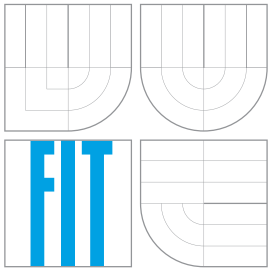# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

METODY AKCELERACE EVOLUČNÍHO NÁVRHU ČÍSLICOVÝCH OBVODŮ
ACCELERATION METHODS FOR EVOLUTIONARY DESIGN OF DIGITAL CIRCUITS

DISERTAČNÍ PRÁCE
PHD THESIS

AUTOR PRÁCE   Ing. ZDENĚK VAŠÍČEK
AUTHOR

VEDOUCÍ PRÁCE   Prof. Ing. LUKÁŠ SEKANINA, Ph.D.
SUPERVISOR

BRNO 2012

Abstrakt

Ačkoliv můžeme v literatuře nalézt řadu příkladů prezentujících evoluční návrh jakožto zajímavou a slibnou alternativu k tradičním návrhovým technikám používaným v oblasti číslicových obvodů, praktické nasazení je často problematické zejména v důsledku tzv. problému škálovatelnosti, který se projevuje např. tak, že evoluční algoritmus je schopen poskytovat uspokojivé výsledky pouze pro malé instance řešeného problému. Vážný problém představuje tzv. problém škálovatelnosti evaluace fitness funkce, který je markantní zejména v oblasti syntézy kombinačních obvodů, kde doba potřebná pro ohodnocení kandidátního řešení typicky roste exponenciálně se zvyšujícím se počtem primárních vstupů.

Tato disertační práce se zabývá návrhem několika metod umožňujících redukovat problém škálovatelnosti evaluace v oblasti evolučního návrhu a optimalizace číslicových systémů. Cílem je pomocí několika případových studií ukázat, že s využitím vhodných akceleračních technik jsou evoluční techniky schopny automaticky navrhovat inovativní/kompetitivní řešení praktických problémů.

Aby bylo možné redukovat problém škálovatelnosti v oblasti evolučního návrhu číslicových filtrů, byl navržen doménově specifický akcelerátor na bázi FPGA. Tato problematika reprezentuje případ, kdy je nutné ohodnotit velké množství trénovacích dat a současně provést mnoho generací. Pomocí navrženého akcelerátoru se podařilo objevit efektivní implementace různých nelineárních obrazových filtrů. S využitím evolučně navržených filtrů byl vytvořen robustní nelineární filtr implusního šumu, který je chráněn užitným vzorem. Navržený filtr vykazuje v porovnání s konvenčními řešeními vysokou kvalitu filtrace a nízkou implementační cenu.

Spojením evolučního návrhu a technik známých z oblasti formální verifikace se podařilo vytvořit systém umožňující výrazně redukovat problém škálovatelnosti evoluční syntézy kombinačních obvodů na úrovni hradel. Navržená metoda dovoluje produkovat komplexní a přesto kvalitní řešení, která jsou schopna konkurovat komerčním nástrojům pro logickou syntézu. Navržený algoritmus byl experimentálně ověřen na sadě několika benchmarkových obvodů včetně tzv. obtížně syntetizovatelných obvodů, kde dosahoval v průměru o 25% lepších výsledků než dostupné akademické i komerční nástroje.

Poslední doménou, kterou se práce zabývá, je akcelerace evolučního návrhu lineárních systémů. Na příkladu evolučního návrhu násobiček s vícenásobnými konstantními koeficienty bylo ukázáno, že čas potřebný k evaluaci kandidátního řešení lze výrazně redukovat (defacto na ohodocení jediného testovacího vektoru), je-li brán v potaz charakter řešeného problému (v tomto případě linearita).

Klíčová slova

návrh číslicových obvodů, evoluční optimalizace, evoluční návrh, násobička s konstantními koeficienty, filtrace obrazu, nelineární filtr, optimalizace kombinačních obvodů, FPGA akcelerace

Citace

Abstract

Although many examples showing the merits of evolutionary design over conventional design techniques utilized in the field of digital circuits design have been published, the evolutionary approaches are usually hardly applicable in practice due to the various so-called scalability problems. The scalability problem represents a general problem that refers to a situation in which the evolutionary algorithm is able to provide a solution to a small problem instances only. For example, the scalability of evaluation of a candidate digital circuit represents a serious issue because the time needed to evaluate a candidate solution grows exponentially with the increasing number of primary inputs.

In this thesis, the scalability problem of evaluation of a candidate digital circuit is addressed. Three different approaches to overcoming this problem are proposed. Our goal is to demonstrate that the evolutionary design approach can produce interesting and human competitive solutions when the problem of scalability is reduced and thus a sufficient number of generations can be utilized.

In order to increase the performance of the evolutionary design of image filters, a domain specific FPGA-based accelerator has been designed. The evolutionary design of image filters is a kind of regression problem which requires to evaluate a large number of training vectors as well as generations in order to find a satisfactory solution. By means of the proposed FPGA accelerator, very efficient nonlinear image filters have been discovered. One of the discovered implementations of an impulse noise filter consisting of four evolutionary designed filters is protected by the Czech utility model.

A different approach has been introduced in the area of logic synthesis. A method combining formal verification techniques with evolutionary design that allows a significant acceleration of the fitness evaluation procedure was proposed. The proposed system can produce complex and simultaneously innovative designs, overcoming thus the major bottleneck of the evolutionary synthesis at gate level. The proposed method has been evaluated using a set of benchmark circuits and compared with conventional academia as well as commercial synthesis tools. In comparison with the conventional synthesis tools, the average improvement in terms of the number of gates provided by our system is approximately 25%.

Finally, the problem of the multiple constant multiplier design, which belongs to the class of problems where a candidate solution can be perfectly evaluated in a short time, has been investigated. We have demonstrated that there exists a class of circuits that can be evaluated efficiently if a domain knowledge is utilized (in this case the linearity of components).

Keywords

Bibliographic citation

Acceleration Methods for Evolutionary Design of Digital Circuits

Prohlášení

Prohlašuji, že jsem tuto disertační práci vypracoval samostatně pod vedením Prof. Ing. Lukáše Sekaniny, Ph.D., a že jsem uvedl všechny literární prameny, ze kterých jsem v průběhu své práce čerpal.

<div align="right">

. . . .

Zdeněk Vašíček

9. března 2012

</div>

Poděkování

Na tomto místě bych rád poděkoval všem, kteří přispěli k tomu, že tato práce vznikla. Především mému vedoucímu disertační práce Prof. Ing. Lukáši Sekaninovi, Ph.D. za řadu podnětných diskuzí týkajících se tématu disertační práce, metodické vedení a spolupráci při výzkumu. Dále děkuji všem spolupracovníkům za vstřícnost a spolupráci při řešení jednotlivých částí disertační práce. V neposlední řadě bych rád poděkoval svým rodičům, ženě a všem blízkým za podporu během studia i psaní této práce.

Contents

Chapter 1

Introduction

The electronic manufacturing industry, especially electronic circuit production, is an area that has gone through a substantial development in the recent fifty years. In the second half of the 20th century, innovations in electronic computer systems made the personal computer a reality. Each new generation of computers was cheaper to purchase, more powerful and easier to operate. Thus the computers shortly became universal computing machines that spread not only among the scientific community but also among the common users. The progress achieved by the 21st century causes the electronic products had transformed the way that people live, work, and communicate. A common cellular phone has been superseded with the devices having the performance comparable with the personal computers and the personal computers are gradually replaced with very popular portable devices.

Comparing the current requirements to the requirements formulated a few years ago, significantly more complex circuits and behaviors are demanded today. This demand is caused by the relentless improvements of the available technologies. While the current advance is driven mainly by the necessity to minimize the overall power consumption of the produced systems, in the 1990s the goal was a relative simple – doubling of the performance of the computer systems and keeping up with the Moore's law. The current situation is much complicated and requires discovering and applying new approaches as the power consumption requirements are generally in contrast with the performance requirements.

One of the main bottlenecks that has been identified by scientific community is a low efficiency of circuit design [54]. Traditional circuit design methodologies rely on rules and design techniques that have been developed over many decades. However, the need for human input to the increasingly complex design process means that the circuit design has to be simplified by imposing greater and greater abstraction to the design space. An example of this approach is the introduction of the hardware description languages. This abstraction allows designers to significantly reduce the time needed to design and produce the intended system. On the other hand, it also results in waste of potential circuit behavior since the conventional design methodologies do not offer too many ways to benefit from the physical dynamics available from the silicon medium [175].

One of crucial parts of the design process is the efficient logic synthesis and optimization. As a part of computer theory, the logic synthesis and optimization have been developed for more than 50 years. Despite the fact that the logic synthesis and optimization are

considered to be very difficult problems, many companies provide commercial tools that allow processing even the systems of the contemporary complexity in a reasonable time. However, the recent work in the area of conventional synthesis has shown that the available synthesis algorithms produce solutions that are far from optimum for many circuit classes [35].

In the beginning of nineties, a new field applying evolutionary techniques to hardware design and synthesis has been established. This field is referred to as Evolvable Hardware [65]. The evolvable hardware draws inspiration from three main fields – biology, computer science and electronic engineering. The aim is to provide (1) electronic systems exhibiting a degree of self-adaptive and self-repair behavior and/or (2) a robust design approach that could even replace a human designer in some cases. Typical application domains include design of digital circuits, analog circuits, antennas, optical systems and MEMS [107, 78, 83].

In the context of the circuit design, the evolvable hardware is very attractive approach as it provides another option to the traditional design methodology – to use evolution to design circuits for us. Moreover, the key strength of the evolvable hardware approach is that it can be applied for designing of the circuits that cannot be fully specified a priori, but where the desired behavior is known. In fact, the search-based approaches seem to be the only viable option in this case. Another often emphasized advantage of this approach is that the circuits can be adopted for a particular environment.

During the last two decades, the evolvable hardware community has demonstrated that very efficient (and sometimes also patentable) implementations of physical designs can be obtained using evolutionary computation. For example, John Koza, the pioneer of the field, dealing primarily with the evolutionary design of analog circuits, has reported tens of human-competitive results in various areas of science and technology. The results were obtained automatically using evolutionary techniques, in particular using genetic programming [102] that has mainly been adopted for analog circuit design [121, 38]. In case of digital logic synthesis, the evolutionary synthesis has also led to several innovative designs [127, 9, 164]; however the obtained results belong to the category of relatively small circuits.

Although the evolutionary design has been shown to be a promising and general-purpose design method, there exist several problems that make the evolutionary approach problematic in some applications [69]. The scalability problem has been identified as one of the most difficult problems the researchers are faced with in the evolvable hardware field. The scalability problem means such situation in which the evolutionary algorithm is able to provide a solution to a small problem instance; however, only unsatisfactory or even none solutions can be obtained for larger problem instances in a reasonable time. Another problem related to this issue is enormous computational power which evolutionary algorithms usually need for obtaining innovative results for some applications.

The scalability problem can primarily be seen from two perspectives: scalability of representation and scalability of fitness evaluation. From the viewpoint of the scalability of representation, the problem is that long chromosomes (a set of genes which defines a candidate solution) which are usually required to represent complex solutions imply large search spaces that are typically difficult to search. Another issue is the scalability of fitness evaluation, i.e. the problem that complex candidate solutions might require a lot of time to be evaluated. For example, in the case of the evolutionary design of combinational circuits,

the evaluation time of a candidate circuit grows exponentially with the increasing number of inputs (assuming that 2^n test vectors are generated for n-input circuit). This represents the main weakness of the evolutionary approach. It also causes that real-world applications of evolutionary circuit design are not able to compete with conventional design.

1.1 Goals of the Thesis

It will be argued in this thesis that the fitness scalability issue can be eliminated by seeking for new sophisticated evaluation methods. We will solely deal with evolvable hardware as the method for automated design, i.e. the scenario in which the evolutionary algorithm is used only in the design phase of a product. The thesis postulates two main objectives.

The **first goal** is to propose problem-specific methods that will allow designers to reduce the scalability problem in the area of digital system design. As the scalability problem represents a general problem, we will consider only a very narrow but important subarea – the scalability of evaluation of a candidate digital circuit.

The **second goal** is to evaluate the impact of the proposed methods and show that by means of the proposed methods it is possible to evolve innovative solutions in various problem domains. In the context of evolutionary circuit design, we mean by the term innovative that a solution exhibits better features with respect to existing designs of the same category.

Thesis Organization

The thesis is organized as follows. The first two chapters contain theoretical background that outlines the basic concepts and ideas utilized in the following chapters. In addition to that, this introductory part also clarifies the motivation of this work. The next three chapters contain three case studies that demonstrate three approaches to elimination of the scalability problem of evolutionary circuit design. Finally, an evolutionary platform designed to accelerate the evolutionary design of digital circuits is introduced. To be more specific:

Chapter 2 provides the necessary background of evolvable hardware which represents the essential concept tightly connected with this thesis. This overview covers the principles and basic concepts of evolutionary algorithms. The chapter is divided into three sections. The first section contains a description of relevant evolutionary techniques, especially Cartesian Genetic Programming that have been utilized in the experiments. The next section summarizes reconfigurable devices that have been used in the evolvable hardware field. The last section comprises of a literature survey of evolvable hardware which represents the research area in which the presented thesis belongs to.

Chapter 3 is devoted to the evolutionary design of digital and analog circuits. The first two sections contain a summary of the electronic circuits designed at various levels of abstraction that have been published in literature. The goal of this chapter is to make an insight to the complexity of the design problems that have been solved so far. The next section discusses the practical aspects of the evolutionary design of digital circuits

3

by means of Cartesian Genetic Programming. In the third section, the shortcomings and bottlenecks of the evolutionary design are discussed. This part deals with the issue of scalability of evolutionary design and the approaches that have been proposed to mitigate or even remove various scalability problems.

The first case study, which is presented in Chapter 4, deals with the evolutionary design of linear transforms. We have identified a class of problems for which a candidate solution can be perfectly evaluated in a very short time. This chapter is divided into four sections. The first section covers the theoretical background related to the linear transforms in general, and multiple constant multiplier blocks in particular. The next three sections contain the description of the proposed method and experimental evaluation of this method.

In Chapter 5, the second case study related to the evolutionary synthesis of complex digital circuits is introduced. The goal of this chapter is to present a new approach to the fitness function implementation which is based on a formal verification algorithm. The proposed method significantly eliminates the scalability problem of fitness function evaluation which has been known from the very beginning of digital evolvable hardware. This part is divided into five sections. The first section describes the problem of combinational equivalence checking and the process of conventional logic synthesis. The proposed method followed by its extensive experimental evaluation is described in the second and third section respectively. The fourth and fifth section describe the improved version of the proposed approach and its evaluation using a set of real-world benchmark circuits.

The case study devoted to the evolutionary design of nonlinear image filters is presented in Chapter 6. This chapter consists of three sections. The first section defines the problem to be solved and introduces the necessary theoretical background connected with the filtration of impulse noise. The second section discusses the evolutionary design of image filters using Cartesian Genetic Programming. Finally, experimental results are summarized in the last section.

A common feature of Chapters 4–6 is that firstly the discussed problem is introduced. Afterwards, the proposed evolutionary design approach followed by experimental evaluation is given. Finally, the obtained results are presented and summarized. The introduction includes not only the description of the problem, but also an overview of the best-known conventional methods that are usually utilized to solve a given problem.

Chapter 7 describes a new hardware accelerator for Cartesian Genetic Programming implemented using FPGA. Two types of application-specific accelerators are in fact proposed. The first one is devoted for symbolic regression problems over the fixed point representation. The second one is designed for evolution of logic circuits.

Chapter 8 summarizes the results obtained in this thesis and outlines directions for the future research.

Chapter 2

From Evolutionary Algorithms to Evolvable Hardware

The purpose of this chapter is briefly introduce the key concepts behind the evolutionary algorithms, reconfigurable devices and evolvable hardware.

2.1 Evolutionary Algorithms

Several decades ago, researchers started to explore how some ideas taken from nature could be employed for solving hard computing problems. Evolutionary algorithms inspired by biological evolution represent one of the most successful examples.

The *evolutionary algorithms* (EAs) [11] are stochastic search algorithms inspired by Darwin's theory of evolution. The common feature of evolutionary algorithms is that they utilize mechanisms that are inspired by principles of biological evolution, namely reproduction, mutation, recombination and selection. In contrast with common search algorithms, such as random search or hill climbing, the EAs are population-based algorithms. It means that they work with more candidate solutions (i.e. individuals) in the same time. By a candidate solution we mean a point in the search space, the space that contains all possible considered solutions to a given problem.

Every new population is formed using genetic operators such crossover and mutation and through a selection pressure. The selection pressure together with the *fitness function*, sometime referred to as objective function, is responsible for guiding the evolution towards better areas of the search space. The guidance is received from the fitness function that assigns so called fitness value to each candidate solution. The fitness value indicates how well a candidate solution fulfills the problem objective; in other words, it indicates how a particular candidate solution meets the specification. A better fitness value implies a greater chance that a candidate solution will remain for a longer while and produce offspring, which inherit parental genetic information. A well-designed evolutionary algorithm should converge to a population containing desired solutions.

Each member of the population (i.e. a candidate solution) consists of a string of parameters, so called genes. This string is usually referred to as an individual or chromosome. A particular value of a gene in chromosome is called allele. Thus, the alleles are the small-

est information units in a chromosome. In nature, alleles exist pair wise, whereas in the evolutionary algorithms, an allele is usually represented by only one symbol. Because the objects in the search space can generally represent arbitrary structure (a vector of real values, digital circuit, antenna, and so on), we distinguish between a search space (or genotype space) and representation space (phenotype space). While the fitness function is applied to evaluate phenotypes, the genetic operators manipulate with genotypes. A small change in the genotype should produce a small change in the phenotype otherwise the evolutionary algorithm is not efficient [15].

The evolutionary algorithms are traditionally divided into four distinct branches: *genetic algorithm* [62], *genetic programming* [100], *evolutionary strategies* [156] and *evolutionary programming* [55]. The algorithms mainly differ in the mechanism of the candidate solutions encoding, implementation of the evolutionary operators applied to the candidate solutions and finally, utilized search strategy that guides the EA through the search space.

2.1.1 Genetic Algorithms

Genetic algorithm (GA) was introduced by John Holland in 1973 and made famous by David Goldberg [62, 82].

Researchers have proposed many different variants of genetic algorithms in the literature. For the illustration, we will use the traditional simple genetic algorithm (the simplest form of GA) defined by Goldberg. This canonical algorithm uses two genetic operators, crossover as the main operator and mutation which serves only as background noise. The structure of a canonical genetic algorithm, as it has been described by David Goldberg in [62], is captured in the following pseudo code.

```
set time t = 0
randomly create initial population P(t)
while (termination condition is false)
     evaluate each individual in P(t)
     if  acceptable solution found then
         break
     reproduce individuals according to their fitnesses into mating pool
     (higher fitness implies more copies of individual in mating pool)
     t = t + 1
     while P(t) is not filled with new offspring do
         randomly take two individuals from mating pool
         use probabilistic random crossover to generate two offspring
         apply probabilistic random mutation to the offspring
         place offspring into population P(t)
```

Algorithm 2.1: Canonical Genetic Algorithm

The simple genetic algorithm uses a population of individuals having the constant size. The individuals are encoded using a vector (or string) of fixed size. GA traditionally operates with binary, integer, real-valued or character-based string. The genetic operators such

as one-point, uniform or n-point crossover are directly applied to the genotypes. In many implementations, crossover produces two new offspring from two parents by exchanging substrings. The mutation operator slightly changes the genotype of an individual.

The basic functionality of a traditional simple GA is relatively simple. After randomly creating and evaluating an initial population, the algorithm iteratively creates new generations. New generations are created by recombining the selected highly fit individuals using a crossover operator and applying mutation to the obtained offspring. Crossover is often used about 70% of the time to generate offspring, for the remaining 30% offspring are simply clones of their parents. Mutations occur rarely and usually modify value of a randomly selected gene of the individual. The selection is typically implemented as a probabilistic operator that is based solely on the fitness value. The genetic algorithm terminates when a sufficient solution is found or a given time limit (or a given number of generations) is exhausted.

For practical problems, the simple genetic algorithm is often considered as a basis for many enhancements, including: heuristic generation of the initial population, multi-point or more complicated crossover, elitism preserving the best individual for the next generation, more realistic selection, etc.

2.1.2 Genetic Programming

Genetic Programming (GP) was introduced by John Koza in late eighties as an extension to genetic algorithms in order to enrich the chromosome representation [100, 101, 105, 107]. Instead of fixed-length strings, GP evolves pieces of code written over a specified alphabet consisting of a set of functions and a set of terminals. The chromosome encoding can be directly executed by the system or compiled (interpreted) to produce a machine executable code. Genetic programming allows automatic programming and program induction (i.e. automatically developing of computer programs). Unlike genetic algorithms, genetic programming does not distinguish between phenotype and genotype. As genetic programming is able to effectively evolve symbolic expressions, the problem of symbolic regression became the most popular application of GP.

The evolved programs are usually represented either as tree structures or in a linear form using a list of machine-language instructions. Similarly to the GA, crossover is considered as a major genetic operator. A typical crossover interchanges randomly chosen subtrees of parents' trees without the disruption of the syntax. A typical mutation, another genetic operator, selects a random subtree and replaces it with a randomly generated one. Selection is typically implemented as a probabilistic operator, using the relative fitness, which determines the selection probability of an individual.

In order to improve the efficiency in GP, John Koza introduced the concept of automatically defined functions (ADFs) [101]. Automatically defined functions enable genetic programming to define useful and reusable subroutines (subtrees) dynamically during evolution. According to the obtained results, genetic programming with ADFs produces solutions that are simpler and smaller than the solutions obtained without automatically defined functions.

The GP representation has also its own pitfalls. An evolved program may contain

segments which do not alter the result of the program execution when they are removed from it. A typical trivial example is the expression $x = x \cdot 1 + 0$ where the addition as well as the multiplication represent redundant operations that can be omitted. The redundant segments are referred to as introns. Another well-known issue of GP is that the program size can grow uncontrollably until it reaches the tree-depth maximum, while the fitness remains unchanged. This effect is called a bloat. These pitfalls and their relations are discussed, for example, in [8, 12].

2.1.3 Evolutionary Strategies

Evolutionary strategies proposed by Bienert, Rechenberg and Schwefel have been developed for optimization purposes in industrial applications [156, 11]. Similarly to the genetic programming, evolutionary strategies do not distinguish between a genotype and phenotype. Each individual is represented as a real-valued vector. Evolution strategies use primarily mutation and selection. Unlike the previous approaches, the mutation operator, which mutates each vector element, is considered as a major genetic operator. Mutation aggregates a normal-distributed random variable and a preselected standard deviation value which are applied on every gene of a candidate vector.

The simplest evolutionary strategy operates on a population of size of two consisting of the current solution (parent) and the result of its mutation (offspring). The selection strategy is strictly deterministic. Only if the offspring's fitness is at least as good as the parent's one, it becomes the parent of the next generation. Otherwise the offspring is disregarded. This basic strategy is known as a $(1 + 1)$-ES. More generally, λ offspring can be generated. The strategy in which the offspring compete with μ parents is called $(\mu + \lambda)$-ES. Another selection scenario (μ, λ)-ES picks the best μ individuals from the both child and parent populations. In the simplified (1,1)-ES variant, the offspring becomes the parent of the next generation while the current parent is always disregarded.

2.1.4 Evolutionary Programming

Evolutionary programming was introduced by Lawrence Fogel in sixties in order to use simulated evolution as a learning process [56, 55]. He has used the evolutionary programming for the design of finite state machines working as predictors. Evolutionary programming exhibits a number of similarities with evolutionary strategies and is becoming harder to distinguish from that paradigm. Important features of advanced evolutionary programming systems typically include a problem-specific representation, self-adaptation and tournament selection. Mutation operator is considered as the only genetic operator, crossover or similar recombination operators are not usually used at all. The search space and the representation space are not distinguished explicitly.

2.1.5 Cartesian Genetic Programming

Cartesian genetic programming (CGP), introduced by Julian Miller and Peter Thomson in 2000, is a variant of genetic programming where the genotype is represented as a list of integers that are mapped to directed oriented graphs rather than trees [131]. The motivation

for this representation came from the previous analysis covering the effectiveness of this approach in learning Boolean functions where the CGP has been proved to be considerably more efficient than any other variant of GP.

Cartesian genetic programming encodes a candidate solution (typically a circuit or a program) using an array consisting of $n_c \times n_r$ programmable nodes. The n_c parameter determines the number of columns whereas n_r determines the number of rows. Each programmable node has the fixed number of inputs, n_{ei}, and outputs n_{eo}; in most cases $n_{ei} = 2$ and $n_{eo} = 1$. The main feature of CGP is that all the parameters including the number of programmable nodes, node inputs and outputs and program inputs, n_i, and program outputs, n_o, are fixed. Each node input can be connected either to the output of a node placed in the previous l columns or to one of the program inputs. The parameter l (referred to as l-back parameter) defines the level of connectivity and thus reduces or extends the search space. For example, if $l=1$ only neighboring columns may be connected; if $n_r = 1$ and $n_c = l$, full connectivity is enabled. Because of the complicated evaluation, feedback is not allowed in the standard version of CGP. Each node can be programmed to perform one of n_{ei}-input functions defined in the set Γ. Let $n_f = |\Gamma|$. Thus, every individual can be encoded using $n_c \times n_r \times (n_{ei} + n_{eo}) + n_o$ integers.
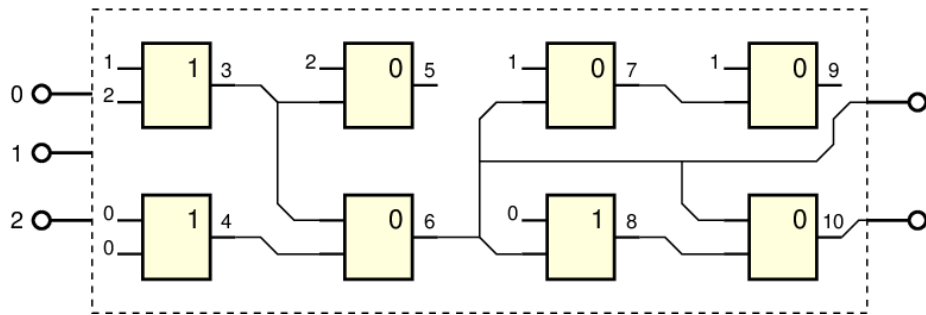
Figure 2.2: Example of a candidate circuit encoded using CGP with the following parameters: $n_c = 4$, $n_r = 2$, $n_i = 3$, $n_o = 2$, $l = 2$, $n_{ei} = 2$, $n_{eo} = 1$, $\Gamma = \{$AND (0), OR(1), XOR (3) $\}$. Chromosome: 1, 2, **1**, 0, 0, **1**, 2, 3, **0**, 3, 4, **0**, 1, 6, **0**, 0, 6, **1**, 1,3, **0**, 6, 8, **0**, 6, 10. Functions of elements are typed in bold. The first 24 integers encode the interconnection of the CGP elements and function of each element. The last two integers indicate the output of the circuit. Elements 5, 7 and 9 are not utilized.

Figure 2.2 shows a digital circuit encoded using CGP representation. The figure also demonstrates the main feature of CGP encoding – while the genotype (i.e. chromosome) is of fixed length, the phenotype is of variable length depending on the number of unexpressed genes. In this example, three nodes do not contribute to the phenotype. The utilized representation also significantly reduces the bloat which is inevitable in GP [126]. This fact has been confirmed by Miller in [129] who claimed that it cannot occur in the genotype just because it is bounded.

Due to the presence of redundancy, there are many genotypes that are mapped to identical phenotypes. The simplest form of redundancy is caused by the presence of genes or nodes that are inactive. These genes influence neither the phenotype nor the fitness value. This kind of redundancy is very high at the beginning of the evolution as many nodes are

not connected in the early populations. With the increasing number of generations, the node redundancy gradually reduces to a level that is determined by the average number of nodes required to obtain a satisfactory solution and the maximum allowed number of nodes [131].

The phenomenon indicating the presence of genotypes with the same fitness is often referred to as neutrality. The role of neutrality has been investigated in detail in several papers [198, 129, 34]. For example, it was discovered that the most evolvable representations occur when the chromosome is extremely large and contains over ninety percent of inactive genes [129]. On contrary, it has been also shown that for some specific problems, the neutrality based search is not the best approach [34].

In CGP, the search is performed using a mutation-based evolutionary strategy $(1 + \lambda)$-ES that does not utilize crossover as it has been discussed in the previous chapters. The influence of the crossover operator has been intensively studied in literature, however, it has been confirmed that crossover does not improve the search [128, 183]. CGP operates with the population of $1 + \lambda$ individuals, where λ is typically from 1 to 15. In case of the evolutionary design, the initial population is usually generated randomly whereas in case of the evolutionary optimization the initial population can be constructed by means of mapping of a known conventional solution to the CGP representation.

The search strategy works as follows. The initial population has to be evaluated using a fitness function and the fittest individual becomes the new parent. Then, every new population consists of the best individual of the previous population and its λ mutated offspring. The offspring are created by a point mutation operator which modifies m randomly selected genes of the parental individual where m is a user-defined value. The mutation operator usually modifies up to 5% of genes of the chromosome. The implementation of the mutation operator has to ensure that the modifications are legal and lead to a valid phenotype. The moment the population is created, the fitness value of each offspring is calculated. The fittest individual in the population is selected and forms the new parent. In case when two or more individuals have received the same fitness score, the individual which has not served as the parent in the previous population has to be selected as the new parent. This strategy is important because it ensures the diversity of population and allows so called neutral search [129]. The evolution is terminated when the maximum number of generations is exhausted.

The CGP became the routinely used approach in the area of evolutionary-based digital circuit synthesis and optimization. The main advantage of CGP is that it generates very compact solutions, i.e. it can effectively reduce the total number of gates in the case of circuit evolution [127]. Even if the CGP was originally defined for gate-level evolution, it can easily be extended for function-level evolution [157]. CGP has been successfully utilized in many applications [125, 91, 131, 127, 182, 157, 4, 57, 186, 223]. In addition to the standard CGP, several extensions have been proposed in recent years; for example, self-modifying CGP [74], modular CGP [188, 92], developmental CGP [164] or multi-chromosome CGP [204]. Some authors have also utilized CGP with a relative encoding of the solution instead of the absolute encoding introduced by Miller [70].

2.2 Reconfigurable Devices

In recent years, we could observe a boom in the area of reconfigurable devices and reconfigurable computing [181]. In comparison to fixed architectures, the structure and parameters of reconfigurable chips can be modified by writing configuration data to the configuration memory. The reconfigurable devices usually consist of configurable blocks whose functions and interconnections are controlled by the configuration bitstream. While the first programmable devices such as Programmable Logic Arrays (PLAs) used hundreds of bits to store the configuration and relatively simple reconfigurable structure, recent devices such as Field Programmable Gate Arrays (FPGAs) require tens of megabytes to store their configuration bitstream. Due to its potential to accelerate a wide variety of applications, reconfigurable computing has become a subject of intensive research. Its key feature is the ability to perform computations in hardware to increase performance, while retaining much of the flexibility of a software solution.

The possibility of reconfiguration is typical for digital architectures. However, reconfigurable devices are now available in the areas of analog circuits, antennas, mirrors, molecular electronics and others. This short survey introduces the concept and basic principles behind the FPGAs as well as other reconfigurable devices that have been used in the evolvable hardware field.

2.2.1 Reconfigurability and Its Benefits

There are several reasons for using reconfigurable hardware. The reconfiguration can extend the lifespan of a system due to the possibility to update the firmware. For example, when a new driver or peripheral device is introduced to a system, existing hardware could have a problem to communicate with it. However, if the system is implemented in a reconfigurable chip, the hardware can be updated by simple reprogramming the configuration memory. In this case, the reconfiguration is performed occasionally and only when the application is suspended.

Another common scenario is to use a reconfigurable chip in order to increase the functional density. The goal is to perform a complex task on a small chip and thus reduce the power consumption, size or weight of the application, even reduce the cost. The application has to be divided into modules whose configurations alternate on the chip. The reconfiguration is performed dynamically at runtime.

The reconfigurability also gives the chance to create an adaptive hardware. In this case, the goal is to dynamically create electronic circuits that are optimized for a given task, time and location of the chip.

And finally, the typical reason why reconfigurable devices are used is shortening the design time. Creating a configuration for a reconfigurable device usually takes much less time than building a new application-specific chip.

2.2.2 Digital Reconfigurable Devices

In order to control the routing among the configurable blocks, a kind of configurable switch matrix is used. To establish the routing on a reconfigurable chip, a passgate structure is

typically employed. Modern digital reconfigurable devices such as Xilinx FPGAs contain a rich routing fabric consisting of millions of routing choice points. The configuration bits directly control the configurable switches, selection signals of multiplexers, contents of lookup tables (LUTs) and some bits used as control signals for computational unit. A single chip can implement many different functions depending on its configuration. The main disadvantage of this approach is that the circuitry established in order to allow the configurability occupies a considerable area on the chip and make the whole system slower in comparison with application specific integrated circuits. Since the FPGAs represent the mainstream in the area of digital reconfigurable devices, we will restrict ourselves to these devices only.
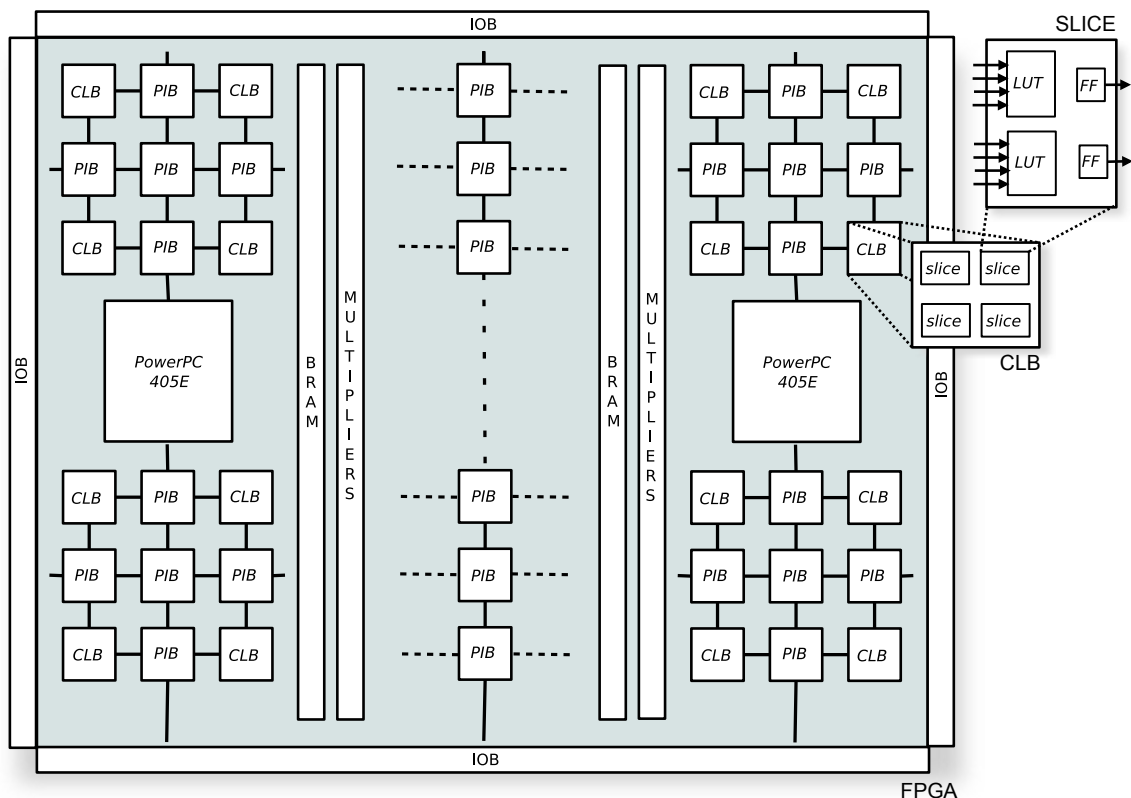
Figure 2.3: The hierarchical architecture of FPGA Virtex II Pro which contains two PowerPC processors, embedded multipliers and memories.

Field Programmable Gate Arrays

Figure 2.3 shows a typical architecture of a modern Xilinx FPGA [192]. FPGA consists of a two-dimensional array of reconfigurable resources that include configurable logic blocks (CLB), programmable interconnect blocks (PIB) and reconfigurable I/O blocks (IOB). The configuration bitstream configuring all these elements is stored in the configuration SRAM memory. A CLB consists of several smaller elements referred to as slices. Each slice contains two function generators implemented using k-bit LUTs, flip-flops and some additional logic. The number of bits k is usually between 3 and 6 depending on the FPGA family. Each

function generator can be programmed into one of the three modes. In the first mode, it can implement a combinational function. In the second mode, it can implement a fast k-bit shift register. And finally, the last mode enables to configure the LUT as a fast synchronous RAM with the total capacity of 2^k bits.

A typical structure of an FPGA logic block consisting of 4-input LUTs is depicted in Figure 2.4. While the LUTs provide some kind of generic logic, the flip-flops can be used for pipelining, registers, stateholding functions for Finite State Machines, or any other situations where clocking is required. Note that the flip-flops typically include programmable set/reset lines and clock signals. These signals may come from global signals routed on special resources, or could be routed via the standard interconnect structures from another input or logic block. The fast carry logic is a special resource provided in the cell to speed up carry-based computations, such as addition, parity, wide bit-wise operations, and other functions. These resources bypass the general routing structure in order to directly connect neighboring CLBs in the same column. Since there are very few routing choices in the carry chain, and thus less delay on the computation, the inclusion of these resources can significantly speed up the carry-based computations.
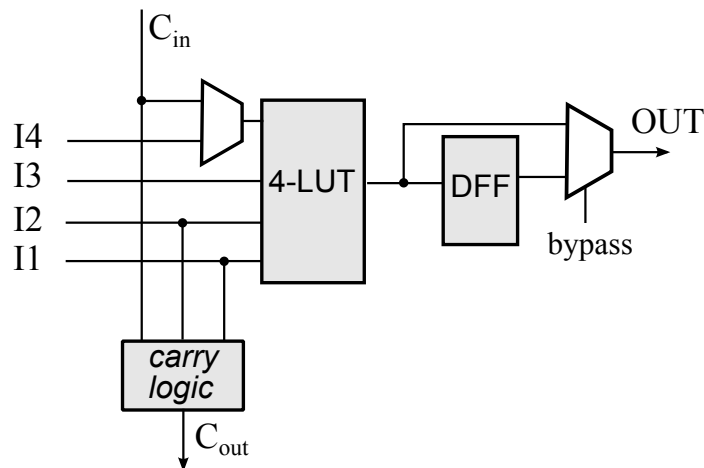
Figure 2.4: A typical structure of an FPGA logic block.

The FPGAs differ in the amount and type of resources available on the chip. The most advanced FPGAs based on 6-input lookup tables contain more than 100 thousands CLBs and integrate, in addition to CLBs, various embedded hard cores such as SRAM memories, fast multipliers, gigabit interfaces, PCI interfaces or even processors (PowerPC or ARM). Because the existence of these cores has been identified as important to designers in the past, it is reasonable to integrate them as hard cores on the chip instead of implementing them using CLBs and other resources. Current FPGAs can compete with application specific integrated circuits (ASICs) in many domains, for example, in applications of advanced signal processing or embedded systems.

Most FPGAs support a dynamic partial reconfiguration which means that some parts of the FPGA can be reconfigured while remaining parts of the FPGA are performing some computation. As it will be mentioned later, the possibility of the partial reconfiguration is crucial for evolvable hardware. FPGAs can be configured either externally or internally. In

the case of external reconfiguration, the configuration bit stream is copied to the configuration memory from an external memory, typically FLASH. The internal reconfiguration is available in Xilinx Virtex FPGAs via the Internal Configuration Access Port (ICAP) which allows for reading and modifying the FPGA configurations by circuits created directly in the same FPGA.

The goal of digital circuit design is to provide such implementation of a target circuit which satisfies the user specification and which is available in a reasonable time. As the conventional circuit design process with FPGAs is very similar to programming, the resultant system can be obtained relatively quickly. Designer has to describe the circuit structure or behavior using a hardware description language (such as VHDL, Verilog, Catapult C, etc.). Then, the source code is automatically transformed into the configuration bit stream for a particular FPGA. The transformation, which includes the synthesis, placement and routing is performed by CAD tools. This process can be constrained using various requirements, e.g. the maximum delay of the circuit can be specified. Also, it is possible to simulate intermediate results of the transformation, modify the original source code when needed and optimize the design.

In most cases, the format of the configuration bit stream is not fully documented for the designer. The reason is that the manufacturers protect their know-how and prevent the designers from potentially dangerous attempts to configure the FPGA without a CAD tool. In case of Xilinx chips, the only exception was the XC6200 family which is nowadays obsolete. The XC6200 family was very popular as it allowed to carry out intrinsic EHW experiments at the lowest possible level of abstraction. Comparing to the modern FPGAs, the basic building block of XC6200 was very simple as it contained several 2-input multiplexers instead of lookup tables. Thus, each programmable logic element could be programmed to implement a common 2-input Boolean functions such as AND, OR, XOR, etc. or 2-input multiplexer.

2.2.3 Analog Reconfigurable Devices

Reconfigurable analog circuits allow, in fact, a software control of analog circuits. In comparison with FPGAs, reconfigurable analog circuits contain fewer configurable blocks and operate at lower frequencies. The reconfiguration is usually based either on configurable transistor switches, analog multiplexers, switching capacitors or operational transconductance amplifiers. Reconfigurable analog chips have been introduced much later than FPGAs. Examples of analog reconfigurable circuits are given in the following sections.

Field Programmable Transistor Arrays

Field Programmable Transistor Array (FPTA-2) developed at NASA JPL employs transistor switches to implement the reconfiguration [170]. The FPTA-2 can implement analog, digital and mixed signal circuits. The architecture of the FPTA consists of an 8x8 array of re-configurable cells. Each cell contains a set of transistors and programmable resources, including programmable resistors and static capacitors. The reconfigurable circuitry consists of 14 transistors connected through 44 switches in each cell. In contrast with FPGAs, only several thousands of bits are used to program the whole chip only. The pattern of

interconnection among cells is similar to the one used in commercial FPGAs. Every cell can be interconnected with its northern, southern, eastern and western neighbors.

Another FPTA was developed at the University of Heidelberg [110]. This chip enables developing circuits directly at the transistor level. Designer can select the transistor type (PMOS or NMOS), its parameters such as channel length and size, and interconnection.

Field Programmable Analog Arrays

The reconfiguration of Field Programmable Analog Arrays (FPAA) is typically based on either switched capacitors or operational transconductance amplifiers.

Switched capacitors perform the function of configurable resistors. FPAAs use the following principle. A capacitor C is connected between two switches controlled by two signals. The switches are implemented using unipolar transistors and the control signals are non-overlapping clocks. The charge Q over one clock period transferred to the capacitor C is given by equation $Q = C(V_1 - V_2)$ representing a discrete version of a well-known differential equation. The average current associated to this charge is $I_a = C(V_1 - V_2)/T$, where T denotes the clock period. Applying both equations, the value of an equivalent resistor can be calculated as $R = (V_1 - V_2)/I_a = T/C$. Thus, the value of a corresponding resistor can be controlled by the switching frequency $f = 1/T$. In comparison to conventional resistors, switching capacitors are advantageous in terms of linearity, dynamic range, precision and size on the chip. As f can be controlled from software, analog circuits (such as filters and oscillators) can be easily tuned. A disadvantage might be that circuits containing switched capacitors operate in discrete domain, i.e. there is a limit in the possible operation frequency which is determined by f.

The commercially available Anadigm AN221E04 FPAA [5], developed using switching capacitors, is an array of four configurable analog blocks (CAB), each of which containing two operational amplifiers, a comparator, and an 8-bit analog-to-digital converter. The device also contains one programmable lookup table that can be used to store information for the generation of arbitrary waveforms. The table is shared amongst the CABs. The configuration bit stream is stored in SRAM and the maximum switching frequency is 16 MHz.

Another approach to software control of analog circuits is based on operational transconductance amplifiers. A typical operational transconductance amplifier (OTA) produces a current output I_o that is linearly depending on an input voltage present at both inverting input (V_-) and non-inverting input (V_+). The output current can be expressed as $I_o = -g_m(V_+ - V_-)$, where g_m is the transconductance of the circuit. The transconductance can be predefined using an external biasing current input. Biasing currents for OTAs are generated using D/A converters. Ideally, the circuit has infinite values for both the input and output impedances. OTAs are the main building blocks of continuous time filters in which the transconductance (and thus the frequency characteristics) can be controlled externally.

An example of FPAA which utilizes configurable OTAs is the FPAA developed at the University of Freiburg [76].

2.3 Evolvable Hardware

A massive application of evolutionary principles to hardware design and self-configuration has led to a new concept called Evolvable Hardware (EHW). The growth of this interest has been caused by emerging a new class of programmable devices, in particular FPGAs. The main idea is to accomplish the whole process of circuit design by evolutionary algorithms. Evolvable hardware refers to a hardware that a) has been created using EA or b) embeds a variant of EA in order to either adapt the system to changing environments, or repair the system autonomously during its lifetime. While the first scenario is usually called evolutionary design, only the second approach can be, according to the [201], referred to as evolvable hardware. The evolutionary design uses evolutionary algorithms to evolve a system that meets a predefined specification. EA is employed only in the design phase. In contrast with this approach, the adaptive systems reconfigure a part or whole existing design to repair the faults or adapt to a changed operational environment. Thus the evolutionary algorithm is an integral part of the adaptive system. Although the terminology has been successively evolved, some literature, e.g. [65], does not distinguish between evolutionary hardware design and evolvable hardware.

The field of evolvable hardware originates from the intersection of three sciences: electronic engineering, biology and computer science. EHW belongs to the area of bioinspired hardware which combines the ideas from biology and electronic engineering. Although evolvable hardware as a research area has been established two decades ago, its roots can be traced to the sixties, when Gordon Pask constructed several electrochemical devices having the ability to adaptively construct their own sensors [141]. In addition to that, evolutionary strategies were used to perform parameter optimizations for a variety of electronic designs. The first experiment which explicitly speaks about evolvable hardware was conducted by Higuchi and his team in 1993 [79]. They utilized a genetic algorithm to find a configuration of a simple programmable logic chip (GAL). The aim of this work was to rediscover an implementation of a common 6-input multiplexer only on the basis of a behavioral specification given in the form of truth table. Because the number of reconfigurations of a GAL chip was limited, a GAL simulator calculated the fitness value of each member of population. At the end of evolution, the best solution was verified using a real GAL chip. This experiment demonstrated that the evolutionary approach is able to synthesize an electronic circuit without being explicitly told how to do it. Since that, the evolvable hardware has been successfully applied in many different areas. The following results are usually mentioned as the most successful applications of EHW: evolutionary designed high-speed robust classifiers [77], high-performance high-quality adaptive hardware compression systems based on the predictive coding [153, 154], adaptive fault-tolerant system with autonomous recovery [58], evolutionary designed antennas optimized for space missions [83] or innovative image filters [158].

2.3.1 Basic Principle of EHW

EHW typically utilizes a reconfigurable hardware, particularly programmable logic devices such as FPGAs. The programmable logic devices allow the candidate solutions to be

tested in situ which is well suited to embedded applications such as adaptive image filters or adaptive controllers. Figure 2.5 shows the basic principle of the evolvable hardware approach.
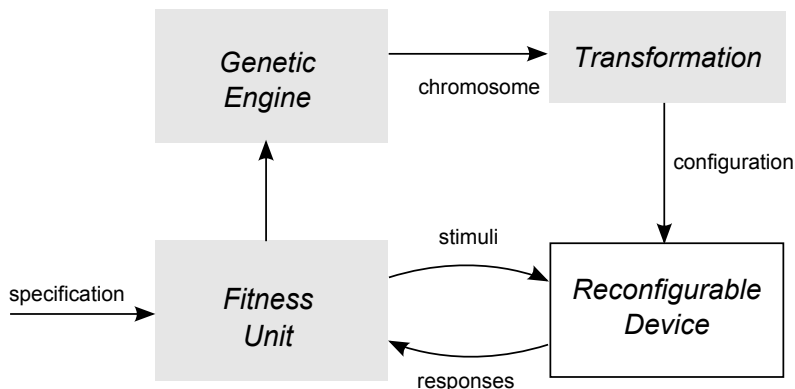
Figure 2.5: The basic principle of the evolvable hardware approach.

The objective of evolutionary algorithm is to design a circuit that meets the specification given by designer. In order to evaluate a candidate circuit, a new configuration of a reconfigurable device is created on the basis of the information stored in the corresponding chromosome. This step usually involves some kind of transformation as the chromosome encoding and configuration string can be generally different. The configuration is uploaded into the reconfigurable device and evaluated for a chosen set of input stimuli. The fitness function, which reflects the problem specification, can include behavioral as well as non-behavioral requirements. For example, the correct functionality is a typical behavioral requirement. As a non-behavioral requirement, we can mention the requirement for minimum power consumption or minimum area occupied on the chip. Once the evaluation of the population of candidate circuits is complete, a new population can be produced. That is typically performed by applying genetic operators (such as mutation and crossover) on existing circuit configurations. High-scored candidate circuits have got a higher probability that their genetic material (configuration bitstreams) will be selected for next generations. The process of evolution is terminated when a perfect or satisfactory solution is obtained or when a certain number of generations is evaluated.

2.3.2 Evaluation of Candidate Circuits

Several schemes have been developed for classifying the evolvable hardware, e.g. [65, 177, 200, 81]. In this section, we will focus on one key feature that is usually considered by the mentioned classifications – hardware evaluation process.

Two scenarios are usually applied for evaluation of candidate circuits. Early evolvable hardware experiments used circuit simulators in order to calculate a fitness value of each member of the population. This approach has been known as *extrinsic evolution*. If all candidate solutions are evaluated in reconfigurable hardware, then the approach is called *intrinsic* evolvable hardware. The off-the-shelf FPGAs represent the most popular intrinsic reconfigurable chips due their availability and outstanding performance.

The importance of intrinsic evolution has been recognized by Thompson who has carried out the first intrinsic experiment using FPGA at the lowest level of abstraction possible [175]. The task was to evolve a circuit that discriminates between 1 kHz and 10 kHz signals. Evolution was able to find a very small circuit, i.e. to perform a task that would require human designers to project larger and clocked circuits, or use passive components, such as capacitors and inductors. In fact, evolution used a digital programmable device in the analog mode to perform this task. However, in spite of the high effort of Thompson involving the usage of analog simulators, the nature of some of the mechanisms used by the evolutionary designed circuit has not been completely understood. This is probably caused by the presence of feedbacks that can not be easily simulated, since they are dependent on the propagation delay of each cell. Thompson's impressive results stimulated other scientists to investigate the field of EHW.

2.3.3 Evolvable Hardware as Design Tool

Using evolution to design electronics brings a number of benefits. Some of the most important areas where evolutionary electronics can successfully be applied include: automatic design of low cost hardware, automatic design of hardware systems for poorly specified problems, innovation in poorly understood design spaces, design of adaptive systems, or design of fault tolerant systems. The ability to generate solutions to poorly specified problems can be considered as a form of creativity which is one of the features of evolutionary processes. In case of the adaptive and fault tolerant system, the evolvable hardware is usually used due to its potential of autonomous adaption to changes in its environment (e.g. noise level in case of adaptive image filters, or presence of faults in case of fault-tolerant adaptive systems). The advantageous feature of the evolutionary approach is that it can not be necessary constrained to the well-known topologies that usually prevent from achieving novel solutions.

On the contrary, the evolutionary design approach has some drawbacks. Evolutionary methods are sometimes criticized that they do not produce robust and trustworthy designs. The evolved circuits are usually different from the well-known and proven structures which complicates their analysis and verification. Another discussed problem is enormous computational power which is usually needed for obtaining a satisfactory result. In some real-time applications (e.g. adaptive and fault tolerant systems), slow convergence or even stuck in a local extreme may represent an issue.

Chapter 3

Evolutionary Design of Analog and Digital Circuits

After reading the previous chapter, the evolutionary circuit design might seem to be substantially ineffective in comparison with conventional approaches. In many cases it is even true; however, there are applications where the evolutionary design brings a number of benefits unattainable by means of conventional design.
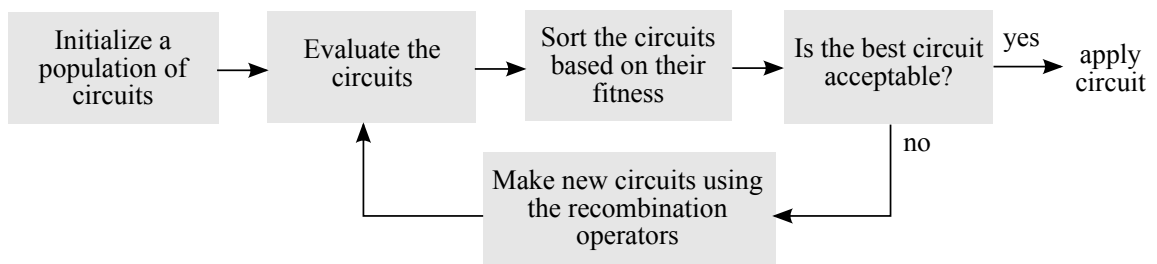
Figure 3.1: The basic principle of the evolutionary design of analog and digital circuits.

The basic principle of the evolutionary design of analog and digital circuits is depicted in Figure 3.1. The evolutionary design approach works as follows. Firstly, a population of initial solutions (circuits) is created. This population is usually generated randomly. Then, the behavior of each circuit is evaluated. In this step, a fitness value determining degree of correspondence with the initial specification is assigned to each circuit. If the fittest solution is acceptable, the algorithm is terminated. Otherwise, the best circuits are combined to generate new circuits and the algorithm continues with evaluation of newly generated circuits. After a number of iterations, the fittest circuit should behave according to the given specification.

This chapter surveys the most important approaches proposed mainly to extrinsic evolutionary design of analog and digital circuits.

3.1 Evolutionary Design of Analog Circuits

The first technique of analog circuit design automation has been reported in the seventies [174]. Since that, several other methods have been published, e.g. [75, 139]. Nevertheless, these methods utilized the approaches known from the area of artificial intelligence (e.g. expert systems, some kind of heuristics, or simulated annealing) rather than the evolutionary techniques. In contrast with the evolutionary design, these systems were very limited as they usually restricted the search space to well-known circuit topologies. A typical task was the optimization of a set of parameters. In the first applications dealing with the synthesis of passive filters, evolutionary approaches were used to perform the selection of topologies, or the simple determination of components' values for fixed topologies [67, 84].

The first truly analog circuit evolution was performed by Genetic Programming. The research group led by John Koza published pioneering methods solving this task, where all the steps necessary to design an analog circuit were handled by GP [103]. The objective was to find not only the values of utilized passive components but also the topology of the circuit. The first results published by Koza's team motivated researchers to deeply investigate this area. Tens of competitive analog circuits designed by means of the evolutionary algorithms have been reported up to now [99].

3.1.1 Synthesis of Analog Circuits Using GP

Passive filters are electronic circuits that consist of an arrangement of resistors, capacitors, and inductors. In order to implement an analog filter with required characteristics, one of the known polynomial filter structures might be used. The filters differ in the construction, complexity of corresponding circuit implementation and the parameters of the frequency response. For example, the Butterworth response is the one that is closer to the ideal frequency response. However, the Butterworth filters require more components than the other ones. As there is a wide range of different aspects that has to be considered, the conventional design of analog filters represents a nontrivial task. The main filter characteristics that have to be reflected include: amplitude of the frequency response, phase of the frequency response, group delay, impulse response, and response to the step function.

In order to evolve analog circuits, Koza came up with a developmental approach to perform a nontrivial mapping between circuit topologies and tree structures [103]. He used GP to find a tree that encodes a program to build a circuit from an initial circuit called the embryonic circuit. The initial circuit structure contains fixed and modifiable part. The fixed part includes the source resistance, output load, source signal and ground. It other words, it covers the essential features of the target circuit that has to be preserved. The modifiable part usually consists of pieces of wires that can be modified by the instructions encoded in the tree. Figure 3.2 depicts an example of a typical embryonic circuit used in Koza's applications.

The nodes of GP trees can be divided into functional and terminal nodes. The functional nodes may either create a component (component-creating functions) or modify a connection (connection modifying functions). Component-creating functions point to a modifiable part of the circuit, and create a particular component in this part of the circuit. In the
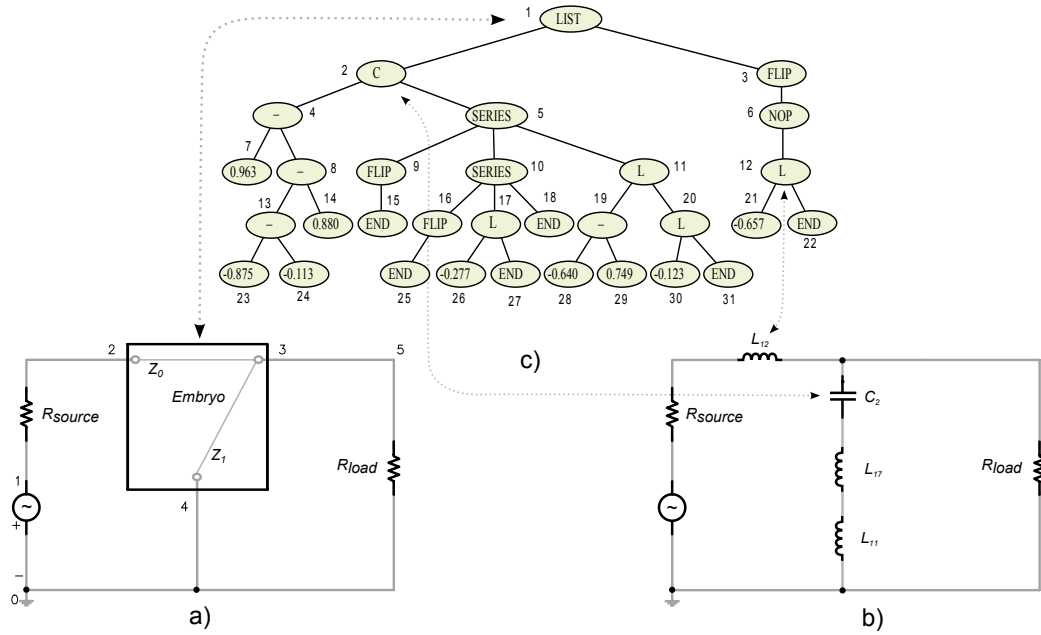
Figure 3.2: a) The initial circuit consisting of two modifiable wires denoted as Z0 and Z1. These wires provide some kind of place holder for inserting additional components. b) The analog circuit that has been created by the application of a candidate program. c) Candidate program encoded using a GP tree [104].

particular case of passive filters, we can have tree functions (R, C or L) which create resistors, capacitors, and inductors, respectively. The value of the corresponding component is determined by the expression which is encoded in the children nodes. The simplest case of expression is represented by a node containing a constant value. Connection Modifying Functions alter the topology of the modifiable parts of the circuit. There are four basic operations – Series (S), Parallel (P), Flip (F) and Ground (G). All these operations point to a particular wire or component in the modifiable part of the circuit. Series and Parallel operations create a series or a parallel connection of a particular component. Flip operation flips the corresponding component. This operation is intended for the components where the polarity plays important role (e.g. diodes). Finally, the Ground operation connects a given node to the ground potential. Apart from the functional nodes, two special terminal nodes are defined. Terminal NOP (N) which represents a no-operation and terminal End (E) which terminates the development in the corresponding branch. The example of this approach is illustrated in Figure 3.2.

In case of the evolutionary design of analog filters, the fitness function has to consider the circuit behavior in the frequency domain. In order to evaluate the quality of a candidate filter, Koza utilized AC signal analysis provided by the SPICE simulator. The fitness value has been calculated according to the following expression:

$$F(t) = \sum_i [W(d(f_i), f_i) \cdot d(f_i)], \qquad (3.1)$$

where i denotes the index of a fitness case (i.e. the index of a frequency response point that

21

is included and considered by the fitness function), f_i is the frequency of the fitness case i, $d(x)$ is the difference between the target and the observed values at frequency x, and $W(y, x)$ is the weight function for a difference y at frequency x. The task of setting the weights is the most serious problem in this fitness equation as it can significantly influence the performance of evolutionary algorithm. Thus it is usually determined by experimentation. This scheme provides a very flexible approach enabling to capture a wide range of different design tasks. Since W depends on the frequency, it allows to specify more or less important bands or even to define don't care bands. Moreover, it depends also on the calculated difference which gives the chance to smooth the search space.

Although the evolutionary synthesis of analog circuits is a time consuming process (a cluster consisting of more than 2000 PCs have been utilized), Koza has reported tens of human-competitive analog circuits automatically designed by means of this evolutionary approach up to now. In addition to the passive circuits, he has evolved circuits containing transistors, circuits with operational amplifiers, regulators or logic circuits. A similar method has been applied also to the evolutionary design of antennas, optical systems, classifiers, predictors and another nontrivial design task [106, 114, 83, 101, 105].

The evolutionary designed analog circuits are sometimes criticized that they are not trustworthy because they have not been verified in silicon and hence they can not be utilized by designers as building blocks. In order to address this issue, McConaghy et al. proposed EA-based system for a structural synthesis of trustworthy analog blocks that can be easily combined in order to create complex analog circuits [122].

Other approaches have been surveyed in [201].

3.2 Evolutionary Design of Digital Circuits

Typical goals of a synthesis algorithm include the minimization of the total number of gates (the area required to implement a circuit on a chip) and the minimization of the number of levels affecting the circuit propagation delay. Even if there exists a completely mechanical procedure of designing a correct circuit for any Boolean function represented by a truth table or as the sum of products, the conventional approaches provide suboptimal solutions in many cases. This is caused by the fact that various assumptions and simplifications have to be applied in order to manage the enormous complexity of current circuits in reasonable time.

In the rest of this chapter, three possible levels of abstraction are investigated in the context of evolutionary design. Even if the representations only differ in the degree of complexity of the basic building blocks, this detail can have significant impact not only on the performance of evolutionary algorithm, but also on the quality and novelty of evolved solutions. The goal of this survey is to show how these representations affect the performance of evolutionary algorithm, representation and fitness function implementation.

3.2.1 Transistor Level Representation

Transistor level representation is considered to be the lowest level of abstraction of digital circuits in this thesis. Nevertheless, conventional design approaches do not synthesize elec-

tronic circuits at this level due to a huge and still increasing complexity of circuits. Instead of this, a circuit designed at gate-level is mapped to transistor-level circuit. The process of mapping is straightforward; the gates are simply replaced by their transistor-level implementations which are available in a particular fabrication technology. In order to reflect the implementation costs, the conventional synthesis tools do not optimize only the number of gates but also the total number of transistors expressed in terms of relative implementation area. For example, while the NOT gate occupies the smallest area, the XOR gate occupies approximately three times larger area depending on the utilized technology.

Even if the occupied area is taken into account, the resulting circuits can be far from optimal solutions. For example, if a simple logic expression $Y = \overline{A \cdot B + C \cdot D}$ is considered, the optimal gate-level representation requires two AND gates, consuming six transistors in ordinary CMOS technology each, and a single NOR gate consuming four transistors. After technology mapping, this circuit will utilize 16 transistors in total. However, the same function can be implemented using 8 transistors only when a special AND-OR-Invert circuit is employed [186]. It is likely, that implicit redundancy will exists for a wide range of other and more complex circuits.

The evolutionary design of digital circuits at the transistor level is similar to the evolutionary design of analog circuits. However, it is useful to restrict the search space to meaningful topologies in order to reduce the search space. For example, the following constraints should be reflected: all the transistor's inputs are required to be connected; the CMOS circuits should contain both complementary branches, there should be a strong driver at the output (e.g. an inverter that restores the full voltage levels at the outputs) etc. Besides the use of a developmental approach proposed by Koza, there are also other approaches that utilize a form of direct encoding similar to CGP, e.g [186, 201].

The fitness value is typically calculated using a simulator, e.g. SPICE, which is able to accomplish the transient analysis. The following scheme is usually recommended. Let T is the total period of transient analysis. Then, for a circuit with n inputs, the period T is divided into 2^n slices of time as there exists 2^n input combinations. Each time slice is sampled using k discrete samples. The fitness value can be calculated using the following equation:

$$F(t) = -\sum_{i=0}^{k(2^n-1)} |d(iT_s) - v(iT_s)| \tag{3.2}$$

In fact, this equation computes the sum of the absolute deviations between the desired voltage, $d(t)$, and real output voltages, $v(t)$, over 2^n fitness cases using $k2^n$ discrete samples acquired every $T_s = T/(k2^n)$ seconds . The negative sign is used because the fitness should be higher for smaller deviations.

In comparison with the gate-level and function-level evolution, only relatively simple circuits were evolved directly at the transistor level. For example, Keane et al. evolved a NAND gate using a developmental genetic programming [107]. Zaloudek and Sekanina proposed a direct representation loosely inspired by the Cartesian genetic programming that has been utilized for the evolutionary design of elementary two-input gates, small multiplexers and adders [186]. Walker et al. proposed evolutionary system that is able to

design variability-tolerant designs for future technology nodes [187]. The unconventional designs of inverter, two AND gates, and an OR gate more tolerant to variability than the conventional designs have been reported. Apart from the extrinsic approaches, various intrinsic experiments were performed on various reconfigurable platforms, including FPTA, FPTA-2 and PAMA [201, 110].

The main limitation of the evolutionary design at this level is primarily caused by an extremely time consuming fitness calculation and the problem of scalability of representation. When the extrinsic evolution is carried out, precise simulations of a candidate circuit must be performed (simulators of the SPICE family are usually used) in order to avoid possible malfunctions such as incorrect transient response, insufficient driving capabilities or incorrect operation at different timescales [171]. On top of that, the number of test-cases increases exponentially with the increasing number of primary inputs.

3.2.2 Gate Level Representation

The gate-level representation represents the most used approach to evolve digital circuits. The literature contains several direct as well as indirect approaches to the gate level encoding.

a	b	c	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

terminals: a, b, c
functions: AND, OR, XOR

chromosome:
AND(c, XOR(AND(a,b), OR(a,b)))
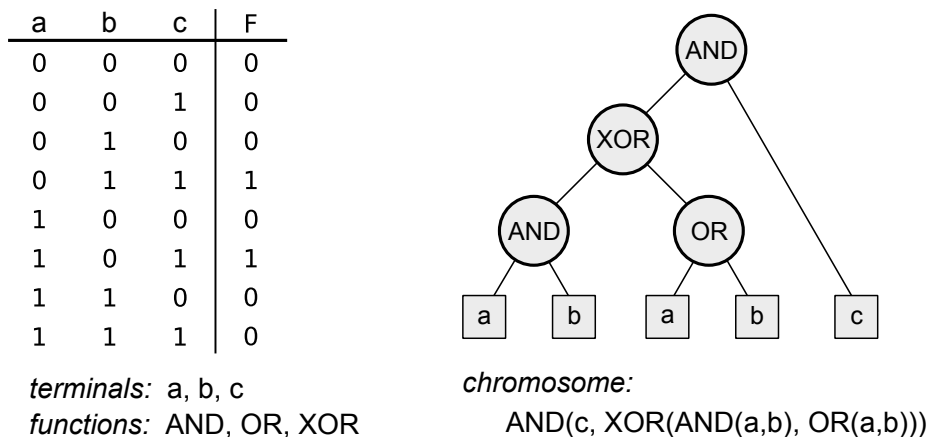
Figure 3.3: The combinational circuit specified by the truth table and its encoding using GP representation.

One can utilize tree representation of genetic programming. However, this approach is suitable only for a subclass of combinational circuits. The tree-based GP is not able to represent sequential circuits as the tree structure does not allow establishing a feedback loop necessary to implement sequential circuits. Common GP does not even allow to encode circuits with multiple outputs as only one root node exists. While the latter drawback can be removed by introducing more root nodes where each root corresponds with one output (e.g. Multi Expression Programming [140]), the efficient representation of the sequential circuits remains unaddressed. Figure 3.3 depicts an example of a combinational circuit specified by the truth table and its encoding using a GP tree.

Another scheme suitable for encoding of combinational circuits is linear genetic programming (LGP). In contrast with tree-based genetic programming, LGP uses an indirect

encoding. The genotype individuals have the form of a linear list of instructions encoded as binary strings [13]. Each instruction consists of the operation code and indexes of registers that store the operands. The program execution is performed using a simple register machine. As soon as the program is terminated, preselected registers contain the computed values. The structure of a linear GP individual is depicted in Figure 3.4. The sequence of instructions can be transformed into an equivalent functional representation in the form of a directed acyclic graph using the algorithm described in [20]. Providing that the instructions evaluate basic Boolean functions, the resulting acyclic graph represents a gate-level combinational circuit.
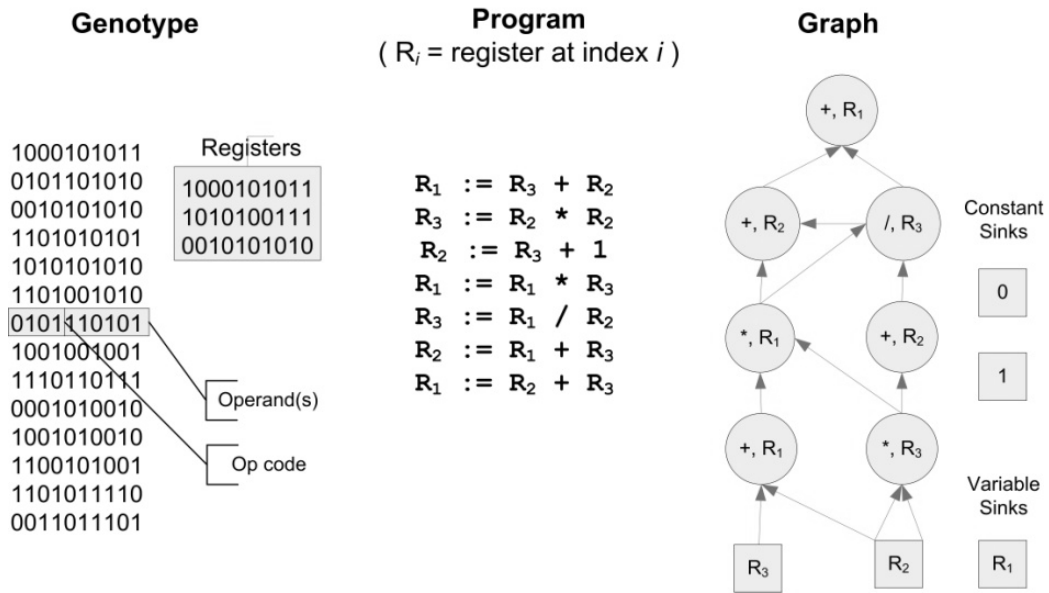
Figure 3.4: Representation of a LGP individual [191]

One advantage of linear GP is that the evolved program can be a binary machine code that can be executed during the fitness evaluation directly without interpretation [138]. Thus LGP is faster in evaluation than the tree-based GP. In [20], Bramier claimed that programs with a linear representation are more suitable to be varied in small steps than in a tree structure. On top of that, the programs in the linear structure are generally more compact due to multiple usages of register contents and an implicit parsimony pressure by the structurally non-effective code. However, the execution of linear GP programs is generally sequential, thus more work is needed in order to find a way to implement repetitions easily in this linear structure. Moreover, programs represented by binary machine code cannot be understood as easily as those in tree-based GP. In the context of evolutionary design of digital circuits, there is also another pitfall; some effort is needed in order to restrict the resulting circuits to the combinational ones.

Iba et al. utilized a form of variable length direct encoding in order to find a configuration of a programmable logic array (PLA) implementing basic combinational functions such as parity circuit and multiplexer [87]. The authors presented this approach as a gate-level EHW, because the PLA components are in fact AND, OR and NOT gates. The chromo-

some consists of one or more alleles containing two integers. The first integer encodes the position of the allele in the PLA fuse array while the second integer defines a connection type (direct and inverted). The inverted connection can be applied only for the AND array, as the OR array does not contain inverters. Figure 3.5 shows an example of a configured PLA array having 14 fuse locations and the corresponding chromosome. While the GP-based approaches are able to encode arbitrary combinational circuit, the PLA-based encoding is limited to the evolutionary design of two-level Boolean functions.
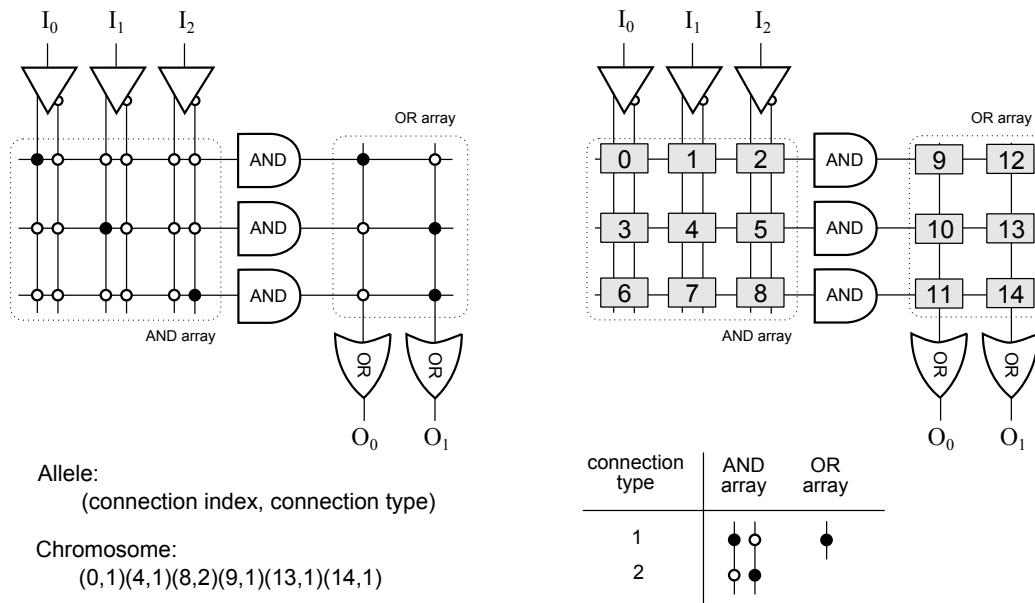
Figure 3.5: An example of a variable length encoding designed for evolutionary design of PLA configuration.
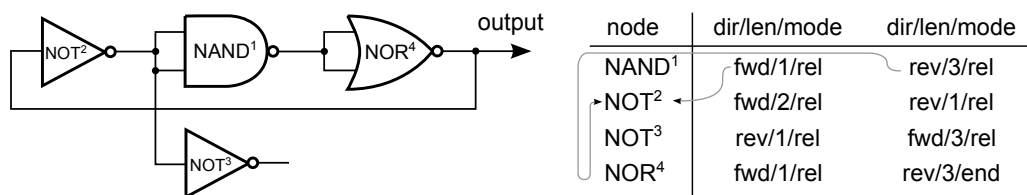
Figure 3.6: Ring oscillator composed of three inverting gates and its encoding using the approach presented in [175]. The chromosome consists of four 24-bit segments encoding 4 gates. The output of oscillator is connected to the last segment.

Another form of direct encoding for digital circuits has been proposed in [33] and adopted for gate-level evolution in [175]. In this scheme, the genotype forms a bit-string consisting of a fixed number of segments, each of which directly encodes the function of a particular gate and the sources of its inputs. The source of each input is specified by counting forwards or backwards along the genotype for a certain number of segments. The counting direction is specified by means of the direction bit, the number of segments is determined by the length field. The counting starts either from the current segment or

from the last segment according to the addressing mode bit. The counting wraps around the boundaries. This scheme is suitable for the evolutionary design of combinational as well as sequential circuits. Figure 3.6 shows an example of a ring oscillator encoded using this approach. The example illustrates the main feature of the encoding – the relative order of gates encoded in genotype is different from phenotype.

In [225], a developmental approach for synthesis of combinational circuits based on enhanced cellular automata (CA) has been proposed. The goal of this work was to demonstrate, that the evolutionary algorithm is capable of creating a cellular automaton that is able to construct a digital circuit as it develops. Each rule of CA's local transition function is connected with an action that creates a gate of a given type. The developmental approach is demonstrated in Figure 3.7. Even if the resulting circuit is relative simple, the proposed CA demonstrates the compactness of developmental encoding. The evolved CA has to apply only ten rules in order to generate a circuit consisting of 12 gates.
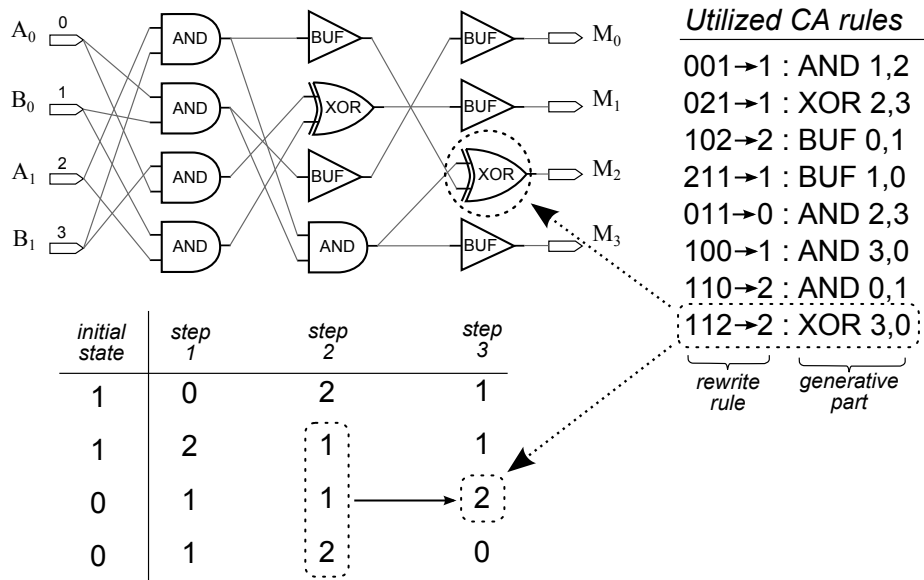
Figure 3.7: Example of the development of a 2x2-bit multiplier using a cellular automaton. The multiplier has been developed from the initial state 1100. A part of local transition function of the CA applied to development of the multiplier is shown on the right-hand side.

If we summarize the mentioned representations, none of the presented approaches could be considered as universal encoding scheme. The genetic programming can not capture asynchronous circuits and according to Koza's experiments it does not scale well probably due to the existence of bloat effect and ineffective representation. The linear genetic programming can represent both sequential and combinational circuits however an additional effort is needed in order to restrict the resulting circuits to the combinational ones due to the indirect instruction-based approach. The encoding used by Thompson has the similar problem due to the variable order of gates in the phenotype that can be changed with each mutation.

In Chapter 2, we have introduced a variant of genetic programming called Cartesian

Genetic Programming that has been primarily designed for evolutionary synthesis of digital circuits at the gate-level. The CGP representation has a number of interesting features. Firstly, as only some nodes are utilized in the genotype, there is a degree of redundancy which has been shown to be very useful [129]. Secondly, as the genotype encodes a graph, some nodes can be reused, which makes the representation very compact and also distinct from the tree based GP. Although CGP is very similar to the linear GP, there is one important difference – the restriction of the feed-forward connectivity. While the CGP restricts connectivity using the l-back parameter, LGP's connectivity is implicit and under evolutionary control as a component of the genotype. In other words, the CGP enables to specify whether combinational or sequential behavior should be evolved whereas LGP does not. This fact significantly complicates the fitness evaluation because asynchronous or sequential circuits have to be evaluated in a different way. The l-back parameter also determines the maximal length of a combinational path and thus it can be used to restrict the propagation delay of a combinational circuit. Similarities of CGP and LGP have been investigated in [191]. Also a variant of CGP allowing the feedback loops has been investigated there. However, the results are hardly interpretable as the authors have used only two instances of regression benchmarks that give questionable results. We can conclude that CGP can be considered as the best-available method for digital circuit evolution.

Evaluation of Candidate Circuits

In case of combinational circuit evolution, the fitness function measures the quality of a candidate solution through the number of correct output bits (i.e. the number of hits), compared to the specified (i.e. target) truth table. When a circuit with n inputs and m outputs ought to be designed, the objective is to find a solution that can attain $m2^n$ hits, corresponding to the size of the truth table and the number of outputs. The fitness value of a candidate circuit that reflects the implementation cost can be defined as [91]:

$$fitness = \begin{cases} b & when & b < m2^n, \\ b + (n_{max} - z) & otherwise, \end{cases} \qquad (3.3)$$

where b is the number of correct output bits obtained as response for all possible assignments to the inputs, z denotes the number of gates utilized in a particular candidate circuit and n_{max} is the total number of available gates ($n_{max} = n_c n_r$ in terms of CGP). It can be seen that the last term $n_{max} - z$ is considered only if the circuit behavior is perfect, i.e. $b = b_{max} = m2^n$. This scheme is referred to as a two-stage fitness strategy. Alternatively, we can replace the number of utilized gates by the number of utilized transistors which is a more precise measure as implementation costs of gates are different [57]. We can observe that the evolution has to discover a perfectly working solution firstly while the size of circuit is not important. Then, the number of gates is optimized. Similarly, delay or power consumption may be optimized. A multi-objective formulation of the circuit evolution problem was also proposed, but evaluated using small benchmark problems only [80].

Assume that the objective is to find a circuit that implements function $\mathbf{y} = \mathcal{F}(\mathbf{x})$ specified in the form of a truth table where $\mathbf{x} = (x_1, \ldots, x_n) \in \{0, 1\}^n$ is the vector corresponding with the circuit's inputs and $\mathbf{y} = (y_1, \ldots, y_m) \in \{0, 1\}^m$ is the vector corresponding with

the required output defined by truth table. The number of correct output bits b can be calculated according to the following equation:

$$b = m2^n - \sum_{x=0}^{2^n-1} \Omega(\mathcal{F}([x]) \oplus \mathcal{G}([x])) \qquad (3.4)$$

where $\mathcal{G}$ is the response of a candidate solution to the input vector $\mathbf{x} = [x]$, Ω is a function that counts the number of bits set to 1, $[x]$ is a binary representation of x at n bits and $\oplus$ is a binary exclusive-or operator. As it will be shown in Chapter 3.3, this equation can be effectively implemented on a common processor. In fact, the sum calculates the Hamming distance between the response and required output value for each test case.

Even if there are circuits that need not to be specified by a complete truth table, e.g. 4-bit priority encoder which can fully be defined using a truth table containing only one quarter of all input combinations, it is clear that this method is not applicable for design or optimization of large circuits because of the time consuming fitness evaluation that increases exponentially with the increasing number of primary inputs. A common solution applied, for example for training neural networks, is to use only some representative inputs vectors. However, it has been shown that the evolved digital circuits do not usually work correctly for some of the remaining input vectors [130]; it is just because it was not required to do that.

In order to overcome this disadvantage and evolve large designs, a kind of decomposition strategy can be employed. For example, Torresen introduced a *divide-and-conquer* approach for the evolution of digital circuits [176]. Kalganova applied a kind of incremental evolution that is able to semi-automatically divide a complex task into simpler subtasks [90]. This method has been extended by Stomeo and referred to as generalized disjunction decomposition [172]. However, the scalability problem has been eliminated only partially.

If a synchronous sequential circuit ought to be evolved, the same scheme can be utilized as for the combinational circuits. The objective of the evolutionary algorithm is to find a combinational circuit that determines the next state and the output value according to the knowledge of the actual state and the current inputs. The memory that keeps the current state is modeled using a set of registers. In case of the asynchronous circuits, it is necessary to evaluate the stability of the circuit because the asynchronous circuits contain feedback loops. This represents an additional evaluation time. Since the evaluation of sequential circuits requires the comparison of a sequence of circuit outputs or internal circuit states against the desired sequence, the evolutionary design of asynchronous sequential circuits is tractable for trivial circuits only.

Survey of Circuits Evolved at Gate Level

Let us conclude this section with the overview of the circuits reported in literature evolved at the gate level. The first results in the area of digital circuit synthesis were reported by Koza, who investigated the evolutionary design of the even-parity problem in his extensive discussions of the standard GP paradigm [100] and ADFs [101]. Although the construction of an optimal parity circuit using XOR gates is a straightforward process, the parity circuits are considered to be appropriate benchmark problem when the AND, OR, NOT gate set is

allowed. Unsurprisingly, solving the parity problems using standard GP without ADFs is computationally expensive and Koza was unable to obtain any result for the parity circuits having more than five inputs. Using ADFs, Koza reported greater success; the parity circuits up to 11 inputs have been successfully evolved. Koza was unable to design larger parity circuits not because GP with ADFs was failing to find a solution, but because the combination of the large population sizes and the increasing number of fitness cases to be evaluated was becoming computationally extremely expensive. Poli et al. extended Koza's GP approach by introducing new search operators and a novel node representation [144]. The goal was to smooth the fitness landscape in order to solve large instances of parity problem. Even if the proposed method does not use ADFs, the authors were able to design parity circuits having up to 22 inputs. However, while Koza utilized only four gates (AND, OR, NAND, NOR), they used a complete set of Boolean functions including XOR.

Thompson used a form of direct encoding in his intrinsic EHW experiment [175]. His goal was to evolve a square wave oscillator (i.e. asynchronous sequential circuit) within a small FPGA (up to 100 gates can be utilized) that oscillates at a much slower timescale than the gate delays. An interesting point is that the circuit behavior was defined using neither truth table nor state table; he counted the number of peaks within a certain time window instead. Even if the evolution found several solutions, the visual inspection on an oscilloscope showed that all of the evolved solutions produce very high frequency waveforms, but that these high frequency components are not crossing the digital logic threshold hence they are not being registered by the counter.

Iba et al. employed evolutionary algorithm to find a configuration of PLA for three circuit instances – a circuit with randomly generated Boolean function having 32 inputs with 6 terms, 6-input multiplexer and 4-input parity [87]. The goal was to find an approximation of the given Boolean functions according to limited training data. At the end of evolution, the randomly generated circuit worked well only for a small fraction of all input combinations (1650 out of 2^{32}).

Miller et al. demonstrated that evolutionary design systems are not only able to rediscover standard designs as it has been shown in past, but they can, in some cases, improve them [132]. He was interested in the evolutionary design of arithmetic circuits such as adders and multipliers. A gate level CGP was employed with the function set including AND, NAND, OR, NOR, XOR, and MUX logic functions. He has reported a one-bit adder and two-bit multiplier designed at the gate-level (i.e, without the MUX gate). Both circuits required fewer resources comparing to the designs produced by human designers. Three-bit multiplier consisting of 24 two-input gates evolved using Cartesian genetic programming has been introduced in [127]. The multiplier is about 20% better (in terms of two-input gates) than the conventional implementation. Four-bit multiplier consisting of 57 two-input gates has been reported in [182]. The circuit was evolved from the conventional multiplier employing 67 cells.

CGP has been also utilized for evolutionary design of digital filters at gate level [125]. It is a quite challenging task because nothing is supposed about properties (e.g. conventional approach always requires that filtering circuits are linear systems). The goal was to evolve the 4th orde digital filters working with 4-bit coefficients (i.e. combinational circuits having 32 inputs and 8 outputs). Resulting filters exhibited elementary functionality, however, only

for training signals. This approach has been thoroughly analyzed in [210]. It was shown that the evolutionary design of digital filters at the gate level does not produce filters that are useful in practice when linearity of filters is not guaranteed by the evolutionary design method. Another problem is that the goal was to find a circuit that minimizes the difference between the obtained signal and expected signal using 128 samples only, instead of finding a combinational circuit for a given truth table having 2^{32} rows. Moreover, target circuit could not fit into the available resources.

In [80], a multi-objective approach has been introduced to gate-level evolution. The goal was to find an optimized solution for several small circuits having up to 40 gates and 6 inputs (3-bit multiplier, 3-bit adder, 7-segment display decoder). The fitness function considered not only the number of logic gates and transistors but also the propagation delay. Resulting solutions are organized on the Pareto front which allows the designers to choose the most suitable one.

Many other approaches were proposed for evolution of small combinational as well as sequential circuits but none of them has provide results competitive with CGP, see e.g. [1, 4, 165].

If we look at the results achieved at the gate-level during the last two decades, it appears that the problems solved now by evolutionary techniques are nearly of the same size and complexity as those that have been solved several years ago [69]. The most complex combinational circuit that has been evolved consists of tens of gates having up to 20 inputs [172]. Apart from the problem related to the fitness calculation time, there is another problem related to the encoding scheme. Long chromosomes which are usually required to represent complex solutions imply large search spaces that are typically difficult to search. In many cases, even a well tuned parallel evolutionary algorithm running on a cluster of workstations fails to find an adequate solution in a reasonable time. In order to evolve large designs and simultaneously keep the size of chromosome small, the function-level evolution seems to be a promising way [134].

3.2.3 Function-level representation

In the function-level evolution, circuits are designed using higher functions such as multiplexers, adders, comparators, LUTs, etc. Function-level approach can evolve solutions for complex problems, but the main weakness is that human has to provide the most appropriate functions for specific problems. It seems that the evolutionary approach can be successfully applied only if a kind of domain knowledge which helps in focusing the search algorithm on promising areas of the search space is introduced. Next paragraphs will survey basic models proposed in the literature.

A single k-input lookup table can implement arbitrary Boolean function defined over k variables, including $\frac{k}{2}$-to-1 multiplexer. As the LUTs can model multiplexers, a cascade of lookup tables can be used to implement any combinational circuit having n inputs. Figure 3.8 shows an example.

Although this representation is very efficient, it has some caveats. In the traditional logic synthesis, the set of considered sub-functions is determined by the set of operators used in the initial synthesis and by the set of gates in the library of logic blocks used in
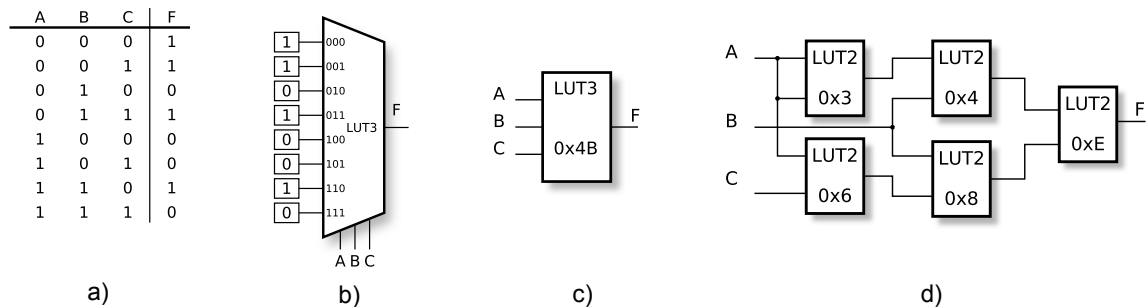
Figure 3.8: a) A Boolean function F with three input variables implemented using b), c) 3-input lookup table (LUT3) and d) a cascade of 2-input lookup tables (LUT2).

technology mapping. In contrast, the k-input LUT can implement 2^{2^k} different functions; e.g. 16,384 functions for LUT having 4-inputs only. Restriction of the set of Boolean sub-functions considered during the synthesis process often results in poor synthesis results for a large class of functions. For LUT-based FPGAs, it is extremely important to consider all available Boolean functions. Therefore, there has recently been research concerning the application of the functional decomposition to the FPGA logic synthesis [32]. Although this domain could be a potential application of EHW, LUT-based representation is not popular in this field since it implies huge and rough search space.

In order to address the problem of insufficient complexity of circuits evolved at gate-level, Murakawa et al. proposed a model for function-level evolution which operates with programmable function units (PFUs) arranged in a grid [134]. Each unit can perform one of high-level functions, such as addition, subtraction, multiplication, division, sine, cosine, constant generation and if-then. The selection of function to be implemented by PFU is determined by a corresponding gene in chromosome. Neighboring columns of PFUs are interconnected by configurable crossbar switches. The output of PFU can be fed only into the input of a PFU located in the next neighboring column. The authors utilized a variable length encoding. Each gene consists of a string of integers that identify a particular PFU, the function of PFU and addresses of input operand(s). Figure 3.9 shows part of the model that is configured using the following chromosome:

$$(1, sin, Y)(2, cos, X)(3, add, X, Y) \cdots (6, mul, 1, 2)(7, if-then, 1, 3, X, 2)(8, sin, 1) \cdots$$

For example, the gene $(6, mul, 1, 2)$ specifies the hardware function executed at the top PFU in the second column (i.e. PFU's number is 6). The function is multiplication using operands produced by the first PFU (i.e., sine) and the second PFU (i.e., cosine) in the first column. The output of the sixth PFU is $sin(Y) \cdot cos(X)$.

Apart from these models, any GP-like representation such as common GP, CGP or LGP can utilize complex computing blocks instead of gates. For example, CGP can easily be modified to evolve larger circuits at function-level [157]. Instead of gates and single-wire connections, application specific functions and multiple-bit connections can be employed. The advantage is that while the size of chromosome is similar to the gate-level evolution, the size of phenotype can be arbitrarily large, depending on the building blocks used. However, there are several issues concerning the use of common CGP in this domain. For example,
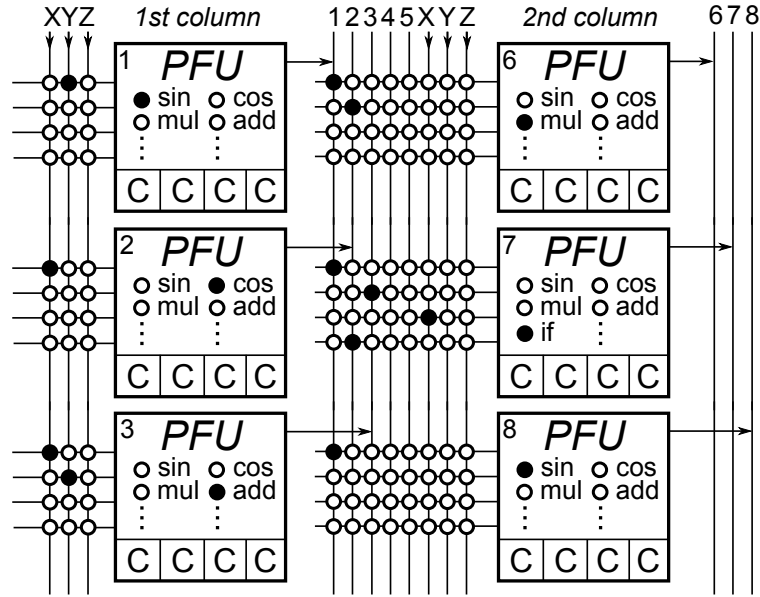
Figure 3.9: The evolutionary function-level model proposed by Murakawa [134]. A part of the grid having three primary inputs denoted as X, Y and Z is shown.

Kalganova proposed an extension of CGP that addresses the problem of multi-input multi-output building blocks typical at function-level [89].

Evaluation of Candidate Circuits

Due to the complexity of candidate circuits, it is impossible to evaluate circuit responses for all possible input vectors. Hence candidate circuits are usually evaluated using a training set. While the training set is applied during evolution, another set referred to as test set has to be used at the end of evolution. The purpose of this step is to validate the obtained results. The validation can consider various aspects, e.g. generality, robustness, functionality, etc. The goal of evolution is typically to minimize the difference between the response of a candidate circuit and the target response. The fitness value is usually calculated as the sum of the absolute deviations using the following equation:

$$fitness = -\sum_i |d(i) - r(i)| \qquad (3.5)$$

where $d(i)$ represents desired value for fitness case i and $r(i)$ is a response of a particular candidate solution to fitness case i. For m-ouput circuits ($m > 1$), the fitness value can be calculated as follows:

$$fitness = -\sum_{j=1}^{m} \sum_i |d(i,j) - r(i,j)| \qquad (3.6)$$

where $d(i,j)$ is a desired value for j-th ouput and fitness case i and $r(i,j)$ is the response of a candidate solution measured on j-th output. Alternatively mean absolute error or mean square error metrics can be used as well.

Note that this scheme can be used only when it is acceptable to evolve a circuit which responds correctly for a certain subset of all possible input vectors. The problem is that the specification is in principle incomplete. Hence this approach is not applicable for arithmetic circuits. On the other hand, in some cases it is sufficient to evaluate only some structural properties of candidate circuits which can be done with a reasonable time complexity. For example, because the testability of a candidate circuit can be calculated in a quadratic time, very large benchmark circuits with predefined testability properties were successfully evolved [143].

Let us conclude this section with a statement of Yao and Higuchi [197]. They claimed that circuits are not evolved from a conceptual viewpoint. Rather, a circuit behavior is evolved as the fitness function does not reflect the internal structure of the circuit. Hence, evolved circuits are not necessarily robust because nothing is known about their behavior in conditions different from those that have been utilized during evolution. In order to evolve robust circuits, some non-behavioral requirements have to be introduced [175]. This yields to the multi-objective approach to the fitness calculation.

Circuits Evolved at Function-Level

The first attempt to apply the function-level evolution in order to evolve large designs has been reported in [134]. Murakawa et al. used the function-level EHW based on FPGA to create an adaptive filter for digital mobile communication, and non-linear prediction functions for lossy data compression system. Another application of function-level EHW is autonomously reconfigurable and evolvable neural network chip [78].

Sekanina extended CGP in order to handle the function-level evolution [157]. As a proof of concept, evolutionary design of image filters has been chosen as a demonstration application. The objective was to design a complete structure of an image filter. The target filters could be composed of simple digital components such as logic gates, adders and comparators. Several Gaussian noise filters have been evolved. Later, image filters for other types of noise and edge detectors were evolved using the same technique [158, 161].

Aoki et al. introduced graph-based evolutionary optimization technique called Evolutionary Graph Generation [10]. Instead of creating bitlevel circuits directly, the proposed EGG system generates arithmetic data-flow graphs that can be transformed into actual bit-level circuit configurations. The advantages of this method were demonstrated through experimental synthesis of arithmetic circuits at different levels of abstraction. Several instances of competitive 16-bit constant multipliers consisting of word-level arithmetic components (such as one-bit full adders or one-bit registers) were evolved.

The most complex circuits have been evolved using a generalized disjunction decomposition introduced by Stomeo et al. in [172]. Among others, 17-bit parity circuit, the 6-bit multiplier, and a circuit with 14 inputs and eight outputs have been evolved using function-level approach (multiplexers and common gates have been utilized as basic building blocks). However, while the method is successful if the number of evaluations is measured, it produces inefficient implementations with respect to the number of gates. Another problem is that the decomposition strategy is a kind of domain knowledge which has to be supplied by designer.

She proposed an EHW chip containing an array of 6×8 4-input LUTs [166]. The objective was to accelerate the evolution of combinational and sequential circuits. In order to evolve larger circuits, he utilized a kind of decomposition method. The evolved circuit is represented using 4-bit LUT tables, multiplexers and eventually D flip-flops. The experimental results showed that this decomposition technique requires fewer generations to evolve fully functional solutions, reduces the time for an experiment, and allows the evolution of large circuits. Among others, a 5-bit multiplier (412 LUTs), 6-bit adder (2516 LUTs) and 14-input circuit CM162 (4667 LUTs) have been successfully evolved.

Shanthi put together previously published modular developmental approach [165] together with partitioning in order to demonstrate that the combined method is able to handle large digital circuits [164]. Even if a 5-bit multiplier has been presented only, is has been shown that compared to the direct evolution technique, the proposed technique reduces the time of evolution five times and improves the area by 5% - 50%.

In addition to the combinational circuits, several synchronous finite state machines having up to 20 states have been successfully evolved at the function-level [135].

3.3 Practical Aspects of the Evolutionary Design of Digital Circuits using CGP

As it has been shown in the previous chapter, genetic programming and its variants have been successfully applied to solve many difficult problems. However, the computational power which the evolutionary approaches need for obtaining satisfactory or innovative results is usually enormous. For example, Koza utilized two clusters of workstations, 1000 x Pentium II/350 MHz processor and 70 x DEC Alpha/533 MHz processor. According to the reported results, approximately 82 hours and 129 generations is needed in average to reaching a solution for 36 analog circuit design tasks solved using GP on the clusters [107]. In case of intrinsic as well as extrinsic evolution, evolutionary system usually spends most of time by running domain-specific simulators which evaluate candidate individuals using large training sets. In order to reduce the computational time, various methods have been proposed.

The parallelization on clusters of workstations represents the most applied approach as it does not require any significant change of source code written for a common workstation. Due to the stochastic nature of evolutionary algorithms, it is usually necessary to execute tens or hundreds of independent evolutionary runs in order to find a satisfactory solution. In this context, the evolutionary approach scales linearly as each workstation can execute a single evolutionary run. A similar approach can be adopted for parallel variants of GP.

In this chapter, we will present and discuss various techniques that can be applied in order to accelerate the evolutionary design technique based on CGP.

3.3.1 Simulators for Circuit Evolution

In contrast with other approaches, the main advantage of CGP is the fixed-length encoding that allows to implement the process of fitness evaluation efficiently not only in software but also in hardware.
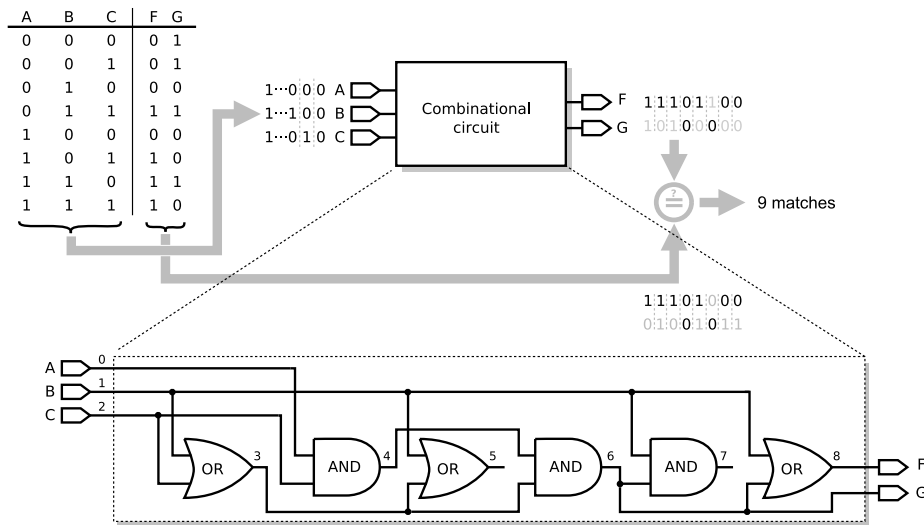
Figure 3.10: Principle of fitness evaluation of a candidate combinational circuit represented using CGP. The candidate circuit has three inputs and two outputs. The circuit is encoded using the following chromosome: (2,1,1)(2,0,0)(3,1,1)(4,3,0)(1,6,0)(1,6,1)(8,6). The set of building blocks includes three common logic gates $\Gamma = \{$AND (0), OR (1), XOR (2)$\}$. As there are 9 matches, the fitness value of the candidate circuit is equal to 9.

Figure 3.10 shows the principle of fitness evaluation when the problem of evolutionary design of digital circuits is considered. The goal is to design a combinational circuit having three primary inputs and two primary outputs. The specification is given in the form of a truth table. The test cases are successively applied to the primary inputs, the candidate circuit is simulated and the calculated response is compared with truth table. If the calculated response matches the specification (i.e. the required output is equal to the required response given by the truth able), the fitness value is incremented. As soon as all test cases are applied, the fitness value holds the number of correct responses.

In order to maximize performance, it is important to simulate candidate circuits effectively. Simulators that are utilized for circuit evolution can be divided into four classes: *interpreted simulators*, *high-level simulators*, *native simulators* and *hardware-based simulators*. The simulators, their simulation processes as well as particular examples are summarized in Figure 3.11.

Interpreted simulation

In case of the interpreted simulators, the simulated circuit is represented using an intermediate language or intermediate code. The simulator can be regarded as a virtual machine that successively executes the instructions of the intermediate language, one instruction at a time. The effect of executing the interpreted object code creates the behavior of the circuit. Any variant of a genetic programming including CGP can be easily evaluated using this approach as the genotype in fact represents a code for interpreter. In comparison with other approaches, the interpreted code is portable. The code can run on any machine that has the same interpreted simulator. However, the interpreted simulation represents the

slowest approach compared with the other three kinds of simulators, because there must be an extra layer of execution on top of the native machine that executes the simulator.
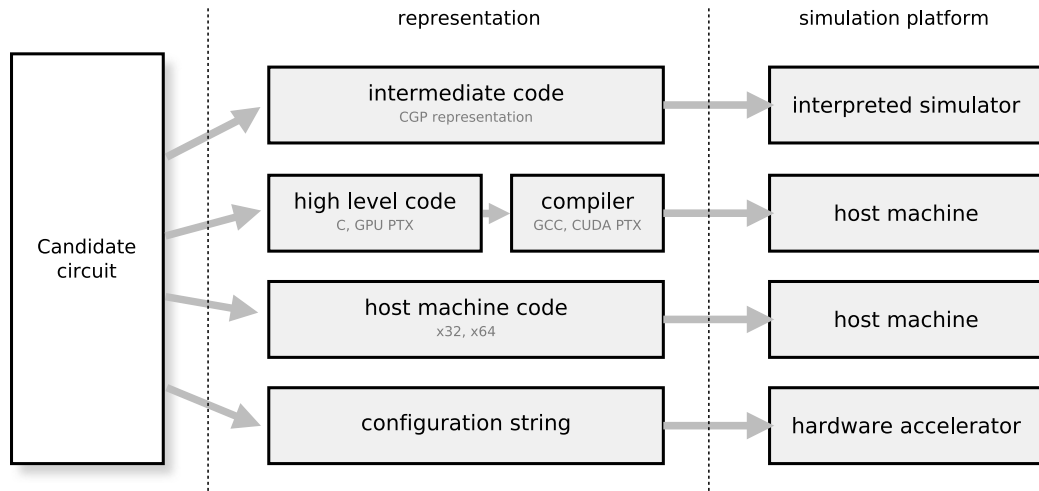
Figure 3.11: The common types of simulation processes that have been applied in the field of evolutionary design of digital circuits

The advantage of CGP encoding is that it can be directly used as an intermediate code that is consequently processed by an interpreter. Two types of interpreters are usually utilized. The interpreter based on recursion and linear interpreter. In the first case, the encoded graph structure is executed by recursion, starting from the output nodes down through the functions, to the input nodes [70]. In this way, the unconnected nodes are not processed and do not affects the performance of the evaluation. The calculated values are stored in local stacks and propagated upwards. For efficiency, it is appropriate to introduce some caching mechanism and evaluate each node only once even if such a node is shared and connected multiple times. Note that each output has to be calculated using its own recursion descent. This is another reason why the caching is important and should be introduced. The linear interpreter works in opposite direction. The execution of the encoded graph starts from the first node and continues according to the increasing node index. Providing the CGP encoding does not allow feedback loops, this execution scheme guarantee the calculated output values to be correct. This scheme represents the most efficient implementation as it does not introduce any overhead due to function calling that have to manipulate with stack. However, all the nodes are evaluated even if they are not connected. In order to improve the performance, a simple preprocessing step that marks the utilized nodes only can be introduced. Let us assume the goal is to evaluate a candidate circuits having n_i primary inputs encoded using n_n CGP nodes. To simulate a candidate circuit, one array consisting of $n_i + n_n$ items is needed. In fact, this array stores the calculated output value for each node. Thus, it can be directly addressed by the indices stored in chromosome. In contrast with the recursive approach, all the output values are calculated in one pass. Both of these interpreters are applicable for absolute [131] as well as relative CGP encoding [70].

Figure 3.12 depicts the interpreted simulation process for both presented approaches.

$F = OR^8(B,$
$\quad AND^6($
$\quad\quad AND^4(A,C),$
$\quad\quad OR^3(B, C)$
$\quad)$
$)$

$G = AND^6($
$\quad AND^4(A,C),$
$\quad OR^3(B, C)$
$)$

$t0 = A, t1 = B, t2 = C$

$t3 = OR^3(t1, t2)$
$t4 = AND^4(t0, t2)$
$t5 = OR^5(t1,t3)$
$t6 = AND^6(t4,t3)$
$t7 = AND^7(t1, t6)$
$t8 = OR^8(t1, t6)$

$F = t8, G = t6$

a)   b)
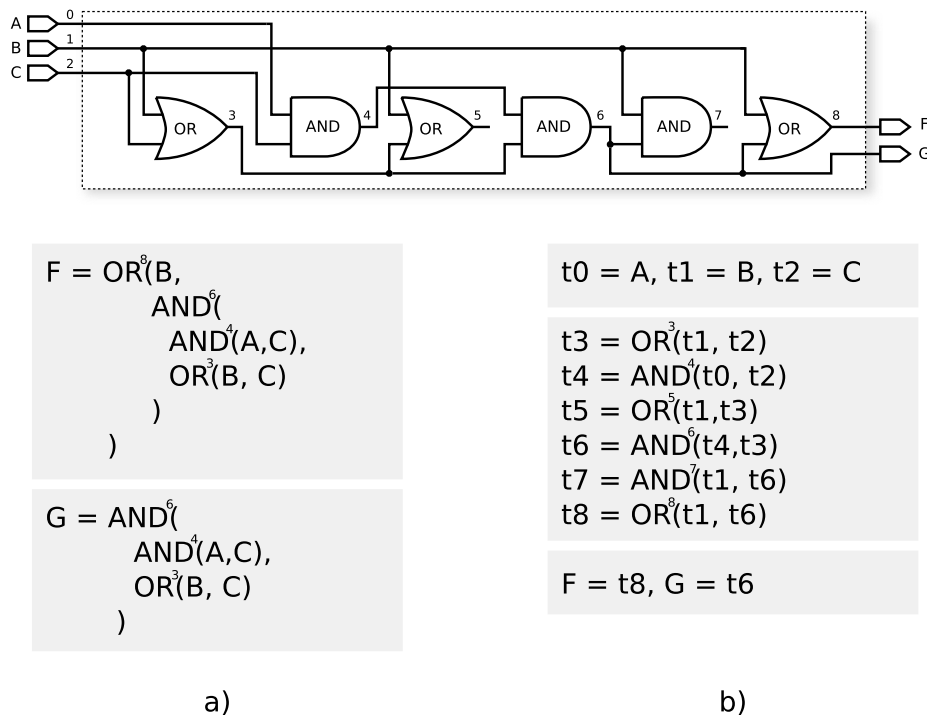
Figure 3.12: The code that have to be evaluated in order to simulate the candidate circuit from Figure 3.10 using a) recursive interpreter b) linear interpreter.

For the simplicity, none of the discussed optimizations is considered. The illustration contains the instructions represented by logic operations that have to be executed by the interpreter, one by one, in order to simulate the given circuit.

High-level simulation

Another common approach is to use a compiler that can compile a circuit into a high-level language such as C. In order to evaluate the response, the generated C description supplemented with the evaluation procedure is compiled with a common C compiler and is run just like any other C program. From the view of portability, the high-level code is not as portable as interpreted code because it needs to be recompiled to the native language of the platform every time it is simulated. Even if the high-level simulators exhibit better performance in contrast with the interpreted simulators, the compilation time may represent a bottleneck of the whole system. If the high-level code is large, the compilation time may be extremely long. On contrary, for the small designs, the compilation time is usually larger than the time needed for evaluation. Another penalty, that should be also considered is the time needed to launch the compiled program.

High-level simulator and interpreted simulator represent two possible methods for implementing genetic programming on GPUs [72]. In order to calculate the fitness values, the candidate solutions are converted to some form of source code that is compiled and executed on GPU. For instance in [73], C programs were generated from the GP individual, compiled to a GPU PTX language and then executed on the GPU. In fact, PTX code

is an intermediate code that is compiled by the graphics driver just in time when it is used. However, the results showed that the process of pre-compilation leads to a significant time overhead. The authors noticed that this approach is suitable for applications where there is a large amount of data to be processed and the evolved programs are sufficiently complicated [72].

Native-code simulation

The most effective approach that reduces the main drawbacks of the high-level compiler is the native code compilation that skips the intermediate code generation. In order to simulate a given circuit, the corresponding representation is directly translated (i.e. compiled using an application specific compiler) to the machine executable code for the given platform. The machine code residing in the same memory space can be easily executed without any time penalties by a simple call. At the expense of portability, native code runs slightly faster than high-level code because the application specific compiler can handle direct machine code optimizations. However, the overall performance is significantly higher as there do not arise any time penalties due to the complex compilation or program launching. Both native code and high-level code are typically about 20 times faster than optimized interpreted code [109]. The major shortcoming for native code compilation is portability.

The idea to use the native-code simulation in connection with CGP in the form which has been presented herein was not published in literature. However a similar idea has been used by Nordin [138]. He introduced a system based on linear genetic programming performing the automatic induction of machine code. The candidate solutions were represented directly using binary machine code and executed directly without passing an interpreter during fitness calculation. The evolved LGP program has been comprised of a sequence of 32-bit machine instructions. When executed, those instructions cause the central processing unit to perform operations on the CPU's hardware registers. The linear machine code approach to GP has been documented to be over 60 times faster when compared to an interpreting C-language implementation and up to 1500 to 2000 times faster when compared to a LISP implementation [137, 138].

Hardware acceleration

If the aim is to reduce the simulation time, the hardware-based simulators (accelerators) offer the highest degree of freedom (especially if modern FPGAs allowing to implement a custom accelerator are taken into account). In this case, the simulated circuit is compiled to a suitable representation that is downloaded to the hardware simulator. The hardware simulator is usually constructed using one or more FPGAs or GPUs that contain several processing units. As soon as the circuit is simulated, the obtained results are downloaded to the workstation. The process of compilation is usually trivial as it involves converting a genotype to a configuration string. Even if the simulations on hardware can be orders of magnitude faster than those running in software, the communication can introduce a significant overhead. There are two potential communication bottlenecks. In case that the hardware accelerator does not contain a sufficient amount of memory to store all the

test-cases, an additional penalty can be introduced due to fetching of data from a host machine. Besides that, the interactions with the host machine represent another problem. These host-hardware interactions should be minimized. Both of these problems represent a serious issue for current GPUs. Another disadvantage is a limited size of the circuit that can fit into the simulator.

The hardware accelerators can be divided into three groups: application-specific (ASIC) chips developed for a given problem [152], application-specific accelerators based on reconfigurable FPGAs [163, 180, 61, 189] and accelerators based on off-the-shelf hardware (e.g. GPUs) [73, 72].

3.3.2 Performance Improvement Using Parallel Simulation

Even if the common CPUs are since nineties equipped with the instructions that can process multiple data in one cycle, this fact has been overlooked by the EA community. One reason is that the first Single Instruction Multiple Data (SIMD) instruction sets tended to slow overall performance of the system due to the reuse of existing floating point registers. Other systems, like MMX offered a support for data types that were not interesting to a wide audience and had expensive context switching instructions to switch between using the FPU and MMX registers. Nowadays, the current systems seem to have settled down and the SSE instruction set represents a powerful and easily applicable system.

In addition, there are also some common instructions that are performed in parallel. For example a bit-wise logical instructions such as AND, OR, XOR and NOT are performed independently for all the bits in the operands; the instructions are executed in one clock cycle by concurrently activation of 32 (or 64) different logic gates within the arithmetic logic unit. In this context, a common CPU can be also seen as SIMD processor consisting of several one-bit processors.
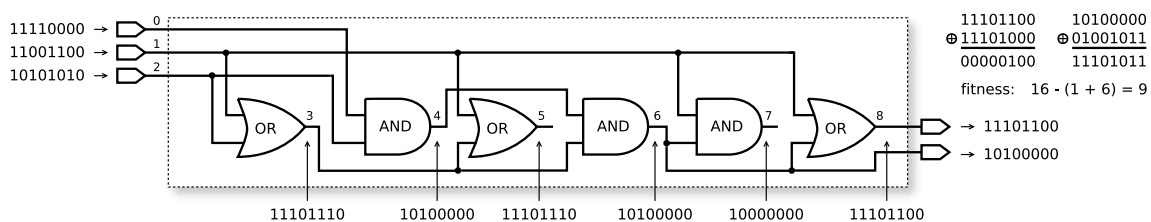
Figure 3.13: The principle of parallel simulation. The response for all eight test-cases is evaluated applying a single 8-bit word to each primary input.

The idea of parallel simulation is to utilize bitwise operators operating on multiple bits to perform more than one evaluation of a gate in a single step [127]. Using this approach, the simulators working at the gate level can be significantly accelerated. For example, when the combinational circuit under simulation has three inputs and it is possible to concurrently perform bitwise operations over $2^3 = 8$ bits in the simulator then the circuit can completely be simulated by applying a single 8-bit test vector at each input (see Figure 3.13). On the other hand when it is impossible to evaluate the vectors in parallel, eight three-bit test vectors must be applied sequentially. Current processors allow us to operate with 64 bit

operands, i.e. it is possible to evaluate the truth table of a six-input circuit by applying a single 64-bit test vector at each input. Therefore, the parallel simulation is 64 times faster than the sequential simulation. Note that in case that the circuit has more than six inputs, the speedup is constant, i.e. 64. This technique can be also utilized in hardware. However, it is mainly useful for gate-level evolution. In case of function-level evolution, for example, over b-bit operators (such as addition, subtraction, maximum etc.) the speedup is only c/b, where c is the number of bits of the operators implemented in hardware.

The parallel simulation can also be combined with native simulator that utilizes modern instruction sets such as SSE/SSE2 enabling to process four 8-bit operations in parallel or Advanced Vector Extension that manipulates with 256-bit data types. The CGP implementation that utilizes SSE/SSE2 extension was introduced in [221]. The proposed system was evaluated using a symbolic regression problem in floating point domain. Using the same 64-bit computer, the authors reported speedup between 20–117 depending on the number of training vectors and the size of CGP array.

3.3.3 Effcient Calculation of Fitness Value

In order to determine the fitness value, the Hamming distance between the obtained 32-bit (or 64-bit) response and desired output value has to be calculated. This task can be solved by applying XOR operation on the vectors and determining the number of non-zero bits in the resulting bit vector. The naive approach requires one operation per bit, until no more bits are set. For a 32-bit word, it will go through 32 iterations which is unacceptable. This task can be solved using an 8-bit lookup table and four lookups. The main drawback of this approach is that a lookup table residing in the main memory can cause additional penalties in case there is a cache miss or it might introduce some latency due to the memory operations. In order to eliminate these problems, it is possible to use some tricks performing SIMD in general-purpose registers. The following code uses a variable-precision algorithm to perform a tree reduction adding the bits in a 32-bit value:

```
unsigned int popcnt (register unsigned int x)
{
  x -= ((x >> 1) & 0x55555555);
  x = (((x >> 2) & 0x33333333) + (x & 0x33333333));
  x = (((x >> 4) + x) & 0x0f0f0f0f);
  x += (x >> 8);
  x += (x >> 16);
  return (x & 0x0000003f);
}
```

Algorithm 3.14: Population count

The operation this algorithm performs is referred to as the population count. It is based on an $O(log(n))$ algorithm that successively groups the bits into groups of 2, 4, 8, 16, and 32, while maintaining a count of the set bits in each group. The first step maps two-bit values into sum of two one-bit values, i.e. it partitions the integer into groups of two bits and computes the population count for each 2-bit group. The second step calculates the

population count of adjacent 2-bit group and stores the sum to the 4-bit group resulting from merging these adjacent 2-bit groups. To do this simultaneously to all groups, one has to mask out the odd numbered groups, mask out the even numbered groups, and then add the odd numbered groups to the even numbered groups. In the next steps, the reduction using a shift and sum approach is applied since the value in each k-bit field are small enough that adding two k-bit fields results in a value that still fits in the k-bit field. The AMD Athlon code optimization guide suggests a very similar algorithm that replaces the last three lines with multiplication (see Algorithm 3.15).

```
unsigned int popcnt (register unsigned int x)
{
  x -= ((x >> 1) & 0x55555555);
  x = (((x >> 2) & 0x33333333) + (x & 0x33333333));
  x = (((x >> 4) + x) & 0x0f0f0f0f);
  return ((x * 0x01010101) >> 24);
}
```

Algorithm 3.15: Population count optimized for AMD Athlon CPU [2]

As the population count is often needed in cryptography and other applications, Intel introduced a POPCNT instruction with the SSE4.2 instruction set extension, firstly available in a Nehalem-based Core i7 processor, released in November 2008. However, the common PCs are rarely equipped with this extension.

3.4 Current Problems of Evolutionary Design

From the view of the design automation and ability to produce novel designs, the evolutionary design represents a promising and general-purpose design method. However, there are known problems that limit the application of evolutionary approach in some domains. The scalability problem means such situation in which the evolutionary algorithm is able to provide a solution to a small problem instance, however, only unsatisfactory or even none solutions can be obtained for larger problem instances in reasonable time. During the last decade, a number of researchers have been addressing the scalability problem. Unfortunately, this issue has not been yet successfully solved [69].

In order to overcome the scalability problem, a kind of domain knowledge is usually employed in focusing the search algorithm on promising areas of the search space and reducing the computation overhead. The scalability problem can primarily be seen from two perspectives: scalability of representation and scalability of fitness evaluation.

3.4.1 Scalability of Representation

In terms of the scalability of representation, the problem is that long chromosomes which are usually required to represent complex solutions imply large search spaces that are typically difficult to search. In order to evolve large designs and simultaneously keep the size of chromosome small, various techniques have been proposed.

Variable Length Representation

Variable length representation of chromosomes provides, in contrast to the fixed length representation, more flexibility to the search algorithm. It allows for sampling a genome space with varying dimensionalities, balancing accuracy and parsimony of the solutions and the manipulation of non-coding segments [202]. Although this technique is not primarily considered as a technique for solving the scalability problem, it is incorporated at the level of genotype or phenotype in almost all methods that attempt to overcome the scalability problem.

Function-Level Evolution

Instead of gates and single-wire connections, the solution is composed of complex application-specific functional blocks (such as adders, multipliers and comparators) connected using multi-bit connections [134, 157, 60]. The advantage is that while the system complexity can be effectively increased, the size of chromosome can remain relatively small. This approach has mainly been utilized in the area of approximate synthesis where resulting innovative circuits can have tens of inputs and outputs. However, the selection of suitable functional blocks represents a domain knowledge that has to be included into the design method.

Incremental Evolution

In order to evolve more complex circuits and without the aim to minimize the number of gates, Torresen proposed a divide-and-conquer approach for the evolution of digital circuits, sometimes referred to as increased complexity evolution [176, 178]. The key idea is to decompose (e.g. according to Shanon) a target circuit on modules that are subsequently evolved separately. The advantage is that the modules are much smaller than the original circuit and so they can be evolved easily. The decomposition can also be applied recursively. Kalganova employed the incremental evolution in two directions [90]. The objective was to semi-automatically divide a complex task into simpler subtasks in order to evolve each of these subtasks and then to incrementally merge the evolved subsystems, reassembling a new evolved complex system. Generalized disjunction decomposition is the latest and most successful version of this method [172]. The 17-bit parity circuit, the 6x6 bit multiplier, and the alu4, which is a circuit with 14 inputs and eight outputs never evolved before with any other techniques, were evolved using the incremental evolution. However, while the method is successful if the time of design is measured (respectively, the number of evaluations), it produces inefficient implementations with respect to the number of gates. Another problem is that the decomposition strategy is a kind of domain knowledge which has to be supplied by designer. Incremental evolution was also combined with the function level evolution [164, 60].

Development

The above mentioned approaches employ a direct encoding of target circuit in the chromosome. Hence the size of the chromosome is proportional to the size of the circuit. As

developmental approaches employ indirect encoding, the chromosome contains a genetic program which is executed in order to construct the target circuit. The genetic program can be implemented using various computational models, e.g. L-system, cellular automaton, if-then-else rules or, in general, as a program.

Arbitrarily large multipliers were constructed using evolved programs working in a grid of programmable nodes [18]. However, no innovation is observable in evolved multipliers in comparison to conventional multipliers. A similar approach was proposed to create arbitrarily large sorting networks which exhibit slightly better properties (in the number of components and delay) than sorting networks created using conventional construction algorithms [159]. Among others, developmental genetic programming introduced by Koza [105, 107], L-system-based antenna design [83] and developmental neural networks [167] are examples of successful application of developmental approach.

Although the developmental approaches do not usually lead to innovative circuit designs, they are useful for investigation of principles of development, genetic regulatory networks, environmental interactions and fault tolerance which could be useful for evolution and adaptation of large-scale digital circuits in future. Designing these developmental encodings is not trivial and a lot of domain knowledge has to be supplied by designer.

Modularization

Some evolutionary algorithms enable to dynamically create and destroy reusable modules (subcircuits). The reuse of modules makes the evolution easier even for large circuits. Various modularization techniques have been introduced for tree-based genetic programming. Among others, the automatically defined functions represent the most popular approach [101]. Cartesian Gentic Programming was extended by utilizing automatic module acquisition, evolution, and reuse [188, 92]. Shanthi and Parthasarathi have proposed modular developmental CGP [164]. In particular, it was shown that the computational effort can be significantly reduced for small combinational circuits in comparison to standard CGP. However, evolved solutions are inefficient with respect to the number of gates.

3.4.2 Scalability of Fitness Evaluation

Even if an encoding is chosen such that it allows the candidate circuits to be represented effectively, there is another scalability problem having substantial impact on the evolutionary design of digital circuits. In case of the combinational circuit evolution, the evaluation time of a candidate circuit grows exponentially with the increasing number of inputs (assuming that all possible input combinations are tested in the fitness function). Hence, the evaluation time becomes the main bottleneck of the evolutionary approach when complex circuits with many inputs are evolved. In case of popular benchmark circuits, the limit when the evolution provides some solution is 17 inputs for parity circuits [172, 188], 8 inputs for combinational multipliers when a novel solution has been obtained [182] and 16 inputs for combinational multipliers when only functionality has been evolved [164]. The 28-input sorting networks evolved using FPGA accelerator are probably the largest circuits that were successfully evolved using the fitness function which evaluates all possible assignments to the inputs, however, only functionality has been evolved [98].

Perfect and Approximate Synthesis

From the viewpoint of the scalability of evaluation, the applications of digital circuit evolution can be divided into two main classes which we will call the *perfect synthesis* and *approximate synthesis*.

In case of the perfect synthesis the goal is to obtain a circuit which responds perfectly for all requested assignments to the inputs and which exhibits a kind of innovation such as close-to optimum number of gates, small delay or low power consumption. The fitness function is usually constructed in such a way that all requested assignments are applied to the inputs of a candidate circuit and the fitness value is defined as the number of bits that the candidate circuit computes correctly (additional criteria can be incorporated as well). The evolution of arithmetic circuits is a typical example of that class. Because the evaluation time depends exponentially on the number of inputs of the circuit there is the scalability problem inherently present. Thus, only relatively small and simultaneously innovative designs have been evolved in this domain.

In case of the approximate synthesis it is sufficient to evolve a circuit which responds correctly for a reasonable subset of all possible input vectors. The problem is that the specification is in principle incomplete. Among others, design of filters, classifiers and predictors are typical examples [78, 157, 60]. The fitness value is usually calculated on the basis of the circuit response obtained for a carefully chosen training set, a subset of all possible input vectors. Applying all possible input vectors is intractable as the circuit can have tens of inputs. This approach is commonly applied during the learning process of neural networks or classification systems. In contrast to the perfect synthesis, the behavior of evolved solution has to be validated using a test set at the end of evolution, i.e. using the vectors unseen during the evolution. To justify the approach, the evolved solution should exhibit a kind of innovation. For example, the goal can be to obtain a solution having better classification accuracy or smaller area overhead with respect to competitive solutions.

In order to assess the evolvability of functionally in case of combinational circuits, Miller and Thomson included only a randomly selected sample of all possible input combinations to the fitness function [130]. Unfortunately, it has been demonstrated that this is not a right way since the evolved digital circuits did not work correctly for the remaining input vectors.

Special Cases of Tractable Applications

Digital circuits evolved so far contain from several gates to thousands of gates. It is typical for evolution of small circuits that all possible input vectors are used in the fitness function and that the aim is to improve circuit parameters in comparison with exiting designs. Evolved mid-size circuits such as image filters or classifiers contain thousands of gates. In this case, the researchers focus on ways to reduce the number of test vectors required for evaluation whilst being able to provide a reliable evaluation method.

An obvious conclusion is that the perfect evolutionary synthesis is currently applicable only for small circuits. On the other hand, when the problem belongs to the class of the approximate synthesis, real-world and innovative circuits are likely to be evolved. However, there exist applications that do not suffer from this problem. Even if a perfect synthesis

scenario is utilized (i.e. a circuit that exactly fulfils the specification ought to be evolved), the evolutionary approach can be applied to solve large instances without having fitness scalability problems.

We have identified that linear systems represent a class of problems that can be effectively evaluated [223]. This domain has been completely ignored by EA community even if it comprises a wide variety of real-world applications. In case that the target system exhibits the linear properties, it is possible to perfectly evaluate a candidate circuit using a single input vector independently of the circuit complexity (i.e. the number of inputs, outputs or components). The system is linear if and only if it consists of linear components (functions). A linear function is a function $f : \mathcal{D} \rightarrow \mathcal{R}$ which satisfies the following two properties:

$$
\begin{array}{ll}
\text{additivity} & f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y}) \\
\text{homogenity} & f(\alpha\mathbf{x}) = \alpha f(\mathbf{x})
\end{array} \tag{3.7}
$$

for every $\mathbf{x} \in \mathcal{D}$, $\mathbf{y} \in \mathcal{D}$ and for all scalars α, where $\mathcal{D}$ and $\mathcal{R}$ are vector spaces over field K. Note that $\mathbf{x}$ and $\mathbf{y}$ are not necessarily vectors of real numbers, but can in general be members of any vector space. This is equivalent to requiring that for any vectors $\mathbf{x_1}, \dots, \mathbf{x_m} \in \mathcal{D}$ and scalars $a_1, \dots, a_m \in K$, the following equality holds

$$
f(a_1\mathbf{x}_1 + \cdots + a_m\mathbf{x}_m) = a_1 f(\mathbf{x}_1) + \cdots + a_m f(\mathbf{x}_m) \tag{3.8}
$$

The concept of linearity can be easily extended to linear operators because the operators represent in fact a mapping from one space to another. This idea can be also utilized at gate-level. In Boolean algebra, a linear function is a function f for which there exists vector $a_0, a_1, \dots, a_n \in \{0, 1\}^{n+1}$ such as

$$
f(b_1, \dots, b_n) = a_0 \oplus (a_1 \wedge b_1) \oplus \dots \oplus (a_n \wedge b_n) \tag{3.9}
$$

for all $b_1, \dots, b_n \in \{0, 1\}^n$. The easy way to determine the linearity of Boolean operations is based on the looking into the truth table. An operation is linear if each variable always makes a difference in the truth value or it never makes a difference. In order to determine the linearity efficiently, a spectral method based on Walsh transformation can be utilized [146].

According to the definition of Boolean linearity, there are only six linear functions of two input variables $x, y \in \{0, 1\}$. Namely, two identities x and y, two logical complements $\neg x$ and $\neg y$ and finally nonequivalence and equivalence relation $x \neq y$ and $x \equiv y$. These functions correspond with the following common two-input gates: BUF, NOT, XOR and XNOR. This set of gates is used in Reed-Muller logic [66]. It is well known that many Boolean functions which can be easily implemented using XOR gates are very inefficiently represented in canonical Boolean logic. The most extreme case represents the n-bit parity circuits which can be realized with $n - 1$ XOR gates only. On contrary, $2^{n-1} - 1$ OR gates and a large number of AND gates are required when a common logic including AND, OR and NOT gates is used [127].

The principle of linearity can be demonstrated on the evolutionary design of parity circuits using XOR and XNOR gates only. Let us assume an n-input parity circuit ought

to be designed. The common approach requires to evaluate 2^n test-cases; i.e. 1024 vectors for a 10-bit parity circuit. If the linearity is taken into account, the number of fitness cases can be reduced to $n + 2$; i.e. 12 vectors for a 10-bit parity circuit. One test-case corresponding to the case when all inputs are set to zero, another one the opposite case, i.e. when all the inputs are set to one, and the rest n input vectors matching the cases when exactly one input is set to one. The latter vectors are required in order to inform the evolution that all the inputs have to contribute to the resulting value.

The evolutionary design of multiple-constant multipliers represents another problem in which the principle of linearity can be utilized to reduce fitness evaluation time. Because the multiple-constant multipliers are composed of adders, subtractors and shifters (i.e. building blocks exhibiting the linear property), a single training vector can be utilized to perfectly evaluate the fitness value. Using this approach multipliers consisting of hundreds of gates can be evolved at function-level (see Chapter 4).

Let us conclude this section with a different problem that also belongs to the class of tractable problems – the evolutionary design of benchmark circuits. In this case it is sufficient to evaluate only some structural properties of combinational circuits. The key feature is that the evaluation of these properties can be done in a reasonable time complexity. For example, the testability of a candidate circuit can be calculated in a quadratic time. This approach has been firstly utilized in [143]. The authors showed that it is possible to evolve very large benchmark circuits with predefined testability properties.

3.5 Summary

Evolutionary circuit design is considered as a very challenging research area. The first difficulty comes from the complexity of the search space. Another difficulty is caused by the presence of very good and robust conventional design tools that have been extensively developed for many years to produce compact and efficient solutions.

There is a belief that evolutionary search works better for analogue circuits rather than digital circuits possibly due to the fact that analogue behaviors provide relatively smoother search spaces [169]. In addition, contrasting with digital design, there is no reliable set of design rules for analog circuit synthesis. As a consequence, the engineer has to rely on his own experience or intuition. Another problem is that automated design tools for analog circuits are not as developed as in the area of digital circuits.

One of the goals of the early pioneers of the evolvable hardware was to evolve complex circuits, overcome the limits of traditional design and find ways how to exploit the vast computational resources available on today's computation platforms. However, the scalability issue for evolvable hardware continues to be out of reach [69].

Although we have shown some possibilities how to improve the time needed to evaluate a candidate solution in CGP, the discussed acceleration techniques have only marginal impact on evolutionary synthesis of complex (i.e. real-world) digital circuits since the evaluation time grows exponentially with increasing number of primary inputs. Thus the only way to eliminate the fitness scalability issue seems to be introducing of new domain-specific evaluation methods.

Chapter 4

Evolutionary Synthesis of Linear Transforms

The linear transforms represent a key concept that is in some way employed in every digital signal processing (DSP) application. In order to reduce area and power requirements for embedded and mobile applications, designers of such systems use various techniques. One of the approaches is focused on the usage of effective finite-precision algorithms. Many numerically intensive applications have computations that are based on linear transforms such as convolutions, the discrete Fourier transform, the discrete cosine transforms, etc. Mathematically, they consist exclusively of additions and multiplications by constants. The algorithms usually involve a large number of multiplications of one variable with several constants. A proper optimization of this part of the computation, referred to as multiple constant multiplication problem (MCM), often results in a significant improvement not only of the performance but also the power consumption. When implemented in hardware, the multiplications by constants are often implemented by a sequence of additions and shifts. Compared to general $n \times n$-bit multipliers, this implementation is less expensive in terms of chip area and power consumption.

The design and optimization of a finite precision implementation for a given application represents a nontrivial task. The designer has to manually choose a numerically robust algorithm considering the least possible precision in the final fixed point implementation. In the second step, accuracy has to be tuned. The goal is to reduce the precision of the multiplicative constants (and thus the number of additions) without exceeding a given error constraint. This step conceals two major problems – the exponentially large number of different configurations of constant precisions and the fact that reducing the precision of one or several constants has a virtually unpredictable impact on the output error and is strongly dependent on the chosen error measure [148].

In order to simplify this process, several approaches have been proposed. For example, Breitzman proposed a system for automatic derivation and implementation of fast convolution algorithms [21]. Algorithms are presented in a uniform mathematical notation that allows automatic derivation, optimization, and implementation. Püschel et al. have proposed another approach for the domain of linear DSP transforms that is able to automate the mentioned design steps [148]. This method is suitable for automatic design

of a close to optimal implementation of a given transform if an error measure and error threshold is given. There also exists a generator for optimized software implementations of DSP transforms called SPIRAL [147]. A given problem is described in Signal Processing Language and automatically transformed using a set of rules, compiled and according to the given performance results eventually modified, recompiled and so on. The process of transformation is driven by the requirements and constraints.

In this chapter, we will introduce an evolutionary method based on Cartesian Genetic Programming that can synthesize complex instances of the MCM problem. The goal of this research is to show that the evolutionary algorithm is able to generate not only complex but also close to optimal structures even if a perfect synthesis scenario is considered. In order to eliminate the scalability problem of a candidate MCM evaluation, the linearity is exploited as it has been discussed in Section 3.4.2. Surprisingly, the proposed method is able to compete with well optimized heuristics in particular problem instances.

4.1 Theoretical Background

The aim of this section is to provide the necessary background on transform algorithms and multiplierless implementation techniques in order to put the MCM problem into the context.

Mathematically, a linear transform can be expressed as multiplication $\mathbf{y} = \mathbf{M}\mathbf{x}$, where $\mathbf{x}$ is an input vector, $\mathbf{M}$ the transform matrix, and $\mathbf{y}$ the output vector. The input vector $\mathbf{x}$ represents a sampled signal, the output vector $\mathbf{y}$ represents a transformed input signal. Surprisingly, for each transform where $\mathbf{M}$ is of size $n \times n$ there is a large number of different fast algorithms, which have similar, close to minimal cost, typically of the order $O(n \log(n))$, but have different structures and different numerical accuracies [148]. The reason for this variety lies in the recursive structure of the algorithms. For a given transform, there are various ways of computing it using other, smaller transforms. The combination of these choices leads to a combinatorial explosion as the number of algorithms grows exponentially with n.

In order to implement a linear transform in hardware, two basic building blocks are used – additions/subtractions and multiplications by constants. When multiplying by constants in hardware, costly combinational $n \times n$-bits multipliers may be avoided by replacing them with structure consisting of additions, subtractions and shifts. In most cases, the shifts can be effectively implemented using the wires. The principles of multiplierless multiplier design will be briefly introduced in the following paragraphs. Firstly, design of a single constant multiplier will be described. Then, its extended version will be discussed.

4.1.1 Single Constant Multiplication

In order to implement a DSP algorithm, each real-valued constant c is firstly replaced by its fixed point approximation $c \approx k/2^n$, where n denotes the number of fraction bits and k represents the corresponding fixed point value. As the denominator has a fixed value, the number of operations required to multiply by c is not affected by the position n of the decimal point (i.e. a fixed-point multiplication is equivalent to a multiplication by an integer

followed by a right shift) and we can restrict our discussion to integer fixed point numbers $c = k$ without loss of generality. The multiplication $y = kx$ of variable x by a known constant k can be decomposed to into additions, subtractions and shifts. The problem of finding the optimal decomposition is known as the single constant multiplication problem (SCM) [185].

A straightforward method for decomposing the multiplication into the additions and shifts can be constructed as follows. Let us assume a common binary representation of a constant k

$$k = \sum_{i=0}^{n-1} b_i 2^i, \quad b_i \in \{0, 1\} \tag{4.1}$$

where n corresponds with the number of bits used to represent a given integer k. Using this representation, the product of k and x can be computed as

$$y = kx = x \sum_{i=0}^{n-1} b_i 2^i. \tag{4.2}$$

Then, a simple method of multiplying can be constructed; for each non-zero bit b_i, one shift and one adder is issued resulting into the adder chain

$$y = kx = \sum_{i=0}^{n-1} x b_i 2^i = x b_0 + (2x b_1 + (4x b_2 + \cdots)). \tag{4.3}$$

This direct method requires as many additions as the number of nonzero bits b_i minus one, which can be as large as $n - 1$. In this sense, the effort in multiplication can be estimated through the number of nonzero bits. Statistically, the half of the digits are zeros if a binary coding is used. Thus $\lfloor (n-1)/2 \rfloor$ adders is required in average. The worst case scenario requires $n - 1$ additions. Figure 4.1a show the implementation of a multiplierless multiplier for $k = 15$.

As it can be easily demonstrated, this method does not produce an optimal decomposition. For example, the multiplication $y = 15x = 8x + (4x + (2x + 1x))$ consisting of three adders and three shifts can be implemented as $y = 15x = 16x - x$ requiring one subtraction and one shift only. To handle this issue and reduce the number of operations, signed digit (SD) representation is commonly used in both hardware as well as software [68, 124]. A constant represented in SD is expressed as

$$k = \sum_{i=0}^{n-1} b_i 2^i, \quad b_i \in \{\bar{1}, 0, 1\}, \tag{4.4}$$

where $\bar{1}$ stands for -1. This scheme recodes each sequence of m consecutive '1' digits in a normal binary representation, where $m > 1$, by an SD sequence of $n - 1$ '0' digits with prefix '1' and suffix '$\bar{1}$'. For example $15_{10} = 1111_2 = 1000\bar{1}_{SD}$. Comparing to the direct approach, the density of zeros increases to two thirds [124]. The worst case scenario, i.e. the alternating one's and zero's digits, requires $\lfloor (n-1)/2 \rfloor$ additions or subtractions. Since the SD representation is non unique, a canonic signed digit system (CSD) having the minimum number of non-zero elements is used instead.
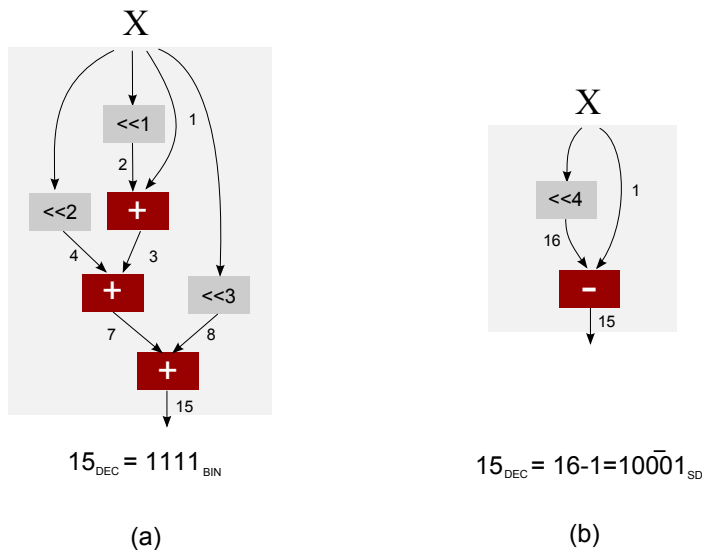
Figure 4.1: The implementation of the multiplierless multiplier for the constant 15. The structure has been designed using a) the direct encoding and b) CSD representation.
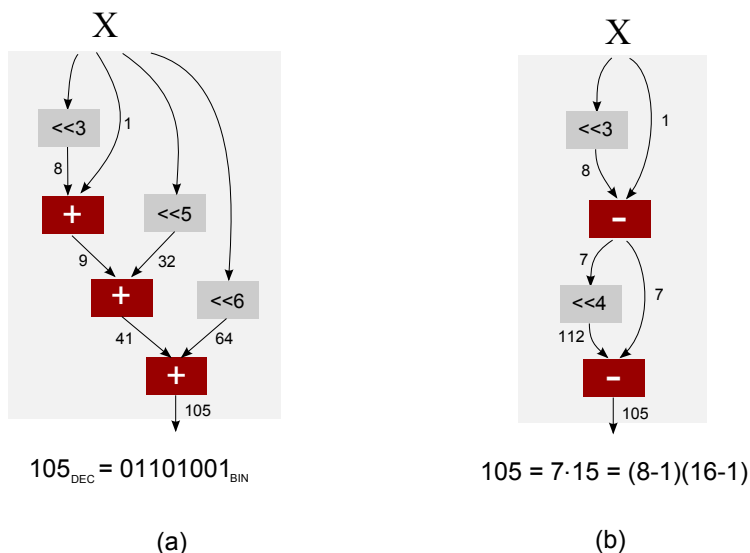
Figure 4.2: Two SCM implementations for the constant 105.

Even if the CSD system minimizes the cost, it is known that CSD in general does not yield the minimum cost solution. It can be sometimes more efficient to firstly factor the coefficient into several factors and implement the individual factors in an optimal CSD sense [42, 43, 124]. The process of factorization is illustrated in Figure 4.2 for the coefficient $105 = 01101001_2 = \bar{1}0101001_{CSD}$. In this case, the direct binary code and CSD require three additions. By factorizing, this coefficient can also be represented as $105 = 7 \times 15 = (8-1)(16-1)$. This implementation requires only one adder for each factor.

In order to find an optimal solution that minimizes the cost, a kind of reusing of intermediate results has to be introduced. However, finding the optimal addition chain for a given
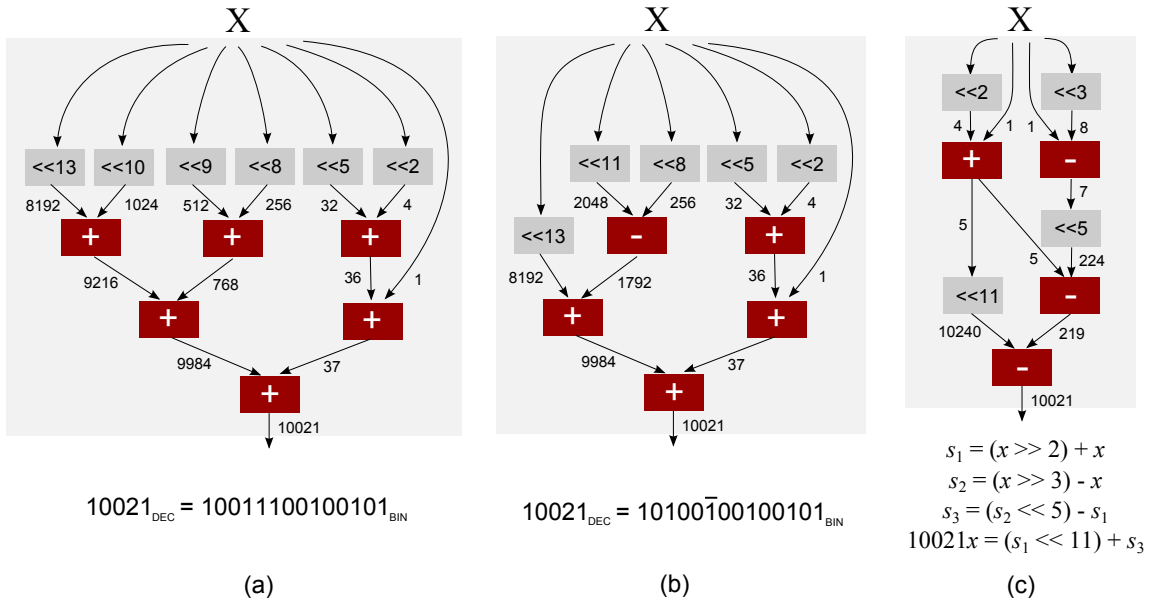
Figure 4.3: The implementation of addition chains using a) direct method, b) CSD representation and c) the optimal solution.

constant k is known to be NP-complete problem [26, 23]. For example, the addition chain to multiply x by $k = 1021$ can be implemented using 4 additions. In this case, the direct method requires 6 additions and the CSD method requires 5 additions. The corresponding implementations are summarized in Figure 4.3.

In order to find the optimal decomposition, Dempster and Macleod designed an exhaustive search algorithm that finds the optimal MCM implementation of constants defined over 12 bits [41]. This work has been extended by Gustafsson et al. for constants up to 19 bits [68]. Although the asymptotic worst-case cost of the optimal decomposition remains an open problem, is has been shown that a maximum of five additions is needed for constants of up to 19 bits.

4.1.2 Multiple Constant Multiplication

The single constant multiplication problem can be extended to the problem of multiplying a variable x with several constants $k_1, \ldots, k_N$ in parallel. The resulting structure referred to as multiplier block can be used to implement digital finite impulse response (FIR) filters (see Figure 4.4), linear signal transforms such as the discrete Fourier transform or discrete cosine transform, and so on [185]. For example, discrete Fourier and trigonometric transform algorithms involve rotations, which require simultaneous multiplication by two constants. The problem of finding the decomposition with fewest operations is known as the multiple constant multiplication (MCM) design problem [185].

Comparing to the previous problem, the design of an optimal multiplierless MCM structure is more complicated since intermediate results of the SCM decompositions may be shared. In addition to that, the optimal decomposition can not be obtained as a simple combination of optimal SCM multipliers obtained for each constant independently. Another

$$y_{[n]} = x_{[n]} * f_{[n]} = \sum_{k=0}^{L} x_{[k]} c_{[n-k]} = \sum_{k=0}^{L} c_{[k]} x_{[n-k]}$$
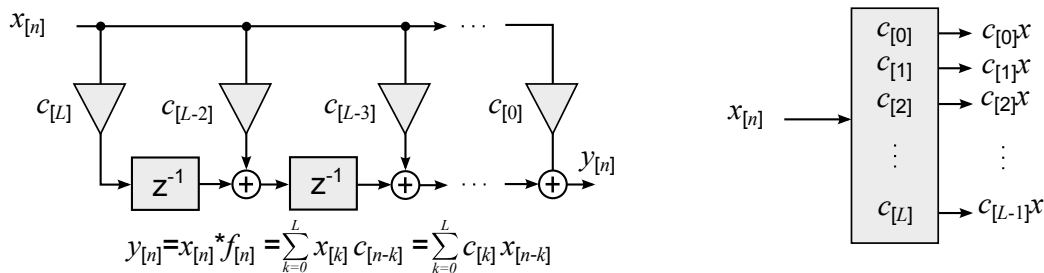
Figure 4.4: Each FIR filter can be expressed as the sum of the input sample and its delayed variants multiplied by a finite number of coefficients. This equation can be transformed and expresses as the sum of the input sample multiplied by finite number of coefficients that are successively summed [124]. FIR filter in the transposed structure (a) implemented using a multiplier block (b).

characteristics that has to be also considered is the number of levels that determines the propagation delay. Figure 4.5 shows an example of a multiplier block which implements the parallel multiplication by 19 and 71 using only 3 add/subtract operations and 3 shifts. The optimal decompositions of 19 and 71 require 2 add/subtract operations and 2 shifts each.

(a)　　　　　(b)　　　　　(c)
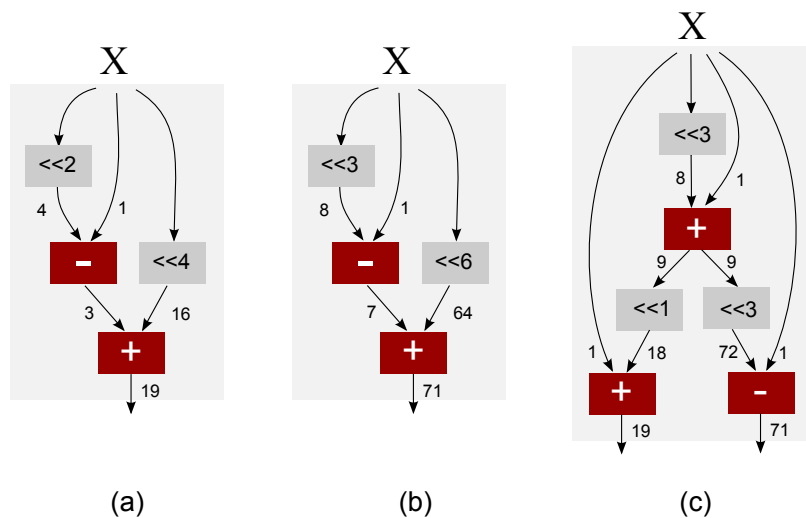
Figure 4.5: a) optimal decomposition of constant 19, b) optimal decomposition of constant 17 and c) the optimal implementation of MCM multiplier block for constants 19 and 71.

Even if the MCM problem is NP-complete, several efficient heuristics have been proposed. A good survey containing the details can be found, for example, in [185]. Apart from the direct simple methods based on CSD representation that do not provide good solutions, some authors applied common subexpression elimination algorithms. The basic idea is to find common subpatterns in representations of constants after the constants are converted to a convenient number system such as CSD. The disadvantage, however, is that the performance of these algorithms depends on the number representation. Graph-based algorithms and hybrid algorithms represent the best approaches proposed in this field. In

contrast with another heuristics, the graph-based algorithms are expected to outperform other methods, since they have the fewest restrictions. Graph-based algorithms iteratively construct the graph representing the multiplier block. The graph construction is guided by a heuristic that determines the next graph vertex to add to the graph. Graph-based algorithms offer more degrees of freedom by not being restricted to a particular representation of the coefficients, or a predefined graph topology (as in digit-based algorithms), and typically produce solutions with the lowest number of operations. A very efficient graph-based heuristic approach was proposed by Voronenko and Püschel in [185]. This algorithm can handle problem sizes as large as one hundred 32-bit constants. The algorithm can be considered as the state of the art method for the MCM design problem.

4.2 Proposed Method

The goal is to synthesize a multiple constant multiplier block which generates N output values $y_i = c_i x$ where $1 \leq i \leq N$, c_i are given constants and x is the input variable. The circuit is composed of high-level linear components such as additions, subtractions and logic shifts. The evolution is conducted at function level. In order to design a multiplier block having the minimal cost, two-stage fitness strategy is employed. At the beginning of the search, the objective of the evolutionary algorithm is to evolve a fully functional multiplier only. Once the first fully functional solution appears, an optimization phase rewarding the solutions with lower cost is conducted. During this stage, the number of components is optimized. The problem is approached using evolutionary algorithm in which the problem representation is borrowed from the CGP.
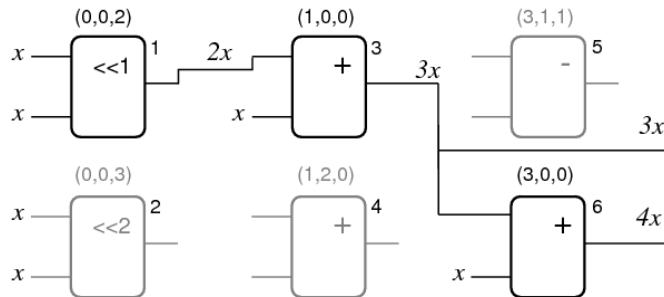
Figure 4.6: Example of a candidate MCM block. CGP parameters are as follows: $l = 3$, $n_c = 3$, $n_r = 2$, $\Gamma = \{\text{add } (0), \text{sub } (1), \text{1b-shift } (2), \text{2b-shift } (3)\}$. Nodes 2, 4 and 5 are not utilized. Chromosome: 0,0,2, 0,0,3, 1,0,0, 1,2,0, 3,1,1, 3,0,0, 3, 6. The last two integers indicate the outputs of the MCM. The input x is encoded as 0.

A candidate multiplier block is represented as an array of n_c (columns) and n_r (rows) of programmable nodes. The number of columns defines the maximum MCM delay. Each node has two inputs where at least the first input is always utilized. The number of inputs, n_i, and outputs, n_o, is fixed and chosen as follows: $n_i = 1$, $n_o = N$. Feedback is not allowed. Each node input can be connected to the output of a node placed in the previous columns or directly to the input variable x. Each node is programmed to perform one of functions defined in the set Γ which includes addition, subtraction, various shifts and

identity function. These functions as well as all connections are used over b bits, where $b = 16$ in our case. Figure 4.6 shows an example of MCM with two coefficients.

EA operates with the population of λ individuals where $\lambda = 5$. The initial population is randomly generated. The goal of EA is to minimize the difference between actual and required products. When a functionally perfect solution is obtained, the fitness function is switched to a new fitness function in which the number of components is optimized. In order to measure the similarity of a candidate solution and the required response, sum of absolute differences (SAD) is used. The fitness value is defined as

$$
fitness = \begin{cases} f_{SAD} & \text{when} \quad f_{SAD} > 0, \quad f_{SAD} = \sum_{x} \sum_{i=1}^{N} |y_i - xc_i| \\ f_C & \text{otherwise}, \quad f_C = \sum_{i=1}^{N_c N_r} cost(\mathcal{N}_i) \end{cases} \tag{4.5}
$$

where $cost(\mathcal{N}_i)$ is the cost of node $\mathcal{N}_i$. The cost function is constructed as follows:

$$
cost(\mathcal{N}_i) = \begin{cases} 0 & \text{for identity function,} \\ 1 & \text{for shift,} \\ 10 & \text{for addition/subtraction.} \end{cases} \tag{4.6}
$$

The evolution is stopped when the best fitness value stagnates or a predefined number of generations is exhausted. As it has been discussed, in theory, it is sufficient to evaluate a candidate solution using one test-case, e.g. $x = 1$. Nevertheless, especially in case when Γ contains right shifts and subtractions, the limited number of bits may introduce a kind of nonlinearity. For example, in order to implement $y = 3x$, the following structure can be evolved $y = ((x << 12) >> 11) + x$. For $x = 1$, the result is correct, however, when $x = 128$ is used, the obtained result $y = 128$ does not correspond with the expected value 384 due to the overflow caused by the left shift. In order to avoid this behavior more test cases should be used. It is usually sufficient to test x with the powers of two.

4.3   Results

In order to evaluate the proposed method, we have chosen to evolve multipliers with 3, 5, 10, 20 and 54 coefficients (given in Table 4.1). The coefficients were encoded at 16 bits. All the multipliers were evaluated using single training vector $x = 1$. The evolved multipliers were verified at symbolic level. All experiments were repeated 200 times with the population of eight individuals and five genes mutated in the chromosome. $\Gamma = \{a, a + b, a - b, 2a, 4a, 8a, \ldots, 8192a \}$. Table 4.1 gives other parameters of the experiments, average results (the number of generations and used adders/subtractors), the success rate and parameters of the best evolved solutions.

Results are compared with the best known heuristic approach [185] which produces very compact solutions. Table 4.1 shows that the proposed evolutionary-based approach is able to generate multipliers that are competitive with results obtained using the state of the art heuristic approach. The evolution can reduce the total number of components as well as

Table 4.1: Results of evolutionary design of MCMs with different coefficients. Population size is 8. Averages are calculated from 200 independent runs.

Settings		Average Results			The Best MCM			
cols × rows	maxgen	geners.	#add/sub	succ. rate	delay	add/sub	shifts	operations
3 constants: 2925, 23111, 13781								
Heuristics [185]					8	8	8	16
5×6	20M	1M62	14	68.5	5	9	8	17
6×6	20M	1M27	14	86.5	6	8	8	16
7×4	40M	2M15	13	99.0	7	8	6	14 (Fig. 4.8)
5 constants: 83, 221, 71, 387, 13								
Heuristics [185]					5	6	6	12
4×6	20M	461k	10	99.5	4	7	6	13
5×6	20M	207k	11	99.5	5	6	6	12
6×6	20M	114k	11	100.0	6	6	5	11
10 constants: 117, 1123, 743, 221, 1069, 7605, 987, 16689, 3033, 29								
Heuristics [185]					8	14	13	27
10×4	40M	4M8	23	99.0	7	15	12	27
7×6	20M	4M7	23	95.5	6	17	11	28
9×4	40M	9M5	22	91.0	9	17	9	26
20 constants: 1, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71								
Heuristics [185]					4	19	8	27
4×10	40M	457k	23	100	4	19	4	23 (Fig. 4.7)
5×10	40M	347k	23	100	4	19	4	23
6×5	40M	772k	21	100	5	19	3	22
54 constants: 1, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251								
Heuristics [185]					6	53	53	106
5×20	40M	12M9	66	98	5	56	17	73
6×14	40M	19M7	63	90	6	56	12	68
6×16	40M	9M8	65	98.5	6	55	19	74

the delay of the designed MCMs. Even if the goal of this research was to demonstrate and confirm the validity of our hypothesis concerning the linear problems, the proposed method has been able to discover better solutions in some instances. In case of the 3 constant MCM, a solution exhibiting lower delay containing fewer shifts has been evolved. In case of the 20 constant MCM, the number of shift has been reduced by one half.
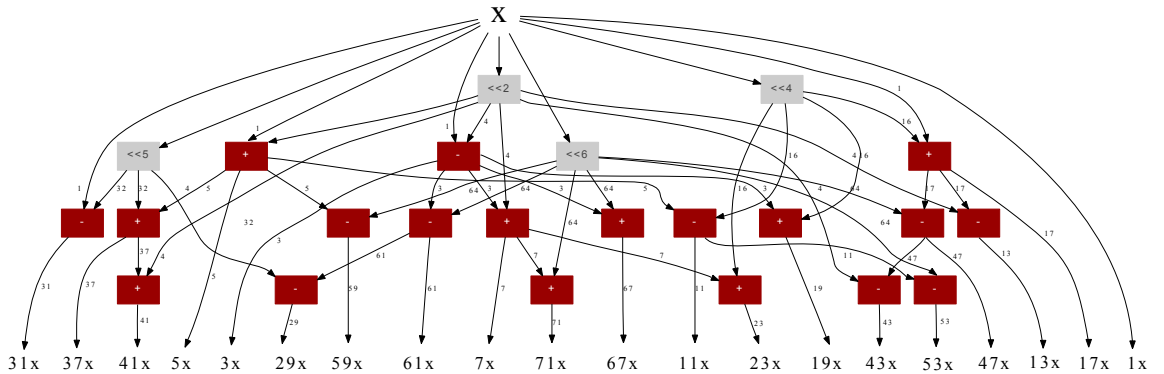
Figure 4.7: The best evolved MCM with 20 constant coefficients

Figure 4.7 shows one of evolved innovative solutions for the 20 constant MCM. This circuit consists of 4 shifters and 19 adders/subtractors. The heuristic approach [185] provides a solution consisting of 19 shifters and 19 adders/subtractors. Delay remains unchanged.
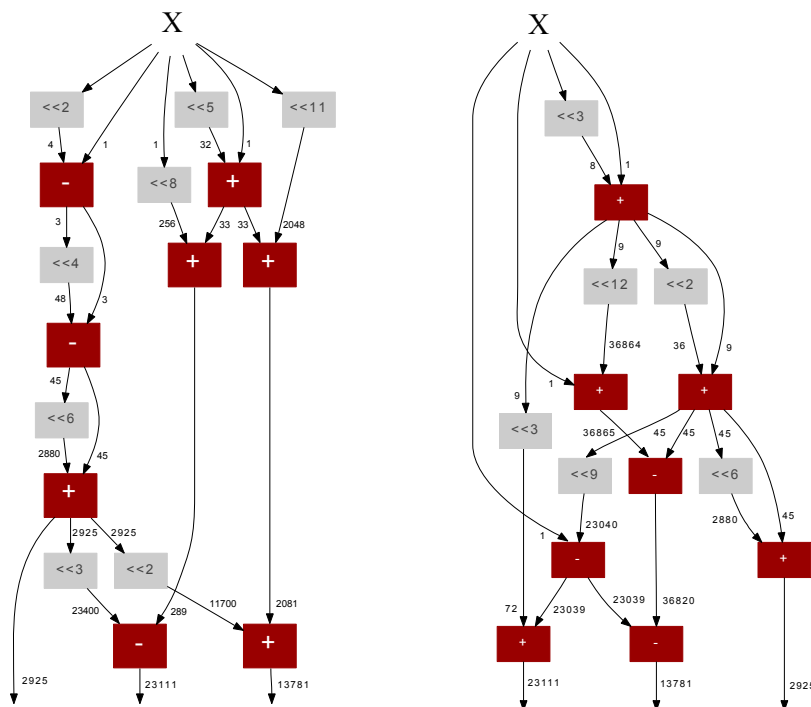
Figure 4.8: MCM with 3 coefficients (2925, 23111, 13781): according to [10] (left), the best evolved solution (right)

Figure 4.8 compares the best evolved solution with the solution provided by the heuristics for the 3 constant MCM. Evolved solution contains 2 shifters less and exhibits shorter delay than the solution provided by the heuristics.

4.4 Summary

A very time-consuming evaluation of candidate configurations is one of problems which influence the applicability of evolutionary circuit design. In this chapter, we focused on such problems in which a candidate solution can be perfectly evaluated in a short time if some domain knowledge is employed. Linear transforms in general, and multiple constant multiplier blocks in particular, belong to this class. Although well-optimized heuristics exist for linear transforms design, we confirmed that novel implementations of multiple constant multipliers can be designed using evolutionary algorithm. As the design of optimal multiplierless implementations of linear transforms is known to be NP-complete, the probability that a novel implementation will be discovered using an evolutionary approach even increases with the increasing number of constants as well as precision bits. Using this method, digital circuits with total output width higher than 850 bits have been successfully evolved.

Chapter 5

Evolutionary Synthesis of Complex Combinational Circuits

Efficient logic synthesis and optimization have been crucial for computer theory as well as computer industry for more than 50 years. Nowadays, many companies provide commercial tools that allow producing reasonable solutions (circuits) in a reasonable time. However, the recent work in the area of conventional synthesis has shown that these tools produce solutions that are far from optimum for many circuit classes [35]. Evolvable hardware community has demonstrated that very efficient implementations of digital circuits can be obtained using evolutionary computation, particularly by means of Cartesian Genetic Programming [182]. Unfortunately, the evolutionary circuit design is able to discover innovative designs only for small circuit instances (approx. up to 20 inputs and 100 gates). One of the key problems is a very time consuming fitness calculation which typically grows exponentially with increasing circuit complexity (number of inputs).

The goal of this chapter is to show that it is possible to significantly reduce the number of gates for complex circuits, too. As it will be demonstrated, very compact implementations can be obtained if the fitness calculation utilizes a formal verification algorithm to check whether a candidate circuit is functionally correct or not. In order to decide the correctness of a candidate solution, we have employed SAT-based equivalence checking. This approach translates the problem of functional equivalence of two combinational circuits to the problem of deciding whether a Boolean formula given in conjunctive normal form is satisfiable or not. We have used SAT-based equivalence checking from several reasons. Firstly, the combinational circuits represented by CGP can be converted to Boolean formula in linear time with respect to the number of CGP nodes. Secondly, as it will be shown, the SAT-based approach enables to apply several optimization techniques specific for the evolutionary design. And finally, the SAT-based equivalence checking becomes to be a preferred method as it outperforms the BDD-based approaches in many problems.

5.1   Theoretical Background

The aim of this section is to provide the necessary background on the problem of checking a functional equivalence of combinational circuits that represents the fundamental part of

the proposed method. We will firstly introduce the Boolean satisfiability problem and the existing SAT solvers. Then, the problem of functional equivalence checking will be introduced. We will briefly summarize how the binary decision diagrams are used in determining functional equivalence and outline the concept of the functional equivalence checking based on SAT. In addition to that, the last part is devoted to the conventional logic synthesis.

5.1.1 Boolean Satisfiability

Boolean satisfiability problem is a well-known decision problem consisting of deciding whether the variables of a propositional formula can be assigned in such a way that the formula evaluates to true. The research area devoted to this problem is today very active as many real-world problems can be effectively solved by transforming them to the SAT problem. However, SAT is also a typical example of NP-complete problem which means that all SAT solvers algorithms require in worst-case exponential time with respect to the size of a given instance [36]. Despite that, modern SAT algorithms are extremely effective at coping with large problem instances and large search spaces [116]. In the field of digital system design, the use of SAT has been investigated for more than twenty years and many powerful tools utilizing SAT solvers have been developed. Test pattern generation [111], identification of functional dependencies in Boolean functions [112], technology-mapping [151], combinational equivalence checking [64] or model checking [123] represent successful examples of practical applications of SAT solvers.

Most of the SAT solvers require to transform the solved problem into Boolean formula in conjunctive normal form (CNF). CNF formula φ consists of a conjunction of clauses denoted as ω_j. Each clause contains a disjunction of literals. A literal is either variable x_i or its complement $\neg x_i$. The clause can contain up to n literals providing there exists exactly n variables. Formula 5.1 contains example of a Boolean formula in CNF of three variables x_1, x_2 and x_3. The given formula is satisfiable because there exists at least one assignment that evaluates the formula to true, e.g. $x_1 = 0, x_2 = 1, x_3 = 1$.

$$\varphi(x_1, x_2, x_3) = (\neg x_1 \vee \neg x_3)(x_1 \vee x_2)(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \qquad (5.1)$$

The modern SAT solver algorithms can be divided into two groups, complete algorithms and incomplete algorithms. Most of the complete algorithms for solving the SAT problem for conjunctive normal forms are based on the Davis-Putnam procedure [40] and Davis-Putnam-Logemann-Loveland procedure (DPLL) proposed in 1962 [39]. SATO [203], Satz [113], Chaff [133], Berkmin [63], MiniSAT [51] and PrecoSAT [19] represent typical SAT solvers based on DPLL algorithm. The incomplete algorithms are based on a local search, e.g. genetic algorithms. The stochastic local search methods are used especially when there is no or limited knowledge of the specific structure of the problem instances to be solved.

5.1.2 Combinational Equivalence Checking

Determining whether two Boolean functions are functionally equivalent represents a fundamental problem in formal verification. Although the functional equivalence checking

is an NP-complete problem, several approaches have been proposed so far to reduce the computational requirement for practical circuit instances.
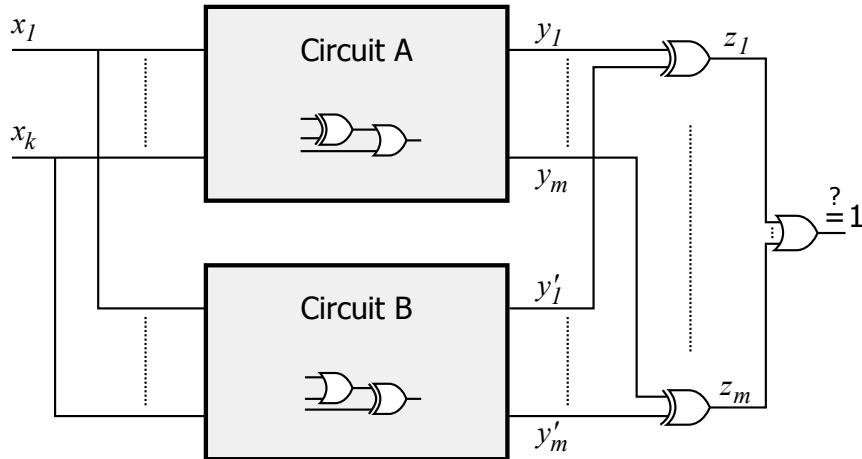
Figure 5.1: Equivalence checking of two combinational circuits using the all outputs approach. The combinational circuits are equivalent if and only if the output evaluates to zero for every input assignment.

Most of the proposed techniques are based on representing the circuit by means of its canonical representation. Generally, two Boolean functions are equivalent if and only if canonical representations of their output functions are equivalent. A brute-force method to determine combinational equivalence is to expand the combinational functions in minterm form (or in a truth table) and compare them term by term (row by row). This method represents an approach routinely used by EHW community. Clearly, this method runs into the problem of exponential size, because the number of minterms (or rows of a corresponding truth table) of a function can grow exponentially with the increasing number of input variables. In order to decide the functional equivalence problem in reasonable time, we need a representation that is both canonical and compact. However, due to the NP-completeness, it is likely that all canonical representations are exponential in size in the worst case. In spite of that, there exist representations that provide reasonable results for many practical applications.

The Reduced Ordered Binary Decision Diagrams (ROBDD) represent the widely used canonical representation in formal verification [196]. ROBDD is a directed acyclic graph that can be obtained by applying certain transformations on the ordered binary decision diagram. Determining whether two circuits represent the same Boolean function is equivalent to determining whether two ROBDDS are isomorphic. Some of methods developed to determine whether two ROBDDS are isomorphic are based on graph-based algorithms. Other methods are based on the combination of ROBDDs with the XOR operation and checking whether the resulting ROBDD is a constant node (zero) [109].

The equivalence checking using BDDs is illustrated in Figure 5.2. The objective is to decide whether a combinational circuit C_A exhibits the same Boolean function as combinational circuit C_B; both of these circuits having three primary inputs and two primary outputs. Note that only the first outputs exhibit the same Boolean function. Firstly a
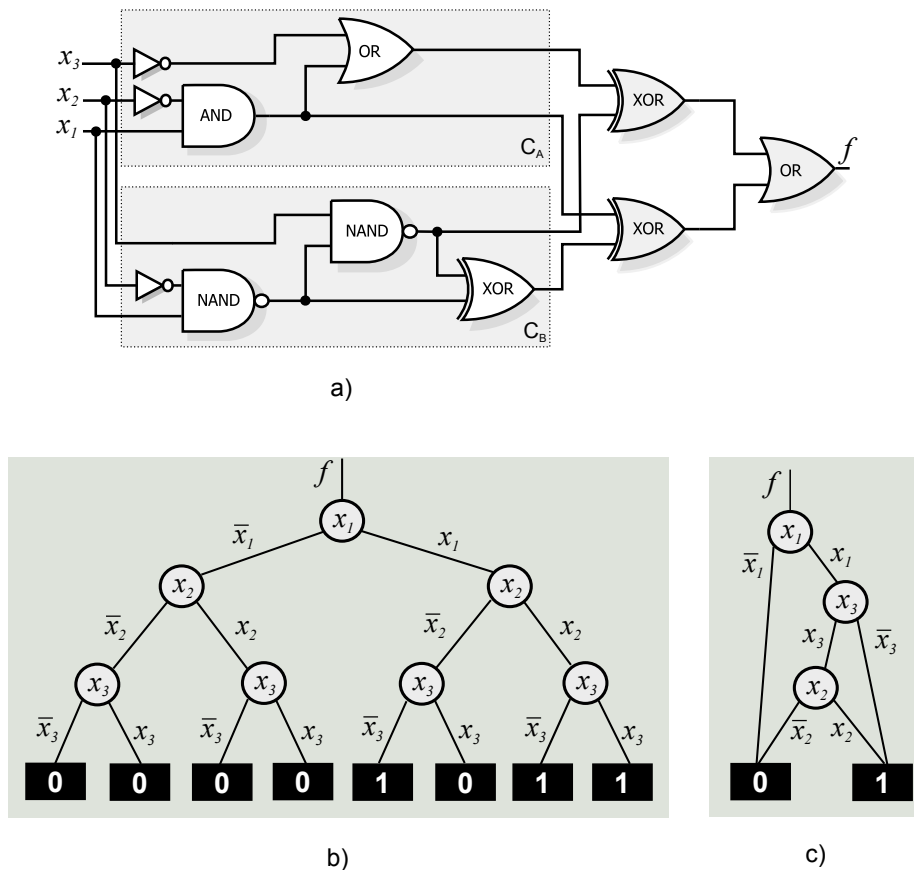
Figure 5.2: Principle of combinational equivalence checking using Binary Decision Diagrams.

miter circuit with Boolean function f is created (a). Miter consists of the checked combinational circuits combined using XOR gates. If C_A and C_B are functionally equivalent, f must evaluate to zero for each input assignment. For illustration, (b) contains common BDD while (c) contains ROBDD. Looking at the ROBDD it can be easily determined that it does not contain a single zero node; thus the circuits are not equivalent. Moreover, using the ROBDD it can be determined that the circuits gives different results for example for the assignment $x_1 = x_2 = x_3 = 1$.

All these graph-based approaches rely on the fact, that the number of nodes in the resulting graph will be relative small; otherwise, the time of the ROBDD construction as well as the time of comparison will be enormous. In practice, these methods are rarely implemented directly without any further circuit preprocessing. The main problems are the need for high memory resources due to the huge number of BDD nodes and significant time requirements. Although many functions in practice can be represented by polynomial number of BDD nodes with respect to the number of inputs, there are functions (e.g. multipliers) that always have the number of nodes exponentially related to the number of inputs [48]. The verification of such functions still represents a challenge. Note that And-or-invert graphs (AIG) represent another canonic representation with similar properties.

High consumption of memory resources has motivated the researchers to look for al-

ternative methods. Since the satisfiability solvers were significantly improved during the last few years, the SAT-based equivalence checking becomes a promising alternative to the BDD-based checking. In this case, the circuits to be checked are transformed into one Boolean formula which is satisfiable if and only if the circuits are functionally equivalent [64].

As it has been mentioned, the SAT solvers require to transform the equivalence checking problem into Boolean formula in conjunctive normal form. Unfortunately, the time complexity of translation process transforming a circuit to optimal CNF is exponential [184]. However, there exist suboptimal techniques that introduce a new variable for each logic gate having the linear complexity. For our purposes, the most suitable transformation of the circuit to CNF is represented by Tseitin's algorithm proposed in [179] that works as follows: Let us consider a combinational circuit C_A with k inputs that is composed of n interconnected logic gates. Without loss of generality, let us restrict the set of all possible gates to the following one-input and two-input gates: NOT, AND, OR, XOR, NAND, NOR and XNOR only. Let $y_i = \Omega(x_{i1}, x_{i2})$ denote a gate i of C_A with function Ω, output y_i and two inputs x_{i1} and x_{i2} ($1 \leq i1, i2 \leq k+n$). The Tseitin transformation is based on the fact that the CNF representation φ captures the valid assignments between the primary inputs and outputs of a given circuit. This can be expressed using a set of valid assignments for every gate. In particular, $\varphi = \omega_1 \wedge \omega_2 \wedge \cdots \wedge \omega_n$ where $\omega_i(y_i, x_{i1}, x_{i2}) = 1$ if and only if the corresponding predicate $y_i = \Omega(x_{i1}, x_{i2})$ holds true. During the transformation a new auxiliary variable is introduced for every signal of C_A. Hence the CNF contains exactly $k+n$ variables and the size of the resulting CNF is linear with respect to the size of C_A.

Let us assume that a common 2-input logic AND gate should be transformed to CNF. The objective is to express a Boolean function of two variables $y = AND(x_1, x_2) = x_1 x_2$ by means of CNF $\omega(y, x_1, x_2)$ that is evaluated to true if and only if the predicate $y = AND(x_1, x_2)$ holds true. The latter statement can be expressed using the implications from both directions as

$$\omega(y, x_1, x_2) = (y \Rightarrow x_1 x_2)(x_1 x_2 \Rightarrow y). \tag{5.2}$$

Using the identity $P \Rightarrow Q \equiv \overline{P} \vee Q$, the expression can be rewritten as

$$\omega(y, x_1, x_2) = (\overline{y} \vee x_1 x_2)(\overline{x_1 x_2} \vee y). \tag{5.3}$$

Finally, applying the second De Morgan's law $\overline{PQ} \equiv \overline{P} \vee \overline{Q}$ and distributive law $P \vee QR \equiv (P \vee Q)(P \vee R)$, the equation can be rewrited as

$$\omega(y, x_1, x_2) = (\overline{y} \vee x_1)(\overline{y} \vee x_2)(\overline{x_1} \vee \overline{x_2} \vee y). \tag{5.4}$$

Table 5.3 contains the CNF representation for the common logic gates. The Tseitin's transformation can be applied to any Boolean function having arbitrary number of variables. The Boolean function with multiple outputs is converted to CNF using the same approach however, each output is converted separately. Because both the size of resulting CNF and the complexity of the translation are linear, the resulting CNF does not necessary have the lowest possible number of literals. In order to improve the size of CNF, some kind of preprocessing (e.g. gate merging or subsuming of inverters) can be introduced [184, 7].
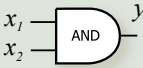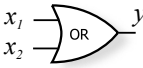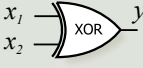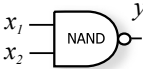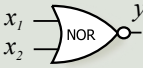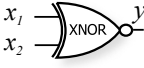
Gate	Boolean function	CNF representation
x_1 —▷○— y (NOT)	$y = \overline{x}_1$	$(\overline{y} \vee \overline{x}_1)(y \vee x_1)$
x_1, x_2 —▷— y (AND)	$y = x_1 x_2$	$(y \vee \overline{x}_1 \vee \overline{x}_2)(\overline{y} \vee x_1)(\overline{y} \vee x_2)$
x_1, x_2 —▷— y (OR)	$y = x_1 \vee x_2$	$(\overline{y} \vee x_1 \vee x_2)(y \vee \overline{x}_1)(y \vee \overline{x}_2)$
x_1, x_2 —▷— y (XOR)	$y = x_1 \oplus x_2$	$(\overline{y} \vee \overline{x}_1 \vee \overline{x}_2)(\overline{y} \vee x_1 \vee x_2)(y \vee \overline{x}_1 \vee x_2)(y \vee x_1 \vee \overline{x}_2)$
x_1, x_2 —▷○— y (NAND)	$y = \overline{x_1 x_2}$	$(\overline{y} \vee \overline{x}_1 \vee \overline{x}_2)(y \vee x_1)(y \vee x_2)$
x_1, x_2 —▷○— y (NOR)	$y = \overline{x_1 \vee x_2}$	$(y \vee x_1 \vee x_2)(\overline{y} \vee \overline{x}_1)(\overline{y} \vee \overline{x}_2)$
x_1, x_2 —▷○— y (XNOR)	$y = \overline{x_1 \oplus x_2}$	$(y \vee \overline{x}_1 \vee \overline{x}_2)(y \vee x_1 \vee x_2)(\overline{y} \vee \overline{x}_1 \vee x_2)(\overline{y} \vee x_1 \vee \overline{x}_2)$

Figure 5.3: CNF representation for the common logic gates

In order to check whether two circuits are functionally equivalent, the following scheme is usually used. Let C_A and C_B be combinational circuits, both with k inputs denoted as $x_1 \ldots x_k$ and m outputs denoted as $y_1 \ldots y_m$ and $y'_1 \ldots y'_m$ respectively. The same approach as in the previous case can be used – the corresponding primary outputs y_i and y'_i are connected using the XOR-gate. The corresponding primary inputs are connected as well. The goal is to obtain one circuit that has only k primary inputs $x_1 \ldots x_k$ and m primary outputs $z_1 \ldots z_m$, $z_i = XOR(y_i, y'_i)$. In order to disprove the equivalence, it is necessary to identify at least one XOR gate which evaluates to 1 for an input assignment, i.e. it is necessary to find an input assignment for which the corresponding outputs y_i and y'_i provide different values and thus $z_i = 1$. This can be done by checking one output after another (i.e. a CNF is created and solved for each XOR gate separately) or by the all outputs approach (all XOR outputs are connected using the m-input OR gate; thus one CNF is created and solved only). Note that both approaches are used in practice.

Figure 5.4 shows an example of equivalence checking based on SAT. The objective is to decide whether a combinational circuit C_A exhibits the same Boolean function as combinational circuit C_B. Firstly a miter circuit combining both of the checked circuits is created (a). The miter is transformed to the CNF (b) using a Tseitin's transformation. For each gate, new variable is created and a CNF according to the Table 5.3 is generated. The obtained CNF instance is solved using a SAT solver. In this case, there exists an assignment $(x_1 \ldots x_{13} = 1100000010101)$ that satisfies the CNF. It means that the circuits are functionally different.

Although the equivalence checking technology has been significantly improved during

a)

$(x_1 + x_4)(\neg x_1 + \neg x_4)$

$(x_2 + x_5)(\neg x_2 + \neg x_5)$

$(x_5 + \neg x_6)(x_3 + \neg x_6)(\neg x_5 + \neg x_3 + x_6)$

$(\neg x_4 + x_7)(\neg x_6 + x_7)(x_4 + x_6 + \neg x_7)$

$(x_2 + x_8)(\neg x_2 + \neg x_8)$

$(x_8 + x_9)(x_3 + x_9)(\neg x_8 + \neg x_3 + \neg x_9)$

$(x_1 + x_{10})(x_9 + x_{10})(\neg x_1 + \neg x_9 + \neg x_{10})$

$(\neg x_9 + x_{10} + x_{11})(x_9 + \neg x_{10} + x_{11})(\neg x_9 + \neg x_{10} + \neg x_{11})(x_9 + x_{10} + \neg x_{11})$

$(\neg x_7 + x_{10} + x_{12})(x_7 + \neg x_{10} + x_{12})(\neg x_7 + \neg x_{10} + \neg x_{12})(x_7 + x_{10} + \neg x_{12})$

$(\neg x_6 + x_{11} + x_{13})(x_6 + \neg x_{11} + x_{13})(\neg x_6 + \neg x_{11} + \neg x_{13})(x_6 + x_{11} + \neg x_{13})$

$(x_{12} + x_{13})$

b)

Figure 5.4: Principle of combinational equivalence checking using SAT.

the last years even for circuits with millions of gates, there exist some specific problems that remain to be extremely difficult. In fact, we have transformed one problem that in the worst case will take exponential time in the number of its circuit inputs into another problem that in the worst case will take exponential time in the number of its variables. For example, formal verification of arithmetic circuits, especially if multiplication is involved, represents one of these problems [168]. It is known, that multipliers lack a compact canonical representation that can be built efficiently from gate level implementations. For example, the equivalence checking of multipliers using the Binary Decision Diagrams is intractable as the number of nodes grows exponentially with the number of inputs. On the other hand, the common SAT-based combinational equivalence checking is also impractical due to the large runtime requirements; despite the compact CNF representation, the number of paths traversed by the SAT solver grows exponentially with the increasing number of inputs (see Figure 5.5). In order to improve performance of the SAT solver in this particular case, various techniques have been proposed to reduce the equivalence checking time [168, 6, 7, 142]. All approaches attempt to exploit specific knowledge about the nature of the problem under verification.
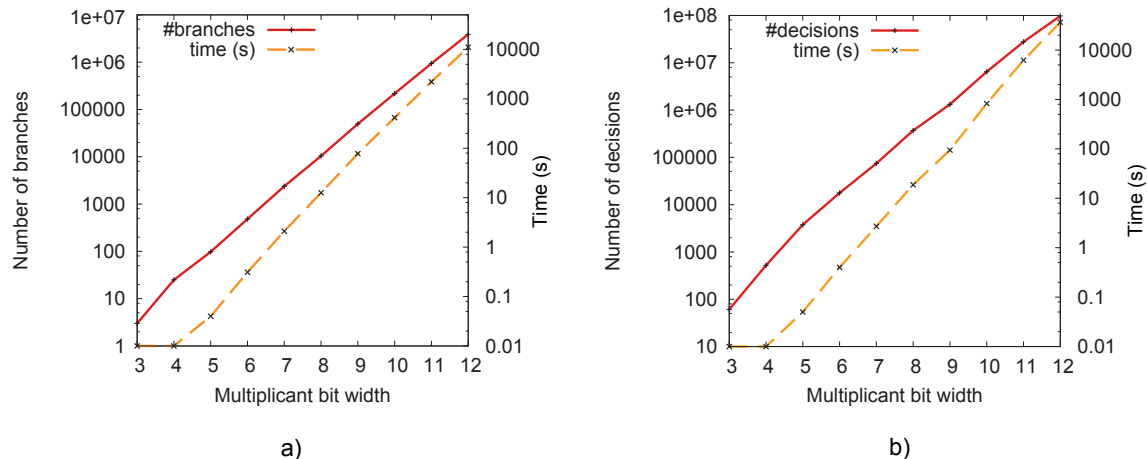
Figure 5.5: The time needed to perform the SAT-based equivalence checking for various multipliers. a) The runtime and number of branches for the Satz SAT solver. b) The runtime and number of decisions for the Minisat SAT solver with preprocessing. Note that the CNF are rather small; e.g for 11-bit multiplier the CNF uses 1400 variables and consists of 4732 clauses [88].

5.1.3 Conventional Logic Synthesis

Among others, SIS [162] and ABC [17] represent the open synthesis tools routinely used in the digital circuit synthesis community. SIS (Sequential Interactive Synthesis) is a system primary designed for synthesis and optimization of sequential circuits. It consists of variety of algorithms and tools including combinational optimization techniques. SIS can synthesize combinational, synchronous and asynchronous circuits, generating either two-level or multi-level equations. The ABC, SIS's successor, is a growing software system for synthesis and verification of combinational as well as sequential circuits. The ABC supports various representations of logic functions such as the Sum of Products or Binary Decision Diagrams, but most of the operations are performed over And-Inverter Graph representation. The basic operations include various conversions, minimizations, combinational equivalence checking, synthesis and technology mapping.

Combinational logic functions are commonly specified by PLA or BLIF files where PLA stands for programmable logic array and BLIF stands for Berkeley Logic Interchange Format. The PLA file is an abbreviated truth table where all inputs are specified. However, it does not list products for which all the outputs are zero or undefined combinations. BLIF which is a list of gates lists all interconnected single output combinational gates (and latches in case of sequential circuits). Implementations of the synthesis tools support various operations with circuits, for example, it is possible to convert PLA to BLIF and vice versa. Circuits specified in BLIF can also be mapped on a chosen set of gates, including lookup tables. Both of these representations can contain don't care symbols to simplify the specification.

The BLIF describes the digital circuit behavioral in a hierarchical manner. Combinational as well as sequential digital circuit can be viewed as a directed graph of combinational logic nodes and sequential logic elements. BLIF encodes each node using a truth table sim-

ilarly to the PLA. Figure 5.6 contains example of two possible BLIF representations that describe the same circuit consisting of five logic gates. The first case encodes the circuit using two-input Boolean functions. In the second case, the same behavior is modeled using two lookup tables (4-LUT and 2-LUT).
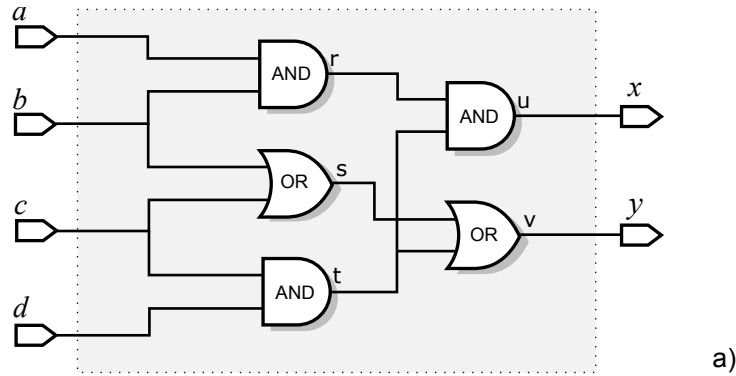
Figure 5.6: Example of two possible BLIF representations describing the same combinational function of four variables. The directive `.inputs` defines the list of input variables, `.outputs` is the list of output variables. Each `.name` directive begins a section that defines the truth table for a given list of variables. In this section, the input and output parts are separated by a space character.

The synthesis process using ABC is based on DAG-aware rewriting of a circuit represented using AIG [17]. Rewriting is performed using a library of pre-computed AIGs (command `rewrite`), or collapsing and refactoring of logic cones with 10-20 inputs (command `refactor`). According to the documentation, iterating these two transformations and interleaving them with AIG balancing (command `balance`) substantially reduces the AIG size and tends to reduce the number of AIG levels. In order to simplify the process of synthesis, ABC offers several synthesis scripts such as resyn, resyn2, and resyn2rs combining the mentioned commands. SIS contains similar commands and provides two basic synthesis scripts, script.rugged and script.algebraic. Combinational logic synthesis in ABC using resyn and resyn2 is typically 10-100x faster compared to SIS scripts giving a comparable

quality measured in terms of the number of AIG nodes and levels in the resulting network. However, the resulting number of factored-form literals is typically larger compared to SIS [17]. The synthesis is followed by technology mapping to LUTs or standard cells (command `map`).

Table 5.1: Recommended synthesis scripts for the SIS and ABC system

SIS	ABC
read PLA file	read PLA file
script_rugged	script_choice
map	map
script_rugged:	**script_choice:**
sweep; eliminate -1	fraig_store;
simplify -m nocomp	resyn; fraig_store;
eliminate -1	resyn2; fraig_store;
sweep; eliminate 5	resyn2rs; fraig_store;
simplify -m nocomp	share; fraig_store;
resub -a	fraig_restore
fx	
resub -a; sweep	
eliminate -1; sweep	
full_simplify -m nocomp	

In this work, we will use the synthesis tools with recommended (standard) setting which is represented by synthesis scripts given in Table 5.1. The synthesis is followed by technology mapping into the MCNC technology library [195] limited to 2-input gates only, for simplicity.

5.2 Proposed Method

The goal of proposed evolutionary circuit synthesis method (primarily intended for the perfect synthesis) is to automatically create complex real-world circuits that will contain fewer gates than the circuits routinely designed using conventional synthesis algorithms. Since the complete truth table evaluation is intractable for large combinational circuits (i.e. the circuits having more than 20 inputs), each candidate circuit produced by CGP is verified against the reference circuit using a SAT-based combinational equivalence checking algorithm.

According to the preliminary experiments, we have chosen MiniSAT 2 (version 070721) SAT solver [50] because it exhibits small runtime requirements, has flexible interface and can be effectively embedded into a custom application. In addition to that, MiniSAT was recently awarded in several industrial categories of the SAT competition [50].

The method consists of three steps that will be described in detail in the following sections. In the first step, the optimized circuit is synthesized using a conventional synthesis tool. The synthesis includes an optimization phase followed by mapping to the standard technology library as it has been described in the previous chapter. In this case, SIS, or its successor ABC can be utilized. In the second step, the synthesized circuit is converted

from the BLIF file format to the CGP representation. Finally, the algorithm based on CGP that employs a formal verification method to reduce the number of gates is used.

The CGP is terminated if either the maximum allowed number of generations has been exhausted or a solution that fulfills all requirements has been discovered. The initial solution (the seed) is constructed by means of mapping of the circuit obtained from conventional synthesis and specified in the BLIF format to the CGP representation. The mapping is straightforward since the CGP representation is in fact a netlist. If the initial circuit consists of m gates, each of them possessing up to γ inputs, then CGP will operate with parameters $n_c = m, n_r = 1, l = n_c, n_a = \gamma$. As it will be explained later, the initial circuit is also transformed into the conjunctive normal form in order to create a reference solution for the formal verification.

5.2.1 Formal Verification in Fitness Function

Assume that C is a k-input/m-output circuit composed of n logic gates and the goal is to reduce the number of gates. The first step involves creating a reference solution (seed) by converting C to the corresponding CNF φ_1 using the approach described above. Let $X = \{x_1, x_2, \ldots, x_N\}$ be a set containing the variables used within φ_1 and $|X| = N = k+n$. The variables corresponding with the primary inputs will be denoted as $x_1, \ldots, x_k$ and the auxiliary variables generated during the transformation process will be denoted as $x_{k+1}, \ldots, x_{k+n}$. Let the last m variables $x_{N-m+1}, \ldots, x_N$ correspond with the primary outputs of C. This step is illustrated in Figure 5.7a). The reference circuit has $k = 4$ inputs and consists of $n = 5$ gates. Thus, nine variables $x_1, \ldots, x_9$ have been created. While the variables $x_1, \ldots, x_4$ correspond with the primary inputs, the variables x_8 and x_9 refer to the primary outputs.

The following steps are used in order to calculate the fitness value of a candidate circuit:

1. A new instance of the SAT solver is created and initialized with the reference circuit C. This comprises creating of N new variables and submitting all clauses of φ_1 into the SAT solver.

2. A candidate solution is transformed to a list of clauses that are submitted into the SAT solver. The transformation includes reading the CGP representation according to the indexes of the nodes. If a CGP node contributes to the phenotype, it is converted to the corresponding CNF according to Table 5.3, otherwise it is skipped. In particular, for each node a new variable is created and a list of corresponding CNF clauses is submitted into the SAT solver. The following input mapping is used in order to form a CNF: If an input of the node situated in row i_r and column i_c is connected to the primary input i, variable x_i is used; otherwise variable x_{N+i} is used where $i = (i_c - 1).n_r + i_r$ denotes the index of the corresponding node. Let variables corresponding with the primary outputs of a candidate solution be denoted $x_{N'-m+1}, \ldots, x_{N'}$ where N' is the number of converted CGP nodes.

Note that although it is possible to include unused gates to CNF without affecting the reasoning, it is preferred to minimize the number of clauses and variables of the resulting CNF since it can decrease the decision time.

This step is depicted in Figure 5.7b). The utilized candidate circuit contains four gates that contribute to the phenotype. For each gate, a new variable $(x_{10}, \ldots, x_{13})$ has to be created. Since the CNF of the reference circuit consists of ten variables, the first variable created in this step is x_{10}. The last two variables x_{12} and x_{13} correspond with the candidate solution's primary outputs.

3. Miter circuit for combinational equivalence checking is created. The XOR gates are applied to each output pair which means that m new variables denoted as $y_1, \ldots, y_m$ have to be created and CNFs of XOR gates $y_i = \text{XOR}(x_{N-i}, x_{N'-i})$, $i = 0 \ldots m - 1$ have to be submitted to the SAT solver.

   Figure 5.7c) shows the resulting structure of miter circuit as well as the CNF clauses submitted to the SAT solver during this phase. Since the optimized circuit has two primary outputs, two new variables y_1 and y_2 have been created.

4. In order to guarantee that the resulting CNF will be satisfiable if and only if at least one miter is evaluated to 1, the outputs of the miters generated in the previous step have to be combined together. The solution is based on combining the outputs by m-input OR gate $z = \text{OR}(y_1, \ldots, y_m)$. As it can be easily derived, the corresponding CNF representation has the form of $\bigwedge_{i=1}^{k}(\neg x_i \vee z) \wedge (\neg z \vee x_1 \vee \cdots \vee x_k)$. In order to provide a CNF instance capable of the equivalence checking, it is necessary to append the clause (z) that implies $z = 1$. However, this CNF can be simplified as follows $\bigwedge_{i=1}^{k}(\neg y_i \vee z) \wedge (\neg z \vee y_1 \vee \cdots \vee y_k) \wedge (z) = (y_1 \vee \cdots \vee y_k)$ because the $z = 1$ can be propagated through the clauses. So, in order to finish the CNF, clause $(y_1 \vee \cdots \vee y_k)$ has to be submitted to the SAT solver (see last clause in Figure 5.7c).

5. Finally, the SAT solver determines whether the submitted set of clauses is satisfiable or not. If the CNF is satisfiable, it means that there exists at least one assignment of input variables for which the reference circuits gives different response. Thus the fitness function returns -1 because the candidate circuit and the reference circuit are not equivalent; otherwise the number of utilized gates is returned.

   The resulting CNF given in Figure 5.7 submitted to the SAT solver is satisfiable, thus the candidate circuit is thrown away since it receives bad fitness value. This result can be easily verified when the following input assignment $x_1 = x_2 = x_3 = x_4 = 0$ is used. For this combination, the output y_1 is evaluated to 1.

5.2.2 Time of Candidate Circuit Evaluation

In order to compare the time of evaluation for the common fitness function and the proposed SAT based fitness function, the parity circuit optimization problem has been chosen. The design of a parity circuit consisting of AND, OR and NOT gates only is considered as a standard benchmark problem for genetic programming [101]. The relevant CGP parameters are as follows: $\lambda = 4$, $\Gamma = \{\text{AND}, \text{OR}, \text{NOT}, \text{BUF}\}$, $l = N_g$, $n_c = N_g$ and $n_r = 1$ where N_g is the number of gates of the reference circuit. One gene of the chromosome undergoes the mutation only. The CGP implementation uses the parallel evaluation described in

CNF

$(x_1 \vee \neg x_5)(x_2 \vee \neg x_5)(\neg x_1 \vee \neg x_2 \vee x_5)$
$(\neg x_2 \vee x_6)(\neg x_3 \vee x_6)(x_2 \vee x_3 \vee \neg x_6)$
$(x_3 \vee \neg x_7)(x_4 \vee \neg x_7)(\neg x_3 \vee \neg x_4 \vee x_7)$
$(x_5 \vee \neg x_8)(x_7 \vee \neg x_8)(\neg x_5 \vee \neg x_7 \vee x_8)$
$(\neg x_6 \vee x_9)(\neg x_7 \vee x_9)(x_6 \vee x_7 \vee \neg x_9)$

$(\neg x_2 \vee x_{10})(\neg x_3 \vee x_{10})(x_2 \vee x_3 \vee \neg x_{10})$
$(x_3 \vee \neg x_{11})(x_4 \vee \neg x_{11})(\neg x_3 \vee \neg x_4 \vee x_{11})$
$(x_{11} \vee x_{12})(\neg x_{11} \vee \neg x_{12})$
$(\neg x_{10} \vee x_{13})(\neg x_{11} \vee x_{13})(x_{10} \vee x_{11} \vee \neg x_{13})$

$(\neg x_8 \vee x_{12} \vee y_1)(x_8 \vee \neg x_{12} \vee y_1)$
$(\neg x_8 \vee \neg x_{12} \vee \neg y_1)(x_8 \vee x_{12} \vee \neg y_1)$

$(\neg x_9 \vee x_{13} \vee y_2)(x_9 \vee \neg x_{13} \vee y_2)$
$(\neg x_9 \vee \neg x_{13} \vee \neg y_2)(x_9 \vee x_{13} \vee \neg y_2)$

$(y_1 \vee y_2)$

CNF is satisfiable if
$x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0, x_5 = 0,$
$x_6 = 0, x_7 = 0, x_8 = 0, x_9 = 0, x_{10} = 0,$
$x_{11} = 0, x_{12} = 1, x_{13} = 0, y_1 = 1, y_2 = 0$
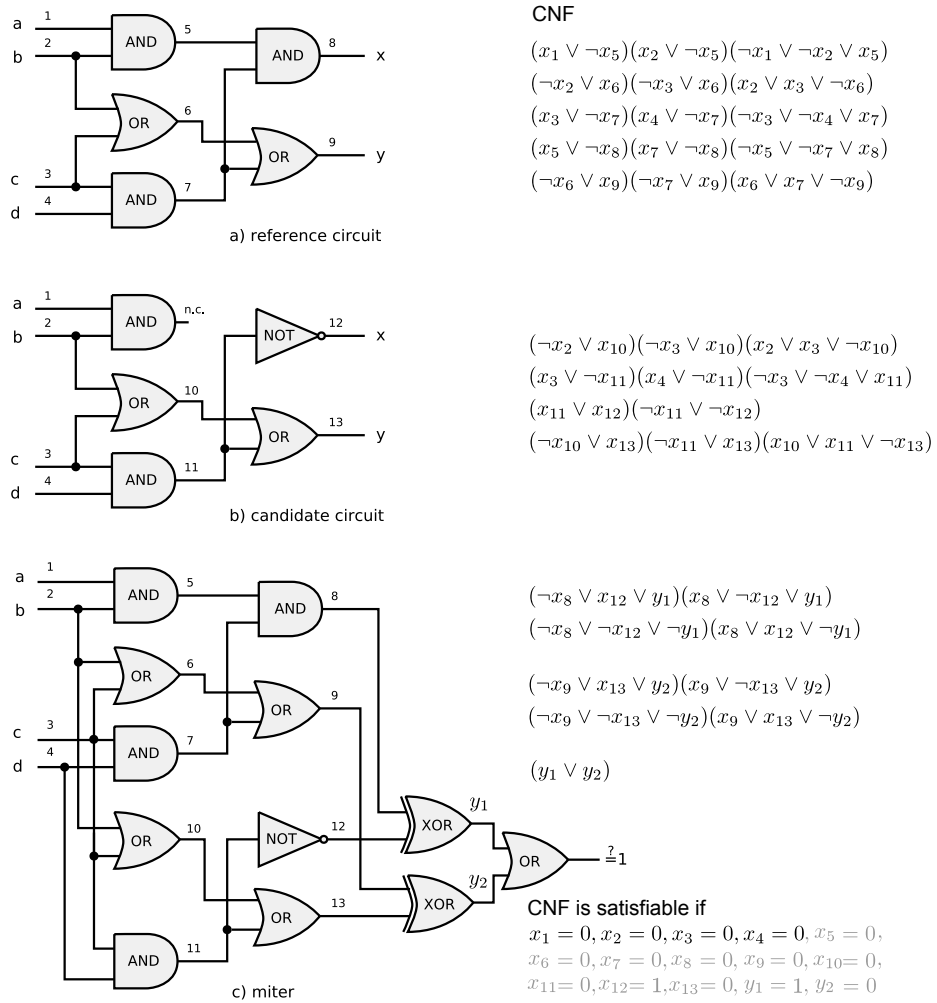
Figure 5.7: Example of transformation of reference circuit, candidate circuit and miter to CNF

Section 3.3.2. The initial circuit (seed) has been obtained by mapping a parity circuit consisting of XOR gates (parity tree) to the 2-inputs gates using ABC.

Table 5.2 gives the mean evaluation time (out of 100 runs) for three fitness functions – the standard fitness function of CGP (t_{cgp}), the optimized and accelerated evaluation (t_{ocgp}), and the proposed SAT-based method (t_{sat}). Last two columns contain the achieved speedup of proposed approach against the common CGP and accelerated CGP. Since t_{cgp} increases exponentially with the increasing number of circuit inputs, the standard CGP approach provides a reasonable evaluation time for parity circuits that contain up to 22 inputs. The optimized evaluation is applicable for up to 24 inputs. In case of the SAT-based approach the evaluation time is almost similar independently of the number of candidate circuit inputs. The experiments were carried out on Intel Core 2 Duo 2.26 GHz processor. For $n_i \geq 26$ only extrapolated values are given as running the experiments is not tractable due to the problems with enormous memory as well as runtime requirements.

The optimized and accelerated CGP implementation works as follows. Because the

71

Table 5.2: The mean evaluation time for the standard fitness function of CGP t_{cgp}, CGP with optimized and accelerated evaluation t_{ocgp} and the SAT-based CGP t_{sat} in the task of k-bit parity circuit. Symbol '*' denotes extrapolated values.

n_i	seed [gates]	t_{cgp} [ms]	t_{ocgp} [ms]	t_{sat} [ms]	t_{cgp}:t_{sat} speedup	t_{ocgp}:t_{sat} speedup
12	69	0.13	0.04	0.348	0.3	0.1
14	87	0.54	0.16	0.438	1.2	0.4
16	103	2.54	0.27	0.531	4.8	0.5
18	115	11.45	1.20	0.722	15.9	1.7
20	125	51.44	5.17	0.776	66.3	6.7
22	135	220	25.11	0.804	273.6	31.2
24	145	1328	139	0.903	1471	153.9
26	171	5962*	626*	1.028	5799	608
28	181	26748*	2820*	1.195	22383	2359
30	199	119996*	12703*	1.211	99088	10489
32	215	538327*	57207*	1.348	399352	42438

initial population already contains a fully functional solution and the elitism is implicit for CGP, there will be at least one perfectly working solution in each population. Hence we can now consider CGP as a circuit optimizer rather than a tool for discovering new circuit implementations from a randomly generated initial population. The fitness evaluation procedure which probes every assignment to the inputs (i.e., $0 \ldots 2^{n_i} - 1$ test cases) is time consuming. In order to make the evaluation of a candidate circuit as short as possible, it is only tested whether a candidate circuit is working correctly or incorrectly. In case that a candidate circuit does not produce a correct output value for the j-th input vector, $j \in \{0 \ldots 2^{n_i} - 1\}$, during the evaluation, the remaining $2^{n_i} - j - 1$ vectors are not evaluated and the circuit gets the worst possible score. Experimental results show that this technique reduces the computational overhead (see Table 5.2), but it does not significantly contribute to solving the scalability problems. Note that this technique cannot be applied for the randomly initialized CGP because we have to know the fitness score as precisely as possible (i.e. the exact number of bits has to be calculated that can be generated by a particular candidate circuit) in order to obtain a reasonably smooth fitness landscape.

5.2.3 CGP-Specific Performance Improvement Techniques

Although the system can be used directly as it was proposed in the previous section, it is possible to introduce some techniques allowing the SAT solver even to increase the performance.

The speed of the SAT-based equivalence checking depends mainly on the number of paths that have to be traversed in order to prove or disprove the satisfiability. The number of paths among others increases with the increasing number of outputs to be compared, i.e. more outputs to be compared more time the SAT-solver needs for the decision. In order to simplify the decision problem and increase the performance, CNF reduction based on finding structural similarities were proposed in literature.

In our case we can apply an elegant and simple solution. Since every fitness evaluation is preceded by a mutation, a list of nodes that are different for the parent and its offspring can be calculated. This list can be used to determine the set of outputs that have to be compared with the reference circuit and only these outputs are included into CNF. This can be achieved by omitting the unnecessary outputs during the miter creation phase.

In order to decrease the number of variables as well as the number of clauses in NOT-intensive circuits, the following approach is used. Let $y_i = NOT(x_i)$, then the NOT gate can be subsumed to CNF of every gate that is connected directly to output y_i. Using literal $\neg x_i$ instead of y_i and literal x_i instead of $\neg y_i$ respectively solves the problem.

Note that proposed approach can easily be combined with other methods designed to speedup the SAT-based equivalence checking, e.g. circuit preprocessing, incremental approach or improved CNF transformation [44, 49, 7, 184].

Table 5.3: The mean time needed to evaluate a candidate solution for plain and optimized SAT-based fitness method

circuit	n_i	n_o	seed [gates]	t_{sat} [ms]	t_{osat} [ms]	$t_{sat} : t_{osat}$ speedup
apex1	45	45	1408	49.80	15.52	3.21
apex2	39	3	235	3.54	2.52	1.40
apex3	54	50	1407	34.56	13.93	2.48
apex5	117	87	784	17.45	5.07	3.44

In order to evaluate the impact of proposed improvements, four complex circuits have been selected for experiments from the LGSynth93 benchmark set. This benchmark set includes nontrivial circuits specified in BLIF format that are traditionally used by engineers to evaluate quality of synthesis algorithms. The benchmark circuits were mapped to 2-input gates using SIS. Parameters of selected circuits as well as obtained results are summarized in Table 5.3. It can be seen that even if the circuits exhibit higher level of complexity in comparison with parity circuits, the average time needed to perform the fitness evaluation remains still reasonable. Note that the same experimental setup mentioned in the previous section has been utilized. Obtained results show that the average time needed to evaluate a candidate solution has been reduced three times in average by means of applying the proposed steps during the transformation of a candidate solution to corresponding CNF.

5.3 Evaluation of the Proposed Method

This section surveys experiments performed to further evaluate the proposed method. In particular, the effect of population sizing, CGP grid sizing, mutation rate and time allowed to evolution are analyzed for benchmark circuits. In all experiments we used the optimized SAT-based fitness function. The experiments were carried out on a cluster consisting of Intel Xeon X5670 2.4GHz processors using the Sun Grid Engine (SGE) that enables to run the experiments in parallel.

5.3.1 Population Size

Table 5.4 surveys the best (minimum) and mean number of gates obtained for $\lambda = 1$ and $\lambda = 4$ out of 100 independent runs. The number of evaluations was limited to 400,000 which corresponds with 100,000 generations for ES(1+4) and 400,000 generations for ES(1+1). The mutation operator modified 1 gene of the chromosome, $l = n_c$ and $\Gamma = \{\text{BUF}, \text{AND}, \text{OR}, \text{NOT}, \text{XOR}, \text{NAND}, \text{NOR}\}$. The best values as well as mean values indicate that ES(1+1) performs better than ES(1+4) which corresponds with the assumption of very rugged fitness landscape.

Table 5.4: The best and mean number of gates for different population sizing.

circuit	n_i	n_o	seed [gates]	ES 1+4 best	mean	ES 1+1 best	mean
apex1	45	45	1408	1240	1267	1201	1255
apex2	39	3	235	138	155	132	146
apex3	54	50	1407	1336	1350	1331	1347
apex5	117	87	784	736	746	730	743
mean			959	863	880	849	873

5.3.2 Mutation Rate and Topology of CGP encoding

Table 5.5 gives the best (minimum) and mean number of gates obtained for different mutation rates (1, 2, 5, 10, 15 genes) and CGP grid setting ($n_c \times 1$ versus $n_c \times n_r^{(i)}$). It will be seen below that the number of rows $n_r^{(i)}$ is variable. The number of evaluations was limited to 400,000 and results were calculated out of 100 independent runs of ES(1+1). Table 5.5 also includes the mean number of bits that were included to create miters and the mean time of a candidate circuit evaluation.

The best results were obtained for the lowest mutation rate. The higher mutation rate the higher mean number of gates in the final circuit. While the mean number of miters grows with increasing of the mutation rate, the mean evaluation time is reduced. This phenomenon can be explained by the fact that higher mutation rate implies more changes that are performed in circuits and thus more miters have to be considered. On the other hand, because of many (mostly harmful) changes in a circuit it is easier to disprove the equivalence for SAT solver and so reduce the evaluation time.

The settings $n_c \times 1$ or $n_c \times n_r$ do not have a significant impact on the resulting number of gates on average. Recall that the values of n_c and n_r are given by the circuit topology which is created by the SIS tool. The number of rows ($n_r^{(i)}$) is considered as variable for a given circuit in order to represent the circuit optimally. For example, the 1408 gates of the apex1 benchmark is mapped on the array of 19x189 nodes; however only 1, 5, 7, 14, 17, 26, 43, 57, 84, 117, 142, 177, 189, 187, 139, 89, 51, 27, 40 gates are utilized in columns $i = 1 \ldots 19$. The advantage of using $n_r > 1$ is that delay of the circuit is implicitly controlled to be below a given maximum value.

Table 5.5: The best (minimum) and mean number of gates, the mean number of miters and the mean evaluation time for different mutation rates (1–20 genes) and CGP grid setting ($n_c \times 1$ versus $n_c \times n_r^{(i)}$)

	mutated genes ($n_c \times 1$)						mutated genes ($n_c \times n_r^{(i)}$)					
	1	2	5	10	15	20	1	2	5	10	15	20
	apex1 - 1408x1						apex1 - 19x189					
best	1240	1290	1351	1377	1382	1393	1260	1290	1351	1379	1385	1392
mean	1269	1313	1367	1387	1396	1399	1287	1326	1369	1390	1395	1399
mean (miters)	3.8	5	8.2	12.3	15.3	17.6	3.6	4.8	8	12.2	15.2	17.6
mean t_{osat} [ms]	15.8	11.2	8.8	7.7	7.7	7.2	11.8	11.5	9.7	7.8	7.9	6.7
	apex2 - 235x1						apex2 - 22x23					
best	164	159	166	181	195	200	165	167	172	186	194	201
mean	170	172	181	195	203	209	171	174	182	195	205	209
mean (miters)	1.8	2.1	2.5	2.7	2.8	2.9	1.8	2	2.5	2.7	2.8	2.9
mean t_{osat} [ms]	1.7	1.7	1.4	1.2	1.1	0.9	1.7	1.6	1.4	1.2	1.0	1.0
	apex3 - 1407x1						apex3 - 24x193					
best	1341	1358	1383	1392	1395	1396	1345	1362	1383	1392	1396	1398
mean	1354	1369	1389	1397	1399	1400	1357	1372	1390	1397	1400	1401
mean (miters)	2.6	3.6	6.2	9.4	12	14	2.6	3.5	6.1	9.4	11.9	14.1
mean t_{osat} [ms]	10.5	10.1	9.0	11.4	8.3	8.0	10.5	10.3	9.8	8.8	9.8	7.2
	apex5 - 784x1						apex5 - 34x117					
best	740	741	755	765	767	774	741	750	757	767	768	771
mean	748	753	764	773	775	779	751	757	766	773	775	777
mean (miters)	4.6	6.4	11.1	18.1	23.7	28.4	4.6	6.4	11.2	18.1	23.7	28.4
mean t_{osat} [ms]	3.3	3.1	3.0	2.9	2.9	2.7	3.1	3.2	2.9	3.0	3.2	2.9

5.3.3 Seeding the Initial Population

In order to investigate the role of seeding of the initial population we have used two seeds obtained after 1 and 1000 iterations of the SIS script. Figure 5.8 shows that convergence curves for two selected LGSynth93 benchmark circuits – apex1 (the largest one) and ex4p (the highest number of inputs) – are very similar for those seeds. We can also observe how the progress of evolution is influenced by restarting CGP (every 3 hours; using the best solution out of 100 independent runs) which can be also considered as a new seeding. Figure 5.8 shows that repeating the synthesis scripts (SIS and ABC are compared) quickly lead to a small reduction of the circuit size; however, no further improvements have been observed in next 1 hour.

5.3.4 Parity Benchmarks

In Section 5.2.2 we compared the evaluation time of the standard fitness function and the SAT-based fitness function in the task of parity circuits optimization. Table 5.6 shows concrete results - the minimum number of gates that were obtained for 12–38 input parity circuits by running the proposed method for 3, 6, 9 and 12 hours. The results are averaged from 100 independent runs of CGP with the following setting: ES(1+1), 1 mutated
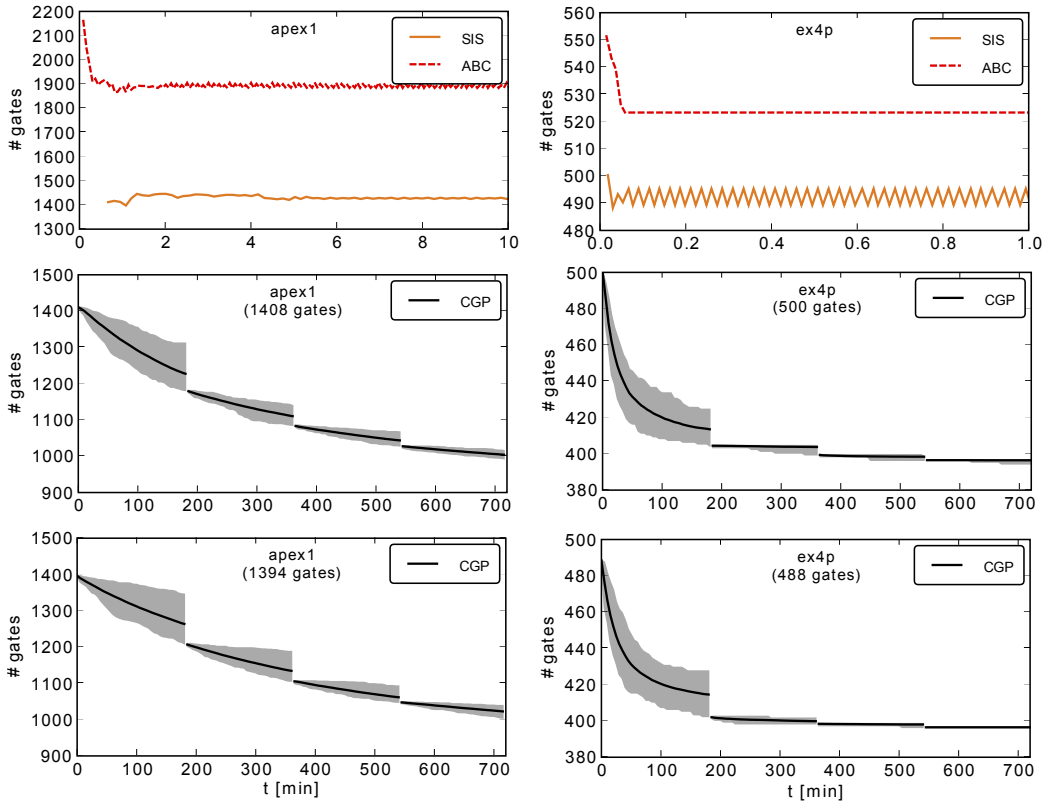
Figure 5.8: Convergence curves for the apex1 and ex4p LGSynth93 benchmarks. The mean, minimum and maximum number of gates from 100 independent runs of CGP when seeded using the result of the 1st iteration and the best result out of 1000 iterations of the SIS tool. ABC and SIS were repeated until stable results observed.

gene/chromosome, $\Gamma = \{\text{BUF}, \text{AND}, \text{OR}, \text{NOT}\}$, and CGP array of $n_c \times 1$ nodes where n_c is the number of gates in the seed – the initial circuit created by SIS. Column TG denotes the number of gates of the optimal solution which is known in this case. It can be calculated as $4w$ where w is the number of XOR gates in the optimized parity tree and 4 denotes the number of gates from Γ needed to form a single XOR gate.

We can observe that the proposed method provides an optimal solution for $n_i \leq 20$ and almost optimal solution for larger problem instances. Last column shows that the proposed method improves the original solution of SIS by 28–42 %.

5.3.5 LGSynth93 Benchmarks

Table 5.7 shows the minimum and mean number of gates that were obtained for real-world benchmark circuits of the LGSynth93 suite (we have selected those with more than 20 inputs) by running the proposed method for 3, 6, 9 and 12 hours. The results are averaged from 100 independent runs of CGP with the following setting: ES(1+1), 1 mutated gate/chromosome, $\Gamma = \{\text{BUF}, \text{AND}, \text{OR}, \text{NOT}, \text{XOR}, \text{NAND}, \text{NOR}\}$, and CGP array of $n_c \times 1$ nodes where n_c is the number of gates in the seed circuit. The initial circuit was obtained by converting the PLA files of LGSynth93 circuits to the 2-input gates of Γ and

Table 5.6: The minimum number of gates that were obtained for parity circuits by running the proposed method for 3, 6, 9 and 12 hours. TG gives the optimum solution.

n_i	seed [gates]	run-time				TG [gates]	relative improv.
		3h	6h	9h	12h		
12	69	45	44	44	44	44	36 %
14	87	54	53	52	52	52	40 %
16	103	64	61	60	60	60	42 %
18	115	74	70	69	69	68	40 %
20	125	82	79	77	76	76	39 %
22	135	95	91	88	87	84	36 %
24	145	110	101	98	96	92	34 %
26	171	134	120	114	111	100	35 %
28	181	151	132	124	121	108	33 %
30	199	165	140	132	129	116	35 %
32	215	186	169	159	143	124	33 %
34	227	214	187	172	160	132	30 %
36	237	220	192	168	162	140	32 %
38	247	235	219	193	177	148	28 %

optimizing them by SIS. Last column shows that the proposed method improves the original solutions obtained from SIS by 22–58%.

5.4 Improved Equivalence Checking

Although the SAT-based equivalence checking applied in the fitness function allows to optimize large logic circuits using genetic programming, there exist circuits for which the runtime of state-of-the-art SAT solvers grows exponentially with the increasing size of the problem instance.

One of the hard cases is the equivalence checking of the combinational multipliers where the time needed to decide whether two multipliers are functionally equivalent is enormous even for instances with operands of modest size. In order to improve the performance of SAT solvers in this particular case, various techniques have been proposed in literature. A common goal of the proposed techniques is to modify (preprocess) the input CNF instance in order to decrease the proving effort of the SAT solver. For example, a preprocessing tool which derives implications according to the computed implication graph is proposed in [7]. The implications are then used to reduce the verification time. Although this tool is able to handle the multipliers having 32-bit operands, the run-time is enormous (5.5 hours).

In order to shorten the decision time and improve the performance of the evolutionary approach, an enhanced method has been proposed. The knowledge of the dissimilarities between the reference circuit and checked (i.e. candidate) circuit is applied to reduce the size of CNF instance. Comparing to the previous approach, this method does not require additional reference circuit since a parental circuit serves simultaneously as a reference. It also means that as the size of the optimized circuit gets smaller, the CNF derived from the reference requires fewer clauses. Thus, the performance has been improved at two different

Table 5.7: The minimum (even rows) and mean number (odd rows) of gates for LGSynth93 circuits obtained from the proposed method after 3, 6, 9 and 12 hours.

circuit	n_i	n_o	seed [gates]	run-time 3h	6h	9h	12h	relative improv.
apex1	45	45	1408	1179	1083	1026	990	30 %
				1230	1108	1042	1001	29 %
apex2	39	3	235	104	101	99	98	58 %
				119	102	100	98	58 %
apex3	54	50	1407	1280	1223	1189	1167	17 %
				1333	1240	1202	1175	16 %
apex5	117	87	784	675	649	640	633	19 %
				692	661	644	636	19 %
cordic	23	2	67	32	32	32	32	52 %
				33	32	32	32	52 %
cps	24	109	1128	870	788	737	698	38 %
				909	806	757	713	37 %
duke	22	29	430	286	274	270	268	38 %
				296	279	272	269	37 %
e64	65	65	192	133	130	129	129	33 %
				139	131	129	129	33 %
ex4p	128	28	500	404	399	396	394	21 %
				414	401	397	395	21 %
misex2	25	18	111	76	73	72	70	37 %
				82	74	72	71	36 %
vg2	25	8	95	79	75	74	74	22 %
				83	77	74	74	22 %

levels – by applying an approach that uses a variable reference and reducing the size of a miter circuit. In order to generate a CNF that is used to decide about the equivalence of the parental and candidate circuit, the proposed extension uses of the following steps.

1. Because the evolutionary approach is based on the modify-and-test approach, we can easily determine the difference between the parent individual and its modified (mutated) version. Let Δ be a set containing (a) the indexes of such gates where at least one gene has been modified and (b) the indexes of the outputs that have been modified. This set can be constructed within the mutation phase.

   The improved Equivalence Checking algorithm is illustrated in Figure 5.9. The example contains the reference circuit and its mutated version. For the simplicity, only one mutation will be considered in this example. The second input of gate 7 has been mutated and is now connected to the output of gate 5. Since only one mutation has been executed, Δ contains one index referring the mutated gate $\Delta = \{7\}$. Note that the gate indices correspond with the encoding introduced in Section 2.1.5. The example circuit has four inputs indexed as $0, \ldots, 3$ and consists of 7 gates indexed as $4, \ldots, 10$.

2. Another set, Δ_e, contains the indexes of all the gates and outputs in the mutated circuit which are directly or indirectly connected to the outputs of the gates of Δ. The Δ_e set

can be constructed in linear time as follows. Let $\Delta_e^{(0)} = \Delta$, $i = 0$ and $d_{min} = min(\Delta)$ be the lowest index of a modified gate (if it exists). Check all gates with the index greater than d_{min} (starting with the lowest index). If at least one of the inputs is connected to a gate whose index is stored in $\Delta_e^{(i)}$ then increase i and add the corresponding index to $\Delta_e^{(i)}$. The same procedure is performed for all the primary outputs. At the end, $\Delta_e = \Delta_e^{(i)}$ holds.

In our example, the gates with indexes 8, 9, 10 are successively tested. The Δ_e is initialized to $\Delta_e^{(0)} = \Delta = \{7\}$. Since the gate with index 9 is connected to gate 7 that is already in $\Delta_e^{(0)}$, the gate is added to the set resulting in $\Delta_e^{(1)} = \{7, 9\}$. Finally, the x output is also labeled as it is connected to previously labeled gate 9, thus $\Delta_e^{(2)} = \{7, 9, x\}$. As there are no other nodes to test, $\Delta_e = \Delta_e^{(2)} = \{7, 9, x\}$.

3. In this step, the Δ_r set which contains the indexes of all the gates in the reference parental circuit that contribute to any of the outputs listed in Δ is constructed. Similarly to the previous step, the Δ_r set can be calculated in linear time as follows. Let $\Delta_r^{(0)}$ contains the indexes of the gates that are directly connected to the primary outputs listed in Δ and $i = 0$. Go through all the gates (starting with the highest index). If a gate index is in $\Delta_r^{(i)}$ then increase i and add all the gate indexes the particular gate is connected with to $\Delta_r^{(i)}$. At the end, $\Delta_r = \Delta_r^{(i)}$.

In our example, the set will contain the gates contributing to the x output. Since there is only one gate connected directly to the output x, the $\Delta_r^{(0)}$ is initialized as follows $\Delta_r^{(0)} = \{9\}$. Then, the gates with indices 9,8,7,6 and 5 are successively tested if their output is connected to a gate listed in Δ_r. The algorithm gradually generates the following sequence: $\Delta_r^{(1)} = \{9, 7\}$, $\Delta_r^{(2)} = \{9, 7, 6\}$ and finally $\Delta_r = \Delta_r^{(3)} = \{9, 7, 6, 4\}$.

4. The Δ_f set containing the indexes of all the gates of mutated circuit that have to be included to CNF is constructed. The Δ_f set is determined as follows. Let $i = 0$ and $\Delta_f^{(0)}$ contain the indexes of the gates that are directly connected to the primary outputs listed in Δ_e and whose indexes d meet the following condition: $(d \in \Delta_e) \vee (d \notin \Delta_e \wedge d \notin \Delta_r)$. Then, go through all the gates of the mutated circuit (starting with the highest index). If a gate is in $\Delta_f^{(i)}$ then for every input connected to this gate (with index d) which the following condition $(d \in \Delta_e) \vee (d \notin \Delta_e \wedge d \notin \Delta_r)$ holds for, increase i and add d to $\Delta_f^{(i)}$.

In our example, gate 9 and then gate 7 are labeled because both are labeled in the reference as well as modified circuit. Gate 4 is not labeled because it is labeled in the reference circuit only. Gate 5 is labeled because it is connected to gate 7 but included neither in modified nor reference circuits. Then, $\Delta_f = \{9, 7, 5\}$.

5. Finally, the SAT solver is applied on the clauses representing all the gates that are included in Δ_r and Δ_f, and only those outputs that are in Δ_e. In our example, the final circuit consists of 8 gates (7 + 1 XOR). This is a significant reduction with respect to the common combinational equivalence checking approach described in Section 5.1.2 that lead to 17 gates (14 + 2 XOR + 1 OR). Using the improved version described in Section 5.2, 15 gates (14 + 1 XOR) are encoded to CNF.

a) reference circuit

b) modified circuit

c) miter

$$(x_1 \lor \neg x_5)(x_2 \lor \neg x_5)(\neg x_1 \lor \neg x_2 \lor x_5)$$
$$(\neg x_2 \lor x_6)(\neg x_3 \lor x_6)(x_2 \lor x_3 \lor \neg x_6)$$
$$(x_3 \lor \neg x_7)(x_4 \lor \neg x_7)(\neg x_3 \lor \neg x_4 \lor x_7)$$
$$(x_5 \lor \neg x_8)(x_7 \lor \neg x_8)(\neg x_5 \lor \neg x_7 \lor x_8)$$
$$(x_5 \lor \neg x_{15})(x_6 \lor \neg x_{15})(\neg x_5 \lor \neg x_6 \lor x_{15})$$
$$(\neg x_5 \lor x_{10})(\neg x_8 \lor x_{10})(x_5 \lor x_8 \lor \neg x_{10})$$
$$(\neg x_5 \lor x_{17})(\neg x_{15} \lor x_{17})(x_5 \lor x_{15} \lor \neg x_{17})$$
$$(\neg x_{10} \lor \neg x_{17})(x_{10} \lor x_{17})$$
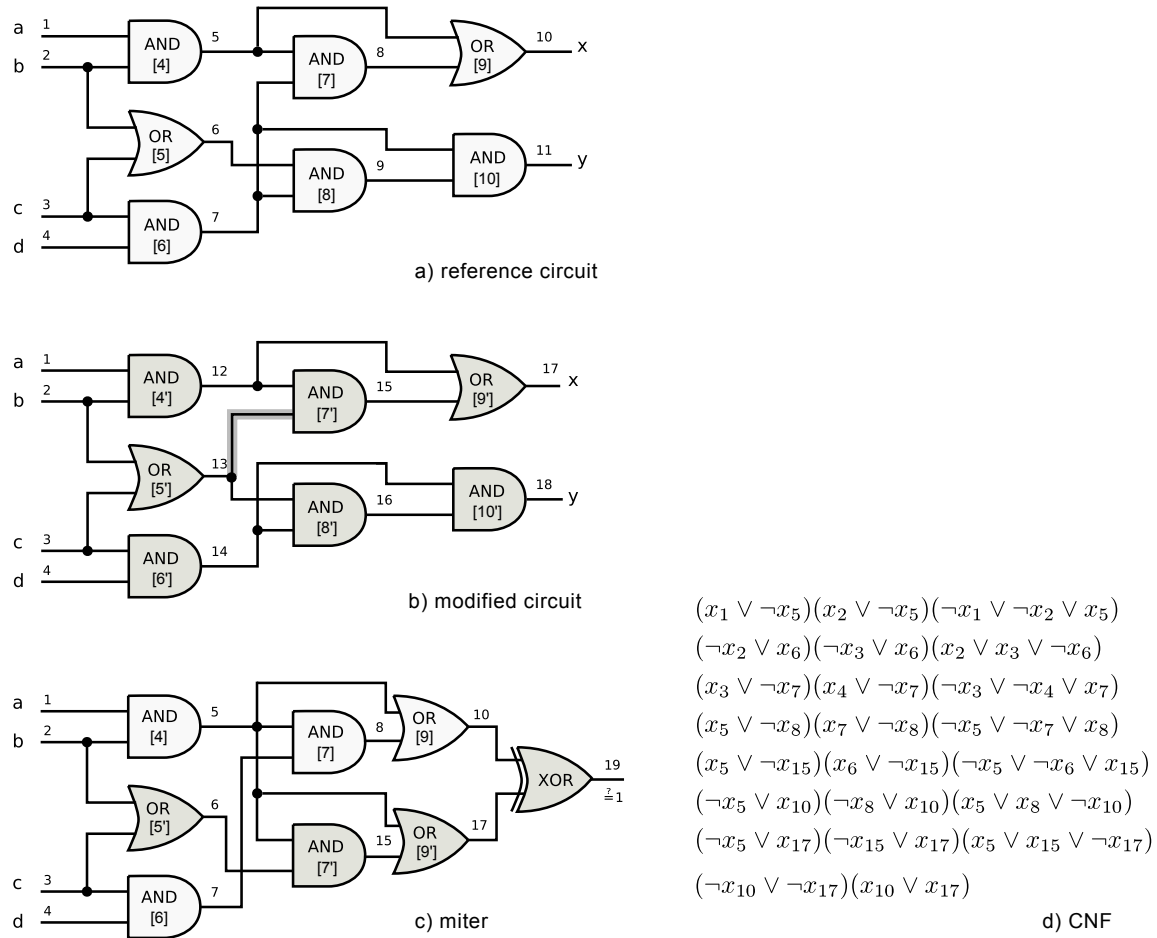
d) CNF

Figure 5.9: Construction of the miter circuit (c) from the reference circuit (a) and modified (mutated) circuit (b) using the improved algorithm. The number of CNF clauses has been significantly reduced. The obtained CNF (d) consists of 23 clauses. The improved CEC proposed in Section 5.2 generates 44 clauses.

5.4.1 Time of Candidate Circuit Evaluation

In order to compare the time of evaluation for standard fitness function t_{CGP}, the proposed SAT-based fitness function t_{sat} (Section 5.2) and the enhanced SAT-based fitness function t_{imp}, the problem of the combinational multiplier optimization has been chosen.

The CGP parameters are as follows: the population size $\lambda = 2$, the set of building blocks consists of 8 common gates $\Gamma = \{\mathrm{BUF, NOT, AND, OR, XOR, NAND, NOR, XNOR}\}$, l-back parameter has been set to maximum value, i.e. $l = N_g$, where N_g is the number of gates of the initial (seed) circuit that has been created using conventional synthesis. One mutation per chromosome has been allowed, one-dimensional CGP structure has been used, i.e. $n_c = N_g$ and $n_r = 1$.

The CGP implementation uses the parallel evaluation described in Section 3.3.2. The circuits were mapped to the 2-input gates using SIS. The experiments were carried out on a cluster consisting of Intel Xeon X5670 2.4GHz processors using the Sun Grid Engine (SGE) that enables to run the experiments in parallel.

Table 5.8 gives the mean evaluation time for the three fitness functions. The results were obtained from fifty 10-minute independent runs of CGP. In the last column, we can observe a significant speedup achieved using the improved SAT-based fitness function.

Table 5.8: The mean evaluation time in milliseconds for three fitness functions in the task of k-bit multiplier evolution.

circuit	n_i	n_o	seed (N_g)	t_{cgp}	t_{sat}	t_{imp}	t_{sat}/t_{imp}
mutliplier 7×7	14	14	238	8	1	4	0,3
multiplier 8×8	16	16	416	45	250	8	33,1
multiplier 9×9	18	18	540	183	1 789	17	105,4
multiplier 10×10	20	20	680	901	6 431	44	146,0
multiplier 11×11	22	22	836	n/a	316 333	88	3 607,8

5.4.2 LGSynth93 Benchmarks

Further experiments were performed using the LGSynth93 benchmark set. In this evaluation, only circuits with more than 20 inputs were considered. These benchmarks are intractable for common CGP based on the standard fitness function that enumerates all the possible input combinations. Table 5.9 compares the number of gates obtained after 3 and 12 hours of optimization using the SAT-based fitness function (Ng_{sat}) and improved SAT-based fitness function (Ng_{imp}). The results clearly show the more runtime available, the more compact circuits obtained in comparison to the reference circuit (the N_g column) synthesized using SIS. The Ne columns give the mean number of evaluations which has been performed within a given time limit. The columns 'speedup' (Ne_{imp}/Ne_{sat}) show that the improved approach is able to evaluate more candidate solutions in a given time limit. The results were obtained from fifty independent runs of CGP.

Figure 5.10 contains convergence curves for selected benchmark circuits – apex1 (the largest one), ex4p (the highest number of inputs), apex3 (the second largest circuit) and apex5 (the second highest number of inputs, the third largest circuit). The graphs contain mean, minimum and maximum number of gates from 50 independent runs of CGP for two variants of fitness function; the SAT-based fitness function proposed in Section 5.2.1 and its enhanced version. We can observe that the improved SAT-based fitness function exhibits better convergence in comparison with the common SAT-based fitness function.

5.5 Experimental Evaluation and Comparison with Conventional Synthesis

5.5.1 Synthesis of LGSynth93 Benchmarks

Table 5.10 contains the best results obtained using the noncommercial and commercial tools. We have used the standard settings for the tools and technology library with the same set of gates as CGP, i.e. $\Gamma = \{BUF, NOT, AND, OR, XOR, NAND, NOR\}$.

It can be seen that the commercial synthesis tools provide results that are comparable with the noncommercial synthesis tools such as ABC and SIS. The results from SIS and

Table 5.9: The min. number of gates obtained using the SAT-based fitness function (Ng_{sat}) and improved SAT-based fitness function (Ng_{imp}) for the LGSynth93 benchmarks. The Ne columns give the mean number of evaluations in millions.

			seed	3h runtime					
circuit	n_i	n_o	N_g	Ng_{sat}	Ng_{imp}	impr.	Ne_{sat}	Ne_{imp}	speedup
apex1	45	45	1408	1179	946	20%	0,22	0,50	2,3
apex2	39	3	235	104	93	11%	2,66	10,04	3,8
apex3	54	50	1405	1280	1099	14%	0,23	0,54	2,4
apex5	117	88	784	675	618	8%	0,93	2,22	2,4
cordic	23	2	67	32	32	0%	10,44	17,46	1,7
cps	24	109	1128	870	643	26%	0,32	0,81	2,5
duke2	22	29	430	286	264	8%	0,98	1,79	1,8
e64	65	65	192	133	138	-4%	3,52	2,39	0,7
ex4p	128	28	500	404	368	9%	1,69	6,79	4,0
misex2	25	18	111	76	73	4%	8,48	12,28	1,4
vg2	25	8	95	79	80	-1%	6,23	5,83	0,9

			seed	12h runtime					
circuit	n_i	n_o	N_g	Ng_{sat}	Ng_{imp}	impr.	Ne_{sat}	Ne_{imp}	speedup
apex1	45	45	1408	921	847	8%	0,22	0,49	2,2
apex2	39	3	235	98	90	8%	3,20	10,77	3,4
apex3	54	50	1405	1167	1038	11%	0,21	0,52	2,5
apex5	117	88	784	633	613	3%	1,02	2,21	2,2
cordic	23	2	67	32	32	0%	13,59	17,84	1,3
cps	24	109	1128	698	585	16%	0,36	0,80	2,2
duke2	22	29	430	268	260	3%	1,22	1,92	1,6
e64	65	65	192	129	129	0%	3,37	2,46	0,7
ex4p	128	28	500	394	349	11%	1,96	7,08	3,6
misex2	25	18	111	70	71	-1%	9,97	13,36	1,3
vg2	25	8	95	74	78	-5%	5,09	5,83	1,1

ABC were obtained by iterative application of the synthesis script (1000 iterations). None of the tools has provide better results than CGP (when CGP is seeded using the first result provided by SIS) with the exception of apex5 where the number of gates is very similar.

5.5.2 Synthesis of Conventionally Hard to Synthesize Circuits

Despite the massive development and more than 50 years of history of logic synthesis and optimization, the latest results provided by Cong and Minkovich in [35] or Schmidt and Fišer in [53] indicate that the current synthesis tools are not able to cope with newly emerging designs. One part of the problem is that most of the currently used gate-level synthesis algorithms and processes have been established in 1980's and they are being used in today's commercial tools; thus the ever-increasing size of the digital circuits becomes a problem. However, there have been discovered very small circuits, for which synthesis tools produce extremely bad results. Cong and Minkovich demonstrated that the number
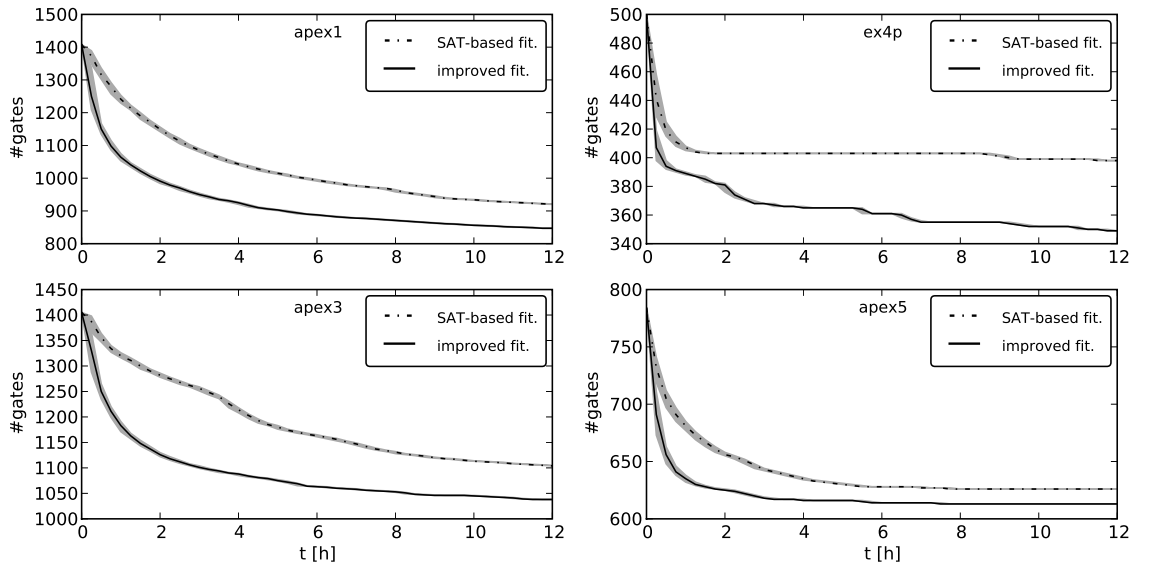
Figure 5.10: Convergence curves for the apex1, ex4p, apex3 and apex5 LGSynth93 benchmarks. The mean, minimum and maximum number of gates from 50 independent runs of CGP for two variants of fitness function

Table 5.10: The min. number of gates obtained using the noncommercial tools (SIS, ABC), commercial tools (C1,C2,C3) and the proposed approach based on CGP.

circuit	SIS	ABC	C1	C2	C3	CGP	impr.
apex1	1394	1862	1439	**1272**	1368	847	33,4%
apex2	**151**	225	221	195	299	90	40,4%
apex3	1405	1737	1494	**1332**	1515	1038	22,1%
apex5	751	768	728	**609**	921	613	-0,7%
cordic	67	61	67	**49**	90	32	34,7%
cps	1128	1109	1150	975	**967**	585	39,5%
duke	406	**356**	417	366	357	260	27,0%
e64	192	384	**183**	191	255	129	29,5%
ex4p	488	523	468	**467**	555	349	25,3%
misex2	111	121	94	**89**	108	71	20,2%
vg2	95	113	88	**83**	109	78	6,0%

of gates of the synthesized circuits is of orders of magnitude higher than the optimum. This study has been extended in [53] where the authors shown that there exists a huge class of real-world circuits for which synthesis fails and provides very poor results. This failure is a problem of both academic (SIS, ABC) and commercial tools. If a large design is broken down to multiple smaller circuits and failures of this kind occur, we obtain an unacceptably large circuit without having any clue for it.

Summarized, up to our knowledge no available conventional synthesis process is able to efficiently discover disclosed structures and to create new, non-standard structures. The synthesis mostly fully relies on local optimizations. Therefore, using some kind of global optimization may overcome drawbacks of present local synthesis algorithms. Genetic pro-

gramming thus becomes an apparent option. The major feature of GP is that it does not employ any deterministic synthesis algorithm; all the optimizations are being done implicitly, without any structural biases.

Experiments

In order to evaluate the proposed system in this task, we have used a set of hard-to-synthesize benchmarks that have been published in literature. In particular, four different classes of benchmark circuits have been employed; difficult standard benchmark circuits, artificially created benchmark circuits, tautology and near-tautology benchmarks and nontrivial parity circuits. Each benchmark circuit was generated by a specific synthesis process using ABC synthesis tool (see [209]). The obtained results (in terms of number of 2-input gates) are shown in Table 5.12, 5.11, 5.13 and 5.14, respectively. Three set of experiments have been used.

Firstly, we have compared the CGP-based optimization with ABC. Both of the tools are utilized as primary optimization processes. The results obtained using ABC are shown in column N_{g_a}, while the results provided by CGP are shown in column N_{g_c}. The relative improvement (the number of gates obtained by CGP in comparison with ABC) is shown in the next column. It can be seen that CGP almost always outperforms ABC, sometimes significantly. The ABC resynthesis completely fails in these cases, while CGP is able to implicitly discover beneficial circuit structures. It is most apparent in the tautology and near-tautology examples. In cases where the difference between ABC and CGP is negligible, most probably the global optimum is approached by both methods.

Secondly, the CGP has been used to optimize circuits obtained by a conventional synthesis. Each of the presented benchmark categories requires a specific synthesis process, to obtain satisfactory results. Generally, it is the capability of XOR decomposition for difficult standard benchmarks and parity benchmark circuits and collapsing for the rest of the benchmarks [209]. We have processed the benchmarks by the respective processes and further minimized by the ABC script, to obtain the best conventional solution. Then, we have processed these circuits by CGP. The obtained results are shown in the $N_{g_{bc}}$ column and the percentage improvement achieved is shown next. It can be seen that the conventional solution was almost always improved.

Finally, we have tried to further optimize the circuits optimized by CGP by running the ABC resynthesis script. Surprising results were obtained, see columns $N_{g_{ca}}$ for the result of CGP optimized using ABC and $N_{g_{bca}}$ for the best conventional result optimized by CGP and ABC. The results were almost always deteriorated by ABC. This gives us a hint that CGP is able to find a very deep local minimum in the circuit size, to refine the structure so that no other synthesis can improve it further more.

Difficult Artificially Created Circuits

The difficult circuit were firstly mentioned in [35]. Even if the commercial tools are able to manage these deceptive benchmarks without problems, the resulting structures contain more than 500-times higher number of gates in comparison with the optimum size. These circuits were called LEKO (Logic Examples with Known Optimum) and LEKU (Logic Ex-

amples with Known Upper Bound) benchmarks. The Cong and Minkovich's LEKU circuits are basically constructed by intentionally introducing a bad structure into the replicated core circuit which made the circuit artificially large. This process may be performed on other, realistic benchmark circuits as well. Collapsing a multi-level network into a two level circuit completely destroys the original structure, which is then very difficult to be recreated. The results obtained for this class of circuits are shown in Table 5.11. In all the cases, the results provided by the conventional synthesis tools have been further improved by CGP. The average improvement was about 17%.

Table 5.11: The minimal number of gates for artificially created difficult benchmarks

circuit	n_i	n_o	N_g	N_{g_a}	N_{g_c}	imp.	N_{g_b}	$N_{g_{bc}}$	imp.	$N_{g_{ca}}$	$N_{g_{bca}}$
LEKU-CB	25	25	759	216	175	19%	235	181	23%	196	189
LEKU-CB_c	25	25	699	214	178	17%	211	176	17%	182	189
LEKU-CD_c	25	25	932	224	186	17%	195	177	9%	195	180

Difficult Standard Benchmark Circuits

Designers have objected to the Cong and Minkovich's LEKU circuits that they are artificially constructed. However, Fišer discovered circuits from the standard LGSynth'93 benchmark set, which are difficult to synthesize as well. Particularly, the capability of the XOR decomposition is required to synthesize these circuits properly. Without using the XOR decomposition, the synthesized circuits are sometimes more than 25 times larger. Unfortunately, the XOR decomposition is not performed in all the available tools (SIS, ABC), except of BDS [194]. Many commercial tools are missing this ability as well. Table 5.12 shows that the conventional synthesis using ABC completely fails as it provides the solutions that are of magnitude higher than the optimum (see columns N_{g_a} and N_{g_c}). The CGP provides the results that are about 90% better than the solutions provided by ABC.

Table 5.12: The minimal number of gates for difficult standard benchmark circuits

circuit	n_i	n_o	N_g	N_{g_a}	N_{g_c}	imp.	N_{g_b}	$N_{g_{bc}}$	imp.	$N_{g_{ca}}$	$N_{g_{bca}}$
9sym	9	1	329	280	27	90%	57	48	16%	37	50
rd84	8	4	713	395	31	92%	85	32	62%	35	33
t481	16	1	1263	420	21	95%	11	11	0%	15	11

Tautology and Near-Tautology Benchmarks

A different kind of artificially complex benchmarks can be created by generating large random sum of products (SOPs). If the number of product terms in the SOP exceeds a particular threshold, the function turns into tautology. Two-level minimization must be run in order to discover the true nature of functions described by this needlessly large amount of SOPs [209]. However, ABC and commercial tools do not do so. If this SOP (in form of a PLA or mapped into technology) is submitted to the synthesis, huge circuits are produced as it is demonstrated in Table 5.13. Comparing to ABC, the proposed CGP-based synthesis

tool is able to discover significantly better structures requiring only a fraction of the original number of gates. Note that the taut1 benchmark circuit is implemented using XNOR gate which inputs are connected to the same input variable.

Table 5.13: The minimal number of gates for tautology and near-tautology benchmarks

circuit	n_i	n_o	N_g	N_{g_a}	N_{g_c}	**imp.**	N_{g_b}	$N_{g_{bc}}$	**imp.**	$N_{g_{ca}}$	$N_{g_{bca}}$
big_pla	25	1	15744	14940	24	100%	29	24	17%	24	24
taut1	25	1	15397	14583	1	100%	1	1	0%	1	1

Parity Benchmark Circuits

In order to obtain difficult benchmarks circuits, Fišer have constructed new benchmark circuits by appending a XOR tree to the circuit's outputs, to obtain one parity bit according to the [53]. The upper bound of the area is the sum of the original circuit size and the size of the XOR tree. The circuit may be then resynthesized, with the hope of decreasing its size. The results for the standard benchmark circuits supplemented with the parity three are shown in Table 5.14.

Table 5.14: The minimal number of gates for the standard circuit benchmarks supplemented with parity tree

circuit	n_i	n_o	N_g	N_{g_a}	N_{g_c}	**imp.**	N_{g_b}	$N_{g_{bc}}$	**imp.**	$N_{g_{ca}}$	$N_{g_{bca}}$
9sym	9	1	217	214	37	83%	57	46	19%	38	49
alu1	12	1	1085	795	52	93%	38	38	0%	57	38
b4	33	1	9645	5008	5003	0%	279	98	65%	4535	104
c8	28	1	1605	486	71	85%	53	51	4%	68	53
cc	21	1	799	347	36	90%	54	36	33%	40	38
count	35	1	1608	921	78	92%	57	54	5%	82	58
ex7	16	1	1985	1392	719	48%	118	74	37%	726	87
i1	25	1	759	397	37	91%	37	35	5%	36	35
in6	33	1	5046	2386	798	67%	118	106	10%	759	111
misex3c	14	1	5869	4445	4444	0%	492	358	27%	4463	355
s1238	32	1	66633	52590	35116	33%	1916	897	53%	26726	935
s298	17	1	2294	1483	52	96%	51	36	29%	61	41
s344	24	1	3387	1910	76	96%	76	61	20%	80	59
s349	24	1	3619	1950	79	96%	82	63	23%	80	73
s420.1	34	1	4098	2521	2541	-1%	80	80	0%	2281	81
s420	35	1	2535	1175	141	88%	123	108	12%	148	109
signet	39	1	49167	36974	45143	-22%	8304	7453	10%	34991	7318
term1	34	1	2397	918	80	91%	136	95	30%	75	101
tt2	24	1	13800	9828	13140	-34%	66	55	17%	10414	62

We have confirmed the findings published in [53]. The conventional synthesis tools are not able to minimize the circuit size efficiently, unless it is collapsed into a two-level SOP network and resynthesized. This process fully resembles the construction of the artificial LEKU benchmarks. The results of the resynthesis are spun between two extreme cases: at

the „good" end, the circuit size is significantly reduced with respect to the upper bound, at the other end the size explodes. The reason for the size explosion is the same as for the LEKU benchmarks – the obtained SOP is too large and the subsequent synthesis is not able to rediscover the original circuit structure. The need for XOR decomposition has been emphasized even more in these experiments. Tools not able to perform the XOR decomposition sometimes produced results 50-times larger than the upper bound.

5.6   Summary

As it has been presented in Chapter 3, the current methods of evolutionary synthesis are capable of evolving either small and simultaneously innovative circuits or larger circuits that are not interesting from the implementation point of view because of their inherent inefficiency. According to our best knowledge, when a perfect synthesis scenario is considered, the most complex combinational circuit has been successfully evolved by Stomeo in [172]. This circuit, specified by a truth table, consists of tens of gates and has 17 primary inputs and one primary output. Even if this result can be considered by the EHW community as a great success because it was evolved from scratch, it has a marginal significance from the viewpoint of the logic synthesis because commercial synthesis tools are able to handle the combinational circuits having hundreds of inputs and thousands of gates. The main reason that prevents EA from evolving large and real-world competitive circuits is primarily caused by the problem of scalability of the fitness evaluation.

We have shown that it is possible to eliminate the mentioned scalability limits by introducing a different fitness evaluation procedure. The proposed method is based on applying formal verification techniques that allow a significant acceleration of the fitness evaluation procedure, overcoming thus the major bottleneck of evolutionary design. In particular, we have used a SAT solver in the fitness function that allows significant reducing of the computational requirements of the fitness function for such combinational circuit optimization problems for which a fully functional initial solution exists before the optimization is started. Proposed algorithm can produce complex and simultaneously innovative designs, quite competitive with the state-of-the art logic synthesis tools. This method can potentially be used to minimize the area on the chip, delay of the circuit, power consumption or to minimize the number of test vectors.

Comparing to the standard CGP, we have demonstrated that the proposed method is able to evaluate over $40000\times$ more candidate solutions in the same time when the common 32-input parity benchmark problem is considered. In addition, we have introduced some CGP-specific techniques that are able to further improve the performance of a SAT solver. Using the multipliers, known as hard benchmark problems, we have shown that the enhanced version of the proposed method tracking the changes between parent and its offspring is able to provide the additional speedup over 3000 when the 11-bit multiplier is considered. Note that the speedup increases mostly exponentially with the increasing complexity of the solved problem.

The proposed technique has been evaluated using the common LGSynth93 benchmark circuits. It has been shown that this approach enables to optimize large logic circuits having

from tens to hundreds of inputs and thousands of logic gates. The most complex LGSynth93 benchmark circuit (apex5) consists of 784 gates, 117 primary inputs and 88 primary outputs. Using another benchmark set, we have demonstrated that the proposed method can handle the circuits that are known to be hard for common synthesis tools. These circuits consist from several hundreds to several thousands of logic gates. The largest circuit that has been successfully processed by the proposed evolutionary method contains 66633 logic gates, 32 primary inputs and one primary output.

We have also demonstrated that despite the fact that various logic synthesis and optimization tools have been proposed in the recent 50 years, the logic synthesis/optimization problem has not been completely solved yet. Using the LGSynth93 benchmark we have shown, that the best-obtained results of conventional synthesis conducted using academia as well as commercial tools can be improved by the proposed method in 20-40%. The experiments with the hard-to-synthesize circuits show that a significant area improvement (33-99%) can be reached using the proposed evolutionary approach. In this case, CGP is able to discover structures, for which conventional synthesis completely fails. As a result, CGP can be efficiently used as a primary circuit optimization process, which, as we have found by processing numerous benchmark circuits, universally produces good results, regardless the original circuit structure.

The main drawback of the CGP optimization is a long runtime (several hours) required to obtain reasonable improvements, especially for large circuits. However, for the cost of runtime, CGP is able to produce results that conventional synthesis is never able to reach. The long runtime drawback may be partially compensated by running CGP as a post-synthesis process. The original circuit is first maximally reduced by a conventional synthesis and then optimized by CGP. As a consequence, the circuit size can be further reduced.

Although the results for LGSynth93 benchmarks are very encouraging, the SAT-based combinational equivalence checking can definitely perform unsatisfactory for some problem instances. However, the proposed method is assumed to be able to handle large-scale optimization problems if more advanced version of SAT solver is utilized.

Chapter 6

Evolutionary design of nonlinear image filters

Image preprocessing, which includes image filtering, edge detection, histogram equalization, brightness and contrast adjustment, and other low level operations over images, is the first stage of many applications. As low-cost digital cameras have entered to almost any place, the need for high-quality, high-performance and low-cost image filters is of growing interest. It is a well-known fact that the quality of preprocessing significantly influences the accuracy, reliability, robustness and performance of subsequent image processing steps such as segmentation, classification, recognition etc. In order to perform the required preprocessing (such as image filtering, edge detection etc.) a problem-specific filter has to be created. Traditionally, engineers use a library of predefined filters and operators and manually tune promising variants of these filters for a given application. In the process of tuning, various properties of filters might be optimized, in particular, their coefficients and structure [24, 47, 145]. There are also other important parameters to be optimized. With emerging of new portable devices, the number of operations should be optimized since it has impact on the performance as well as power consumption. In case that the filter should be implemented as a digital circuit, the parameters such as area, delay and power consumption play an important role.

Historically, *linear filters* became the most popular filters in image processing. The reason of their popularity is caused by the existence of robust mathematical models which can be used for their analysis and design. However, there exist many areas in which the *nonlinear filters* provide significantly better results [46]. The advantage of nonlinear filters lies in their ability to preserve edges and suppress the noise without loss of detail. The success of nonlinear filters is caused by the fact that image signals as well as existing noise types are usually nonlinear. As there is no suitable general theory for the design of nonlinear operators, evolutionary design techniques have been utilized to accomplish this task in the recent years. The pioneer work in this area has been done by Sekanina who applied Cartesian Genetic Programming in the image filter design task [157].

Sekanina has shown that evolutionary design techniques are able to generate slightly better solutions than the standard filters [158]. Unfortunately, his direct evolutionary design approach which works for low noise intensity does not work for higher noise intensities.

The goal of the research presented in this chapter is to show that by an innovative combination of evolved designs and conventional designs we are able to propose the systems that exhibit at least comparable quality with respect to the conventionally used approaches and simultaneously significantly reduce the overall implementation cost on a chip in comparison to standard approaches based on sophisticated filtering schemes, such as adaptive median filter. In this research, the evolutionary design approach is presented as a tool that can automatically discover nonlinear image filters that are competitive with filters designed conventionally in terms of filtering quality as well as implementation cost on a chip.

6.1 Theoretical Background

In this section, we will briefly introduce the principles of the image filters. In order to compare the results produced by CGP we will introduce the most popular conventional methods that are utilized to suppress selected types of non-linear noise. In order to demonstrate the advantage of the evolutionary designed filters, the conventional filters are also discussed from the hardware implementation point of view. In this research we will take into account grey-scaled images only; however, the concept can be naturally extended to color images.

6.1.1 Image Filters and Sliding Window Function

The image filters operate in the spatial or frequency domain. While the linear filters are implemented in frequency domain, the software as well as hardware implementations of non-linear image filters operate in the spatial domain. As spatial filters operate with pixel values in the neighbourhood of the centre pixel (so-called filter window or kernel), it is necessary to implement a *local neighbourhood function* (sometimes referred to as *a sliding window function*). This function is applied separately on all pixel locations and is typically invariable for all locations (i.e. spatially invariant). Figure 6.1 shows the concept of sliding window function.
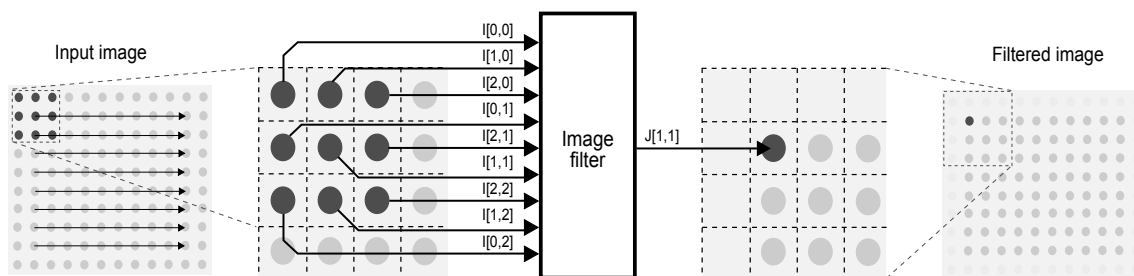
Figure 6.1: The concept of sliding window function. The output of the image filter for location $[x, y]$ $(x, y = 1)$ is calculated according to a value of the pixel situated at location $[x, y]$ and its surrounding pixels. In this case, 3×3 sliding windows is considered.

Figure 6.2 shows the most common hardware architecture of the sliding window function that uses the row buffers. This approach assumes that one image pixel is read from memory in one clock cycle. The pixels are read row by row. When buffers are filled (which is done

with a fixed latency), this architecture provides the access to the entire pixel neighborhood every clock cycle. Note that when local neighborhood function is applied at edge locations some of the neighborhood is not defined. In order to cope with this problem and produce the images with the same size, the undefined pixels can be assigned a value of 0, or can reflect pixel values across each edge.

Figure 6.2: Implementation of a 3×3 filter window. Row buffers are used to reduce the memory access to one pixel per clock cycle.
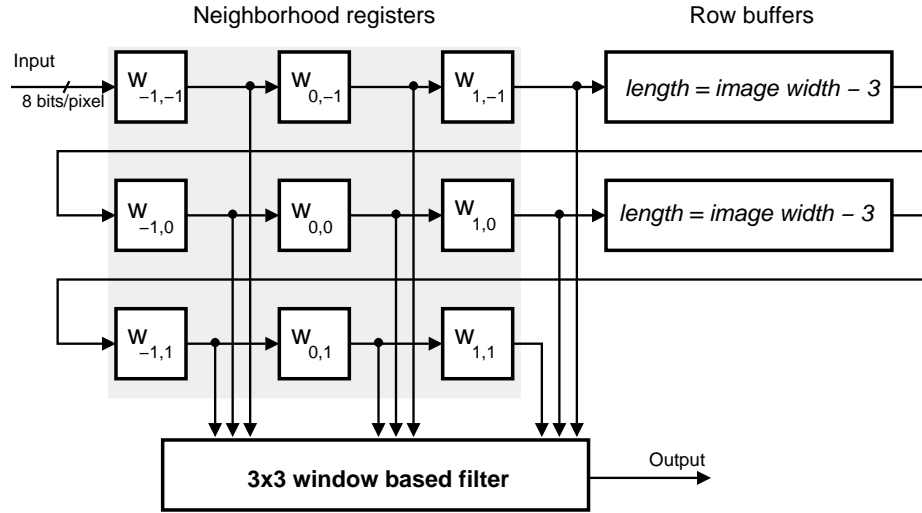
The length of the shift registers depends on the width of the input image. In order to implement a sliding window, several image rows have to be stored. The number of rows corresponds to the window size. Another approach is to choose a fixed row length and divide the input image into strips. However, this method leads to decreasing of the performance due to data overlapping (when compared to the usage of full-length row buffers).

While the architecture places the lowest demand on external memory bandwidth, the highest demand is placed on internal memory bandwidth. This architecture is suitable for FPGA devices as the modern FPGAs contain large amount of fast embedded memory. If the embedded memory is not available, it is necessary to access the external memory for more than one pixel in one clock cycle. Thus this approach requiring the external memory can be efficient only for small window sizes. In consequence of that it is rarely used in high performance image processing since the memories represent a performance bottleneck.

As the implementation cost of buffers implementation depends on the size of the input image and as the buffers have to be implemented for every window-based spatial filter, we will not consider this implementation cost in the comparisons which will be performed later.

6.1.2 Impulse Noise

Due to the imperfections of image sensors, images are often corrupted by a noise. The impulse noise is the most frequently referred type of noise. In most cases, impulse noise is caused by malfunctioning pixels in camera sensors, faulty memory locations in hardware, or errors in the data transmission. We distinguish two common types of impulse noise;

the *salt-and-pepper noise* (commonly referred to as intensity spikes or speckle) and the *random-valued shot noise*. For images corrupted by salt-and-pepper noise, the noisy pixels can take only the maximum or minimum values. In case of the random-valued shot noise, the noisy pixels have an arbitrary value. *Impulse burst noise* represents another type of impulse noise that consists of sudden step-like transitions between two or more discrete values at random and unpredictable times. In fact it is a variant of a random-valued shot noise that is characterized by longer duration. The main reason for the occurrence of bursts is the interference of frequency modulated carrying signal with the signals from other data sources. This interference can occur several times during a transmission of a single image and corrupt several image pixels in one or more neighboring rows. Impulse burst noise is also often accompanied by salt-and-pepper noise and multiplicative noise [96]. In case that the images are transferred row-wise, the impulse burst noise causes horizontal strikes. The similar effect occurs if the images are transferred column-wise. The impulse burst noise is a specific kind of noise which is difficult to filter even if a non-linear filter is used. With the increased noise intensity, more consecutive rows may be affected and subsequent noise filtering becomes difficult as the filtered value need not be determined according to the values of the neighboring pixels. Therefore, a larger filter window ought to be considered in order to obtain a satisfactory quality of the filtered image.
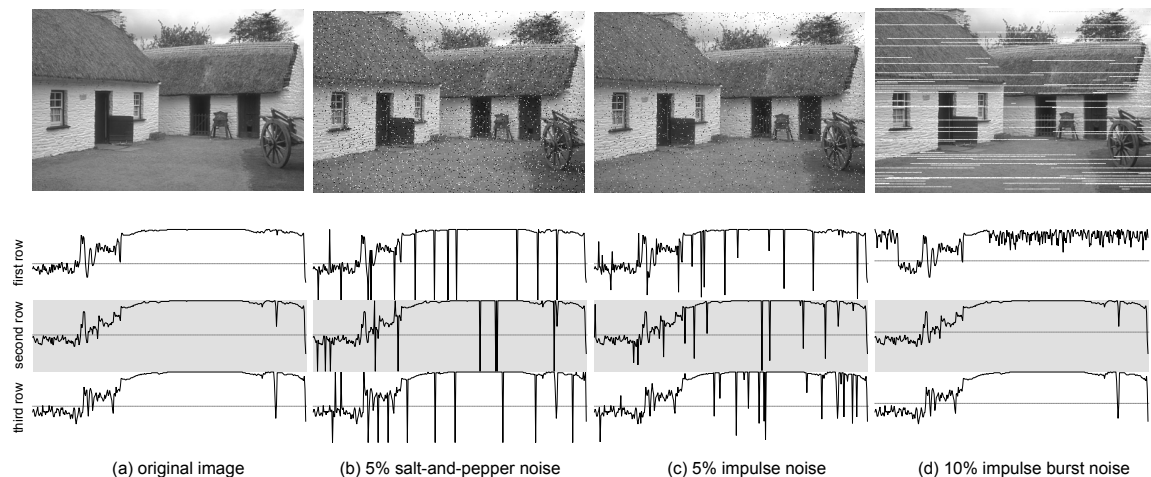
Figure 6.3: Image consisting of 384×256 pixels corrupted by salt-and-pepper noise with $p = 5\%$ (b), random-valued impulse noise with $p = 5\%$ (c) and impulse burst noise with $p = 10\%$, $q = 128$ (d). For the illustration, a snapshot of the first three rows represented as one-dimensional signals is depicted below each image.

The shot noise and impulse noise in general can be modeled as follows. Consider image I and observation image J of the same size $w \times h$ pixels. The image J corrupted by the impulse noise can be expressed as

$$J_{xy} = \begin{cases} N_{xy} & \text{with probability} \quad p \\ I_{xy} & \text{with probability} \quad 1 - p \end{cases} \tag{6.1}$$

where I_{xy} and J_{xy} denotes the pixel values at location (x, y) of the original image and the noisy image respectively, N_{xy} a noise value independent on I_{xy}, $x = 1, \ldots, w$ and

$y = 1, \ldots, h$. The p is the probability that a given pixel is affected by a noisy value, $0 < p < 1$. For gray level images encoded using 8 bits per pixel, the N_{xy} can take up to 256 discrete values. In case of the salt-and-pepper noise, N_{xy} is equal to 0 or 255 each with equal probability. In case of the random-valued shot noise, N_{xy} is usually modeled using the discrete uniform distribution.

The burst noise can be characterized using two parameters: p and q. Similarly to the previous model, let p denote a probability that a certain pixel is affected by the noise. Let q be a parameter which determines the maximal duration of a burst expressed as the maximal number of consecutive pixels which are affected by an impulse. The number of burst fragments in the image depends on both these parameters; the higher q, the lower number of burst fragments for a given value of p. Figure 6.3 shows example of an image corrupted by various types of impulse noise.

6.1.3 Nonlinear Impulse Noise Filters

Traditionally, the impulse noise is removed by a *median filter* [3] which represents the most popular nonlinear filter even if the quality of the filtered images is poor in comparison with other advanced techniques. The output of the median filter is calculated as the median value of the kernel. The success of the common median filter is mainly based on its simple and efficient software as well as hardware implementation which is straightforward and does not require many resources. However, the standard median filter gives a mediocre performance even for images corrupted by impulse noise with lower intensity. Even if the common median filter utilizing 3×3 or 5×5-pixel window is able to repair all the noisy pixels for the noise intensity less than approx. 10-20%, it simultaneously degrades the filtered image because it replaces all the pixels with the median value. When the intensity of noise is increasing, a simple median filter leaves many shots unfiltered. In order to increase the performance, it is possible to increase the size of filter window, however, the larger filter kernel results in loosing of more details.

The median-based filtering approach has been intensively studied and extended to promising approaches such as *center weighted median filter* (CWMF) [94], more general *weighted median filter* (WMF) [22] or *order statistic* and *weighted order statistic filter* [117]. A good survey of the existing methods can be found for example in [155]. Nevertheless, all these median-based methods tend to smudge the image since applying the median filtering to the entire image would inevitably remove details presented in the image. Almost all alternatives to median filters have already been implemented in hardware [52, 25, 115, 29, 108, 28].

In order to overcome the main drawback of the median-based filters, a *switching-based median filtering* concept has been proposed [173]. This concept splits the filtering process into two parts – noise detection and noise replacement. The noise detector determines which pixels are affected by the impulse noise and only these pixels are replaced. Noise detection can be based on various concepts: a median-based filter [173], fuzzy techniques [150] or neural networks [95]. However, the common problem of the proposed detection mechanisms is the necessity to predetermine the value of a threshold parameter which significantly influences the filtering quality.

The *adaptive median filter* (AMF) proposed in [85] is a robust approach which tries to identify and replace the affected pixels only. In contrast with the previous approaches, the detection method is based on the statistical ordered filters with gradually increasing kernel size. Compared to the common median-like filters, AMF provides significantly better results even for images corrupted with high intensity impulse noise. Similarly to the median-based approaches, the adaptive median filter can be efficiently implemented in hardware [218]. In addition to filtering, adaptive median filters can be also used as detectors of corrupted pixels (detection statistics) [199, 136].

Apart from the non-iterative algorithms, the iterative algorithms such as *pixel-wise median of the absolute deviations from the median* (PWMAD) [37] or *directional weighted median filter* (DWMF) outlined in [45] have been introduced. These approaches provide very good results if the random valued impulse noise is considered; they do not contain any varying parameters and require no previous training or optimization. The main disadvantage is apparent – the iterative approach places higher requirements for the memory resources especially in case of hardware implementation.

While the common impulse noise can be successfully filtered using DWMF or PWMAD, the removal of the impulse burst noise using these filters fails especially if the noise intensity is higher. All these median-like filters rely on the principle of spatial locality which is violated. Various filters have been proposed to suppress impulse burst noise in the recent years. Apart from the AMF and WMF which produce images of reasonable quality, specific filters developed for impulse burst noise such as training-based optimized soft morphological filters and variational approaches [97, 96, 136, 46] have been introduced. Unfortunately, it is much more difficult to implement these filters in hardware than median filters because they use for example unlimited kernel size, nontrivial restoring algorithm (e.g. solving of differential equations), etc. Thus, AMF, PWMAD and WMF represent the tradeoff between filtering quality and implementation cost.
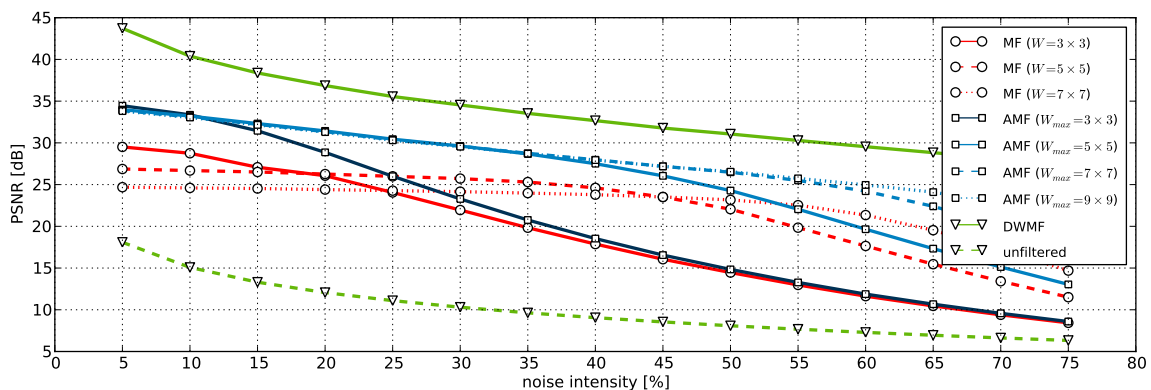
Figure 6.4: Comparison of various image filters using a set of 25 test images corrupted by salt-and-pepper noise of intensity 5-75%. Mean PSNR is reported.

Figure 6.4 summarizes the results of filtering properties of adaptive median filters (with filtering windows 5×5, 7×7 and 9×9) and standard median filters (with filtering windows 3×3, 5×5 and 7×7) that were evaluated using a set of 25 test images (see [218]). All images were corrupted by salt-and-pepper noise of intensity 5-75%. The results were also

compared to the best known software solution (DWMF [45]) which utilizes filtering windows of unlimited size. The visual quality of filtered images is numerically expressed by the peak signal-to-noise ratio (PSNR). The higher the PSNR value, the better filtering quality.

| (a) original image | (b) corrupted image | (c) MF 3x3 |

| (d) MF 5x5 | (e) AMF 5x5 | (f) AMF 7x7 |
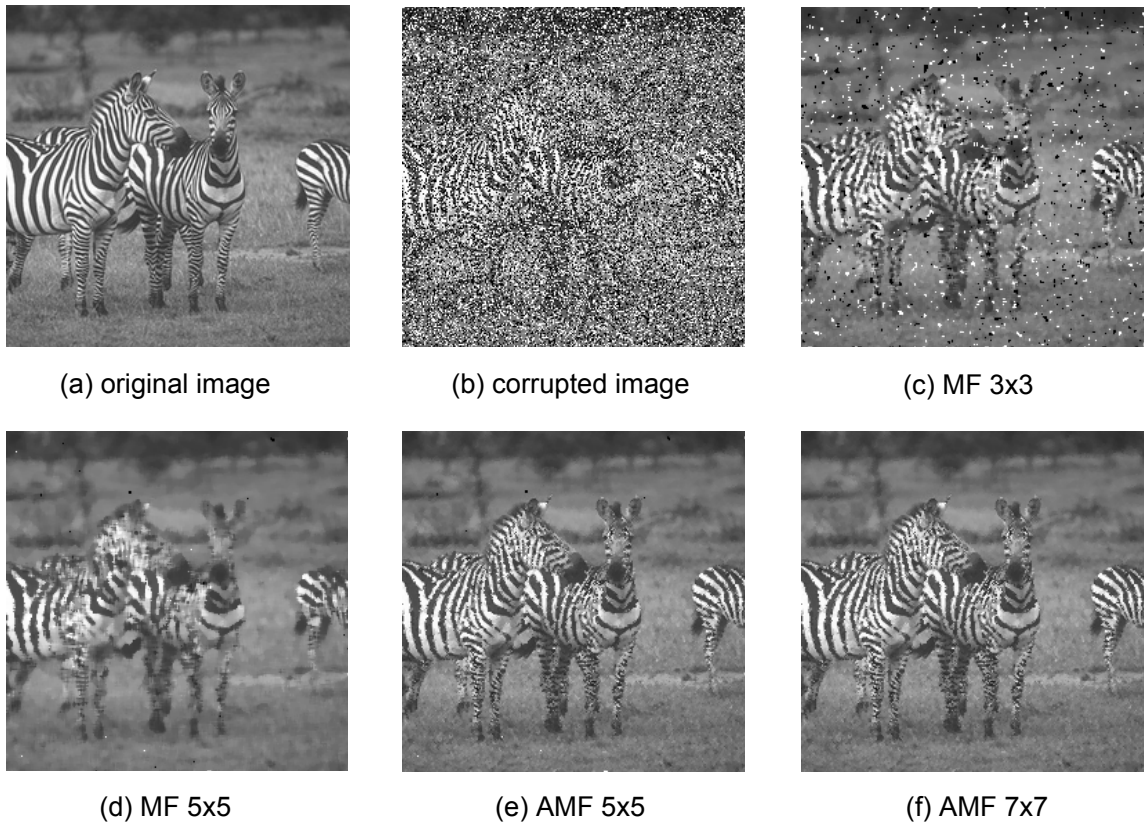
Figure 6.5: Image corrupted by salt-and-pepper impulse noise filtered using conventional filters. (a) Original image (b) Image corrupted by 40% salt-and-pepper noise (PSNR: 9.535 dB) (c) Filtered by median filter with the kernel size 3×3 (PSNR: 16.796 dB) (d) Filtered by median filter with the kernel size 5×5 (PSNR: 18.309 dB) (e) Filtered by adaptive median with the kernel size up to 5×5 (PSNR: 22.021 dB) (f) Filtered by adaptive median with the kernel size up to 7×7 (PSNR: 22.078 dB)

Figure 6.5 contains examples of filtered images. Increasing the size of filtering window allows the standard median filter to remove a great deal of noisy pixels; however because the standard median filters modify almost all pixels, images become smudged and detail less. Thus it is suitable to utilize the smallest window as possible. Nevertheless such a filter fails when the noise intensity is higher than approx. 10-20%. On the other hand, adaptive median filters work correctly not only for higher but also for lower noise intensities because they try to use the smallest possible window and modify only corrupted pixels. In this case, the size of filtering window influences the quality of filtering when noise intensity is higher than 40%.

Order statistic and median filters

Consider a sequence $\{x_1, x_2, \ldots, x_N\} = \{x_i\}$, $1 \le i \le N$ that consists of N elements generated by a random variable X. Let $\{x_i\}$ be arranged in ascending order so that $x_{(1)} \le x_{(2)} \le \cdots \le x_{(k)} \le \cdots \le x_{(N-1)} \le x_{(N)}$. Then, element $x_{(k)} = S_{(k)}\{x_i\}$ is so-called *k-th order statistic*. Note that element $x_{(1)}$ corresponds to the *minimum* of the observed sequence and $x_{(N)}$ to the *maximum*. In case that $k = (N+1)/2$, where N is odd, $x_{(k)}$ is the *median* of the given sequence.

Let M be the length of the filter window, $M = 2L+1$, and $\{x_i\}$ is the input sequence, $1 \le i \le N$ and $N \ge M$. Then the filter defined by specifying its output y_j ($j = L+1, \ldots, N-L$) as $y_j = S_{(k)}\{x_{j-L}, \ldots, x_{j+L}\}$ is denoted as the *k-th order statistic filter* (OSF). It is obvious that if the $k = (N+1)/2$ then the k-th order statistic filter defines the standard median filter.

So-called weighted OSF [30] assigns a weight to every element of the observation window. This generalization allows the usage of some elements of window more than once. On contrary, some elements need not to be included into the process of filtering.

$$[x_{i,j}] = \begin{bmatrix} 2 & 2 & 1 & 2 & 2 \\ 1 & 1 & 2 & 1 & 2 \\ 2 & 3 & 3 & 3 & 2 \\ 4 & 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 4 & 2 \end{bmatrix} \qquad [y_{i,j}] = \begin{bmatrix} 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 2 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 3 & 3 & 4 & 4 \end{bmatrix}$$

$$S_6(\{2,3,3,4,1,2,3,2,1\}) = S_6(\{1,1,2,2,2,3,3,3,4\}) = 3$$

Figure 6.6: Filtering using a two dimensional 6th order statistic filter (a 3×3 filter window is used)

Because each pixel of a given image can be treated as a random variable, statistic order filter can be used for the filtering of the images. However, in this case we need a two dimensional variant of statistic filter which can be obtained as an extension of the one-dimensional case mentioned above. Instead of one-dimensional observed sequence $\{x_i\}$, we have to consider a two-dimensional matrix $[x_{i,j}]$. Each element of this matrix corresponds to one pixel of observed input image. Similarly, the output of the filter is a two-dimensional matrix $[y_{i,j}]$ (see Figure 6.6). Note that the two-dimensional statistic order filter does not have to use every element of the rectangular filtering window.

The hardware implementation of median-based filters can be divided into three classes [30]: *array*-based architectures, *stack filter*-based architectures, and *sorting network*-based architectures.

The array architectures use a large number of simple processors arranged into a systolic linear array. Each processor processes one value of the filter window. Even if the processors can be pipelined and can provide high throughput, this architecture is not suitable for manipulating with large windows. Unfortunately, large windows are typical for adaptive median filters.

The most efficient approach is based on stack filters. A stack filter uses a transformation

of process of filtering into the binary domain. This transformation uses threshold decomposition. Processing in the binary domain is very efficient and can be easily parallelized. The main disadvantage of this approach is the requirement for a high number of decomposition levels which depends exponentially on the number of bits used to represent each pixel. On the other hand, in the serial bitwise version, the stack filters usually allow the most area efficient implementation [108].

Sorting networks-based architectures can be used to implement arbitrary rank order filters. The samples of observed filter window are sorted by a sorting network. Then, the value in the middle of sorted sequence represents the median value. Sorting network is defined as a network of elementary operations denoted as *compare&swap* elements (sometimes called comparators) that sorts all input sequences. A compare&swap (CS) of two elements (a, b) compares a and b and exchanges (if it is necessary) the elements in order to obtain sorted sequence. A sequence of compare&swap operations depends only on the number of elements to be sorted, not on the values of the elements. The main advantage of the sorting network is that the sequence of comparisons is fixed. Thus it is suitable for parallel processing and pipelined hardware implementation. In hardware, CS is implemented using two multiplexers that are controlled by means of a comparator that determines the maximum of the two. As the sorting network can be easily pipelined, the approach provides the best performance. There exist different types of sorting networks. The sorting networks constructed using Batcher's bitonic sort and Batcher's odd-even merge sort provide the best results in terms of implementation cost [218].

(a) 9-input sorting network

(b) pipelined hardware implementation
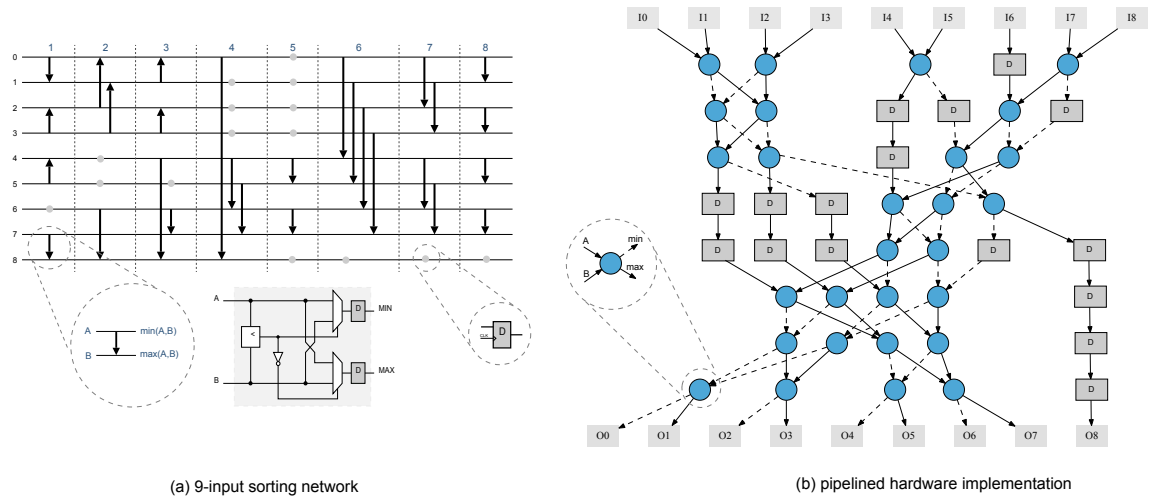
Figure 6.7: Structure of 9-input sorting network (a) and its pipelined hardware implementation (b). Each vertical line represents one compare&swap operation. The arrow determines the position of maximum of the two inputs. The pipelined hardware implementation consists of compare&swap operations and buffers. Providing that the O4 is the output and I0-I7 inputs connected to the sliding window function, the shown structure represents a median filter.

The bitonic sorter [14] is developed on the basis of the 0-1 principle [93]. It is based on merging of two so-called bitonic sequences. A 0-1-sequence is called bitonic if it contains at

most two changes between 0 and 1. The main idea is to recursively divide the input sequence into several parts. In each part, bitonic sequences are created and subsequently merged in order to create 1) another larger bitonic sequence and 2) sorted sequence. After all merging tasks, the sequence is sorted. Structure of the 9-input sorting network created using the bitonic sorter and its corresponding hardware implementation is depicted in Figure 6.7.

Results of synthesis of the pipelined median filter are summarized in Table 6.1. The median filters were described in VHDL, simulated using ModelSim and synthesized using Xilinx ISE tools to Virtex II Pro XC2VP50-7 FPGA. The implementation cost is expressed in terms of slices. The utilized FPGA contains 23616 slices in total.

Table 6.1: Results of synthesis (number of slices) of common median filters

| # inputs | median filter | | | max. freq. |
	optimal	using bitonic SN	using oe-merge SN	
9 (3x3)	268	297	289	305 MHz
25 (5x5)	1506	1706	1582	305 MHz
49 (7x7)	unknown	4815	4426	303 MHz
81 (9x9)	unknown	10315	9719	302 MHz

Switching-based filters

The switching-based approach outlined in [173] can be considered as a general process of filtering that operates in two steps. In the first step, the noisy pixels are detected using a detection algorithm. Then, the new values of the corrupted pixels are estimated using an estimation algorithm.
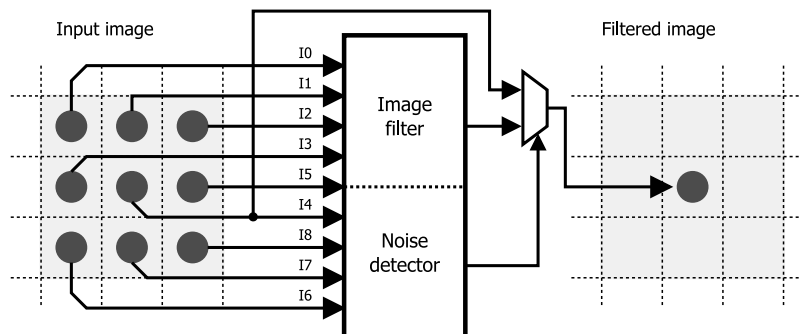
Figure 6.8: The concept of the switching-based filtering using a 3×3 filter kernel

Let x_{ij} and y_{ij} denote pixels with coordinates i, j in noisy image and filtered image respectively. If the estimated value of the corrupted pixel x_{ij} is z_{ij}, the switching filter concept can be defined as

$$y_{ij} = s_{ij} \cdot z_{ij} + (1 - s_{ij}) \cdot x_{ij} \qquad (6.2)$$

where s_{ij} is a binary noise map – an output produced by the estimation algorithm. Noise map s_{ij} contains ones at the positions of pixels detected as noisy pixels.

In general, s_{ij} is determined by comparing the absolute difference between the original pixel value x_{ij} and some local statistics $\Omega(x_{ij})$ with a threshold T. Statistics $\Omega(x_{ij})$ can

be produced by common median filter, weighted median filter, adaptive median filter or using a complex detection mechanism, e.g. DWMF or PWMAD. Since the value of T is highly correlated to the image contents, noise probability and distribution, T has to be calculated for each filtered image. This is unpractical since the problem of finding the optimal threshold is a complex task. While setting T too high leaves a lot of the noisy pixels unfiltered, too low T causes that image details will be treated as noise and the overall image quality will be degraded. In order to avoid setting of this parameter, the process of noise map estimation is usually applied iteratively with varying threshold (e.g. DWMF). The objective of this approach is to make the choice of optimal T irrelevant. Estimated value of the filtered pixel z_{ij} is usually based on common median filter or its variants (e.g. weighted median filter).

The concept the switching-based filtering is shown in Figure 6.8. The noise detector provides single bit value according to which the filter action is determined (i.e. whether the processed pixel is a noise that needs to be filtered or it is an uncorrupted pixel that passes the filter unchanged).

Adaptive median filter

The adaptive median filter (AMF) can be defined in several ways [86, 85]. We will use the definition based on the order statistic. In this sense, AMF can be considered as *iterative order statistic filter.* The iterative processing was introduced in order to detect and replace *corrupted* pixels only. In each iteration, filtering windows of *different* sizes are utilized.
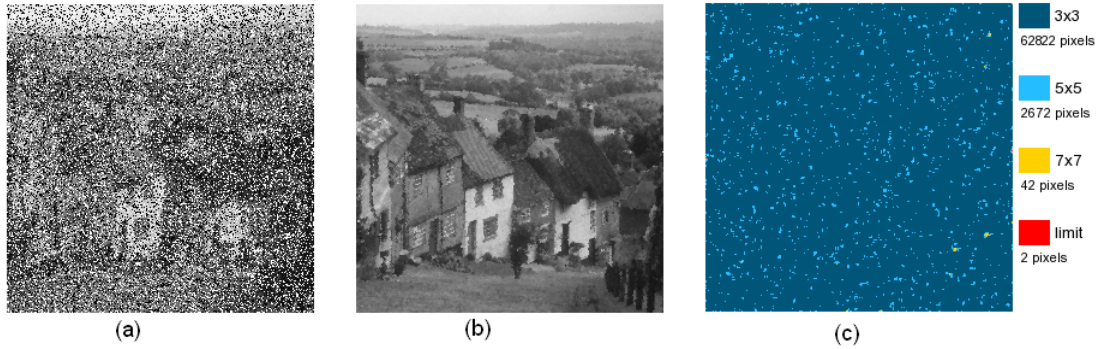
Figure 6.9: (a) Image corrupted by 40% salt-and-pepper noise, (b) Image filtered by adaptive median filter with $W_{max} = 9 \times 9$, (c) Size of the filtering window used for processing each pixel of the input image and the processed number of pixels in each stage

In order to simplify the description, we will deal only with one filter window located at position (u, v). Let a two-dimensional matrix $[x_{i,j}]$ describe the input image and W is the size of the filtered window. Let the sequence $[w_{k,l}]$ be the output of a local neighborhood function which contains just $N = W \times W$ samples of filter window located at position (u, v) (assume that W is odd). Let x_{uv} denote the value of pixel $x_{u,v}$ which corresponds to the value of a pixel at position (u, v) of the input image. Let y_{uv} be the output of the AMF located at position (u, v). The algorithm of adaptive median filter is as follows:

1. *Initialization*
   Start with the smallest windows size $W = 3$. Let the maximum window size be W_{max} (again, an odd number).

2. *Computation of order statistic*
   Let $x_{min} = S_{(0)}([w_{k,l}])$ be the output of the 0-th order statistic filter. $x_{max} = S_{(N)}([w_{k,l}])$ is the output of the N-th order statistic filter And $x_{med} = S_{((N+1)/2)}([w_{k,l}])$ is the output of the median filter.

3. *Evaluation of the terminating condition*
   If the condition $x_{min} < x_{med} < x_{max}$ is satisfied then the processing ends with the computation of the output value which is defined as follows: If $x_{min} < x_{uv} < x_{max}$ then the pixel is not corrupted by noise and the output value is the value of the original pixel, i.e. $y_{uv} = x_{uv}$. If $x_{min} < x_{uv} < x_{max}$ is not satisfied then the output value is the median of the window, i.e. $y_{uv} = x_{med}$. If the condition is not satisfied then the computation continues.

4. *Increasing of the window size*
   If the condition $x_{min} < x_{med} < x_{max}$ is not satisfied, it can be interpreted as follows. If many pixels have the same value then it is impossible to determine (with the current window size) whether the pixels are corrupted with high intensity noise or whether it is the constant area with all pixels of the same color. This is the reason why the window size has to be increased.

   If the window W is smaller than W_{max}, increase the size of the window, i.e. $W = W + 2$, and repeat the computation from step 2. If the size of the window W reaches the maximum value W_{max}, the processing ends and the output value is defined as $y_{uv} = x_{med}$.

Figure 6.9 demonstrates the filtering capabilities of adaptive median filter. The image containing 256×256 pixels corrupted by 40% impulse noise is filtered by adaptive median filter using three levels (i.e. with the window which can take up to 7×7 pixels). It can be seen that all the noisy pixels were successfully detected and removed. More than 95% pixels (62822 out of 65536) were processed in the first level using the 3×3 pixel window. Then, 4% pixels were processed in the next level using the 5×5 window.

Although the adaptive median filter is defined as an iterative filter, the result can be computed in a two-phase process [218]. The idea is to implement a set of sorting networks of different number of inputs (from 3×3 to $W_{max} \times W_{max}$). The minimum, maximum and median value of each sorting network is utilized. As these sorting networks have different latencies it is necessary to include registers at suitable positions to synchronize the computation. In the second phase, the outputs of sorting networks are combined together using a simple combination logic. Because we will need a reference hardware implementation of adaptive median filter in next chapters, we have implemented the filter in FPGA. The proposed hardware architecture of the adaptive median filter implemented using the two-phase scheme is depicted in Figure 6.10. Note that the hardware architecture can be
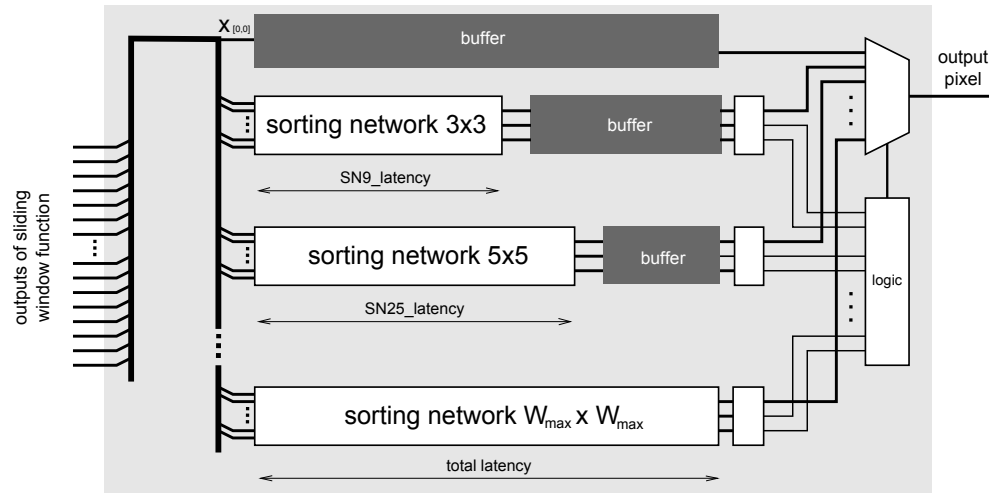
Figure 6.10: Hardware implementation of the adaptive median filter based on sorting networks [218]

optimized and only one sorting network that provides all the required values can be utilized. Such a sorting network can be designed using a cascade of unbalanced sorting networks.

Table 6.2: Results of synthesis of proposed adaptive median filter

W_{max}	SN bitonic		SN oe-merge		Latency [delay]
	# slices	max. freq	# slices	max. freq	
5x5	2220	305 MHz	2024	303 MHz	15
7x7	7297	302 MHz	6567	298 MHz	21
9x9	18120	302 MHz	16395	298 MHz	28

Results of synthesis of the pipelined adaptive median filter are summarized in Table 6.2. The adaptive median filters were described in VHDL, simulated using ModelSim and synthesized using Xilinx ISE tools to Virtex II Pro XC2VP50-7 FPGA. The implementation cost is expressed in terms of slices. The utilized FPGA contains 23616 slices in total. Adaptive median filter with filtering window 7x7 exhibits a very good performance/cost ratio in comparison to standard median filters. This filter occupies approx. 30% of the chip and is able to remove noise up to 60% intensity. As the design of AMF is pipelined and without iterations, it provides the same performance as standard median filters (i.e. approx. 300M processed pixels per second).

6.2 Evolutionary Design of Image Filters using CGP

Firstly, let us describe the evolutionary method that has been proposed by Sekanina in [158] and utilized to create image filters with the 3×3 pixel filter window. In the next chapter, we will introduce several extensions that have been proposed to evolve the filters with better filtering properties.

Every image filter is considered as a function (a digital circuit in the case of hardware implementation) of nine 8-bit inputs and a single 8-bit output, which processes grayscale

images. As Figure 6.11 shows, every pixel value of the filtered image is calculated using a corresponding pixel and its eight neighbors in the processed image. In order to evolve an image filter which suppresses a given type of noise from corrupted image, we need (a) training data consisting of corrupted and original version of the image that will be used to determine the fitness score of the candidate filters (i.e., to evaluate the quality of any candidate filter) and (b) a set of suitable 8-bit functions that can be used by CGP. The generality of evolved filters (i.e., the ability to operate sufficiently also for other images containing the same type of noise the filters have not been trained for) is tested by means of a test set.
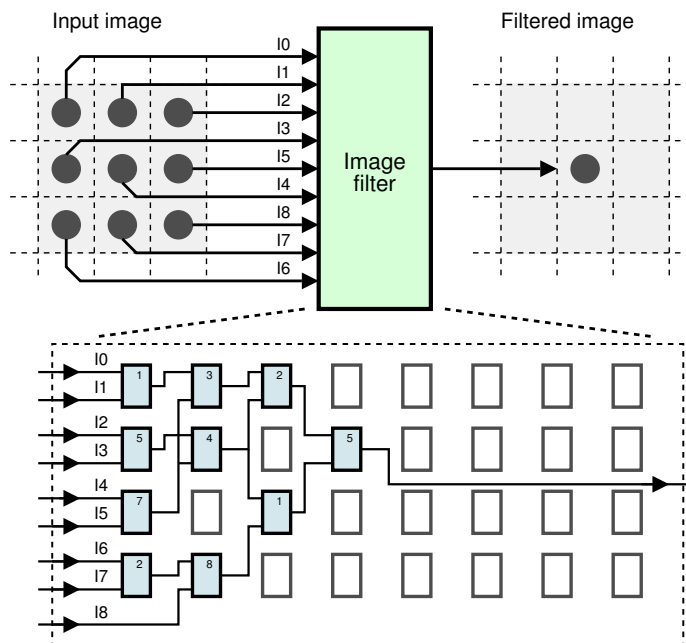
Figure 6.11: The concept of evolutionary design of image filters that utilizes the 3×3 filter window

6.2.1 Encoding of a Candidate Filter

A candidate filter is represented using $n_c \times n_r$ nodes arranged in a grid, where a typical grid size is $(n_c = 8) \times (n_r = 4)$. The setting of other CGP parameters is: $n_i = 9$, $n_o = 1$, $n_a = 2$, $\lambda = 8$ and $l = n_c$. Each node represents a two-input function which receives two 8-bit values and produces an 8-bit output. The 8-bit node output is utilized to ensure a straightforward connectivity of the nodes in hardware. Table 6.3 lists the functions that were confirmed as useful for this task [158]. We note that these functions are also suitable for hardware implementation (i.e. there are no complex functions, such as multiplication or division). A node input may be connected either to an output of another node, which is placed anywhere in the preceding columns or to a primary input of the filter. The filter circuits are encoded as array of integers of the size $3 \cdot n_r \cdot n_r + 1$.

6.2.2 Fitness Function

In order to evolve an image filter capable of removing a given type of noise, the original uncorrupted image is needed to determine the fitness values of candidate filters. The goal of CGP is to minimize the difference between the original image and the filtered image.

Usually, the fitness function is implemented as the *mean difference per pixel* also known as the *mean absolute error*. Let u denote a corrupted image, v the filtered image and w the original (uncorrupted) version of u. Let the image size be $M \times N$ pixels. Due to the one pixel neighborhood in kernel, the area of $(M-2) \times (N-2)$ pixels is processed only. Without loss of generality, the pixel values at the borders are ignored and thus remain unfiltered. The fitness value of a candidate filter is obtained by calculating the error function:

$$fitness = \frac{1}{(M-2)(N-2)} \sum_{i=1}^{M-2} \sum_{j=1}^{N-2} |v(i,j) - w(i,j)|.$$

The objective is to design a filter producing images with minimal error, i.e. the lower fitness value the better filter. Note that it is practically impossible to obtain a filter possessing the zero fitness value (i.e. an ideal filter) since the filter manipulates with corrupted images only (i.e. missing and incomplete information) and it can not predict the original values perfectly for an arbitrary input image. Only in rare cases (e.g. a training image with simple pattern), it is possible to evolve a filter that exhibits the zero fitness value but this filter will not be probably robust (i.e., it will work only for the selected training image). Thus, if a candidate filter fulfills a given criterion of quality (e.g. the mean difference per pixel is less than a predefined error ε), it is usually considered as a solution to the problem.

It is evident that the robustness of evolved filter depends on the selection of the training data. Thus, generality of evolved filters (i.e., whether the filters can operate sufficiently also for other images containing the same type of noise) has to be tested by means of a test (validation) set. There exists several metrics for expressing of the quality of filtering. For this purposes, the peak signal-to-noise ratio (PSNR) or mean square error (MSE) is usually used in image processing. PSNR is defined as

$$\text{PSNR} = 10 \log_{10} \frac{255^2}{\frac{1}{MN} \sum_{i,j} (v(i,j) - w(i,j))^2} \tag{6.3}$$

Table 6.3: List of functions implemented in each programmable node

code	function	description	code	function	description
0	255	constant	8	$x \gg 1$	right shift by 1
1	x	identity	9	$x \gg 2$	right shift by 2
2	$255 - x$	inversion	10	$swap(x,y)$	swap nibbles
3	$x \vee y$	bitwise OR	11	$x + y$	+ (addition)
4	$\bar{x} \vee y$	bitwise $\bar{x}$ OR y	12	$x +^S y$	+ with saturation
5	$x \wedge y$	bitwise AND	13	$(x + y) \gg 1$	average
6	$\overline{x \wedge y}$	bitwise NAND	14	$max(x,y)$	maximum
7	$x \oplus y$	bitwise XOR	15	$min(x,y)$	minimum

103

where $N \times M$ is the size of image, v denotes the filtered image and w denotes the original image. The higher PSNR value the better filter. Note that the denominator represents the mean square error. It has been shown experimentally, that a suitable image containing 128×128 pixels provides the sufficient amount of training data for evolution of robust 3×3 salt-and-pepper filters [119].

6.3 Experimental Results

As it has been discussed in previous chapters, the evolutionary design of image filters is a time consuming process. There is large amount of training data that have to be evaluated in order to determine the fitness value. In order to speed up the evolutionary design process and give the evolutionary algorithm possibility to explore large portion of search space, we have proposed an FPGA-based accelerator which hardware architecture is described in Chapter 7.

The aim of the first experiment is to apply the proposed accelerator to evolve filters for salt-and-pepper noise working with the 3×3 pixel filter window. Then, in order to improve the filtering properties, we have combined several filters and create a more robust bank filter working with the 3×3 pixel filter window. As it will be shown, this filter is able to compete with conventionally used filters working with larger filter windows, however, the resulting filter is probably the best solution that can be obtained using 3×3 pixel filter window. Thus, the last section deals with the evolutionary design of filters utilizing concept of so called switching filters. The common goal of these experiments is to experimentally evaluate whether the evolutionary design is able to discover solutions that exhibit better properties in terms of filtering capability and implementation cost comparing to the conventionally used approaches.

6.3.1 Evolutionary Design of Salt-and-pepper Noise Filters and Noise-Resistant Edge Detectors

In order to evaluate the performance of the proposed FPGA-based evolutionary platform, we have arranged four experiments. The objective was to evolve filter for salt-and-pepper noise of (1) 5%, (2) 10% and (3) 20% intensity and (4) design an edge detector which is able to deal with input images corrupted by the salt-and-pepper noise. Note that a 3×3 filter window is considered in all cases. Except the 5%-salt-and-pepper noise, the other problems were not approached by means of evolutionary design techniques in literature. In order to compare the quality of the obtained filters with the results published in [161, 160], visual quality of filtered images is expressed in terms of mdpp which stands for the mean difference per pixel between the filtered image and original image.

The following experimental setup was applied. The CGP array was comprised of 4×8 programmable nodes. Each CGP node can implement one of the sixteen 8-bit functions listed in Table 6.3. The l-back parameter was set to one; i.e. the inputs of a certain node can be connected either to the output of a node situated in the previous column or to the primary input. The chromosome consists of 384 bits; a single node is configured using 12 bits. Thus the search space contains 2^{384} possible solutions.
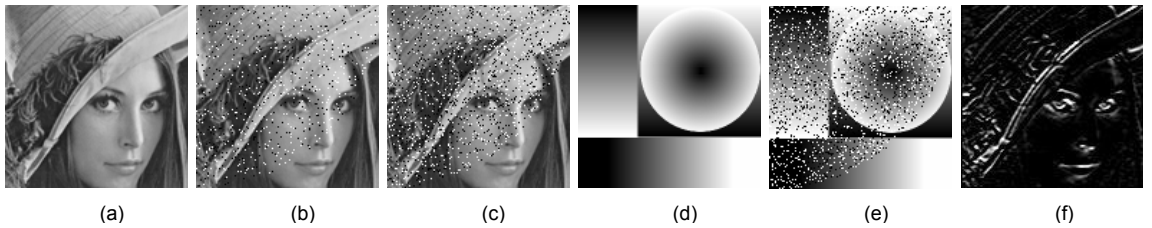
Figure 6.12: Training images utilized for the evolutionary design of 5% (a,b), 10% (a,c) and 20% (d,e) salt-and-pepper noise filter, and the evolutionary design of noise-resistant edge detector (f,b).

The population contains eight individuals ($\lambda = 8$). The initial population is generated randomly. Then, two offspring are generated from each parent using a bit-mutation operator. A new population is selected from the eight parents and their sixteen offspring. We utilized a deterministic selection in which the eight-best scored individuals are selected as new parents. The evolutionary algorithm utilizes a single genetic operator – mutation, which is applied with the probability of 4.7–6.3% per bit. This mutation intensity was experimentally confirmed as the most suitable. No crossover operator is utilized in this type of EA [158].

Table 6.4: Parameters of the experiments

exp. no	input image	target image	bits mutated	runs (N_r)	evaluations per run
E1	5%-noise Lena128	Lena128	18	64	160,000
E2	10%-noise Lena128	Lena128	24	349	320,000
E3	20%-noise Lena128	Lena128	20	139	320,000
E4	5%-noise Lena128	edges in Lena128	18	389	160,000

The evolution is stopped when a predefined number of evaluations is exhausted. Table 6.4 provides the parameters that have been used in each experiment. The parameters have been determined experimentally [215]. Because it is intractable to evaluate all possible input combinations, there exists $2^{9 \cdot 8} = 2^{72}$ possible input vectors, approximate synthesis scenario has been applied. As a training image, we used the 128×128-pixel version of Lena image referred to as Lena128 which is corrupted with a given type of noise in some regions. Figure 6.12 depicts the training images that have been applied in this experiment.

Table 6.5: The fitness value of the evolved filters for the four test problems expressed in terms of mean difference per pixel (mdpp)

exp. no	training image	best solution	worst solution	average solution	conventional approach
E1	6.049	0.410	3.190	0.967 ±0.581	4.796 (median)
E2	12.382	0.982	3.280	1.720 ±0.337	5.207 (median)
E3	25.766	1.870	4.350	2.850 ±0.510	6.383 (median)
E4	n/a	1.100	2.660	1.910 ±0.419	11.329 (sobel)

Table 6.5, which summarizes the obtained results, contains the fitness values of the best, worst and an average solution. The statistics is calculated from the N_r independent evolutionary runs. The number of runs for each experiment is given in Table 6.4. The last column of Table 6.6 gives the results of conventional filters. As it can be seen, the evolutionary designed filters significantly outperform the conventional solutions.

Table 6.6: Comparison of mdpp of the best evolved filters and 3×3 median filter on a test set of 256×256 images.

test	5% noise			10% noise			edge detection	
image	evolved	MF	AMF	evolved	MF	AMF	evolved	SO
Airplane	0.338	3.536	1.046	0.874	3.843	1.227	0.988	2.902
Bird	0.147	1.514	0.598	0.389	1.648	0.651	0.467	2.827
Bridge	0.657	7.830	2.545	1.386	8.165	2.765	1.688	2.856
Camera	0.627	4.413	1.589	0.850	4.746	1.707	1.108	2.786
Goldhill	0.451	5.870	2.053	0.962	6.134	2.191	1.161	2.812
Lena	0.367	3.577	1.209	0.863	3.893	1.437	1.022	2.832

Table 6.6 compares mdpp of the best-evolved filters and conventional filters (median filter denoted as MF, adaptive median filter denoted as AMF and Sobel operator denoted as SO) on a set of 256×256-pixel test images. In order to fairly evaluate the quality, all the filters use the 3×3 filter window. Even if the evolutionary design does not in general guarantee robustness of the evolved filters (i.e. that they exhibit a similar quality independently on the image content), the obtained results show that the evolved filters exhibit very good performance not only for the training images but also for the testing images. As it can be seen, the results outperform not only the common median filter but also the more advanced adaptive median filter.
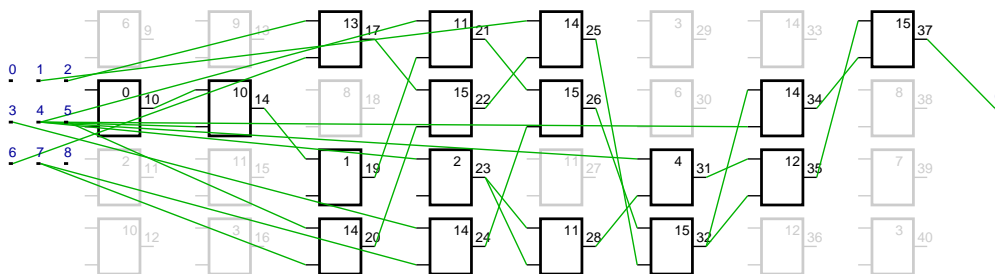
Figure 6.13: The best evolved filter for 5% salt-and-pepper impulse noise.

Figures 6.13 and 6.14 depict structure of the best evolved filters for 5% and 10% respectively salt-and-pepper noise. Figure 6.15 gives examples of images filtered using the best-evolved filters. As it can be seen, the images filtered by evolved filters are not as smudged as the images filtered by median filters. The most significant improvement has been achieved in the last experiment which combines the edge detector with noise removal. If a conventional edge detector is applied to the image corrupted by the salt-and-pepper noise, the noisy pixels are significantly amplified.
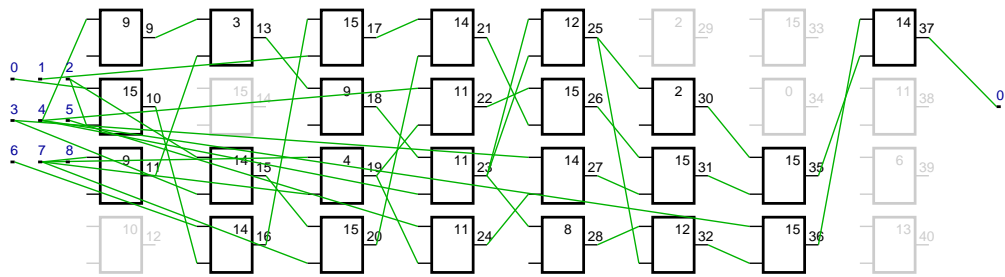
Figure 6.14: The best evolved filter for 10% salt-and-pepper impulse noise.

(a) corrupted image   (b) conventional filter   (c) evolved filter   (d) desired output

(e) corrupted image   (f) conventional filter   (g) evolved filter   (h) desired output

(i) corrupted image   (j) conventional filter   (k) evolved filter   (l) desired output
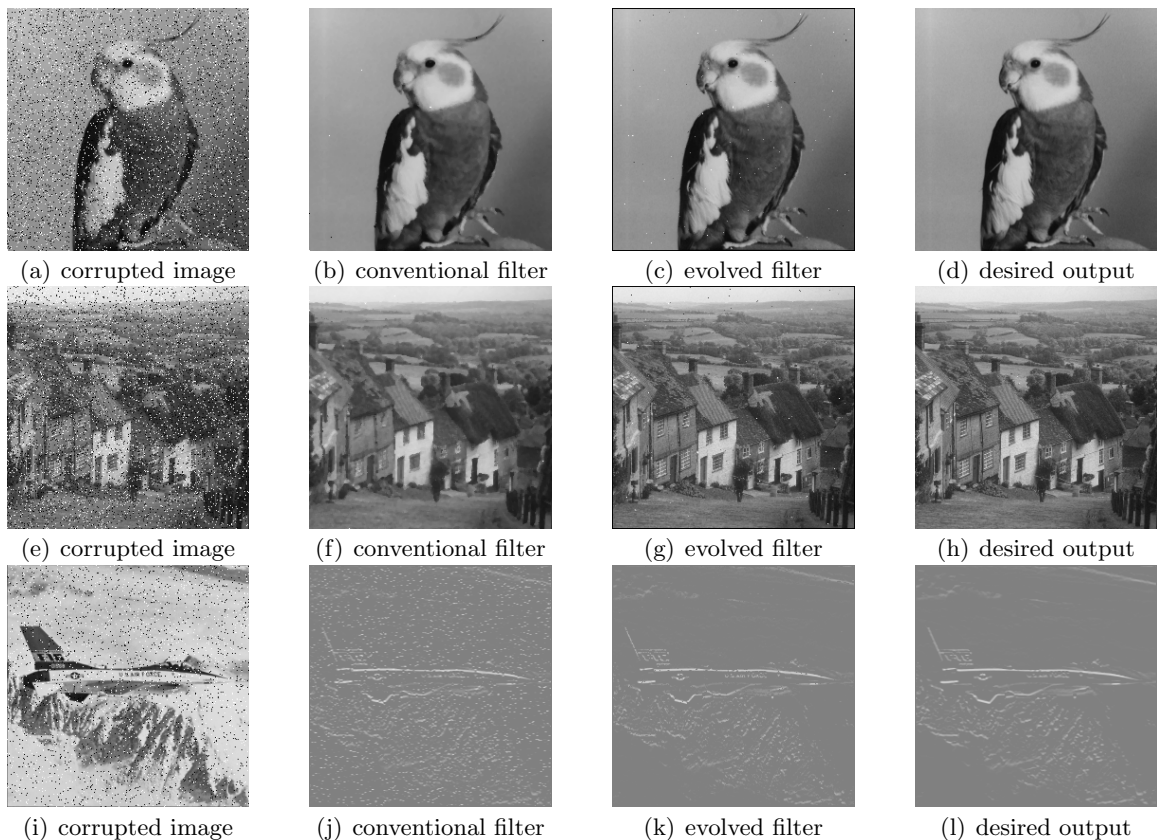
Figure 6.15: The Bird (a-d) and Goldhill (e-h) images from test set in the 10% salt-and-pepper noise removal task. The edge detection in images corrupted by the 5% salt-and-pepper noise, the Airplane image (i-l).

6.3.2   Evolutionary Design of Robust Salt-and-pepper Noise Filter

Even if the results presented in the previous section clearly demonstrate that the evolutionary design approach is able to automatically evolve competitive image filters, unfortunately this method is not able to evolve competitive filters for higher noise intensities (e.g. 40%). It may seem that this failure is caused by the insufficient information provided by the 3×3 filter window. On the other hand, the missing information can be derived from the unaffected pixels since there are five out of nine pixels (in average) that are unaffected by the

noise if a 40% noise level is considered. The goal of this experiment is to experimentally evaluate whether is it possible to design an image filter working with 3×3 filter window that can provide sufficient quality even for higher noise intensities. And moreover, to provide a solution suitable for pipelined hardware implementation in FPGA that can compete to the quality of adaptive median filter.

The main feature of evolutionary design of image filters is that each evolutionary run typically produces a solution having different structure as well as properties. This behavior is usually undesirable because the evolutionary design does not guarantee the evolved filters are robust (i.e. that the proposed filters exhibit the constant quality for the whole class of images corrupted with a given type of noise). However, this feature can be exploited to create robust image filters.
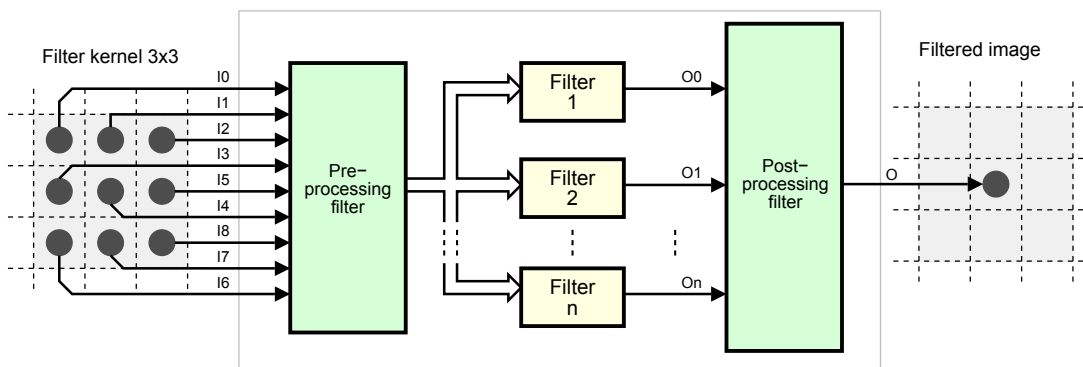

Figure 6.16: The proposed architecture for the impulse noise removal

In order to create more robust salt-and-pepper noise filter, we have proposed to combine several simple image filters utilizing the 3×3 window that are designed by an evolutionary algorithm. As Figure 6.16 shows the procedure has three steps: (1) the reduction of a dynamic range of noise, (2) processing using a bank of n filters and (3) deterministic selection of the best result.

The first step reduces the large dynamic range of corrupted pixels (0/255) using a component which inverts all pixels with value 255, i.e. all shots are transformed to have a uniform value. This task is easy to implement in hardware using comparators. This step has been introduced according to the analysis presented in [158] where we have recognized that the evolved salt-and-pepper noise filters have problems with the large dynamic range of corrupted pixels (0/255). Note that the comparators can be replaced with a more sophisticated algorithm that replaces the affected pixels with zero value which indicates the noisy pixel that should be replaced.

The preprocessed image then enters a bank of n filters which operate in parallel. We selected n evolved filters which produce different results and which exhibit better-than-average filtering quality and utilized them in the bank. Note that all these filters were designed by EA using the same type of noise and training image and with the same aim: to remove the 40% salt-and-pepper noise.

Finally, the outputs coming from banks $1 \ldots n$ are combined by n-input median filter which can be easily implemented using comparators [93].
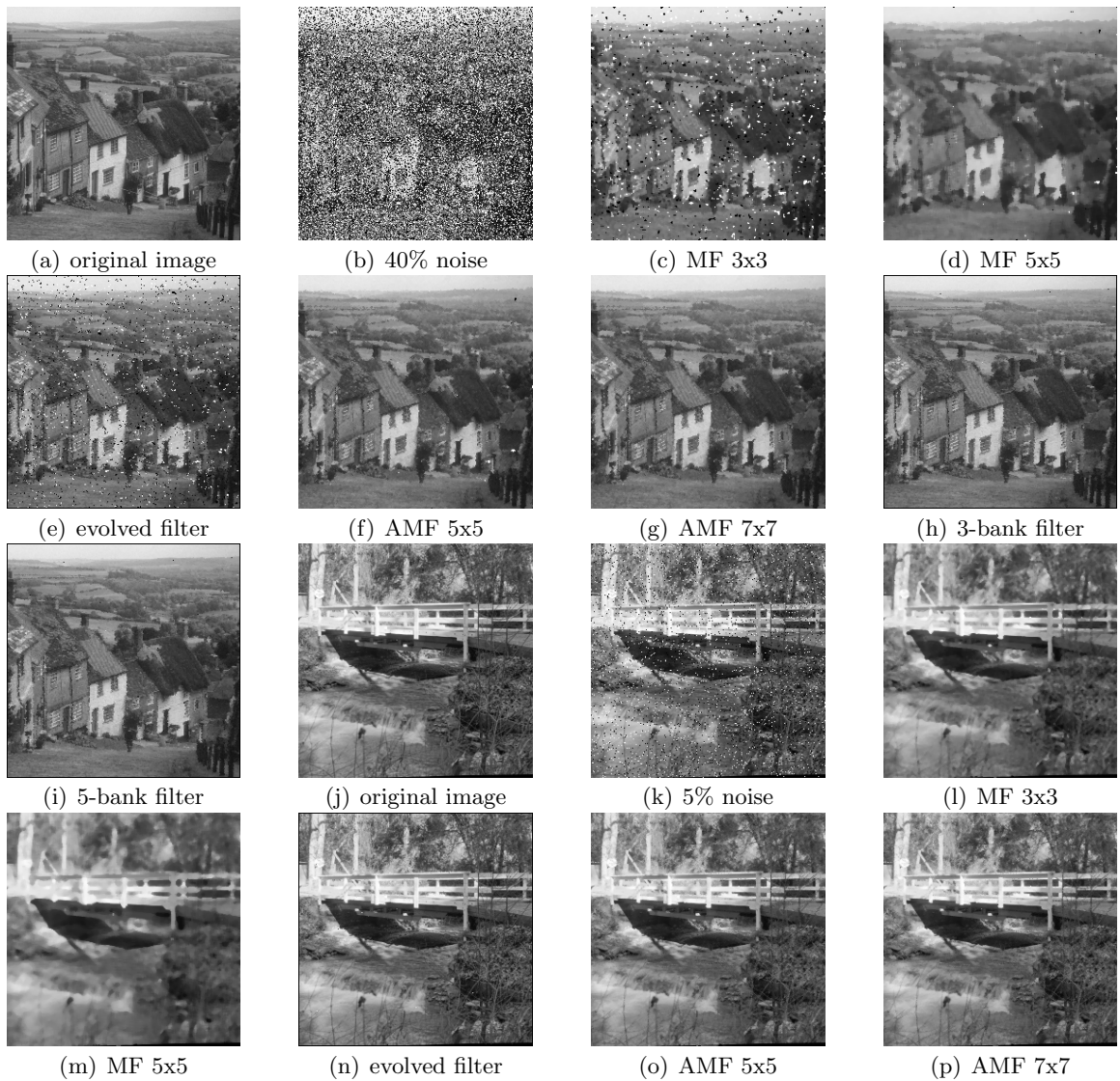
Figure 6.17: Filtering the images corrupted by 40% salt-and-pepper noise (a–i) and filtering the images corrupted by 5% salt-and-pepper noise (j–p)

In order to evolve the filters for the bank, we have applied the same experimental setup as in the previous experiment. As a training image we utilized 128×128-pixel version of Lena (Lena128) which was partially corrupted by 40% salt-and-pepper noise. Evolution was repeated 100 times; 1.5 million evaluations were performed in each run. CGP operated with an eight-member population and the 5% mutation. According to the chromosomes of the five best-scored filters we created corresponding pipelined VHDL models and synthesized them. The first part of Table 6.7 shows that the implementation cost of evolved filters is much lower than the cost of the 3×3 median circuit given in Table 6.1.

To evaluate the quality of the proposed bank, the proposed approach and adaptive median filters are compared on several test images of size 256×256 pixels which contain the salt-and-pepper noise with the intensity of 5%, 10%, 20%, 40%, 50% and 70% corrupted pixels. Table 6.8 summarizes the results obtained for selected test images and two versions

Table 6.7: Result of synthesis for evolved filters utilized in the bank (filter1-5) and for the whole bank filters

filter	# slices	area	max. frequency	latency
filter1	156	0.7%	316 MHz	8
filter2	199	0.8%	318 MHz	8
filter3	137	0.6%	308 MHz	8
filter4	183	0.8%	321 MHz	8
filter5	148	0.6%	320 MHz	8

filter	# slices	area	max. frequency	latency
3-bank	500	2.1%	308 MHz	11
5-bank	843	3.6%	305 MHz	13

of the adaptive median filter and two versions of the bank filter (which contain the filters from Table 6.7). The higher PSNR, the better results.

Surprisingly, only three filters utilized in the bank are needed to obtain a bank filter which produces images of at least comparable visual quality to the adaptive median filter. This fact is demonstrated by Figure 6.17f–i where the visual quality of the images filtered by the adaptive median and 3-bank filter is practically undistinguishable. The structure of the best evolved filters that are utilized in the 3-bank filter is shown in Figure 6.18.

An obvious question is how it is possible that three (five, respectively) filters utilizing a relatively small 3×3 filter window evolved with the aim of removing 40%-salt-and-pepper noise are able to suppress the salt-and-pepper noise with the intensity up to 70%? Moreover, none of these filters does work sufficiently in the task which it was trained for (the 40% noise). We are convinced that this success is caused by the fact that although these filters perform the same task, they operate in a different way. While the median filter gives as its output one of the pixels of the filtering window, evolved filters can sometime produce new pixel values. By processing these n-values in the n-input median, the shot is suppressed. We tested several variants of evolved filters in the bank but never observed a significant degradation in the image quality.

Table 6.8: PSNR for adaptive median filters with the kernel size up to 7×7 and bank filters containing 3 and 5 filters

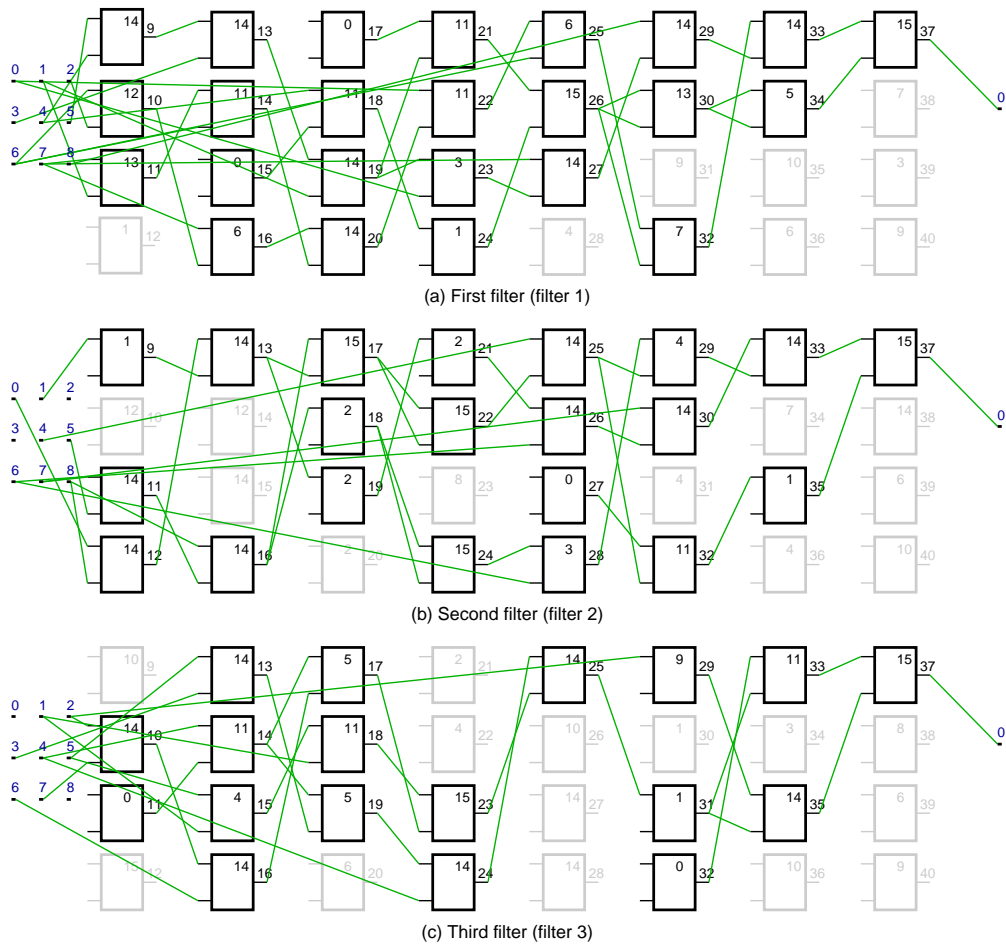|	Adaptive median filter 5×5						Bank filters 3-bank					
img/noise	5%	10%	20%	40%	50%	70%	5%	10%	20%	40%	50%	70%
goldhill	31.60	31.15	30.08	26.90	24.29	15.85	36.61	33.75	30.61	27.71	25.86	19.09
bridge	29.93	29.47	28.06	24.99	22.56	14.78	34.06	31.45	28.99	25.83	24.28	18.33
lena	34.43	33.66	31.21	27.17	24.43	15.46	31.42	30.30	28.16	25.68	24.13	18.32
pentagon	33.10	32.76	31.46	28.23	25.21	16.31	37.44	34.63	31.89	28.68	26.57	18.43
camera	30.86	30.36	28.56	25.14	22.67	14.97	34.25	30.57	28.18	25.28	23.72	17.85
	Adaptive median filter 7×7						Bank filters 5-bank					
img/noise	5%	10%	20%	40%	50%	70%	5%	10%	20%	40%	50%	70%
goldhill	31.60	31.15	30.08	27.31	25.96	20.88	37.21	34.39	31.13	27.96	25.96	19.07
bridge	29.94	29.47	28.05	25.17	23.71	19.06	34.82	32.32	29.71	26.12	24.44	18.32
lena	34.42	33.65	31.20	27.52	25.98	20.45	31.44	30.39	28.42	25.88	24.20	18.31
pentagon	33.10	32.76	31.46	28.62	27.17	21.65	38.01	35.20	32.41	28.94	26.68	18.43
camera	30.86	30.36	28.56	25.29	23.85	19.24	34.62	31.09	28.74	25.57	23.91	17.84

Figure 6.18: Three filters evolved for the 40% salt-and-pepper noise and utilized in the 3-bank filter

The results obtained for this class of images are quite promising from the application point of view. We can reach the quality of adaptive median filtering using a 3-bank filter; however four times less resources are utilized. This can potentially lead to a significant reduction of power consumption of a target system. Moreover, as the adaptive medians require larger filtering windows than bank filters they also require more logic to implement input FIFOs. For example, the adaptive median with kernel size up to 7×7 pixels needs seven input FIFOs. Each FIFO stores the whole row of the filtered image. This overhead is not included in the implementation cost of filters given in Tables 6.2 and 6.7. Note that the proposed filter bank, which can comprise an arbitrary number of evolutionary designed filters working in parallel, needs three input FIFOs only.

The extended version of the 3-bank filter, has been registered as Czech Utility Model under No. UV020017/2009. The proposed filter is optimized for high-performance impulse noise removal task. The proposed filter works in three phases. The first phase (detection phase) tries to identify the noisy pixels and replace them with a constant value. Some kind of inaccuracy is not crucial in this step. The second phase comprises the filtering of the
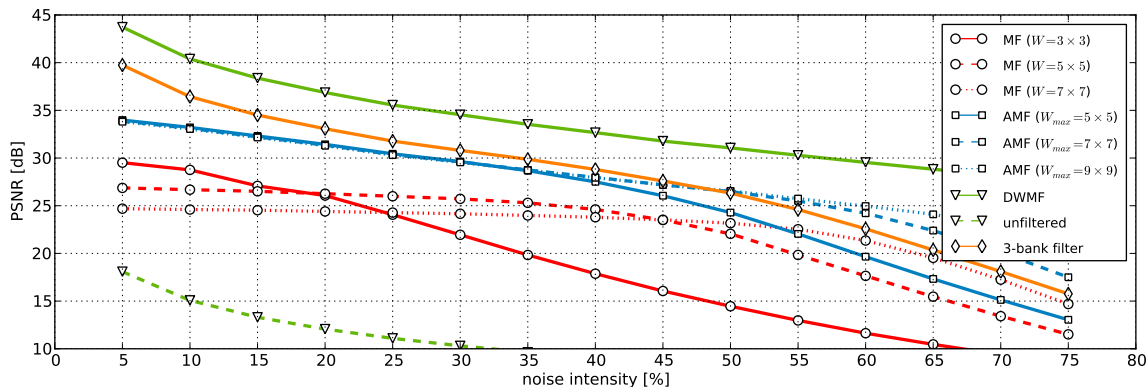
Figure 6.19: Comparison of various image filters and the extended version of the 3-bank filter using a set of 25 test images corrupted by salt-and-pepper noise of intensity 5-75%.

resulting image using a bank of evolutionary designed filters. Finally, a selection mechanism that determines the best result is applied. The resulting value is calculated according to the knowledge of the value of central pixel and the results produces by the evolutionary designed filters. Figure 6.19 summarizes the results of filtering properties of the proposed 3-bank filter, adaptive median filters (with filtering windows 5×5, 7×7 and 9×9), standard median filters (with filtering windows 3×3, 5×5 and 7×7) and the DWMF filter which utilizes filtering windows of unlimited size.

6.3.3 Evolutionary Design of Switching Filters

The main disadvantage of the common median-based filters is that the filtering transformation is applied on all the pixels of the image regardless if the pixel represents the noise or not. Thus this approach results in loss of the image details and causes the degradation of the image quality especially if a larger filter kernel is used (see Figure 6.4). In order to improve the filtering quality, so called switching-based median filter, which combines median filter with a noise detector, has been proposed in [173].
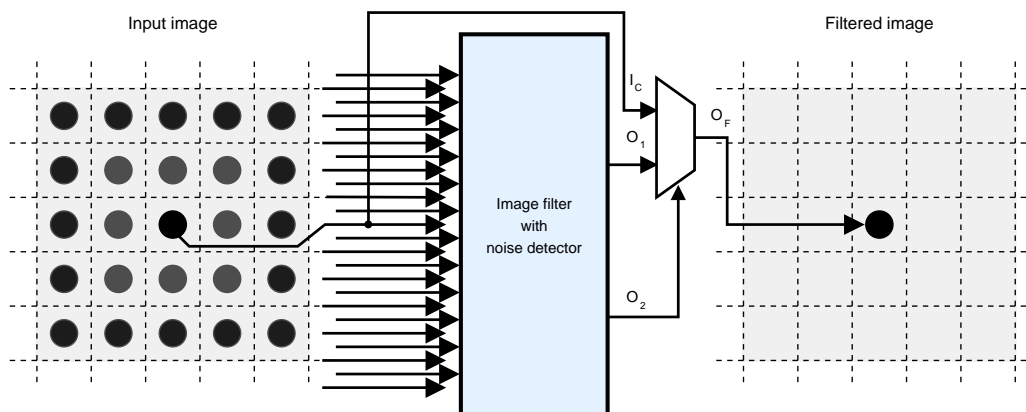

Figure 6.20: The structure of the filter under evolution that utilizes the concept of switching filter with 5×5 filter window.

Table 6.9: The list of functions that can be implemented in each programmable node

code	function	description	code	function	description
0	255	constant	7	$x + y$	addition
1	x	identity	8	$x +^S y$	addition with saturation
2	$255 - x$	inversion	9	$(x + y) \gg 1$	average
3	$max(x, y)$	maximum	10	y if $(x > 127)$ else x	conditional assignment
4	$min(x, y)$	minimum	11	$|x - y|$	absolute difference
5	$x \gg 1$	division by 2	12	$x \ll 1$	multiplication by 2 with saturation
6	$x \gg 2$	division by 4	13	$x \ll 2$	multiplication by 4 with saturation

The goal of this work is to experimentally evaluate whether it is possible to design filters that are able to compete with conventionally used non-iterative as well as iterative filters suitable for the impulse noise removal task. In addition to the salt-and-pepper noise, random-valued impulse noise and impulse burst noise will be investigated. The objective is to design filters based on the switching concept. In particular, the evolutionary algorithm has to design a filter system consisting of a noise detector and nonlinear image filter. Both parts are evolved in parallel. This enforces the node sharing. Similarly to the previous experiments, the filters should be suitable for FPGA-based implementation.

In our case, the image filter produces filtered value O_1 and noise detector output O_2, both are 8-bit values. The structure of the image filter is illustrated in Figure 6.20. The MSB of O_2 controls the multiplexer that implements the switching algorithm. The switching filter works as follows. If O_2 is less than 128 (i.e. the MSB of O_2 equals to 0), then the value I_C was detected as noise and the final output of the filter O_F equals the filtered value O_1, otherwise O_F equals the original value I_C. In fact, the noise detector represents an additional logic of the filter circuitry that is capable of determining whether the value of the pixel to be filtered is a noise value or a correct (uncorrupted) value.

The following experimental setup was used. In order to evolve an image filter, the CGP at the functional level was utilized. The CGP array consists of $n_c \times n_r = 7 \times 9$ nodes. Each node can implement one of the high-level functions listed in Table 6.9. The l-back parameter has been set to $l = n_c$ (i.e. the full connectivity has been enabled). Only the elements situated in the first four columns can be connected directly to the primary inputs.

The evolutionary algorithm works with the population of $\lambda = 8$ individuals. Up to 15 genes in an individual can be mutated. The initial population is generated randomly. The results were obtained from 100 independent runs of the CGP system. Each single experiment takes 200,000 generations. This artificial limit has been chosen in order to provide the tradeoff between the quality of the evolved filters and the time needed for the evolution. The goal of this experiment was to confirm the hypothesis instead of find the best possible solution. The objective is to design filters working with 5×5-pixel kernel that are optimized for the removal of a) salt-and-pepper, b) random-valued impulse noise and c) impulse burst noise.

For the first three problems, an artificial image corrupted by 20% noise consisting of 256×256 pixels was used as training data. The training images for the investigated problems are shown in Figure 6.21. We carried out an analysis of the images and recognized that the training data set for the evolutionary design of a salt-and-pepper noise filter consist of 62,061 unique training vectors extracted from Figure 6.21b. The training data for the
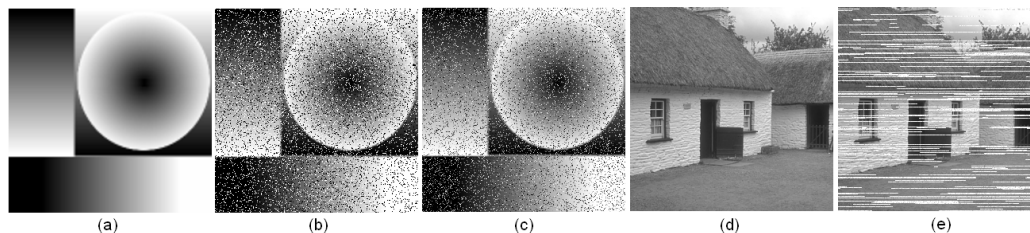
Figure 6.21: The training data utilized in the experiments: (a,d) the reference image, (b) the image corrupted by 20% salt-and-pepper noise, (c) the image corrupted by 20% random-valued impulse noise and (e) the image corrupted by 20% impulse burst noise. The noise intensity 20% means that 20% of the total number of pixels of the reference image is corrupted by the noise.

evolutionary design of a random-valued impulse noise filter comprise 63,437 unique training vectors extracted from Figure 6.21c. Note that the utilized images can provide up to $252 \cdot 252 = 63504$ different training vectors if a filter window of 5×5 pixels is considered. The artificial image has been utilized because it contains a representative sample of training vectors possessing crucial features; for example smooth gradients of different types combined with sharp edges. These components showed to be important for the filter training. The previous experiments showed that if the noise intensity is low during the training process ($\leq 10\%$), the evolutionary design approach can not guarantee that the evolved filter works also for a high intensity noise. On the other hand, if the training noise intensity is high ($\geq 30\%$), substantial amount of (training) image data is lost and the resulting filters do not exhibit reasonable filtering quality. During the evolutionary design of impulse burst noise filter, a training image consisting of 384×256 pixels selected from [16] was used. This image contains 92,813 unique training vectors. The left part of this image is shown in Figure 6.21e. The impulse burst noise model corresponds to the following parameters: $p = 0.01 \div 0.3$, $q = 128$, $\sigma = 30$.

The quality of the evolved filters is compared with several conventional single-step and iterative filters that are known to provide good results in removing of the particular type of impulse noise. In order to show the ability of the evolved solutions to improve the filtered image using the iterative processing, one and two iterations of these filters will be performed. The results are compared to the images filtered by the conventionally used approaches such as DWMF and PWMAD. Moreover, the filtering results will also be compared to standard median filter (MF) and adaptive median filter (AMF). We have used the recommended settings of the PWMAD filter, i.e. five iterations, the value of threshold was set to 5 and the filter window consisting of 5×5 pixels has been utilized. In case of the impulse burst noise, center weighted median filter (CWMF) has been also included in the comparison. We have chosen the center weight equal to 3 and the kernel size 3×3 pixels.

The evolved filters were evaluated using a set of 30 randomly selected images from [16], each of which was corrupted by noise of 1%–30% intensity. Therefore, in total more than 200 test images were utilized during the evaluation process. The filtering quality (expressed by means of PSNR) for each noise intensity is calculated as average of the PSNR for each image in the evaluation set.

The experiments were conducted on a cluster consisting of more than 200 PCs (Xeon E5345, 2.33GHz, 8GB RAM) using the Sun Grid Engine (SGE) that enables to run all the experiments in parallel. A highly optimized software implementation of CGP described in Chapter 3.3.1 has been utilized. The evolution time of a single run is approximately 8 hours until the CGP algorithm reaches 200,000 generations.

Salt-and-pepper noise

Table 6.10 summarizes the obtained results for the evolutionary design of switching salt-and-pepper noise filter. The evolved filter is denoted as F18. The results show that the evolutionary designed filter exhibits the best results for lower noise intensity (1%–15%) in comparison with the conventional filters. For higher noise intensity (i.e. greater than 20%) the AMF produces the images with the highest values of PSNR. However, the difference between F18 and AMF for these noise intensities is negligible.

Table 6.10: Comparison of the salt&pepper noise filters in terms of mean PSNR (dB). The size of the kernel is also specified for each filter.

filter	noise intensity						
	1%	5%	10%	15%	20%	25%	30%
F18 5x5	**39.0**	**36.4**	33.7	31.2	28.5	25.9	23.4
F18, 2 iter.	38.1	35.9	**34.0**	**32.5**	31.3	30.1	29.0
PWMAD 3x3	33.0	32.4	30.7	27.7	24.5	21.6	19.1
PWMAD 5x5	29.0	28.9	28.8	28.4	27.7	26.2	24.0
DWMF 5x5	28.8	28.3	27.8	27.2	26.6	25.9	25.0
AMF 5x5	34.3	33.9	33.2	32.2	**31.4**	**30.5**	**29.5**
MF 5x5	26.5	26.4	26.2	26.0	25.8	25.6	25.3
unfiltered	25.1	18.1	15.1	13.3	12.1	11.1	10.3

In order to demonstrate the visual quality, Figure 6.22 contains example of the filtered images for the input image corrupted by 15% and 30% salt-and-pepper noise. In addition to the evolutionary designed filter, the conventionally used approaches are also included in this comparison. Note that the standard median filter as well as the DWMF filter are not included in this comparison, since the filtered images are smudged (see the results given in Table 6.10). Whilst the evolved filter F18 provides a very good result of the filtered image even if a single iteration is utilized, the iterative PWMAD filter leaves a significant amount of noisy pixels in the resulting image (see Figure 6.22b,i,j).

The evolved filter F18 provides very good results even for images corrupted by 30% salt-and-pepper noise. The visual quality is comparable to the AMF (see Figure 6.22g,h). A single application of F18 is not sufficient to obtain the best quality for the image corrupted by 30% noise (as evident in Figure 6.22d). However, images filtered by F18 filter exhibit better quality in comparison with the PWMAD filter that leaves a lot of noise in the filtered image and makes a loss of some detail (see Figure 6.22d,k).

To summarize the obtained results, the CGP-based evolutionary system succeeded in searching a robust salt-and-pepper noise filter whose filtering quality can compete with the iterative filters and especially the adaptive median filter even for high noise intensity.
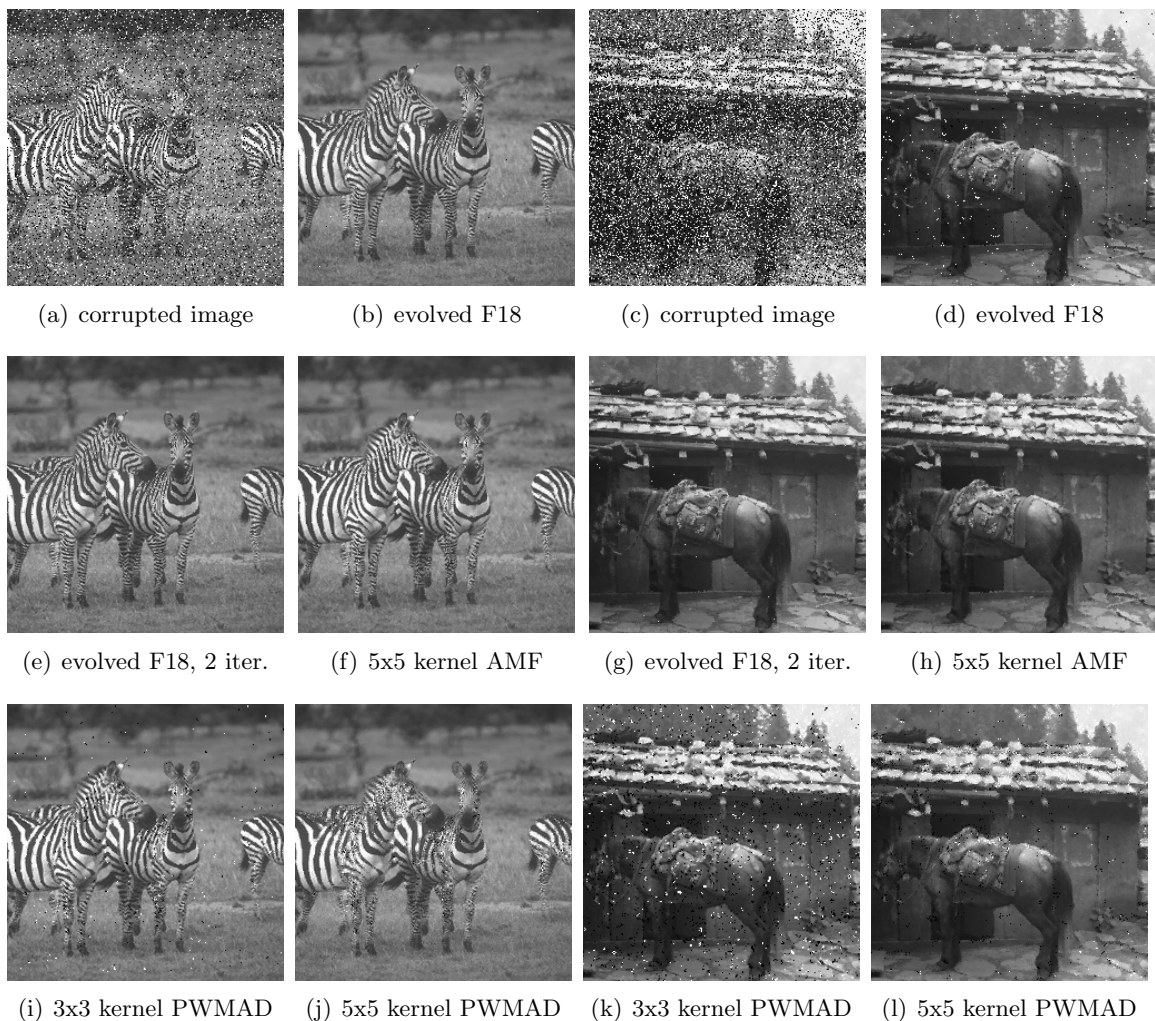
(a) corrupted image   (b) evolved F18   (c) corrupted image   (d) evolved F18

(e) evolved F18, 2 iter.   (f) 5x5 kernel AMF   (g) evolved F18, 2 iter.   (h) 5x5 kernel AMF

(i) 3x3 kernel PWMAD   (j) 5x5 kernel PWMAD   (k) 3x3 kernel PWMAD   (l) 5x5 kernel PWMAD

Figure 6.22: Filtering the images corrupted by 15% (a,b,e,f,i,j) and 30% (c,d,g,h,k,l) salt-and-pepper noise using different filters

Random-valued impulse noise

The random-valued noise represents a more realistic type of impulse noise in which the corrupted pixels can take an arbitrary value from the entire scale available for the given class of images. Therefore, in this case the noise may be represented by an arbitrary grayscale value from 0 to 255. It is thus more difficult to distinguish between the noisy and the uncorrupted pixels. In the consequence of a false detection, a filter may have a tendency to deteriorate the overall quality of the filtered image.

The obtained results for the evolutionary design of switching random-valued impulse noise filters are summarized in Table 6.11. The best evolved filter optimized for random-valued impulse noise is denoted as F17. Similarly to the previous experiment, the obtained results are compared with the results of conventional filters. Surprisingly, the evolutionary approach succeeded in the search for a robust filter for the noise of this type. As expected, the higher noise intensity requires more iterations of the filter to obtain an acceptable result.

Table 6.11: Comparison of the random valued noise filters in terms of mean PSNR (dB). The size of the kernel is also specified for each filter.

filter	noise intensity						
	1%	5%	10%	15%	20%	25%	30%
F17 5x5	**36.0**	**33.4**	30.9	28.7	26.6	24.7	23.0
F17, 2 iter.	34.4	32.5	30.9	**29.6**	**28.4**	**27.2**	25.9
PWMAD 3x3	33.1	32.5	**31.2**	29.5	27.4	25.3	23.3
PWMAD 5x5	29.1	29.0	28.7	28.3	27.8	27.0	**26.0**
DWMF 5x5	28.9	28.4	27.8	27.3	26.8	26.2	25.7
AMF 5x5	33.9	30.0	26.0	23.3	21.2	19.6	18.3
MF 5x5	26.6	26.5	26.3	26.1	25.9	25.5	25.2
unfiltered	28.5	21.5	18.5	16.7	15.5	14.5	13.7

The comparison of the visual quality of different filters considering images corrupted by the random-valued noise is shown in Figure 6.23. Very good results can be obtained using two iterations of the evolved filter F17 or conventional PWMAD filter with the 5×5 filter window. Unlike the case of the salt-and-pepper noise, the adaptive median filter fails in filtering random-valued noise even for lower intensity. On the other hand, the conventional DWMF filter exhibits a good quality slightly loosing some detail in comparison with the proposed F17 (compare Figure 6.23e–h).

Impulse burst noise

In comparison with the previous types of noise, impulse burst noise represents a serious issue because the principle of spatial locality is violated in this case. With the increased noise intensity, more consecutive rows may be affected and subsequent noise filtering becomes difficult as the filtered value need not be determined according to the values of the neighboring pixels.

Table 6.12: Comparison of the impulse burst noise filters in terms of mean PSNR (dB). The size of the kernel is also specified for each filter.

filter	noise intensity									
	1%	3%	5%	7%	10%	12%	15%	20%	25%	30%
F32 5x5	**35.4**	**33.9**	**33.0**	**31.9**	**30.9**	**29.8**	**28.7**	**27.0**	25.3	23.4
F32, 2 iters	32.6	31.6	30.9	30.1	29.3	28.7	28.0	**27.0**	**25.9**	**24.7**
CWM 3x3	32.5	30.2	27.1	24.8	21.7	20.1	18.7	16.2	14.6	13.3
PWMAD 5x5	32.3	29.9	26.6	24.3	21.2	19.6	18.2	15.8	14.3	12.9
DWMF 5x5	26.5	24.3	22.4	21.2	19.5	18.4	17.4	15.6	14.4	13.3
AMF 5x5	26.8	23.3	20.7	19.5	17.6	16.7	15.7	14.1	13.0	12.0
MF 3x3	29.4	28.0	25.6	23.8	21.0	19.6	18.2	15.8	14.3	12.9
unfiltered	25.3	21.4	18.8	17.6	15.9	15.1	14.2	12.9	12.0	11.2

The results for the evolutionary design of switching impulse burst noise filter are summarized in Table 6.12. The proposed filter F32 exhibits the best filtering quality in comparison with the commonly used conventional filters, surprisingly, even if we do not apply iterative

(a) corrupted image   (b) evolved F17   (c) corrupted image   (d) evolved F17

(e) evolved F17, 2 iter.   (f) 5x5 kernel DWM   (g) evolved F17, 2 iter.   (h) 5x5 kernel DWM

(i) 3x3 kernel PWMAD   (j) 5x5 kernel AMF   (k) 5x5 kernel PWMAD   (l) 5x5 kernel AMF
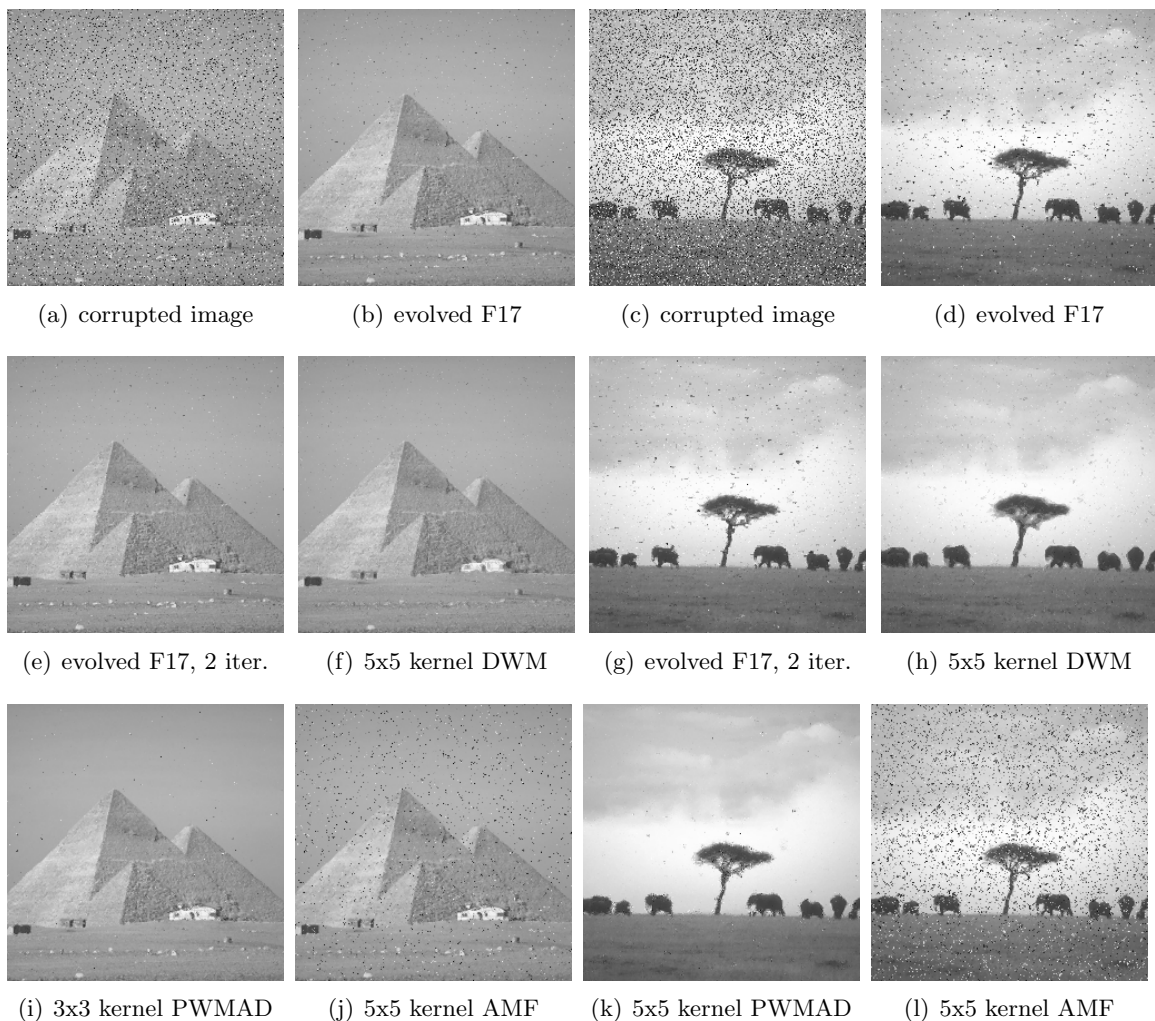
Figure 6.23: Filtering the images corrupted by 15% (a,b,e,f,i,j) and 30% (c,d,g,h,k,l) random-valued impulse noise using different filters

filtering. We have analyzed the best evolved filter (denoted as F32) and recognized that this filter tries to avoid the calculations based solely on horizontal information [213]. This interesting property shows that the evolutionary approach was able to detect, that the burst noise affects adjacent horizontal pixels. If the burst noise has been applied in vertical direction, the pixels of filter window utilized by the evolved filters changed.

Figure 6.24d shows that the proposed filter exhibits very good quality even for the lower noise intensity – in this case 1% impulse burst noise was generated. In case of the image filtered by the PWMAD filter, we can see that a perceptible part of the noise remains in the image (see Figure 6.24c). In contrary, the image produced by the CWMF does not contain any impulse; however, the image is smudged and lacks the details in comparison with the original image.

Similar comparison was performed considering the images corrupted with higher noise intensity (30% impulse burst noise). The obtained results are shown in Figure 6.24e-h.

(a) 5x5 kernel AMF (b) 3x3 kernel CWMF (c) 3x3 kernel PWMAD (d) evolved F32

(e) 5x5 kernel AMF (f) 3x3 kernel CWMF (g) 3x3 kernel PWMAD (h) evolved F32

(i) corrupted image (j) 3x3 kernel CWMF (k) 3x3 kernel PWMAD (l) evolved F32
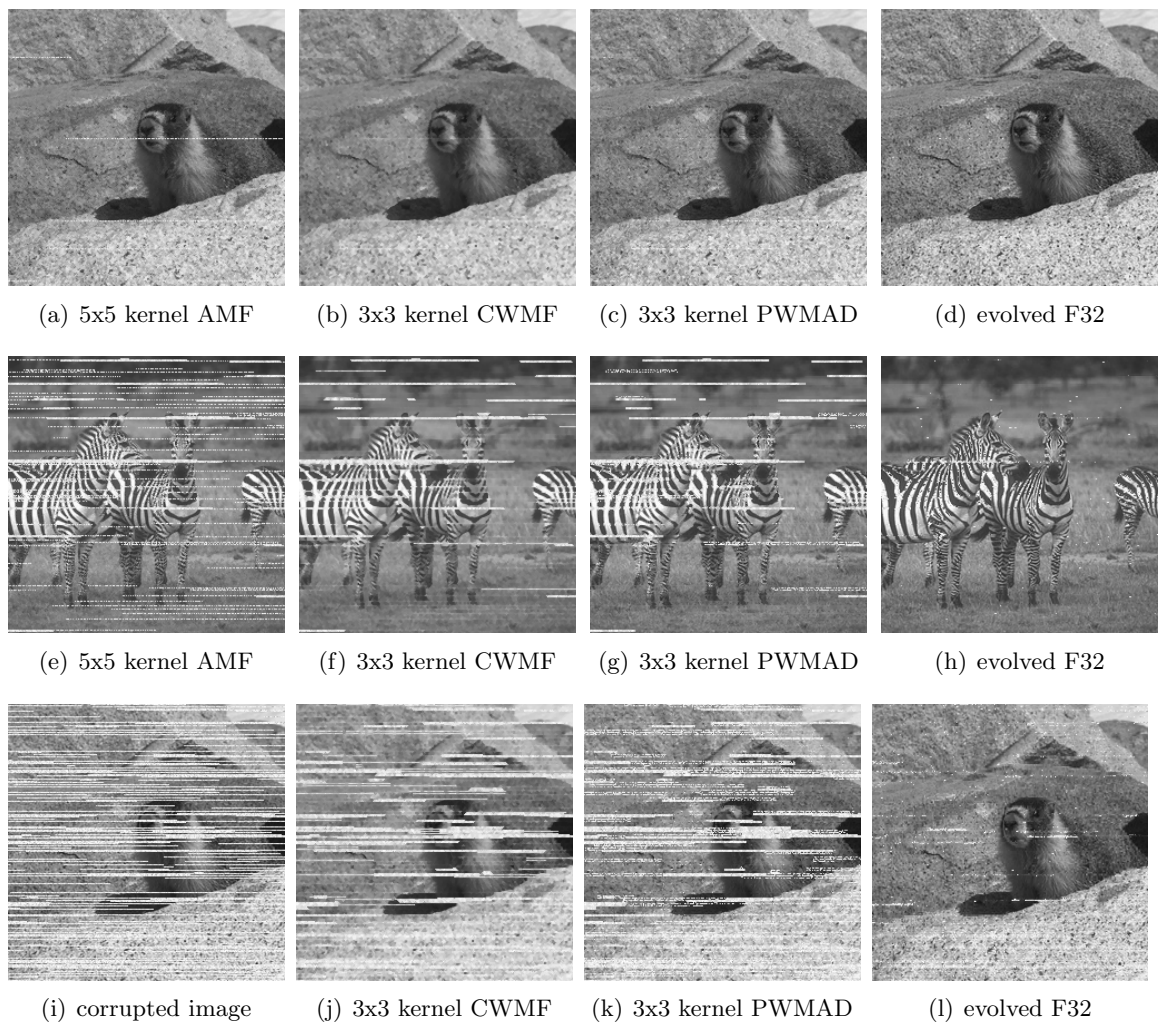
Figure 6.24: Filtering the images corrupted by 1% (a,b,c,d), 15% (e,f,g,h) and 30% (j,k,l) impulse burst noise

Whilst the proposed filter was able to detect and remove a great portion of the noisy pixels, the conventional filters have serious problems and fail to remove the noise. The failure of the conventional filters probably lies in the fact that the bursts are accumulated in the neighboring rows of the image and thus it is difficult to estimate the correct pixel values using median filter.

In comparison with the previous results, the last example depicted in Figure 6.24i-l represents a serious problem for the filters. This data set shows an image containing several sharp and contrast transitions that are very similar to the noise. Even if the proposed filter is able to provide better image (shown in Fig. 6.24d) in comparison with the CWMF and PWMAD filter, it can be seen that the image is degraded slightly. It is interesting to note that the proposed filter is efficient not only from the point of view of the filtering quality but also from the point of hardware/software implementation – it consists of simple operations and does not require iterative processing.

Evolved filters

Figure 6.25, 6.26 and 6.27 show the structure of the best evolved filters for salt-and-pepper
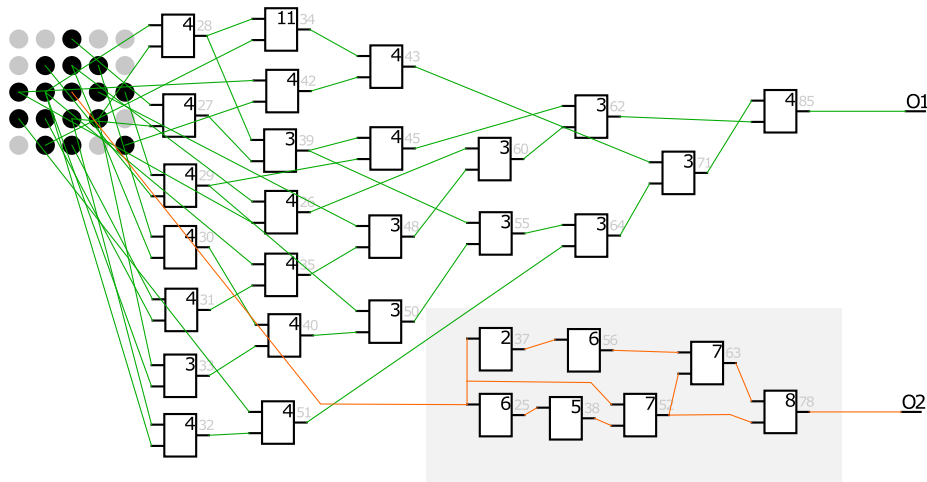shot noise, random-valued impulse noise and impulse burst noise.

Figure 6.25: Structure of the best evolved filters for salt-and-pepper shot noise

In contrast with the other investigated problems, the evolved salt-and-pepper noise
filter, which consists of 31 nodes (operations), employs a relative simple circuitry for noise
detection that is based only on the knowledge of value of the central pixel. This circuitry
consists of 7 nodes.

Let I_c be value of the central pixel and O_2 be the 8-bit output of the detection part.
Then the output O_2 can be expressed as follows.

$$O_2 = ((I_c + I_c/4/2) + (255 - I_c)/4) +^S (I_c + I_c/4/2) = (7I_c/8 + 63) +^S (9I_c/8)$$

If we analyze this equation, we will find out that the O_2 provides the value higher than
127 only in such cases in which I_c holds the following inequality: $33 \leq I_c \leq 227$. The
evolutionary approach discovered that the noisy pixels are affected by the values that are
close to 0 or 255.

On the other hand, the estimation part of this filter is relative complicated and consists
of three operations: minimum, maximum and difference. There are two possible expla-
nations for such a structure. The first theory is that the evolution determined that it is
not necessary to have a robust statistics that detects exactly the noisy pixels because they
can be easily removed using an algorithm that has the capability to suppress the outliers
(e.g. median filter). The second theory is that the detection is used as a pre-filtering that
separates the pixels that are definitely uncorrupted and thus it reduces the space of all
possible input combinations that have to be processed by estimation part.

The evolved random-valued impulse noise filter, which consists of 33 nodes, represents
the opposite example. In this case, the detection part is a relative complicated circuitry uti-
lizing 23 nodes. The calculation is based on the values spread out over four directions from
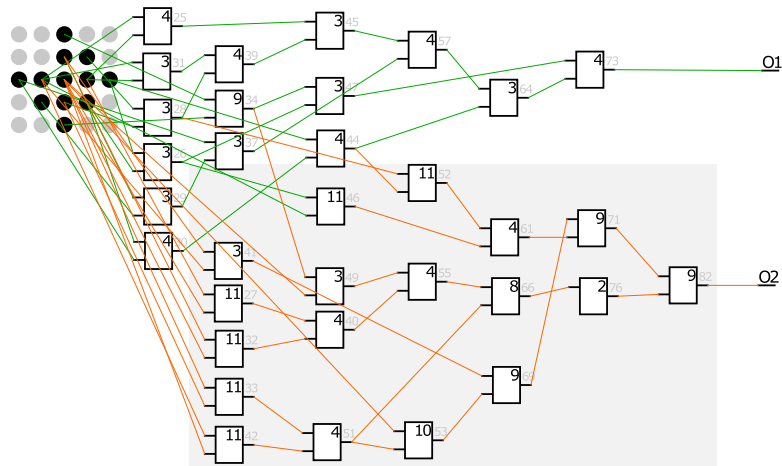
Figure 6.26: Structure of the best evolved filters for random-valued impulse noise

central pixel. The estimation part mainly utilizes the minimum and maximum operations. Interestingly, five nodes are shared between estimation and detection circuitry.
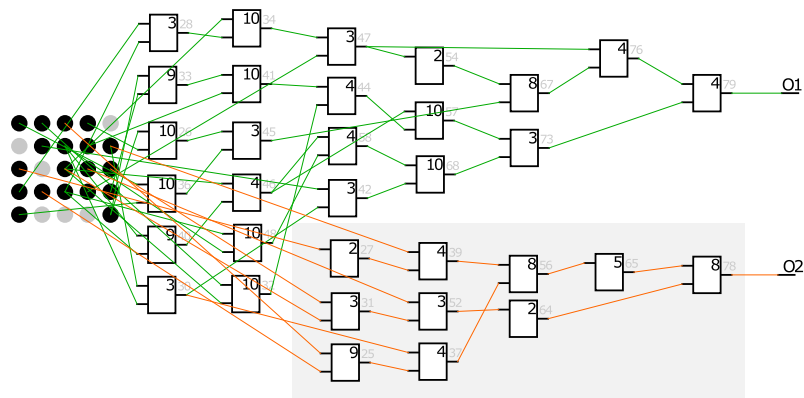

Figure 6.27: Structure of the best evolved filters for impulse burst noise

The impulse burst noise filter shown in Figure 6.27 contains 33 nodes. In this case, the evolutionary approach evolved a relative complicated nonlinear structure that contains eight conditional assignments. The estimation part consists of 10 nodes and utilizes the information from eight pixels of the filter window. The interesting feature is that the estimation part uses (in addition to the value of central pixel) only two pixels that are utilized also by the estimation part.

6.4 Summary

The experimental results clearly show, that the evolutionary design approach can automatically produce image filters that are competitive with conventional filters such as MF, CWMF, AMF, DWMF or PWMAD not only in terms of filtering quality but also if the im-

plementation cost is considered. We observed that images filtered by evolved filters preserve more details (and thus provide a higher visual quality) than images filtered by conventional filters (e.g. median filters).

It has been also shown, that the designed filters require less FPGA resources than conventional filters. For example, the proposed 3-bank filter provides the same filtering capability as a standard adaptive median filter; however, using four times less slices. The more detailed analysis can be found in [214, 220].

The best results have been achieved when the concept of so called switching filter consisting of the detection and estimation part was applied during the evolutionary process. The switching filters evolved for salt-and-pepper shot noise are able to overcome the conventional filters especially for lower noise intensity ($\leq 15\%$). Interesting results have also been achieved if the iterative filtering process has been taken into account. Whilst DWMF and PWMAD require from 5 to 10 iterations, the evolutionary designed random-valued impulse noise filter can produce the images of similar quality using two iterations. Thus, in contrast with the conventional iterative filters, the satisfactory quality is accomplished using less number of resources and operations.

We have also demonstrated that the combination of several evolutionary designed filters leads to the significant improvement of the quality of filtering even if the same filter window is utilized. This concept can be employed in the switching-based filters. We can combine several evolutionary designed detectors as well as estimators together to produce one robust filter of high-quality.

Chapter 7

Hardware Accelerator of Cartesian Genetic Programming

According to John Koza, evolutionary algorithms in general and genetic programming in particular can routinely deliver high-return human-competitive machine intelligence [107]. The competitiveness and performance of the evolutionary approaches have been demonstrated in many tasks and design areas. Unfortunately, the computational power which evolutionary algorithms need for obtaining innovative results is enormous for most applications. This kind of inefficiency is caused by the fact, that the evolutionary algorithms usually spends most of time by running domain-specific simulators which evaluate candidate individuals using large training sets. Even if the principles of evolutionary algorithms are known from nineties, the large computation requirements caused that the EAs became popular in recent decades, when the performance of the personal computers has been dramatically improved.

In order to reduce the computational time of EAs, various methods are usually employed. In general, they can be divided into the following classes: (1) algorithmic – the use of smart search strategies, evolutionary operators and fitness evaluation strategies, (2) source code optimization for a given platform, (3) parallel implementations on clusters of workstations and (4) hardware accelerators. However, even with a parallel implementation, the evolution can be very time consuming.

In contrast with clusters of workstations, the domain-specific hardware accelerators represent a very promising solution due to the high performance, low implementation cost and low power consumption. As the fitness evaluation of a candidate program is the most time consuming part of EA, hardware acceleration are primarily devoted to the fitness calculation. A straightforward implementation involves multiple fitness calculation units which work concurrently. In addition to application-specific chips such as [152], Field Programmable Gate Arrays have been utilized [180, 163, 118, 61, 189]. Modern FPGAs provide a cheap, flexible and powerful platform, often outperforming common workstations or even clusters of workstations in particular applications. For example, Martinek and Sekanina proposed a complete hardware implementation of a simple population-oriented evolutionary algorithm. As it has been demonstrated in [119], a single-chip FPGA-based accelerator running at 50 MHz can provide approx. 20 times higher performance in comparison with a

common workstation running at gigahertz frequency. This speedup has been reported for evolutionary design of image filters. Martin designed linear genetic programming system in an FPGA operating with fixed point expressions encoded as linear programs [118]. For the even 6-parity problem, he achieved the speedup of 18 (two hardware fitness units have been utilized) respectively 419 (for 64 fitness units) in comparison with the PowerPC processor running at 200 MHz. Although the hardware accelerators are able to provide high performance, the key issue usually is whether the particular problem requires the floating-point operations or fixed-point operations. The fixed-point arithmetic circuits or even logic circuits can be accelerated in a much easier way than floating-point operations on a commonly accessible hardware such as FPGA.

Recently, Graphics Processing Units (GPUs) that are available in common desktop computers have been used to parallelize the fitness evaluation. Chitty [31] reports the speedup of 0,4–30 depending on target problem. Harding and Banzhaf have shown how the speedup of candidate individual evaluation depends on the expression length for various problems [71]. With the growing expression length and growing number of test cases, GPU becomes more effective than CPU. The maximum speedup is approx. 1000 for Boolean expressions and 14 for a protein classification problem. These results show only the number of times faster evaluating evolved GP expressions is on the GPU (NVidia GeForce 7300 GO) compared to CPU implementation (Intel Centrino T2400 running at 1.83 GHz); the speedup of evolution was not reported. Unfortunately, for training sets of a common size, the overhead of transferring data to the GPU and for constructing the GPU programs leads to a worse performance than CPU.

This chapter is focused on the implementation of a modular FPGA-based accelerator designed to accelerate the Cartesian Genetic Programming. The first two sections are devoted to the description and evaluation of the proposed hardware accelerator with one fitness unit. The second two sections provide the details of the enhanced version supporting of multiple fitness units and its evaluation.

7.1 Target FPGA Platform

In order to implement the proposed system, a COMBO6X card equipped with Virtex II Pro XC2VP50 FPGA has been used [27]. This platform has been developed to accelerate time-critical and high-speed applications especially from the area of network applications. The whole system contains two FPGAs. The smaller FPGA XC2VP4 of Virtex II family serves as a PCI interface while the larger FPGA of Virtex II Pro family which contains 23 616 slices, 49 788 flip flops, 852 IO blocks and 232 Block RAM modules (2kB each) is intended for implementation of application-specific hardware accelerators. The FPGA can utilize three types of memories: ternary CAM memory having 2 Mb of total capacity; three synchronous SRAMs, 2MB of each, organized as $512k \times 36$ and DDR DRAM memory with the capacity up to 2GB.

The Virtex II Pro FPGA contains two instances of IBM PowerPC 405 core which is able to operate at 400 MHz each. As shown in Figure 7.1, the PowerPC is equipped with a 5-stage pipeline, a virtual-memory-management unit, separate instruction-cache and data-

cache units, 3 programmable timers, on-chip memory controller (OCM) and variety of interfaces, including processor local bus (PLB) interface, device control register (DCR) interface and JTAG port interface.
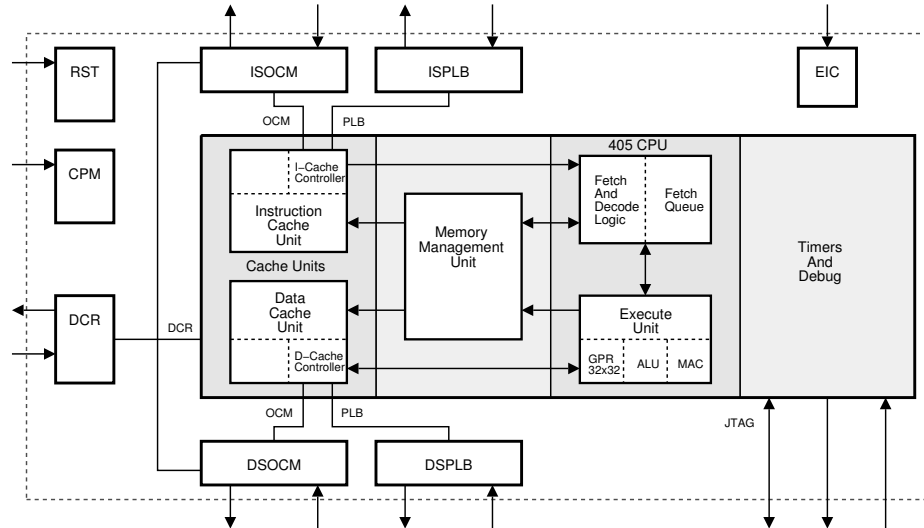

Figure 7.1: Architecture and interface of the PowerPC 405 processor (adopted from [193])

Table 7.1 summarizes basic parameters of the PowerPC 405 interfaces. Although the PLB controller is more complicated than OCM controller, it provides a higher throughput. Further details of the PowerPC 405-processor architecture are available in [193].

The FPGA chip can be configured either externally or internally, using the so-called Internal Configuration Access Port (ICAP). Although the port can operate at 66 MHz, it is not used for evolutionary filter design due to the low throughput, insufficient for our target evolvable hardware applications [59, 119].

Table 7.1: A comparison of basic interface parameters of PowerPC 405. IS stands for Instruction Side, DS stands for Data Side. The C405 column summarizes the performance of the whole PowerPC core.

Interface	DCR	ISOCM	DSOCM	ISPLB	DSPLB	C405
Maximal throughput [MB/s]	300	1 200	600	2 400 [1]	2 400 [1]	1 200
Data bus width [b]	32	64	32	64	64	32
Address space [B]	1 k	16 M	16 M	4 G	4 G	4 G
Variable latency support	Yes	No	No	Yes	Yes	n/a

[1] The maximal throughput is higher that the throughput of PowerPC core C405 due to the presence of PLB fill buffer.

7.2 CGP Accelerator with a Single Fitness Unit

The basic idea of the CGP accelerator is that a given instance of CGP (i.e. a reconfigurable array consisting of $u \times v$ programmable nodes) is implemented as a reconfigurable circuit on the FPGA. Its configuration is defined using a bitstream which is stored in a configuration

register implemented also in the FPGA. This concept is called the virtual reconfigurable circuit [158].

7.2.1 Architecture Overview

The proposed CGP accelerator is completely implemented in a single FPGA and consists of Genetic unit (GU), Fitness Unit (FU) and Control Unit (CU) (see Figure 7.2). Training data are stored in external SRAM memories. The GU as well as FU are connected to the internal FPGA bus which provides an effective communication interface between FPGA and PCI bus. The host PC is used to load training data, read the results, and define the parameters of CGP. The FU contains one or more instances of Virtual Reconfigurable Circuit (VRC). The VRC is, in fact, a second reconfiguration layer developed on the top of an FPGA in order to obtain a fast reconfiguration and application-specific programmable elements.
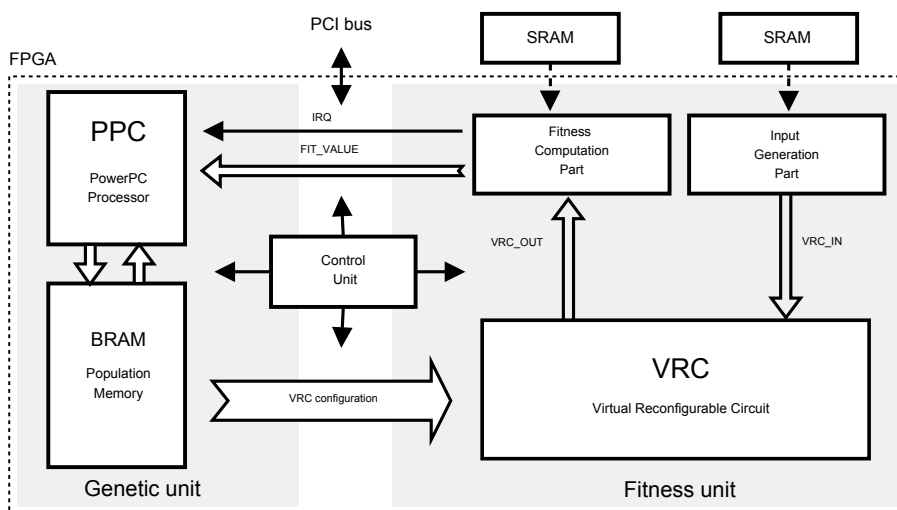

Figure 7.2: Architecture of the proposed CGP accelerator

In order to maximize the overall performance, the CU plays the role of master, controls the entire system and provides an interface to the host PC. In particular, it starts/stops the evolution, handles the generation counter and issues the control signals for the remaining units. The CU consists of two subcomponents working concurrently. The first subcomponent reconfigures the VRCs according to the configuration stored in the population memory. The second subcomponent is responsible for sending the fitness value to the PowerPC processor. The PowerPC generates a new candidate individual when a request is issued. The instruction memory of the PowerPC is implemented using on-chip synchronous Block RAM (BRAM) memories. The memory is connected to the PCI bus interface in order to upload the PowerPC programs from PC. Since our search algorithm is optimized for space, it is completely executed from an instruction cache.

The population of candidate configurations is also stored in on-chip BRAM memories. The population memory is divided into N_b banks; each of them contains one configuration

bitstream. Each bitstream consists of the configuration data that are necessary to configure one VRC. An additional bit (associated with every bank) determines the data validity; only valid configurations can be evaluated. In order to overlap the evaluation of a candidate configuration with generating a new candidate configuration, at least two memory banks have to be utilized. While a candidate solution is evaluated, the new candidate configuration is generated. The population memory provides two independent ports:

1. the 32-bit read/write port A connected to the PowerPC processor and

2. the m-bit read-only port B connected to the fitness unit used for the reconfiguration of VRC.

Note that the width of the B port must be chosen with respect to the implementation limits (m must be an integer divisible by 128) and the number of bits of a part of bitstream used to configure one column of VRC.

The process of evaluation works as follows:

1. When a valid configuration is available, the CU initiates the reconfiguration of VRC.

2. As soon as the first column of configurable logic blocks (CFBs) has been reconfigured, CU initiates the fitness calculation process performed by the FU.

3. When the last column of CFBs has been reconfigured, a corresponding memory bank is invalidated and the bank counter is incremented.

4. Three clock cycles before the end of evaluation the FU indicates the forthcoming end of evaluation.

5. The CU initiates a new configuration of VRC and repeats the sequence 1-4 again.

6. As soon as the fitness value is valid, an interrupt request (IRQ) is generated to activate a service routine of the PowerPC. In this routine, PowerPC reads the fitness value together with some additional data (corresponding bank number) and new candidate configurations are generated for the given bank. The PowerPC processor acknowledges the interrupt and sets up the validity bit.

7.2.2 Genetic Unit

Due to the presence of the PowerPC processor, the proposed system allows the use of various search algorithms. These algorithms utilize a population of candidate solutions and a single genetic operator — mutation, which inverts k bits of the chromosome (i.e. of the configuration). No crossover operator is used. An analysis of various search strategies, mutation operators and pseudorandom number generators is presented in the Section 7.3.2. Hill climbing, genetic algorithm and parallel version of random search strategy was implemented. In order to exploit the performance of the proposed platform, the search strategy has to generate a new candidate solution as soon as a candidate solution is evaluated. This concept differs from standard implementations on common CPUs where a new population is usually generated the moment the whole population is evaluated.

7.2.3 Fitness Unit

The fitness calculation is carried out by the Fitness Unit. The fitness unit consists of three components: (1) the training data generation part, (2) the fitness computation part and (3) Virtual Reconfigurable Circuit. According to the size of training data, the training data can be stored using internal BRAM memory or external SRAM memory. Note that in case of evolutionary design of image filters, all image data are stored in external SRAM memories due to the limited capacity of internal BRAMs available in the FPGA chip. The first part of the fitness unit loads the training data and forwards them to the inputs of VRC. VRC is utilized to evaluate the response for the training vectors. The response of a candidate circuit is sent back to the Fitness Unit, where it is compared with the required response which is stored in another internal/external memory. The implementation of the circuit responsible for the computation of fitness value depends on the problem to be solved.
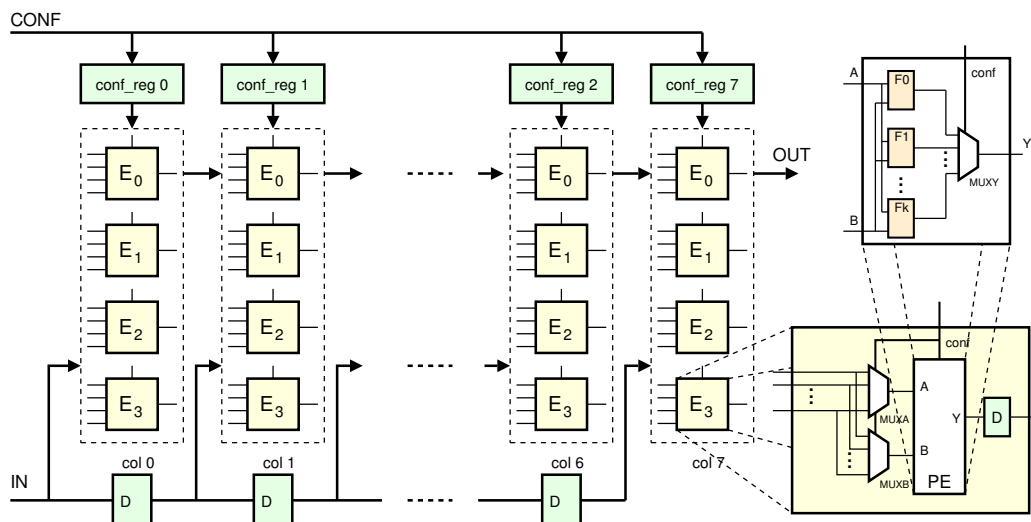

Figure 7.3: VRC for symbolic regression problems

7.2.4 VRC for Symbolic Regression Problems

Figure 7.3 shows the VRC implemented for the image filter design problem, which is a kind of a symbolic regression problem over the FX representation [206]. Every candidate program (image filter) is considered as a digital circuit of nine 8-bit inputs and a single 8-bit output.

The VRC consists of 2-input Configurable Logic Blocks (CFBs), denoted as E_i, placed in a grid of 8 columns and 4 rows. Any input of each CFB may be connected either to a primary circuit input or to the output of a CFB, which is placed anywhere in the preceding column. Any CFB can be programmed to implement one of 16 function from Γ, where Γ includes addition, subtraction, shift, minimum, maximum and logic functions. All these functions operate with 8-bit operands and produce 8-bit results. The reconfiguration is performed column by column. The computation is pipelined; a column of CFBs represents a stage of the pipeline. Registers (denoted D) are inserted between the columns in order

to synchronize the input pixels with CFB outputs. The configuration bitstream of VRC, which is stored in a register array *conf_reg*, consists of 384 bits. A single CFB is configured by 12 bits, 4 bits are used to select the connection of a single input, 4 bits are used to select one of the 16 functions. Evolutionary algorithm directly operates with configurations of the VRC; simply, a configuration is considered as a chromosome.

In tasks of symbolic regression, training data are stored in external SRAM memories. Fitness unit loads training data from external SRAM1 memory and forwards them to the inputs of VRC. The outputs of VRC, y_i, are compared with required outputs, r_i, (which are loaded from another external memory, SRAM2) and simultaneously stored into the third external memory, SRAM3. The FU can be considered as an extension of the VRC pipeline because in each clock cycle, a temporary fitness value is updated by a new difference, $|y_i - r_i|$. Due to pipelined reconfiguration as well as execution of VRC, the evaluation of a candidate program (circuit) requires k clock cycles, where k is the number of training vectors.
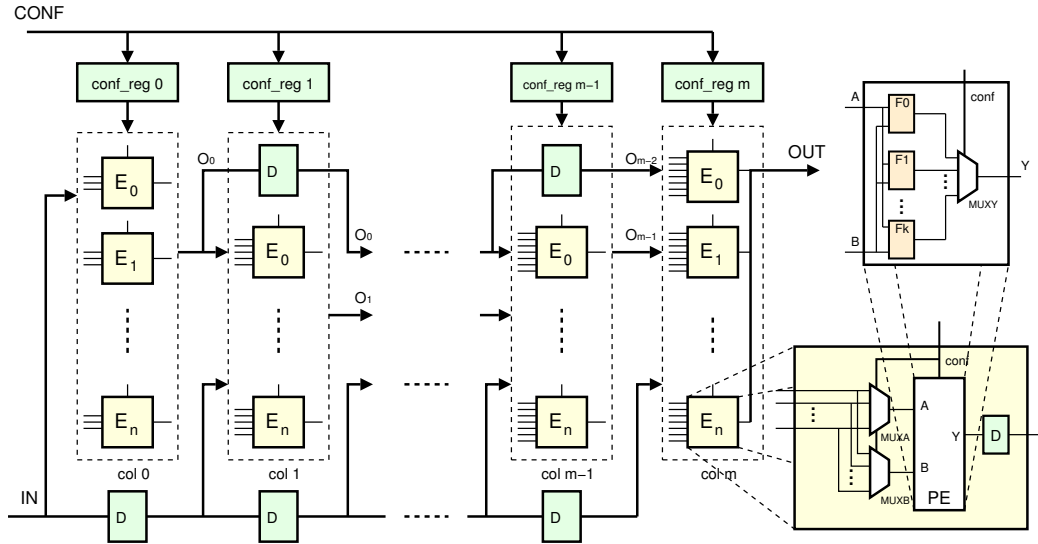

Figure 7.4: VRC for evolution of digital circuits

7.2.5 VRC for Logic Expressions

The architecture of VRC for the evolutionary design of logic circuits is similar to the VRC for symbolic regression. There are four main differences: PEs contain only logic functions, *L*-back=2 is supported, the size of phenotype can be calculated and a data parallel operation of PEs (the same as used in the software parallel simulation) is introduced (see Section 3.3.2). If PEs operate at dw bits then the speedup against the bit-level execution is dw-times. In order to support *L*-back=2, additional registers (D) have been used to store the results of stage $i - 2$ for stage i of the pipeline (see Figure 7.4). The number of configuration bits for a single column is $2 * \log_2(n_i + 2u) + \log_2(n_f)$. In contrast to symbolic regression, the training data (truth table) is stored in BRAMs. For example, if $n_i = 16$ then 64 BRAMs are utilized. All possible input combinations are generated using a binary counter and need
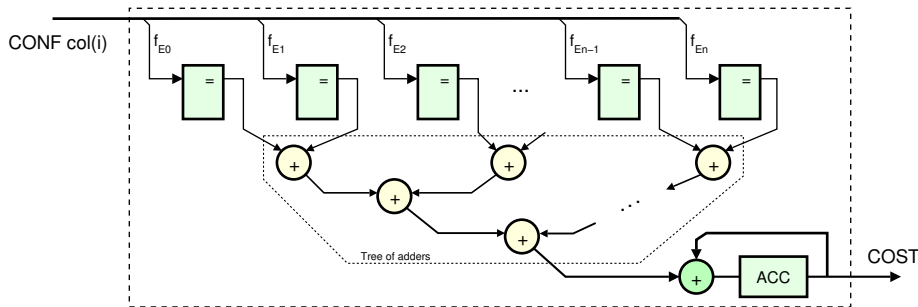
Figure 7.5: Circuit for the calculation of the size of a phenotype

not to be stored in BRAMs. When the size of circuit is not optimized, the maximum fitness value is $2^{n_i} n_o$.

Figure 7.5 explains the calculation of the size of a candidate circuit. The method assumes that a PE can implement a single wire. Once a functionally-perfect solution is found, the size is optimized. The objective is to maximize the number of PEs which operate as wires. The configuration of a single column of VRC is analyzed using comparators. The comparator returns 1 in case that a particular PE operates as a wire. These 1s are added using a tree of adders. This calculation is performed when the column of PEs is configured. It costs no extra time. The size of phenotype is stored to the least significant bits of the fitness value.

7.3 Experimental Evaluation

7.3.1 Theoretical Performance

Since the steps of the evaluation process described in Section 7.2.1 are pipelined in such manner that there are no idle clock cycles, time of evolution t_{total} can be expressed as

$$t_{total} = t_{init} + N_e t_{eval} = t_{init} + N_e N_t \frac{1}{f} \tag{7.1}$$

where t_{init} corresponds with the time needed for the initialization, N_e is the number of evaluations, N_t is the number of training vectors and f is the operation frequency (in our case, $f = 100$ MHz).

7.3.2 Evolution of Image Filters

Results of synthesis for VRC consisting of 4×8 CFBs are summarized in Table 7.2. While the PowerPC works at 300 MHz, the logic supporting the PowerPC works at 150 MHz. The remaining FPGA logic (including VRC and FU) works at 100 MHz. Experimental results show that approximately 6,000 candidate programs can be evaluated per second when the training set consists of 15876 vectors (training image consisting of 128×128 pixels is considered) which is 44 times faster than the same algorithm running at the Celeron 2.4 GHz [206]. This accelerator was utilized to discover novel implementations of image filters discussed in Section 6.3.1.

Table 7.2: Results of synthesis for the symbolic regression problems

VRC	IO blocks	BRAM	Slices	DFF
Available	852	232	23 616	49 788
4×8 CFBs	602	12	4 591	3 638
used	70%	5%	20%	7%

Experiments were arranged to find a suitable mutation rate, an efficient pseudo-random number generator and search strategy. The objective was to (1) remove the salt-and-pepper noise with intensity of 5%, 10% and 20% from real-world images and (2) design an edge detector which is able to deal with input images corrupted by the salt-and-pepper noise. A visual quality of filtered images is expressed using mean difference per pixel (mdpp) between the filtered image and original image.

Search strategy

As the search algorithm is stored in the program memory of the PowerPC processor, the proposed platform allows the designer to easily modify the search algorithm. Three search algorithms are evaluated: a random search, a hill-climbing algorithm and a genetic algorithm. As training images we have used 128×128-pixel version of Lena image (see Figure 6.12). The parameters for each experiment such as maximum number of evaluations, mutation rate and number of evolutionary runs are given in Table 6.4.

Random search (RS)

This algorithm operates with p individuals that are generated randomly at the beginning of the evolution. Then an offspring is created using a bit-mutation operator from each parent and evaluated. If the offspring is equal or better than its parent then the offspring replaces the parent in the new population. In fact, p standard random search algorithms run in parallel. This algorithm was implemented in [119] as a special circuit. Figure 7.6 shows concurrent operations of several processes running in hardware and the PowerPC processor (including the configuration of the VRC, evaluation of candidate filters and generation of candidate configurations). These processes are synchronized in such a way that no clock cycle is lost because of waiting on some resources. Note that only two banks are considered in this example. Parameter p was chosen as $p = 8$.
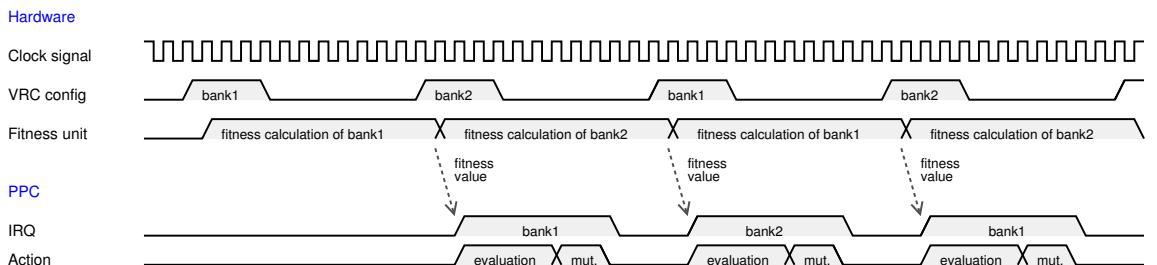

Figure 7.6: Example of timing for 2 banks: the reconfiguration of VRC costs 4 clock cycles, the evaluation costs 12 clock cycles and the interrupt routine requires 8 clock cycles.

Hill Climbing search (HC)

This algorithm operates with p individuals that are generated randomly at the beginning of the evolution. After their evaluation, r offspring configurations are generated for each parent using a bit-mutation operator. The best offspring of the r offspring configurations replaces the corresponding parent; however, only in case that its fitness value is equal or better than the parent's fitness value. Again, in fact, p standard hill climbing algorithms run in parallel. The following setup was applied in this experiment: $p = 8, r = 2$.

Genetic algorithm (GA)

The initial population of p individuals is generated randomly. Then, r offspring are generated from each parent using a bit-mutation operator. A new population consisting of p individuals is formed from p parents and their $p.r$ offspring. We used a deterministic selection in which p-best scored individuals are selected as new parents. The following setup was applied in this experiment: $p = 8, r = 2$.

Table 7.3: The experimental evaluation of the search strategies on four test problems

noise	search	mean difference per pixel			
type	algorithm	min	max	mean	std.dev.
5% salt-and-pepper	RS	0.410	3.190	0.967	0.581
noise	HC	0.432	3.320	1.060	0.615
	GA	0.333	3.450	2.010	1.240
10% salt-and-pepper	RS	0.982	3.280	1.720	0.337
noise	HC	0.913	48.01	4.370	3.730
	GA	0.828	7.390	2.650	2.190
20% salt-and-pepper	RS	1.870	4.350	2.850	0.510
noise	HC	1.650	4.190	2.880	0.587
	GA	0.870	12.10	2.680	1.330
5% noise,	RS	1.100	2.660	1.910	0.419
edge detection	HC	1.380	2.960	2.310	0.421
	GA	1.070	2.660	2.400	0.453

The results are summarized in Table 7.3. We can observe that while the best mdpp (in average) is always obtained by means of the RS algorithm, the GA always produces the filters with the smallest mdpp at all. Recall that the number of evaluations is identical; however, RS always produces more generations than the GA. The fitness value of the best-evolved filters are shown in Table 6.5. The LFSR generator has been used in this experiment.

Mutation operator implementation and mutation rate

Our strategy is to estimate the suitable mutation rate using not so many evaluations (less than 100,000 evaluations allowed) and then to utilize the discovered mutation rate in long-time experiments. Figure 7.7 and Figure 7.8 show boxplot diagram containing the average mdpp, median and standard deviance calculated from the best values obtained at the end of 32 independent runs of the RS algorithm ($p = 8$) for each of k inverted bits in the
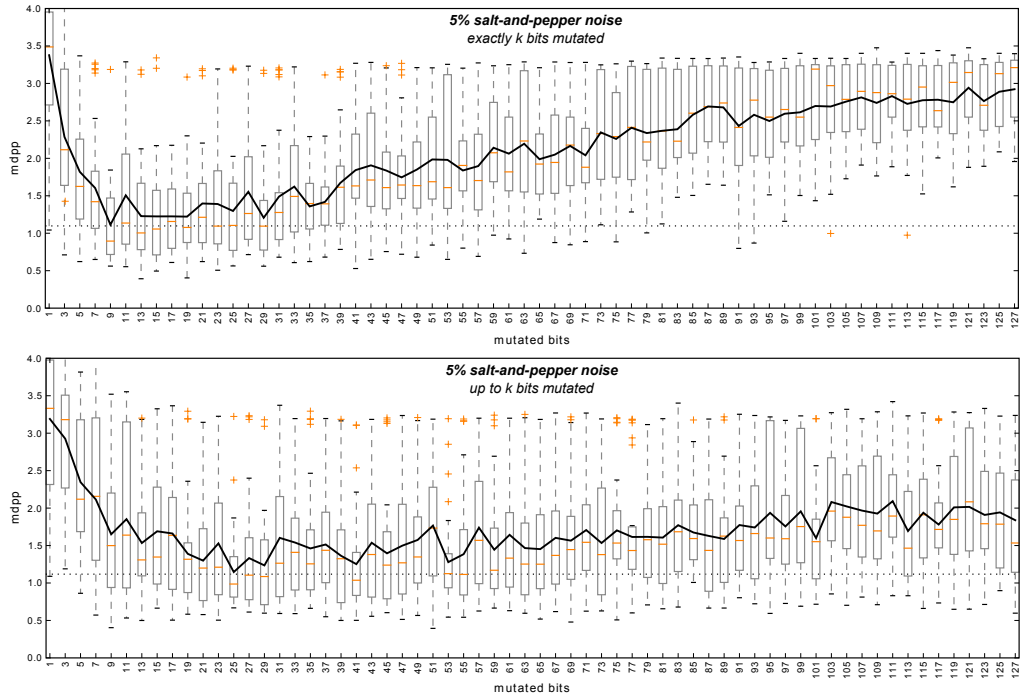
Figure 7.7: The results of the evolutionary design of 5% salt-and-pepper noise for various mutation rate calculated from 32 independent runs.

chromosome ($k = 1 - 127$). Two methods are used: exactly k bits are always inverted and a randomly chosen number of bits is inverted; however, limited by k.

We can observe that the mutation rate which allows minimizing the fitness value fluctuates around 20 mutated bits per chromosome. This value corresponds with mutation ratio of 5.2%. It is also more efficient to invert exactly 20 bits than to randomly generate a number from interval $1 - 20$.

Pseudorandom number generator

As the outputs of pseudorandom number generators (PRNG) only approximate some of the properties of random numbers, we have to determine a suitable one for the proposed architecture. The following three PRNGs were evaluated:

Linear congruential generators (CG)
Linear congruential generators represent the oldest and best-known pseudorandom number generator algorithms. It is, however, well known that the properties of this class of generators are far from ideal. The applied linear congruential generator operates according to formula

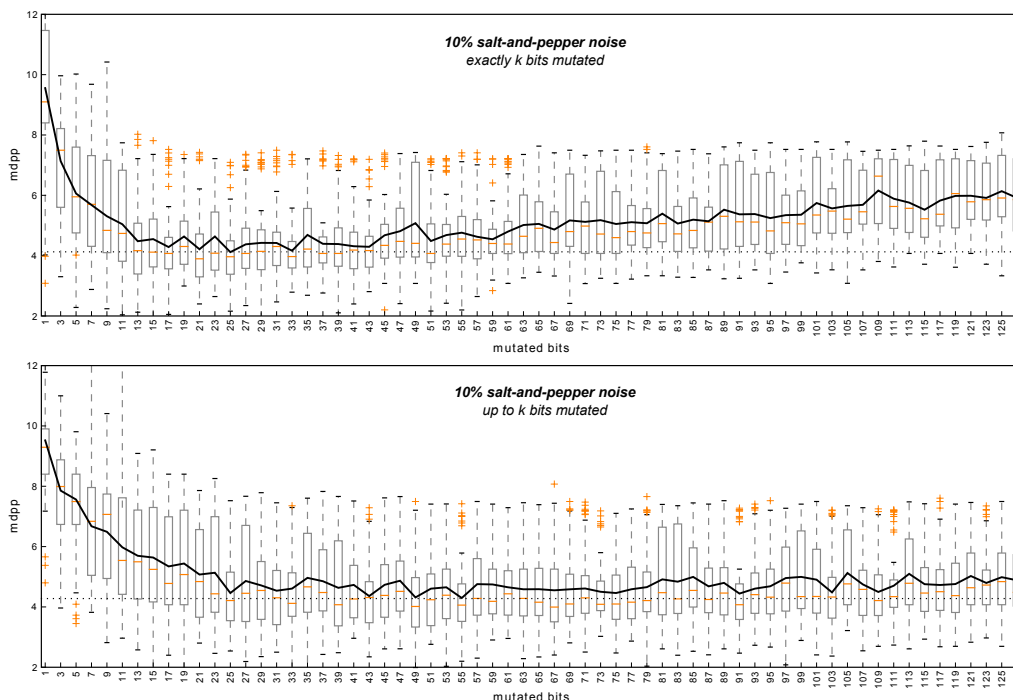$$V_{j+1} = (1103515245 \times V_j + 12345) \bmod 2^{32} \tag{7.2}$$

Figure 7.8: The results of the evolutionary design of 10% salt-and-pepper noise for various mutation rate calculated from 32 independent runs.

Linear Feedback Shift Register (LFSR)

Linear Feedback Shift Register is a shift register whose input bit is driven by the exclusive-or (xor) of some bits of the overall shift register value. As for this PRNG is also known that output bits do not pose a good distribution we used a parallel LFSR consisting of 16 independent and different LFSRs seeded identically. The LFSR generators operate according to formula

$$V_{j+1}^i = \begin{cases} V_j^i \text{ shr } 1 & \text{if LSB bit of } V_j^i = 1 \\ V_j^i \text{ shr } 1 \oplus C^i & \text{otherwise} \end{cases} \tag{7.3}$$

where C^i is a suitable constant for i-th LFSR generator (e.g. 0x805FDF47).

Mersenne Twister (MT)

Mersenne Twister algorithm is a twisted generalized feedback shift register that avoids many of the problems with earlier generators. It has the period of $2^{19937} - 1$ iterations, is proven to be equidistributed in (up to) 623 dimensions (for 32-bit values). A standard implementation of Mersenne Twister was utilized [120].

Figure 7.9 shows average mdpp and corresponding standard deviations obtained from 32 independent runs (after 12,288 evaluations in each run) using the RS algorithm ($p = 8$, mutation applied on 20 bits). The three generators are compared on two problems: removing 10% salt-and-pepper noise from Lena image and edge detector design. Surprisingly, there are not any significant differences in the quality of obtained results.
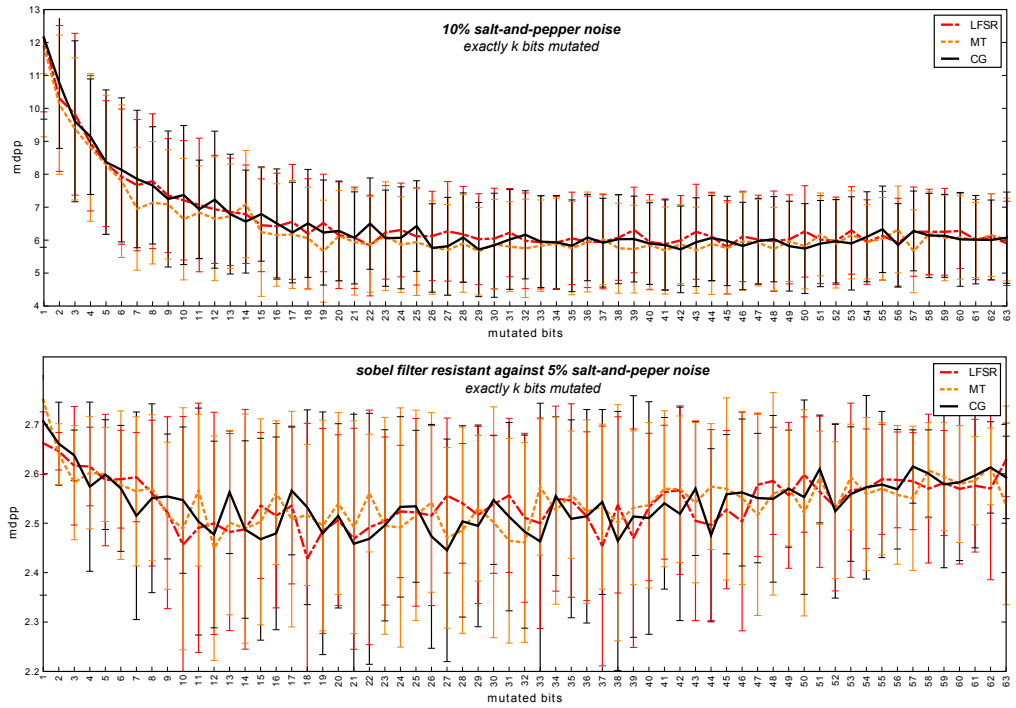
Figure 7.9: The impact of the quality of pseudo random numbers generators on the results of the evolutionary design of 10% salt-and-pepper noise filter and noise-resistant edge detector. The average fitness value expressed in terms of mdpp is calculated from 32 independent runs. The error bars shows the standard deviance.

7.3.3 Evolution of Digital Circuits

Table 7.4 provides results of synthesis for various parameters of VRC. While the size of VRC and the number of inputs and outputs are fixed, the number of test vectors evaluated in parallel (i.e. dw) increases from 1 to 12. When no data parallel execution is used, the whole design occupies approx. 10% resources; when $dw = 12$ (i.e. 12 test vectors are evaluated in parallel by a PE) the design occupies approx. 90% resources. Using this setup we can achieve 27 times faster evaluation in comparison with a highly optimized SW

Table 7.4: Results of synthesis for VRC with 10x10 PEs, 9 inputs, 9 outputs and 4 logic functions per PE (XC2VP50-ff1517 Xilinx FPGA). DFF is the number of flip-flops and FG is the number of function generators

		# of vectors evaluated in parallel (dw)				
resource	available	1	2	4	8	12
BRAMs	232	14	16	20	28	36
	used	6.0%	6.9%	8.6%	12.1%	15.5%
DFFs	49788	2743	2993	3533	4709	5843
	used	5.5%	6.0%	7.1%	9.5%	11.7%
FGs	47232	4836	7813	14164	26734	41281
	used	10.2%	16.5%	30.0%	56.6%	87.4%

Table 7.5: Results of synthesis for various VRCs of 9 inputs, 9 outputs, 4 logic functions and $dw = 2$ (FPGA XC2VP50-ff1517)

		VRC size			
resource	**available**	10×10	12×12	14×14	16×16
DFFs	49788	1644	2336	3634	4664
	used	3.3%	4.7%	7.3%	9.4%
FGs	47232	6242	9012	26700	32352
	used	13.2%	19.1%	56.5%	68.5%
# of conf. bits		1200	2016	2744	3584

implementation running at a CPU Intel Xeon 3 GHz processor (and utilizing a parallel simulation at 32 bits), even if the VRC works at 100 MHz.

Table 7.5 contains the results of synthesis for various VRC sizes. The number of inputs, outputs, logic functions and data width are fixed. The last row shows the number of configuration bits of VRC.

In order to investigate the impact of the L-back parameter, we created two VRCs with $L = 1$ and $L = 2$. Proposed implementations were evaluated in the task of multiplier evolution, a traditional hard benchmark problem for evolutionary circuit design. A parallel version of Hill Climbing algorithm with neighbourhood of two and population size of 8 individuals was used. Table 7.6 summarizes results of 10 independent experiments for each problem. We can see that the increasing value of L-back parameter has the positive effect on the average number of generations and the success rate. Obtained results are comparable to the best-known results [182] (where the authors allowed the maximum value of L-back parameter).

Table 7.7 compares the number of evaluated candidate circuits per one second in a highly optimized SW implementation and proposed HW accelerator. In case of the SW implementation, the time of circuit evaluation depends on the size of the phenotype and the number of training vectors. On the other hand, in hardware, this time depends only

Table 7.6: Results for evolution of multipliers ($\Gamma = \{$BUF, AND, XOR, $\bar{a}$ AND $b\}$)

Parameters of evolution										
multiplier		**2 × 2**		**2 × 3**		**3 × 3**		**3 × 4**		**4 × 4**
l-back	1	2	1	2	1	2	1	2	1	2
VRC	8x8	8x8	10x10	10x10	10x10	10x10	10x10	10x10	16x16	16x16
inputs	4	4	5	5	6	6	7	7	8	8
gener. (max)	10k	10k	100k	100k	1M	1M	10M	10M	20M	20M

Results										
success rate	91%	96%	92%	100%	72%	96%	18%	84%	0%	4%
gates (min)	7	7	13	13	29	24	60	45	-	125
gates (max)	19	13	20	21	45	47	67	68	-	156
gates (avg)	9	8	15	15	34	33	61	57	-	138
gener. (avg)	1.8k	1.5k	20k	13k	22k	284k	4.84M	3.84M	-	14.2M

on the number of training vectors. Hence, the accelerator becomes more useful for complex VRCs and larger sets of training data.

Table 7.7: The number of evaluations per second. VRC operates at 100 MHz ($dw = 4$), SW is executed on the Intel(R) Xeon(TM) CPU 3.06 GHz ($dw = 32$)

#	VRC size (SW)			VRC size (HW)			evaluation
inputs	10×10	12×12	16×16	10×10	12×12	16×16	speedup
6	400	296	222	6250	6250	6250	15–28
7	250	173	89	3125	3125	3125	12–35
8	154	95	51	1563	1563	1563	10–30
9	85	50	25	781	781	781	9–31

7.4 CGP Accelerator with Multiple Fitness Units

As it has been demonstrated, the evolution using the proposed accelerator containing a single fitness unit running at 100 MHz is significantly faster that the software implementation running on the common PC at GHz processor. In the task of image filter evolution, which can be considered as a symbolic regression problem, the FPGA-based accelerator exhibits the 44 times higher performance. Looking at the results of synthesis, the FPGA offers the capacity to increase this speedup by creating a system with multiple fitness units. The architecture of the accelerator with multiple fitness units is shown in Figure 7.10. Similarly to the previous architecture, the system consists of genetic unit, fitness unit and control unit. All the units have the same meaning as it has been described in the previous section.
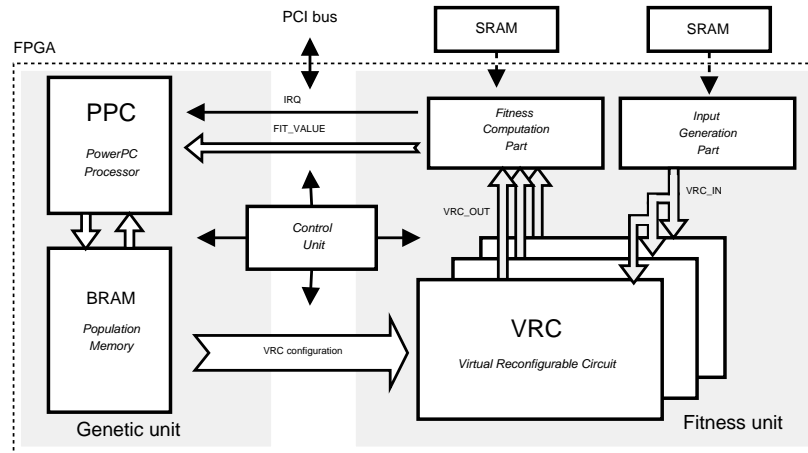

Figure 7.10: Architecture of the proposed CGP accelerator with multiple fitness units

The population of candidate configurations is stored in on-chip BRAM memories. The population memory is divided into N_b banks; each of them contains N_c configuration bitstreams. Each bitstream consists of the configuration data that are necessary to configure one VRC. All the bitstreams stored within a bank are evaluated in parallel. An additional bit (associated with every bank) determines the data validity; only valid configurations

can be evaluated. While the candidate solutions are evaluated, the N_c new candidate configurations are generated. Similarly to the previous architecture, the population memory provides also two independent ports; the m-bit read-only port B is connected to the fitness unit and used for the reconfiguration of all VRCs. Since corresponding columns of VRCs are reconfigured at the same time (i.e. in parallel), the part of bitstream which encodes one column of VRC can contain up-to m/N_c bits. Note that the width of the B port must be chosen with respect to

1. the implementation limits (m must be an integer divisible by 128),

2. the number of bits of a part of bitstream used to configure one column of VRC and

3. the number of VRC instances N_c.

7.4.1 Fitness Unit

The fitness unit consists of N_c instances of VRC and two subcomponents: (a) the input generation part and (b) the fitness computation part. The training data are stored in external SRAM memories. The fitness unit loads training data from the external SRAM1 memory and forwards them to the inputs of VRCs. The VRCs exhibit the same architecture as it has been described in the previous section.

In case of the evolutionary design of image filters it is necessary to implement a local neighborhood function (also referred to as a sliding window function) producing wk^2 bits per one clock cycle that have to be forwarded to the inputs of VRCs, where k is the size of the filter window and w is the data width (in our case $k = 3$ and $w = 8$). The local neighborhood function can be efficiently implemented using k row buffers as shown in Figure 7.11.

In case of common one-dimensional symbolic regression problems, the training data can be forwarded directly from the SRAM1 to the VRC inputs. In case that the problem to be solved involves the utilization of a history of previous samples, the input generation part of the fitness unit will contain a buffer for previous samples. This buffer can be implemented using registers or BRAM memories.
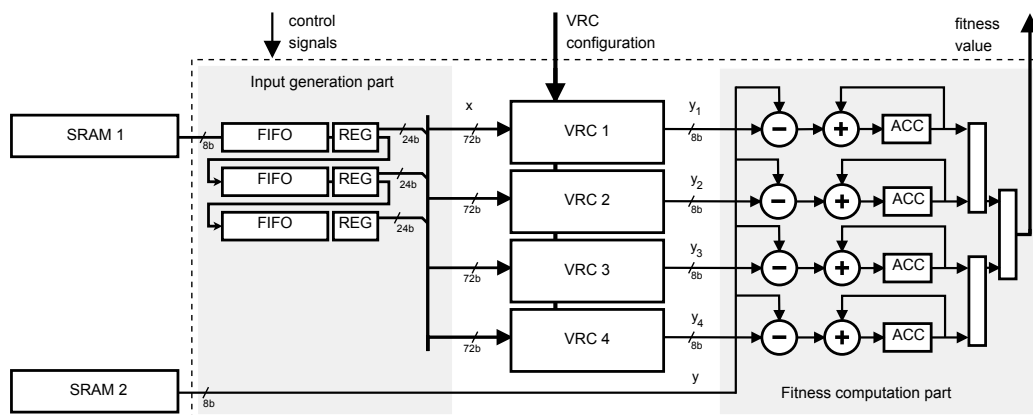

Figure 7.11: Architecture of the fitness unit with multiple VRCs ($N_c = 4$)

The fitness computation part consists of N_c instances of a circuit that computes the fitness value; each VRC utilizes its own instance. In this experiment, four VRCs with k^2 inputs and one output are used. For each VRC i, the absolute difference between the output value y_i and the required output value y (which is obtained from the external memory SRAM2) is calculated. Then, a temporary fitness value stored in accumulator (ACC_i) is updated by the difference $|y_i - y|$. As soon as FU evaluates the last training vector, the best fitness value together with the index of corresponding VRC is sent to the PowerPC. VRCs are then reconfigured using new bitstreams.

7.4.2 Genetic Unit

The introduction of multiple VRC instances requires designing of a problem specific memory interface that allows avoiding the idle clock cycles. The memory banks are used in order to overlap the evaluation of the candidate solutions with the generation of new chromosomes. Moreover, each bank is divided into N_c equivalent sections, each of them is used to configure a single VRC. The population memory consists of several instances of BRAM memories arranged together to provide the required number of bits. This arrangement enables to reconfigure all VRC instances in parallel. In order to reduce the number of memory accesses issued by the PowerPC processor, the population memory is equipped with a logic that enables to store only the differences between the configurations of neighboring sections.
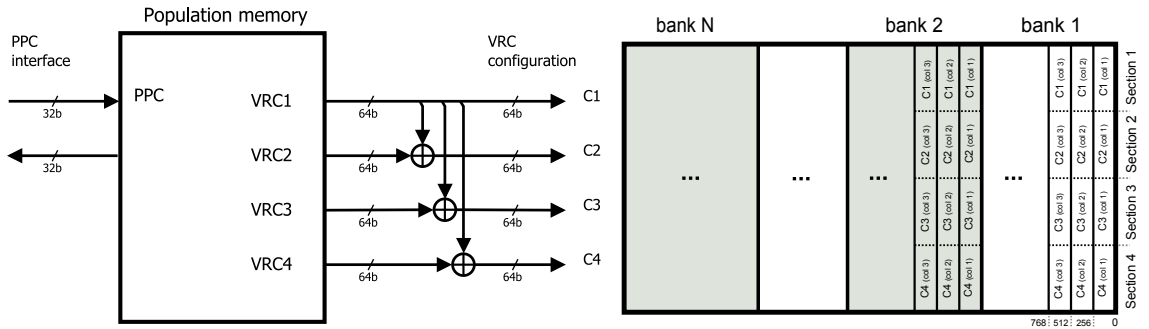

Figure 7.12: Population memory and its internal organization

In order to exploit the performance of proposed highly-parallel architecture, GU has to generate N_c new candidate configurations while another N_c candidate configurations are evaluated. Because the search algorithm utilizes a population of candidate solutions, a single genetic operator is used (i.e. mutation which inverts h bits of the configuration) and no crossover operator is applied, the number of memory accesses can be minimized by storing the differences between the configuration bitstream of the first offspring and remaining offspring.

The PowerPC keeps only the information about mutations (i.e. indices of inverted bits) and the best fitness value. FU contains a circuit generating a complete configuration bitstream for each VRC according to the partial information stored in the sections.

The mechanism controlling the bitstream generation works as follows. As soon as the evaluation is finished, the best fitness value f_{best} (out of the four evaluated individuals)

together with the index of the corresponding VRC i is sent to the PowerPC. The three situations can occur

1. if $f_{best} < f_{parent}$ then the bitstream of the first mutant is reverted to the parent bitstream by applying the mutations leading to this configuration, however in reverse order,

2. if $i > 1$ then the differences between the first mutant and i-th mutant stored in i-th section have to be reflected to the first bitstream,

3. if $i = 1$ then nothing has to be done; the configuration bitstream corresponds with the new parent bitstream.

By applying the previous steps, the first section contains the parental bitstream and a new generation can be created. Note that the inverted bits stored in sections have to be cleared before a new generation is created. The same principle is applied for remaining banks.

7.5 Experimental Evaluation

In order to evaluate the performance of the proposed solution, the problem of evolutionary design of image filters will be investigated. We will consider VRC that consists of 8 columns and 4 rows ($u = 8$ and $v = 4$). The configuration bitstream which is used to configure one VRC consists of 384 bits; i.e. 48 bits per a column are used. A single CFB is configured by 12 bits, 4 bits are used to select the connection of a single input, 4 bits are used to select one of the 16 functions. The population memory consists of 8 BRAM memories that provide 256 bit wide output. Hence a VRC with the configuration bitstream containing up to 64 bits per column can be used.

7.5.1 Theoretical Performance

Due to the pipelined reconfiguration as well as execution of VRC, the evaluation of N_c candidate programs requires $(M - 2)(N - 2)$ clock cycles, where $M \times N$ is the number of pixels of training image. The time t_{eval} needed to evaluate N_c candidate solutions can be expressed as

$$t_{eval} = (M - 2)(N - 2)\frac{1}{f} = (M - 2)(N - 2)\frac{1}{100}\mu s,$$

where f is the operation frequency ($f = 100\,\mathrm{MHz}$). Since the evaluation of a candidate solution is pipelined, the t_{eval} depends only on the number of training vectors. Note that in case of the software implementation of CGP, the evaluation time depends not only on the number of training vectors but also on the number of CGP nodes. As the generation of new candidate configurations is overlapped with the evaluation of other candidate solutions, the total time t_{total} can be expressed as

$$t_{total} = t_{init} + N_g \lceil \frac{p}{N_c} \rceil t_{eval},$$

where t_{init} corresponds with the time needed for the initialization (i.e. transferring the training data and programming the PowerPC processor), N_g is the number of generations, p is the population size and N_c is the number of VRC instances. The proposed platform provides the best performance if the number of VRC instances is equal to the population size or the population size is a multiple of the number of VRC instances ($p = kp$, where $k \in \mathbb{N}^+$). If the previous condition is met, all the VRC instances are utilized without stalls. Note that this condition does not represent any limitation since the population size is typically chosen between five and ten individuals and moreover, the population size can be adjusted according to the number of utilized VRC instances.

7.5.2 Results of Synthesis

Results of synthesis for the accelerator containing up to four VRC instances (4×8 CFBs each) are summarized in Table 7.8. The proposed system is implemented using Virtex II Pro XC2VP50 FPGA. The PowerPC works at 300 MHz, the memory interface at 150 MHz and the remaining FPGA logic including FU at 100 MHz. According to the detailed synthesis report, one instance of VRC occupies 3275 slices and 1084 flip-flops. The whole design occupies approx. 60% of the FPGA for $N_c = 4$, the four VRC instances represent approx. 90% of the design size (see Figure 7.13).

Table 7.8: Results of synthesis for various number of VRC instances (Virtex II Pro)

resource	avail.	$\mathbf{N_c = 1}$		$\mathbf{N_c = 2}$		$\mathbf{N_c = 4}$	
IO blocks	852	602	70%	602	70%	602	70%
BRAM	232	16	7%	16	7%	16	7%
SLICES	23 616	4 651	20%	7 961	34%	14 582	60%
DFF	49 788	3 536	7%	4 691	9%	7 001	14%

Table 7.9 summarizes the results of synthesis for the XC5VFX100T FPGA. This FPGA is available on the second generation of the COMBO cards (COMBO-LXT). The main difference between Virtex-5 and Virtex II Pro family is the internal structure of the basic building blocks (LUTs); while the Virtex II Pro chip contains 4-input LUTs the Virtex-5 chip utilizes LUTs with 6 inputs. Moreover, the Virtex-5 family is equipped with more powerful PowerPC processor, faster logic and larger BRAM memories. Thus a well written design usually works on higher frequency and occupies smaller area.

Table 7.9: Results of synthesis for various number of VRC instances (Virtex-5, 100 MHz)

resource	avail.	$\mathbf{N_c = 1}$		$\mathbf{N_c = 2}$		$\mathbf{N_c = 4}$		$\mathbf{N_c = 8}$	
IO blocks	640	640	94%	640	94%	602	94%	602	94%
BRAM	228	8	4%	8	4%	8	4%	12	5%
SLICES	16 000	1 828	11%	3 157	20%	5 819	36%	11 158	70%
DFF	65 280	3 633	6%	4 788	7%	7 098	11%	11 718	18%

According to the detailed synthesis report, one instance of VRC occupies 1290 slices and 1084 flip-flops. The whole design occupies approx. 40% of the FPGA for $N_c = 4$. The number of occupied resources indicates that this FPGA is able to contain approximately

2.5 times higher number of VRC instances and thus provide 2.5 times higher computational power with nearly the same power consumption.

7.5.3   Evolution of image filters

Experimental results show that approximately 25,000 candidate filters can be evaluated per second when the training set consists of 15876 8-bit vectors (i.e. a training image containing 128×128 pixels is used) and four instances of VRC are employed. Table 7.10 contains the comparison of the proposed accelerator against the recently published works dealing with the evolutionary design of image filters in terms of the number of evaluated candidate solutions per second as well as the estimated power consumption. Note that the number of evaluations per second has been calculated for the image containing 128×128 pixels. The last column of Table 7.10 contains the relative speedup. It can be seen that the proposed solution works approximately 170 times faster than the highly optimized software version of the same algorithm written in C running at the Celeron 2.4 GHz processor. Estimated results indicate that a cluster of 30 FPGAs will have the same power consumption as a common processor (65 W). Nevertheless, the cluster is capable of providing the speedup of more than 5000 supposing that one independent run of CGP is carried out using one FPGA.
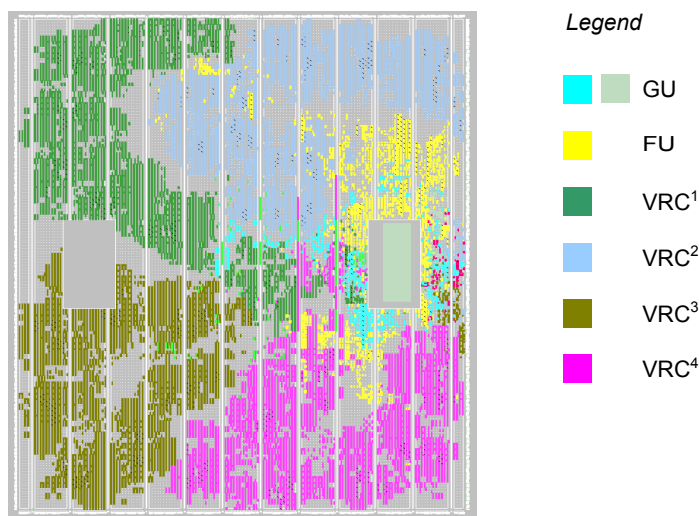

Figure 7.13: FPGA Virtex II Pro XC2VP50 utilization for the accelerator containing four VRC instances.

Apart from the FPGA-based accelerators, several papers have been published in recent years dealing with the acceleration of CGP using common GPUs [71, 31, 149]. Harding and Banzhaf achieved the speedup between 0,02 and 100 for the problem of symbolic regression using the GPU NVidia GeForce 7300 GO [71]. The direct comparison between the results is difficult, as they used extremely large CGP array (10000 nodes) and relative small number of training vectors (2000) in order to reduce the huge overhead arising during the data transfer to the GPU or accessing the content of the GPU memory. Another common approach to increase the speedup on GPUs is to introduce higher level of parallelism by increasing

Table 7.10: Comparison of the proposed accelerator with published approaches

Approach	Platform	clock freq.	power cons.	evals per sec	speedup
Accelerator with PowerPC (4 VRCs)	FPGA XC2VP50	100 MHz	2W	25195	1
Accelerator with PowerPC (1 VRC)	FPGA XC2VP50	50 MHz	2W	3150	8
Complete HW accelerator [119]	FPGA XC2V3000	50 MHz	1W	3150	8
Complete HW accelerator [189]	FPGA XCV2000	33 MHz	1W	1935	13
Muli-VRC accelerator [190]	FPGA XCV2000	30 MHz	1W	1935	13
Highly optimized SW [215]	CPU Celeron	2.4 GHz	65W	145	170
SW [189]	CPU Pentium IV	2.0 GHz	60W	16	1495

the number of individuals in the population [31, 149]. Although this approach enables to overlap the expensive data transfers with the evaluation of other individuals, the method seems to be unpractical. According to the published works, it appears that population-parallel approaches are more effective for smaller data sets but unable to compete with the FPGA-based accelerators on very large data sets.

7.6 Summary

The goal of this chapter was to present a parallel highly optimized pipelined hardware architecture designed for the acceleration of Cartesian genetic programming. The proposed accelerator consists of two main units – genetic engine and fitness unit. The fitness unit contains multiple instances of virtual reconfigurable circuit to evaluate several candidate solutions in parallel. The genetic engine is reused in all applications. Two types of virtual reconfigurable circuits are proposed. While the first VRC is devoted for symbolic regression problems over the FX representation, the second one is designed for evolution of logic circuits. In both cases a significant speedup of evolution was obtained in comparison with a highly optimized software implementation of CGP running at common gigahertz processor. The speedup can be even increased by using a modern FPGA chip (e.g. Virtex-5) doubling thus resources now capable of running at higher frequencies. In contrast with the CGP implementations based on GPUs or common CPUs, the proposed hardware accelerator provides constant speedup independently on the size of the training set (if a suitable FPGA or external memory is chosen).

Chapter 8

Conclusions

The primary goal of the research presented in this thesis is an acceleration of the evolutionary design in the field of digital systems design and optimization. We have postulated the hypothesis that the evolutionary design approach can produce interesting and human competitive solutions when the problem of scalability is reduced and thus sufficient number of generations can be utilized. Because the scalability problems significantly limit the application of evolutionary algorithms, we were primarily focused on the reduction of the fitness evaluation time which represents a serious issue in circuit evolution since the complex candidate solutions require a lot of time to be evaluated.

Proposed Approaches to the Scalability Problem Reduction

The contribution of this research can be summarized as follows. The thesis dealt with the design of various acceleration techniques that can significantly eliminate the scalability problem of evolutionary design of digital circuits at various levels of abstraction. In order to confirm our hypothesis, the work has addressed three different design classes.

In order to reduce the fitness evaluation time, a domain-specific single-chip FPGA-based accelerator has been proposed. This accelerator is designed to address the problem of the necessity of huge computation power for designing of digital circuits at the function-level. These circuits cannot be fully specified a priori, but their desired behavior is known. A typical example is the regression problem which includes e.g. evolutionary design of nonlinear image filters. The common feature of this class of circuits is that small imperfections in circuit behavior are tolerable, e.g. it is acceptable that the error of filtering is not zero but reasonably small value.

A different approach has been proposed in the area of logic synthesis, where the resulting circuits must perfectly meet the specification. A method based on applying formal verification techniques that allow a significant acceleration of the fitness evaluation procedure was proposed, overcoming thus the major bottleneck of the evolutionary synthesis at gate level. The proposed algorithm can produce complex and simultaneously innovative designs, improving thus the state-of-the art logic synthesis tools.

Finally, we have shown that there are applications that require a single training vector in order to calculate the fitness value of a candidate solution. This approach is applicable

in such cases where the utilized building blocks satisfy the properties of linearity. This method can be used at the gate level as well as function level.

The Obtained Results

It has been demonstrated that in case of the evolutionary design of nonlinear as well as linear image filters, the proposed single-chip accelerator running at 100 MHz can provide, using a moderate Xilinx FPGA XC2VP50, approximately 170 times higher performance in comparison with a common PC running at 2.4 GHz. This performance has been achieved by introducing a deeply pipelined architecture which significantly accelerates the evaluation of a candidate solution. Note that even higher performance can be achieved using the same architecture on the latest FPGA chips. The modern FPGAs are able to work on higher frequencies and provide significantly more resources that can be utilized to parallelize the process of evaluation.

By means of the proposed FPGA accelerator, very efficient nonlinear image filters have been designed. The performance of the proposed FPGA accelerator and its ability to design innovative solutions was investigated in several papers dealing with evolution of edge detectors and various impulse noise filters of lower [215] as well as high intensity [214, 213]. Extensive testing was devoted to the analysis of the optimal choice of the population size, mutation rate, size of the CGP array, size of the training image, set of node functions, pseudo-random number generator and search algorithm [215, 206]. One of the discovered implementations of an impulse noise filter consisting of four evolutionary designed filters working with the 3×3 pixel filter window combined in a bank of filters is protected by the Czech utility model (a patent application was submitted in 2009). The resulting filter exhibits high filtration quality while the implementation cost remains very low. Apart from that, some other new impulse noise filters have been evolved using the CGP-based approach [220, 212, 211].

Using the proposed acceleration algorithm employing the formal verification algorithm, a system was developed that is able to optimize the number of gates (and potentially power consumption or delay) of combinational circuits having from tens to thousands primary inputs. The proposed method has been evaluated using a set of LGSynth93 benchmark circuits and compared with conventional academia as well as commercial synthesis tools [209, 208, 219]. It was shown that the proposed method can handle circuits intractable for common EHW-based approaches utilized so far (one of the benchmark circuits has over 100 inputs and more than 1000 gates). In contrast with the conventional synthesis tools, the average improvement in terms of the number of gates provided by our system was approximately 25%. Apart from the fact that the obtained result indicates the evolutionary approach is able to generate the solutions better than conventional techniques, it also confirms the recent hypothesis that the conventional logic synthesis produces the results that are far from optimum [209].

The problem of the multiple constant multiplier design, which belongs to the class of problems where a candidate solution can be perfectly evaluated in a short time, has been investigated [223]. Although the multiple constant multiplier is not a complex circuit in

its own right, it represents a basic component of more complex circuits that form an integral part of every linear transformation (e.g. the discrete Fourier transform). Although well-optimized heuristics exist for linear transforms design, we confirmed that novel implementations of multiple constant multipliers can be designed using evolutionary algorithm.

To summarize, it has been demonstrated that the evolutionary approach is able to produce innovative solutions if an efficient evaluation procedure is employed. We have presented three different approaches to increase the performance of evolutionary algorithms and showed their applicability.

Future Work

There are several directions how to continue with this research.

It has been shown that the virtual reconfigurable circuit utilized in the proposed accelerator is able to deliver significant speedup, however due to its implementation based on multiplexers it simultaneously consumes many FPGA resources. The amount of occupied resources could then be substantially reduced when a modern FPGA supporting a fast dynamical partial reconfiguration is applied. The multiplexers that are used to connect the processing elements could be removed since the connections can be established dynamically using partial dynamical reconfiguration. If we reduce the area occupied by the VRC, we can synthesize more VRCs on a single FPGA chip and thus increase the overall performance.

The system employing the formal verification algorithm can also be improved in several ways. The SAT-based approach can perform unsatisfactory for some problem instances. For example, the time needed to decide whether two multipliers are functionally equivalent grows exponentially with the increasing number of inputs. There are tens of extensions and algorithms that have been proposed by the SAT community to improve the performance of digital circuit equivalence checking. Some of the extensions can be adopted in order to improve performance of the proposed system. In future research it is also necessary to confirm that the proposed method is able to handle large-scale optimization problems if more advanced version of the SAT solver is utilized.

Since the evolutionary synthesis based on the formal verification algorithm can handle real-world (i.e. complex) circuits, it will be probably necessary to investigate the scalability of CGP representation, efficiency of utilized genetic operators and the utilized search algorithm. As it has been shown, the evolutionary strategy with population containing only two individuals surprisingly provided the best results.

We believe that there are other applications of evolvable hardware where formal verification algorithms are directly or indirectly applicable. Further investigation is needed to identify more complex applications that can benefit from this technique.

Bibliography

[1] M. Abd-El-Barr, S. Sait, B. Sarif, and U. Al-Saiari. A modified ant colony algorithm for evolutionary design of digital circuits. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, volume 1, pages 708 – 715, 2003.

[2] Advanced Micro Devices, Inc. AMD Athlon Processor x86 Code Optimization Guide. `http://support.amd.com/us/Processor_TechDocs/22007.pdf`, 2000.

[3] M. O. Ahmad and D. Sundararajan. A fast algorithm for two-dimensional median filtering. *IEEE Transactions on Circuits and Systems*, 34:1364–1374, 1987.

[4] B. Ali, A. E. A. Almaini, and T. Kalganova. Evolutionary algorithms and theirs use in the design of sequential logic circuits. *Genetic Programming and Evolvable Machines*, 5(1):11–29, 2004.

[5] Anadigm, AN221E04 – Field Programmable Analog Arrays – User Manual, 2007. URL: `http://www.anadigm.com/_doc/UM021200-U007.pdf`.

[6] F. V. Andrade, M. C. M. Oliveira, A. O. Fernandes, and C. J. N. Coelho. SAT-based equivalence checking based on circuit partitioning and special approaches for conflict clause reuse. *Design and Diagnostics of Electronic Circuits and Systems*, pages 1–6, 2007.

[7] F. V. Andrade, L. M. Silva, and A. O. Fernandes. Improving SAT-based combinational equivalence checking through circuit preprocessing. In *26th International Conference on Computer Design, ICCD 2008*, pages 40–45, 2008.

[8] P. J. Angeline. Subtree crossover causes bloat. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 745–752. Morgan Kaufmann, 1998.

[9] T. Aoki, N. Homma, and T. Higuchi. Evolutionary Synthesis of Arithmetic Circuit Structures. *Artificial Intelligence Review*, 20(3–4):199–232, 2003.

[10] T. Aoki, N. Homma, and T. Higuchi. Evolutionary synthesis of arithmetic circuit structures. *Artificial Intelligence Review*, 20:199–232, 2003.

[11] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1996.

[12] W. Banzhaf and W. B. Langdon. Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines*, 3(1):81–91, 2002.

[13] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic Programming - An Introduction. On the Automatic Evolution of Computer Programs and its Application*. Morgan Kaufmann, Heidelberg/San Francisco, 1998.

[14] K. E. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computing Conference*, pages 307–314, 1968.

[15] P. J. Bentley. *Evolutionary Design by Computers*. Morgan Kaufmann, San Francisco CA, 1999.

[16] Berkeley Segmentation Dataset. Images, 2003. `http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/BSDS300/html/dataset/`.

[17] Berkley Logic Synthesis and Verification Group. Abc: A system for sequential synthesis and verification.

[18] M. Bidlo and J. Skarvada. Instruction-based development: From evolution to generic structures of digital circuits. *International Journal of Knowledge-Based and Intelligent Engineering Systems*, 12(3):221–236, 2008.

[19] A. Biere. Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. Technical report, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, 2010. Technical Report 10/1, August 2010, FMV Reports Series.

[20] M. Brameier and W. Banzhaf. *Linear genetic programming*. Springer, 2007.

[21] A. F. Breitzman. *Automatic Derivation and Implementation of Fast Convolution Algorithms*. PhD thesis, Philadelphia, PA, USA, 2003.

[22] D. R. K. Brownrigg. The weighted median filter. *Commun. ACM*, 27(8):807–818, 1984.

[23] D. Bull and D. Horrocks. Primitive operator digital filters. *Circuits, Devices and Systems, IEE Proceedings G*, 138(3):401 – 412, 1991.

[24] A. Burian and J. Takala. Evolved Gate Arrays for Image Restoration. In *Proc. of 2004 Congress on Evolutionary Computing CEC'04*, pages 1185–1192. IEEE Publ. Press, 2004.

[25] D. Caban. FPGA implementation of positional filters. In *Design of Embedded Control Systems*, pages 243–249. Springer-Verlag, 2005.

[26] P. Cappello and K. Steiglitz. Some complexity issues in digital signal processing. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 32(5):1037 – 1041, 1984.

[27] Cesnet, z.s.p.o. Liberouter COMBO cards. `http://www.liberouter.org/hardware.php`, 2005.

[28] C. Chakrabarti. Novel sorting network-based architectures for rank order filters. *IEEE Transactions on Very Large Scale Integration Systems*, 2(4):502–507, 1994.

[29] C. Chakrabarti. Sorting network based architectures for median filters. *Transaction on Signal Processing*, 1994.

[30] C. Chakrabarti and L. E. Lucke. VLSI architectures for weighted order statistic (WOS) filters. *Signal Processing archive*, 80(8):1419–1433, 2000.

[31] D. M. Chitty. A data parallel approach to genetic programming using programmable graphics hardware. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1566–1573, London, 2007. ACM Press.

[32] A. Chojnacki. Effective and efficient fpga synthesis through functional decomposition based on information relationship measures. In *Proceedings of the Euromicro Symposium on Digital Systems Design*, DSD '01, pages 30–, Washington, DC, USA, 2001. IEEE Computer Society.

[33] D. Cliff, I. Harvey, and P. Husbands. Explorations in evolutionary robotics. *Adaptive Behavior*, 2(1):73–110, 1993.

[34] M. Collins. Finding needles in haystacks is harder with neutrality. *Genetic Programming and Evolvable Machines*, 7:131–144, 2006.

[35] J. Cong and K. Minkovich. Optimality Study of Logic Synthesis for LUT-Based FPGAs. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 26(2):230–239, 2007.

[36] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[37] V. Crnojevic, V. Senk, and Z. Trpovski. Advanced impulse detection based on pixel-wise MAD. *SPLetters*, 11(7):589–592, July 2004.

[38] A. Das and R. Vemuri. A graph grammar based approach to automated multi-objective analog circuit design. In *DATE 2009*, pages 700–705. IEEE, 2009.

[39] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5:394–397, 1962.

[40] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. ACM*, 7:201–215, 1960.

[41] A. Dempster and M. Macleod. Constant integer multiplication using minimum adders. *Circuits, Devices and Systems, IEE Proceedings -*, 141(5):407 – 413, 1994.

[42] A. Dempster and M. Macleod. Use of minimum-adder multiplier blocks in fir digital filters. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 42(9):569 – 577, 1995.

[43] A. Dempster and M. Macleod. Comments on ldquo;minimum number of adders for implementing a multiplier and its application to the design of multiplierless digital filters rdquo;. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 45(2):242 – 243, 1998.

[44] S. Disch and C. Schollm. Combinational equivalence checking using incremental SAT solving, output ordering, and resets. *Asia and South Pacific Design Automation Conference*, pages 938–943, 2007.

[45] Y. Dong and S. Xu. A new directional weighted median filter for removal of random-valued impulse noise. *Signal Processing Letters*, 14(3):193–196, 2007.

[46] E. R. Dougherty and J. T. Astola, editors. *Nonlinear Filters for Image Processing*. SPIE/IEEE Series on Imaging Science & Engineering. SPIE/IEEE, 1999.

[47] J. Dumoulin, J. Foster, J. Frenzel, and S. McGrew. Special Purpose Image Convolution with Evolvable Hardware. In *Real-World Applications of Evolutionary Computing – Proc. of the 2nd Workshop on Evolutionary Computation in Image Analysis and Signal Processing EvoIASP'00*, volume 1803 of *LNCS*, pages 1–11. Springer-Verlag, 2000.

[48] R. Ebendt, G. Fey, and R. Drechsler. *Advanced BDD Optimization*. Springer, 2000.

[49] N. Een, A. Mishchenko, and N. Sorensson. Applying logic synthesis for speeding up SAT. *Lecture notes in computer science*, page 272, 2007.

[50] N. Een and N. Sorensson. MiniSAT. http://minisat.se.

[51] N. Een and N. Sorensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing*, pages 333–336, 2004.

[52] S. A. Fahmy, P. Y. K. Cheung, and W. Luk. Novel FPGA-based implementation of median and weighted median filters for image processing. In *FPL*, pages 142–147, 2005.

[53] P. Fiser and J. Schmidt. Small but nasty logic synthesis examples. In *Proc. 8th Int. Workshop on Boolean Problems*, pages 183–190, 2008.

[54] M. Flynn and P. Hung. Microprocessor design issues: thoughts on the road ahead. *Micro, IEEE*, 25(3):16 – 31, 2005.

[55] D. B. Fogel. *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*. IEEE Press, 1995.

[56] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966.

[57] Z. Gajda and L. Sekanina. Reducing the number of transistors in digital circuits using gate-level evolutionary design. In *Genetic and Evolutionary Computation Conference*, pages 245–252. Association for Computing Machinery, 2007.

[58] M. Garvie. *Reliable Electronics through Artificial Evolution*. PhD thesis, University of

Sussex, 2005.

[59] K. Glette and J. Torresen. A flexible on-chip evolution system implemented on a xilinx virtex-ii pro device. In *Evolvable Systems: From Biology to Hardware*, volume 3637 of *LNCS*, pages 66–75. Springer, 2005.

[60] K. Glette, J. Torresen, and M. Yasunaga. An online EHW pattern recognition system applied to face image recognition. In *Applications of Evolutinary Computing, EvoWorkshops 2007*, volume 4448 of *LNCS*, pages 271–280. Springer, 2007.

[61] K. Glette, J. Torresen, M. Yasunaga, and Y. Yamaguchi. On-Chip Evolution Using a Soft Processor Core Applied to Image Recognition. In *Proceedings 1st NASA /ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 373–380. IEEE CS Press, 2006.

[62] E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[63] E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat-solver. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '02, pages 142–, Washington, DC, USA, 2002. IEEE Computer Society.

[64] E. Goldberg, M. Prasad, and R. Brayton. Using SAT for combinational equivalence checking. In *DATE '01: Proceedings of the conference on Design, automation and test in Europe*, pages 114–121, Piscataway, NJ, USA, 2001. IEEE Press.

[65] T. G. W. Gordon and P. J. Bentley. On evolvable hardware. In *Soft Computing in Industrial Electronics*, pages 279–323, London, UK, 2002. Physica-Verlag.

[66] D. Green. *Modern Logic Design.* Addison-Wesley, 1986.

[67] J. Grimbleby. Automatic analogue network synthesis using genetic algorithms. In *Genetic Algorithms in Engineering Systems: Innovations and Applications, 1995. GALESIA. First International Conference on (Conf. Publ. No. 414)*, pages 53 –58, 1995.

[68] O. Gustafsson, A. Dempster, and L. Wanhammar. Extended results for minimum-adder constant integer multipliers. In *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, volume 1, pages I–73 – I–76, 2002.

[69] P. C. Haddow and A. Tyrrell. Challenges of evolvable hardware: past, present and the path to a promising future. *Genetic Programming and Evolvable Machines*, 12:183–215, 2011.

[70] S. Harding. Evolution of image filters on graphics processor units using cartesian genetic programming. In *2008 IEEE World Congress on Computational Intelligence*, pages 1921–1928, Hong Kong, 2008. IEEE Computational Intelligence Society, IEEE Press.

[71] S. Harding and W. Banzhaf. Fast genetic programming on GPUs. In *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 90–101, Valencia, Spain, 2007. Springer.

[72] S. Harding and W. Banzhaf. Implementing cartesian genetic programming classifiers on graphics processing units using gpu.net. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, GECCO '11, pages 463–470, New York, NY, USA, 2011. ACM.

[73] S. L. Harding and W. Banzhaf. Distributed genetic programming on GPUs using CUDA. In I. Hidalgo, F. Fernandez, and J. Lanchares, editors, *Workshop on Parallel Architectures and Bioinspired Algorithms*, pages 1–10, Raleigh, NC, USA, 2009. Universidad Complutense de Madrid.

[74] S. L. Harding, J. F. Miller, and W. Banzhaf. Self modifying cartesian genetic programming: Parity. In *2009 IEEE Congress on Evolutionary Computation*, pages 285–292. IEEE Press, 2009.

[75] R. Harjani, R. A. Rutenbar, and L. R. Carley. A prototype framework for knowledge-based

analog circuit synthesis. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, DAC '87, pages 42–49, New York, NY, USA, 1987. ACM.

[76] F. Henrici, J. Becker, A. Buhmann, M. Ortmanns, and Y. Manoli. A continuous-time field programmable analog array using parasitic capacitance gm-c filters. In *Proc. IEEE International Symposium on Circuits and Systems*, pages 2236–2239. IEEE, 2007.

[77] T. Higuchi, M. Iwata, I. Kajitani, H. Iba, Y. Hirao, T. Furuya, and B. Manderick. Evolvable hardware and its application to pattern recognition and fault-tolerant systems. In *Towards Evolvable Hardware*, Lecture Notes in Computer Science, pages 118–135. Springer Berlin / Heidelberg, 1996.

[78] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, M. Salami, N. Kajihara, and N. Otsu. Real-World Applications of Analog and Digital Evolvable Hardware. *IEEE Transactions on Evolutionary Computation*, 3(3):220–235, 1999.

[79] 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. of the 2nd International Conference on Simulated Adaptive Behaviour*, pages 417–424. MIT Press, 1993.

[80] J. A. Hilder, J. A. Walker, and A. M. Tyrrell. Use of a multi-objective fitness function to improve cartesian genetic programming circuits. In *NASA/ESA Conference on Adaptive Hardware and Systems*, pages 179–185. IEEE, 2010.

[81] A. J. Hirst. Notes on the evolution of adaptive hardware. In *Proc. of Adaptive Computing in Engineering Design and Control*, pages 212–219. Plymouth, U.K., 1996.

[82] J. Holland. *Adaptation in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press, 1975.

[83] G. Hornby, A. Globus, D. S. Linden, and J. Lohn. Automated Antenna Design with Evolutionary Algorithms. In *Proc. 2006 AIAA Space Conference*, page 8, San Jose, CA, 2006. AIAA.

[84] D. Horrocks and M. Spittle. Component value selection for active filters using genetic algorithms. *First On-line Workshop on Soft Computing (WSC1), Special Session on*, page 19, 1996.

[85] H. Hwang and R. Haddad. Adaptive median filters: new algorithms and results. *IP*, 4(4):499–502, April 1995.

[86] H. Hwang and R. A. Haddad. New algorithms for adaptive median filters. In *Proc. SPIE Vol. 1606, Visual Communications and Image Processing '91*, pages 400–407, 1991.

[87] H. Iba, M. Iwata, and T. Higuchi. Machine learning approach to gate-level evolvable hardware. In *Evolvable Systems: From Biology to Hardware*, volume 1259 of *Lecture Notes in Computer Science*, pages 327–343. Springer Berlin / Heidelberg, 1997.

[88] M. Järvisalo. Equivalence checking hardware multiplier designs, 2007. SAT Competition 2007 benchmark description. Available at http://www.satcompetition.org/2007/contestants.html.

[89] T. Kalganova. An extrinsic function-level evolvable hardware approach. In *Proceedings of the European Conference on Genetic Programming*, pages 60–75, London, UK, 2000. Springer-Verlag.

[90] T. Kalganova. Bidirectional incremental evolution in extrinsic evolvable hardware. In *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 65–74. IEEE Computer Society, Silicon Valley, USA, July 2000.

[91] T. Kalganova and J. F. Miller. Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. In *The First NASA/DoD Workshop on Evolvable Hardware*, pages 54–63. IEEE Computer Society, 1999.

[92] P. Kaufmann and M. Platzner. Advanced techniques for the creation and propagation of modules in cartesian genetic programming. In *Proc. of Genetic and Evolutionary Computation Conference, GECCO 2008*, pages 1219–1226. ACM, 2008.

[93] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching (2nd ed.)*. Addison Wesley, 1998.

[94] S. Ko and Y. Lee. Center weighted median filters and their applications to image enhancement. *IEEE Transactions on Circuits and Systems*, 15:984–993, 1991.

[95] H. Kog and L. Guan. A noise-exclusive adaptive filtering framework for removing impulse noise in digital images. *IEEE Signal Processing Letters*, 45:422–428, 1998.

[96] P. Koivisto, J. Astola, V. Lukin, V. Melnik, and O. Tsymbal. Removing Impulse Bursts from Images by Training-Based Filtering. *EURASIP Journal on Applied Signal Processing*, 2003(3):223–237, 2003.

[97] P. Koivisto, H. Huttunen, and P. Kuosmanen. Training-based optimization of soft morphological filters. *Journal of Electronic Imaging*, 5(3):300–322, 1996.

[98] J. Korenek and L. Sekanina. Intrinsic evolution of sorting networks: A novel complete hardware implementation for FPGAs. In *Evolvable Systems: From Biology to Hardware*, volume 3637 of *LNCS*, pages 46–55. Springer Verlag, 2005.

[99] J. R. Koza. The Annual „HUMIES" Awards. http://www.genetic-programming.org/hc2011/combined.html.

[100] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[101] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA, 1994.

[102] J. R. Koza. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3–4):251–284, 2010.

[103] J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane. Four problems for which a computer program evolved by genetic programming is competitive with human performance. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 1 –10, 1996.

[104] J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane. The design of analog circuits by means of genetic programming. In P. J. Bentley, editor, *Evolutionary Design by Computers*, chapter 16, pages 365–385. Morgan Kaufmann, San Francisco, USA, 1999.

[105] J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[106] J. R. Koza and L. W. Jones. Automated re-invention of six patented optical lens systems using genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2005*, pages 1953–1960. ACM Press, 2005.

[107] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003.

[108] P. Lakamsani, R. Yang, B. Zeng, and M. Liou. Design and implementation of a programmable stack filter. In *ICIP94*, pages 664–667, 1994.

[109] W. K. Lam. *Hardware Design Verification: Simulation and Formal Method-Based Approaches*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2008.

[110] J. Langeheine. *Intrinsic Hardware Evolution on the Transistor Level*. PhD thesis, 2005.

[111] T. Larrabee. Test pattern generation using boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 11:4–15, 1992.

[112] C.-C. Lee, J.-H. R. Jiang, C.-Y. R. Huang, and A. Mishchenko. Scalable exploration of functional dependency by interpolation and incremental SAT solving. In *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, ICCAD '07, pages 227–233, Piscataway, NJ, USA, 2007. IEEE Press.

[113] C. M. Li. A constraint-based approach to narrow search trees for satisfiability. *Information Processing Letters*, 71:75–80, 1999.

[114] D. S. Linden. *Automated design and optimization of wire antennas using genetic algorithms*. PhD thesis, 1997.

[115] R. Maheshwari, S. S. S. P. Rao, and E. G. Poonacha. FPGA implementation of median filter. In *VLSI Design*, pages 523–524, 1997.

[116] J. Marques-Silva. Practical applications of boolean satisfiability. In *Workshop on Discrete Event Systems (WODES'08)*. IEEE Press, 2008.

[117] S. Marshall. New direct design method for weighted order statistic filters. *VISP*, 151(1):1–8, February 2004.

[118] P. Martin. *Genetic Programming in Hardware*. PhD thesis, University of Essex, 2003.

[119] 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 3637 of *LNCS*, pages 76–85. Springer Verlag, 2005.

[120] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[121] T. McConaghy, P. Palmers, G. G. E. Gielen, and M. Steyaert. Simultaneous multi-topology multi-objective sizing across thousands of analog circuit topologies. In *DAC 2007*, pages 944–947. IEEE, 2007.

[122] T. McConaghy, P. Palmers, M. Steyaert, and G. G. E. Gielen. Trustworthy genetic programming-based synthesis of analog circuit topologies using hierarchical domain-specific building blocks. *Evolutionary Computation, IEEE Transactions on*, 15(4):557–570, 2011.

[123] K. L. McMillan. Interpolation and SAT-based model checking. *Computer Aided Verification*, pages 1–13, 2003.

[124] U. Meyer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays*. Springer, 2007.

[125] J. F. Miller. Digital filter design at gate-level using evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 1999*, pages 1127–1134. Morgan Kaufmann, 1999.

[126] J. F. Miller. What bloat? Cartesian Genetic Programming on Boolean problems. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2001*, pages 295–302. Morgan Kaufmann Publishers, 2001.

[127] J. F. Miller, D. Job, and V. K. Vassilev. Principles in the Evolutionary Design of Digital Circuits – Part I. *Genetic Programming and Evolvable Machines*, 1(1):8–35, 2000.

[128] J. F. Miller, D. Job, and V. K. Vassilev. Principles in the Evolutionary Design of Digital Circuits – Part II. *Genetic Programming and Evolvable Machines*, 1(2):259–288, 2000.

[129] J. F. Miller and S. L. Smith. Redundancy and Computational Efficiency in Cartesian Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174, 2006.

[130] J. F. Miller and P. Thomson. Aspects of digital evolution: Geometry and learning. In

Proceedings of the Second International Conference on Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science, pages 25–35. Springer-Verlag, 1998.

[131] J. F. Miller and P. Thomson. Cartesian Genetic Programming. In *Proc. of the 3rd European Conference on Genetic Programming EuroGP2000*, volume 1802 of *LNCS*, pages 121–132. Springer, 2000.

[132] J. F. Miller, P. Thomson, and T. Fogarty. Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study. In *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, pages 105–131. Wiley, 1997.

[133] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, DAC '01, pages 530–535, New York, NY, USA, 2001. ACM.

[134] M. Murakawa et al. Evolvable hardware at function level. In *In Proc. of the Parallel Problem Solving from Nature IV*, volume 1141 of *LNCS*, pages 62–71. Springer Berlin / Heidelberg New York, 1996.

[135] N. Nedjah and L. de Macedo Mourelle. Evolutionary Synthesis of Synchronous Finite State Machines. In *Evolvable Machines: Theory and Practice*, pages 103–127, Berlin, 2005. Springer.

[136] M. Nikolova. A variational approach to remove outliers and impulse noise. *J. Math. Imaging Vis.*, 20(1-2):99–120, 2004.

[137] P. Nordin. *A compiling genetic programming system that directly manipulates the machina code*, pages 311–331. MIT Press, Cambridge, MA, USA, 1994.

[138] P. Nordin. *Evolutionary Program Induction of Binary Machine Code and its Applications*. PhD thesis, 1997.

[139] E. Ochotta, R. Rutenbar, and L. Carley. Synthesis of high-performance analog circuits in astrx/oblx. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 15(3):273 –294, 1996.

[140] M. Oltean and C. Grosan. Evolving digital circuits using multi expression programming. In *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*, pages 87–97, Seattle, 2004. IEEE Press.

[141] G. Pask. Physical analogues to the growth of a concept. In A. Uttley, editor, *Mechanisation of thought processes*, pages 765–794. National Physical Laboratory H.M.S.O., 1958.

[142] E. Pavlenko, M. Wedler, D. Stoffel, W. Kunz, O. Wienand, and E. Karibaev. *A New Verification Technique for Custom-Designed Components at the Arithmetic Bit Level*, volume Languages for Embedded Systems and their Applications of *Lecture Notes in Electrical Engineering*, chapter 17, pages 257–272. Springer Netherlands, 2009.

[143] T. Pecenka, Z. Kotasek, L. Sekanina, and J. Strnadel. Automatic discovery of RTL benchmark circuits with predefined testability properties. In *2005 NASA / DoD Conference on Evolvable Hardware*, pages 51–58. IEEE Computer Society, 2005.

[144] R. Poli and J. Page. Solving high-order boolean parity problems with smooth uniform crossover, sub-machine code gp and demes. *Genetic Programming and Evolvable Machines*, 1:37–56, 2000.

[145] R. Porter. *Evolution on FPGAs for Feature Extraction*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2001.

[146] P. Porwik. The spectral test of the boolean function linearity. *Int. J. Appl. Math. Comput. Sci*, 13:567–575, 2003.

[147] M. Püschel, B. Singer, J. Xiong, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, and R. W. Johnson. SPIRAL: A generator for platform-adapted libraries of signal processing

algorithms. *Journal of High Performance Computing and Applications, special issue on "Automatic Performance Tuning"*, 18(1):21–45, 2004.

[148] M. Püschel, A. C. Zelinski, and J. C. Hoe. Custom-optimized multiplierless implementations of DSP algorithms. In *International Conference on Computer-Aided Design (ICCAD)*, pages 175–182, 2004.

[149] D. Robilliard, V. Marion-Poty, and C. Fonlupt. Population parallel gp on the g80 gpu. In *Proc. of European Conference on Genetic Programming*, volume 4971 of *LNCS*, pages 98–109. Springer-Verlag, 2008.

[150] F. Russo and G. Ramponi. A fuzzy filter for images corrupted by impulse noise. *IEEE Transactions on Circuits and Systems*, 45:168–170, 1996.

[151] S. Safarpour, A. Veneris, G. Baeckler, and R. Yuan. Efficient SAT-based boolean matching for FPGA technology mapping. In *Proceedings of DAC 2006*, 2006.

[152] H. Sakanashi, M. Iwata, and T. Higuchi. EHW Applied to Image Data Compression. In T. Higuchi, Y. Liu, and X. Yao, editors, *Evolvable Hardware*, pages 19–40. Springer, 2006.

[153] M. Salami, M. Murakawa, and T. Higuchi. Data compression based on evolvable hardware. In *Evolvable Systems: From Biology to Hardware*, Lecture Notes in Computer Science, pages 167–179. Springer Berlin / Heidelberg, 1997.

[154] M. Salami, H. Sakanashi, M. Tanaka, M. Iwata, T. Kurita, and H. T. On-line compression of high precision printer images by evolvable hardware. In *Proc. of the Data Compression Conference*, pages 219–228, Los Alamitos, CA, U.S.A, 1998.

[155] S. Schulte, M. Nachtegael, V. D. Witte, D. V. der Weken, and E. E. Kerre. Fuzzy impulse noise reduction methods for color images. In *Computational Intelligence, Theory and Applications International Conference 9th Fuzzy Days in Dortmund*, pages 711–720. Springer Verlag, 2006.

[156] H. P. Schwefel. *Evolution and Optimum Seeking*. John Wiley, New York, 1995.

[157] L. Sekanina. Image Filter Design with Evolvable Hardware. In *Applications of Evolutionary Computing – Proc. of the 4th Workshop on Evolutionary Computation in Image Analysis and Signal Processing EvoIASP'02*, volume 2279 of *LNCS*, pages 255–266, Kinsale, Ireland, 2002. Springer Verlag.

[158] L. Sekanina. *Evolvable components: From Theory to Hardware Implementations*. Natural Computing. Springer-Verlag Berlin, 2004.

[159] L. Sekanina and M. Bidlo. Evolutionary design of arbitrarily large sorting networks using development. *Genetic Programming and Evolvable Machines*, 6(3):319–347, 2005.

[160] L. Sekanina and T. Martínek. Evolving image operators directly in hardware. In S. Cagnoni, E. Lutton, and G. Olague, editors, *Genetic and Evolutionary Computation for Image Processing and Analysis, EURASIP Book Series on Signal Processing and Communications, Volume 8*, pages 93–112. Hindawi Publishing Corporation, 2007.

[161] L. Sekanina and R. Růžička. Easily Testable Image Operators: The Class of Circuits Where Evolution Beats Engineers. In *Proc. of the 2003 NASA/DoD Conference on Evolvable Hardware*, pages 135–144, Chicago, USA, 2003. IEEE Computer Society.

[162] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-vincentelli. Sis: A system for sequential circuit synthesis. Technical report, University California, Berkeley, 1992.

[163] B. Shackleford. A high-performance, pipelined, FPGA-based genetic algorithm machine. *Genetic Programming and Evolvable Machines*, 2(1):33–60, 2001.

[164] A. P. Shanthi and R. Parthasarathi. Practical and scalable evolution of digital circuits. *Applied Soft Computing*, 9(2):618–624, 2009.

[165] A. P. Shanthi, L. K. Singaram, and R. Parthasarathi. Evolution of asynchronous sequential circuits. In *Evolvable Hardware'05*, pages 93–96, 2005.

[166] X. She. Fast evolution of large digital circuits. *W. Trans. on Comp.*, 7:1988–2000, 2008.

[167] K. O. Stanley and R. Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9:93–130, 2003.

[168] D. Stoffel and W. O. Kunz. Equivalence checking of arithmetic circuits on the arithmetic bit level. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2004.

[169] A. Stoica, D. Keymeulen, R. Tawel, C. Salazar-Lazaro, and W.-t. Li. Evolutionary experiments with a fine-grained reconfigurable architecture for analog and digital cmos circuits. In *Proceedings of the 1st NASA/DOD workshop on Evolvable Hardware*, EH 1999, pages 76–84, Washington, DC, USA, 1999. IEEE Computer Society.

[170] A. Stoica, R. S. Zebulum, M. I. Ferguson, D. Keymeulen, and V. Duong. Evolving circuits in seconds: Experiments with a stand-alone board-level evolvable system. In *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware (EH'02)*, pages 67–64, Washington, DC, USA, 2002. IEEE Computer Society.

[171] A. Stoica, R. S. Zebulum, X. Guo, D. Keymeulen, M. Ferguson, and V. Duong. Taking evolutionary circuit design from experimentation to implementation: some useful techniques and a silicon demonstration. *IEE Proceedings - Computers and Digital Techniques*, 151(4):295–300, 2004.

[172] E. Stomeo, T. Kalganova, and C. Lambert. Generalized disjunction decomposition for evolvable hardware. *IEEE Transaction Systems, Man and Cybernetics, Part B*, 36(5):1024–1043, 2006.

[173] T. Sun and Y. Neuvo. Detail-preserving median based filters in image processing. *Pattern Recognition Letters*, 16:341–347, 1994.

[174] G. Sussman and R. Stallman. Heuristic techniques in computer-aided circuit analysis. *Circuits and Systems, IEEE Transactions on*, 22(11):857 – 865, 1975.

[175] A. Thompson. Silicon evolution. In *Proceedings of the First Annual Conference on Genetic Programming*, GECCO '96, pages 444–452, Cambridge, MA, USA, 1996. MIT Press.

[176] J. Torresen. A Divide-and-Conquer Approach to Evolvable Hardware. In *Proc. of the 2nd International Conference on Evolvable Systems: From Biology to Hardware ICES'98*, volume 1478 of *LNCS*, pages 57–65, Lausanne, Switzerland, 1998. Springer.

[177] J. Torresen. Possibilities and limitations of applying evolvable hardware to real-world applications. In *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*, volume 1896 of *Lecture Notes in Computer Science*, pages 230–239. Springer Berlin / Heidelberg, 2000.

[178] J. Torresen. A scalable approach to evolvable hardware. *Genetic Programming and Evolvable Machines*, 3(3):259–282, 2002.

[179] G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.

[180] G. Tufte and P. C. Haddow. Prototyping a GA Pipeline for Complete Hardware Evolution. In A. Stoica, D. Keymeulen, and J. Lohn, editors, *Proc. of the 1st NASA/DoD Workshop on Evolvable Hardware*, pages 143–150, Pasadena, CA, USA, 1999. IEEE Computer Society.

[181] A. Upegui and E. Sanchez. Evolvable FPGAs. In *Reconfigurable Computing*, pages 725–752. Morgan Kaufmann, 2008.

[182] V. K. Vassilev, D. Job, and J. F. Miller. Towards the Automatic Design of More Efficient Digital Circuits. In *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 151–160, Los Alamitos, CA, USA, 2000. IEEE Computer Society.

[183] V. K. Vassilev, J. F. Miller, and T. C. Fogarty. On the nature of two-bit multiplier landscapes. In *EH 1999: Proceedings of the 1st NASA/DOD workshop on Evolvable Hardware*, page 36, Washington, DC, USA, 1999. IEEE Computer Society.

[184] M. N. Velev. Efficient translation of boolean formulas to CNF in formal verification of microprocessors. *Asia and South Pacific Design Automation Conference*, pages 310–315, 2004.

[185] Y. Voronenko and M. Püschel. Multiplierless multiple constant multiplication. *ACM Transactions on Algorithms*, 3(2), 2007.

[186] L. Žaloudek and L. Sekanina. Transistor-level evolution of digital circuits using a special circuit simulator. In *Evolvable Systems: From Biology to Hardware*, volume 5216 of *Lecture Notes in Computer Science*, pages 320–331. Springer Berlin / Heidelberg, 2008.

[187] J. A. Walker, J. A. Hilder, and A. M. Tyrrell. Evolving variability-tolerant cmos designs. In *Evolvable Systems: From Biology to Hardware*, volume 5216 of *Lecture Notes in Computer Science*, pages 308–319. Springer Berlin / Heidelberg, 2008.

[188] J. A. Walker and J. F. Miller. The Automatic Acquisition, Evolution and Re-use of Modules in Cartesian Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 12(4):397–417, 2008.

[189] J. Wang, Q. Chen, and C. Lee. Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware. *IET computers and digital techniques*, 2(5):386–400, 2008.

[190] J. Wang, C. Piao, and C. Lee. Implementing multi-vrc cores to evolve combinational logic circuits in parallel. In *Evolvable Systems: From Biology to Hardware*, volume 4684 of *LNCS*, pages 23–34, 2007.

[191] G. Wilson and W. Banzhaf. A comparison of cartesian genetic programming and linear genetic programming. In *Genetic Programming*, volume 4971 of *Lecture Notes in Computer Science*, pages 182–193. Springer Berlin / Heidelberg, 2008.

[192] Xilinx Inc. Xilinx FPGAs. http://www.xilinx.com/products/silicon-devices/fpga/index.htm.

[193] Xilinx Inc. Xilinx Virtex-II Pro Platform FPGAs. http://www.xilinx.com/partinfo/ds031.pdf, 2005.

[194] C. Yan, M. Ciesielski, and V. Singhal. BDS: a BDD-based logic optimization system. In *Design Automation Conference, 2000. Proceedings 2000. 37th*, pages 92 –97, 2000.

[195] S. Yang. Logic synthesis and optimization benchmarks user guide version 3.0. Technical report, 1991. MCNC, Technical Report.

[196] S. Yanushkevich, D. M. Miller, V. P. Shmerko, and R. S. Stankovic. *Decision Diagram Techniques for Micro- and Nanoelectronic Design Handbook*. CRC, 2006.

[197] X. Yao and T. Higuchi. Promises and Challenges of Evolvable Hardware. *IEEE Transactions on Systems, Man, and Cybernetics – Part C*, 29(1):87–97, 1999.

[198] T. Yu and J. F. Miller. Finding needles in haystacks is not hard with neutrality. In *Proceedings of the Fifth European Conference on Genetic Programming (EuroGP-2002), volume 2278 of LNCS*, pages 13–25. Springer-Verlag, 2002.

[199] S.-Q. Yuan and Y.-H. Tan. Erratum to „impulse noise removal by a global-local noise detector and adaptive median filter": [signal processing 86 (8) (2006) 2123-2128]. *Signal Processing*, 87(5):1171, 2007.

[200] R. S. Zebulum, M. Pacheco, and M. Vellasco. Evolvable systems in hardware design: Taxonomy, survey and applications. In *Evolvable Systems: From Biology to Hardware*, volume 1259 of *Lecture Notes in Computer Science*, pages 344–358. Springer Berlin /

Heidelberg, 1997.

[201] R. S. Zebulum, M. Pacheco, and M. Vellasco. *Evolutionary Electronics - Automatic Design of Electronic Circuits and Systems by Genetic Algorithms.* The CRC Press International Series on Computational Intelligence, 2002.

[202] R. S. Zebulum, M. S. Vellasco, and M. A. Pacheco. Variable length representation in evolutionary electronics. *Evol. Comput.*, 8(1):93–120, 2000.

[203] H. Zhang. SATO: An efficient propositional prover. In *Proceedings of the 14th International Conference on Automated Deduction*, CADE-14, pages 272–275, London, UK, 1997. Springer-Verlag.

[204] S. Zhao and L. Jiao. Multi-objective evolutionary design and knowledge discovery of logic circuits based on an adaptive genetic algorithm. *Genetic Programming and Evolvable Machines*, 7(3):195–210, 2006.

Author's publications

Journal papers

[205] Z. Vašíček and L. Sekanina. Evoluční návrh kombinačních obvodů. *Elektrorevue*, 2004(43):6, 2004.

[206] Z. Vašíček and L. Sekanina. An evolvable hardware system in Xilinx Virtex II Pro FPGA. *International Journal of Innovative Computing and Applications*, 1(1):63–73, 2007.

[207] Z. Vašíček and L. Sekanina. Hardware accelerator of cartesian genetic programming with multiple fitness units. *Computing and Informatics*, 29(6+):1359–1371, 2010.

[208] Z. Vašíček and L. Sekanina. Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. *Genetic Programming and Evolvable Machines*, 12(3):305–327, 2011.

Conference papers

[209] P. Fišer, J. Schmidt, Z. Vašíček, and L. Sekanina. On logic synthesis of conventionally hard to synthesize circuits using genetic programming. In *Proc. of the 13th Int. IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 346–351. IEEE Computer Society, 2010.

[210] L. Sekanina and Z. Vašíček. On the practical limits of the evolutionary digital filter design at the gate level. In *Applications of Evolutionary Computing*, volume 3907 of *Lecture Notes in Computer Science*, pages 344–355. Springer Berlin / Heidelberg, 2006.

[211] Z. Vašíček and M. Bidlo. Evolutionary design of robust noise-specific image filters. In *IEEE Congress on Evolutionary Computation*, pages 269–276. IEEE Computer Society, 2011.

[212] Z. Vašíček, M. Bidlo, L. Sekanina, and K. Glette. Evolutionary design of efficient and robust switching image filters. In *Proc. of the 2011 NASA/ESA Conference on Adaptive Hardware and Systems*, pages 192–199. IEEE Press, 2011.

[213] Z. Vašíček, M. Bidlo, L. Sekanina, J. Torresen, K. Glette, and M. Furuholmen. Evolution of impulse bursts noise filters. In *Proc. of the 2009 NASA/ESA Conference on Adaptive Hardware and Systems*, pages 27–34. IEEE Press, 2009.

[214] Z. Vašíček and L. Sekanina. An area-efficient alternative to adaptive median filtering in

FPGAs. In *Proc. of 2007 Conf. on Field Programmable Logic and Applications*, pages 216–221. IEEE Computer Society, 2007.

[215] Z. Vašíček and L. Sekanina. Evaluation of a new platform for image filter evolution. In *Proc. of the 2007 NASA/ESA Conference on Adaptive Hardware and Systems*, pages 577–584. IEEE Computer Society, 2007.

[216] Z. Vašíček and L. Sekanina. Reducing the area on a chip using a bank of evolved filters. In *Evolvable Systems: From Biology to Hardware*, volume 4684 of *Lecture Notes in Computer Science*, pages 222–232. Springer Verlag, 2007.

[217] Z. Vašíček and L. Sekanina. Hardware accelerators for cartesian genetic programming. In *European Conference on Genetic Programming*, volume 4971 of *Lecture Notes in Computer Science*, pages 230–241. Springer Verlag, 2008.

[218] Z. Vašíček and L. Sekanina. Novel hardware implementation of adaptive median filters. In *Proc. of 2008 IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop*, pages 110–115. IEEE Computer Society, 2008.

[219] Z. Vašíček and L. Sekanina. A global postsynthesis optimization method for combinational circuits. In *Proc. of the Design, Automation and Test in Europe, DATE*, pages 1525–1528. IEEE Computer Society, 2011.

[220] Z. Vašíček, L. Sekanina, and M. Bidlo. A method for design of impulse bursts noise filters optimized for FPGA implementations. In *DATE 2010: Design, Automation and Test in Europe*, pages 1731–1736. European Design and Automation Association, 2010.

[221] Z. Vašíček and K. Slaný. Efficient phenotype evaluation in cartesian genetic programming. In *15th European Conference on Genetic Programming*, volume 7244 of *Lecture Notes in Computer Science*, pages 265–276. Springer Verlag, 2012.

[222] Z. Vašíček, L. Čapka, and L. Sekanina. Analysis of reconfiguration options for a reconfigurable polymorphic circuit. In *Proceedings of the 2008 NASA/ESA Conference on Adaptive Hardware and Systems*, pages 3–10. IEEE Computer Society, 2008.

[223] Z. Vašíček, M. Žádník, L. Sekanina, and J. Tobola. On evolutionary synthesis of linear transforms in FPGA. In *Proc. of the 8th Int. Conference on Evolvable Systems: From Biology to Hardware*, volume 5216 of *LNCS*, pages 141–152, Berlin, 2008. Springer Verlag.

Books

[224] L. Sekanina, Z. Vašíček, R. Růžička, M. Bidlo, J. Jaroš, and P. Švenda. *Evoluční hardware: Od automatického generování patentovatelných invencí k sebemodifikujícím se strojům*. Academia, Praha, 2009.

Other publications

[225] M. Bidlo and Z. Vašíček. Cellular automata-based development of combinational and polymorphic circuits: A comparative study. In *Evolvable Systems: From Biology to Hardware*, volume 6274 of *Lecture Notes in Computer Science*, pages 106–117. Springer Verlag, 2008.

[226] M. Bidlo and Z. Vašíček. Comparison of the uniform and non-uniform cellular automata-based approach to the development of combinational circuits. In *Proc. of NASA/ESA Conference on Adaptive Hardware and Systems*, pages 423 – 430. IEEE Computer Society, 2009.

[227] M. Bidlo and Z. Vašíček. Investigating gate-level evolutionary development of combinational

multipliers using enhanced cellular automata-based model. In *IEEE Congress on Evolutionary Computation*, pages 2241–2248. IEEE Computer Society, 2009.

[228] M. Bidlo, Z. Vašíček, and K. Slaný. Sorting network development using cellular automata. In *Evolvable Systems: From Biology to Hardware*, volume 6274 of *Lecture Notes in Computer Science*, pages 85–96. Springer Berlin / Heidelberg, 2010.

[229] T. Dulík, Z. Křivka, J. Kadlec, M. Bližňák, V. Budíková, O. Jirák, N. Olšarová, J. Trbušek, and Z. Vašíček. *Virtuální laboratoř pro vývoj aplikací s mikroprocesory a FPGA*. CERM, Brno, 2011.

[230] O. Jirák, Z. Křivka, N. Olšarová, and Z. Vašíček. Odvozování propojení komponent pro podporu návrhu pro malé fpga čipy. In *Proc. of the DATAKON 2010*, pages 81–90. VŠB TU, 2010.

[231] O. Jirák, Z. Křivka, and Z. Vašíček. Component interconnection inference tool supporting the design of small fpga-based embedded systems. In *Proc. of the IADIS International Conference Applied Computing 2010*, pages 230–234. IADIS Press, 2010.

[232] O. Jirák, Z. Křivka, and Z. Vašíček. Integrated development environment for virtual laboratory. In *International Technology, Education and Development Conference*, page 10. IATED, 2011.

[233] Z. Křivka and Z. Vašíček. The virtualization of development boards in the virtual laboratory of microprocessor technology. In *12th International Carpathian Control Conference*, pages 424–428. VŠB TU, 2011.

[234] L. Sekanina, R. Růžicka, Z. Vašíček, R. Prokop, and L. Fujčík. Repomo32 – new reconfigurable polymorphic integrated circuit for adaptive hardware. In *2009 IEEE Workshop on Evolvable and Adaptive Hardware*, pages 39–46. IEEE Computational Intelligence Society, 2009.

[235] Z. Vašíček. Evolutionary synthesis of gate-level digital circuits. In *Proceedings of 11th Conference and Competition HONEYWELL EMI 2005*. FEKT VUT, 2005.

[236] Z. Vašíček. Implementation of high-performance reconfigurable systems on a chip. In *Proceedings of 12th Conference and Competition STUDENT EEICT 2006 Volume 2*, pages 232–234. FEKT VUT, 2006.

[237] Z. Vašíček. Reálné aplikace evolučního návrhu. In *Počítačové architektury a diagnostika 2007. Česko-slovenský seminář pro studenty doktorandského studia*, pages 137–142. University of West Bohemia in Pilsen, 2007.

[238] Z. Vašíček. Towards automatic design of competitive image filters in fpgas. In *Proceedings of Junior Scientist Conference*. TU-Wien, 2008.

[239] Z. Vašíček, L. Čapka, and L. Sekanina. Analysis of reconfiguration options for a reconfigurable polymorphic circuit. In *Proceedings of the 2008 NASA/ESA Conference on Adaptive Hardware and Systems*, pages 3–10, Washington, DC, USA, 2008. IEEE Computer Society.

[240] L. Čapka and Z. Vašíček. Investigating the influence of mutation operators in cartesian genetic programming. In *13th International Conference on Soft Computing*, pages 43–47. Faculty of Mechanical Engineering BUT, 2007.