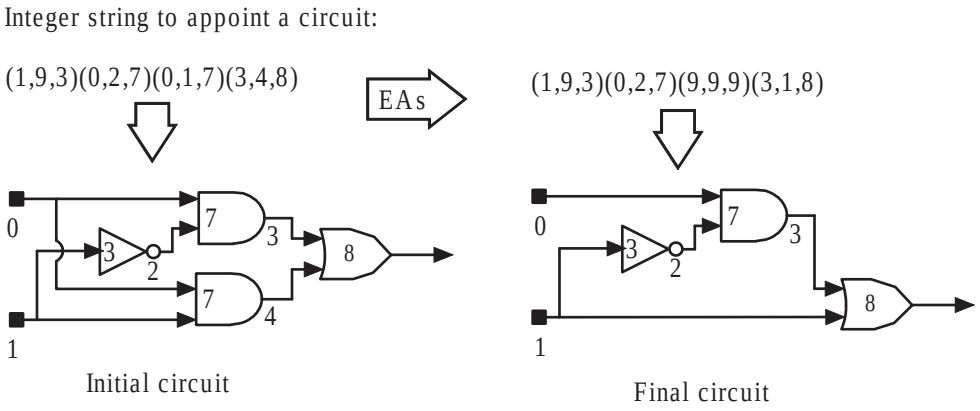


Figure 1.3: An example of evolutionary circuit design.

Thus, we need to consider the attribute of gates or CLBs, such as complexity, power, and signal delay. In mixed constrained circuit design optimization, it needs more processing time, when the number of gates or CLBs is larger.

Evolutionary circuit design uses EAs to design a circuit that meets a predefined specification. First, an integer string is used to appoint a circuit. Then, the integer string is changed by EAs to get better circuit. In the end, a desired circuit can be created, as shown in Figure 1.3.

1.4 Thesis Organization

The contents of following chapters are shown in Table 1.1.

In chapter 2, we propose a new method for circuit design optimization by GA, where mixed constraints on circuit complexity, power, and signal delay are considered. First, we introduce the evaluating value about correctness, complexity, power, and signal delay to the fitness function. Then GA can autonomously synthesize a circuit whose function is equivalent to a conventional design, but which is simpler and has better performance.

One-point crossover, two-point crossover, and parameterized uniform crossover

Table 1.1: Contents of following chapters.

Item	Method	Level	Circuit	Optimization	Fault-tolerant
Chapter 2	GA	Gate	Adder	Yes	No
Chapter 3	GAdss	Gate	Adder	Yes	No
Chapter 4	GA	Function	Image filter	Yes	No
Chapter 5	PSO	Function	Image filter	Yes	No
Chapter 6	GA	Function	Image filter	Yes	Yes

[20] are typical crossover methods used in GA. Parameterized uniform crossover is much more likely to distribute its disruptive trials in an unbiased manner over larger portions of the space, then it has more exploratory power than one and two-point crossover. Thus it enhances more diversification to searching spaces, then it has more chances of finding better solution. Therefore we use GA with parameterized uniform crossover, named GApuc.

In chapter 3, we propose a new approach for circuit design optimization by GA with Different Structure Selection (GAdss), where mixed constraints on circuit complexity, power and signal delay are considered. We introduce the evaluating value about correctness, complexity, power and signal delay for the fitness function in order to meed the mixed constrains. In the first step, the fitness function is used to find the solutions with 100% correctness of the target circuit, and with maximal evaluating values about complexity, power and signal delay. Then, GAdss can autonomously synthesize a circuit that is equivalent to a conventional design in functionality, but is simpler and has better performance. As a result, GAdss can find a better circuit, compared to traditional GA. To verify an effectiveness of our approach, a simple 2-bit half adder circuit is experimentally synthesized.

Chapter 4 describes mixed constrained image filter design for noise reduction using

a GA where parameterized uniform crossover is adopted on a reconfigurable processing array. The circuit complexity, power and signal delay in both logic blocks and wires are optimized. In this design, first, the evaluating values about correctness, complexity, power and signal delay are introduced to the fitness function. Then GA autonomously synthesizes an image filter which is simple and has better performance. To verify the validity of the proposed method, an image filter for noise reduction is experimentally synthesized.

Chapter 5 applies PSO to mixed constrained image filter design for noise reduction. The circuit complexity, power and signal delay which are caused by both logic gates and wires, are optimized. In this design, first, the evaluating value about correctness, complexity, power and signal delay are introduced to the fitness function. Then PSO autonomously synthesizes an image filter which is simpler and has better performance than the conventional design. To verify the validity of our method, an image filter for reducing noise is experimentally synthesized.

Chapter 6 describes mixed constrained image filter design with fault tolerant using GA on a reconfigurable processing array. There may be some faulty Configurable Logic Blocks (CLBs) in a reconfigurable processing array at random. The proposed method with GA autonomously synthesizes a filter fitted to the reconfigurable device with some faults, evaluating the complexity, power and signal delay in both CLBs and wires. An image filter for noise reduction is experimentally synthesized to verify the validity of our method. By evolution, the quality of the optimized image filter on a reconfigurable device with faults is almost same as that with no fault.

Finally, this paper concludes with a summary of the results in chapter 7.

Chapter 2

GATE LEVEL CIRCUIT DESIGN OPTIMIZATION USING GENETIC ALGORITHM WITH PARAMETERIZED UNIFORM CROSSOVER

We propose optimal circuit design by using GA with parameterized uniform crossover (GApuc) and with fitness function composed of circuit complexity, power, and signal delay. Parameterized uniform crossover is much more likely to distribute its disruptive trials in an unbiased manner over larger portions of the space, then it has more exploratory power than one and two-point crossover, so we have more chances of finding better solutions. Its effectiveness is verified by experiments. From the results, we can see that the best elite fitness, the average value of fitness of the correct circuits and the number of the correct circuits of GApuc are better than that of GA with one-point crossover or two-point crossover. The best case of optimal circuits generated by GApuc is 10.18% and 6.08% better in evaluating value than that by GA with one-point crossover and two-point crossover, respectively.

2.1 Introduction

EAs is a generic population-based heuristic optimization algorithm. It uses some mechanisms inspired by biological evolution, such as selection, crossover, mutation, and replacement.

One of the interesting application of EAs, called EHW [13–19] has been researched for hardware design since early 1990s'. It is classified into two categories: evolutionary circuit design (or named "original design") and adaptive systems. Evolutionary circuit design uses EAs to design a system that meets a predefined specification, and

Table 2.1: The related researches.

Year	Author	Level	Circuit	Method	Fitness	Experiment
1992	J. R. Koza et al.	Gate	Boolean 11-multiplexer	GP	Correctness	Software
1994	T. Higuchi et al.	Gate	Arbitration logical circuit	GA	Correctness	PLDs
2000	D. Keymeulen et al.	Transistor	Fault-tolerant circuit (XNOR, multiplier)	GA	Correctness	FPTA
2002	J. F. Miller et al.	Gate	4*4-Bit Multiplier	GP	Correctness	Software
2003	K. A. Vinger et al.	Function	FIR-filter	GA	Correctness	Xilinx Virtex XCV1000 FPGA
2004	J. Torresen et al.	Function	Sign Number Recognition	GA	Correctness	Software
2004	Y. Zhang et al.	Function	Image filter	GP	Correctness	Xilinx Virtex XCV1000 FPGA
2007	E. Benkhelifa et al.	Gate	3-bit adder	GA	Correctness and Number of gate	Software
2007	L. Sekanina et al.	Function	Image filter	GA	Correctness	Xilinx Virtex II Pro FPGA

FIR-filter: Finite input response filter.

adaptive systems reconfigure an existing design to counteract faults or to adapt to a variable operational environment. EHW can be used as an alternative to conventional hardware design methodology. This field has been actively investigated by a number of researchers such as Higuchi [21], Julian Miller [22] and Adrian Thompson [23] since FPGA appeared first as a hardware device. The application of the EHW technique seems to be successful and promising, because it could automatically generate digital circuits by using EAs. EHW has been used to design digital filter [24], neural network chip [25], robot controller [26], multiplier [27], traffic signs classifier [28], digital image filter [29], polymorphic digital circuits [30] and so on. The related researches are listed in Table 2.1. However, there still remain critical issues such as scalability, maintainability and generalization [31–33] to apply EHW for practical design problems. One of them is a circuit optimization problem, where mixed design constraints are subjected.

In this chapter, we propose a new method for circuit design optimization by GA [7, 8] which is one of typical EAs, where mixed constraints on circuit complexity, power, and signal delay are considered. First, we introduce the evaluating value about

correctness, complexity, power, and signal delay to the fitness function. Then GA can autonomously synthesize a circuit whose function is equivalent to a conventional design, but which is simpler and has better performance.

One-point, two-point crossover, and parameterized uniform crossover [20] are typical crossover methods used in GA. Parameterized uniform crossover is much more likely to distribute its disruptive trials in an unbiased manner over larger portions of the space, then it has more exploratory power than one and two-point crossover. Thus it enhances more diversification to searching spaces, then we have more chances of finding better solution. Therefore we use GA with parameterized uniform crossover, named GApuc.

To verify the effectiveness of our method, a simple 2-bit full adder circuit is experimentally synthesized.

Section 2.2 describes the use of GApuc as a new method for the automatic design of an optimized circuit. Section 2.3 shows experiments on a 2-bit full adder circuit design as an example. Finally, this chapter concludes with a summary of the results in Sect. 2.4.

2.2 Gate Level Circuit Design Optimization using GApuc

2.2.1 Objective

The overall objective is to discover novel solutions by the application of GApuc in the circuit design optimization process. The target circuit has to provide identical functional behavior equivalent to the specification, but requires less complexity, less power, and less signal delay.

In this section, we will demonstrate the principle of GApuc in the circuit design process using a 2-bit full adder as a sample logic circuit.

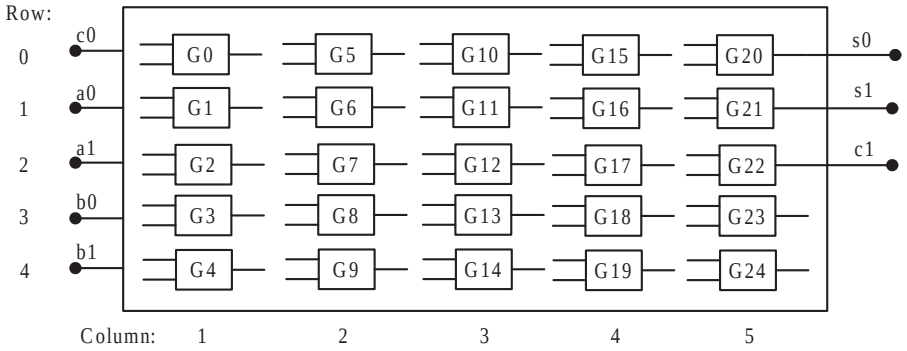


Figure 2.1: An initial 5*5 array with the input/output function for a 2-bit full adder.

2.2.2 Genetic Encoding

To process a genetic encoding easily, the logic circuit under consideration is assumed to be organized on a two dimensional array of gates, which was proposed in [19], shown in Fig. 2.1. Each gate accepts two inputs and produces one output according to its logical function. The gates in the first column of the array are set with predefined inputs. For the purpose of this experiment, the combinatorial circuit takes five primary inputs. Therefore there are 32 input patterns of the circuit. Gates in the following columns receive outputs from gates in the preceding columns.

The chromosome is a string of integers where each three continuous genes constitute a gate. Each triplet of the gate encodes the two inputs and the type of a gate, respectively, such as:

$$(Input\_1, Input\_2, Gate\_type).$$

In this experiment using a 2-bit full adder, the last two gates are not used, so the chromosome length is calculated by the following equation:

$$3 * ((numberofcolumns) * (numberofrows) - 2).$$

Table 2.2: Information of gates.

GT	LF	GC	EC	Power	EP	SD	ESD
0	NAND	4	6	3	7	4	6
1	NOR	4	6	3	7	4	6
2	XNOR	8	2	4	6	6	4
3	NOT(in1)	2	8	2	8	3	7
4	NOT(in2)	2	8	2	8	3	7
5	WIRE(in1)	0	10	6	4	8	2
6	WIRE(in2)	0	10	6	4	8	2
7	AND	6	4	5	5	7	3
8	OR	6	4	5	5	7	3
9	XOR	8	2	4	6	6	4
-	(not used)	0	20	0	20	0	20

GT: gate type.

LF: logical function.

GC: gate complexity.

EC: evaluation of complexity.

EP: evaluation of power.

SD: signal delay.

ESD: evaluation of signal delay.

(2.1)

In the experiment, the array is a fixed size of 5*5 gates (shown in Fig. 2.1), thus the length of the chromosome is 69 which is caused by $(3*(5*5-2))$. The inputs of each gate in the first column of the array can take the value of any integer in the range of [0, $(max\_number\_inputs-1)$]. On the other hand, gates on the other columns can take any integer value in the range of [0, $((now\_columns-1)*(numberofrows)-1)$]. As for the third gene in the triplet, gate type is defined as shown in Table 2.2, which was proposed in [34, 35].

In order to judge the difference of the complexity, power, and signal delay between different gates, we assign values about complexity, power, and signal delay to each gate. In the evolution, the larger the fitness is, the better the circuit is. So we use evaluating values about complexity, power, and signal delay in fitness function. In Table 2.2, "GC" is the complexity about CMOS circuit of one gate, defined by the number of CMOS, "EC" equals to $(10 - GC)$. "$power$" is the value about power of one gate, defined by the power of CMOS circuit, "EP" equals to $(10 - power)$. "SD" is the value about signal delay of one gate, defined by the signal delay of CMOS circuit, and "ESD" equals to $(10 - SD)$.

A typical chromosome then can be a sequence of triplets such as:

$$([0, X], [0, X], [0, 9])...([0, X], [0, X], [0, 9]).$$

Here, X means a range of a position of the corresponding signal. For primary input, $X = 4$. For input from output of a gate Gm shown in Fig. 2.1, $X = m$.

2.2.3 Fitness Function

The fitness function in this experiment aims at accepting solutions with 100% correctness of the target circuit, and with maximal evaluating values about complexity, power, and signal delay. We use two functions F_1 and F_2. The former is a ratio of correct outputs to all test data, and the latter is an evaluating function of circuit complexity, power, and signal delay. The following equations show how the fitness of individuals is calculated [34, 35]:

$$F_1 = \frac{num\_rightout * 100}{num\_trainingdata}. \tag{2.2}$$

num_rightout: the number of correct outputs generated from circuit individual.

Table 2.3: A truth table of an adder with 5 inputs and 3 outputs.

inputs					*outputs*		
a_1	a_0	b_1	b_0	c_0	c_1	s_1	s_0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	1
0	0	0	1	1	0	1	0
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	1
0	0	1	1	0	0	1	1
0	0	1	1	1	1	0	0
0	1	0	0	0	0	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	0	1	0
0	1	0	1	1	0	1	1
0	1	1	0	0	0	1	1
0	1	1	0	1	1	0	0
0	1	1	1	0	1	0	0
0	1	1	1	1	1	0	1
1	0	0	0	0	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	0	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	0	1	0	0
1	0	1	0	1	1	0	1
1	0	1	1	0	1	0	1
1	0	1	1	1	1	1	0
1	1	0	0	0	0	1	1
1	1	0	0	1	1	0	0
1	1	0	1	0	1	0	0
1	1	0	1	1	1	0	1
1	1	1	0	0	1	0	1
1	1	1	0	1	1	1	0
1	1	1	1	0	1	1	0
1	1	1	1	1	1	1	1

num_trainingdata: the number of all training data obtained from the truth table (Table 2.3) of a 2-bit full adder where inputs are a_1, a_0, b_1, b_0 and c_0.

$$F_2 = (\sum_{i \in N} ecv_i) * \alpha_c + (\sum_{i \in N} epv_i) * \alpha_p + (\sum_{j \in Cols} (\min_{k \in Rows} edv_{jk})) * \alpha_d. \quad (2.3)$$

ecv_i: evaluation of complexity value of the gate i.

epv_i**:** evaluation of power value of the gate i.

$edv_{j,k}$**:** evaluation of signal delay value of the gate on column j and row k in an array.

α_c**:** the weight of complexity (set to 1 here).

α_p**:** the weight of power (set to 1 here).

α_d**:** the weight of signal delay (set to 1 here).

N**:** the number of gates in the array.

*Cols***:** the number of columns in the array.

*Rows***:** the number of rows in the array.

$$Fitness = \begin{cases} F_1, & when(F_1 < 100), \\ F_1 + F_2, & when(F_1 = 100). \end{cases} \tag{2.4}$$

The first part F_1 of the fitness function compares the output response of the evolved circuit with the desired ones from a truth table (such as Table 2.3). If all matching, then the fitness value for the correctness is 100. The second fitness F_2 searches for the most optimum solution in terms of complexity, power, and signal delay. This is done by designating gates with different evaluating values about complexity, power, and signal delay (shown in Table 2.2).

2.2.4 Tournament Selection

Tournament selection runs a "tournament" among several individuals (such as two individuals in this research) chosen at randomly from the population and selects the winner which with better fitness value.

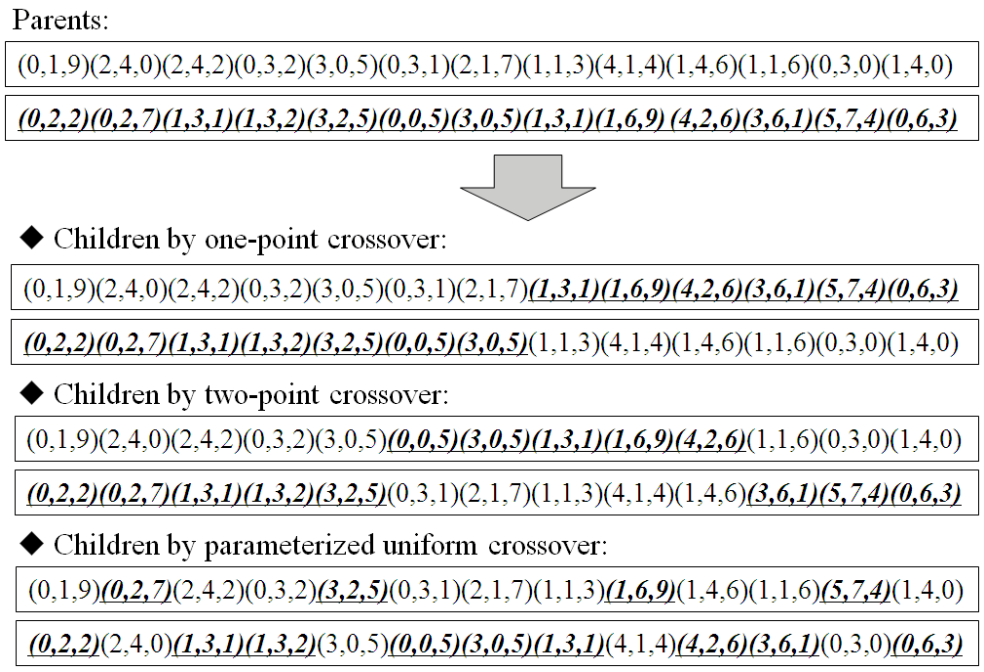


Figure 2.2: An example of crossover.

2.2.5 Crossover

The crossover is operated between two parents, and two new individuals are generated. One-point crossover, two-point crossover, and parameterized uniform crossover are typical crossover methods, an example of these three types of crossover are shown in Fig. 2.2.

2.2.5.1 One-point Crossover

A single crossover point on both parental chromosomes is selected randomly. All data beyond that point in either chromosome are swapped between the two parental chromosomes. The resulting chromosomes are the children.

2.2.5.2 Two-point Crossover

Two-point crossover needs two points to be selected randomly on the parental chromosomes. Everything between the two points is swapped between the parental chromosomes, then generate two children chromosomes.

2.2.5.3 Parameterized Uniform Crossover

The parameterized uniform crossover exchanges some gates with a probability of $Ppuc$ (range: (0,0.5], see section 2.2.5.4). The procedure of the parameterized uniform crossover is given as follows.

1. Select two individuals as parents using tournament selection.

2. Some gates in the parents are selected as the crossover gates with a probability of $Ppuc$. We generate a random number Rn (range: [0,1]) at each gate, which will be selected if $Rn \leq Ppuc$.

3. Two parents exchange the crossover gates with each other.

4. The two new individuals become the individuals of the next generation.

In general, uniform crossover is much more likely to distribute its disruptive trials in an unbiased manner over larger portions of the space. To see that this is true, consider the extreme case in which one parent is a string of all 0s and the other all 1s. Clearly uniform crossover can produce offspring anywhere in the space, while one and two-point crossovers are restricted to rather small subsets. Therefore, uniform crossover has the additional property that it has more exploratory power than that of one or two-point crossover.

The disruption potential is easily controlled via a single parameter $Ppuc$. This suggests the need for only one crossover form (parameterized uniform crossover), which is adapted to different situations by adjusting $Ppuc$.

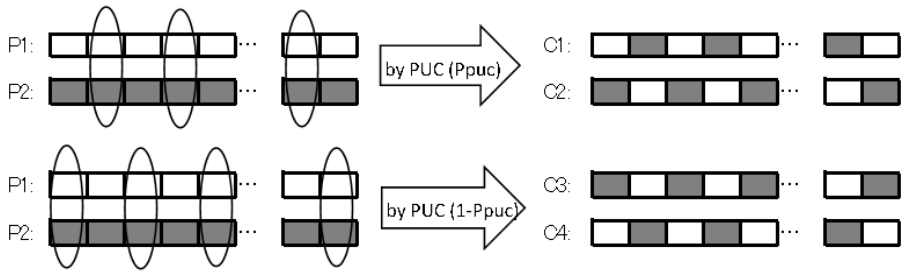


Figure 2.3: An example of Parameterized Uniform Crossover with a probability ($Ppuc$ and $(1 - Ppuc)$).

2.2.5.4 The range of probability in Parameterized Uniform Crossover

For parameterized uniform crossover, the range of the probability (Ppuc) is from 0 to 0.5. Effect of $Ppuc$ equals to that of $(1 - Ppuc)$, for example, the effect of ($Ppuc =$ 0.6 or 0.7) equals to that of ($Ppuc = 0.4$ or 0.3).

As shown in Fig. 2.3, $P1$ and $P2$ are two parental individuals, $C1$ and $C2$ are produced by parameterized uniform crossover (PUC) with a probability ($Ppuc$), $C3$ and $C4$ are produced by parameterized uniform crossover with a probability $(1 - Ppuc)$.

$C1 = P1 * (1 - Ppuc) + P2 * (Ppuc)$,

$C2 = P2 * (1 - Ppuc) + P1 * (Ppuc)$,

$C3 = P1 * (1 - (1 - Ppuc)) + P2 * (1 - Ppuc) = P1 * (Ppuc) + P2 * (1 - Ppuc) = P2 * (1 - Ppuc) + P1 * (Ppuc)$,

$C4 = P2 * (1 - (1 - Ppuc)) + P1 * (1 - Ppuc) = P2 * (Ppuc) + P1 * (1 - Ppuc) = P1 * (1 - Ppuc) + P2 * (Ppuc)$,

So, $C1 = C4$, $C2 = C3$.

2.2.6 Mutation

Mutation is executed on one parent so as to generate a new individual, as shown in Figure 2.4. The procedure of mutation is given as follows.

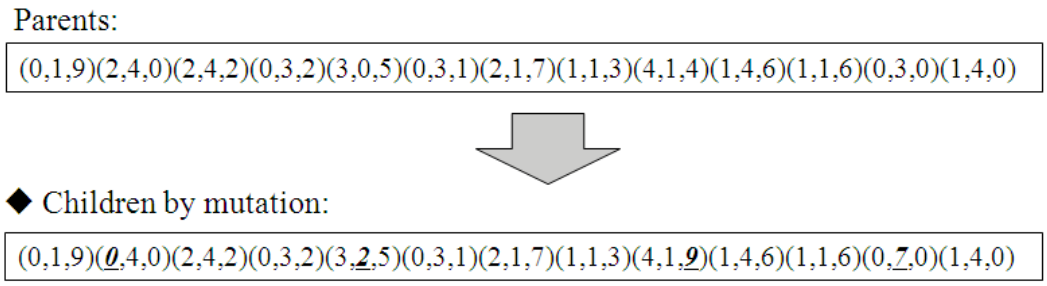


Figure 2.4: An example of mutation.

1. Select one individual as a parent using tournament selection.

2. Some genes are selected with a probability of Pm. We generate a random number Rn (range: [0,1]) at each gene, which will be selected if $Rn \leq Pm$. The selected genes are changed randomly and the new individual is generated.

3. The new individual becomes the individual of the next generation.

2.2.7 Replacement

In the evolution of GA, for next generation, some elite individuals (Elite Size) are preserved, some new individuals (Crossover Size) are produced by crossover method, some new individuals (Mutation Size) are produced by mutation method. The population size is the sum of elite size, crossover size, and mutation size.

2.2.8 Evolutionary Process

GA is a widely used search technique to find optimum or quasi optimum solutions for optimization or search problems. The overview of our GA method is shown in Fig. 2.5.

Figure 2.6 shows a graphical representation of the GA mechanisms. GA involves a search from a population of individuals. In the initialization of a GA population, each

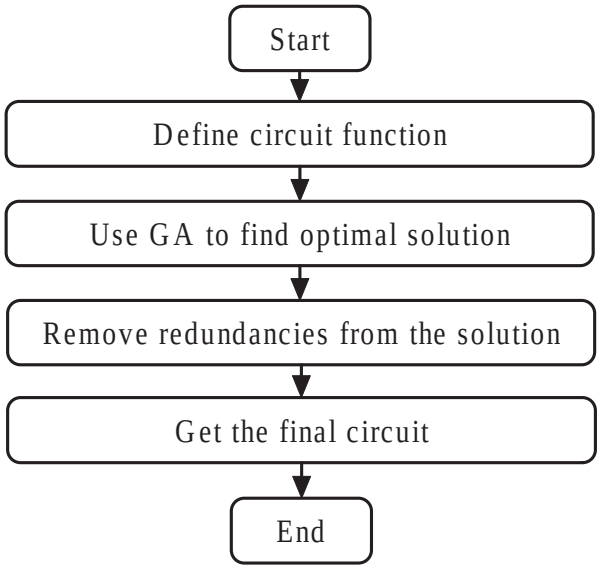


Figure 2.5: The process of our method.

individual is randomly generated. In the evaluation, GA evaluates each candidate according to a fitness function, which indicates how well a candidate satisfies the design specification. In each generation, the elite individuals are preserved and the rest of the individuals are replaced by the new ones generated by crossover and mutation.

GA continues to evolve until it satisfies stop criteria (such as the maximum number of generations). With each generation involving in a competitive selection that rejects and discards poor solutions, based on the survival of the fittest paradigm, the elite fitness of the population is expected to increase in each generation. Therefore, a desired solution can be extracted from the population in the end. This process makes GA well suit combinatorial and continuous problems.

2.3 Experiments and Discussion

This experiment aims to verify circuit design optimization by GApuc. Table 2.4 shows the parameters of the evolution of GApuc. There is no fixed method to define the number of generations, population size, crossover probability and mutation proba-

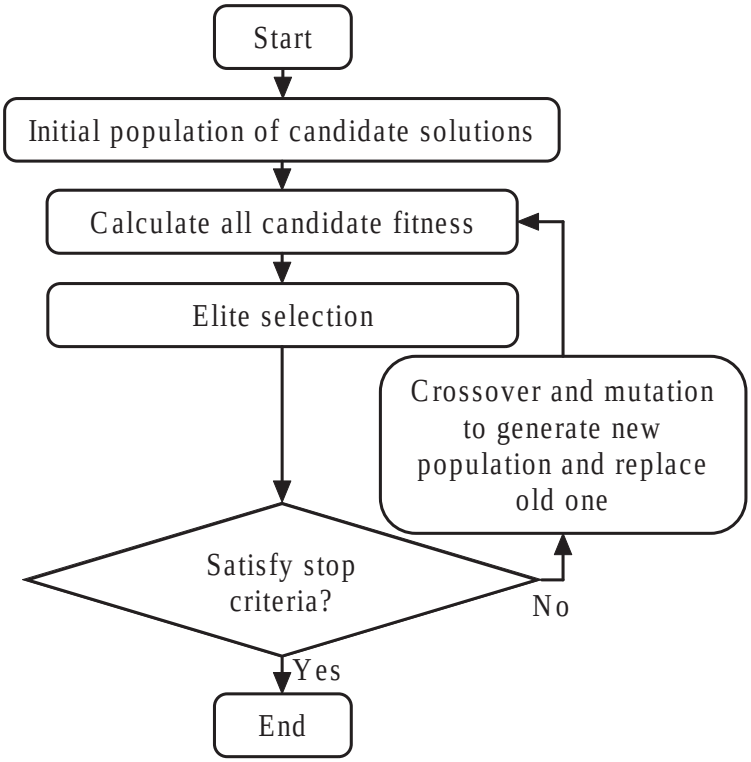


Figure 2.6: The evolutionary process of GA.

bility. Therefor some preliminary experiments were performed in advance to decide parameters suitable for our experiment.

The proposed method has been implemented in Eclipse SDK 3.1.1 with jre 1.6.0; and tested on a PC with Inter(R) Core(TM)2 CPU at 2.67 GHz and 2.0 GB RAM.

2.3.1 Evaluation of GAs

Figure 2.7 shows the elite fitness of GA vs the number of generations during the circuit evolution, where GA with one-point crossover, with two-point crossover, and with parameterized uniform crossover were experimented. From the results, we can see that the elite fitness jumps when F_1 reaches 100, and it is increasing during the evolution period. When *Fitness* is less than 100, GA evolves to get correct circuit;

Table 2.4: Conditions for evolution.

Number of Generation : 600
Population Size : 6010 Elite Size : 10 Crossover Size : 3000 Mutation Size : 3000
Crossover Probability (Pc) : 1 Probability in PUC ($Ppuc$) : 0.3, 0.4, 0.5 Mutation Probability (Pm) : 0.015

PUC: Parameterized Uniform Crossover.

after $Fitness$ reaches 100, GA evolves to get more optimized circuit.

2.3.2 Experimental Results

Table 2.5 shows the results of GA with different crossover. For each GA, we select the successful results over 50 independent trials. In Table 2.5, "Best" means the best elite fitness value; "Quality1" the percent of better in evaluating value of best individual compared to that of GA with one-point crossover; "Quality2" the percent of better in evaluating value of best individual compared to that of GA with two-point crossover; "Average" the average fitness value of the top three individuals; "Var" the sample variance of elite fitness of correct individuals; "Quantity" the number of correct individuals over 50 independent trials; "Time" the average running time of one trial from 50 trials.

From the results, we can see that GApuc produces better solutions than GA with one-point crossover or two-point crossover, from the points of the best elite fitness, the average value of top three fitness and the number of correct circuits. Compared to

Table 2.5: Results of GA with different crossover.

Item	GA(*one*)	GA(*two*)	GApuc(0.3)	GApuc(0.4)	GApuc(0.5)
Best	670	692	716	722	728
Quality1	-	3.86%	8.07%	9.12%	10.18%
Quality2	-	-	4.05%	5.06%	6.08%
Average	633.25	651.75	691.5	669.75	706.25
Var	5678.80	1529.58	2210.53	2194.00	2096.13
Quantity	5	4	9	7	8
Time(m)	5.16	5.22	5.2	5.24	5.26

GA(*one*): GA with one-point crossover.

GA(*two*): GA with two-point crossover.

GApuc(0.3): GApuc with ($Ppuc : 0.3$).

GApuc(0.4): GApuc with ($Ppuc : 0.4$).

GApuc(0.5): GApuc with ($Ppuc : 0.5$).

one-point crossover and two-point crossover, the parameterized uniform crossover is much more likely to distribute its disruptive trials in an unbiased manner over larger portions of the space, thus it can find better solution.

2.3.3 Discussion

In the experiments, the optimized circuits were obtained. Some chromosomes are given as follows:

The chromosome with fitness 670, obtained by GA with one-point crossover:

$$(0,3,2)(2,4,2)(2,4,0)(0,1,9)(1,0,3)$$
$$(3,1,8)(0,1,7)(1,4,1)(2,0,8)(0,3,0)$$
$$(2,7,2)(4,6,7)(3,0,6)(1,4,9)(1,9,1)$$

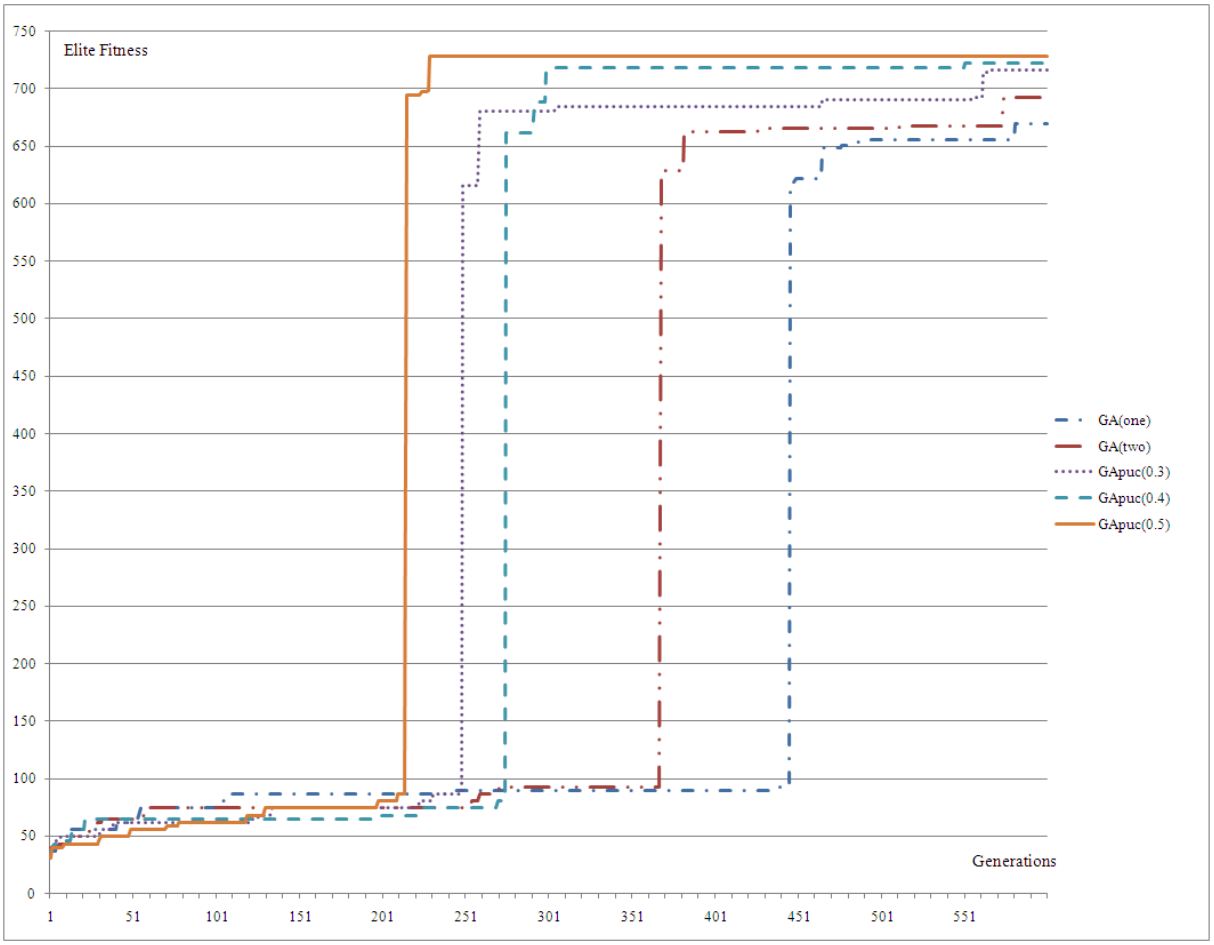


Figure 2.7: Elite fitness of GA with different crossover vs the number of generations.

$$(0,3,6)(9,1,1)(0,0,9)(1,4,8)(9,4,6)$$
$$(0,4,9)(9,13,2)(10,14,9).$$

The chromosome with fitness 692, obtained by GA with two-point crossover:

$$(2,4,1)(2,4,2)(0,1,9)(0,1,0)(0,3,6)$$
$$(2,4,0)(3,1,1)(0,4,6)(3,1,8)(1,0,2)$$
$$(3,5,7)(4,0,3)(1,7,3)(7,8,3)(3,9,6)$$

$$(10,12,0)(0,10,4)(12,7,7)(12,4,8)(3,13,3)$$
$$(2,4,9)(1,10,9)(0,15,9).$$

The chromosome with fitness 716, obtained by GApuc with $(Ppuc : 0.3)$:

$$(0,1,0)(0,3,6)(2,4,2)(2,4,0)(0,1,9)$$
$$(1,4,0)(2,4,6)(1,0,7)(2,0,4)(0,4,7)$$
$$(0,4,3)(6,5,8)(8,7,6)(4,1,4)(0,5,2)$$
$$(2,14,1)(13,8,2)(2,13,5)(0,3,0)(3,13,2)$$
$$(1,4,9)(2,14,9)(3,15,2).$$

The chromosome with fitness 722, obtained by GApuc with $(Ppuc : 0.4)$:

$$(2,4,0)(2,4,2)(0,0,3)(1,3,7)(1,3,2)$$
$$(1,4,5)(4,2,5)(1,2,6)(2,0,6)(2,4,1)$$
$$(5,9,5)(9,4,8)(4,2,4)(0,1,4)(3,9,1)$$
$$(7,14,4)(10,11,0)(11,8,4)(8,3,1)(1,14,1)$$
$$(2,4,9)(1,14,9)(0,19,2).$$

The chromosome with fitness 728, obtained by GApuc with $(Ppuc : 0.5)$:

$$(2,4,9)(2,4,0)(0,1,9)(0,1,0)(3,3,5)$$
$$(1,0,0)(3,0,2)(3,1,2)(3,2,8)(2,4,0)$$
$$(5,3,4)(9,3,0)(8,9,6)(0,8,9)(3,6,9)$$
$$(8,7,2)(12,0,0)(9,10,7)(0,11,0)(12,1,2)$$
$$(2,4,9)(0,11,9)(1,18,0).$$

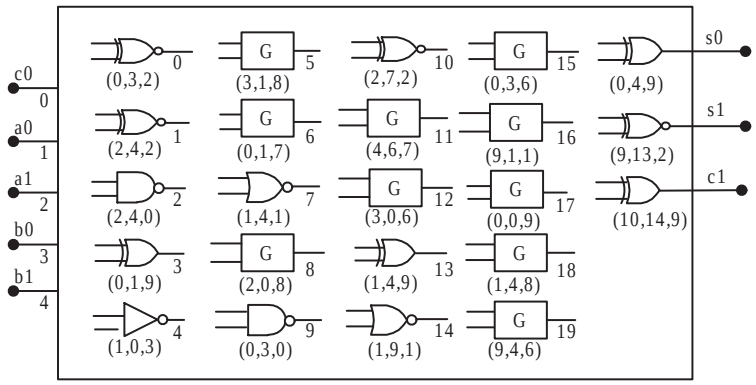


Figure 2.8: The graphical representation of chromosome with fitness 670.

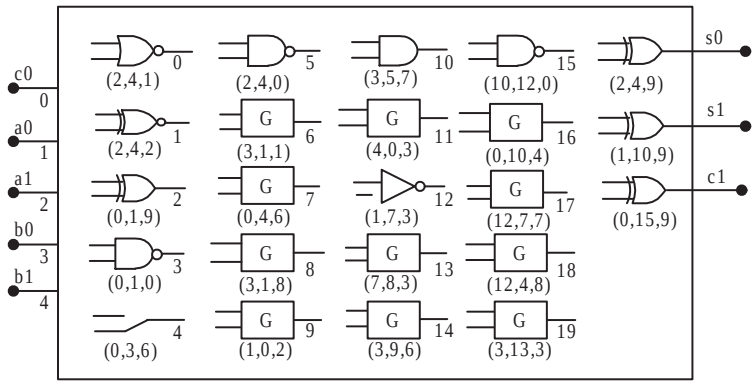


Figure 2.9: The graphical representation of chromosome with fitness 692.

The graphical representation of these chromosomes are shown in Figures 2.8, 2.9, 2.10, 2.11 and 2.12, respectively. In these figures, the gates used in the final circuit are shown by logic gates symbols.

In the Table 2.6, we show the evaluating values about complexity, power, and signal delay of each gate in five experimented circuits. Then, calculate the sum of the evaluating values about complexity and power, and also calculate the min of the evaluating values about signal delay of each column. From this table, we can see how to calculate the fitness function of F_2 and $Fitness$.

Figures 2.13, 2.14, 2.15, 2.16 and 2.17 show the optimized circuits with fitness

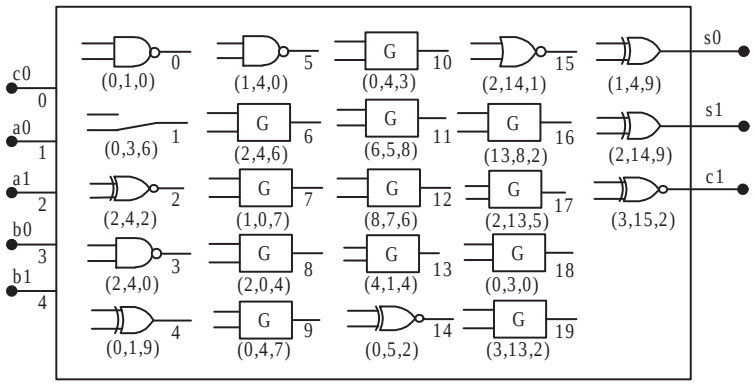


Figure 2.10: The graphical representation of chromosome with fitness 716.

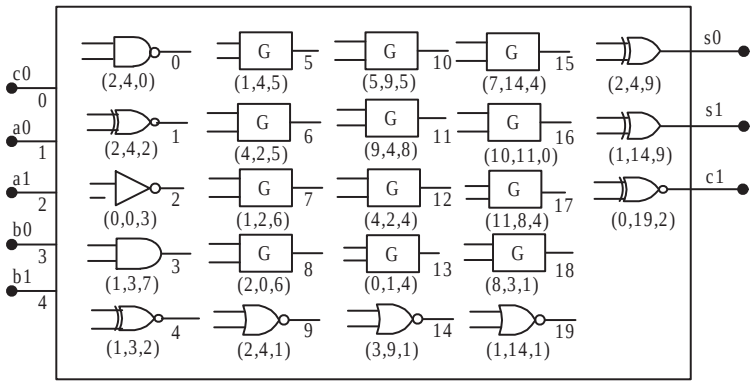


Figure 2.11: The graphical representation of chromosome with fitness 722.

670, 692, 716, 722 and 728, respectively, after removing unnecessary gates.

The circuits in Figures 2.15, 2.16 and 2.17 are superior to the circuits in Figures 2.13 and 2.14 in quality, because the former has larger fitness and is composed of less gates, this leads a circuit with less complexity, less power, and less signal delay.

The circuit in Fig. 2.16 has one more gates than the circuit in Fig. 2.15. But the gates in Fig. 2.16 has less complexity, such as 'NOT' in col_1 and 'NOR' in col_3. From Table 2.6, we can see the value of F_2 of the circuit in Fig. 2.16 is lager than that of the circuit in Fig. 2.15. So the circuit in Fig. 2.16 has less complexity, less power, and less signal delay than the circuit in Fig. 2.15.

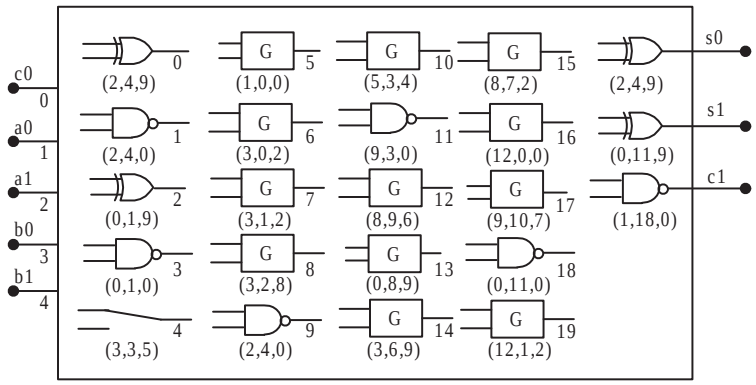


Figure 2.12: The graphical representation of chromosome with fitness 728.

2.4 Conclusion

This chapter applied GA to circuit design optimization. First, we introduce the evaluating value about correctness, complexity, power, and signal delay to the fitness function. Then GA can autonomously synthesize a circuit that is equivalent to a conventional design in function, but is simpler and has better performance. By evolution, GApuc can find optimized circuits with less complexity, less power, and less signal delay than GA with one-point crossover or two-point crossover.

However, the number of correct individuals is low, and the running time is long, which points out the future research direction. Also, there are other crossover methods, such as arithmetic crossover [36] and heuristic crossover [37]. We will compare parameterized uniform crossover with other crossover methods in the problem of circuit design optimization, to search a better crossover method.

We will also apply GA to autonomous design circuits for more complex functional requirements, and enhance more practical information about circuit to fitness function. In the future, we will develop the adaptive systems which reconfigure an existing design by GA to adapt to a variable operational environment.

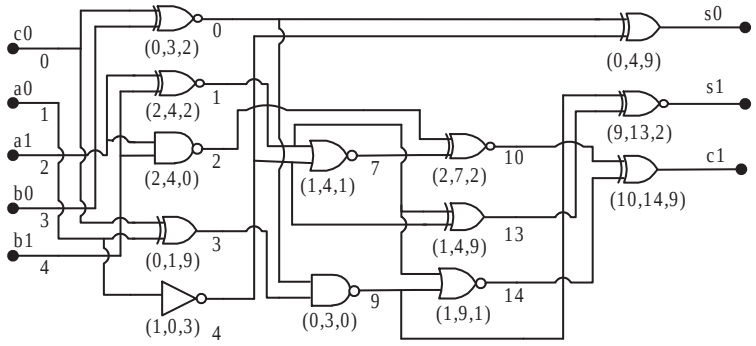


Figure 2.13: The optimized circuit with fitness 670 after removing unnecessary gates.

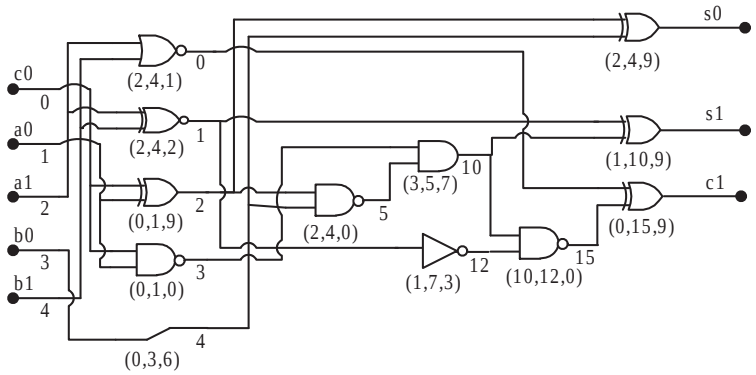


Figure 2.14: The optimized circuit with fitness 692 after removing unnecessary gates.

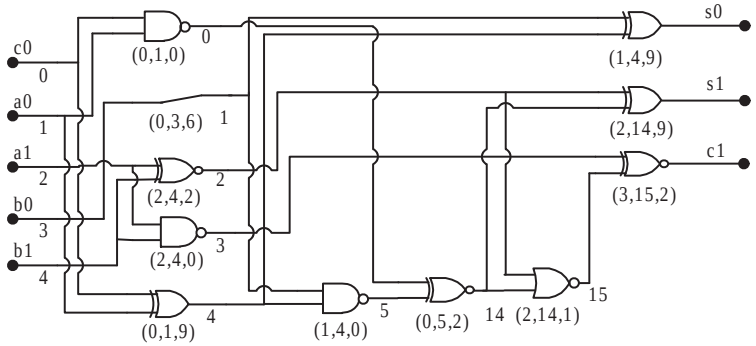


Figure 2.15: The optimized circuit with fitness 716 after removing unnecessary gates.

Table 2.6: The account of five elite fitness.

item	col_1			col_2			col_3			col_4			col_5			F_2/F
values	EC	EP	ESD	EC	EP	ESD	EC	EP	ESD	EC	EP	ESD	EC	EP	ESD	-
row0	2	6	4	20	20	20	2	6	4	20	20	20	2	6	4	-
row1	2	6	4	20	20	20	20	20	20	20	20	20	2	6	4	-
row2	6	7	6	6	7	6	20	20	20	20	20	20	2	6	4	-
row3	2	6	4	20	20	20	2	6	4	20	20	20	-	-	-	-
row4	8	8	7	6	7	6	6	7	6	20	20	20	-	-	-	-
sum/min	20	33	4	72	74	6	50	59	4	100	100	20	6	18	4	570/670
row0	6	7	6	6	7	6	4	5	3	6	7	6	2	6	4	-
row1	2	6	4	20	20	20	20	20	20	20	20	20	2	6	4	-
row2	2	6	4	20	20	20	8	8	7	20	20	20	2	6	4	-
row3	6	7	6	20	20	20	20	20	20	20	20	20	-	-	-	-
row4	10	4	2	20	20	20	20	20	20	20	20	20	-	-	-	-
sum/min	26	30	2	86	87	6	72	73	3	86	87	6	6	18	4	592/692
row0	6	7	6	6	7	6	20	20	20	6	7	6	2	6	4	-
row1	10	4	2	20	20	20	20	20	20	20	20	20	2	6	4	-
row2	2	6	4	20	20	20	20	20	20	20	20	20	2	6	4	-
row3	6	7	6	20	20	20	20	20	20	20	20	20	-	-	-	-
row4	2	6	4	20	20	20	2	6	4	20	20	20	-	-	-	-
sum/min	26	30	2	86	87	6	82	86	4	86	87	6	6	18	4	616/716
row0	6	7	6	20	20	20	20	20	20	20	20	20	2	6	4	-
row1	2	6	4	20	20	20	20	20	20	20	20	20	2	6	4	-
row2	8	8	7	20	20	20	20	20	20	20	20	20	2	6	4	-
row3	4	5	3	20	20	20	20	20	20	20	20	20	-	-	-	-
row4	2	6	4	6	7	6	6	7	6	6	7	6	-	-	-	-
sum/min	22	32	3	86	87	6	86	87	6	86	87	6	6	18	4	622/722
row0	2	6	4	20	20	20	20	20	20	20	20	20	2	6	4	-
row1	6	7	6	20	20	20	6	7	6	20	20	20	2	6	4	-
row2	2	6	4	20	20	20	20	20	20	20	20	20	6	7	6	-
row3	6	7	6	20	20	20	20	20	20	6	7	6	-	-	-	-
row4	10	4	2	6	7	6	20	20	20	20	20	20	-	-	-	-
sum/min	26	30	2	86	87	6	86	87	6	86	87	6	10	19	4	628/728

F: Fitness.

EC: evaluation of complexity.

EP: evaluation of power.

ESD: evaluation of signal delay.

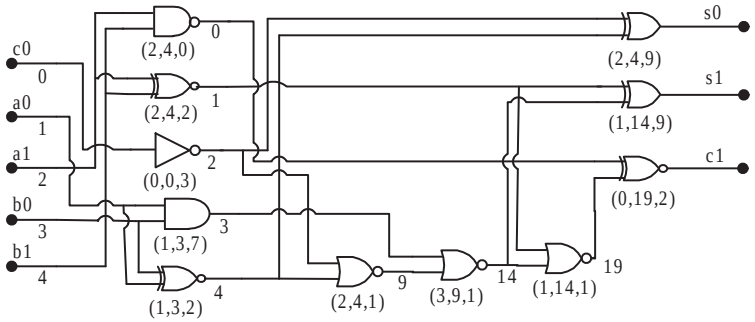


Figure 2.16: The optimized circuit with fitness 722 after removing unnecessary gates.

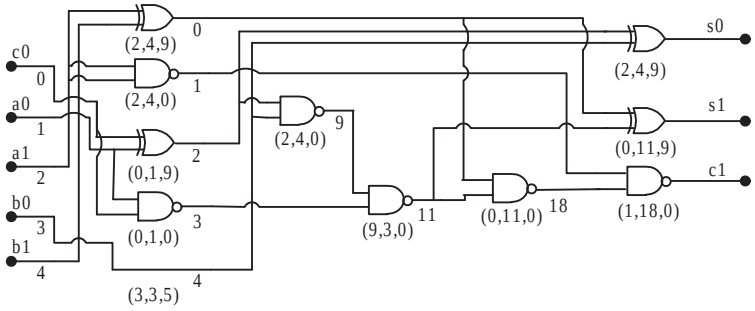


Figure 2.17: The optimized circuit with fitness 728 after removing unnecessary gates.

Chapter 3

GATE LEVEL CIRCUIT DESIGN OPTIMIZATION USING GENETIC ALGORITHM WITH DIFFERENT STRUCTURE SELECTION

In traditional GA, the tournament selection for crossover and mutation is based on fitness of individuals. It can make convergence easily, but maybe lose some useful genes. In selection, besides fitness, we consider the different structure from individuals comparing to the elite one. First, some individuals are selected using more different structures, then crossover and mutation are performed for these ones to generate new individuals. By this way, GA can increase diversification to searching spaces, so that it can find better solution. One of the promising application of GA is EHW, which is a new research field to synthesize an optimal circuit. We propose optimal circuit design by using GA with different structure selection (GAdss) and with fitness function composed of circuit complexity, power and signal delay. Its effectiveness is verified by simulations. From the results, we can see that the best elite fitness, the average value of fitness of correct circuits and the number of correct circuits of GAdss are better than traditional GA. The best case of optimal circuits generated by GAdss is 8.1% better in evaluating value than that by traditional GA.

3.1 Introduction

In this chapter, we propose a new approach for circuit design optimization by GA with Different Structure Selection (GAdss), where mixed constraints on circuit complexity, power and signal delay are considered. We introduce the evaluating value about correctness, complexity, power and signal delay for the fitness function in order to

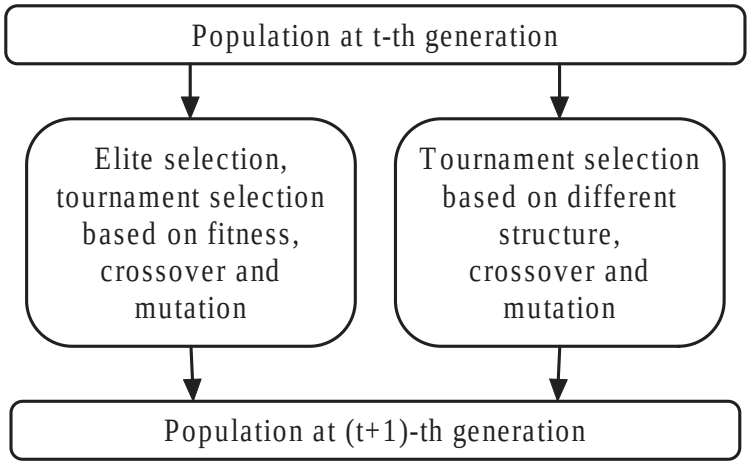


Figure 3.1: Reproduction of GA, left: traditional GA, right: GAdss.

meed the mixed constrains. In the first step, the fitness function is used to find the solutions with 100% correctness of the target circuit, and with maximal evaluating values about complexity, power and signal delay. Then, GAdss can autonomously synthesize a circuit that is equivalent to a conventional design in functionality, but is simpler and has better performance. As a result, GAdss can find a better circuit, compared to traditional GA. To verify an effectiveness of our approach, a simple 2-bit half adder circuit is experimentally synthesized.

In the next section, a brief overview of GAdss is described. Section 3.3 describes the use of GAdss as a new approach for the automatic design of an optimized circuit. Section 3.4 shows experiments on a 2-bit half adder circuit design as an example. Finally, the chapter concludes with a summary of the results in section 3.5.

3.2 Genetic Algorithm with Different Structure Selection

In traditional GA, elite selection and tournament selection are based on the fitness of individuals. This is good for GA to find the local best solution, but it may be premature convergence. To make up for the weakness, we consider the different

structure of individuals compared to the elite one. The value of different structure is defined by the sum of difference of genes between two individuals. We select some individuals with different structure to do crossover and mutation, to generate some new individuals to the next populations. This method can extend diversification to search spaces, so can find a better solution.

Figure 3.1 shows a graphical representation of the reproduction of GAdss. In each generation, the elite individuals are preserved. In selecting some individuals, half of them are processed by tournament selection based on fitness, and another half are processed by tournament selection based on different structure. Then crossover and mutation are performed to create new ones for next generation.

3.2.1 Tournament selection

Here, tournament selection runs a "tournament" among two individuals chosen at random from the population, and selects the winner which with the better fitness or has more different structure.

3.2.2 Crossover

The crossover is operated between two parents, and two new individuals are generated. This procedure is shown as follows.

1. Select two individuals as parents using tournament selection.

2. Some bits in the parents are selected as the crossover bits with the given probability of Pc.

3. Two parents exchange the corresponding selected bits with each other.

4. The two new individuals become the individuals of the next generation.

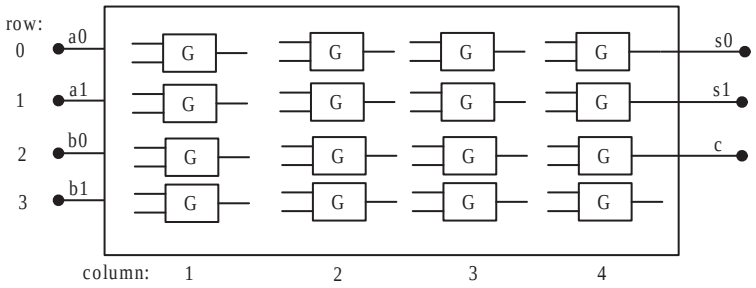


Figure 3.2: An initial 4*4 array with the input/output function for a 2-bit half adder.

3.2.3 Mutation

Mutation is executed for one parent, and a new individual is generated. This procedure is shown as follows.

1. Select one individual as a parent using tournament selection.

2. Some bits are selected with the given probability of Pm. The selected bits are changed randomly and the new individual is generated.

3. The new individual becomes the individual of the next generation.

3.3 Gate Level Circuit Design Optimization using GAdss

3.3.1 Objective

The overall objective is to discover novel solutions by the application of GA in the circuit design process. The target circuit has to provide identical functional behavior equivalent to the specification but require less complexity, less power and less signal delay.

In this section, we will demonstrate the principle of GAdss in the design process using a 2-bit half adder as a sample logic circuit.

3.3.2 Genetic Encoding

To process a genetic encoding easily, the logic circuit under consideration is assumed to be organized on a two dimensional array of cells [19]. Each cell accepts two inputs and produces one output. The cells in the first column (the left most in Fig. 3.2) of the array are set with predefined inputs. For the purpose of this experiment, the combinatorial circuit takes four primary inputs. Therefore there are 16 input patterns of the circuit. Cells in the following columns receive outputs from cells in the previous columns. The chromosome is a string of integers where each three continuous genes embody a cell. Each triplet in the chromosome encodes the two inputs for the first two genes and the type of a cell for the third gene, respectively. In this experiment, the last cell in the bottom right corner in Fig. 3.2 is not used, so the chromosome length is calculated by the following formula:

$$3 * ((numberofcolumns) * (numberofrows) - 1). \tag{3.1}$$

In the experiment, the array is a fixed size of 4*4 cells (shown in Fig. 3.2), thus the length of the chromosome is 45 $(= 3 * (4 * 4 - 1))$. The inputs of each cell in the first column of the array can take the value of any integer in the range of [0 to $(max\_number\_inputs - 1)$]. Cells in all other columns can take any integer value in the range of [0 to $((now\_column - 1) * (numberofrows) - 1)$]. As for the third gene in the triplet, cell type is defined as shown in Table 2.2.

A typical chromosome then can be a sequence of triplets such as:

$$([0, X], [0, X], [0, 9])...([0, X], [0, X], [0, 9]),$$

where, $([0, X], [0, X], [0, 9])$ is a triplet in a cell.

Here, X is 3 in the first four cells (in the first column); X is $((now\_column-1)*4-1)$ in the other columns, *now_column* is the number of (2,3,4) column where the cell is

Table 3.1: Conditions for evolution.

Number of Generation : 500
Population Size : 1210
Elite Size : 10
Crossover Size : 600
Mutation Size : 600
Crossover Probability (Pc) : 0.2, 0.5
Mutation Probability (Pm) : 0.023

placed. $[0, 9]$ is the gate type, defined as shown in Table 2.2.

3.4 Experiments and Results

This experiment aims to verify circuit optimization by GAdss. Table 3.1 shows the parameters of the evolution of GAdss. There is no fixed method to define the number of generations, population size, crossover probability and mutation probability. Therefore some preliminary experiments were performed in advance to decide parameters suitable for our experiment.

The proposed method has been implemented in Eclipse SDK 3.1.1 with jre 1.6.0; and tested on a PC with Inter(R) Core(TM)2 CPU at 2.67 GHz and 2.0 GB RAM.

Table 3.2 shows the results of experiments using GAs. For each GA, we select the successful results over 60 independent trials. In Table 3.2, "Best" means the best elite fitness value; "Quality" the percent of better in evaluating value of best individual compared to that of GA; "Average" the average fitness value of top three individuals; "Quality" the number of correct individuals over 60 independent trials; "Time" the running time of 60 trials.

From the results, we can see that the best elite fitness, the average fitness value

Table 3.2: Results of different GA.

Item	GA(0.2)	GAdss(0.2)	GA(0.5)	GAdss(0.5)
Best	447	465	471	501
Quality	-	5.2%	-	8.1%
Average	428.4	438	464.4	500.4
Quantity	3	7	14	16
Time(m)	19	19	25	25

GA(0.2): GA with ($Pc : 0.2$)
GAdss(0.2): GAdss with ($Pc : 0.2$)
GA(0.5): GA with ($Pc : 0.5$)
GAdss(0.5): GAdss with ($Pc : 0.5$)

of top three correct circuits, and the number of correct circuits of GAdss are better than GA. Compared to traditional GA, GAdss considers the different structure from individuals comparing to elite one, then it can enhance diversification to searching spaces, so that it can find better solution.

In the experiments, the optimized circuit with fitness 501 was obtained by the GAdss (Pc: 0.5). This chromosome is as follows:

$(0,2,2)(0,2,7)(1,3,1)(1,3,2)(3,2,5)(0,0,5)(3,0,5)(1,3,1)$
$(1,6,9)(4,2,6)(3,6,1)(5,7,4)(0,6,3)(1,3,2)(2,7,1).$

The graphical representation of this chromosome is shown in Figure 3.3. In this figure, we show the useful gates with logic gates symbols. Figures 3.4 and 3.5 show the optimized circuits with fitness 501 and 471, respectively, after removing unnecessary gates. The circuit in Figure 3.4 is obviously better than the one in Figure 3.5, because the former is composed of less gates, so that the lager fitness can produce a circuit

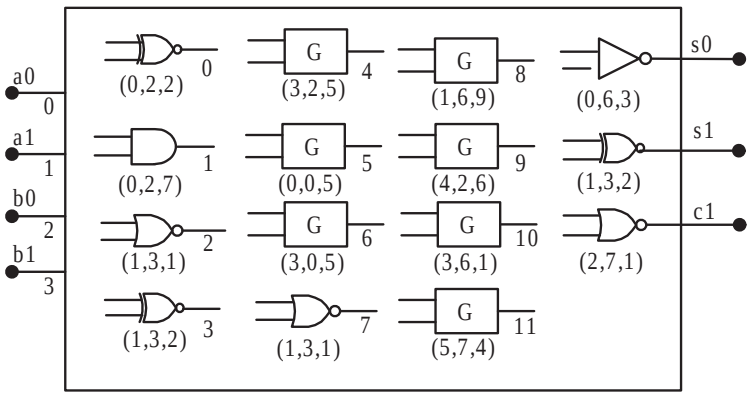


Figure 3.3: The graphical representation of chromosome (501).

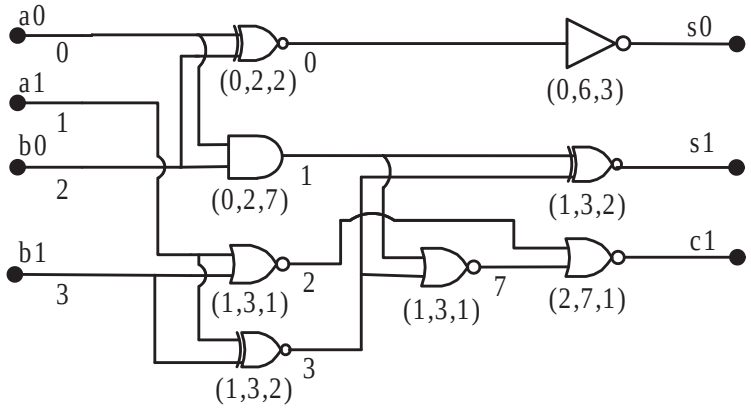


Figure 3.4: The optimized circuit after removing unnecessary gates (501).

with less complexity, less power and less signal delay.

3.5 Conclusion

This chapter proposed GAdss and its application to autonomous design optimization for combinatorial circuits. By evolution, GAdss can find optimized circuits with less complexity, less power and less signal delay than traditional GA, because different structure selection has effectiveness.

We can also apply GAdss to autonomous design circuits for more complex func-

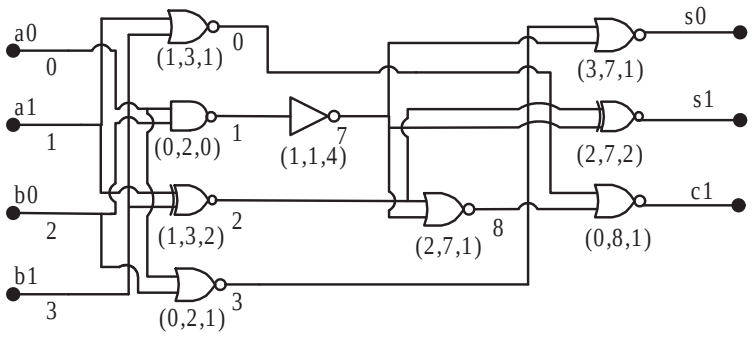


Figure 3.5: The optimized circuit after removing unnecessary gates (471).

tional requirements, and enhance more exact information about circuit to fitness function.

Chapter 4

MIXED CONSTRAINED IMAGE FILTER DESIGN FOR NOISE REDUCTION USING GENETIC ALGORITHM

This chapter describes mixed constrained image filter design for noise reduction using a Genetic Algorithm (GA) with parameterized uniform crossover. The proposed method with GA autonomously synthesizes a filter suitable for the reconfigurable processing array, evaluating the complexity, power and signal delay in both Configurable Logic Blocks (CLBs) and wires. An image filter for noise reduction is experimentally synthesized to verify the validity of the proposed method. By the evolution, the quality of the optimized image filter is better than that reported in other papers.

4.1 Introduction

Evolutionary hardware design is well suited for adaptive image processing systems, because some intelligent preprocessing is usually required in these systems as the input data streams come from complex practical situations via non-ideal cameras. In these applications, image/video preprocessing (filtering), segmentation, recognition and compression are included.

The image filter design problem is often approached by means of evolutionary design techniques. In addition to an optimization of filter coefficients [38], evolutionary approaches are applied to find a complete structure of image filters. In [39], gaussian noise filters were evolved using a variant of Cartesian Genetic Programming in which target filters were composed of simple digital components, such as logic gates, adders and comparators. Later, image filters for other types of noise and edge detectors were evolved using the same technique [40–43].

Table 4.1: Features of related works.

Item	Level	Circuit	Optimization
Vasicek [42]	function	image filter	no
Bao [44]	gate	adder	yes
Proposed	function	image filter	yes

Vasicek et al. [42, 43] discussed image filter design at function level, but they did not discuss any circuit constraints such as complexity, power consumption and signal delay, as shown in Table 4.1. While we have proposed mixed constrained design optimization using GA for some combinational circuits (such as an adder), the discussion was at gate level [44–46]. This chapter applies the optimization method to image filter design at function level, in order to deal with more complex functions and larger sized circuits. The function of an image filer is more complex than that of a 2-bit full adder circuit in the paper [44], the circuit size of an image filter is also lager than that of the adder circuit.

This chapter describes mixed constrained image filter design for noise reduction using a Genetic Algorithm (GA) where parameterized uniform crossover is adopted on a reconfigurable processing array. The circuit complexity, power and signal delay in both logic blocks and wires are optimized. In this design, first, the evaluating values about correctness, complexity, power and signal delay are introduced to the fitness function. Then GA autonomously synthesizes an image filter which is simple and has better performance and fits to the reconfigurable processing array. To verify the validity of the proposed method, an image filter for noise reduction is experimentally synthesized.

The organization of this chapter is as follows: Section 4.2 describes an image filter design for noise reduction using GA. Section 4.3 shows the experimental results.

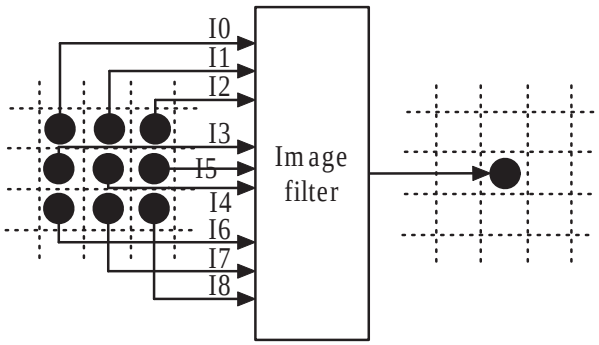


Figure 4.1: An example image filter.

Finally, Sect. 4.4 concludes this chapter.

4.2 Image Filter Design for Noise Reduction using GA

GA is applied to search good solutions to optimize the image filter design for noise reduction on a reconfigurable processing array.

The resultant image filter has an identical functional behavior with less complexity, less power and less signal delay.

4.2.1 Image Filter

Every image operator is considered as a digital circuit with nine 8-bit inputs and a single 8-bit output, which processes gray-scaled (8-bit/pixel) images.

As shown in Fig. 4.1, every pixel value of the filtered image is calculated using a corresponding pixel and its eight neighbors in the processed image [41, 42].

4.2.2 Reconfigurable Processing Array for Image Filter

The reconfigurable image filter is implemented as a Virtual Reconfigurable Circuit (VRC) (Fig. 4.2) [41]. As a new pixel value is calculated using nine pixels, the VRC has got nine 8-bit inputs and a single 8-bit output. The VRC consists of two-input

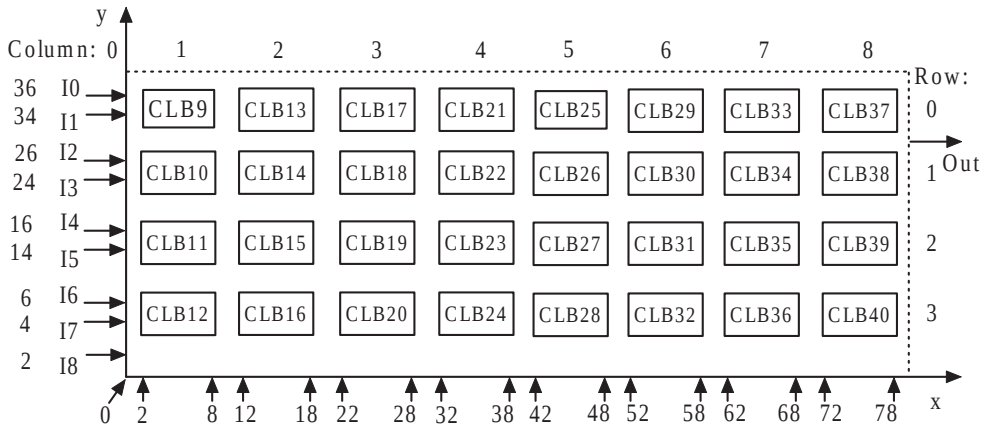


Figure 4.2: A reconfigurable processing array.

Configurable Logic Blocks (CLBs) placed in a $8*4$ array. Any input of each CLB may be connected to either a primary circuit input or the output of a CLB in the preceding columns. Any CLB can be programmed to implement one of the functions given in Table 4.2 [47, 48]. All these functions operate with 8-bit operands and produce 8-bit results.

The CLB with different function has different complexity, power and signal delay. The values of complexity (FC), power (FP) and signal delay (SD) are newly defined for each function in a CLB, as shown in Table 4.2. The gate level circuit with different function has different number of gates. The values of FC, FP and SD in each function are set considering the number of gates, the power of gates and the critical path of the circuit, respectively.

Figure 4.2 shows coordinates of inputs and output of each logic block, therefor the length of connection wires can be calculated.

Table 4.2: Functions implements in a CLB.

ID	Function	Description	FC	FP	SD
0	255	Constant	8	5	1
1	x	Identity	16	10	2
2	$255 - x$	Inversion	24	15	3
3	$x \vee y$	Bitwise OR	32	20	3
4	$\overline{x} \vee y$	Bitwise $\overline{x}$ OR y	40	25	4
5	$x \wedge y$	Bitwise AND	32	20	3
6	$not(x \wedge y)$	Bitwise NAND	40	25	4
7	$x \oplus y$	Bitwise XOR	64	38	4
8	$x \gg 1$	Right shift by 1	15	9	2
9	$x \gg 2$	Right shift by 2	14	8	2
10	$(x \ll 4) \vee (y \gg 4)$	Swap	16	10	2
11	$x + y$	+ (addition)	358	215	18
12	$x +^s y$	+ with saturation	367	220	19
13	$(x + y) \gg 1$	Average	350	210	18
14	$max(x, y)$	Maximum	240	145	16
15	$min(x, y)$	Minimum	240	145	16
-	(wire)	(wire)	16	10	2

FC: function complexity.

FP: function power.

SD: signal delay.

4.2.3 Genetic Encoding

The chromosome is a string of integers where each three continuous integers constitute a logic block. Each triplet in the chromosome encodes the two inputs and the function type of a logic block, respectively, such as [45, 46]:

$$(Input\_1, Input\_2, Function\_type).$$

A typical chromosome then can be a sequence of triplets such as:

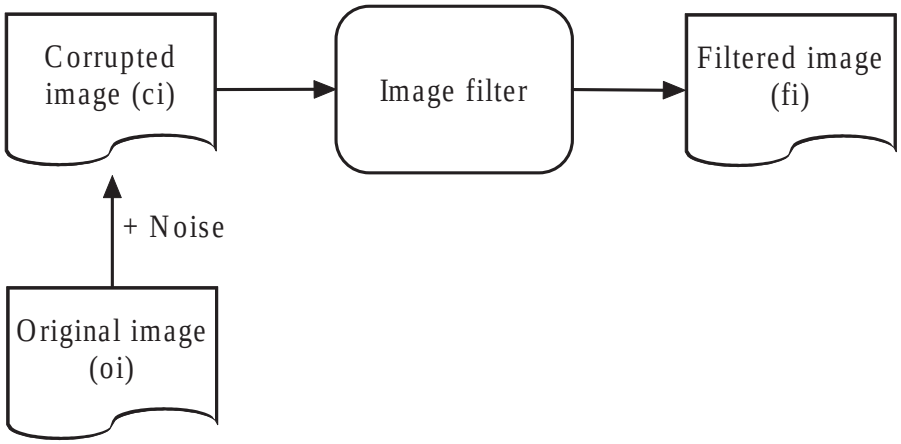


Figure 4.3: The input and output of a image filter for noise reduction.

$$(IN_1^1, IN_2^1, F_{type}^1) \cdots (IN_1^i, IN_2^i, F_{type}^i) \cdots$$

Here, IN_1^i and IN_2^i mean the positions of the corresponding input signals. F_{type}^i means the function type of a logic block. For a primary input, $0 \leq IN^i \leq 8$. For the input from the output of a logic block $CLBm$ shown in Fig. 4.2, $IN^i = m$. The function in a CLB is defined as shown in Table 4.2, and F_{type}^i has the same value as the ID in the Table 4.2.

4.2.4 Fitness Function

The pixels of corrupted image *cv* are used as the inputs of VRC. The pixels of filtered image fi are generated and compared to the pixels of original image *oi*, as shown in Figure 4.3.

The design objective is to minimize the difference between the filtered image and the original image. The image size is $M*N$ pixels, but only the area of $(M-2)*(N-2)$ pixels is considered, because the pixel values at the borders are ignored, and remain unfiltered. The fitness value of a candidate filter is obtained as follows:

(1) VRC is configured using a candidate chromosome,

(2) the created circuit is used to produce pixel values in the image fi, and

(3) the fitness value is calculated as

$$Fitness = (-1) * (F_1 * \beta + F_2). \tag{4.1}$$

where,

F_1 and F_2 mean the correctness and the quality of the circuit respectively, and are defined as follows. β is the weight on F_1.

$$F_1 = (\sum_{i=1}^{M-2} \sum_{j=1}^{N-2} |fi(i,j) - oi(i,j)|). \tag{4.2}$$

where,

M**:** the number of columns of the pixels in an image.

N**:** the number of rows of the pixels in an image.

$fi(i,j)$**:** the pixel (i, j) in filtered image fi, the value range is [0,255].

$oi(i,j)$**:** the pixel (i, j) in original image oi, the value range is [0,255].

$$\begin{aligned} F_2 = \quad & SD * \alpha_{sd} + Pb * \alpha_{pb} + Cb * \alpha_{cb} + \\ & Pw * \alpha_{pw} + Cw * \alpha_{cw}. \end{aligned} \tag{4.3}$$

where,

SD**:** signal delay of a circuit individual, determined by a critical path.

Table 4.3: An example of weight value setting.

Item	An example of value	Item with weight	Weight value
F1	61377	F1 * 10^8	10^8
SD	133	SD * 10^8	10^8
Pb	1935	Pb * 10^5	10^5
Cb	3210	Cb * 10^3	10^3
Pw	515	Pw * 10^2	10^2
Cw	816	Cw * 10^0	10^0

α_{sd}**:** the weight on signal delay in F_2.

Pb**:** power of logic blocks in a circuit, calculated by summation of all logic block's power.

α_{pb}**:** the weight on power of logic blocks in F_2.

Cb**:** complexity of logic blocks in a circuit, calculated by summation of all logic block's complexity.

α_{cb}**:** the weight on complexity of logic blocks in F_2.

Pw**:** power of all wires in a circuit, calculated by summation of all wire's power.

α_{pw}**:** the weight on power of wires in F_2.

Cw**:** complexity of wires in a circuit, calculated by summation of all wire's complexity.

α_{cw}**:** the weight on complexity of wires in F_2.

The priority of evaluating values in Eq. (4.1) is: $F_1 > F_2$. In the experiment, β is set to $0.1 * 10^9$. The priority of evaluating values in Eq. (4.3) is set as: $SD > Pb > Cb > Pw > Cw$ in the experiment, in order to get the circuit which has less signal delay (SD), less power of CLBs (Pb), less complexity of circuit (Cb), less power of

wires (Pw), and less length of wires (Cw). So, α_{sd} is set to $1*10^9$, α_{pb} is set to $1*10^5$, α_{cb} is set to $1*10^3$, α_{pw} is set to 100, and α_{cw} is set to 1. All α's and β are empirically assigned in the experiment by the priority and the range of values, as shown in Table 4.3. These factors (SD, Pb, Cb, Pw, Cw) have the possible relation one another, but they need to be considered independently in order to see the value of each factor.

In the signal processing system, the quick response is usually the most important, so the priority of SD is set as first. In recent years, low power is requested, so the priority of Pb is set as second. In the end, the priority of Cb is set as third, to evaluate the complexity of circuit. To fit the request of an applied system, the priority of evaluating values could be changed. Compared with wires, the effect of power and complexity of CLBs are lager, so the priority of Pw and Cw are set as fourth and fifth, respectively.

F_1 is the difference between pixels in filtered image and original image; the less the value is, the better the circuit is. F_2 searches for the optimum solution considering complexity, power and signal delay simultaneously in both logic blocks and wires; the less the value is, the better the quality of circuit is. In the evolution, the larger the fitness is, the better the quality of image filter is.

4.3 Experimental Results

Table 4.4 shows the parameters of the evolution of GA used in the experiment. These parameters suitable for the experiment are decided based on the results of papers [42, 44], and some preliminary experiments were performed in advance. Crossover could exchange some logic blocks between two individuals by $Ppuc$, and here is 10%, 20%, 30%, 40% and 50% of all logic blocks. Mutation could change some genes in one individual by Pm, and here is 10% of all genes, input number or function type of a logic block in an image filter. The image filter design is a so difficult problem, that it costs evolution times. How to set a less population size and a less number of generation to reduce CPU time, will be reported in another paper.

Table 4.4: Conditions for evolution.

Number of Generation : 3000
Population Size : 1210
Elite Size : 10
Crossover Size : 600
Mutation Size : 600
Probability in PUC ($Ppuc$) : 0.1, 0.2, 0.3, 0.4, 0.5
Mutation Probability (Pm) : 0.1

PUC: Parameterized Uniform Crossover.

The proposed method was implemented in Eclipse SDK 3.5.1 with Java Runtime Environment(JRE) 1.6.0; and tested on a PC with Inter(R) Core(TM) i7 CPU at 3.33 GHz and 9.0 GB RAM.

There are two popular types of noise, impulse noise and gaussian noise. Salt-and-pepper noise is a kind of impulse noise. Salt-and-pepper noise is a form of noise typically seen on images. It represents itself as randomly occurring white and black pixels. Gaussian noise is statistical noise that has a probability density function of the normal distribution (also known as gaussian distribution). In other words, the values of the noise are gaussian-distributed. The experiments are done on salt-and-pepper noise and gaussian noise.

4.3.1 Experiments on salt-and-pepper noise

The image filter is evolved for a 512∗512 Lena image corrupted by 5% salt-and-pepper noise, shown in Fig. 4.6 (a).

Figure 4.4 shows the elite fitness of GA with parameterized uniform crossover (GApuc) with ($Ppuc$: 0.5) vs. the number of generations during the image filter

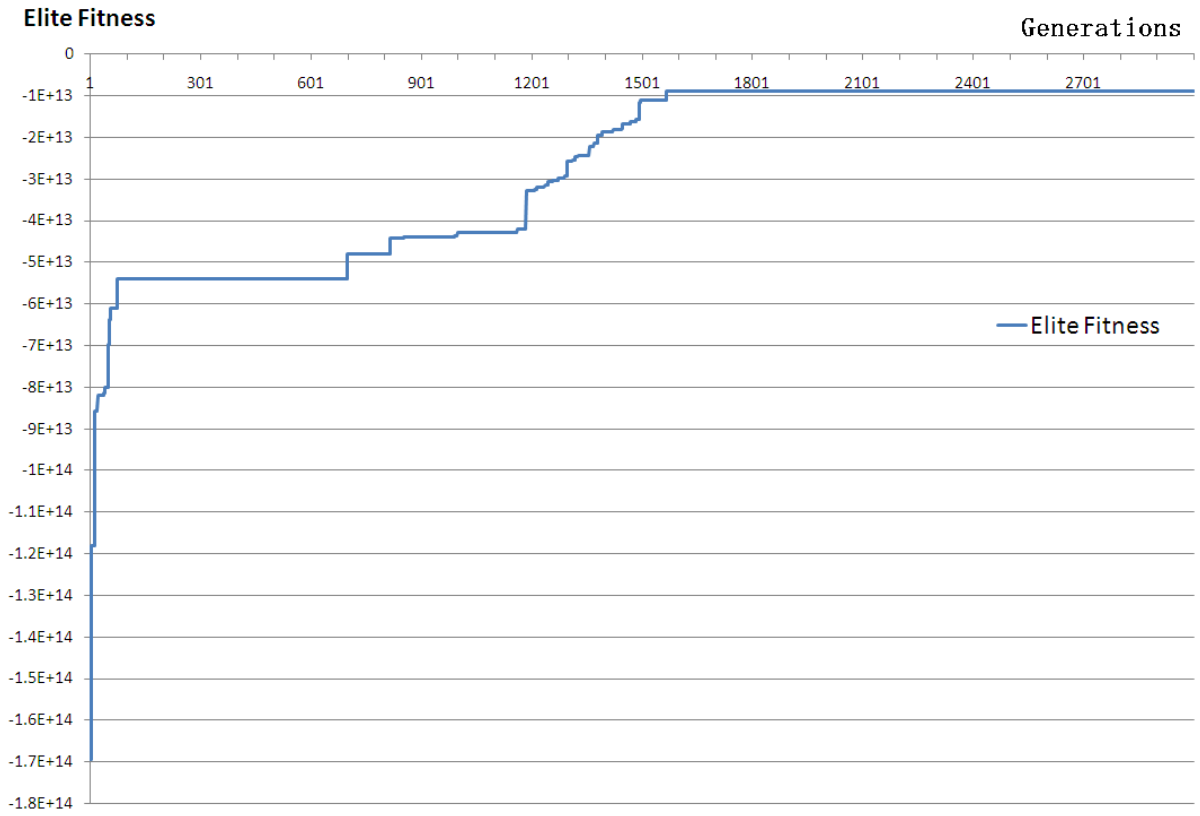


Figure 4.4: Elite fitness of GA (Y-axis) vs. the number of generations (X-axis).

evolution. The elite fitness is increasing during the evaluation time.

Table 4.5 shows the results of GA with one-point crossover, GA with two-point crossover and GApuc. For each GA, the results over 10 independent trials are used. "Average" means the average fitness value of 10 individuals; "Best one" means the best elite fitness value from 10 trials; "Worst one" means the worst elite fitness value from 10 trials; "time" means the average running time. The Mean Difference Per Pixel (MDPP) [49] is one of the indices to measure image visual quality, defined by: $MDPP = F_1/(M-2)/(N-2)$.

"Ratio" is the relative value of MDPP of each case compared to that of $GApuc(0.5)$. The larger the fitness is, the better the quality of the image filter is. The less the

Table 4.5: Results of GA with different crossover on salt-and-pepper noise.

Item	Average				Best one			Worst one		
	Fitness	MDPP	Ratio	Time	Fitness	MDPP	Ratio	Fitness	MDPP	Ratio
GA(one)	-30366060770697	1.164	1.782	3008	-21281786918352	0.813	2.398	-40986003212711	1.573	1.386
GA(two)	-28047502361136	1.075	1.647	3078	-13985925592497	0.535	1.577	-34228038597079	1.312	1.156
GApuc(0.1)	-23793601855133	0.911	1.396	3122	-13365407294613	0.510	1.503	-31858306263898	1.222	1.077
GApuc(0.2)	-22713411861392	0.870	1.332	3057	-12637228640667	0.482	1.422	-33754998637883	1.295	1.141
GApuc(0.3)	-28325975394119	1.086	1.662	3047	-10964236877206	0.418	1.233	-34714221520211	1.330	1.172
GApuc(0.4)	-20179654261105	0.772	1.182	3012	-9054626762316	0.343	1.012	-31888871497524	1.223	1.078
GApuc(0.5)	-17077462155600	0.653	1.000	3004	-8917525353615	0.339	1.000	-29615620999819	1.135	1.000

$GA(one)$: GA with one-point crossover.

$GA(two)$: GA with two-point crossover.

$GApuc(x)$: GA with parameterized uniform crossover with $(Ppuc : x)$, $x = 0.1, 0.2, 0.3, 0.4, 0.5$.

$MDPP$: the Mean Difference Per Pixel.

MDPP value is, the better the quality is. The less the ratio is, the better the quality is.

According to these results, GApuc produces better solutions than other GA, from the point of the best elite fitness. The parameterized uniform crossover is much more likely to distribute its disruptive trials in an unbiased manner over larger portions of the space, therefor it could find better solution. The parameterized uniform crossover is a special case of a uniform crossover, in which the parameter of crossover probability is used. The performance of GApuc was discussed in the paper [44].

The difference between the best and the worst case is large. The proposed method will be improved and compared with more other methods in the future work.

An example of chromosome of best one of $GApuc(0.5)$ is as follows:

$(4, 6, 1)(1, 7, 15)(3, 8, 15)(4, 1, 9)(12, 11, 9)(0, 0, 0)$
$(0, 0, 0)(0, 0, 0)(0, 0, 0)(13, 9, 11)(0, 0, 0)(0, 0, 0)(16, 19, 0)$
$(0, 0, 0)(0, 0, 0)(0, 0, 0)(21, 13, 11)(0, 0, 0)(0, 0, 0)$
$(18, 11, 14)(10, 28, 14)(0, 0, 0)(0, 0, 0)(0, 0, 0)(25, 9, 14)$

Table 4.6: Experiment environments

Item	Proposed	Vasicek [42]
GA	Elite selection	Elite selection
	Mutation	Mutation
	Crossover	-
Population	1210	8
Generation	3000	160000
Runs	10	64
All_individuals	36300000	81920000
Hardware	PC	FPGA

$All\_individuals = Population * Generation * Runs.$

$(0,0,0)(0,0,0)(0,0,0)(29,33,15)(0,0,0)(0,0,0)(0,0,0)$

The graphical representation of this chromosome is shown in Fig. 4.5.

Table 4.7 shows that the best MDPP value of $GApuc(0.5)$ is better than that in ref [42] which was the best one in this research field until now. Complexity, power and signal delay of this evolved image filter are also less than that in ref [42], as shown in Table 4.8, because less CLBs are used . Note that complexity, power and signal delay of ref [42] were recalculated by the proposed method. Table 4.6 shows the experiment environments in this chapter and the paper [42]. The number of all individuals of the proposed method is less than that of ref [42]. In the evolution of GA, crossover method could improve GA to find better results. F_2 makes the evolution to find the sample circuit of less complexity, power and signal delay.

Figure 4.6 shows the input images with 5% salt-and-pepper noise. The MDPP value of these images are 6.343, 6.314, 6.399, 6.373 and 6.386, respectively. Figure 4.7 shows the output images by the image filter of Fig. 4.5. The MDPP value of these

Table 4.7: Test results (MDPP) between different papers.

Test image	Proposed	Vasicek [42]	Median [42]
Lena	0.339	0.367	3.577
Aireplane	0.322	0.338	3.536
Bridge	0.563	0.657	7.830
Camera	0.434	0.627	4.413
Goldhill	0.389	0.451	5.870

The test images are with 5% salt-and-pepper noises.

Table 4.8: The best evolved image filter between different papers.

Item	Used	SD	Pb	Cb	Pw	Cw
Proposed	12	100	1331	2208	449	715
Vasicek [42]	17	132	2085	3457	792	1261

$Used$: the number of the used CLBs.

images are 0.339, 0.322, 0.563, 0.434 and 0.389, respectively. Obviously, the evolved image filter could reduce noise for all cases.

The image filter was evolved using Lena image and tested on other images. As the image is relatively large, the evolved filter is general. The filter can remove the same type of noise even if other images are used. In order to verify the general property, more other training images, test images and the type of noise will be used in the future work.

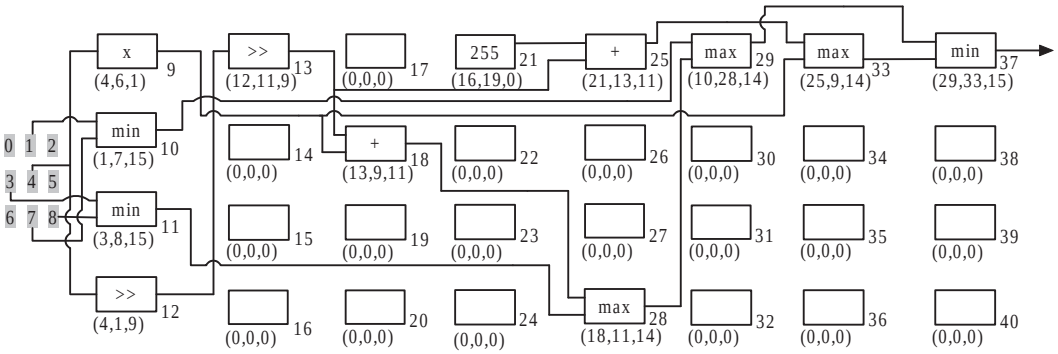


Figure 4.5: The optimized image filter of the best one.

Table 4.9: Results of GA with different crossover on gaussian noise.

Item	Average				Best one			Worst one		
	Fitness	MDPP	Ratio	Time	Fitness	MDPP	Ratio	Fitness	MDPP	Ratio
GA(one)	-92232132117266	3.543	1.032	3115	-90909133615331	3.491	1.053	-95937598641947	3.686	1.052
GA(two)	-91413860405079	3.511	1.022	3124	-90648118865578	3.482	1.050	-92518547446667	3.554	1.014
GApuc(0.1)	-91355184876030	3.509	1.022	3126	-88637725581150	3.405	1.027	-93288503206412	3.584	1.023
GApuc(0.2)	-90771275511862	3.486	1.015	3152	-88520140327854	3.399	1.025	-92874525580133	3.567	1.018
GApuc(0.3)	-90519932559665	3.476	1.012	3161	-88183518975416	3.386	1.021	-93090403211696	3.576	1.021
GApuc(0.4)	-89663479902260	3.444	1.003	3163	-87249310838306	3.351	1.010	-91809435240150	3.527	1.006
GApuc(0.5)	-89413330235145	3.434	1.000	3179	-86364048263411	3.317	1.000	-91202103208038	3.504	1.000

4.3.2 Experiments on gaussian noise

The image filter was evolved using Lena image and tested on other images with gaussian noise ($\mu = 0, \sigma = 65$) over 10 independent trials. Table 4.9 shows the results of GA with one-point crossover, GA with two-point crossover and GApuc on gaussian noise.

Table 4.10 compares the noise reduction between different noise of the best evolved image filter. The reduction ratio on salt-and-pepper noise is better than that on gaussian noise. The proposed method has better effective on salt-and-pepper noise, but not on gaussian noise. The noise value and noise distribution of gaussian noise are different from those of salt-and pepper noise. Therefore, another consideration is

Table 4.10: Comparison of the noise reduction between different noise.

Test image	Salt-and-pepper noise			Gaussian noise		
	Inputs	Outputs	Ratio	Inputs	Outputs	Ratio
Lena	6.343	0.339	0.947	6.082	3.317	0.455
Aireplane	6.314	0.322	0.949	6.121	3.296	0.462
Bridge	6.399	0.563	0.912	6.033	3.117	0.483
Camera	6.373	0.434	0.932	5.853	3.114	0.468
Goldhill	6.386	0.489	0.939	6.095	3.423	0.438

The test images are with 5% salt-and-pepper noises or gaussian noise ($\mu = 0, \sigma = 65$).

Ratio: 1-(Outputs/Inputs), the ratio of noise reduction.

required.

4.4 Conclusions

This paper described mixed constrained image filter design for noise reduction using a GApuc (Genetic Algorithm with parameterized uniform crossover) on a reconfigurable processing array. The complexity, power and signal delay in both CLBs (Configurable Logic Blocks) and wires are considered. An image filter for noise reduction is experimentally synthesized, to verify the validity of the proposed method. By evolution, the quality of the optimized image filter on reducing salt-and-pepper noise is better than that of other papers. Because the parameterized uniform crossover has effectiveness. Consequently the proposed design method is effective for mixed constrained image filter design on reducing salt-and-pepper noise.

In the future works, we will try to use more pixels as inputs to check whether the more pixels inputs can improve the quality of the evolved filter, consider how to reduce the noise at the border of the output image, do the experiment on FPGA to

check the efficiency of evolved instance on hardware in practice, and consider the new method for reducing gaussian noise.

(a) Lena (MDPP: 6.343). (b) Aireplane (MDPP: 6.314). (c) Bridge (MDPP: 6.399).

(d) Camera (MDPP: 6.373). (e) Goldhill (MDPP: 6.386).

Figure 4.6: The input images with noise.

(a) Lena (MDPP: 0.339). (b) Aireplane (MDPP: 0.322). (c) Bridge (MDPP: 0.563).

(d) Camera (MDPP: 0.434). (e) Goldhill (MDPP: 0.389).

Figure 4.7: The output images by the evolved filter of Fig. 4.5.

Chapter 5

MIXED CONSTRAINED IMAGE FILTER DESIGN USING PARTICLE SWARM OPTIMIZATION

This chapter describes evolutionary image filter design for noise reduction using particle swarm optimization (PSO), where mixed constraints on the circuit complexity, power and signal delay are optimized. First, the evaluating values about correctness, complexity, power and signal delay are introduced to the fitness function. Then PSO autonomously synthesizes a filter. To verify the validity of our method, an image filter for noise reduction is synthesized. The performance of resultant filter by PSO is similar to that of Genetic Algorithm (GA), but the running time of PSO is 10% shorter than that of GA.

5.1 Introduction

We proposed mixed constrained design optimization using GA for some combinational circuits [46, 47]. The proposed method could synthesize good circuits about complexity, power and signal delay, but it took the large running time for GA process. As another optimization method, particle swarm optimization (PSO) [10–12] was proposed and evaluated, and it is promising to find a good solution in shorter time, compared to GA.

This chapter applies PSO to mixed constrained image filter design for noise reduction, shown in Fig. 5.1. The circuit complexity, power and signal delay which are caused by both logic gates and wires, are optimized. In this design, first, the evaluating value about correctness, complexity, power and signal delay are introduced to the fitness function. Then PSO autonomously synthesizes an image filter which is simpler

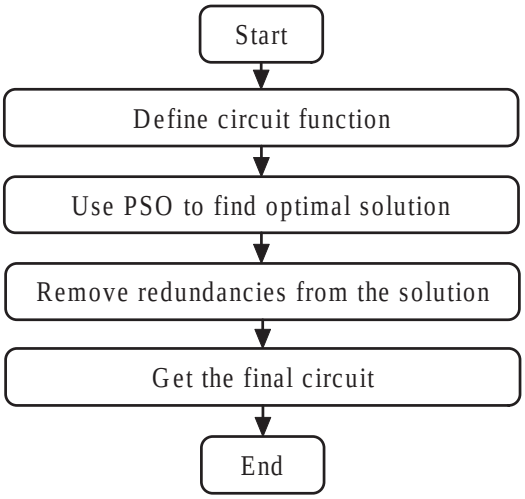


Figure 5.1: The overview of our method.

and has better performance than the conventional design. To verify the validity of our method, an image filter for reducing noise is experimentally synthesized.

The organization of this chapter is as follows: a brief overview of PSO is described in the next section. Section 5.3 describes design optimization for an image filter using PSO. Section 5.4 shows the experimental results. Finally, Sect. 5.5 concludes this chapter.

5.2 Particle Swarm Optimization

Particle swarm optimization (PSO) is an algorithm model on swarm intelligence that finds a solution to an optimization problem in a search space, shown in Fig. 5.2.

In PSO, a particle represents a candidate solution to the problem. Each particle is treated as a point in the D-dimensional problem space. The i-th particle is represented as $X_i = (x_{i1}, x_{i2}, \cdots, x_{iD})$. The best previous position (the position giving the best fitness value) of the i-th particle is recorded and represented as $P_i = (p_{i1}, p_{i2}, \cdots, p_{iD})$. The index of the best particle among all the particles in the population is represented by the symbol g. The rate of the position change (velocity) for particle i is represented

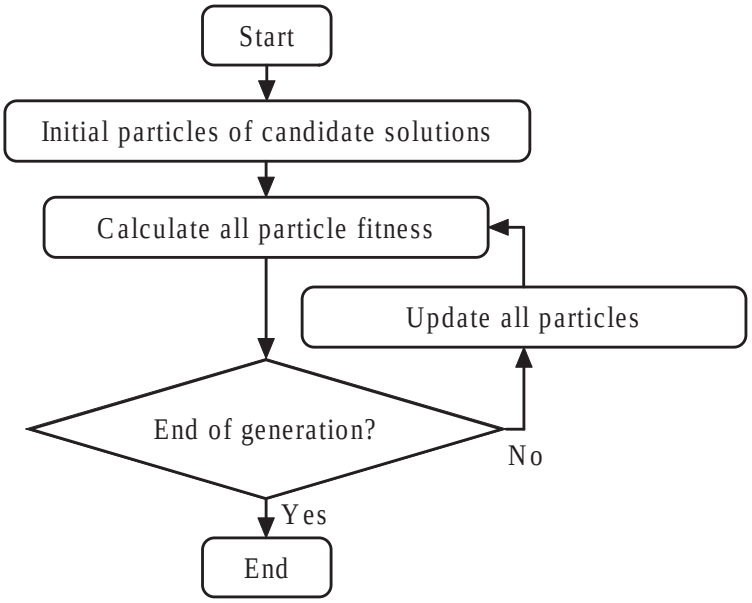


Figure 5.2: The evolutionary process of PSO.

as $V_i = (v_{i1}, v_{i2}, \cdots, v_{iD})$. The particle is updated according to the following equations:

$$\begin{aligned} v_{id}^{(t+1)} = \quad & w * v_{id}^{(t)} + c_1 * rand() * (p_{id}^{(t)} - x_{id}^{(t)}) \\ & + c_2 * Rand() * (p_{gd}^{(t)} - x_{id}^{(t)}), \qquad (5.1) \\ x_{id}^{(t+1)} = \quad & x_{id}^{(t)} + v_{id}^{(t+1)}. \qquad (5.2) \end{aligned}$$

where,

$0 \leq i \leq (n-1), 1 \leq d \leq D$.

n**:** number of particles in a group.

D**:** number of members in a particle.

t**:** pointer of iterations (generations).

w**:** inertia weight factor.

c_1, c_2**:** acceleration constant.

$rand(), Rand()$**:** uniform random value in the range [0,1].

$v_{id}^{(t)}$**:** velocity of particle i at iteration t, $V_{id}^{min} \leq v_{id}^{(t)} \leq V_{id}^{max}$.

$x_i^{(t)}$**:** current position of particle i at iteration t.

The inertia weight factor w is employed to control the impact of the previous history of velocities on the current velocity, thereby influencing the trade-off between global (wide-ranging) and local (fine-grained) exploration abilities of the "flying points". A larger inertia weight facilitates global exploration (searching new areas) while a smaller inertia weight tends to facilitate local exploration to free-tune the current search area. Suitable selection of the inertia weight provides a balance between global and local exploration abilities and thus requires fewer iterations on average to find the optimum. Good values of w are usually slightly less than 1 [12]. It could be randomly initialized for each particle. Or a high value of w at the beginning of the run facilitates global search, while a small w tends to localize the search.

c_1 and c_2 are constants that say how much the particle is directed towards good positions. They represent a "cognitive" and a "social" component, respectively, in that they affect how much the particle's personal best and the global best (respectively) influence its movement. Usually we take $c1 = c2 = 2$ [12].

5.3 Image Filter Design using PSO

PSO is applied to search good solutions to optimize the image filter design.

The target image filter is to provide identical functional behavior with less complexity, less power and less signal delay.

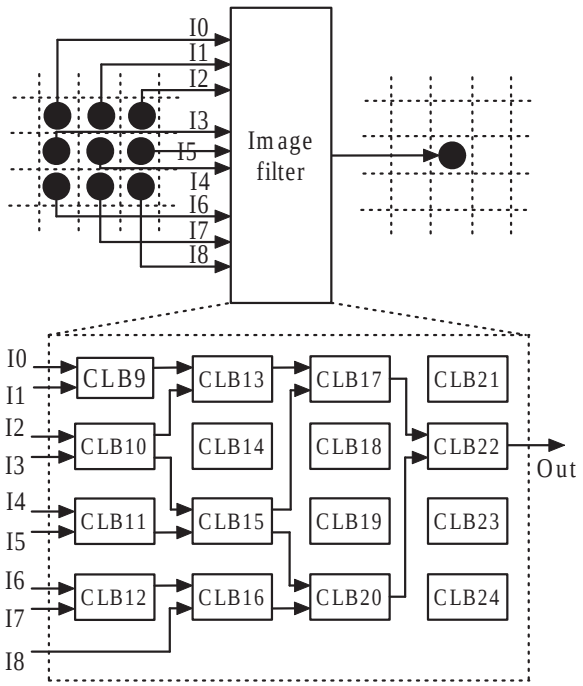


Figure 5.3: An example image filter.

5.3.1 Image Filter

Every image operator is considered as a digital circuit with nine 8-bit inputs and a single 8-bit output, which processes gray-scaled (8-bit/pixel) images.

As shown in Fig. 5.3, every pixel value of the filtered image is calculated using a corresponding pixel and its eight neighbors in the processed image [42].

5.3.2 Reconfigurable Processing Array for Image Filter

Similarly to [41], the reconfigurable image filter is implemented as a Virtual Reconfigurable Circuits (VRC) (Fig. 5.4). As a new pixel value is calculated using nine pixels, the VRC has got nine 8-bit inputs and a single 8-bit output. The VRC consists of two-input CLBs (Configurable Logic Blocks in FPGA) placed in a $4 * 4$ array. Any input of each CLB may be connected to either a primary circuit input or the output

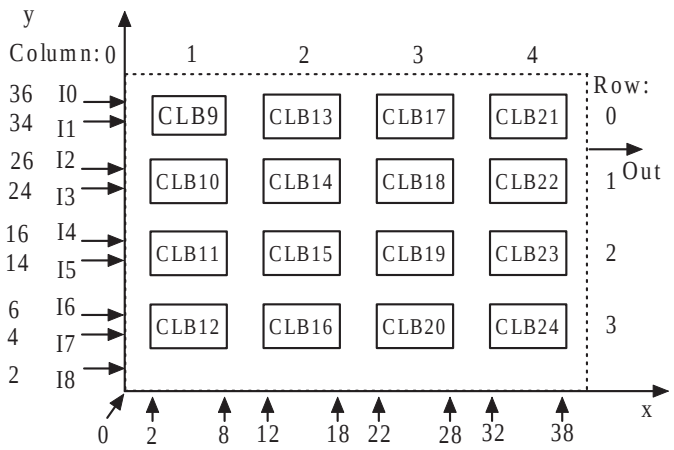


Figure 5.4: A reconfigurable processing array.

of a CLB in the preceding column. Any CLB can be programmed to implement one of the functions given in Table 4.2 [42], all these functions operate with 8-bit operands and produce 8-bit results. Table 4.2 also gives value of complexity (FC), power (FP) and signal delay (SD) for each function in a CLB.

In Fig. 5.4, there are position of inputs and output of each logic block. Therefore, we add values of wire about complexity, power and signal delay.

5.3.3 Genetic Encoding

The chromosome (particle) is a string of integers where each three continuous integers constitute a logic block. Each triplet in the chromosome encodes the two inputs and the function type of a logic block, respectively, such as:

$$(Input\_1, Input\_2, Function\_type).$$

A typical chromosome then can be a sequence of triplets [46, 47], such as:

$$((IN_1^1, IN_2^1, F_{type}^1) \cdots (IN_1^i, IN_2^i, F_{type}^i) \cdots)$$

Here, IN_1^i and IN_2^i mean the positions of the corresponding input signals. F_{type}^i means the function type of a logic block. For a primary input, $0 \leq IN^i \leq 8$. For the input from the output of a logic block $CLBm$ shown in Fig. 5.4, $IN^i = m$. The function in a CLB is defined as shown in Table 4.2.

5.3.4 Fitness Function

The pixels of corrupted image ci are used as inputs of VRC. Pixels of filtered image fi are generated, which are compared to the pixels of original image oi.

The design objective is to minimize the difference between the filtered image and the original image. The image size is $M*N$ pixels, but only the area of $(M-2)*(N-2)$ pixels is considered, because the pixel values at the borders are ignored, and thus remain unfiltered. The fitness value of a candidate filter is obtained as follows:

(1) the VRC is configured using a candidate chromosome

(2) the created circuit is used to produce pixel values in the image fi and

(3) the fitness value is calculated as

$$Fitness = (-1) * (F_1 * \beta + F_2). \tag{5.3}$$

where,

F_1 and F_2 are defined as follows and β is the weight on F_1.

$$F_1 = \sum_{i=1}^{M-2} \sum_{j=1}^{N-2} (|fi(i,j) - oi(i,j)|). \tag{5.4}$$

where,

M: the number of columns of the pixels in the image.

N: the number of rows of the pixels in the image.

$fi(i,j)$: the pixel (i, j) in filtered image fi, the value range is [0,255].

$oi(i,j)$: the pixel (i, j) in original image oi, the value range is [0,255].

$$\begin{aligned} F_2 = \quad & SD * \alpha_{sd} + Pg * \alpha_{pg} + Cg * \alpha_{cg} \\ & + Pw * \alpha_{pw} + Cw * \alpha_{cw}. \end{aligned} \tag{5.5}$$

where,

SD: signal delay of a circuit individual, determined by a critical path.

α_{sd}: the weight on signal delay in F_2.

Pg: power of logic blocks in a circuit, calculated by summation of all logic block's power.

α_{pg}: the weight on power of logic blocks in F_2.

Cg: complexity of logic blocks in a circuit, calculated by summation of all logic block's complexity.

α_{cg}: the weight on complexity of logic blocks in F_2.

Pw: power of all wires in a circuit, calculated by summation of all wire's power.

α_{pw}: the weight on power of wires in F_2.

Table 5.1: Conditions for evolution.

Number of Generation : 100.
Population Size : 600.
Inertia weight factor w : [0.4, 1.0]. Limit of change in velocity of each member in an individual: $V_{id}^{max} = 0.5 * p_{id}^{max}, V_{id}^{min} = -0.5 * p_{id}^{max}$. Acceleration constant : c1 = 2, c2 = 2.

Cw**:** complexity of wires in a circuit, calculated by summation of all wire's complexity.

α_{cw}**:** the weight on complexity of wires in F_2.

The priority of evaluating values in Eq. (5.3) is: $F_1 > F_2$. In this experiment, β is set to $0.1 * 10^9$. The priority of evaluating values in Eq. (5.5) is: $SD > Pg > Cg > Pw > Cw$. In this experiment, α_{sd} is set to $0.1 * 10^6$, α_{pg} is set to $1 * 10^3$, α_{cg} is set to $0.1 * 10^3$, α_{pw} is set to 10, and α_{cw} is set to 1. All α's and β are empirically assigned in our experiment.

5.4 Experimental Results

Table 5.1 shows the parameters of the evolution of PSO used in this experiment. Some preliminary experiments were performed in advance to decide parameters suitable for our experiment.

The proposed method was implemented in Eclipse SDK 3.1.1 with jre 1.6.0; and tested on a PC with Inter(R) Core(TM)2 CPU at 2.67 GHz and 2.0 GB RAM.

The image filter is evolved for a $512 * 512$ Lena image corrupted by 5% salt-and-pepper noise, shown in Fig. 5.7 (a).

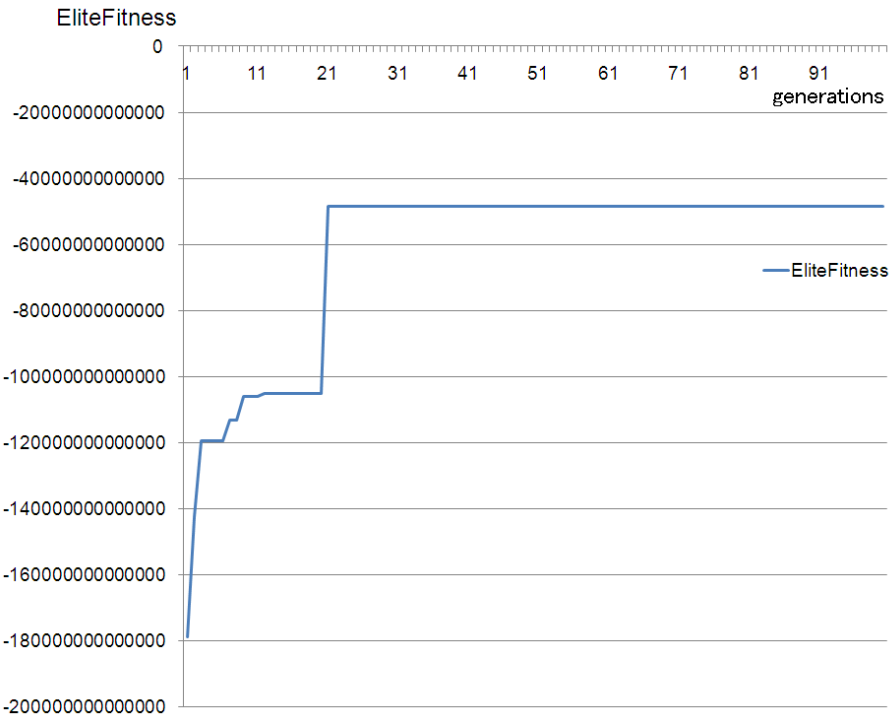


Figure 5.5: Elite fitness of PSO (Y-axis) vs the number of generations (X-axis).

Fig. 5.5 shows the elite fitness of PSO with $w = 0.9$ vs the number of generations during the image filter evolution. The elite fitness is increasing during evaluation time.

Table 5.4 shows the results of PSO with different w. For each PSO, we use results over 10 independent trials. "max" means the best elite fitness value from 10 trials; "average" means the average fitness value of 10 individuals; "time" means the average running time (minutes) of one trial from 10 trials. The larger the fitness is, the better the image filter is. The less the ratio is, the better the image filter is.

From the results, we can see that PSO(0.9) produces better solutions than others, from the point of the best elite fitness.

An example of chromosome by PSO(0.9) with fitness -48550809287267 is as follows:

Table 5.2: Fitness values of PSO with different w, and GA.

Item	Max		Average		Running time	
	Fitness	Ratio	Fitness	Ratio	Time(m)	Ratio
$PSO(0.4)$	-80005207479684	1.65	-99745634920892	1.11	42.80	0.99
$PSO(0.5)$	-80535402393364	1.66	-94873624466988	1.06	43.60	1.01
$PSO(0.6)$	-79181905512328	1.63	-97375325953839	1.09	43.20	0.99
$PSO(0.7)$	-56076103257304	1.15	-94746895485425	1.06	42.60	0.98
$PSO(0.8)$	-57981907477527	1.19	-91487054845692	1.02	41.70	0.96
$PSO(0.9)$	-48550809287267	1.00	-89652125806386	1.00	43.00	1.00
$PSO(1.0)$	-85478004803230	1.76	-102629344653216	1.14	40.00	0.92
$PSO(w1)$	-82558108959223	1.70	-93726626111394	1.05	42.80	0.99
$PSO(w2)$	-63176503079148	1.30	-91741585167119	1.02	42.00	0.97
GA	-47595302921595	0.98	-64678393342856	0.72	47.70	1.10

$PSO(w)$: PSO with $w = 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0$.

$PSO(w1)$: PSO with $w = 1.0 - 0.6 * generation/generation\_size$.

$PSO(w2)$: PSO with $w = 0.4 + 0.6 * random()$.

GA: Number of Generation, Population Size is same to that of PSO; Elite Size = 10, Crossover Probability (Pc) = 0.3, Mutation Probability (Pm) = 0.1, (ref. [47]).

$(0, 0, 0)(1, 7, 15)(3, 5, 14)(8, 5, 15)(0, 0, 0)(0, 0, 0)$
$(0, 0, 0)(11, 12, 14)(0, 0, 0)(0, 0, 0)(0, 0, 0)(4, 16, 15)$
$(10, 20, 14)(0, 0, 0)(0, 0, 0)(0, 0, 0)$

The graphical representation of this chromosome is shown in Fig. 5.6.

As the image is relatively large, we can assume that the evolved filter is general. The filter is able to remove the same type of noise also from other images. The image filter was evolved using Lena image and tested on other images.

Figures in Fig. 5.7 show the input images with 5% salt-and-pepper noise, the

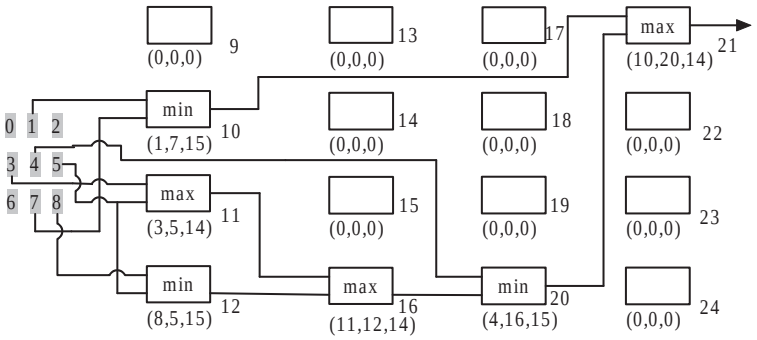
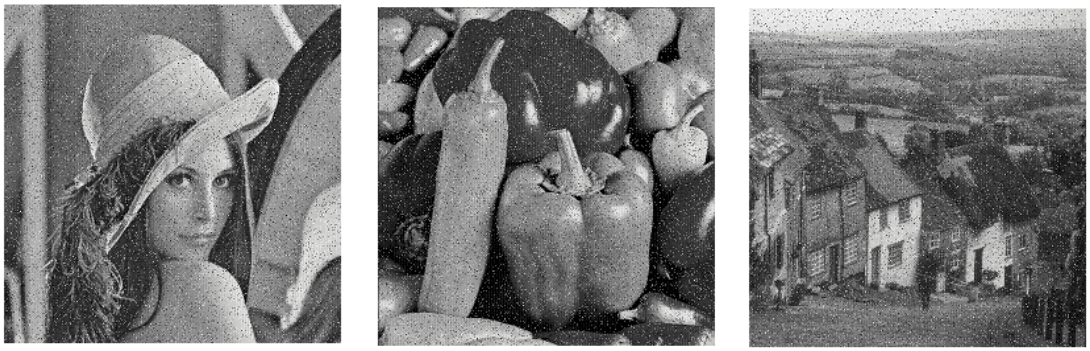


Figure 5.6: The optimized image filter by PSO (0.9).

MDPP value of these images are 6.42, 6.29 and 6.35, respectively. Figures in Fig. 5.8 show the output images by the image filter in Fig. 5.6, the MDPP value of these images are 1.87, 2.37 and 2.51, respectively. Obviously, this image filter could reduce noise for all cases.

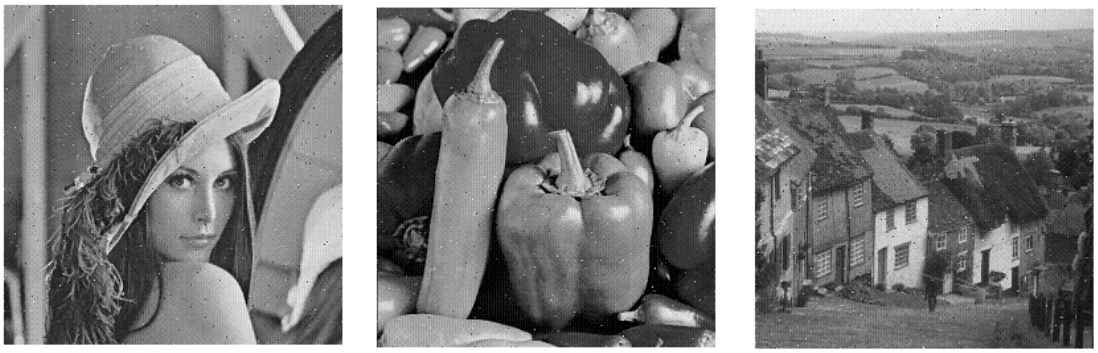
5.5 Conclusions

This chapter proposed the use of PSO (Particle Swarm Optimization) for mixed constrained image filter design for noise reduction. The complexity, power and signal delay both of the CLBs (Configurable Logic Blocks) and wires are considered. An image filter for removing noise is experimentally synthesized using PSO, to verify the validity of our method. By evolution, quality of the optimized image filter by PSO(0.9) is almost same as that of GA, but the running time of PSO is 10% shorter than that of GA.



(a) Lena (MDPP: 6.42) (b) Peppers (MDPP: 6.29) (c) Goldhill (MDPP: 6.35)

Figure 5.7: The input images with noise.



(a) Lena (MDPP: 1.87) (b) Peppers (MDPP: 2.37) (c) Goldhill (MDPP: 2.51)

Figure 5.8: The output images by the evolved filter of Fig. 5.6.

Chapter 6

FAULT-TOLERANT IMAGE FILTER DESIGN USING GA

This paper describes mixed constrained image filter design with fault tolerant using Genetic Algorithm (GA) on a reconfigurable processing array. There may be some faulty Configurable Logic Blocks (CLBs) in a reconfigurable processing array at random. The proposed method with GA autonomously synthesizes a filter fitted to the reconfigurable device with some faults, evaluating the complexity, power and signal delay in both CLBs and wires. An image filter for noise reduction is experimentally synthesized to verify the validity of our method. By evolution, the quality of the optimized image filter on a reconfigurable device with faults is almost same as that with no fault.

6.1 Introduction

The high density of chips increases the possibility of faulty components and the complexity of designs increases the probability of human errors. The acceptance for faults is diminishing as the systems are demanded for high reliability. The need for fault-tolerant designs is stated as the long-term grand challenges in [50].

The above may be summarized as two key demands: novel automated design and fault tolerance. Recently, a new research field exploring solutions to these problems has grown, that is the field of bio-inspired hardware design. Biological organisms such as humans are in many ways extremely complex, yet nature has managed to evolve creatures that utilize their physical, chemical, electrical and biological properties in intricate complex dynamical ways. In addition, they are tolerant of faults on many levels in that they keep on functioning even though cells or sometimes even entire

Table 6.1: Features of related works.

Item	Level	Circuit	Optimization	Fault
Vasicek [42]	function	image filter	no	no
Bao [44]	gate	adder	yes	no
Bao [47]	function	image filter	yes	no
Proposed	function	image filter	yes	yes

limbs fail [51].

Vasicek et al [42, 43] discussed image filter design at function level by using evolutional approach, but they did not discuss any circuit constraints such as complexity, power consumption and signal delay, and did not deal with fault tolerant. While we have proposed mixed constrained design optimization using Genetic Algorithm (GA) for some combinational circuits (such as adder), the discussion was at gate level [44–46], but also did not deal with fault tolerant.

This chapter describes mixed constrained image filter design for noise reduction using GA on a reconfigurable processing array [52]. There may be some faulty Configurable Logic Blocks (CLBs) in a reconfigurable processing array at random. The circuit complexity, power and signal delay in both logic blocks and wires are optimized. The differences among the related works are shown in Table 6.1. In this design, first, the evaluating value about correctness, complexity, power and signal delay are introduced to the fitness function. Then GA autonomously synthesizes an image filter which is simple and has better performance and fits to the reconfigurable processing array with faults. To verify the validity of our method, an image filter for noise reduction is experimentally synthesized.

The organization of this paper is as follows: Section 6.2 describes fault-tolerant design optimization for an image filter using GA. Section 6.3 shows the experimental

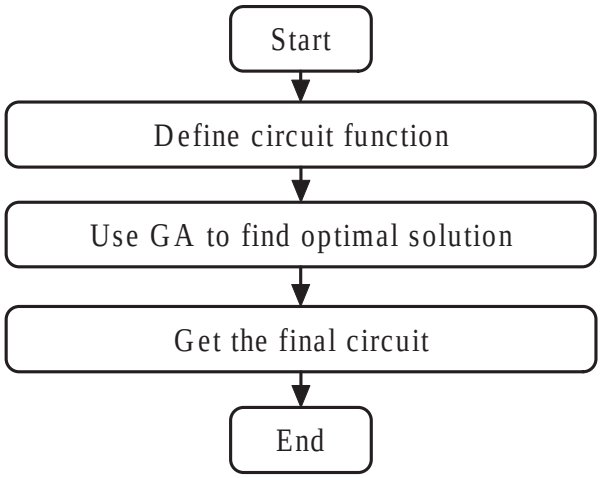


Figure 6.1: The overview of our method.

results. Finally, Sect. 6.4 concludes this paper.

6.2 Fault-tolerant Image Filter Design using GA

GA is applied to search good solutions to optimize the image filter design on a reconfigurable processing array with some faults.

The resultant image filter has an identical functional behavior with less complexity, less power and less signal delay.

6.2.1 Image Filter

Every image operator is considered as a digital circuit with nine 8-bit inputs and a single 8-bit output, which processes gray-scaled (8-bit/pixel) images.

As shown in Fig. 4.1, every pixel value of the filtered image is calculated using a corresponding pixel and its eight neighbors in the processed image [41, 42].

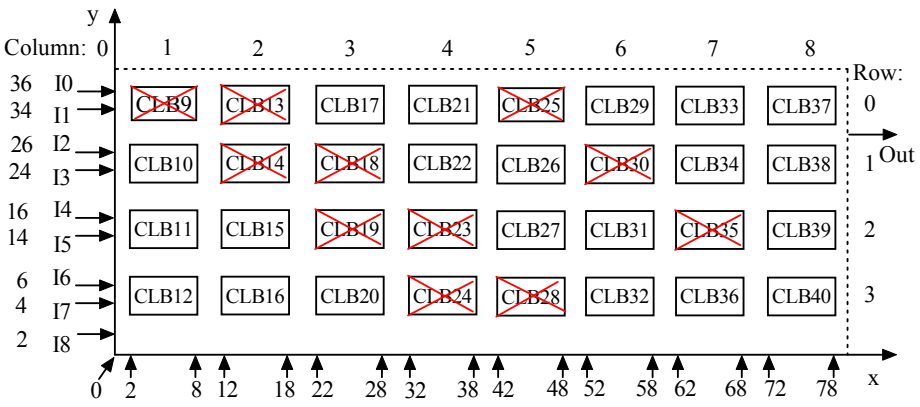


Figure 6.2: A reconfigurable processing array with faults.

6.2.2 Reconfigurable Processing Array for Image Filter

The reconfigurable image filter is implemented as a Virtual Reconfigurable Circuits (VRC) (Fig. 6.2) proposed in [41]. As a new pixel value is calculated using nine pixels, the VRC has got nine 8-bit inputs and a single 8-bit output. The VRC consists of two-input Configurable Logic Blocks (CLBs) placed in a $8 * 4$ array. Any input of each CLB may be connected to either a primary circuit input or the output of a CLB in the preceding column. Any CLB can be programmed to implement one of the functions given in Table 4.2 [47, 48], all these functions operate with 8-bit operands and produce 8-bit results.

The CLB with different function, has different complexity, power and signal delay. We newly define the values of complexity (FC), power (FP) and signal delay (SD) for each function in a CLB, as in Table 4.2. The gate level circuit with different function, has different number of gates. We define the values of FC, FP and SD considering the number of gates in each function.

As shown in Fig. 6.2, the coordinates of inputs and output of each logic block are defined based on VRC, so that we can calculate the critical length of connection

wires.

There may be some faulty CLBs in a reconfigurable processing array at random. The output of a faulty CLB is a value in the range [0,255] at random.

6.2.3 Genetic Encoding

The chromosome is a string of integers where each three continuous integers constitute a logic block. Each triplet in the chromosome encodes the two inputs and the function type of a logic block, respectively, such as [45, 46]:

$$(Input\_1, Input\_2, Function\_type).$$

A typical chromosome then can be a sequence of triplets such as:

$$(IN_1^1, IN_2^1, F_{type}^1) \cdots (IN_1^i, IN_2^i, F_{type}^i) \cdots$$

Here, IN_1^i and IN_2^i mean positions of the corresponding input signal. F_{type}^i means function type of logic block. For primary input, $0 \leq IN^i \leq 8$. For input from output of a logic block $CLBm$ shown in Fig. 6.2, $IN^i = m$. Function in a CLB is defined as shown in Table 4.2.

6.2.4 Fitness Function

The pixels of corrupted image ci are used as inputs of VRC. Pixels of filtered image fi are generated, which are compared to the pixels of original image oi.

The design objective is to minimize the difference between the filtered image v and the original image w. The image size is $M * N$ pixels, but only the area of $(M-2) * (N-2)$ pixels is considered, because the pixel values at the borders are ignored, and thus remain unfiltered. The fitness value of a candidate filter is obtained as follows:

(1) VRC is configured using a candidate chromosome,

(2) the created circuit is used to produce pixel values in the image fi, and

(3) the fitness value is calculated as

$$Fitness = (-1) * (F_1 * \beta + F_2). \tag{6.1}$$

where,

F_1 and F_2 mean correctness and quality of the circuit respectively, and are defined as follows. β is the weight on F_1.

$$F_1 = (\sum_{i=1}^{M-2} \sum_{j=1}^{N-2} |fi(i,j) - oi(i,j)|). \tag{6.2}$$

where,

M**:** the number of columns of the pixels in the image.

N**:** the number of rows of the pixels in the image.

$fi(i,j)$**:** the pixel (i, j) in filtered image fi, the value range is [0,255].

$oi(i,j)$**:** the pixel (i, j) in original image oi, the value range is [0,255].

$$\begin{aligned} F_2 = \quad & SD * \alpha_{sd} + Pb * \alpha_{pb} + Cb * \alpha_{cb} + \\ & Pw * \alpha_{pw} + Cw * \alpha_{cw}. \end{aligned} \tag{6.3}$$

where,

SD**:** signal delay of a circuit individual, determined by a critical path.

α_{sd}**:** the weight on signal delay in F_2.

Pb**:** power of logic blocks in a circuit, calculated by summation of all logic block's power.

α_{pb}**:** the weight on power of logic blocks in F_2.

Cb**:** complexity of logic blocks in a circuit, calculated by summation of all logic block's complexity.

α_{cb}**:** the weight on complexity of logic blocks in F_2.

Pw**:** power of all wires in a circuit, calculated by summation of all wire's power.

α_{pw}**:** the weight on power of wires in F_2.

Cw**:** complexity of wires in a circuit, calculated by summation of all wire's complexity.

α_{cw}**:** the weight on complexity of wires in F_2.

The priority of evaluating values in Eq. (6.1) is: $F_1 > F_2$. In this experiment, β is set to $0.1 * 10^9$. The priority of evaluating values in Eq. (6.3) is: $SD > Pb > Cb > Pw > Cw$. In this experiment, α_{sd} is set to $1 * 10^9$, α_{pb} is set to $1 * 10^5$, α_{cb} is set to $1 * 10^3$, α_{pw} is set to 100, and α_{cw} is set to 1. All α's and β are empirically assigned in our experiment.

F_1 is the difference between pixels in filtered image and original image; the less the value is, the better the circuit is. F_2 searches the optimum solution considering complexity, power and signal delay simultaneously in both logic blocks and wires; the less the value is, the better the quality of circuit is. In the evolution, the larger the fitness is, the better the quality of image filter is.

Table 6.2: Conditions for evolution.

Number of Generation : 3000
Population Size : 1210
Elite Size : 10
Crossover Size : 600
Mutation Size : 600
Probability in PUC ($Ppuc$) : 0.5
Mutation Probability (Pm) : 0.1
Faulty CLBs : 0,2,4,6,8,10,12

PUC: Parameterized Uniform Crossover.

6.3 Experimental Results

Table 6.2 shows the parameters of the evolution of GA used in this experiment, where faulty CLBs are set. These parameters are decided based on the results of papers [42, 44], and some preliminary experiments were performed in advance. Crossover may exchange some logic blocks between two individuals by $Ppuc$, here is 50% of all logic blocks. Mutation may change some genes in one individual by Pm, here is, 10% of all genes, input number or function type of a logic block in an image filter. The number of faulty CLBs in the reconfigurable processing array is set as $0, 2, 4, 6, 8, 10, 12$, at random position.

The proposed method was implemented in Eclipse SDK 3.5.1 with Java Runtime Environment(JRE) 1.6.0; and tested on a PC with Inter(R) Core(TM) i7 CPU at 3.33 GHz and 9.0 GB RAM.

The image filter is evolved for a $512 * 512$ Lena image corrupted by 5% salt-and-pepper noise, shown in Fig. 6.5 (a).

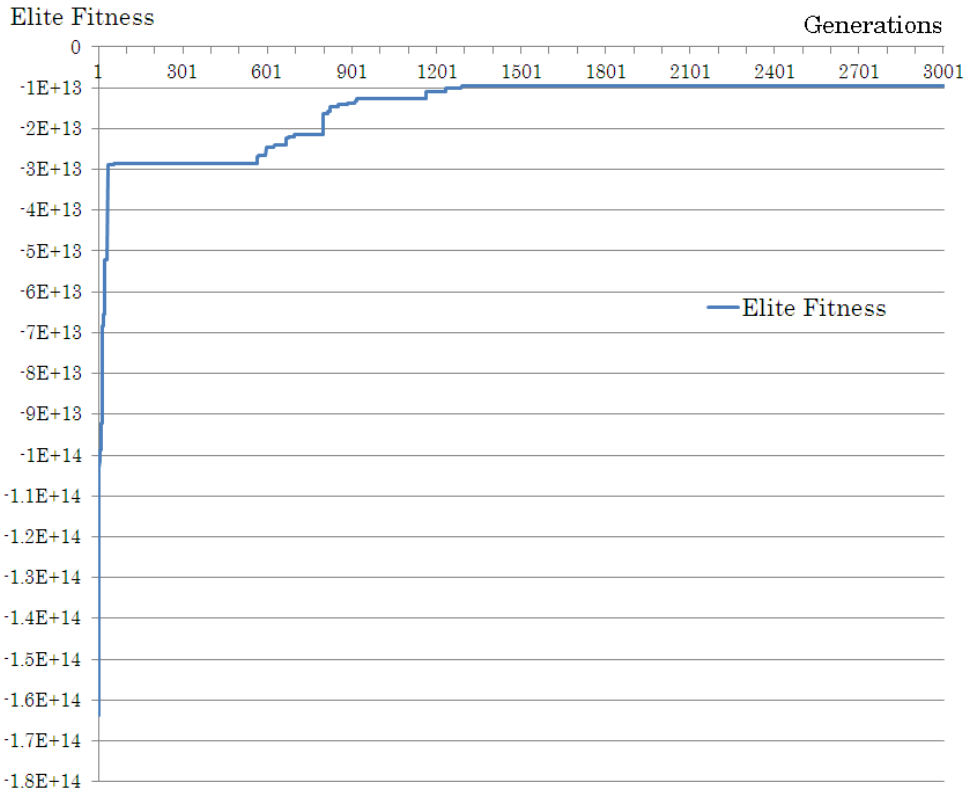


Figure 6.3: Elite fitness of GA (Y-axis) vs. the number of generations (X-axis).

Fig. 6.3 shows the elite fitness of GA with parameterized uniform crossover (GA-puc) with ($Ppuc$: 0.5) vs. the number of generations during the image filter evolution. The elite fitness is increasing during evaluation time.

Table 6.3 shows the results on a reconfigurable processing array with different faults. For each case, we execute over 10 independent trials. "Available" means the number of available CLBs. "Average" means the average values of 10 individuals, there are the number of used CLBs, the value of the MDPP and running time. "Ratio" is the relative value of MDPP of each case compared to that of no fault $Faults(0)$. The larger the fitness is, the better the quality of image filter is. The less the MDPP value is, the better the quality is. The less the ratio is, the better the quality is.

"Best one" means the best optimized image filter from 10 trials. "Worst one"

Table 6.3: Results on a reconfigurable processing array with different faults.

Item	CLBs	Average				Best one								Worst one		
	Available	Used	MDPP	Ratio	Time	Used	MDPP	Ratio	SD	Pb	Cb	Pw	Cw	Used	MDPP	Ratio
Faults(0)	29	9.3	0.760	1.000	3004	10	0.339	1.000	112	1055	1749	449	713	8	1.071	1.000
Faults(2)	27	8.2	1.101	1.448	3051	9	0.364	1.075	99	705	1166	314	498	6	2.035	1.900
Faults(4)	25	6.7	1.545	2.032	3046	8	0.376	1.110	74	677	1123	350	558	6	2.107	1.968
Faults(6)	23	7.2	1.464	1.926	3054	7	0.388	1.146	101	693	1148	339	539	6	2.058	1.922
Faults(8)	21	7.5	1.508	1.984	3048	8	0.442	1.306	98	890	1476	376	599	7	2.091	1.952
Faults(10)	19	6.5	1.750	2.302	3043	7	0.450	1.328	76	665	1102	288	458	5	2.172	2.027
Faults(12)	17	5.7	1.544	2.044	3049	6	0.524	1.547	90	668	1108	308	482	3	2.205	2.058

Available: In a 8*4 CLBs array, only one is useful on the last column, so the max number of available CLBs is 29.

$Faults(x)$: A reconfigurable processing array with x faulty CLBs at random position, $x = 0, 2, 4, 6, 8, 10, 12$.

$Used$: the number of the used CLBs.

means the worst optimized image filter from 10 trials.

The quality of the optimized image filter on a reconfigurable processing array with a few faults (2-6 faults) is almost same as that on a reconfigurable processing array with no fault, that is less than 14.6% in the item of different value of MDPP of the best one.

The quality of the optimized image filter of $Fault(2)$ is only 7% less than that of $Fault(0)$ in the item of the MDPP value of the best one, but the values of SD, Pg, Cg, Pw, Cw of the circuit of $Fault(2)$ are better than that of $Fault(0)$.

An example of chromosome of best one of $fault(2)$ is as follows:

$$(1,7,15)(3,5,13)(4,5,1)(0,0,0)(9,10,14)(11,9,2)$$
$$(0,0,0)(0,0,0)(0,0,0)(0,0,0)(0,0,0)(0,0,0)(0,0,0)$$
$$(0,0,0)(0,0,0)(13,14,12)(0,0,0)(0,0,0)(0,0,0)$$
$$(14,9,5)(0,0,0)(0,0,0)(0,0,0)(0,0,0)(0,0,0)(0,0,0)$$
$$(0,0,0)(11,24,15)(28,36,14)(0,0,0)(0,0,0)(0,0,0).$$

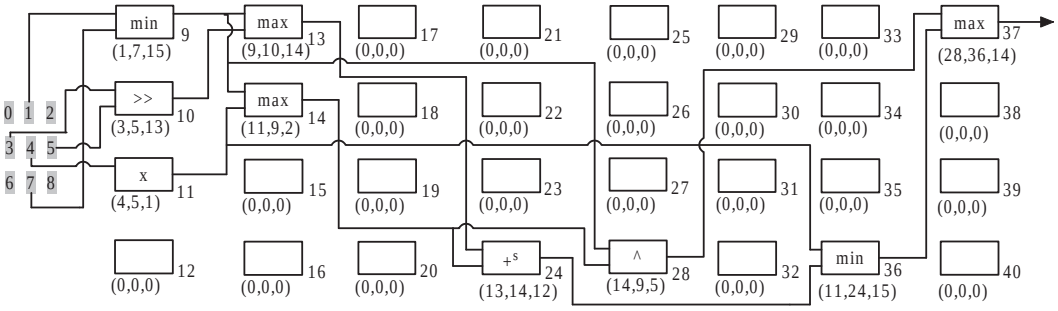


Figure 6.4: The optimized image filter of $Fault(2)$.

The graphical representation of this chromosome is shown in Fig. 6.4.

As the image is relatively large, we can say that the evolved filter is general. The filter is able to remove the same type of noise also from other images. The image filter was evolved using Lena image and tested on other images.

Figures in Fig. 6.5 show the input image with 5% salt-and-pepper noise, the MDPP value of these images are 6.380, 6.458 and 6.321, respectively. Figures in Fig. 6.6 show the output image by the image filter of Fig. 6.4, the MDPP value of these images are 0.364, 0.377 and 0.420, respectively. Obviously, this image filter could reduce noise for all cases.

6.4 Conclusions

This chapter described mixed constrained image filter design with fault tolerance for noise reduction using GA (Genetic Algorithm) on a reconfigurable processing array. The complexity, power and signal delay in both CLBs (Configurable Logic Blocks) and wires are considered. An image filter for noise reduction is experimentally synthesized, to verify the validity of our method. By evolution, the quality of the optimized image filter on a reconfigurable processing array with a few faults is almost same as that on a reconfigurable processing array with no fault. Consequently our proposed design method is effective for fault-tolerant optimization.

(a) Lena (MDPP: 6.380) (b) Cameraman (MDPP: 6.458) (c) Peppers (MDPP: 6.321)

Figure 6.5: The input images with noise.

(a) Lena (MDPP: 0.364) (b) Cameraman (MDPP: 0.377) (c) Peppers (MDPP: 0.420)

Figure 6.6: The output images by the evolved filter of Fig. 6.4.

We will also apply GA to autonomous design circuits for more complex functional requirements, and enhance more practical information about circuit to fitness function. Future works are to find better genetic encoding method to apply GA to large sized circuits, and to improve GA to reduce the processing time.

Chapter 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

This research focus on mixed constrained circuit design by evolutionary technologies, to get the optimization circuit in short processing time, and deal with the fault tolerance circuit design.

Chapter 2 applied GA to gate level circuit design optimization. First, we introduced the evaluating value about correctness, complexity, power, and signal delay to the fitness function. Then GA can autonomously synthesize a circuit that is equivalent to a conventional design in function, but is simpler and has better performance. To verify the effectiveness of our method, a simple 2-bit full adder circuit is experimentally synthesized. By evolution, GApuc (Genetic Algorithm with parameterized uniform crossover) can find optimized circuits with less complexity, less power, and less signal delay than GA with one-point crossover or two-point crossover.

Chapter 3 proposed GAdss (Genetic Algorithm with different structure selection) and its application to autonomous design optimization for combinatorial circuits. By evolution, GAdss can find optimized circuits with less complexity, less power and less signal delay than traditional GA, because different structure selection has effectiveness.

Chapter 4 described mixed constrained image filter design for noise reduction using a GApuc on a reconfigurable processing array. The complexity, power and signal delay in both CLBs (Configurable Logic Blocks) and wires are considered. An image filter for noise reduction is experimentally synthesized, to verify the validity of the proposed method. By evolution, the quality of the optimized image filter on

reducing salt-and-pepper noise is better than that of other papers. Consequently the proposed design method is effective for mixed constrained image filter design on reducing salt-and-pepper noise.

Chapter 5 proposed the use of PSO (Particle Swarm Optimization) for mixed constrained image filter design for noise reduction. The complexity, power and signal delay both of the CLBs and wires are considered. An image filter for removing noise is experimentally synthesized using PSO, to verify the validity of our method. By evolution, quality of the optimized image filter by PSO(0.9) is almost same as that of GA, but the running time of PSO is 10% shorter than that of GA.

Chapter 6 described mixed constrained image filter design with fault tolerance for noise reduction using GA on a reconfigurable processing array. The complexity, power and signal delay in both CLBs and wires are considered. An image filter for noise reduction is experimentally synthesized, to verify the validity of our method. By evolution, the quality of the optimized image filter on a reconfigurable processing array with a few faults is almost same as that on a reconfigurable processing array with no fault. Consequently our proposed design method is effective for fault-tolerant optimization.

7.2 Future work

We will also apply evolutionary technologies to autonomous design circuits for more complex functional requirements, and enhance more practical information about circuit to fitness function. In the future, we will develop the adaptive systems which reconfigure an existing design by evolutionary technologies to adapt to a variable operational environment.

ACKNOWLEDGMENTS

First, I would like to thank my supervisor, Professor Takahiro Watanabe, for his constructive instruction and helpful discussion in my research. Also, I want to express my sincere acknowledgement to his support for my Doctor thesis research.

I give my acknowledgement to Professor Takeshi Yoshimura, Professor Shinji Kimura and Professor Hiroshi Miyashita for their suggestion in my research topic.

I show my thankfulness to Professor Kotaro Hirasawa for his valuable advices and discussion for my paper review and research selection. Also, I study many knowledge in evolutionary algorithms by his generous guidance.

I give my acknowledgement to Professor Satoshi Goto for his financial support for my Doctor thesis research.

I also thank Mr. Keyan Chen, Mr. Dawei Cao, Mr. Yuehui Shi, Mrs. Meiyan Li, Mrs. Fangfang Wang, Mrs. Yiwen Su, Mrs. Xiaoming Zhao and all members of our laboratory, for their kind help and nice advices in my laboratory study and daily life.

I want to thank the excellent research collaboration with Prof. Jinglu Hu, Dr. Shingo Mabu, Dr. Yun Yang, Dr. Song Chen, Dr. Zhangcai Huang, Dr. Jin Zhou, Dr. Guangfei Yang, Dr. Shuming Wang, Dr. Benhui Chen, Dr. Lu Yu, Dr. Boyang Li, Dr. Xinjie Yu, Dr. Feng Wen, Mr. Jun Wang, Mr. Yang Chen and Mr. Wenqiang Zhang. Their discussions are fruitful and necessary for my research and study. Their friendly relationship gives me happy research environment and I enjoy all friends in Waseda University with me.

Finally, I would like to dedicate this thesis to my loving parents, Mr. Shuntong Bao and Mrs. Linghuan Zhang, for their helpful advice and durative support.

BIBLIOGRAPHY

[1] Kenneth A. De Jong. Evolutionary computation - a unified approach. In *Genetic Programming and Evolvable Machines*, volume 8(3), pages 293–295. MIT Press, Springer, 2007.

[2] A. P. Alves da Silva and P. J. Abrao. Applications of evolutionary computation in electric power systems. In *Proc. of the 2002 Congress on Evolutionary Computation (CEC 2002)*, pages 1057 – 1062, Honolulu, HI, May 2002.

[3] E. Eberbach. On expressiveness of evolutionary computation: is ec algorithmic? In *Proc. of the 2002 Congress on Evolutionary Computation (CEC 2002)*, pages 564 – 569, Honolulu, HI, May 2002.

[4] T. Back. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford Univ. Press, USA, Jan. 1996.

[5] D. Ashlock. *Evolutionary Computation for Modeling and Optimization*. Springer, Dec. 2005.

[6] T. Schnier and Xin Yao. Using multiple representations in evolutionary algorithms. In *Proc. of the 2000 Congress on Evolutionary Computation*, volume 1, pages 479 – 486, La Jolla, CA, July 2000.

[7] John Henry Holland. *Adaptation in Natural and Artificial systems*. University of Michigan Press, Dec. 1975.

[8] Y. Chen, J. Hu, K. Hirasawa, and S. Yu. Solving deceptive problems using a genectic algorithm with reserve selection. In *Proc. IEEE Congress on Evolutionary Computation 2008 (CEC 2008)*, pages 884–889, Hongkong, June 2008.

[9] Gerardo Beni and Jing Wang. Swarm intelligence in cellular robotic systems. In *Proc. NATO Advanced Workshop on Robots and Biological Systems*, Italy, June 1989.

[10] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[11] Russell C. Eberhart and Yuhui Shi. Comparison between genetic algorithms and particle swarm optimization. In *EP '98: Proceedings of the 7th International Conference on Evolutionary Programming VII*, pages 611–616, London, UK, 1998. Springer-Verlag.

[12] Jacob Robinson and Yahya Rahmat-Samii. Particle swarm optimization in electromagnetics. *IEEE Transactions on Antennas and Propagation*, 52(2):397–407, Feb. 2004.

[13] T. Arslan, D. H. Horrocks, and E. Ozdemir. Structural cell-based vlsi circuit design using a genetic algorithm. In *Proc. 1996 IEEE International Symposium On Circuits And Systems*, pages 308–311, Atlanta, Georgia, USA, May 1996.

[14] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti. Toward robust integrated circuits: The embryonics approach. *Proc. the IEEE*, 88(4):516–541, April 2000.

[15] X. Zhang, G. Dragffy, A. G. Pipe, and Q. M. Zhu. Ontogenetic cellular hardware for fault tolerant systems. In *Proc. International Conference on Embedded Systems and Applications 2003 (ESA 2003)*, pages 144–150, Las Vegas, USA, June 2003.

[16] J. D. Lohn and G. S. Hornby. Evolvable hardware: using evolutionary computation to design and optimize hardware systems. *IEEE Computational Intelligence Magazine*, 1(1):19–27, Feb. 2006.

[17] E. Stomeo, T. Kalganova, and C. Lambert. A novel genetic algorithm for evolvable hardware. In *Proc. IEEE Congress on Evolutionary Computation 2006 (CEC 2006)*, pages 134–141, Canada, July 2006.

[18] A. Stoica and R. Andrei. Adaptive and evolvable hardware - a multifaceted analysis. In *Proc. Second NASA/ESA Conference on Adaptive Hardware and Systems 2007 (AHS 2007)*, pages 486–498, Edinburgh, UK, Aug. 2007.

[19] E. Benkhelifa, A. Pipe, G. Dragffy, and M. Nibouche. Towards evolving fault tolerant biologically inspired hardware using evolutionary algorithms. In *Proc. IEEE Congress on Evolutionary Computation 2007 (CEC 2007)*, pages 1548–1554, Singapore, Sept. 2007.

[20] W. M. Spears and K. A. De Jong. On the virtues of parameterized uniform crossover. In *Proc. the Fourth International Conference on Genetic Algorithms*, 1991.

[21] T. Higuchi, M. Iwata, I. Kajitani, et al. Evolvable hardware with genetic learning. *Proc. IEEE International Symposium on Circuits and Systems 1996 (ISCAS '96)*, 4:29–32, May 1996.

[22] J. F. Miller, P. Thomson, and T. Fogarty. Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study. In D. Quagliarella, J. Periaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, chapter 6, pages 105–131. John Wiley & Sons Ltd., Morgan Kaufmann, Chichester, England, 1997.

[23] A.Thompson. Silicon evolution. In J. R. Koza et al., editors, *Proc. Genetic Programming 1996 (GP '96)*, pages 444–452. MIT Press, 1996.

[24] D. J. Xu and M. L. Daley. Design of optimal digital filter using a parallel genetic algorithm. *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, pages 673–675, Oct. 1995.

[25] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, N. Salami, N. Kajihara, and N. Otsu. Real-world applications of analog and digital evolvable hardware. *IEEE Trans. on Evolutionary Computation*, pages 220–235, Sept. 1999.

[26] A. Thompson, P. Layzell, and R. S. Zebulum. Explorations in design space: unconventional electronics designthrough artificial evolution. *IEEE Trans. on Evolutionary Computation*, pages 167–196, Sept. 1999.

[27] V. K. Vassilev, D. Job, and J. F. Miller. Towards the automatic design of more efficient digital circuits. In *Proc. The Second NASA/DoD Workshop on Evolvable Hardware 2000 (EH 2000)*, pages 151–160, Pao Alto, CA, USA, Jul. 2000.

[28] J. Torresen, J. W. Bakke, and L. Sekanina. Recognizing speed limit sign numbers by evolvable hardware. In *Proc. Parallel Problem Solving from Nature - PPSN VIII*, pages 682–691, 2004.

[29] Y. Zhang, S. L. Smith, and A. M. Tyrrell. Digital circuit design using intrinsic evolvable hardware. In *Proc. NASA/DoD Conference on Evolvable Hardware 2004 (EH 2004)*, pages 55–62, June 2004.

[30] L. Sekanina. Evolutionary design of gate-level polymorphic digital circuits. In *Proc. Applications on Evolutionary Computing*, pages 185–194, 2005.

[31] X. Yao and T. Higuchi. Promises and challenges of evolvable hardware. *IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 29(1):87–97, Feb. 1999.

[32] V. K. Vassilev and J. E. Miller. Scalability problems of digital circuit evolution evolvability and efficient designs. In *Proc. The Second NASA/DoD Workshop on Evolvable Hardware 2000 (EH 2000)*, pages 55–64, Palo Alto, CA, USA, July 2000.

[33] L. Sekanina. Evolutionary design of digital circuits: Where are current limits? In *Proc. The First NASA/ESA Conference on Adaptive Hardware and Systems 2006 (AHS 2006)*, pages 171–178, June 2006.

[34] Z. Bao and T. Watanabe. A new approach for circuit design optimization using genetic algorithm. In *Proc. International SoC Design Conference 2008 (ISOCC 2008)*, pages 383–386, Busan, Korea, Nov. 2008.

[35] Z. Bao and T. Watanabe. A novel genetic algorithm with cell crossover for circuit design optimization. In *Proc. IEEE International Symposium on Circuits and Systems 2009 (ISCAS 2009)*, pages 2982–2985, Taipei, Taiwan, China, May 2009.

[36] T. Yalcinoz and H. Altun. A new genetic algorithm with arithmetic crossover to economic and environmental economic dispatch. *Engineering intelligent systems for electrical engineering and communications*, 13(1):45–52, 2005.

[37] F. Herrera and M. Lozano. Heuristic crossovers for real-coded genetic algorithms based on fuzzy connectives. In *Proc. of the 4th International Conference on Parallel Problem Solving from Nature*, pages 336 – 345, 1996.

[38] J. Dumoulin, J. Foster, J. Frenzel, and S. McGrew. *Real-World Applications of Evolutionary Computing*, volume 1803 of *Lecture Notes in Computer Science*, chapter Special Purpose Image Convolution with Evolvable Hardware, pages 111–125. Springer, 2000.

[39] L. Sekanina. *Applications of Evolutionary Computing, (the 4th Workshop on Evolutionary Computation in Image Analysis and Signal Processing, EvoIASP 2002)*, volume 2279 of *Lecture Notes in Computer Science*, chapter Image Filter Design with Evolvable Hardware, pages 255–266. Springer, 2002.

[40] L. Sekanina. *Evolvable Components: From Theory to Hardware Implementations*. Natural Computing Series. Springer, January 2004.

[41] T. Martinek and L. Sekanina. An evolvable image filter - experimental evaluation of a complete hardware implementation in fpga. In *Evolvable Systems - From Biology to Hardware*, volume 2005(3637), pages 76–85. Lecture Notes in Computer Science, Springer, 2005.

[42] Z. Vasicek and L. Sekanina. Evaluation of a new platform for image filter evolution. In *Proc. the Second NASA/ESA Conference on Adaptive Hardware and Systems, 2007 (AHS 2007)*, pages 577–586, Scotland, United Kingdom, Aug. 2007.

[43] Z. Vasicek and L. Sekanina. An evolvable hardware system in xilinx virtex ii pro fpga. *Int. J. Innovative Computing and Applications*, 1(1):63–73, 2007.

[44] Z. Bao and T. Watanabe. Circuit design optimization using genetic algorithm with parameterized uniform crossover. *IEICE Trans. on Fundamentals*, E93-A(01):281–290, Jan. 2010.

[45] Z. Bao and T. Watanabe. A novel ga with multi-level evolution for mixed constrained circuit design optimization. In *Proc. 2009 RISP International Workshop on Nonlinear Circuits and Signal Processing (NCSP '09)*, pages 411–414, Honolulu, Hawaii, USA, March 2009.

[46] Z. Bao and T. Watanabe. A novel genetic algorithm with cell crossover for circuit design optimization. In *Proc. IEEE International Symposium on Circuits and Systems 2009 (ISCAS 2009)*, pages 2982–2985, Taipei, Taiwan, May 2009.

[47] Z. Bao and T. Watanabe. Evolutionary design for image filter using ga. In *Proc. IEEE TENCON 2009*, pages 1–6, Singapore, Nov. 2009.

[48] Z. Bao and T. Watanabe. Mixed constrained image filter design using particle swarm optimization. In *Proc. The Fifteenth International Symposium on Artificial Life and Robotics 2010 (AROB 15th 2010)*, pages 230–235, Beppu, Oita, Japan, Feb. 2010.

[49] B. Rajan and S.Ravi. Fpga based hardware implementation of image filter with dynamic reconfiguration architecture. *IJCSNS International Journal of Computer Science and Network Security*, 6(12):121–127, Dec. 2006.

[50] International Technology Roadmap for Semiconductors. *Executive Summary*, international roadmap committee edition, 2009. http://www.itrs.net/Links/2009ITRS/Home2009.htm.

[51] M. Hartmann and P. C. Haddow. Evolution of fault-tolerant and noise-robust digital designs. *IEE Proc. Computers and Digital Techniques*, 51(4):287–294, July 2004.

[52] Z. Bao, F. Wang, X. Zhao, and T. Watanabe. Fault-tolerant image filter design using ga. In *Proc. IEEE TENCON 2010*, pages 1–6, Fukuoka, Japan, Nov. 2010.

[53] T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, and T. Furuya. Evolving hardware with genetic learning: a first step towards building a darwin machine. In *Proc. the Second International Conference on Simulated Adaptive Behaviour*, pages 417–424. MIT Press, 1993.

[54] H. de Garis. Evolvable hardware: Genetic programming of a darwin machine. In Rudolf F. Albrecht, Nigel C. Steele, and Colin R. Reeves, editors, *Proc. International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 441–449, Innsbruck, Austria, 1993. Springer.

[55] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, N. Salami, N. Kajihara, and N. Otsu. Real-world applications of analog and digital evolvable hardware. *IEEE Trans. on Evolutionary Computation*, 3(3):220–235, Sept. 1999.

[56] X. Yao and T. Higuchi. Promises and challenges of evolvable hardware. *IEEE Trans. on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 29(1):87–97, Feb. 1999.

[57] L. Sekanina. Evolutionary design of digital circuits: Where are current limits? In *Proc. The First NASA/ESA Conference on Adaptive Hardware and Systems 2006 (AHS 2006)*, pages 171–178, Piscataway, US, June 2006. IEEE Computer Society.

PUBLICATIONS

Journal Papers

1. **Zhiguo Bao**, Fangfang Wang, Xiaoming Zhao and Takahiro Watanabe, "Mixed Constrained Image Filter Design for Salt-and-pepper Noise Reduction using Genetic Algorithm," *IEEJ Trans. EIS*, Vol.131, No.3, pp. 363-368, Mar. 2011.

2. **Zhiguo Bao** and Takahiro Watanabe, "Circuit Design Optimization Using Genetic Algorithm with Parameterized Uniform Crossover," *IEICE Trans. on Fundamentals*, Vol.E93-A, No.01, pp. 281-290, Jan. 2010.

3. **Zhiguo Bao** and Takahiro Watanabe, "Mixed constrained image filter design using particle swarm optimization," *Journal of Artificial Life and Robotics*, Vol. 15, No. 3, pp. 363-368, 2010.

4. **Zhiguo Bao** and Takahiro Watanabe, "A novel genetic algorithm with different structure selection for circuit design optimization," *Journal of Artificial Life and Robotics*, Vol. 14, No. 2, pp.266-270, 2009.

5. Etsushi Ohkawa, Yan Chen, **Zhiguo Bao**, Shingo Mabu, Kaoru Shimada and Kotaro Hirasawa, "Buying and Selling Stocks of Multi Brands Using Genetic Network Programming with Control Nodes," IEEJ Trans. EIS, Vol. 128, No. 12, pp.1811-1819, 2008.

International Conference Papers

6. **Zhiguo Bao**, Fangfang Wang, Xiaoming Zhao and Takahiro Watanabe, "Fault-tolerant Image Filter Design using Particle Swarm Optimization," Proc. The Sixteenth International Symposium on Artificial Life and Robotics 2011 (AROB 16th 2011), pp. 653-658, Beppu,Oita, Japan, Jan. 2011.

7. **Zhiguo Bao**, Fangfang Wang, Xiaoming Zhao and Takahiro Watanabe, "Fault-tolerant Image Filter Design using GA," *Proc. IEEE TENCON 2010*, pp. 1-6, Fukuoka, Japan, Nov. 2010.

8. **Zhiguo Bao** and Takahiro Watanabe, "Mixed Constrained Image Filter Design Using Particle Swarm Optimization," Proc. AROB 15th 2010, pp. 230-235, Beppu, Oita, Japan, Feb. 2010.

9. **Zhiguo Bao** and Takahiro Watanabe, "Evolutionary Design for Image Filter using GA," *Proc. IEEE TENCON 2009*, pp. 1-6, Singapore, Nov. 2009.

10. **Zhiguo Bao** and Takahiro Watanabe, "A Novel Genetic Algorithm with Cell Crossover for Circuit Design Optimization," *Proc. The IEEE International Symposium on Circuits and Systems 2009 (ISCAS 2009)*, pp.2982-2985, Taipei, Taiwan, May 2009.

11. **Zhiguo Bao** and Takahiro Watanabe, "A Novel GA with multi-level evolution for Mixed constrained Circuit Design Optimization," *Proc. 2009 RISP International Workshop on Nonlinear Circuits and Signal Processing (NCSP '09)*, pp. 411-414, Honolulu, Hawaii, USA, Mar. 2009.

12. **Zhiguo Bao** and Takahiro Watanabe, "A Novel Genetic Algorithm with Different Structure Selection for Circuit Design Optimization," Proc. AROB 14th 2009, pp. 218-222, Beppu, Oita, Japan, Feb. 2009.

13. **Zhiguo Bao** and Takahiro Watanabe, "A New Approach for Circuit Design Optimization using Genetic Algorithm," *Proc. International SoC Design Conference (ISOCC) 2008*, pp. 383-386, Busan, Korea, Nov. 2008.

14. **Zhiguo Bao**, Shigo Mabu, Kotaro Hirasawa and Jinglu Hu, "Buying and Selling Stocks of Multi Brands using Genetic Network Programming with control nodes," Proc. SICE (The Society of Instrument and Control Engineers) Annual Conference 2007, pp. 1569-1576, Kagawa, Japan, Sept. 2007.

Domestic Conference Papers

15. YiWen Su, **Zhiguo Bao**, Kuoyang Tu and Takahiro Watanabe, "Circuit Design Using Genetic Algorithm combined with Taguchi method and Particle Swarm Optimization," The 63rd Joint Conference of Electrical and Electronics Engineers in Kyushu, pp. 115-116, Fukuoka, Japan, Sep. 2010.

16. Shi Yuehui, **Zhiguo Bao**, Wang Yang, Xiao Zuojun and Watanabe Takahiro, "P/G Network Design to Optimize Area, Performance and Power Consumption," The 62nd Joint Conference of Electrical and Electoronics Engineers in Kyushu (2009/09), pp. 10-1A-14, Fukuoka, Japan, Sep. 2009.

INDEX