VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

NEW METHODS FOR SYNTHESIS AND APPROXIMATION OF LOGIC CIRCUITS

HABILITAČNÍ PRÁCE
HABILITATION THESIS

AUTOR PRÁCE
AUTHOR

Ing. Zdeněk Vašíček, Ph.D.

BRNO 2016

Preface

This thesis presents seven research papers related to the synthesis of logic circuits which I have published, mostly with my colleagues, since 2011. An overarching idea that ties together the included papers is the application of unconventional techniques, in particular evolutionary algorithms, in the area of synthesis and optimization of digital systems. This work is motivated by the fact that the alternative approaches have inherent ability to provide results that are unreachable by ordinary synthesis tools. Unfortunately, the evolutionary algorithms are known to be computationally demanding. Hence, we are trying to combine them with various formal approaches to improve scalability of the design process.

Two papers address the problem of evolutionary synthesis of common logic circuits. In both papers satisfiability solvers were employed to improve scalability. The remaining five papers belong to the relative young and rapidly growing research area known as approximate computing. The goal of synthesis process is to approximate original circuits in order to improve their energy efficiency while keeping introduced error at reasonable level. Two papers discuss the approximation of key arithmetic circuit such as adders and multiplier. The remaining three papers are devoted to the approximation of complex problem instances such as median circuits, sorting networks or neural networks. In these papers, binary decision diagrams are typically employed to exactly measure the error introduced by approximation.

Four of the presented papers were published in international journals with an impact factor including *IEEE Transactions on Evolutionary Computation* and *Genetic Programming and Evolvable Machines*. The remaining three papers were presented at prestigious conferences in the field such as *International Conference On Computer Aided Design*. One paper without coauthors, presented at the leading European event on bio-inspired computation, *European Conference on Genetic Programming*, was awarded Best Paper Award in 2015. In addition to that, two journal papers were awarded Human-Competitive Award, a prestigious award which is annually presented at the *Genetic and Evolutionary Computation Conference* for outstanding results that have been obtained automatically using evolutionary approaches and that are considered human-competitive.

The thesis contains an introductory part which is followed by an overview of the papers constituting the core contribution of this research. The last part is concluded with discussion about the future work. Reprints of the papers are enclosed in appendices.

Brno, October 5th, 2016 Zdeněk Vašíček

Acknowledgements

First of all, I would like to thank Lukáš Sekanina for sparking my interest in evolutionary computation, for keeping my administrative workload as small as possible, for his permanent availability for discussions and for giving me the great opportunity to work with him and realize and develop my ideas without experiencing financial problems.

I also thank the members of the Computer Systems department at Faculty of Information technology and members of Evolvable hardware group, for a great working environment. I should not forget to thank our faculty for providing computer facilities and for technical support using them. Without the cluster of super computers, this work would never exist.

During my time I had the pleasure to supervise several great students, but I would like to thank especially to Vojtěch Mrázek for hard work and valuable contributions.

Finally, my greatest thanks go to my family and my wife Monika for her support, her patience with me, and her never-ending love, my daughter Alexandra and my son Jan for being a source of motivation and pleasure.

Parts of the work presented in this thesis have been supported by the Czech Science Foundation under project P103/10/1517 (*Natural Computing on Unconventional Platforms*), 14-04197S (*Advanced Methods for Evolutionary Design of Complex Digital Circuits*), 16-17538S (*Relaxed equivalence checking for approximate computing*), by the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070, by Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II), project IT4Innovations excellence in science – LQ1602 and by Brno University of Technology under Grant FIT-S-11-1 (*Advanced secured, reliable and adaptive IT*) and FIT-S-14-2297 (*Architecture of parallel and embedded computer systems*).

Contents

Appendices - Paper reprints

Chapter 1

Introduction

With the increasing degree of integration and scaling, the power consumption of electronic systems has emerged as a pressing issue. Energy efficiency is one of the major driving forces of current semiconductor industry, which is relevant for supercomputers on the one hand, as well as small portable personal electronics and sensors on the other hand.

Recently, the worldwide semiconductor industry formally acknowledged a fact which has become more or less inevitable. Moore's law, the principle that has powered the information-technology revolution since the 1960s, is nearing its end [Wal16]. Moore's law is seen as a rule accepted and followed by the industry. It states that the number of transistors in an integrated circuit will double almost every two years which generally means that the chip's performance will double too.

The exponential improvement that the law describes significantly transformed the electronic manufacturing industry, especially electronic circuit production. 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. Nowadays, we are surrounded with ubiquitous high-speed Internet, smartphones, intelligent appliances such as TVs, refrigerators, cars or even buildings and many other devices that are able to interoperate within the Internet.

The advances in technology enabled to place more and more transistors in the same area. Recently released Intel Xeon E5-2600 has circuit features that are around 14 nanometres across and contains approximately 7.2 billion transistors on 456 square millimeters. The circuit density is remarkable, more than 15 million transistors are crammed on the square millimeter. Compared to the technology available in 1975, the current technology enables to place more than six orders of magnitude of transistors into the same area. Unfortunately, the increasing integration density is inevitably connected with a side effect – increasing heat dissipation. As electrons had to move faster and faster through silicon circuits that are smaller and smaller, the chips began to get too hot.

Around 2004, the processors hit the power wall. Manufacturers decided to stop to increase the clock frequency and had to break the frequency scaling to keep the amount of dissipated heat at manageable level. To keep the chips moving along the Moore's law performance curve, the CPU manufacturers had to redesign their processors and propose an alternative way to improve performance – multi-core processors. Despite of that, the doubling started to falter as a natural response to the introduced frequency limit and other fundamental technology limits. Due to the huge amount of transistors, only a fraction of an integrated circuit can actually be active at any given point in time without violating power constraints. We gradually entered into the era of Dark silicon – the growing gap between the number of transistors that can be placed into a chip with each advancement in technology and the number of transistors we can actually use simultaneously with a given power budged. It appears that the only route to enable scaling is a specialization – spending a large fraction of the area on specialized cores which are much faster at some useful tasks. The requirements on computing increasingly becomes to be defined by the needs of server farms (known as the cloud), high-end smartphones and tablets. Hence, specialized and energy efficient subcircuits needs to be introduced and embedded in the next-generation processors to keep up with conflicting constraints, to prevent mobile devices from draining the battery on the one hand and to avoid burning the chips in data centers on the other hand. Authors of the latest technology roadmap for semiconductors predict that the increasing integration density and computational potential of ubiquitous smart sensors allow not only to retrieve but also analyze, assemble and summarize information and provide actionable recommendations. The energy efficiency thus plays and extremely important role to ensure a sustainable growth of semiconductor industry.

The power consumption can be reduced at different levels including the architectural level, circuit level, layout level, and the fabrication process technology level [RP09]. For more than two decades the designers could rely more or less on the advances in technology that are typically connected with drop in supply voltage resulting in decrease of power dissipation. Around 2000, a new limiting factor of future microprocessor integration emerged and disrupted this rule [K+03]. In order to explain this phenomenon, the nature of power consumption in digital circuits needs to be discussed in a greater detail. The power dissipation in digital circuits can be decomposed to dynamic and static part. The dynamic power arises from the logic transitions and is present only when a transistor is switching, that is, when a gate is changing its output value, and charges or discharges the load capacitances. The static power dissipation, sometimes denoted as leakage power, is caused by the current that leaks through transistors even when they are turned off. When we decrease the supply voltage, the dynamic power dissipated by common CMOS circuits decreases as well, but quadratically. If an energy-efficient and low-power circuit were needed, the most efficient approach would be to fabricate this circuit using the newest technology process that provides transistors with lower parasitic capacitances and lower supply voltage. In addition to that, it would be beneficial to optimize the circuits for switching activity because the dynamic power consumption constitutes a predominant part of the total power consumption. As the features began to shrink below about 65 nm, the leakage power has become a top concern for designers in deep submicron process technology nodes because it has increased to 30-50% of the total power consumption [Sha12]. It means that not the internal activity of the circuit, but the number of active transistors on a chip began to have significant impact on the total power consumption.

An obvious method to reduce power consumption is to shut down a part of a circuit when it is not in operating conditions, i.e. to introduce a power management strategy. Despite of its efficiency, this approach, however, does not represent a universal solution that could be applicable in any situation because not all circuits contain sub-circuits that can be turned off. In addition to the reduction of load capacitances, power supply voltage reduction and introducing an intelligent power management, a considerable potential for power saving exists at the circuit design level or architectural design level. Compared to the other methods, the improvement of design techniques is very cost effective because the investment to reduce power by design is relatively small [IP98]. The dynamic power dissipation can be reduced by reducing the switching activity of a logic circuit which involves to alter (optimize) the circuit's structure. The leakage power consumption of digital circuits is, in general, proportional to the area of the circuits [IP98; Mat+15]. Hence, when we optimize the circuits for area, we can expect a reduction in the power consumption. One of the possibilities how to improve the area of a circuit is to reduce its complexity provided that the Boolean function captured by this circuit remains unchanged. This kind of optimization, typically performed at gate-level, is widely supported by state-of-the-art synthesis tools. Another possibility is to revise the algorithms and applications implemented on a chip in the top-down fashion. It has been shown recently, for example, that about 83 % of runtime is spent in computations exhibiting an inherent tolerance to errors in computation [Chi+13]. The inherent error resilience means that it is not always necessary to implement precise and usually area expensive circuits. Instead, much simpler, approximate, circuits may be used to solve a given problem without any significant degradation in the output quality.

1.1 Logic synthesis and its efficiency

Logic synthesis, as understood by hardware community, is a process that transforms a high-level description into a gate-level or transistor-level implementation [WCC09]. Due to the complexity of the problem, the synthesis process is typically broken into a sequence of steps which gradually transform the abstract description into an actual implementation. The transformation steps are subdivided into three classes: operations on abstract representations (high-level synthesis), operations on logic descriptions (logic-level synthesis) and operations on geometric representations (physical-design synthesis). The first two steps are part of the so called *frontend VLSI design flow*. The third step is part of the backend process that is responsible for the physical implementation.

In this thesis, we are interested only in the logic-level (i.e. logic) synthesis. The logic synthesis typically starts with description given at register-transfer level (RTL) in the form of a VHDL or Verilog code, but the specification can be given even at lower levels. For example, a sub-optimal gate-level netlist can be the starting point. In such a case we speak about logic optimization, because the goal of the logic optimization is to transform a sub-optimal solution into an optimal gate-level implementation with respect to given synthesis goals.

Logic synthesis is further subdivided into two phases – technology-independent phase and technology-dependent phase (see Figure 1.1). First, the input description is optimized using several technology-independent techniques before technology-dependent optimizations are performed. Due to the scalability issues, the problem is typically represented using a suitable internal representation. The typical cost function during the technology-independent optimizations is total literal count of the factored representation of the logic function (which correlates quite well with circuit area). Then, the technology-dependent optimization is employed. In this step, the internal representation is mapped into a network of gates in a given technology. The simple cost estimates are replaced by more accurate ones. Mapping is constrained by many factors such as the set of available gates (logic functions and drive strength), delay, power, and area on a chip.

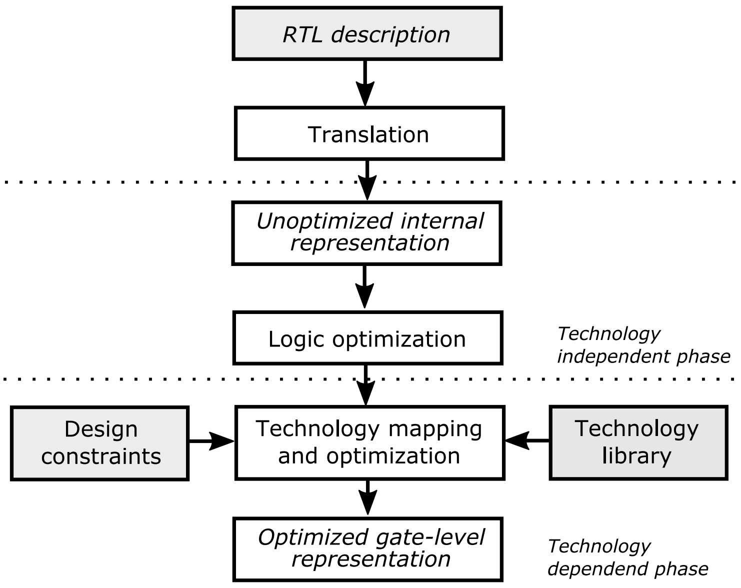

RTL description
Translation
Unoptimized internal representation
Logic optimization
Technology independent phase
Design constraints
Technology mapping and optimization
Technology library
Optimized gate-level representation
Technology dependend phase

Figure 1.1: Traditional VLSI design flow

Current state-of-the-art logic synthesis tools, such as ABC [Mis12], represent circuits using a directed acyclic graph composed of two-input AND nodes connected by direct or negated edges denoted as and-inverter graph (AIG). The typical synthesis flow is as follows. The gate-level circuits are loaded, translated and internally represented using AIG, followed by optimization performed on top of AIG. Finally, the optimized AIG is mapped back to gates. The goal of synthesis is to perform delay-aware minimization of the number of AND nodes as this number correlates to the number of gates after implementation. Three basic operations are employed to minimize AIG: refactoring, rewriting and balancing [MCB06]. The balancing is used to ensure that the number of logic levels is not increased after AIG rewriting. The AIG representation is simple and scalable, and leads to simple algorithms. At the same time, it suffers from an inherent bias in representation. While eight of ten possible two-input logic gates may be represented by means of a single AIG node, XOR and XNOR gate require three AIG nodes each. Despite of unquestionable advantages of several dozens of algorithms developed on top of AIGs, the efficiency of synthesis is limited as it mostly fully relies on local optimizations disallowing to increase the number of AIG nodes. It has been shown that there exists a huge class of real-world circuits for which the synthesis fails and provides very poor results, i.e. circuits whose size is far from optimum after the synthesis [FS08]. Unfortunatelly, the ability to capture XOR gates is essential especially for new emerging technologies enabling compact representation of arithmetic and XOR-intensive logic [Gai+15].

The inherent inefficiency of state-of-the-art logic synthesis tools provides an opportunity how to improve the synthesis results, decrease complexity of obtained circuits and consequently reduce the area on a chip and power consumption. In order to improve the efficiency, it is necessary to either introduce more advanced methods in traditional synthesis or employ some of the unconventional synthesis techniques.

1.2 From error resilience to power-aware logic synthesis

The demand for energy-efficient systems led to a new paradigm known as approximate computing. One of the approaches that belongs to this research area, is approximate logic synthesis. The difference between the classical logic synthesis and approximate logic synthesis is in the way how the synthesis process treats the specification. While the common logic synthesis guarantees functional equivalence at all levels, i.e. the specification in the form of a RTL description is functionally equivalent with gate-level as well as transistor-level implementation, the approximate approximate synthesis relaxes some of the strict rules at some levels. In order to guarantee functional equivalence, the common logic synthesis typically employs formal methods that are able to prove or disprove equivalence across various levels. The approximate computing methods build on various *error metrics* that measures distance between specification and approximate implementation because it is necessary to keep the error under some level to obtain a reasonable implementation.

The logic synthesis tools originally aimed at performance optimization and area minimization. However, the requirement for energy-efficient circuits forced designers to update the available design methods and include power dissipation as the third design parameter. Power optimization and power estimation were incorporated not only into the technology dependent phase, but also into technology-independent part of the VLSI design flow. While the power optimization techniques are applied to meet the design constrains, the role of power estimation techniques is to evaluate the effect of design modifications on power consumption. As discussed in [SPG02], the design becomes more difficult because speed and low power dissipation are in most cases contradictory factors. Approximate design flow extends this concept and introduces fourth design parameter – the error. The evolution of major design parameters considered in VLSI design flow is summarized in Figure 1.2. In approximate design flow, various error metrics are employed to quantify various aspects of "the quality" of circuits. Typically an energy-accuracy trade-off is sought, but other trade-offs could be obtained in such a complex design space.

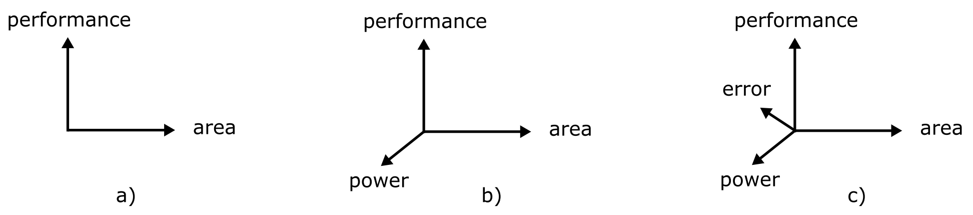


Figure 1.2: Design parameters considered in a) traditional design flow, b) low-power design flow and c) approximate design flow.

Error resilience of many real-world applications probably offers the greatest potential for power savings. Unfortunately the conventional synthesis tools have never been constructed to perform the synthesis of approximate circuits where no golden design exists for an approximate circuit albeit logic synthesis and optimization represents the research area with more than fifty years of history. One of the possibilities how to design energy efficient approximate circuits is to constrain the number of available resources (gates) and let the synthesis tool to produce a circuit with minimal error with respect to the accurate circuit. Interestingly, there is no conventional method that could directly solve such an optimization problem.

1.3 Research motivation

In the beginning of nineties, a new field applying evolutionary techniques to hardware design and synthesis has been established. This multi-disciplinary field, reffered to as evolvable hardware, offers a promising way how to overcome some of the drawbacks of conventional synthesis algorithms [HLY06]. The evolutionary algorithms (EAs) do not require any auxiliary data structure and can operate directly on gate-level representation. The main feature of EAs is that they do not employ any deterministic synthesis algorithm and all the optimizations are being done implicitly, without any structural biases as inevitable for transformations with local scope. Evolutionary design is capable of constructing partially working solutions even if sufficient resources which are required for implementing a fully functional solution are not available. Despite of many advantages and great success stories in many different areas, it is fair to say that there is an issue tightly connected with evolvable hardware since its early beginnings – bad scalability. The most complex circuit that was directly evolved at gate-level before 2010 consists of tens of gates and has around 20 inputs [SKL06a]. It is clear that those results could barely compete with conventional circuit design tools producing circuits counting thousands of gates and hundreds of inputs.

For more than 12 years, my research interests have revolved around application of evolutionary approaches in the area of circuit design inspired, as many others, by many success stories achieved within the field of evolutionary computation in the past. Since 2010, I have been engaged in the problem of evolutionary synthesis of logic circuits, motivated partly by the fact that the logic synthesis still represents an open problem and partly by the desire to improve scalability of the evolutionary approach itself. Currently, I'm interested mainly in the application of formal techniques in the area of evolutionary synthesis of digital circuits as it seems to be the only solution that could address the scalability issues.

Nowadays, we can argue that we were able to break through the imaginary limits of evolvable hardware and push this research field a bit closer to industry. It is fair to say, however, that it would not happen without a bit of luck and many hours spent investigating at first sight blind directions.

Note that the evolutionary community strictly distinguishes between two different problems. If the evolution starts with an existing conventional solution (a gate-level netlist, or a RTL netlist, for example) and the goal is to improve some circuit parameters, we speak about *evolutionary optimization*. In the case when only a behavioral specification, given for example in the form of a truth table or some other canonical representation such as binary decision diagram, is available, the evolution needs to be initialized using randomly generated circuits and we speak about *evolutionary design* or *evolution from scratch*. The first goal is to discover circuit structure and then optimize its properties.

1.4 Research objectives

The goal of this thesis is to map our progress and summarize the main results that we have achieved in the area of evolutionary synthesis of accurate and approximate logic circuits. As the evolutionary synthesis of logic circuits represents a multidisciplinary problem, the contribution is twofold. We will discuss the achievements from the perspective of the evolutionary community on the one hand, and achievements from the perspective of the hardware community on the other hand.

1.5 Thesis organization

The rest of the thesis is organized as follows. First two chapters are devoted to the evolutionary synthesis of logic circuits. The next three chapters are then related to the approximate computing. In Chapter 2, we provide a brief introduction to evolvable hardware, the essential concept tightly connected with this thesis, followed by a critical review of this research field. The following chapter summarizes the results obtained in the area of evolutionary synthesis of logic circuits. Chapter 4 introduces approximate computing and discusses the main challenges. Results obtained in the context of approximate computing are divided into two chapters. Chapter 5 describes the methods we proposed to cope with the problem of design of approximate logic circuits. Chapter 6 summarizes the formal methods we developed to exactly assess the quality evaluation of approximate circuits. The thesis concludes with prospects of future research. The research summary given in chapters 3, 5 and 6 is accompanied by seven papers which provide more details about the discussed topics.

Chapter 2

Evolvable hardware and logic synthesis

Advancements in technology developed in the early nineties enabled researchers to sucessfully apply techniques of evolutionary computation in various problem domains. In the middle nineties, Higuchi and Thompson, two of the most prominent pioneers, demonstrated that evolutionary algorithms are able to solve non-trivial hardware-related problems [Hig+93; Tho96]. The achievements presented in the seminal paper of Higuchi et al. [Hig+93] motivated other scientists to intensively explore a new and promising research topic. As a consequence of that a new research direction referred to as *Evolvable hardware* (EHW) has emerged [GB02]. Evolvable hardware, a field of evolutionary computation, focuses on the use of evolutionary algorithms to create specialized electronics without manual engineering. The vision of EHW is to replace expensive and sometimes unreliable designers and develop robust, flexible and survivable autonomous systems. EHW draws inspiration from three fields, namely, biology, computer science and electronic engineering.

Several schemes have been developed for classifying the evolvable harware [GB02]. Usually, two research areas are distinguished: *evolutionary circuit design* and *evolvable circuits*. In the first case, evolutionary algorithms are used as a tool that is employed to design a system that meets a predefined specification. For example, genetic programming can be used to discover an area-efficient implementation of a circuit whose function is specified by a truth table. In the second case, the evolutionary algorithm is an inherent part of an evolvable circuit. The resulting adaptive system is autonomously reconfigured with evolved configurations to adapt or repair its functionality in a changing environment.

In the context of circuit design, the evolvable hardware is a very attractive approach as it provides another option to the traditional design methodology – to use evolution to design circuits for us. The key strength of the evolvable hardware is that it allows solutions to be generated from a behavioral description. In addition to that, evolvable hardware can be applied for designing solutions to poorly specified problems, i.e. problems that cannot be fully specified a priori, but whose desired behavior is known. Signal filtering is a typical example of a poorly specified problem. In this case, artificially created training data are usually available and can be employed to specify the behavior of a filter. Another often emphasized advantage of this approach is that circuits can be customized and adopted for a particular environment. For example, if we know that some input combinations in our target application occur with a relative low probability, we can take this information into account, simplify the specification and design a circuit which shows better parameters such as reduced size, delay or power consumption.

The gate-level evolution has been addressed only rarely before the year 2000. The first results in the area of digital circuit synthesis were reported by Koza in 1992, who investigated the evolutionary design of even-parity circuits in his extensive discussions of the standard genetic programming (GP) paradigm [Koz92]. Later, Thompson used a form of direct encoding loosely based on the structure of an FPGA in his experiment with evolution of a square wave oscillator [Tho96]. Genetic algorithm has been employed also by Coello who evolved various 2-bit adders and multipliers [CCA98]. Finally, 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 [MTF97; Mil99a]. He was interested in the evolutionary design of simple arithmetic circuits and digital filters.

The method of evolving digital circuits developed by Miller in 1997 [MTF97] was subsequently revised and a new evolutionary algorithm known as Cartesian genetic programming (CGP) was introduced in 2000 [MT00]. CGP, which is a general form of genetic programming, was designed to address two issues related to the efficiency of common tree-based genetic programming. Firstly, as GP represents candidate solutions using trees, it does not naturally capture the structure of digital circuits. Secondly, GP exhibits the so-called bloat effect enabling the programs to grow uncontrollably until they reach the GP's tree-depth maximum. In order to avoid the bloat, Miller proposed to represent given problems using a two-dimensional grid of nodes. Compared to the generic programming, the number of nodes is fixed. Despite of that, the representation allows to capture any directed acyclic graph because not all the nodes need to be referenced in the path from inputs to outputs. CGP uses a very simple yet efficient integer-based encoding scheme which can directly be used as an intermediate code for an interpreter employed to determine response for a given input sequence. The search is performed using a mutation-based evolutionary strategy denoted as $(1 + \lambda)$-ES that is operating with the population of $1 + \lambda$ individuals. The search strategy works as follows. The initial population is created, evaluated using a fitness function and the fittest individual is identified. The fitness value, calculated by the fitness function, indicates how well a candidate solution fulfills the problem objective; in other words it indicates how a particular candidate solution meets the specification. Then, every new population consists of the best individual of the previous population and its λ offspring. The offspring are created by a point mutation operator which modifies a predefined number of randomly selected genes of the best individual and produces new, but valid, candidate solution. The moment the population is created, the fitness value of each offspring is calculated. The fittest individual in the population is selected and the evolutionary loop continues by creating of new population. The evolution is terminated when the maximum number of generations is exhausted or when a satisfactory solution is found. For detailed description of CGP, see attached paper [Vas+11c] (Appendix A).

CGP has been used to demonstrate that evolutionary computing can improve results of conventional circuit synthesis and optimization algorithms. As a proof-of-concept, small arithmetic circuits were considered. A 4-bit multiplier was the most complex circuit evolved in this category [VJM00]. For the next decade, however, the problems addressed by the EHW community remained nearly of the same complexity. The most complex combinational circuits that were directly evolved during the first two decades of EHW consisted of tens of gates and had around 20 inputs [SKL06a].

During the next decade (since the 2000), many researchers invested an enormous effort on proposing new ways enabling to simplify the problem for evolution in terms of finding more effective approaches to explore the search space. Many novel techniques including decomposition, development, modularization and even new problem representations have been proposed [SKL06a; SP09; Mil11; ZJ06]. Despite of that, only a little progress was achieved and the gap between the complexity of problems addressed in industry and EHW continued to widen as the advancements in technology developed. This supported a belief that evolutionary design works better for analogue circuits rather than digital circuits possibly due to the fact that analogue behaviors provide relatively smoother search spaces [Sto+99].

In order to address the increasing complexity of real-world designs, some authors escaped from the gate-level representation and used function-level evolution. Instead of the usage of simple gates, larger building blocks such as adders and multipliers are employed. Several patentable implementations of digital circuits were discovered, especially in the area of digital signal processing [Sek04; Mil11; Vas+13]. One of the most complex circuits evolved by means of the function-level approach is a random shot noise image filter with 25 inputs consisting of more than 1,500 two-input gates when synthesized [Vas+13]. Despite of that, evolvable hardware found itself in a critical stage around the year 2010 and it was not clear whether there exists a path forward which would allow the field to progress [HT11]. The scalability problem has been identified as one of the most difficult problems the researchers are faced in this field and that should be, among others, addressed by the next generation of EHW.

The poor scalability typically causes that evolutionary algorithm is able to provide solutions to small problem instances only and a partially working solution is usually returned in other cases. The scalability problem can primarily be seen from two perspectives: *scalability of representation* and *scalability of fitness evaluation* [TT15]. From the viewpoint of the scalability of representation, the problem is that long chromosomes are usually required to represent complex solutions. Long chromosomes, however, imply large search spaces that are typically difficult to search. The scalability of fitness evaluation represents another big challenge. The problem is that complex candidate solutions might require a lot of time to be evaluated. As a consequence of that, only a small fraction of the search space may be explored in a reasonable time.

Chapter 3

Evolutionary synthesis of logic circuits

Notwithstanding the pessimism surrounding the EHW community, researchers continued to investigate how to overcome the scalability issues. Various techniques were proposed in literature to improve the scalability of evaluation and increase the performance of the evolutionary synthesis of logic circuits. Some authors coped with the bad scalability by introducing various decomposition techniques that break complex problems into small instances [SKL06b; SP09]. Other authors introduced CGP-based accelerators benefiting from the fixed-length representation of CGP. In this category, FPGA-based [WCL08; Vas+10], GPU-based [HB11] and even CPU-based [Vas+12b] accelerators were created. Unfortunatelly, none of these approaches has provided a general recipe how to solve the problem of scale. The evolutionary synthesis naturally tends to produce sub-optimal solutions when a decomposition is introduced. The proposed accelerators are able to deliver a substantial, yet linear, speedup. Hence the limits of evolutionary design are pushed forward only slightly as the time needed to evaluate a candidate solution typically grows exponentially.

Moreover, we noticed that there is an additional problem related to the scalability of fitness evaluation, *scalability of specification* [Vas+14b]. The problem is that the frequently used behavioral specification in the form of truth table does not scale itself. Not only is it impossible to specify complex circuits in practice, it is also infeasible to evaluate their response using a circuit simulator. The reason is that the amount of memory required to store the whole truth table as well as the number of rows (and consequently the number of input combinations that need to be checked against specification) grows exponentially with the increasing number of inputs.

3.1 Synthesis of logic circuits using satisfiability solvers

Interestingly, the poor scalability of specification represents a problem that was overlooked by the EHW community since introducing the concept of evolvable hardware. An increasing number of researchers has been drawing attention to the another problem of scale – scalability of representation that was believed to be the root preventing EAs to handle complex problem instances. The most important feature of the evolutionary optimization is that each candidate solution must be functionally equivalent with its parent in order to be further evaluated. This feature was first utilized in [Vas+11c] (Appendix A) and further elaborated in [Vas+11a] where we demonstrated that it is feasible to handle complex digital circuits provided that a common truth table based fitness evaluation procedure is replaced with a formal verification method. In order to reduce the time needed to determine the fitness value, an approach routinely used in the area of logic synthesis known as combinational equivalence checking was employed. In particular, combinational equivalence checking using satisfiability solvers was employed. The combinational equivalence checking has a great potential to improve the scalability of fitness evaluation and, at the same time, it enables to avoid truth tables to specify circuit behavior. Any circuit implementing desired Boolean function (specification) could be employed instead. Such a circuit can easily be obtained by means of a common synthesis tool. At this point, the reader is referred to paper [Vas+11c] (Appendix A) which describes the proposed method. The following paragraphs provide additional observations.

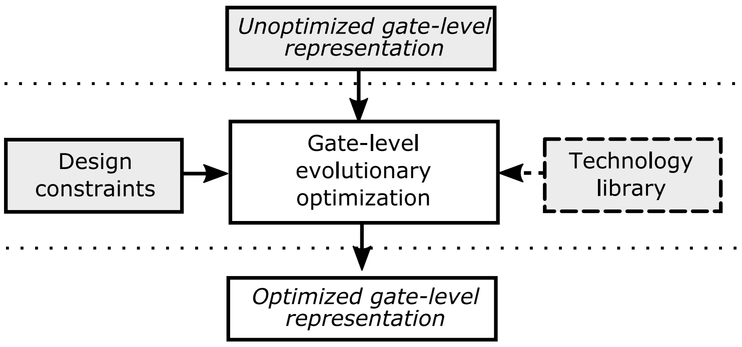

Unoptimized gate-level representation
Design constraints
Gate-level evolutionary optimization
Technology library
Optimized gate-level representation

Figure 3.1: Evolutionary synthesis of logic circuits.

But it was not the equivalence checking alone that enabled to achieve speedups of several orders of magnitude. The proposed approach benefits from a tight connection between the evolutionary algorithm and combinational equivalence checking [Vas+11a]. Since every fitness evaluation is preceded by a mutation (as briefly explained in previous section), 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 and gates that have to be compared with the reference circuit, and only these outputs and gates are checked. If an empty list is obtained (e.g. some inactive gate was modified), the equivalence checking is skipped and the corresponding offspring receives the same fitness as its parent. In addition to that, the parental circuit serves simultaneously as a reference. This helps to improve the efficiency of equivalence checking because the complexity of the optimized circuit typically decreases in the course of evolution. All these tricks helped to significantly improve the scalability of the equivalence checking even for a pathological cases of circuits where the SAT-based equivalence checking does not scale well. Speedup factor higher than 3.6 thousands was reported for evolutionary optimization of 11-bit multipliers [Vas+11a].

The performance of the proposed method can be discussed from two perspectives – the ability to improve scalability of fitness evaluation and the ability to handle and optimize large digital circuits. 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 evaluation performed on circuits having from 12 to 32 inputs revealed that while the time of the improved functional equivalence checking increases linearly with the increasing number of inputs, the time required to evaluate response for all input combinations grows exponentially. In the case of 30-input parity benchmark, the proposed SAT-based method was able to evaluate about 138 thousands times more candidate solutions per second. The speedup even increases with increasing the number of inputs. With such an improvement in performance, common benchmark sets, routinely applied in the area of logic synthesis for many years, could be optimized by CGP. The experiments performed using the LGSynth93 benchmark set clearly demonstrated the hidden power of evolutionary approaches on the one hand, and inefficiency of state-of-the-art synthesis algorithms on the other hand. The proposed method achieved 37.8% reduction in the number of gates (on average) across various benchmark circuits counting from 67 to 1408 gates and having from 22 to 128 inputs [Vas+11c] (Appendix A). The results of an extended evaluation were presented in [Vas+11a]. In this paper, a comparison with two academia and three commercial conventional synthesis tools was included. Surprisingly, the results confirmed that the CGP-based method is able to significantly outperform all the conventional tools. There was only one case for which evolution produced slightly worse result. The evolved circuit contains four more gates than the best result obtained by conventional tools. The achieved reduction ranges from -0.7% to 40.4% despite the fact that all the synthesis tools shared the same experimental setup. It seems that they are all based on the same principles and algorithms. Apart from the ability to perform XOR decomposition, the advantage of the evolutionary synthesis is that it enables to skip the whole technology-independent phase (see Figure 3.1 and 1.1) which may introduce a bias. The optimization is performed directly at the gate-level representation taking into account physical properties of the gates.

The efficiency of the SAT-based method was further improved in [Vas15] (Appendix B) where we combined an adaptive high-performance circuit simulator with formal verification in order to detect the functional non-equivalence of the parent and its offspring. This approach is based on the following observations. Firstly, candidate solutions that are not functionally equivalent with a given specification form a predominant part of the total number of generated candidate solutions. Secondly, the time needed to simulate a given candidate circuit using a limited set of test vectors is significantly lower than the time which is consumed by a SAT solver. Hence we can employ a circuit simulator to quickly disprove the equivalence between a candidate solution and its parent. The number of test vectors is adaptively modified during evolution in order to achieve the best possible performance. An extensive set of 100 real-world benchmarks circuits was used to evaluate the performance the proposed method. The least complex circuit consisted of 106 gates and had 15 primary inputs and 38 outputs. The most complex circuit, an audio codec controller, contained 16,158 gates and used 2,176 inputs and 2,136 outputs. One half of the benchmark circuits had more than 50 primary inputs and consisted of more than thousand gates. For more than half of the benchmark circuits, approximately five times higher number of evaluations was performed within the same time period compared to the previous approach that utilizes only a SAT solver. Unfortunately, the value of speedup noticeably varies across the benchmarks. There are cases for which the speedup factor exceeded 30. On the other hand, nearly no improvement was obtained for five benchmarks. For more information, see Figure 6 in [Vas15] (Appendix B). While the previous method was able to reduce these benchmark circuits by 21% in average, the proposed method led to a 34% reduction in the number of gates. The results are summarized in Figure 7 in [Vas15] (Appendix B). Considering the fact that the runtime of the optimization process was 15 minutes, the obtained results are very encouraging.

3.2 Synthesis of multi-functional logic circuits

Obtaining flexibility, adaptation and multi-functionality directly at the hardware level represents another goal we can observe in the EHW field. One of possible approaches to achieving a low cost reconfiguration could be based on multi-functional gates. The multi-functional gates have special physical structures enabling to behave differently with respect to external conditions. Multi-functional logic gates based on graphene P-N junctions were designed, for example. These gates are capable of performing several logic functions just by adjusting some control voltages [Tan+10]. In addition to that, CMOS-based polymorphic gates controlled via power supply voltage (Vdd) or even temperature were fabricated [SZK01; Sto+04]. As the Vdd-based control does not require any additional wires, the use of polymorphic gates could reduce interconnecting networks in reconfigurable chips significantly. In order to address this problem, we have introduced a tool for evolutionary synthesis of polymorphic circuits consisting of polymorphic gates operating in two different modes [SV12]. The proposed tool is based on a SAT-based equivalence checking and it was designed to address not only the performance bottleneck of the previously published evolutionary approaches [GS11], but also inefficiency of common synthesis approaches based on polymorphic multiplexing [GS11] or Poly-BDDs [GS11] that tend to produce far-from-optimum solutions. The principle remains nearly the same as for SAT-based evolutionary synthesis of logic circuits. The only difference is that it is necessary to perform the functional checking twice because each polymorphic gate operates inherently in two modes. In fact, two specifications are required – one specifying Boolean function implemented in the first mode and the second one to specify behavior of the circuit when the gates are switched to the second mode. The specification is derived from the reference solution. In order to perform the functional equivalence check efficiently, only the cone of influence (COI) determined according to the points of mutation enters the SAT solver. In fact, COI is computed for each mode independently and the result is combined to simplify the resulting CNF submitted to the SAT solver. The optimization starts with a reference solution that is obtained by applying polymorphic multiplexing. The experimental evaluation was performed on sixteen benchmark circuits whose complexity ranges from 264 to 2325 gates having up to 49 inputs. Two experimental sets were considered – benchmarks created using polymorphic multiplexing and the same benchmarks optimized using ABC. Interestingly, very similar results were obtained in both cases. Although the second set contains smaller circuits compared to the unoptimized benchmark set, we obtained more compact circuit only in a few cases. The reference solutions obtained using the state-of-the-art synthesis approaches were reduced by 35% (the number of gates) in average.

3.3 Binary decision diagrams in synthesis of logic circuits

If we replace the SAT-based equivalence checking with equivalence checking based on binary decision diagrams (BDDs), it is possible to evolve digital circuits from scratch that means without seeding the initial population with an already working circuit [Vas+14b]. It is fair to admit, however, that obtaining a fully functional solution from a randomly seeded population would consume in general a considerable time because the evolutionary design approach exploits the generate-and-test principle and no additional knowledge about the problem is available. Hence, the evolutionary synthesis from scratch can hardly compete with conventional state-of-the-art synthesis tools when the time of synthesis is considered. Despite of that, this problem represents an interesting research challenge that could also serve as a good benchmark for performance evaluation of various evolutionary algorithms. In addition to that, the evolutionary synthesis from scratch may be utilized in adaptive evolvable embedded systems where a simple logic circuit has to be created for some reason and running a standard circuit design packages is usually infeasible on such systems.

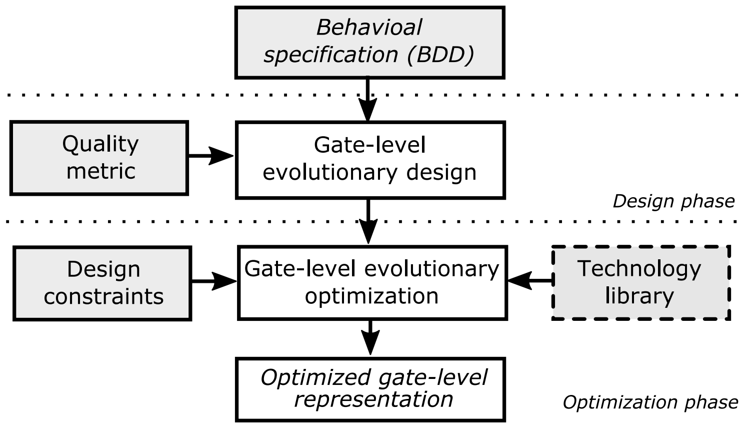

Behavioal specification (BDD)
Quality metric
Gate-level evolutionary design
Design phase
Design constraints
Gate-level evolutionary optimization
Technology library
Optimized gate-level representation
Optimization phase

Figure 3.2: Evolutionary synthesis of logic circuits from scratch.

Although there are some pathological cases of circuits for which BDDs do not scale well, BDDs are known to be an efficient tool for representation and manipulation with digital circuits. In contrast to the SAT-based equivalence checking, the BDD-based equivalence checking enables not only prove the equivalence but also to determine Hamming distance between a candidate circuit and specification. The reader is referred to Section 3.3 of paper [Vas+16b] (Appendix D) for more details regarding to the determining of Hamming distance using BDDs.

The principle of the evolutionary synthesis of logic circuits (from scratch) is depicted in Figure 3.2. The synthesis process is typically broken into two phases. The goal of the first phase is to evolve a fully functional circuit. It means to obtain a circuit that does not violate the behavioral specification. In order to do that, the fitness function needs to measure the distance between a candidate solution and specification. This is achieved by means of a quality metric. The quality is typically expressed in terms of Hamming distance, but some different measure can be introduced. The only condition is that the measure needs to be deterministic and needs to provide a sufficient resolution to prevent evolution to get stuck in a local extreme. As soon as a fully functional circuit is discovered, the circuit is optimized. While the first phase is driven solely by the quality metric, the second phase takes also into account circuit parameters. Note that both phases are conducted at the gate-level representation.

Our experiments confirmed that few milliseconds are required to calculate Hamming distance even for circuits having more than 40 inputs [Vas+14b]. Compared to the well-optimized common CGP, the proposed method achieves the speedup ranging from four to seven orders of magnitude when circuits having from 23 to 41 inputs are considered. As the proof-of-concept, 16 nontrivial circuits (fairly out of the scope of well-optimized standard CGP utilizing simulation-based fitness function) were evolved. Correctly working circuits were reported in twelve cases. In addition to that, it has been shown that the evolution was able to improve the results of conventional synthesis tools. For example, the evolution discovered a 28-input circuit having 57% less gates than the result obtained from the state-of-the-art synthesis tool. An average gate reduction of 48.7% was reported for all evolved circuits [Vas+14b].

3.4 Area-aware logic synthesis

At the beginning of our research, there was a simple goal – to improve scalability of the methods designed during the first fifteen years of evolvable hardware, enable them to handle more complex instances and repeat the success of Miller who demonstrated that evolutionary approaches are capable to provide more compact solutions compared to the state-of-the-art synthesis tools [VJM00].

Originally, the number of gates was the only criterion that was optimized within the evolutionary circuit design community. This cost function arises from the theoretical computer science. The number of gates and logic levels represent two metrics for assessment of the complexity of Boolean functions. We adopted this approach also in our aforementioned work as this represented the only way how to fairly compare our results across existing methods and state-of-the-art synthesis tools.

The problem is that the circuit size does not directly correlate with the cost of this circuit when implemented on a chip. Each gate, in general, requires a different area to be implemented on a chip. A two-input XOR gate, for example, represents the most complex gate which occupies typically two times larger area compared to NAND gate. It does not mean that the results are invalid or worthless for the hardware community. This metric could, however, represent an issue for small or XOR-intesive circuits. In addition to that, the number of gates is hardly applicable in the case when we need to reduce leakage power as discussed in the first chapter.

We proposed a different cost function which applies a more realistic non-uniform measurement of circuit size. The cost of gate is not constant but determined by the area needed to implement a particular gate on a chip. The area is specified relatively to NAND gate. More than twenty benchmark circuits taken from LGSynth93 benchmark suite were optimized by CGP with SAT solver. The experimental results confirmed that CGP can significantly improve the results of state-of-the-art synthesis tools even if a modified cost function is utilized instead of the number of gates. In average, the circuits were improved by 24% in comparison to conventional methods such as SIS and ABC [Vas+11b; Vas+12a].

Chapter 4

Approximate computing

The approaches discussed so far assumed that a circuit fully compliant with the specification must be delivered. In recent years, a new research field was established to investigate how computer systems can be made more energy efficient, faster, and less complex by relaxing the requirement that they are exactly correct. This field, known as *approximate computing*, exploits the fact that many applications are error resilient and the errors in computing are thus either invisible or acceptable. The concept of approximation has intensively been studied, developed and applied not only in computer science, but also in mathematics and engineering disciplines. However, it has never been applied in the areas in which only accurate implementations have traditionally been accepted. Nowadays, the designers intentionally introduce errors into computation to satisfy the never ending requirement for lowering of power dissipation.

As one of the most promising energy-efficient computing paradigms that is able to cope with current challenges of computer engineering, approximate computing has gained a lot of research attention in the past few years. We can identify two main directions in approximate computing: energy-efficient computing with unreliable components and approximation of systems implemented on common platforms [Mit16].

In the first case, the problem is that the exact computing utilizing nanometer transistors provided by recent technology nodes is extremely expensive in terms of energy requirements and reliable behavior. An open question is how to effectively and reliably compute with a huge amount of unreliable components. This concept reflects the fact that after several decades of continual scaling of technology nodes, intrinsic parameters variability of transistors started to negatively influence chips' properties in terms of intra-die or die-to-die variations, fluctuations in power consumption and maximum operational frequency, susceptibility to errors and yield.

The second research direction is motivated by the fact that many applications (typically in the areas of multimedia, graphics, data mining, and big data processing) are inherently *error resilient*. This resilience can be exploited in such a way that the error is exchanged for improvements in power consumption, throughput or implementation cost. After analyzing many applications, Chippa et al. [Chi+13] reported that about 83 % of runtime is spent in computations that can be approximated.

Various approximation techniques have been proposed recently. A good survey of the proposed approaches can be found, for instance, in [Mit16; XMK16]. According to the level of the computer stack where the approximations are conducted, the approaches could be roughly divided into software-level and hardware-level. At the software level, for example, we could selectively ignore certain computations and/or memory accesses that are not critical for obtaining desired quality of result. At the hardware layer, we could either use a less accurate yet more energy-efficient circuit for computation or purposely reduce the supply voltage for certain hardware components to trade-off energy and accuracy. In this thesis, we restrict ourselves to the hardware-level techniques and in particular to approximate circuit design.

As discussed in the first chapter, we could reduce energy consumption of a given circuit by lowering its supply voltage without reducing the corresponding operational frequency. In order to further decrease the energy consumption, one could use so-called overscaling technique for approximate computing. Overscaling means, that we let the supply voltage drop even below a critical point that guarantees reliable operation. Unfortunately, overscaling-induced timing errors occurring on critical paths often lead to large computational errors, unless the circuit is designed in a scalability friendly manner [XMK16]. In addition to that, the energy efficiency gain for such approximation method is relatively small. Consequently, the researches are gradually turning away from overscaling and majority of the recently proposed methods resorts to *functional approximation*.

4.1 Functional approximation

The principle of functional approximation is to implement a slightly different function to the original one provided that the error is acceptable and key system parameters are improved. A typical goal is to obtain a good energy-accuracy trade-off. Note that the functional approximation can be conducted not only at the level of hardware but also at the software layer. An approximate circuit is typically obtained by a heuristic procedure that modifies the original implementation. Typically, the synthesis problem is formulated as finding the approximate design with minimum area under a given error constraint [XMK16]. A logic synthesis approach to the design of combinational circuits that implement approximate versions of the given Boolean function was proposed in [SG10]. The authors proposed a heuristic which identifies min-terms complements that produce an approximate circuit version that has the smallest number of literals for a given error rate threshold. Later, logic synthesis approach for combinational circuits based on Boolean relation minimization was introduced [MGO13]. Other approaches utilize a different strategy. They start with an existing accurate circuit that is subsequently refined to meet the error constraint. For example, SALSA [Ven+12] introduces the quality function which takes the outputs from both the original circuit and approximate circuit and decides if the quality constraints are satisfied. The quality function outputs a single Boolean value. The SALSA algorithm attempts to modify the approximate circuit with the goal of keeping the output of the quality function unchanged. Another method, SASIMI, tries to identify signal pairs in the circuit that exhibit the same value with a high probability, and substitutes one for the other [VRR13]. These substitutions introduce functional approximations. Unused logic can be eliminated from the circuit which results in area and power savings. The common feature of the currently available methods is that they are error-oriented in the sense that all logic optimizations leading to an approximate solution are constrained by a predefined error criterion.

4.2 Current challenges

The key challenge in approximate circuit synthesis is how to effectively represent quality constraints and bring them into the synthesis procedure efficiently. There are basically two issues connected with quality constraints that the researchers are currently facing with. First issue is that there is no suitable error model for a general logic circuits. Unlike arithmetic units where we can apply many arithmetic metrics to quantify the error, for example, there does not exist well-accepted error model for arbitrary circuits. Introducing approximations to general logic could be dangerous in many cases, especially for controllers and clearly it needs to be applied with caution. The error probability, one of the most popular metrics, is hardly applicable in this case. The second issue is the lack of formal methods that are able to efficiently determine quality of a given circuit. In the field of digital system design, the use of SAT solvers has been investigated for more than twenty years and many powerful SAT-based methods have been developed. Nowadays, SAT-based equivalence checking routinely handles complex circuit instances and represents an essential part of every logic synthesis tools. Unfortunately, this approach is hardly applicable for quantifying errors because a binary output is provided only. In order to escape from this problem, the authors typically use simulation – they apply a randomly generated set of test vectors to assess the quality of an approximate circuit. This approach, however, provides no guarantee on the error and make it difficult to predict the behavior of an approximate circuit under different conditions (e.g. when different data-width is used or data with different input distribution are processed). Relaxed equivalence checking (equivalence checking performed up to some bound) thus represents a problem that should be, among others, addressed in near future.

In this brief intro, we left undiscussed other crucial issues associated with the current status of approximate computing, for example, testability, dependability, security, and manufacturability of approximate circuits. For further details, the reader is referred to consult [Mit16].

Chapter 5

Evolutionary synthesis of approximate logic circuits

Synthesis of approximate circuits is in principle an incompletely specified problem. Among others, design of image filters, classifiers or predictors represent typical examples of incompletely specified problems. The common feature of this class of problems is that the target logic circuit implementing the required behavior is unknown a priory. If we cannot specify the problem, however, we cannot easily employ standard synthesis tools to design such a circuit.

The main reasons why EHW has mainly been studied and developed include its ability to (i) provide novel designs hardly reachable by means of conventional methods and (ii) deliver good solutions for problems where the specification is inherently incomplete and any golden solution does not exist. To address the problem of incomplete specification, an artificial specification is typically created. The evolution employs a training set consisting of known input-output pairs that are used to determine the quality of candidate solutions. The goal is to minimize the difference between the specification and circuit response. The evolutionary loop is terminated when a predefined amount of time was exhausted or when required design constrains were met. Maximum acceptable difference is typically employed as a design constrain.

5.1 Early approaches

Interestingly, not only functional approximation but also energy-efficient computing with unreliable components can be traced in the history of evolvable hardware. In the middle nineties, Adrian Thompson evolved a tone discriminator circuit directly in the FPGA chip. The discriminator required significantly less resources than a common conventionally designed solution [TLZ99]. Despite a huge effort, Thompson has never fully understood the evolved design. The evolved discriminator was fully functional, but its robustness was limited. For example, a higher sensitivity to fluctuations in environment such as temperature or power supply voltage was reported. This result, showing an innovative trade-off between the robustness and the amount of resources in the FPGA can be considered as an early approach to approximate circuit design by means of evolutionary algorithms.

The same FPGA platform was employed in another work that addressed the problem of evolutionary design of digital circuits that can approximate real-valued mathematic functions [MT98]. As the authors stated, the aim of this work was not to accurately build mathematical function circuits but to merely explore the degree to which evolved circuits could approximate the desired real signal response in a stable way. An error-based fitness function with randomly generated test vectors was employed to determine approximation quality. This approach would be called functional approximation nowadays.

In 1999, Julian Miller introduced a CGP-based method for finite impulse response filter design [Mil99b]. In this method, candidate filters are composed of elementary logic gates, ignoring thus completely the well-developed techniques based on multiply–and–accumulate structures. Evolved networks of gates are extremely area-efficient (and thus potentially energy efficient) in comparison with conventional filters. However, only partial functionally has been obtained because of the overall simplicity of the logic networks. The evolved circuits are not, in fact, filters. In most cases, they are combinational quasi-linear circuits trained on some data.

Later, a new concept which is currently referred to as the *evolution in materio* has been developed [MD02]. The evolution in materio was designed to automatically create useful functions in a physical system without understanding the principles behind. In order to study and exploit this concept in computer simulation, concept of *messy gates* has been introduced. A messy gate is a gate-like component with added noise. CGP was extended to support noise modeling and then used to evolve small combinational circuits composed of messy gates. The experiments demonstrated that the evolution is able to discover circuits exhibiting implicit fault tolerance. Moreover, surprisingly efficient and robust designs were obtained for small combinational circuits.

Kneiper et al. investigated robustness of EHW-based classifiers [Kni+10]. A classifier system was reported which is able to cope with changing resources at run-time. During optimization, the number of pattern matching elements was modified and its influence on classification accuracy was studied. The performance and accuracy was recognized as sufficient as long as a certain amount of resources is present in the system.

In addition to these examples, there are many other approaches that could be nowadays considered as methods of approximate computing. Despite of some nuances (e.g. suitable error metric needs to be integrated into the fitness function), there is a high chance that the EHW methods will be applicable not only for logic synthesis but also for approximate circuit design.

5.2 Principle of the evolutionary approximation

The first EA-based design method that explicitly addressed the problem of approximate circuit synthesis was proposed in 2013 [Vas+15b] (Appendix C). The principle of the evolutionary synthesis of approximate circuits is illustrated in Figure 5.1. In fact, this method represents a hybrid approach that combines some features of evolutionary optimization with some other features of evolutionary design.

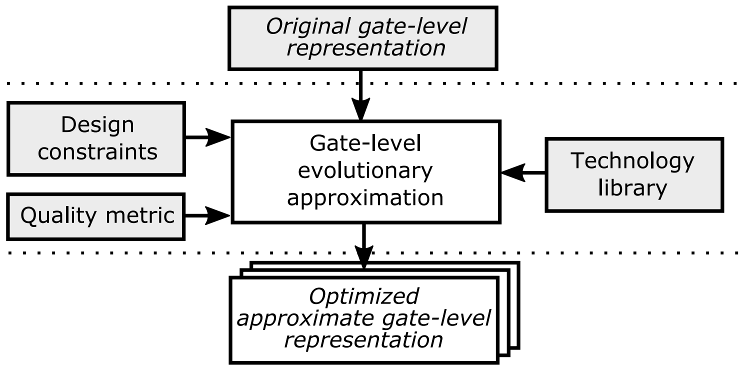

Original gate-level representation
Design constraints
Quality metric
Gate-level evolutionary approximation
Technology library
Optimized approximate gate-level representation

Figure 5.1: Evolutionary synthesis of approximate logic circuits.

Similarly to the evolutionary optimization (see Figure 3.1), this method starts with a fully functional solution that is subsequently approximated and optimized. This scheme is extended by a quality metric representing a typical feature of the evolutionary design from scratch (shown in Figure 3.2). The quality metric (error measure) is employed to quantify how far a given approximation is from the original solution. In addition to that, the importance of technology library increased. Technology library is in fact a mandatory part of the whole system because it is necessary during evolution (at least to some extent) to determine circuit parameters including power dissipation, area and delay. While the standard logic synthesis flow needs to be substantially modified to support approximate circuit design, there is no significant difference between evolutionary synthesis of logic circuits and evolutionary synthesis of approximate logic circuits. In the simplest case, it is sufficient to appropriately modify the fitness function to take the error into account.

The following sections will present three methods to digital circuit approximation that we developed: a) area-oriented method (Section 5.3), b) error-oriented method (Section 5.4) and c) multi-objective method (Section 5.5).

5.3 Area-oriented method

As a proof-of-concept, we first investigated whether EAs in general and CGP in particular are able to approximate various arithmetic circuits, provide resonable tradeoffs and compete with other approximate design methods proposed in literature [SV13]. The proposed method exploits the fact that EA can produce a partially working solution even if sufficient resources for constructing an exact circuit are not available. As power consumption is often highly correlated with occupied resources, we can evolve a partially working circuit using constrained resources and assume that the circuit's power consumption will be reduced. This assumption enables to simplify the design process because we can get rid of technology library. In principle, the evolution could start with a randomly generated initial point (as followed in [SV13]), however, our extensive experimental evaluation demonstrated that it is more efficient to start with an accurate gate-level circuit obtained by a common logic synthesis tool [Vas+15b] (Appendix C). The benefits are not only in improving the quality of evolved circuits, but also in reducing the time of optimization. For more complex problem instances such as 25-input median for example, the results of randomly seeded CGP were far from optimum.

The idea is elaborated as follows. Let $|C|$ be the number of gates of a circuit C (original accurate circuit) that implements a Boolean function that should be approximated. The approximate circuit is synthesized using CGP which can use up to n gates ($n < |C|$) and whose objective is to minimize the error. The parameter n represents a design constrain that needs to be supplied externally by designer. If various other approximations are requested, CGP is executed multiple times with a gradually reduced amount of available gates. The designer thus obtains a set of approximate combinational circuits, each of which typically exhibits different trade-offs between the functionality and the number of gates. The proposed design approach can be considered as an *area-oriented* method because the user can control the used area (and so power consumption) more comfortably than by means of the error-oriented methods. The synthesis starts with C that is subsequently modified by a heuristic procedure in order to obtain the initial design point C' consisting of n gates. The heuristic procedure employed in our paper simply replaces appropriately chosen gates by wires with the aim to reduce the number of gates while keeping the error as low as possible. The obtained circuit C' is then optimized to minimize the error. For a more detailed description, see Section III in [Vas+15b] (Appendix C).

The area-oriented method was employed to approximate single-output benchmark combinational circuits [SV13], small adders (up to 4 bits) [SV13], and small multipliers (up to 4 bits) [Vas+15b] (Appendix C). Two important error metrics were used to measure the quality of the approximate designs compared to the correct designs. *Error rate* was utilized to assess the error of the combinational circuits. This metric counts for the percentage of input assignments which the function value differs for. In the case of arithmetic circuits, *average-case arithmetic error* was employed. The average-case arithmetic error is calculated as the sum of absolute differences in magnitude between the correct and approximate circuit, averaged over all input assignments. As only small circuit instances having up to 20 inputs were considered, the exact error value was determined in both cases. In order to accelerate the computation of quality metric, we adopted an approach that we proposed in [Vas+12b]. This method exploits the parallel simulation of candidate circuits [Sek12] and direct circuit translation to the binary machine code. The main idea is to avoid running a time-consuming CGP interpreter. As the modern CPUs are equipped with instructions enabling to process up to 256-bit vectors in parallel, this method allows to calculate response for all 2^8 input vectors in one pass. Consequently, few microseconds are required on a common processor to determine the error.

Paper [Vas+15b] (Appendix C) shows that we rediscovered by means of CGP one of the first approximate multipliers – 2-bit manually designed multiplier consisting of five gates that was then employed to create larger multipliers [KGE11]. This 2-bit multiplier produces an erroneous output only when both inputs are equal to 3. Then, it returns 7 instead of 9. This form of inaccuracy enables to improve not only area, but also reduce the number of output signals and improve delay when employed in a large multiplier where long carry chains are typically present. It is expectable, however, that better parameters can be achieved when we directly approximate a 4-bit architecture. Unfortunately, such a multiplier contains more than 60 gates which is too complex to be handled manually. By means of CGP, we were able to directly evolve approximate 4-bit multipliers. The obtained approximate designs exhibit significantly better trade-offs even if they are used in 8-bit and 16-bit multipliers [Vas+15b]. The 8-bit multipliers were approximated also in [VRR13], however as the authors utilized a different technology, applied various technology dependent operations such as downsizing of gates, and started with unspecified fully functional multipliers, it is not easy to provide a fair comparison. However, the rough comparison performed in [Vas+15b] indicates that the evolutionary approach is able to deliver better trade-offs.

In addition to that, we investigated the approximation of complex problem instances such as 9-input and 25-input median circuits operating over 8 bits [Vas+15b]. Compared to the arithmetic circuits where we employed a technology library consisting of common two-input logic gates, more complex primitives were used – a minimum and maximum operation, both operating on 8-bit. Each operation requires 66 two-input logic gates to be implemented. The correct 9-input median has 72 inputs and consists of more than 2 500 gates when implemented as a gate-level logic circuit. The 25-input median requires more than 14 500 gates. In order to determine quality of approximate designs, the average-case arithmetic error was used. In contrast to the previous case, the error was determined using randomly generated test set because it is intractable to evaluate all possible input combinations (256^9 and 256^{25} vectors). An interesting finding, at least from the theoretical point of view, was that it seems that solving the 25-median design problem from scratch is impossible for any EA based on direct encoding. Although CGP could utilize up to more than 200 operations, the most complex circuit use half of them (see Figure 10 in [Vas+15b] (Appendix C)). Even if the randomly generated initial design point comprised all gates, most of them were disconnected in the course of evolution, reaching about 100 operations again. From the practical point of view, very encouraging results were discovered. Median circuits are very good examples of circuits for which it makes sense to introduce their approximate versions. The mean error remains relatively low, even if many components are disconnected. Hence significant improvements in energy consumption are obtained. In addition to that, it is worth noting that the methods such as SALSA or SASIMI cannot be competitive in approximation of circuits such as the median because they are working at the bit level only.

The power consumption and delay were calculated at the end of the evolution using SIS. Even if the number of gates is a relative rough measure of power consumption, a general observation is that the power dissipation (and spread of power dissipation) decreases with decreasing the number of available gates. It is fair to say, however, that the spread is relative large especially for small arithmetic circuits implemented using a small number of gates. When we reduce the number of gates of 4-bit multiplier to 50%, for example, the spread in power consumption of various discovered approximate designs is around 40%. The proposed approach is suitable for more complex circuit instances having hundreds of gates. When the circuits shows some degree of complexity, the power dissipation strongly correlates with the number of gates.

5.4 Error-oriented method

Later, we introduced a complementary design approach supporting the error-oriented design scenario [Vas+14a]. At the beginning, the user is supposed to provide an accurate gate-level circuit that ought to be approximated and specify the synthesis goal, i.e. circuit parameters (or additional quality parameters) to be optimized such as area, power consumption or delay. In addition to that, the user defines the target level of error and metrics that will be utilized to guarantee the chosen error level. It means that he or she specifies, for example, that the average-case error magnitude should be less or equal to ε. Then, CGP-based evolutionary approximation is employed to obtain a circuit with the required quality. The goal of evolution is to produce a gate-level implementation showing the required error level ε (or at least as close as possible to ε) that is optimized with respect to the considered circuit parameters.

To achieve this objective, we had to cope with the problem that the evolutionary approximation represents in fact a multi-objective design problem in which the accuracy and power consumption are conflicting design objectives. A straightforward approach to the multi-objective optimization is converting the problem to a single objective one by means of a weight function. The proper setting of weight is not an easy task and usually based on user intuition [Vas+15a]. Another limitation of the weight function lies in the fact that certain Pareto-optimal solutions are not reachable in the case of non-convex objective space. Since it is difficult to detect whether the resulting objective space is non-convex, the weight function has to be applied with caution. As a result of many experiments with various approaches, we introduced a two-stage procedure that utilizes two fitness functions in a common CGP. Our motivation was simple – to avoid a single fitness function combining the objectives using a weight function. The goal of the first phase is to modify the original circuit in order to obtain an approximate circuit showing error as close as possible to the target error level. The fitness function is expressed as the distance between ε and the error of a candidate approximate circuit. It means that the role of the error metric is to guide the evolution through the design search space. After obtaining desired circuit, CGP continues by the second phase. It can minimize the number of gates or other criteria providing that ε is left unchanged. A different fitness function (optimization criterion) is employed in the second phase. The fitness function considers only the circuit parameters specified by the designer. The error level (error metric) servers as a design constrain. Circuits violating this constrain are discarded. Note that the evolutionary approach is constructed in such a way that some small deviances from ε are allowed in both phases otherwise the search could easily stuck in a local extreme.

The proposed method was extensively evaluated in the task of combinational approximate multiplier design [Vas+14a]. The goal was to approximate common 4-bit, 5-bit, 6-bit, 7-bit and 8-bit gate-level multipliers. The largest accurate circuit (8-bit multiplier) consists of 320 gates. We analyzed three scenarios in order to deliver the most practically useful solutions. The aim of the first scenario was to synthesize approximate multipliers showing the required worst-case error and simultaneously having a minimal possible average-case error. The second scenario was similar, however the goal was to obtain a solution which occupies the minimum number of gates for a given worst-case error. The goal of the last scenario was to evolve approximate multipliers which utilize the minimum number of gates for a certain average-case error. Note that a 5% deviance from the required error level was tolerated. This kind of flexibility was introduced also because for some multipliers it is not possible to obtain an implementation which has exactly the specified error level.

Interestingly, the design of an approximate multiplier exhibiting the required error level represents a problem that can be relatively easily solved. The CGP optimization took a few minutes. Nevertheless, the third scenario required two orders of magnitude higher number of generations in contrast with other scenarios. This indicates that the discovering of an approximate circuit with a certain average-case error magnitude represents a more difficult problem than the design of an approximate circuit with a certain worst-case error magnitude. In all three cases, the circuits obtained after the first optimization step exhibit a small improvement of the utilized area. However, a different situation occurs during the second phase of the optimization. While the first scenario did not achieve any consequent power reduction, the remaining scenarios continued in improving of the occupied area. The reason of this behavior is probably caused by the fact that more gates have to be employed to obtain an approximate circuit which minimizes two error metrics simultaneously. An overall conclusion is that the error-oriented approach tends to be less computationally demanding than the resources oriented method.

In contrast to the approximate circuits available in literature, very efficient approximate multipliers were obtained. It was shown, for example, that the proposed design technique is able to generate multipliers that are unreachable using the technique based on combining of small 2-bit approximate multipliers [KGE11]. Our approach delivered circuits showing one order of magnitude better errors for the equivalent power reduction. Comparison with other approaches is a bit problematic due to the missing details in corresponding papers, but let us give at least some examples. The authors of SALSA reported 80% reduction in power consumption for 8-bit multiplier with worst-case error equal to 10% [Ven+12]. In our case, 96% power reduction was obtained for the same error level. Using SASIMI approach, approx. 45% power improvement was achieved for 8-bit multiplier average-case error equal to 0.5% [VRR13]. We discovered an implementation with the same error showing 79% power reduction.

Recent advances in artificial intelligence methods and a huge amount of computing resources available on a single chip have led to a renewed interest in efficient implementations of complex neuromorphic systems based on artificial neural networks (NNs). However, implementing complex convolutional NNs in low power embedded systems requires careful optimization strategies at various levels. In order to address this problem, the error-oriented method has recently been used to design 7-bit and 11-bit multipliers optimized for the usage in NNs [Mra+16] (Appendix G). Our extensive analysis of NNs revealed that it is necessary to design approximate multipliers having one specific feature: the multiplication by zero must be accurate. From the perspective of evolutionary synthesis, this requirement represents a simple constrain that can be easily incorporated into the CGP (as shown in Figure 5.1). Candidate solutions violating this constrain are simply discarded. For more details about the proposed method please consult Section 3 in [Mra+16] (Appendix G). This method helped us to build a large database consisting of more than eight hundred of trade-offs between the accuracy and power consumption. The multiplier were extended using one's complement and applied in the pretrained neural network (all accurate multipliers were replaced with the evolved approximate multiplier). The results were remarkable, the approximate multipliers introduced into NN enabled to reduce the power consumption of convolution sub-circuits of NN by more than 80%. The classification accuracy decreased only by 1.89% for SVHN dataset and 0.36% for MNIST dataset.

5.5 Multi-objective method

Let us conclude this chapter with a brief description of the most advanced method we have developed to address the problem of evolutionary approximation of logic circuits.

As it was discussed in the previous part, approximation is in general a multi-objective design problem. Because we are using conflicting objective functions, there does not exist a single solution that simultaneously optimizes each objective. In fact, there possibly exists a huge amount of so called Pareto optimal solutions [Deb01]. A good approximate circuit design tool should provide a set of solutions which exhibit various trade-offs among key circuit parameters. These solutions should, in an idealized scenario, perfectly match the so-called Pareto optimal front. The available tools such as SALSA, SASIMI and ABACUS solve this problem by multiple executions of approximation engines that are typically constructed as single-objective optimizers. It is clear that when we need to obtain the whole Pareto front, this approach may become very time consuming, especially when the Pareto dominant solutions are not distributed evenly in the design space.

The multi-objective evolutionary optimization represents a well studied problem whose roots can be traced to mid eighties. Since that, many powerful multi-objective evolutionary algorithms (MOEAs) have been introduced, for example, Vector Evaluated Genetic Algorithm (VEGA), Strength Pareto Evolutionary Algorithm (SPEA2) and non-dominated sorting genetic algorithm (NSGA-II) [Deb+02]. Most of them are based on the idea of *Pareto dominance*. Contrasted to the single-objective optimization algorithms, they internally sort individuals according to the dominance relation, build archives of non-dominating solutions, and ensure population diversity to avoid converging to a single solution. The goal of MOEAs is to precisely approximate the whole Pareto-optimal front and obtain various diverse non-dominate solutions in a single run of an optimizer. In the context of evolutionary circuit design, multi-objective methods were utilized for digital as well as analogue designs [Mil11; EMG05].

Instead of running evolution for every possible number of gates or error (as we have seen in the previous two sections), we combined CGP encoding with NSGA-II, one of the most efficient multi-objective evolutionary algorithms. The $1 + \lambda$ search strategy utilized in CGP is replaced by procedures of NSGA-II which implement non-dominated sorting of the population consisting of λ' individuals and diversity preservation mechanisms [Vas+15a; HMV16]. While λ is typically low in CGP, λ' needs to be much higher as it has impact on the number of Pareto dominant solutions obtained at the end. In the ideal situation, λ' different trade-offs should be produced. Note that the original non-dominated sorting algorithm was adapted to allow the evolution to continue in searching for better solutions within the neutral space which may prevent early stagnation of the algorithm. When all objectives of the fitness score of a parent and its offspring remain unchanged, the offspring is classed as dominating the parent, and is therefore ranked higher than the parent. The maximum allowed error which the designer is going to observe and accept in the resulting Pareto fronts acts as a constraint. One fitness function is constructed for each objective. In order to simplify the problem, all the fitness functions are designed to be minimized.

As a proof-of-concept, we employed the proposed multi-objective CGP in the task of 4-bit and 8-bit adder and multiplier approximation [Vas+15a]. Three objectives were considered: error, area and delay. Similarly to the previous experiments, it was assumed that the power consumption highly correlates with the area which seems to be valid at least for the considered 350 nm technology nodes. The area and delay were calculated using the parameters defined in the Liberty timing file. This file defines basic characteristic of cells, i.e. logic gates, flip-flops, latches, and buffers, pertaining to a particular technology node. Among others, functional definition, timing, power, and noise information are provided as a result of characterization process.

The proposed multi-objective CGP was compared with single-objective CGP. The single-objective CGP works exactly as the error-oriented approach; the only difference is that it utilizes fitness function with weighted objectives in the second stage. Both methods were initialized with accurate gate-level circuits. To construct a Pareto front, single objective CGP was executed several times; one run for one error level. According to our expectations, it was shown that the single-objective (SO) approach outperforms the multi-objective (MO) approach when more time is available. We believe that the SO approach exploits the fact that the error is fixed and the overall effort can be put into minimizing the area and delay. On the other hand, MO has to cover the whole Pareto front and the available time seems to be insufficient to compete with SO. If the number of evaluations is low, MO approach produces substantially better results. It is worth noting, however, that we spend lot of time by tuning the weights utilized in SO's fitness function. We identified that the weight of the area is extremely important and its unsuitable setting can substantially influence the quality of the resulting Pareto front. More detailed evaluation and discussion related to the obtained results can be found in [Vas+15a].

To be honest, there is no definite winner. Both methods have specific strengths and weaknesses depending on the goal of synthesis. There are two possible scenarios in practice. When the target error level is known by designer a priory, the error-oriented single-objective approach is undoubtedly the method of the first choice. In this scenario, a single highly optimized solution provided by CGP is sufficient. On the other hand, when the designer does not exactly know the target error, it is beneficial to provide more trade-offs, i.e. execute a single run of the multi-objective CGP. It also seems reasonable to combine both approaches together. At the beginning, a multi-objective approach is employed to quickly explore the design space. The designer then selects one or more design points that are further optimized by means of the single objective optimizer.

The multi-objective method was further improved and applied to the evolution of 8-bit adders and multipliers [HMV16]. In contrast with our previous study, 180 nm technology nodes were utilized. Our goal was to obtain the best possible trade-offs. Hence, we allowed CGP to utilize half and full adders as building blocks that are available in almost every technology library. This step was motivated by the fact that these high-level cells are highly optimized directly at transistor level. Consequently, they occupy lower area and exhibit better energy-efficiency compared to their corresponding gate-level implementations. Contrasted to the [Vas+15a], power dissipation was directly utilized as one of the design objectives. In addition to the power dissipation, area and delay were considered. We obtained more than four hundreds of Pareto optimal implementations for adders as well as multipliers. The obtained results confirmed that significant savings can be achieved when relaxing the requirement of perfect functionality.

As we have mentioned in the introductory part, chip manufacturers need to take the route of specialization to enable further scaling. However, specialization is connected with the necessity to make a shift in the paradigm how the chips are developed. It was argued that instead of starting from scratch each time, the designers should create new devices by combining large chunks of existing circuitry that have known functionality [Wal16]. The main motivation is that design of circuits is in general very time consuming and costly process. It seems that this is also the case of the area of approximate computing in which many different architectures were proposed in recent years. The authors of papers dealing with approximate circuit design spent many hours by approximating various circuits, but the resulting architectures are typically not available for download. The results are hardly comparable and hardly applicable from a practical point of view. Except of a single library of few 16-bit adders, there does not exist an universal library containing ready-to-use approximate components. The problem is that a few approximate implementations are typically created from the original circuit and reported in literature. It is probably caused by the fact that the process of approximation is typically a computationally demanding task especially when simple heuristics are applied. Interestingly, our multi-objective approach enabled us to obtain a rich library of adders and multipliers showing different errors and parameters in a reasonable time. All the 400 Pareto optimal circuits were discovered in less than 300 CPU hours (a cluster of Intel Xeon CPUs was employed). This library can be utilized in future applications of approximate computing.

Chapter 6

Exact quality metrics based on binary decision diagrams

So far we have discussed the approximation methods that evaluate the candidate solutions by applying a set of input vectors and measuring the error of the output vectors with respect to an exact solution. This approach is not, however, applicable when approximating complex circuits. When a subset of all possible input vectors is adopted, the error is only estimated. If the exact error of the approximation has to be determined, formal *relaxed* equivalence checking is requested, stressing the fact that the considered systems will be checked to be equal up to some bound w.r.t. a suitably chosen distance metric. This research area is rather unexplored as almost all formal approaches have been developed for exact equivalence checking [Hol+16]. Checking the worst error can be based on satisfiability (SAT) solving as outlined in [Ven+11]. However, while violating the worst error can be detected, no efficient method capable of establishing, for example, the average error using a SAT solver has been proposed up to now. Hence, binary decision diagrams seem to be the only viable option, at least for this moment.

To the best of our knowledge, there are three papers in which the authors employed BDDs to determine some error metric. The average-case arithmetic error, worst-case arithmetic error and error rate were discussed in [Soe+16] and applied to approximate six benchmark circuits. Later, the same approach for error rate was described in [YC16] and utilized to approximate 8-bit multipliers and various speculative and accuracy configurable 64-bit adders. Finally, the bit-flip error was introduced in [Cha+16] and used for approximation of 8-bit and 16-bit circuits represented using AIG.

Decision diagrams, and especially Binary Decision Diagrams (BDDs), are the most frequently used data structure for representation and manipulation of Boolean functions in the area of digital circuit design. They are driven by the Shannon's expansion to recursively decompose a Boolean function into cofactors until the constant logic values are encountered. On a more abstract level, BDDs can be considered as a compact representation of sets or relations. For BDD's definition, see [DB13] or Section 3 of paper [Vas+16b] (Appendix D). Even if the concept of BDDs dates back to the 1950s, it began attracting the attention of many researchers in late eighties with the work of Randal Bryant who demonstrated how to exploit the full potential of this data structure [HS96]. Since 1986, when BDDs were unknown in logic synthesis, BDDs have penetrated virtually every subfield in the areas of synthesis and verification. Bryant extended the concept of BDDs by introducing some restrictions (a fixed variable ordering) and efficient manipulation algorithms resulting in a canonical form known as Reduced Ordered Binary Decision Diagrams (ROBDDs). This extension significantly simplified otherwise expensive operations such as testing of equivalence or satisfiability of logic circuits. This has led to significant breakthroughs in circuit optimization, testing, and equivalence checking of combinational as well as sequential circuits.

Although the ROBDDs offer an efficient way of representing Boolean functions and provide a tool for solving many practical problems in digital circuit design, it is fair to say that there are situations in which BDDs perform unsatisfactory. It is the requirement of canonicity which makes BDDs inefficient in representing certain classes of functions. For example, multipliers are known for their exponential memory requirements for any variable ordering. It was shown in [Bry91] that the BDD for the multiplier of two n-bit numbers has at least $2^{n/8}$ nodes. It is also a well known fact that the size of BDD (i.e. the number of non-terminal nodes) for a given function is very sensitive to the chosen variable order. Depending on the actual variable order, there are Boolean functions for which the size of the ROBDD can be either linear or exponential in the number of nodes [EFD00]. For this reason, several extensions of BDDs have been suggested.

One of the advantages of ROBDDs is the possibility to efficiently perform many of the operations needed for the manipulation of Boolean functions. The synthesis in general, and Boolean operations in particular, probably represent the most important operations since they can be used to construct ROBDDs. In order to treat the synthesis in a unified way, the so-called If-Then-Else (ITE) operator was introduced. The implementation of the synthesis operator depends on a particular BDD package. For example, Buddy or CUDD offer not only ITE operator as function $ite(f, g, h)$ which takes three ROBDDs as its arguments, but also a ternary function $apply(\Omega, a, b)$ which is optimized for binary operations. This function takes a binary operator Ω and two ROBDDs a and b as arguments and returns a ROBDD corresponding with the result of $a\ \Omega\ b$.

The ROBDDs also enable to efficiently implement operations for examining the set of satisfying truth assignments $sat(f)$ of a given function f. It means such assignments $a \in sat(f)$ that evaluate $f(a)$ to true. There are three algorithms defined on top of ROBDDs: *SATone*, *SATcount* and *AllSAT*. The aim of the first operation, *SATone*, is to find an input assignment a for which $f(a) = 1$ or inform that no such assignment exists. As it is sufficient to consider a single path from a terminal node to the root node representing f, a satisfying assignment can be easily computed in linear time with respect to the number of BDD variables. The second algorithm, *SATcount*, computes the size of $sat(f)$. This can be done in linear time with respect to the number of BDD nodes. Finally, the *AllSAT*, finds all satisfying truth-assignments leaving out irrelevant variables from the ordering. While the BDD could be efficiently traversed, the result of AllSAT operation can be exponentially large, so the running time exhibits exponential dependency on the number of BDD nodes. Note that the equivalence test of two functions f and g can be done even in constant time because almost every BDD package implements node sharing. Hence it is sufficient to check whether pointers for f and g lead to the same node.

6.1 Hamming distance

In the context of evolutionary circuit design, the fitness function based on the Hamming distance computed using BDDs was firstly introduced in [Vas+14b]. Later, we applied a similar method in the context of approximate computing where we dealt with the approximation of general logic [Vas+16b] (Appendix D). As most error metrics are based on arithmetic errors, we suppose in this work that no additional information is usually available to establish a suitable error metric for general logic. Hence we proposed to express the error of approximation in terms of average Hamming distance between the output values produced by an approximate circuit and the accurate circuit. Introducing approximations to general logic could be dangerous in many cases (e.g. for controllers), but there is still an important class of circuits in which the error can safely be exchanged for energy reduction. Among others, various pattern matching circuits and complex encoders represent instances that can be safely approximated in many real applications. In addition to that, approximate computing seems to be a viable approach how to deal with the reduced reliability of digital circuits implemented using nanometer technology nodes. For example, in [San+16], the authors demonstrated how the approximate circuits can reduce overhead of dependable systems based on triple modular redundancy. The main idea is to replace the exact modules with slightly different approximate circuits provided that the resulting system will be still able to detect or correct errors. In order to guarantee such a behavior, the logic function implemented by approximate circuits need to overlap with the original circuit to some extent. The degree of overlap (i.e. the quality of an approximate circuit) was measured using Hamming distance.

The Hamming distance can be obtained by converting the approximate circuit and the specification to corresponding ROBDD and calling suitable operators over ROBDD. Let us briefly discuss this problem as it will help us to understand the metrics discussed in the next sections.

Each combinational logic circuit with n inputs and m outputs computes a completely-specified multiple-output Boolean function $\mathcal{F} : \mathbb{B}^n \to \mathbb{B}^m$, $\mathbb{B} = \{0, 1\}$, that maps n-input Boolean vector $x = \langle x_0, \dots, x_{n-1} \rangle$ to an m-output Boolean vector $y = \langle y_0, \dots, y_{m-1} \rangle$. Let $f : \mathbb{B}^n \to \mathbb{B}^m$ be the specification that describes correct functionality and $\hat{f} : \mathbb{B}^n \to \mathbb{B}^m$ be an approximate function, both implemented by two combinational circuits, namely F and $\hat{\text{F}}$. Using this notion, the average-case Hamming distance HD_{avg} between F and $\hat{\text{F}}$ can be expressed as follows:

$$HD_{avg}(f, \hat{f}) = \frac{1}{2^n} \sum_{x \in B^n} 1's(f(x) \oplus \hat{f}(x)), \tag{6.1}$$

This definition is based on the fact that the Boolean function $f(x) \oplus \hat{f}(x)$ returns a nonzero value if and only if the approximate circuit $\hat{\mathrm{F}}$ provides a response that does not match the response given by the specification. If we sum the number of bits set to 1 over all input combinations (i.e. $x \in \mathbb{B}^n$), we obtain the Hamming distance. The number of bits different from the zero is defined as Hamming weight and denoted as $1's$.

As evident, the formula requires to enumerate all input assignments. This is feasible, however, only for small circuits as the number of input assignments grows exponentially with the increasing number of inputs (n). Fortunately, we are able to reformulate Equation 6.1 as follows:

$$\begin{aligned} HD_{avg}(f,\hat{f}) &= \frac{1}{2^n}\sum_{x\in B^n} 1's(f(x)\oplus\hat{f}(x)) = \frac{1}{2^n}\sum_{x\in B^n}\Big(\sum_{0\le i<m} f_i(x)\oplus\hat{f}_i(x)\Big) \\ &= \frac{1}{2^n}\sum_{0\le i<m}\Big(\sum_{x\in B^n} f_i(x)\oplus\hat{f}_i(x)\Big) = \frac{1}{2^n}\sum_{0\le i<m} SATcount(f_i\oplus\hat{f}_i). \end{aligned}$$

We have employed the fact that each Boolean function with m outputs can be represented by m Boolean functions. In addition to that, we followed the definition of SATcount operation. From the practical point of view, it means that we have to build m ROBDDs for the logic expression $f \oplus \hat{f}$, call SATcount operation for each ROBDD to determine the number of assignments that evaluate $f_i \oplus \hat{f}_i$ to one and finally sum the obtained results. The resulting value is equal to the Hamming distance. Since many of the available BDD packages implement node sharing, it is assumed that this operation will be executed quickly for many circuits relevant to practice as its complexity is linear with respect to the number of BDD nodes.

The BDD-based quality metric was evaluated using 16 benchmark circuits which are difficult for the previous evolutionary approximation methods, because they have too many primary inputs (27 – 50 inputs) and gates (151 – 868 gates) [Vas+16b] (Appendix D). As far as we know, this was the first paper that addressed the problem of approximation of non-arithmetic circuits in the context of approximate computing. The problem was formulated as a multi-objective optimization problem in which Pareto fronts showing trade offs between the error, area and delay were sought. In order to construct Pareto front, we follow the approach described in Section 5.4 in which a single-objective CGP (utilizing a linear aggregation of objectives) is executed multiple times with different target errors. Among others, we measured the time needed to calculate the Hamming distance between two circuits with respect to a given error. In most cases, the time was less than a few milliseconds. There were two cases (partly arithmetic circuits) in which few hundreds or even thousands of milliseconds were required to determine the Hamming distance. When we compared the mean runtime in the first and second stage, we discovered that more evaluations per second can be performed in the second stage of the optimization. The reason is that BDDs are in average smaller than in the first stage. The BDD-based approach enabled us to achieve speedups in several orders of magnitude in comparison with the simulation-based approach.

There is another quality metric that can be determined in a similar way – the *error rate*. The error rate is defined as the percentage of input vectors for which the approximate output differs from the original one. It means that the output is classified as invalid if at least one bit is different. It means that we do not need to inspect each output bit separately. Instead, we combine them using logical disjunction. The error rate can be defined as follows:

$$ER(f,\hat{f}) = \frac{1}{2^n}\sum_{x\in B^n}\left(\bigvee_{0\leq i<m} f_i(x)\oplus\hat{f}_i(x)\right) = \frac{1}{2^n}SATcount(\bigvee_{0\leq i<m} f_i(x)\oplus\hat{f}_i(x))$$

The advantage of this metric is that it is general and can be applied to arithmetic as well as logic circuits. Hence, many authors utilize the error rate to assess the quality of approximation. Unfortunately, it seems to be difficult to interpret the results in practice. For example, there can exist a an implementation slightly modifying one half of the output values, but still providing good performance if used, for example, in image filtering (see e.g. [Vas+16a]). On the other hand, there can exist circuits providing invalid output for few input assignments, but the error magnitude can be enormous. Hence, we are trying to avoid direct optimization for error rate in our work. Note that the same procedure for error rate, as discussed herein, was introduced in [Soe+16] recently.

6.2 Average-case arithmetic error

The design of low-power variants of key arithmetic circuits such as adders and multipliers represents an intensively studied problem not only in context of approximate computing. The basic arithmetic operations are favorite targets of many researchers because they are widely used in digital signal processing. Hence a small improvement of the energy efficiency of these arithmetic blocks may yield a great power savings at the system level. It is therefore no surprise that the researchers try to investigate how to exactly compute various metrics in order to approximate large arithmetic blocks. Recently, a new BDD-based method for arithmetic worst-case and average-case error analysis was introduced [Soe+16]. Independently on that, we designed an alternative approach how to determine average-case arithmetic error. The main advantage of our method is that it does not involve to construct characteristic function as it is required in [Soe+16].

The average-case error is defined as the sum of absolute differences in magnitude between the original and approximate circuits, averaged over all inputs. The average-case error can be expressed using the notion introduced in the previous section as follows:

$$AE_{avg}(f,\hat{f}) = \frac{1}{2^n}\sum_{x\in\mathbb{B}^n}|nat(f(x)) - nat(\hat{f}(x))| = \frac{1}{2^n}\sum_{x\in\mathbb{B}^n}D(x), \tag{6.2}$$

where $nat(x)$ represents a function $nat : \mathbb{B}^m \to \mathbb{Z}$ returning a decimal value of the m-bit binary vector x. We will consider natural binary representation, i.e. $nat(x) = \sum_{0\leq i<m} x_i \cdot 2^i$. To simplify the next notion, let $D(x)$ denote the absolute difference. It was argued in

[Soe+16] that this formula does not enable to construct an efficient algorithm because it would be necessary to traverse all 2^n rows of the truth table. We demonstrate, however, how to reformulate this problem to be directly solved using BDDs.

It is necessary to realize that the value of $D(x)$ is bounded by interval $[0, 2^m)$ and can be thus represented using m-bit vector. From the practical point of view, $D(x)$ can be easily obtained by a circuit consisting of one two's complement subtractor followed by a circuit which determines the absolute value. As the output of $D(X)$ is a natural number, we can substitute $D(X)$ for its binary expansion $D(x) = \sum_{0 \leq i < m} d_i(x) \cdot 2^i$. Based on this idea, we can modify Equation 6.2 as follows:

$$\begin{aligned} AE_{avg}(f, \hat{f}) &= \frac{1}{2^n} \sum_{x \in \mathbb{B}^n} D(x) = \frac{1}{2^n} \sum_{x \in \mathbb{B}^n} \Big(\sum_{0 \leq i < m} d_i(x) \cdot 2^i \Big) \\ &= \frac{1}{2^n} \sum_{0 \leq i < m} \Big(2^i \sum_{x \in \mathbb{B}^n} d_i(x) \Big) = \sum_{0 \leq i < m} 2^{i-n} \cdot SATcount(d_i) \end{aligned}$$

To summarize this approach, we need to create a virtual circuit that is subsequently represented using ROBDD. The virtual circuit takes the input x and produces $D(x)$. The average-case arithmetic error can be then obtained by m calls of SATcount operation, one per each bit of D. Subtraction in virtual circuit can be calculated using m full-adders with first carry-in set to 1 and inverting each bit of the subtrahend. The absolute value can be computed using one m-bit subtractor and $m - 1$ XOR gates [And05].

This BDD-based metrics was applied in our initial experiments devoted to an efficient implementation of Discrete Cosine Transform blocks employed in video compression based on the High Efficiency Video Coding standard. In commonly-optimized DCT blocks, multiplication operations are replaced with additions, subtractions and shifts to reduce power consumption. As video compression is, in principle, an error resilient application, DCT can further be approximated. Our goal was to replace the accurate adders and subtractors with their approximate versions. For 16-bit adders (i.e. circuits with 32 inputs), the achieved speedup factor was greater than two hundreds compared to the highly optimized parallel simulation. Considering the multipliers, the parallel simulation provides more than five times better runtime even for 12-bit multiplier. Unfortunately, BDDs do not offer any way to efficiently handle multipliers.

6.3 Problem-specific quality metrics

In mid fifties, the concept of comparator networks was introduced [Knu98]. A comparator network is an abstract structure consisting of a sequence of elementary operations denoted as compare-and-swap operations swapping the values on the wires if they are not in a desired order. A comparator network that sorts all input elements is called sorting network. The key feature of sorting networks is that the sequence of comparisons is defined in advance, regardless of the outcome of previous comparisons. This independence of comparison sequences is useful for efficient execution in software as well as for parallel implementation in hardware. Each sorting network can be understood as a structure that computes n quantiles in parallel. As a special case of sorting network, median networks determining the median of the input data can be constructed. For more details about sorting and median networks please refer to the [Vas+16a] (Appendix E) and [MV16] (Appendix F). The following paragraphs deals with approximate sorting and median networks.

The sorting networks are constructed in such a way that the sorting is data independent process. The common problem of the generic metrics such as average-case arithmetic error is that they do not reflect the quality of the sorting process because they are data dependent. In order to investigate the impact of the approximations on the quality of obtained results, regardless of the values of the input items, we introduced a new problem-specific metric. It is guaranteed by construction that each comparison network produces a permutation of the input sequence. It means that there exists one to one mapping between the values obtained at the output of comparison network and the values at the input, so no new value can arise during the exchanging performed by compare-and-swap elements. Formally, $C : \pi(x_1, \ldots, x_n) \rightarrow \pi(x_1, \ldots, x_n)$. Hence, every approximate sorting network must produce a partially ordered output for at least one input sequence. To model the error introduced by the approximations, we can measure the distance between the rank of the returned element and rank given by the specification. This metric was denoted *distance error* [Vas+16a]. Two additional metrics were inferred from the distance error: *average-case distance error* defined as the sum of error distances averaged over all input combinations producing an invalid output value and *worst-case distance error* defined as the maximal distance error calculated over all input combinations.

In general, there exist 2^{wn} different input sequences that can be processed by an n-input comparator network operating at w-bits [Vas+16a]. We demonstrated that it is sufficient to reduce the number of the possible input combinations to $n!$ carefully chosen sequences to exactly prove the validity of a sorting network and determine the distance error. This claim was based on the existence of so called permutation principle that we introduced and formally proved in [Vas+16a] (Appendix E). There exist infinite number of sequences, but the most convenient sequence can be obtained by permuting the vector $\langle -\lfloor n/2 \rfloor, \ldots, \lfloor n/2 \rfloor \rangle$. Then, the output value is equals to the distance error in case of median networks. Although this result allows us to exactly evaluate the quality of approximate comparator networks, it is intractable for instances with more than $n = 12$ input elements. Hence, we sought a different approach.

Literature shows that there is deep and complex theory behind sorting networks. The invention of zero-one principle was probably one of the most important breakthroughs in this domain. The zero-one principle states that if a sorting network with n inputs sorts all 2^n input sequences of 0's and 1's into a nondecreasing order, it will sort any arbitrary sequence of n elements into a nondecreasing order [Knu98]. This claim substantially simplifies the problem of verification of sorting networks. In order to verify that a comparison network is a sorting network, it is sufficient to replace each compare-and-swap element with AND and

OR operations and use a SAT solver to check that the outputs are sorted. Based on zero-one principle, we were able to construct an algorithm which is able to efficiently determine the worst-case error using BDDs, but we spend a lot of time by finding a way how to exactly determine the error distribution. The main problem is that we are not able to distinguish which value comes from what input (there are only two values – 0's and 1's). For a median network with a single output, we have problem to determine even the error rate. After many experiments, we proposed an algorithm that is able exactly determine the mentioned error metrics using BDDs. The main idea is outlined in [MV16] (Appendix F). It is based on the fact, that we are able to formulate our problem as Pseudo-Boolean Constraint Satisfaction Problem (CSP) that can be efficiently solved using BDDs. As a result, we are able to obtain true error distribution even for large comparator networks. Few seconds are required to compute the error distribution for $n = 256$ input median networks.

The proposed method was used to approximate sorting networks having 16, 256, 512 and 1024 inputs [MV16] (Appendix F). Experimental evaluation confirmed that sorting (median) networks are highly error resilient. For example, 20% reduction in power consumption can be achieved by introducing a small error in 256-input sorting network. The difference in rank is proved to be no worse than 2 for more than 99% of input combinations. In the remaining cases, it is guaranteed that the worst-case difference is not worse than six ranks.

Chapter 7

Conclusions and future directions

Logic synthesis and optimization is one of the most extensively and intensively studied problems in computer-aided design simply as it represents a fundamental task of many EDA tools. In this thesis, we approached this problem from a different, unconventional, perspective. We employed artificial intelligence to synthesize and optimize logic circuits. In particular, we adopted Cartesian Genetic Programming, representing probably one of the most efficient techniques developed within the field of evolvable hardware. Our main motivation for the research reported in this thesis was exploring the ways how to overcome the main problem of evolutionary computation – bad scalability – that prevents EAs to handle complex instances.

Very pessimistic future for EHW-based digital circuit synthesis was predicted in 2006 [GT06]. However, we have developed an approach easily overcoming the previous empirical limitation of evolutionary design represented by a digital circuit having about twenty inputs and up to hundred gates. Since that, we developed a robust tool which is able to handle circuits having hundreds of inputs and thousands of gates [Vas15] (Appendix B). Contrasted to the various FPGA-based or even GPU-based accelerators proposed to accelerate the simulation of logic circuits, the speedup factor in several orders of magnitude was achieved by adopting the state-of-the-art formal approaches. Nowadays, we are able not only to optimize complex circuits but also design them from scratch [Vas+14b]. Albeit the usage of the design from scratch is rather limited, it has been mentioned as one of the targets of the pioneers of evolvable hardware field.

It has been known for many years that the EA-based synthesis is able to produce results that are unreachable by conventional logic synthesis techniques [VJM00]. On the other hand, there was no evidence that this claim is valid also for complex problems. There were some indications related to the inefficiency of logic synthesis tools even in the hardware community. It was shown, for example, that the state-of-the-art synthesis tools perform poorly for a certain set of artificially created benchmark circuits with known optimal implementation [CM07]. In our experiments with complex circuits, we confirmed the inherent inefficiency of conventional synthesis manifesting itself in producing sub-optimal solutions. What is worse, this inefficiency is not related to a specific class of synthetic benchmarks but it occurs across a comprehensive set of benchmarks. We believe that this comes mainly from the fact that the conventional algorithms rely on local transformations and that there is a bias in AIG representation.

There is no doubt that the evolutionary approaches are able to provide results that have never been reported in the literature dealing with conventional logic synthesis. As many others, even our experiments demonstrated this fact. It is fair to admit, however, that the attitude of hardware community to the evolutionary techniques seems to be a rather skeptical. The evolutionary design of digital circuits is sometimes criticized due to its inherently non-deterministic nature. To understand this attitude, it is sufficient to realize how the conventional synthesis tools are implemented. Virtually all tools are based on applying of deterministic transformations in an iterative manner. Despite the fact that EAs are routinely used to solve real problems in various domains, the hardware community feel that the random nature of EAs cannot provide any guarantee that a circuit of some quality will be obtained when e.g. one hundred iterations are employed. One of the possibilities, how to reduce or even break this mental barrier is to embed the evolutionary-based logic optimizer into widely respected academia synthesis tools such as ABC or Yosys.

Interestingly enough, the same community seems to accept EAs when employed in the approximate synthesis scenario. There are probably two reasonable explanations. While there are about fifty years of history of logic synthesis, the conventional synthesis tools have never been constructed to perform the synthesis of approximate circuits. In addition to that, no golden design exists for an approximate circuit. Because of the nature of approximate circuits (in fact, partially working circuits are sought) and principles of evolutionary circuit design (evolutionary-based improving of partially working circuits), a search based method such as EA seems to be the approach of the first choice. The second explanation is that the approximate computing is strictly driven by motivation to create energy-efficient systems. As the problem is inherently multi-objective and more difficult than conventional synthesis, randomized global search based heuristics could be acceptable.

When we started to investigate the problem of approximate synthesis of logic circuits in 2013, there was prevalence of manually created results. Since then the number of papers presented each year on major hardware-related conferences sharply increased. In addition to that, various events specific to the approximate computing such as WAPCO (Workshop On Approximate Computing), WAX (Workshop on Approximate Computing Across the Stack) or AC Workshop (Workshop on Approximate Computing) have emerged. Nowadays, several automatic design methods for approximate synthesis are available. Compared to the traditional logic synthesis, there has not yet been established a broadly recognized set of benchmark circuits synthesized to various technology nodes. Consequently, the proposed methods are only rarely compared against competitive approximation methods. Typically it is hard or even impossible to perform such evaluation because either implementation of the resulting (approximate) circuit is not available, or parameters (implementation) of the original (accurate) circuit are unknown. Lack of formal methods of relaxed equivalence checking represents another key issue that should be addressed in near future. As a temporary solution, a simulation-based approach is typically applied. It is typically unclear, however, if a given number of test vectors used to evaluate approximate circuits is sufficient for obtaining a trustworthy error quantification. Our experience in the evolutionary synthesis of logic circuits points out on the fact that if there is even a small degree of uncertainty in specification, usually an unsatisfactory result is obtained.

Approximate computing is a research area that offers a great opportunity for evolutionary computing community which has a rich experience with single- and multi-objective optimization. Considering additional optimization criteria thus represents a well-known problem from the perspective of EAs. In our research and in this thesis, we focused only on synthesis of approximate gate-level circuits. However, the approximation can be conducted not only at the level of gates but also on different levels. The circuits can be synthesized, for example, at function level, i.e. from high-level functions such as adders, subtracters or even more complex blocks. At the lowest level of abstraction, approximate synthesis for post-CMOS technology nodes represent a possible directions for future research. In addition to that, the evolutionary approaches can be employed to approximate computer architectures, memories or even software implementations.

Our work on presented publications revealed a number of possible directions for future research and development. Let us mention two of them. The main handicap of the evolutionary approaches is that they are time consuming and generally not scalable when compared with methods of conventional logic synthesis. Very encouraging results were obtained in [Vas15] (Appendix B), especially when we consider the fact that the runtime of the optimization process was 15 minutes. At the same time, however, we revealed an enormous inefficiency of the evolutionary approach itself. The ratio between the number of acceptable functionally equivalent candidate solutions and invalid solutions violating specification was worse than 1:180 in average. It means that approximately 99.5% of the whole runtime is wasted by generating and evaluating invalid candidate circuits that are lying beyond the desired space of potential solutions. Hence, introducing more domain knowledge in EA and utilizing more advanced evolutionary operators seem to be a viable approach how to enable evolvable hardware field to deal with this inherent inefficiency and cope with relentlessly increasing complexity of digital circuits. The second problem is related to the evolutionary mechanisms responsible for the success in logic synthesis. It is well known fact that the nature of a fitness landscape has a strong relationship with the effectiveness of the evolutionary search. Interestingly, only a little is known about the phenotypic search space of digital circuits despite the fact that every search algorithm provides implicit assumptions about the search space [HT11]. Even less is known for approximate circuits.

Bibliography

[And05] S. E. Anderson. *Bit Twiddling Hacks*. http://graphics.stanford.edu/~seander/bithacks.html. Accessed Oct, 2016. 2005.

[Bry91] R. E. Bryant. “On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication”. In: *IEEE Transactions on Computers* 40.2 (Feb. 1991), pp. 205–213.

[CCA98] C. C. A. Coello, A. D. Christiansen, and A. H. Aguirre. “Automated Design of Combinational Logic Circuits by Genetic Algorithms”. In: *Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Norwich, U.K., 1997*. Vienna: Springer Vienna, 1998, pp. 333–336.

[Cha+16] A. Chandrasekharan et al. “Approximation-aware Rewriting of AIGs for Error Tolerant Applications”. In: *35th International Conference On Computer Aided Design (ICCAD)*. Austin, TX, US, 2016. to appear in.

[Chi+13] V. K. Chippa et al. “Analysis and characterization of inherent application resilience for approximate computing”. In: *The 50th Annual Design Automation Conference 2013, DAC'13*. ACM, 2013, pp. 1–9.

[CM07] J. Cong and K. Minkovich. “Optimality Study of Logic Synthesis for LUT-Based FPGAs”. In: *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems* 26.2 (2007), pp. 230–239.

[DB13] R. Drechsler and B. Becker. *Binary Decision Diagrams: Theory and Implementation*. Springer US, 2013.

[Deb+02] K. Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE Trans. Evol. Comput.* 6.2 (2002), pp. 182–197.

[Deb01] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, 2001.

[EFD00] R. Ebendt, G. Fey, and R. Drechsler. *Advanced BDD Optimization*. Berlin: Springer, 2000.

[EMG05] T. Eeckelaert, T. McConaghy, and G. Gielen. “Efficient multiobjective synthesis of analog circuits using hierarchical Pareto-optimal performance hypersurfaces”. In: *Design, Automation and Test in Europe, DATE 2005*. IEEE, 2005, pp. 1070–1075.

[FS08] P. Fiser and J. Schmidt. “Small but Nasty Logic Synthesis Examples”. In: *Proc. 8th Int. Workshop on Boolean Problems*. 2008, pp. 183–190.

[Gai+15] P.-E. Gaillardon et al. “A Survey on Low-Power Techniques with Emerging Technologies: From Devices to Systems”. In: *J. Emerg. Technol. Comput. Syst.* 12.2 (Sept. 2015), 12:1–12:26.

[GB02] T. G. W. Gordon and P. J. Bentley. “On evolvable hardware”. In: *Soft Computing in Industrial Electronics*. Heidelberg, Germany: Physica-Verlag, 2002, pp. 279–323.

[GS11] Z. Gajda and L. Sekanina. “On Evolutionary Synthesis of Compact Polymorphic Combinational Circuits”. In: *Journal of Multiple-Valued Logic and Soft Computing* 17.6 (2011), pp. 607–631.

[GT06] G. W. Greenwood and A. M. Tyrrell. *Introduction to Evolvable Hardware: A Practical Guide for Designing Self-Adaptive Systems (IEEE Press Series on Computational Intelligence)*. Wiley-IEEE Press, 2006.

[HB11] S. L. Harding and W. Banzhaf. “Hardware Acceleration for CGP: Graphics Processing Units”. In: *Cartesian Genetic Programming*. Springer Science + Business Media, 2011, pp. 231–253.

[Hig+93] T. Higuchi et al. “Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine”. In: *Proc. of the 2nd International Conference on Simulated Adaptive Behaviour*. MIT Press, 1993, pp. 417–424.

[HLY06] T. Higuchi, Y. Liu, and X. Yao, eds. *Evolvable Hardware*. Springer Science+Media LLC, New York, 2006.

[HMV16] R. Hrbacek, V. Mrazek, and Z. Vasicek. “Automatic Design of Approximate Circuits by Means of Multi-Objective Evolutionary Algorithms”. In: *Proc.of the 11th Int. Conf. on Design and Technology of Integrated Systems in Nanoscale Era*. IEEE, 2016, pp. 239–244.

[Hol+16] L. Holik et al. “Towards Formal Relaxed Equivalence Checking in Approximate Computing Methodology”. In: *2nd Workshop on Approximate Computing (WAPCO 2016)*. HiPEAC, 2016, pp. 1–6.

[HS96] G. D. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Springer US, 1996, p. 564.

[HT11] P. C. Haddow and A. Tyrrell. “Challenges of evolvable hardware: past, present and the path to a promising future”. In: *Genetic Programming and Evolvable Machines* 12 (3 2011), pp. 183–215.

[IP98] S. Iman and M. Pedram. *Logic Synthesis for Low Power VLSI Designs*. Springer Science + Business Media, 1998.

[K+03] N. S. Kim, T. Austin, D. Blaauw, et al. “Leakage Current: Moore’s Law Meets Static Power”. In: *Computer* 36.12 (Dec. 2003), pp. 68–75.

[KGE11] P. Kulkarni, P. Gupta, and M. D. Ercegovac. “Trading Accuracy for Power in a Multiplier Architecture”. In: *J. Low Power Electronics* 7.4 (2011), pp. 490–501.

[Kni+10] T. Knieper et al. “Coping with Resource Fluctuations: The Run-time Reconfigurable Functional Unit Row Classifier Architecture”. In: *Proc. of the 9th Int. Conf. on Evolvable Systems: ¿From Biology to Hardware*. Vol. 6274. LNCS. Springer, 2010, pp. 250–261.

[Knu98] D. E. Knuth. *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., 1998.

[Koz92] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

[Mat+15] J. M. Matos et al. “Mapping Circuits with Simple Cells from Xor-And-Inverter Graphs”. In: *Proc. of the 24th International Workshop on Logic and Synthesis*. 2015.

[MCB06] A. Mishchenko, S. Chatterjee, and R. Brayton. “DAG-aware AIG Rewriting a Fresh Look at Combinational Logic Synthesis”. In: *Proceedings of the 43rd Annual Design Automation Conference*. DAC ’06. San Francisco, CA, USA: ACM, 2006, pp. 532–535.

[MD02] J. F. Miller and K. Downing. “Evolution in Materio: Looking Beyond the Silicon Box”. In: *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware (EH’02)*. IEEE Computer Society, 2002, pp. 167–176.

[MGO13] J. Miao, A. Gerstlauer, and M. Orshansky. “Approximate Logic Synthesis Under General Error Magnitude and Frequency Constraints”. In: *Proceedings of the International Conference on Computer-Aided Design*. ICCAD ’13. San Jose, California: IEEE Press, 2013, pp. 779–786.

[Mil11] J. F. Miller. *Cartesian Genetic Programming*. Springer-Verlag, 2011.

[Mil99a] J. F. Miller. “Digital Filter Design at Gate-level Using Evolutionary Algorithms”. In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 1999*. Morgan Kaufmann, 1999, pp. 1127–1134.

[Mil99b] J. F. Miller. “On the Filtering Properties of Evolved Gate Arrays”. In: *1st NASA-DoD Workshop on Evolvable Hardware*. IEEE Computer Society, 1999, pp. 2–11.

[Mis12] A. Mishchenko. *ABC: A System for Sequential Synthesis and verification, Berkley Logic Synthesis and Verification Group*. 2012.

[Mit16] S. Mittal. “A Survey of Techniques for Approximate Computing”. In: *ACM Comput. Surv.* 48.4 (2016), 62:1–62:33.

[Mra+16] V. Mrazek et al. “Design of Power-Efficient Approximate Multipliers for Approximate Artificial Neural Networks”. In: *Proceedings of the 35th IEEE/ACM International Conference on Computer-Aided Design (ICCAD).* Austin, TX, US, 2016. to appear in.

[MT00] J. F. Miller and P. Thomson. “Cartesian Genetic Programming”. In: *Proc. of the 3rd European Conference on Genetic Programming EuroGP2000*. Vol. 1802. LNCS. Springer, 2000, pp. 121–132.

[MT98] J. F. Miller and P. Thomson. “Evolving Digital Electronic Circuits for Real-Valued Function Generation using a Genetic Algorithm”. In: *University of Wisconsin*. Morgan Kaufmann, 1998, pp. 863–868.

[MTF97] 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*. Wiley, 1997, pp. 105–131.

[MV16] V. Mrazek and Z. Vasicek. “Automatic Design of Arbitrary-Size Approximate Sorting Networks with Error Guarantee”. In: *26rd International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2016*. Bremen, DE: IEEE Computer Society, 2016, (to appear).

[RP09] K. Roy and S. Prasad. *Low-Power Cmos VLSI Circuit Design*. Wiley India Pvt. Limited, 2009.

[San+16] A. J. Sanchez-Clemente et al. “Error Mitigation Using Approximate Logic Circuits: A Comparison of Probabilistic and Evolutionary Approaches”. In: *IEEE Transactions on Reliability* online first (2016), pp. 1–13.

[Sek04] L. Sekanina. *Evolvable Components: From Theory to Hardware Implementations*. Natural Computing Series, Springer Verlag, 2004.

[Sek12] L. Sekanina. “Evolvable hardware”. In: *Handbook of Natural Computing*. Berlin, DE: Springer Verlag, 2012, pp. 1657–1705.

[SG10] D. Shin and S. K. Gupta. “Approximate logic synthesis for error tolerant applications”. In: *Design, Automation and Test in Europe, DATE 2010*. IEEE, 2010, pp. 957–960.

[Sha12] E. N. Shauly. “CMOS Leakage and Power Reduction in Transistors and Circuits: Process and Layout Considerations”. In: *Journal of Low Power Electronics and Applications* 2.1 (2012), p. 1.

[SKL06a] E. Stomeo, T. Kalganova, and C. Lambert. “Generalized Disjunction Decomposition for Evolvable Hardware”. In: *IEEE Transaction Systems, Man and Cybernetics, Part B* 36.5 (2006), pp. 1024–1043.

[SKL06b] E. Stomeo, T. Kalganova, and C. Lambert. “Generalized Disjunction Decomposition for Evolvable Hardware”. In: *IEEE Transaction Systems, Man and Cybernetics, Part B* 36.5 (2006), pp. 1024–1043.

[Soe+16] M. Soeken et al. “BDD Minimization for Approximate Computing”. In: *Proceedings of the 21st Asia and South Pacific Design Automation Conference (ASP-DAC 2016)*. Macao SAR, China: IEEE, Jan. 2016, pp. 474–479.

[SP09] A. P. Shanthi and R. Parthasarathi. “Practical and scalable evolution of digital circuits”. In: *Applied Soft Computing* 9.2 (2009), pp. 618–624.

[SPG02] D. Soudris, C. Piguet, and C. Goutis. *Designing CMOS Circuits for Low Power*. European low-power initiative for electronic system design. Springer, 2002.

[Sto+04] A. Stoica et al. “Taking evolutionary circuit design from experimentation to implementation: some useful techniques and a silicon demonstration”. In: *Computers and Digital Techniques, IEE Proceedings - Volume 151, Issue 4*. 2004, pp. 295–300.

[Sto+99] A. Stoica et al. “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. Washington, DC, USA: IEEE Computer Society, 1999, pp. 76–84.

[SV12] L. Sekanina and Z. Vasicek. “A SAT-based Fitness Function for Evolutionary Optimization of Polymorphic Circuits”. In: *Proc. of the Design, Automation and Test in Europe, DATE*. European Design and Automation Association, 2012, pp. 715–720.

[SV13] L. Sekanina and Z. Vasicek. “Approximate circuit design by means of evolvable hardware”. In: *Evolvable Systems (ICES), IEEE International Conference on*. Singapur, SG: IEEE Computer Society, Apr. 2013, pp. 21–28.

[SZK01] A. Stoica, R. S. Zebulum, and D. Keymeulen. “Polymorphic Electronics”. In: *Proc. of International Conference on Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science, volume 2210*. Springer-Verlag, 2001, pp. 291–302.

[Tan+10] S. Tanachutiwat et al. “Reconfigurable Multi-Function Logic Based on Graphene P-N Junctions”. In: *Design Automation Conference, DAC*. ACM, 2010, pp. 883–888.

[Tho96] A. Thompson. “Silicon evolution”. In: *Proceedings of the First Annual Conference on Genetic Programming*. GECCO ’96. Stanford, California: MIT Press, 1996, pp. 444–452.

[TLZ99] A. Thompson, P. Layzell, and S. Zebulum. “Explorations in Design Space: Unconventional Electronics Design Through Artificial Evolution”. In: *IEEE Transactions on Evolutionary Computation* 3.3 (1999), pp. 167–196.

[TT15] M. A. Trefzer and A. M. Tyrrell. *Evolvable Hardware: From Practice to Application*. Springer-Verlag Berlin Heidelberg, 2015.

[Vas+10] Z. Vasicek and L. Sekanina. “Hardware Accelerator of Cartesian Genetic Programming with Multiple Fitness Units”. In: *Computing and Informatics* 29.7 (2010), pp. 1359–1371.

[Vas+11a] Z. Vasicek and L. Sekanina. “A Global Postsynthesis Optimization Method for Combinational Circuits”. In: *Proc. of the Design, Automation and Test in Europe, DATE*. IEEE Computer Society, 2011, pp. 1525–1528.

[Vas+11b] Z. Vasicek and L. Sekanina. “Extensions of Cartesian Genetic Programming for Optimization of Complex Combinational Circuits”. In: *Proc. of the 20th International Workshop on Logic and Synthesis*. San Diego, US: University of California San Diego, 2011, pp. 55–61.

[Vas+11c] Z. Vasicek and L. Sekanina. “Formal Verification of Candidate Solutions for Post-Synthesis Evolutionary Optimization in Evolvable Hardware”. In: *Genetic Programming and Evolvable Machines* 12.3 (2011), pp. 305–327.

[Vas+12a] Z. Vasicek and L. Sekanina. “On area minimization of complex combinational circuits using cartesian genetic programming”. In: *2012 IEEE Congress on Evolutionary Computation*. June 2012, pp. 1–8.

[Vas+12b] Z. Vasicek and K. Slany. “Efficient Phenotype Evaluation in Cartesian Genetic Programming”. In: *Proc. of the 15th European Conference on Genetic Programming*. LNCS 7244. Springer Verlag, 2012, pp. 266–278.

[Vas+13] Z. Vasicek, M. Bidlo, and L. Sekanina. “Evolution of efficient real-time nonlinear image filters for FPGAs”. In: *Soft Computing* 17.11 (2013), pp. 2163–2180.

[Vas+14a] Z. Vasicek and L. Sekanina. “Evolutionary Design of Approximate Multipliers Under Different Error Metrics”. In: *IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems 2013*. IEEE, 2014, pp. 135–140.

[Vas+14b] Z. Vasicek and L. Sekanina. “How to Evolve Complex Combinational Circuits From Scratch?” In: *2014 IEEE International Conference on Evolvable Systems Proceedings*. IEEE, 2014, pp. 133–140.

[Vas+15a] Z. Vasicek and L. Sekanina. “Circuit Approximation Using Single- and Multi-Objective Cartesian GP”. In: *Proc. of the 18th European Conference on Genetic Programming – EuroGP*. LNCS 9025. Springer, 2015, pp. 217–229.

[Vas+15b] Z. Vasicek and L. Sekanina. “Evolutionary Approach to Approximate Digital Circuits Design”. In: *IEEE Transactions on Evolutionary Computation* 19.3 (2015), pp. 432–444.

[Vas+16a] Z. Vasicek and V. Mrazek. “Trading between Quality and Non-functional Properties of Median Filter in Embedded Systems”. In: *Genetic Programming and Evolvable Machines* 2016.3 (2016), pp. 1–38.

[Vas+16b] Z. Vasicek and L. Sekanina. “Evolutionary Design of Complex Approximate Combinational Circuits”. In: *Genetic Programming and Evolvable Machines* 17.2 (2016), pp. 169–192.

[Vas15] Z. Vasicek. “Cartesian GP in Optimization of Combinational Circuits with Hundreds of Inputs and Thousands of Gates”. In: *Proceedings of the 18th European Conference on Genetic Programming – EuroGP*. LCNS 9025. Springer International Publishing, 2015, pp. 139–150.

[Ven+11] R. Venkatesan et al. “MACACO: Modeling and analysis of circuits for approximate computing”. In: *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2011, pp. 667–673.

[Ven+12] S. Venkataramani et al. “SALSA: systematic logic synthesis of approximate circuits”. In: *The 49th Annual Design Automation Conference 2012, DAC ’12*. ACM, 2012, pp. 796–801.

[VJM00] V. 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*. Ed. by J. Lohn et al. Los Alamitos, CA, USA: IEEE Computer Society, 2000, pp. 151–160.

[VRR13] S. Venkataramani, K. Roy, and A. Raghunathan. “Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits”. In: *Design, Automation and Test in Europe, DATE’13*. EDA Consortium San Jose, CA, USA, 2013, pp. 1367–1372.

[Wal16] M. M. Waldrop. “The chips are down for Moore’s law”. In: *Nature* 530.7589 (Feb. 2016), pp. 144–147.

[WCC09] L.-T. Wang, Y.-W. Chang, and K.-T. (Cheng, eds. *Electronic Design Automation: Synthesis, Verification, and Test*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009.

[WCL08] J. Wang, Q. S. Chen, and C. H. Lee. “Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware”. In: *IET computers and digital techniques* 2.5 (2008), pp. 386–400.

[XMK16] Q. Xu, T. Mytkowicz, and N. S. Kim. “Approximate Computing: A Survey”. In: *IEEE Design Test* 33.1 (Feb. 2016), pp. 8–22.

[YC16] C. Yu and M. Ciesielski. “Analyzing Imprecise Adders Using BDDs – A Case Study”. In: *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. July 2016, pp. 152–157.

[ZJ06] S. Zhao and L. Jiao. “Multi-objective evolutionary design and knowledge discovery of logic circuits based on an adaptive genetic algorithm”. In: *Genetic Programming and Evolvable Machines* 7.3 (2006), pp. 195–210.

Appendices - Paper reprints

Appendix A

Formal Verification of Candidate Solutions for Post-Synthesis Evolutionary Optimization in Evolvable Hardware

VASICEK, Zdenek and SEKANINA, Lukas. “Formal Verification of Candidate Solutions for Post-Synthesis Evolutionary Optimization in Evolvable Hardware”. In: *Genetic Programming and Evolvable Machines* 12.3 (2011), pp. 305–327.

IF=1.000 (2011), contribution of the author of the thesis: 50%

Referenced on pages: 10, 14, 15

Genet Program Evolvable Mach (2011) 12:305–327
DOI 10.1007/s10710-011-9132-7

Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware

Zdenek Vasicek · Lukas Sekanina

Received: 1 September 2010 / Revised: 15 February 2011 / Published online: 13 March 2011

Abstract We propose to utilize a formal verification algorithm to reduce the fitness evaluation time for evolutionary post-synthesis optimization in evolvable hardware. The proposed method assumes that a fully functional digital circuit is available. A post-synthesis optimization is then conducted using Cartesian Genetic Programming (CGP) which utilizes a satisfiability problem solver to decide whether a candidate solution is functionally correct or not. It is demonstrated that the method can optimize digital circuits of tens of inputs and thousands of gates. Furthermore, the number of gates was reduced for the LGSynth93 benchmark circuits by 37.8% on average with respect to results of the conventional SIS tool.

Keywords Cartesian genetic programming · Circuit optimization · SAT solver · Evolvable hardware

1 Introduction

In the evolvable hardware field, evolutionary algorithms (and other bio-inspired algorithms) are applied either for automated hardware design or dynamic hardware adaptation or repair [16, 20, 30, 39, 53, 54]. According to Gordon and Bentley, the field of evolvable hardware originates from the intersection of computer science, electronic engineering and biology and typically includes aspects of hardware design and optimization techniques, particularly logic synthesis, technology mapping, placing and routing [14].

Z. Vasicek · L. Sekanina (✉)
Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic
e-mail: sekanina@fit.vutbr.cz

Z. Vasicek
e-mail: vasicek@fit.vutbr.cz

In this article we will only deal with evolvable hardware as a method for automated design, i.e. with a scenario in which the evolutionary algorithm is used only in design and optimization phase of a product. In this context, evolvable hardware potentially offers promising solutions to logic synthesis and optimization where new problems have recently been identified. It was shown that commonly used logic synthesis algorithms are not capable of efficient synthesis and optimization for some circuit classes, especially for large circuits and circuits containing hard-to-synthesize substructures [5, 10]. In some cases the size of synthesized circuits is of orders of magnitude greater than the optimum.

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 solutions can be generated for larger problem instances. Although various methods have been proposed to eliminate the scalability problem (see Sect. 2), only a partial success has been achieved in some domains.

We will consider a subarea of the scalability problem—the *scalability of evaluation*, in the context of optimization problems. We will show that it can reasonably be eliminated in a task of *gate-level post-synthesis optimization* of complex combinational circuits consisting of thousands of gates and having tens of inputs and outputs. The method assumes that a fully functional circuit is available in a standard netlist format which can be obtained using a conventional synthesis algorithm. The main goal is to reduce the number of gates.

We propose to use modern formal verification methods that have been overlooked by the evolvable hardware community so far. The proposed method utilizes equivalence checking algorithms (those used by conventional synthesis algorithms) that allow a significant acceleration of the fitness evaluation procedure. Particularly, the method is based on a post-synthesis optimization of combinational circuit conducted using Cartesian genetic programming (CGP) [34] which evaluates candidate solutions using the satisfiability (SAT) solver [9]. The technique relies on functional correctness of an initial solution (a seed for CGP). Note that not all applications of evolvable hardware fall into this category because such a seed is not generally available. We have also introduced some techniques that explore the CGP representation and operators to reduce the number of clauses for the SAT solver and thus further shorten the evaluation time.

Optimized circuits are compared with the most compact circuits that we obtained from iterative application of decomposition and re-synthesis process which is conducted by conventional synthesis tools such as ABC and SIS.

The plan for this article is as follows. Section 2 introduces the concept of evolvable hardware and surveys the scalability problems. In Sect. 3, the proposed method is explained. The key contribution of this article, the construction of the fitness function on the basis of formal verification techniques is introduced in Sect. 4 The experimental evaluation of the proposed method represents the content of Sect. 5. Some practical aspects of the method are discussed in Sect. 6. Section 7 gives our conclusions from the experimental evaluation and also some suggestions for further work.

2 Evolvable hardware and its scalability

2.1 Motivation for circuit evolution

Figure 1 explains the concept of evolvable hardware: Electronic circuits that are encoded as finite strings of symbols are constructed and optimized by the evolutionary algorithm to obtain a circuit implementation satisfying a specification given by designer. Since the introduction of evolvable hardware at the beginning of nineties [11, 21], the main motivation for circuit evolution can be seen in the fact that evolutionary approach can lead to fully functional designs without being instructed how to construct them. Hence one of the goals is to evolve as complex circuit as possible with a minimum computational effort and domain knowledge supplied [41, 43, 51]. A typical application could be a reactive robot controller which is evolved in a sufficiently large reconfigurable device where there is no need to optimize the number of gates and delay [27].

In many applications a perfect circuit response must by obtained for all requested assignments to the inputs. 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. When target functionality is obtained additional criteria can be optimized. Evolution of arithmetic circuits is a typical example of that class [32, 49]. To give examples where partially imperfect solutions are acceptable we can mention evolution of image filters, classifiers or predictors [12, 19, 38]. In addition to functionality, another goal can be to obtain a solution which exhibits a better quality in some aspects with respect to existing designs of the same category. For example, a solution would occupy a smaller area on a chip, compute faster, provide a better precision, reduce the energy consumption, increase the reliability etc.

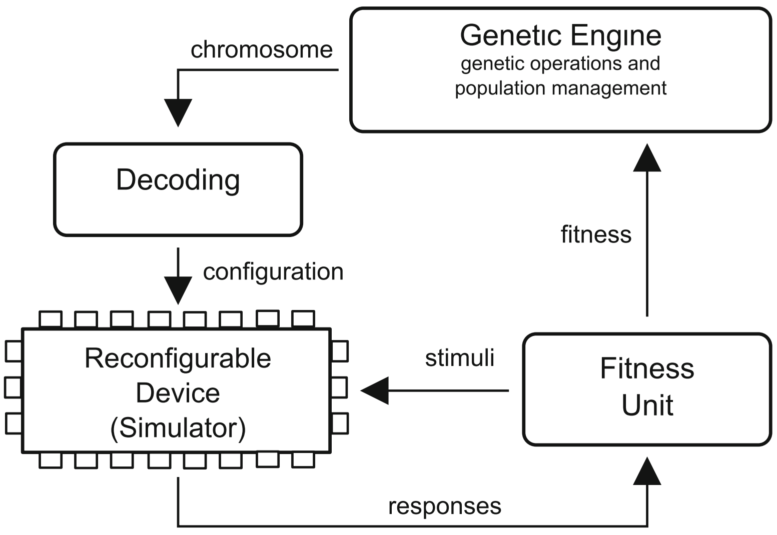


Fig. 1 The principle of evolvable hardware

2.2 Scalability of fitness evaluation

In case of 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). This fitness calculation method is currently applicable for circuits with up to 10–20 inputs (depending on a particular target function) [37, 41, 43, 49, 51]. In order to reduce the time of evaluation, various techniques can be adopted:

- Only a subset of all possible input vectors is utilized. That is typical for synthesis of filters, classifiers or robot controllers. Unfortunately, the approach is not applicable for synthesis of arithmetic circuits as it does not ensure that correct responses will be obtained for those input combinations which were not used during evolution [23].
- In some cases it is sufficient to evaluate only some structural properties (not the functionality!) of candidate circuits which can be done with a reasonable time overhead. For example, because testability of a candidate circuit can be calculated in the quadratic time complexity, very large benchmark circuits with predefined testability properties (more than 1 million gates) were evolved [36].
- In case that a target system is linear, it is possible to perfectly evaluate a candidate circuit using a single input vector independently of the circuit complexity. Multiple-constant multipliers composed of adders, subtractors and shifters were evolved for a 16-bit input and tens of 16-bit outputs [48].

An obvious conclusion is that the evaluation time becomes the main bottleneck of the evolutionary approach when complex digital circuits with many inputs are evolved or optimized.

2.3 Scalability of representation

From the viewpoint 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, including functional-level evolution [35, 39], incremental evolution [43, 44, 45], modularization [26, 51] and their combinations [12, 41]. Despite the fact that a new field of computational development has attracted a lot of attention in this area and brought some theoretical as well as practical results [15, 17, 18, 22, 29, 31, 42, 47, 55] the problem of scalability is still an open issue.

3 Proposed method

The goal of proposed method is to minimize the number of gates in a functionally correct combinational circuit that is typically obtained using a conventional synthesis tool. The method consists of three main steps that will be described in detail in the following sections:

1. Perform the synthesis/optimization using a conventional synthesis algorithm.
2. Convert resulting circuit to the CGP representation and use it to seed the initial population of CGP.
3. Run CGP that uses a formal verification method that will be described in Sect. 4 to reduce the number of gates. CGP is terminated if either the maximum allowed number of generations has been exhausted or a solution that fulfills the requirements has been discovered.

3.1 Conventional circuit synthesis

Combinational logic functions are commonly specified by PLA files where PLA stands for programmable logic array. 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. A circuit can also be represented as a netlist of gates in BLIF (Berkeley Logic Interchange Format) format. BLIF lists all interconnected combinational gates (and latches in case of sequential circuits).

Since proposed method is intended for a gate-level optimization, other steps of the circuit design process such as mapping, routing, placement and subsequent technology-specific optimizations are not considered in this paper. From conventional and routinely used synthesis methods we have chosen the SIS [40] tool (version sis1.2) which provided in most cases better results than other tools such as ABC [3] (version abc70930) or Espresso [4].

Implementations of 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 or look-up tables. The ABC and SIS tools are deterministic. They attempt to apply various circuit decomposition and re-synthesis techniques to transform a circuit under optimization and generate optimized netlist. We have used them with recommended (standard) setting which is represented by synthesis scripts given in Table 1. In order to improve their results we applied them on their own results iteratively as suggested in [3]. That technique will be discussed in Sect. 5.5.

3.2 Cartesian genetic programming

Cartesian Genetic Programming is a widely-used method for evolution of digital circuits [32, 34]. CGP was originally defined for gate-level evolution; however, it can easily be extended for functional level evolution [38]. In its basic version, candidate circuits are directly represented in the chromosome. The following paragraphs describe how we have used CGP in the proposed method.

3.2.1 Representation

A candidate entity (circuit) is modeled as an array of n_c (columns) $\times$ n_r (rows) of programmable nodes (gates). The number of inputs, n_i, and outputs, n_o, is fixed. Each node input can be connected either to the output of a node placed in the

Table 1 Synthesis scripts for the SIS and ABC method

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	

previous l columns or to one of the circuit inputs. The l-back parameter, in fact, defines the level of connectivity and thus reduces/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. Feedback is not allowed. Each node is programmed to perform one of n_a-input functions defined in the set Γ (n_f denotes $|\Gamma|$). As Fig. 2 shows, while the size of chromosome is fixed, the size of phenotype is variable (i.e. some nodes are not used). Every individual is encoded using $n_c \times n_r \times (n_a + 1) + n_o$ integers.

3.2.2 *Search algorithm*

CGP operates with the population of $1 + \lambda$ individuals (typically, $\lambda = 4$). The initial solution (the seed) is constructed by means of mapping of the circuit obtained

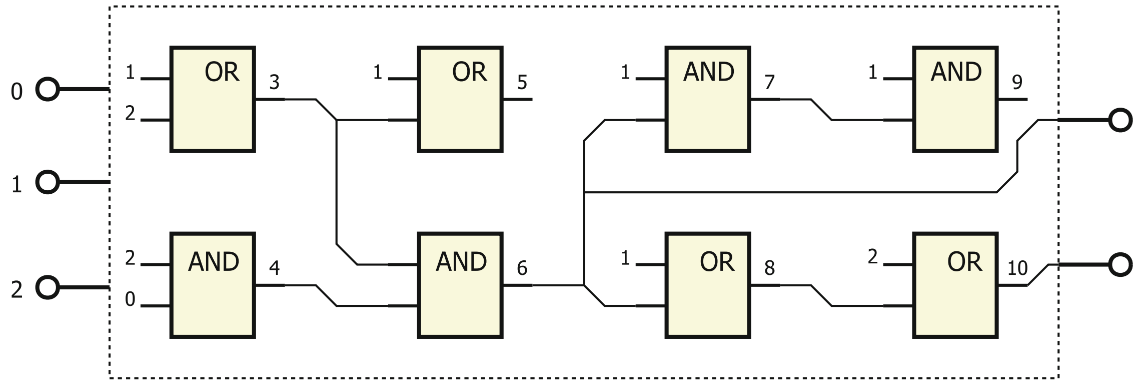


Fig. 2 Example of a candidate circuit. CGP parameters are as follows: $l = 3$, $n_c = 4$, $n_i = 3$, $n_o = 2$, $n_r = 2$, $\Gamma = \{\text{AND (0), OR (1)}\}$. Nodes 5, 7 and 9 are not utilized. Chromosome: 1,2,**1**, 2,0,**0**, 1,3,**1**, 3,4,**0** 1,6,**0**, 1,6,**1**, 1,7,**0**, 2,8,**1**, 6, 10. The last two integers indicate the outputs of circuit. The function of a gate is typed in bold

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 $n_c = m$, $n_r = 1$, $l = n_c$, $n_a = \gamma$. The circuit is also transformed into the conjunctive normal form in order to create a reference solution for the formal verification (see the method in Sect. 4 and Fig. 5).

The seed together with λ offspring created using a point mutation operator form the initial population which has to be evaluated. Every new population consists of the best individual of the previous population and its λ offspring. In case when two or more individuals have received the same highest fitness score in the previous generation, one of individuals which have not served as a parent in the previous population will be selected as a new parent. This strategy is important because it contributes to ensuring the diversity of population [33].

3.2.3 *Fitness function*

When the objective is to minimize the number of gates the fitness value of a candidate circuit may be defined in CGP as [24]:

$$fitness = B + (n_c n_r - z) \tag{1}$$

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 $n_c.n_r$ is the total number of gates available. The last term $n_c n_r - z$ is considered only if the circuit behavior is perfect, i.e. $B = n_o 2^{n_i}$.

The fitness calculation carried out by the proposed method differs from equation 1. Instead of evaluating all possible assignments to the inputs, a candidate circuit is verified against a reference circuit as described in Sect. 4. The result of the verification algorithm is a Boolean value. If the value is negative then the fitness score is the worst-possible value. If the value is positive, the fitness value is just the number of utilized gates (assuming that the goal is to minimize here) which can easily be obtained from the CGP representation of a candidate solution.

3.2.4 *Acceleration techniques for standard CGP*

We will also utilize fitness calculation according to (1) in order to compare the results with the formal verification-based fitness calculation. However, two modifications are incorporated to the implementation of (1) to reduce the computational overhead:

Because the 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., $0...2^{n_i} - 1$ 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 jth input vector, $j \in \{0...2^{n_i} - 1\}$, during the evaluation, the remaining $2^{n_i} - j - 1$ vectors are not evaluated and the circuit gets the worst possible score (0). Experimental results show that this technique reduces the computational overhead (see Table 3), 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.

Parallel simulation is another technique that can be used to accelerate circuit evaluation [32, 37]. The idea of parallel simulation is to utilize bitwise operators operating on multiple bits in a high-level language (such as C) to perform more than one evaluation of a gate in a single step. For example, when a combinational circuit under simulation has three inputs and it is possible to concurrently perform bitwise operations over $2^3 = 8$ bits in a simulator then the circuit can completely be simulated by applying a single 8-bit test vector at each input (see the encoding in Fig. 3). In contrast, when it is impossible then 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 a truth table of a six-input circuit by applying a single 64-bit test vector at each input. Therefore, the obtained speedup is 64 against the sequential simulation. In case that the circuit has more than 6 inputs then the speedup is constant, i.e. 64.

4 Formal verification approach in the fitness function

We propose to replace the fitness calculation approach based on testing of all possible assignments to the inputs by a functional equivalence checking algorithm. In order to specify the problem, a set of Boolean functions $F = \{f_1, f_2, ..., f_n\}$ can be used. Let each function f_i represent Boolean function of the ith output of a candidate circuit. Then the set F can be used to check whether a candidate solution meets the specification or not.

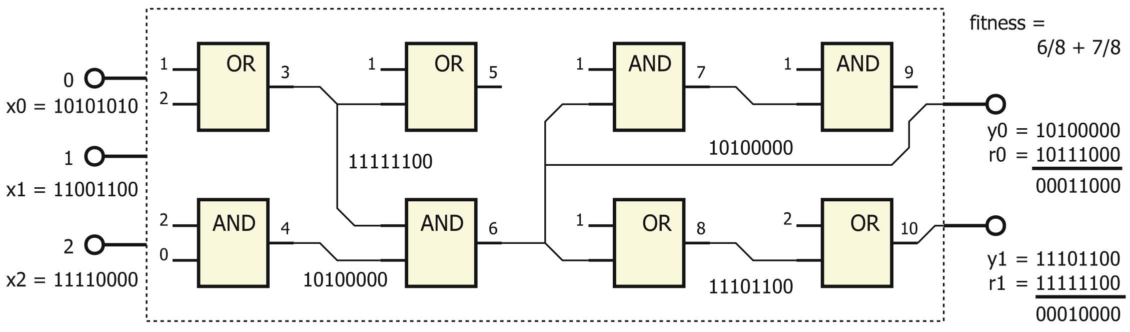


Fig. 3 Parallel simulation of a combinational circuit. The values y_0 and y_1 are the results of simulation; r_0 and r_1 are the required outputs

4.1 Functional 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.

Most of proposed techniques are based on representing a 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. The Reduced Ordered Binary Decision Diagrams (ROBDD) represent a widely used canonical representation in formal verification [52]. 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 (see Fig. 4) and checking whether the resulting ROBDD is a constant node (zero). And-or-invert graphs represent another canonic representation with similar properties. 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 a 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 [7]. The verification of such functions still represents a challenge.

High consumption of memory resources has motivated researchers to look for alternative methods. Since the satisfiability (SAT) solvers were significantly improved during the last few years, the SAT-based equivalence checking becomes

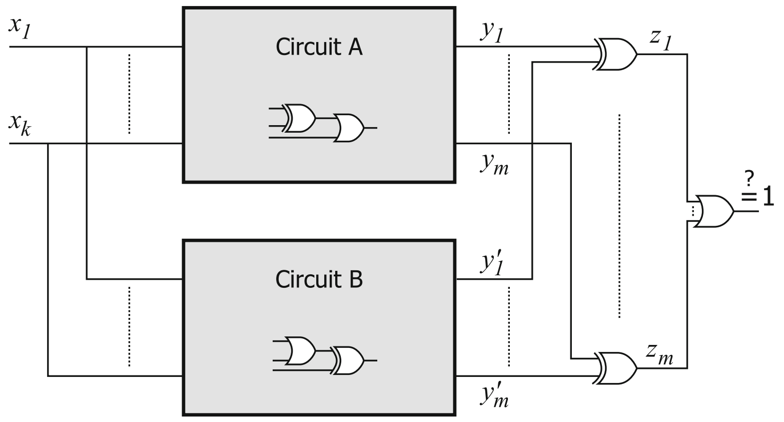


Fig. 4 Equivalence checking of two combinational circuits using the all outputs approach

to be a promising alternative to the BDD-based checking. In this case, circuits to be checked are transformed into one Boolean formula which is satisfiable if and only if the circuits are functionally equivalent [13]. In this article we will use the SAT-based equivalence checking because: (1) combinational circuits represented by CGP can be converted to Boolean formula in linear time with respect to the number of CGP nodes, (2) several optimization techniques specific for the evolutionary design can be applied and (3) the SAT-based checking becomes to be a preferred method as it outperforms the BDD-based approaches.

SAT solvers assume that the equivalence checking problem is expressed using Boolean formula in conjunctive normal form (CNF). CNF formula φ consists of a conjunction of clauses denoted as ω. Each clause contains a disjunction of literals. A literal is either variable x_i or its complement $\neg x_i$. Each clause can contain up to n literals providing there exists exactly n variables.

For our purposes, the most suitable transformation of the circuit to CNF is represented by Tseitin's algorithm proposed in [46] 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, and NOR 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, $\varphi = \omega_1 \wedge \omega_2 \wedge \ldots \wedge \omega_n$ 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 CNF contains exactly $k + n$ variables and the size of the resulting CNF is linear with respect to the size of C_A. Table 2 contains the CNF representation for the gates utilized in this article.

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. For SAT based equivalence checking of two circuits, corresponding primary outputs y_i and y_i' are connected using the XOR-gate. This gate is denoted as a *miter*. 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

Table 2 CNF representation of some common gates

Gate	Corresponding CNF representation
$y = \mathrm{NOT}(x_1)$	$(\neg y \vee \neg x_1) \wedge (y \vee x_1)$
$y = \mathrm{AND}(x_1,x_2)$	$(y \vee \neg x_1 \vee \neg x_2) \wedge (\neg y \vee x_1) \wedge (\neg y \vee x_2)$
$y = \mathrm{OR}(x_1,x_2)$	$(\neg y \vee x_1 \vee x_2) \wedge (y \vee \neg x_1) \wedge (y \vee \neg x_2)$
$y = \mathrm{XOR}(x_1,x_2)$	$(\neg y \vee \neg x_1 \vee \neg x_2) \wedge (\neg y \vee x_1 \vee x_2) \wedge$ $(y \vee \neg x_1 \vee x_2) \wedge (y \vee x_1 \vee \neg x_2)$
$y = \mathrm{NAND}(x_1,x_2)$	$(\neg y \vee \neg x_1 \vee \neg x_2) \wedge (y \vee x_1) \wedge (y \vee x_2)$
$y = \mathrm{NOR}(x_1,x_2)$	$(y \vee x_1 \vee x_2) \wedge (\neg y \vee \neg x_1) \wedge (\neg y \vee \neg x_2)$

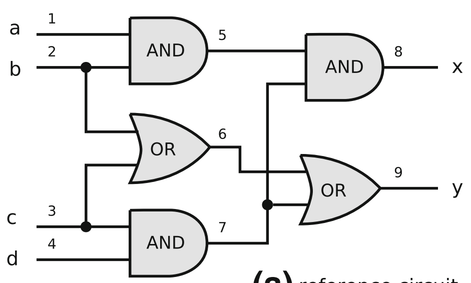
disprove the equivalence, it is necessary to identify at least one miter which evaluates to 1 for an input assignment, i.e. it is necessary to find an input assignment for which the corresponding outputs y_i and y_i' provide different values and thus $z_i = 1$. This can be done by checking one miter after another (i.e. a CNF is created and solved for each miter output separately) or by the all outputs approach (all miter 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 4 shows the all output approach adopted in this article.

4.2 Proposed 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 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 (see Fig. 5a).

The fitness calculation consists of the following steps:

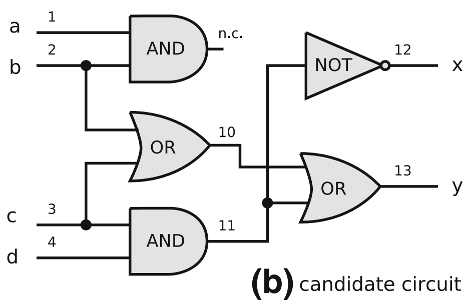
1. A new instance of the SAT solver is created and initialized with the reference circuit. This comprises creating of N new variables and submitting all clauses of φ_1 into the SAT solver.
2. A candidate solution is transformed to a list of clauses that are submitted into the SAT solver (see Fig. 5b). 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 2, 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.
3. Miters are 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 $y_i = \mathrm{XOR}(x_{N-i}, x_{N'-i})$, $i = 0\ldots m-1$ have to be submitted to the SAT solver (see Fig. 5c).
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



(a) reference circuit

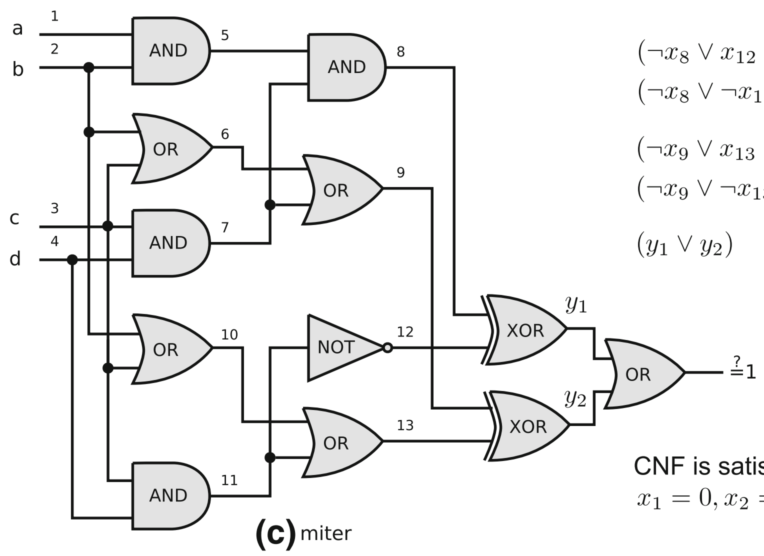
CNF

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



(b) candidate circuit

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



(c) miter

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

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

$(y_1 \vee y_2)$

CNF is satisfiable if
$x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0$

Fig. 5 Example of transformation of reference circuit, candidate circuit and miter to CNF

the outputs by m-input OR gate $z = \mathrm{OR}(y_1, \ldots, y_m)$. The corresponding CNF representation has the form of $(\neg z \vee x_1 \vee \cdots \vee x_k) \wedge \bigwedge_{i=1}^{k}(\neg x_i \vee z)$. In order to provide a CNF instance capable of the equivalence checking, it is necessary to append the clause (z) that implies $z = 1$, thus $(\neg z \vee y_1 \vee \cdots \vee y_k) \wedge \bigwedge_{i=1}^{k}(\neg y_i \vee z) \wedge (z) = (y_1 \vee \cdots \vee y_k)$. 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 Fig. 5c).

5. The SAT solver determines whether the submitted set of clauses is satisfiable or not. If the CNF is satisfiable, the fitness function returns 0 (the candidate circuit and the reference circuit are not equivalent); otherwise the number of utilized gates is returned.

4.3 Time of candidate circuit evaluation

In order to compare the time of evaluation for the common fitness function (Eq. 1) 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 [28]. The relevant CGP parameters are as follows: $\lambda = 4$, $\Gamma = \{\text{AND, OR, NOT, Identity}\}$, $l = N_g$, $n_c = N_g$ 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 Sect. 3.2.4. 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 3 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}, see Sect. 3.2.4) and the SAT-based method (t_{sat}). Last two columns contain the achieved speedup of proposed approach against the common and accelerated CGP. 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 practical. The MiniSAT 2 (version 070721) has been used as a SAT solver [9] because it can be effectively embedded into a custom application.

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.

Table 3 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}

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	1,328	139	0.903	1,471	153.9
26	171	5,962*	626*	1.028	5,799	608
28	181	26,748*	2,820*	1.195	22,383	2,359
30	199	119,996*	12,703*	1.211	99,088	10,489
32	215	538,327*	57,207*	1.348	399,352	42,438

Symbol ‘*’ denotes extrapolated values

4.4 CGP-specific performance improvement techniques

Although the system can be used directly as it was proposed in the previous section, we have introduced 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 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 a very 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 proposed. 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 [2, 6, 8, 50].

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 4. 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 Sect. 4.3 has been utilized. Obtained results show that the average time needed to evaluate a candidate

Table 4 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	1,408	49.80	15.52	3.21
Apex2	39	3	235	3.54	2.52	1.40
Apex3	54	50	1,407	34.56	13.93	2.48
Apex5	117	87	784	17.45	5.07	3.44

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 Results

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.

5.1 Population size

Table 5 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 $\Gamma = \{$Identity, AND, OR, NOT, XOR, NAND, NOR$\}$. The best values as well as mean values indicate that ES(1+1) performs better than ES(1+4) which corresponds with our intuitive assumption of very rugged fitness landscape.

5.2 Mutation rate and CGP grid size

Table 6 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 6 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

Table 5 The best and mean number of gates for different population sizing

Circuit	n_i	n_o	Seed [gates]	ES 1+4		ES 1+1	
				Best	Mean	Best	Mean
Apex1	45	45	1,408	1,240	1,267	1,201	1,255
Apex2	39	3	235	138	155	132	146
Apex3	54	50	1,407	1,336	1,350	1,331	1,347
Apex5	117	87	784	736	746	730	743
Mean			959	863	880	849	873

Table 6 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	1,240	1,290	1,351	1,377	1,382	1,393	1,260	1,290	1,351	1,379	1,385	1,392
Mean	1,269	1,313	1,367	1,387	1,396	1,399	1,287	1,326	1,369	1,390	1,395	1,399
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	1,341	1,358	1,383	1,392	1,395	1,396	1,345	1,362	1,383	1,392	1,396	1,398
Mean	1,354	1,369	1,389	1,397	1,399	1,400	1,357	1,372	1,390	1,397	1,400	1,401
Mean (miters)	2.6	3.6	6.2	9.4	12	14	2.6	3.5	6.1	9.4	11.9	14.1
Meant_{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

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 $i = 1...19$. The advantage of using $n_r > 1$ is that delay of the circuit is implicitly controlled to be below a given maximum value.

5.3 Parity benchmarks

In Sect. 4.3 we compared the evaluation time of the standard fitness function and the SAT-based fitness function in the task of parity circuits optimization. Table 7 shows

Table 7 The minimum number of gates that were obtained for parity circuits by running the proposed method for 3, 6, 9 and 12 h. TG gives the optimum solution

n_i	Seed [gates]	Run-time				TG [gates]	Relative improv. (%)
		3 h	6 h	9 h	12 h		
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

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 h on a 2.4 GHz processor. The results are averaged from 100 independent runs of CGP with the following setting: ES(1+1), 1 mutated gene/chromosome, $\Gamma = \{$Identity, AND, OR, 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 %.

5.4 LGSynth93 benchmarks

Table 8 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 h on a 2.4 GHz processor. The results are averaged from 100 independent runs of CGP with the following setting: ES(1+1), 1 mutated gate/chromosome, $\Gamma = \{$Identity, AND, OR, NOT, XOR, NAND, 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 optimizing them

Table 8 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 h

Circuit	n_i	n_o	Seed [gates]	Run-time				Relative improv. (%)
				3 h	6 h	9 h	12 h	
Apex1	45	45	1,408	1,179	1,083	1,026	990	30
				1,230	1,108	1,042	1,001	29
Apex2	39	3	235	104	101	99	98	58
				119	102	100	98	58
Apex3	54	50	1,407	1,280	1,223	1,189	1,167	17
				1,333	1,240	1,202	1,175	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	1,128	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

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

5.5 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 1,000 iterations of the SIS script. Figure 6 shows that convergence curves for two selected 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 h; using the best solution out of 100 independent runs) which can be also considered as a new seeding. Figure 6 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 h.

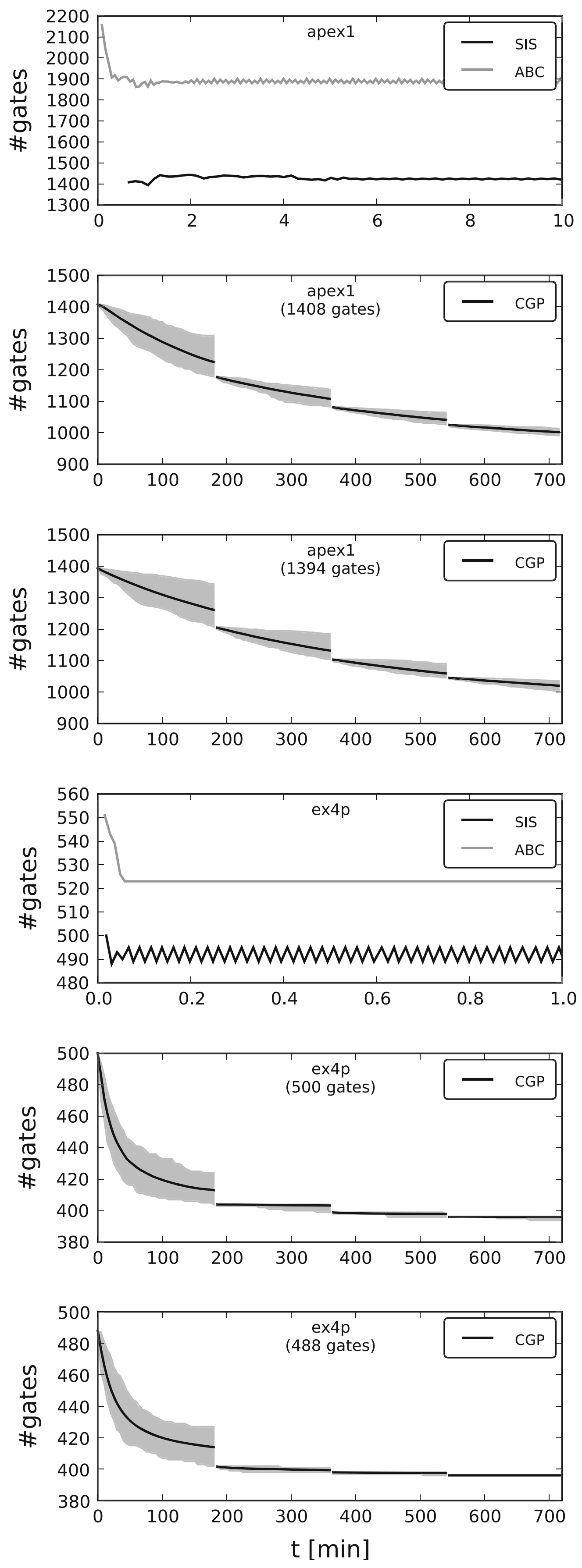


Fig. 6 Convergence curves for the apex1 and ex4p benchmarks. The mean, minimum and maximum number of gates from 100 independent runs of CGP when seeded using the result of the 1st iteration and the best result out of 1,000 iterations of the SIS tool. ABC and SIS were repeated until stable results observed

6 Discussion

Applying the SAT solver in the fitness function allowed us to significantly reduce 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. In this category of problems, we were able to optimize much larger circuit instances than standard CGP. Furthermore, we reduced the number of gates in solutions that can be delivered by conventional synthesis methods. However, proposed method requires significantly more computational time in comparison to conventional synthesis tools.

Although the results for LGSynth93 benchmarks are very encouraging, the SAT-based combinational equivalence checking can definitely perform unsatisfactory for some problem instances, for example for multipliers where the number of paths traversed by the SAT solver grows enormously with the increasing number of inputs. In order to improve performance of the SAT solver in this particular case, various techniques have been proposed to reduce the equivalence checking time [1, 2]. The proposed method is assumed to be able to handle large-scale multipliers optimization if more advanced version of SAT solver is utilized. Other techniques exist that can be employed to improve the proposed fitness function, e.g. CNF preprocessing, BDD-based checking, hierarchical equivalence checking etc. [25].

7 Conclusions

We demonstrated that some applications of evolvable hardware could benefit from formal verification techniques. The main advantage of our method is that the time of evaluation can significantly be reduced in comparison to the standard fitness function in cases when a fully functional solution exists before optimization is started. Consequently, we demonstrated that the circuit post-synthesis optimization conducted by CGP is applicable on complex digital circuits. CGP reduced the number of gates for the LGSynth93 benchmark circuits by 37.8% on average with respect to the SIS tool. Future research will be oriented towards improving the formal verification module by using more sophisticated verification algorithms and applying the proposed method in various domains, including software evolution, developmental CGP and numerous real-world evolvable hardware problems.

Acknowledgments This work was partially supported by the Czech Science Foundation under projects *Natural Computing on Unconventional Platforms* P103/10/1517 and *Mathematical and Engineering Approaches to Developing Reliable and Secure Concurrent and Distributed Computer Systems* GD102/09/H042 and by the research programme *Security-Oriented Research in Information Technology* MSM 0021630528.

References

1. F.V. Andrade, M.C.M. Oliveira, A.O. Fernandes, C.J.N. Coelho, Sat-based equivalence checking based on circuit partitioning and special approaches for conflict clause reuse, in *IEEE Design and Diagnostics of Electronic Circuits and Systems* (IEEE Comp. Society, 2007), pp. 1–6

2. F.V. Andrade, L.M. Silva, A.O. Fernandes, in *26th International Conference on Computer Design, ICCD 2008*. Improving SAT-based combinational equivalence checking through circuit preprocessing, 40–45 (2008)
3. Berkley Logic Synthesis and Verification Group: ABC: A System for Sequential Synthesis and verification. http://www.eecs.berkeley.edu/~alanmi/abc/
7. R.K. Brayton, G.D. Hachtel, C.T. McMullen, A.L. Sangiovanni-Vincentelli, Logic Minimization Algorithms for VLSI Synthesis. (Kluwer, Boston, MA, USA, 1984)
5. J. Cong, K. Minkovich, Optimality study of logic synthesis for LUT-based FPGAs. IEEE Trans. Comput. Aided Des. Integ. Circuits Syst. **26**(2), 230–239 (2007)
6. S. Disch, C. Schollm, in *Asia and South Pacific Design Automation Conference*. Combinational equivalence checking using incremental SAT solving, output ordering, and resets (2007), pp. 938–943
7. R. Ebendt, G. Fey, R. Drechsler, Advanced BDD Optimization. (Springer, Berlin, 2000)
8. N. Een, A. Mishchenko, N. Sorensson, Applying logic synthesis for speeding up SAT, in *Theory and Applications of Satisfiability Testing*, LNCS, vol. 4501 (Springer, Berlin, 2007), pp. 272–286
9. N. Een, N. Sorensson, MiniSAT. http://minisat.se
10. P. Fiser, J. Schmidt, in *Proceedings of 8th International Workshop on Boolean Problems*. Small but nasty logic synthesis examples (2008), pp. 183–190
11. H. de Garis, in *International Conference on Artificial Neural Networks and Genetic Algorithms ICANNGA'93*. Evolvable Hardware—Genetic Programming of a Darwin Machine. Innsbruck, Austria (1993)
12. K. Glette, J. Torresen, M. Yasunaga, in *Applications of Evolutinary Computing, EvoWorkshops 2007, LNCS*, vol. 4448. An Online EHW Pattern Recognition System Applied to Face Image Recognition (Springer, 2007), pp. 271–280
13. E. Goldberg, M. Prasad, R. Brayton, in *DATE '01: Proceedings of the Conference on Design, Automation and Test in Europe*. Using SAT for combinational equivalence checking (IEEE Press, Piscataway, NJ, USA, 2001), pp. 114–121
14. T.G.H. Gordon, P.J. Bentley, in *Handbook of Nature-Inspired and Innovative Computing*, ed. by A.Y. Zomaya. Evolving hardware (Springer, UK, 2006), pp. 387–432
15. T.G.W. Gordon, P.J. Bentley, in *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware*. Towards development in evolvable hardware (IEEE Computer Society Press, Washington, DC, US 2002), pp. 241–250
16. G. Greenwood, A.M. Tyrrell, Introduction to Evolvable Hardware. (IEEE Press, New York, 2007)
17. P.C. Haddow, G. Tufte, P. van Remortel, in *Proceedings of the 4th International Conference on Evolvable Systems: From Biology to Hardware, LNCS*, vol. 2210. Shrinking the genotype: L-systems for EHW? (Springer, Berlin, 2001), pp. 128–139
18. S. Harding, J.F. Miller, W. Banzhaf, in *2009 IEEE Congress on Evolutionary Computation*. Self Modifying Cartesian Genetic Programming: Parity (IEEE Press, New York, 2009), pp. 285–292
19. T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, M. Salami, N. Kajihara, N. Otsu, Real-world applications of analog and digital evolvable hardware. IEEE Trans. Evol. Comput. **3**(3), 220–235 (1999)
20. T. Higuchi, Y. Liu, X. Yao, Evolvable Hardware. (Springer, Berlin, 2006)
21. T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, T. Furuya, in *Proceedings of the 2nd International Conference on Simulated Adaptive Behaviour*. Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine (MIT Press, 1993), pp. 417–424
22. G. Hornby, A. Globus, D. Linden, J. Lohn, in *Proc. 2006 AIAA Space Conference*. Automated Antenna Design with Evolutionary Algorithms (AIAA, San Jose, CA, 2006), p. 8
23. K. Imamura, J.A. Foster, A.W. Krings, in *Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware*. The Test Vector Problem and Limitations to Evolving Digital Circuits (IEEE Computer Society Press, 2000), pp. 75–79
24. T. Kalganova, J.F. Miller, in *The First NASA/DoD Workshop on Evolvable Hardware*. Evolving More Efficient Digital Circuits by Allowing Circuit Layout Evolution and Multi-Objective Fitness (IEEE Computer Society, Pasadena, California, 1999), pp. 54–63
25. H. Katebi, I.L. Markov, in *Design, Automation and Test in Europe, DATE 2010*. Large-Scale Boolean Matching (IEEE, 2010), pp. 771–776
26. P. Kaufmann, M. Platzner, *Proceedings of Genetic and Evolutionary Computation Conference, GECCO 2008*. Advanced Techniques for the Creation and Propagation of Modules in Cartesian Genetic Programming (ACM, 2008), pp. 1219–1226

27. D. Keymeulen, M. Durantez, K. Konaka, Y. Kuniyoshi, T. Higuchi, in *Proceedings of the 1st International Conference on Evolvable Systems: From Biology to Hardware ICES'96, LNCS*, vol. 1259, eds. by T. Higuchi, M. Iwata, W. Liu. An Evolutionary Robot Navigation System Using a Gate-Level Evolvable Hardware (Springer, Tsukuba, Japan, 1997), pp. 195–209
28. J.R. Koza, Genetic Programming II: Automatic Discovery of Reusable Programs. (MIT Press, Cambridge, MA, 1994)
29. J.R. Koza, F.H. Bennett, D. Andre, M.A. Keane, Genetic Programming III: Darwinian Invention and Problem Solving. (Morgan Kaufmann Publishers, San Francisco, CA, 1999)
30. J.R. Koza, M.A. Keane, M.J. Streeter, W. Mydlowec, J. Yu, G. Lanza, Genetic Programming IV: Routine Human-Competitive Machine Intelligence. (Kluwer, Dordrecht, 2003)
31. D. Mange, M. Sipper, A. Stauffer, G. Tempesti, Towards robust integrated circuits: the embryonics approach. Proc. IEEE **88**(4), 516–541 (2000)
32. J.F. Miller, D. Job, V.K. Vassilev, Principles in the evolutionary design of digital circuits—part I. Genetic Programm. Evol. Mach. **1**(1), 8–35 (2000)
33. J.F. Miller, S.L. Smith, Redundancy and computational efficiency in cartesian genetic programming. IEEE Trans. Evol. Comput. **10**(2), 167–174 (2006)
34. J.F. Miller, P. Thomson, in *Proceedings of the 3rd European Conference on Genetic Programming EuroGP2000, LNCS*, vol. 1802. Cartesian Genetic Programming (Springer, 2000), pp. 121–132
35. M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, T. Higuchi, in *Parallel Problem Solving from Nature—PPSN IV, LNCS*, vol. 1141. Evolvable Hardware at Function Level (Springer, 1996), pp. 62–71
36. T. Pecenka, L. Sekanina, Z. Kotasek, Evolution of synthetic RTL benchmark circuits with predefined testability. ACM Trans. Des. Autom. Electron. Syst. **13**(3), 1–21 (2008)
37. R. Poli, J. Page, Solving high-order boolean parity problems with smooth uniform crossover, sub-machine code gp and demes. Genetic Programm. Evol. Mach. **1**(1–2), 37–56 (2000)
38. L. Sekanina, in *Applications of Evolutionary Computing—Proceedings of the 4th Workshop on Evolutionary Computation in Image Analysis and Signal Processing EvoIASP'02, LNCS*, vol. 2279. Image Filter Design with Evolvable Hardware (Springer Verlag, Kinsale, Ireland, 2002), pp. 255–266
39. L. Sekanina, Evolvable Components: From Theory to Hardware Implementations. (Natural Computing Series, Springer, Berlin, 2004)
40. E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, A. Sangiovanni-vincentelli, Sis: A system for sequential circuit synthesis. Technical report, University California, Berkeley (1992)
41. A.P. Shanthi, R. Parthasarathi, Practical and scalable evolution of digital circuits. Appl. Soft Comput. **9**(2), 618–624 (2009)
42. K.O. Stanley, R. Miikkulainen, A taxonomy for artificial embryogeny. Artif. Life **9**, 93–130 (2003)
43. E. Stomeo, T. Kalganova, C. Lambert, Generalized disjunction decomposition for evolvable hardware. IEEE Trans. Syst. Man Cybernet. Part B **36**(5), 1024–1043 (2006)
44. J. Torresen, in *Proceedings of the 2nd International Conference on Evolvable Systems: From Biology to Hardware ICES'98, LNCS*, vol. 1478, eds. by M. Sipper, D. Mange, A. Perez-Uribe. A Divide-and-Conquer Approach to Evolvable Hardware (Springer, Lausanne, Switzerland, 1998), pp. 57–65
45. J. Torresen, A scalable approach to evolvable hardware. Genetic Programm. Evol. Mach. **3**(3), 259–282 (2002)
46. G.S. Tseitin, in *Studies in Constructive Mathematics and Mathematical Logic, Part II.* On the Complexity of Derivation in Propositional Calculus (1968), pp. 115–125
47. G. Tufte, P.C. Haddow, Towards development on a silicon-based cellular computing machine. Nat. Comput. **4**(4), 387–416 (2005)
48. Z. Vasicek, M. Zadnik, L. Sekanina, J. Tobola, in *Proceedings of the 8th Conference on Evolvable Systems: From Biology to Hardware, LNCS*, vol. 5216. On Evolutionary Synthesis of Linear Transforms in FPGA (Springer, Berlin, 2008), pp. 141–152
49. V. Vassilev, D. Job, J.F. Miller, in *Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware*, eds. by J. Lohn, A. Stoica, D. Keymeulen, S. Colombano. Towards the Automatic Design of More Efficient Digital Circuits (IEEE Computer Society, Los Alamitos, CA, USA, 2000), pp. 151–160
50. M.N. Velev, Efficient translation of boolean formulas to CNF in formal verification of microprocessors, in *Asia South Pacific Design Automation Conference* (IEEE Computer Society, 2004), pp. 310–315

51. J.A. Walker, J.F. Miller, The automatic acquisition, evolution and re-use of modules in cartesian genetic programming. IEEE Trans. Evol. Comput. **12**(4), 397–417 (2008)
52. S. Yanushkevich, D.M. Miller, V.P. Shmerko, R.S. Stankovic, Decision Diagram Techniques for Micro- and Nanoelectronic Design Handbook. (CRC, Boca Raton, 2006)
53. X. Yao, T. Higuchi, Promises and challenges of evolvable hardware. IEEE Trans. Syst. Man Cybernet. Part C **29**(1), 87–97 (1999)
54. R. Zebulum, M. Pacheco, M. Vellasco, Evolutionary Electronics—Automatic Design of Electronic Circuits and Systems by Genetic Algorithms. (The CRC Press International Series on Computational Intelligence, Boca Raton, 2002)
55. S. Zhan, J.F. Miller, A.M. Tyrrell, in *Proc. of the 8th Int. Conference on Evolvable Systems: From Biology to Hardware, LNCS*,, vol. 5216. A Developmental Gene Regulation Network for Constructing Electronic Circuits (Springer, Berlin, 2008), pp. 177–188

Appendix B

Cartesian GP in Optimization of Combinational Circuits with Hundreds of Inputs and Thousands of Gates

VASICEK, Zdenek. “Cartesian GP in Optimization of Combinational Circuits with Hundreds of Inputs and Thousands of Gates”. In: *Proceedings of the 18th European Conference on Genetic Programming – EuroGP*. LCNS 9025. Springer International Publishing, 2015, pp. 139–150.

acceptance rate: 33%, this paper received *best paper award*

Referenced on pages: 15, 16, 45, 47

Cartesian GP in Optimization of Combinational Circuits with Hundreds of Inputs and Thousands of Gates

Zdenek Vasicek( )

Faculty of Information Technology, IT4Innovations Centre of Excellence,
Brno University of Technology, Brno, Czech Republic
vasicek@fit.vutbr.cz

Abstract. A new approach to the evolutionary optimization of large digital circuits is introduced in this paper. In contrast with evolutionary circuit design, the goal of the evolutionary circuit optimization is to minimize the number of gates (or other non-functional parameters) of already functional circuit. The method combines a circuit simulation with a formal verification in order to detect the functional inequivalence of the parent and its offspring. An extensive set of 100 benchmarks circuits is used to evaluate the performance of the method as well as the utilized evolutionary approach. Moreover, the role of neutral mutations in the context of evolutionary optimization is investigated. In average, the method enabled a 34 % reduction in gate count even if the optimizer was executed only for 15 min.

Keywords: Genetic programming · Cartesian Genetic Programming · Evolutionary optimization · Combinational circuits · Formal verification

1 Introduction

One of the most serious problems of evolvable hardware, especially in the area of evolutionary synthesis of logic circuits, is a very time consuming evaluation of candidate circuits. This problem is known as the problem of scalability. It causes that the evolutionary synthesis can handle only small and usually simple problems that are far from real-world problem instances.

In order to improve the scalability of evaluation, application-specific hardware as well as software methods were designed to increase the performance of the evolutionary optimization and design of logic circuits, see e.g. [2,4–6,9]. These methods enabled to increase the complexity of problem instances that can be solved in a reasonable time. Unfortunately, the methods are not scalable. The time needed to evaluate a candidate solution usually grows exponentially with the increasing number of primary inputs, but the accelerators are usually able to deliver a linear speedup only. Introducing more domain knowledge and utilizing more advanced evolutionary methods seem to be the only viable approach for dealing with the real-world problem instances. A breakthrough in the field of

P. Machado et al. (Eds.): EuroGP 2015, LNCS 9025, pp. 139–150, 2015.
DOI: 10.1007/978-3-319-16501-1_12

evolvable hardware was achieved with the introduction of a method which ties formal verification together with evolutionary optimization and substantially reduces the scalability issue of the evaluation [7]. Vasicek and Sekanina demonstrated that the previous empirical limitation of evolutionary design represented by a digital circuit having about 20 inputs can easily be overcome.

The goal of this paper is to introduce and evaluate a new approach which extends the method published in [8]. The advantage of the improved approach, which combines formal verification with simulation-based verification, is the ability to optimize digital circuits (i.e. to reduce the number of gates, improve power consumption, delay, etc.) represented at the gate level having hundreds of inputs and consisting of thousands of gates. The circuits of such a complexity have never been either evolved or optimized in the field of evolvable hardware at the gate level directly. In contrast with previously published works which are evaluated using a few benchmark circuits, an extensive set of 100 benchmarks circuits is used to evaluate the performance the proposed method. In addition to that, we would like to identify the key weaknesses of the evolutionary approach and propose future directions that could help the evolutionary approaches to penetrate into the area of real applications. In particular, we analyzed the role of neutral mutations in the context of evolutionary optimization.

2 Evolutionary Optimization of Combinational Circuits

2.1 Cartesian Genetic Programming

Cartesian Genetic Programming can be considered as one of the most efficient methods for evolutionary design and optimization of digital combinational circuits [3]. A candidate circuit is represented using an array of gates arranged in a matrix consisting of n_c columns and n_r rows. Each gate can be connected either to the output of a gate placed in previous l columns or to one of the circuit inputs. It means that no feedback is allowed. This requirement guarantees that only the combinational circuits will arise. Each gate is programmed to perform one of n_a-input functions defined in the set Γ. The number of circuit inputs, n_i, and outputs, n_o, is fixed. Every candidate circuit is encoded using $n_c \cdot n_r \cdot (n_a + 1) + n_o$ integers. The main advantage of the utilized encoding is that the size of phenotype is variable even if the size of chromosome is fixed. The variability is given by the fact that some nodes need not be employed in encoded circuit.

CGP operates with the population of $1+\lambda$ individuals. The initial population is usually seeded randomly. However, in order to optimize a known circuit (i.e. to minimize the number of gates), it is useful to seed the initial population by this circuit. Every new population consists of the best individual of the previous population and its λ offspring individuals. The offspring individuals are created using a point mutation operator which modifies h randomly selected genes of the chromosome. An important rule for selection of the new parent is utilized. In the case when two or more individuals can serve as the parent, an individual which has not served as the parent in the previous generation will be selected as a new parent. This strategy is important because it ensures the diversity of population [3]. The algorithm is terminated when the maximum number of generations is exhausted or a sufficient solution is obtained.

In case of digital circuit evolution, the fitness value of a candidate circuit is defined as follows. If a fully functional solution is evolved, the fitness value consist of the number of correct output bits obtained as response for all possible assignments to the inputs plus the number of unused CGP nodes. Otherwise, only the number of correct output bits is used. It means 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.

2.2 Speeding up the Fitness Evaluation Using a SAT Solver

Contrary to the evolutionary design, the evolutionary optimization of digital circuits begins with the population seeded by a fully functional circuit. Usually, the goal is to minimize the number of gates. The most important feature of the evolutionary optimization is that each candidate solution created by means of genetic operators must be functionally equivalent with its parent in order to be further evaluated. This feature was utilized in [7] and furthermore elaborated in [8]. Equivalence checking was applied to decide whether a candidate circuit is functionally correct or not. In order to calculate the fitness value, the candidate circuit as well as its parent are converted to a Boolean formula whose satisfiability is investigated using a SAT solver. In fact, the parent serves as a golden reference for combinational equivalence checking. The advanced version, introduced in [8], utilizes another feature of evolutionary-based approach – the knowledge of the points in a candidate circuit that may break the correct function. This information is available because each offspring was created by a mutation from its parent. Hence, only a 'difference' (so-called cone of influence) between the candidate solution and its parent can be calculated. The Boolean formula can be derived from this 'difference'. Since the cone of influence usually represents only a small part of the candidate circuit, the time needed to decide the satisfiability of the Boolean formula can significantly be reduced.

If the obtained Boolean formula is satisfiable, a negative fitness value is assigned to the candidate circuit because the candidate circuit captures a different Boolean function. Otherwise, the candidate circuit is functionally equivalent with the specification and the fitness value is calculated according to the objective of the optimization. For example, the number of utilized gates was used in [7,8].

The usage of SAT solver helped to reduce the most time consuming part of the evolutionary algorithm, the evaluation of candidate solutions. In contrast with a common fitness function based on computing a Truth table, the time of evolution was reduced by several orders depending on the circuit parameters [8].

3 Proposed Method

In order to improve the performance of the evolutionary optimizer, i.e. to increase the number of candidate solutions that can be evaluated within a period of time, we suggest to combine SAT solver with a circuit simulator which will be used to disprove the equivalence between a candidate solution and its parent. This approach is based on the assumption that the time needed to simulate a given candidate circuit using N_V ($N_V \ll 2^{n_i}$) test vectors (t_{sim}) is significantly lower than the time which is consumed by a SAT solver (t_{sat}).

The correctness of a candidate solution is determined as follows. Firstly, a circuit simulator is applied to the difference circuit between a candidate solution and its parent (difference circuit is calculated according to [8]). The simulator can use up to N_V randomly generated test vectors. If there is a test vector which evaluates the output of the difference to one, the simulator is terminated and a negative fitness value is assigned to the corresponding candidate solution. Since it is guaranteed that the candidate solution is not functionally equivalent with its parent, it is not necessary to call SAT solver to prove that fact. Otherwise, when all N_V test vectors are applied and the output of the difference evaluates to zero in all the cases, a SAT solver has to be used to prove or disprove the equivalence because the limited number of test vectors cannot guarantee that there is not a vector that differentiates the circuits.

The speedup of the proposed method combining a simulator and SAT solver can be defined as follows:

$$gain = \frac{t_{sat}}{t_{sim} + \sigma_{fail} t_{sat}} = \frac{1}{t_{sim}/t_{sat} + \sigma_{fail}}, \tag{1}$$

where $\sigma_{fail} = [0, 1]$ is a coefficient which determines the fail-rate of the simulation-based equivalence checking. The σ_{fail} may also be understood as the probability of occurrence of an undetected fault.

If we want to maximize the gain, i.e. the overall performance of the optimizer, we need to minimize not only the value of the ratio t_{sim}/t_{sat}, but also the value of σ_{fail}. Even if the simulator is e.g. 1000 times faster than SAT solver, a negligible improvement will be achieved if the value of σ_{fail} is close to one. The value of t_{sim} as well as σ_{fail} depend on the number of test vectors that can be used in the simulator to disprove the equivalence. While t_{sim} increases linearly with increasing N_V and the size of the difference entering the simulator, σ_{fail} decreases with increasing N_V. Hence, appropriate value of N_V has to be determined in order to maximize the gain.

4 Experimental Results

4.1 Benchmark Circuits

In order to evaluate the performance of the proposed method, we utilized a set of 100 randomly chosen circuits form QUIP, WLSI and ACM/SIGDA benchmark set (only circuits with 15 and more primary inputs are considered). These circuits were synthesized and optimized by ABC[1] using ‘choice’ script. The result of ABC was utilized as the input to the evolutionary optimizer.

[1] ABC is a system for sequential synthesis and verification by A. Mishchenko.

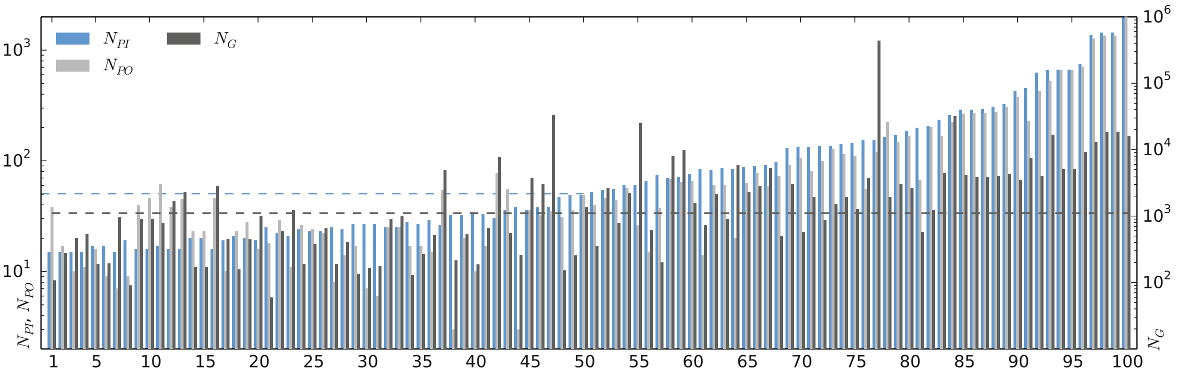


Fig. 1. The number of primary inputs (N_{PI}), primary outputs (N_{PO}) and gates (N_G, right axis) for each benchmark circuit. The X-axis contains the index of benchmark circuit. The benchmarks are arranged according to the increasing complexity expressed as $2^{N_{PI}} N_G$. Note that both Y-axes have a logarithmic scale (The list of benchmark circuits is available at http://www.fit.vutbr.cz/~vasicek/gp15).

The basic parameters of the benchmark circuits are given in Fig. 1. The circuits are arranged according to the increasing complexity. The complexity is expressed as a time needed to evaluate a candidate solution using a common fitness function (i.e. the fitness function based on a truth table). In such a case, the evaluation time is dependent on two factors: the number of primary inputs (N_{PI}), and the number of gates (N_G). As the time needed to evaluate a candidate solution increases exponentially with the increasing number of primary inputs, N_{PI} represents the key parameter which has a great impact on the total time.

The least complex circuit, 'alcom' circuit with index 1, consists of 106 gates and utilizes 15 primary inputs and 38 outputs. The most complex circuit, audio codec controller 'ac97_ctrl' with index 100, contains 16158 gates and uses 2176 inputs and 2136 outputs. One half of the benchmark circuits have more than 50 primary inputs and consist of more than 1000 gates.

4.2 Role of Neutral Mutations

The objective of the first experiment was to confirm or reject hypothesis about the importance of neutral mutations in evolutionary optimization of combinational circuits. Two variants of the mutation operator were implemented in order to evaluate the significance of neutrality. The first implementation does not impose any special limitations on the mutation operator. The only requirement is to modify the value of a randomly chosen gene to a different one (but legal). On the other hand, three restrictions are applied in the second implementation: (1) inactive gates are never modified; (2) it is not possible to connect an active gate (or primary output) to an inactive gate; (3) the gene which encodes the connection of the second input of a single-input gate is never mutated. These restrictions were introduced in order to mitigate the neutral mutations.

The CGP parameters were chosen as follows: $n_c = N_G$, $n_r = 1$, $l = N_G$, $\lambda = 1$, $h = 2$, $\Gamma = \{\text{BUF, INV, AND, OR, XOR, NAND, NOR, XNOR}\}$. These parameters were chosen according to the [8]. No redundancy in CGP encoding is used; the number of nodes is equal to the size of a benchmark circuit obtained from ABC. The goal of CGP is to minimize the number of utilized gates, i.e. the fitness value is equal to the number of active CGP nodes. The fitness function utilizes SAT solver only. In order to perform a statistical evaluation, fifteen independent evolutionary runs were executed for each benchmark circuit. Note that median value will be used to analyze the impact of a particular parameter because no Gaussian distribution can be observed among the benchmarks. The evolution is terminated after 15 min[2]. We do not use the number of evaluations as a termination condition because this number is very sensitive to the structural properties of an optimized circuit and it is impossible to determine an appropriate value in advance.

The performance of both approaches is evaluated using the number of generations (G_{impr}) that enabled an improvement of the fitness value. This parameter can be seen as a measure of mutation operator's performance (i.e. the ability to generate a candidate solution which is valid and simultaneously improved). The reason behind the usage of this metric is that the number of evaluations cannot be compared directly because the neutral mutations are detected and the created candidate solutions do not enter the time-consuming fitness evaluation procedure (it is guaranteed that they have the same fitness value as their parent) resulting in the fact that significantly more generations can be produced if the occurrence of neutral mutations is high.

Let $G = G_{valid} + G_{invalid}$ be the total number of generations where G_{valid} is the number of generations in which a valid candidate solution (i.e. functionally equivalent with a parental circuit) is generated from a parental solution by applying the mutation operator. Then, G_{valid} can be expressed as $G_{valid} = G_{impr} + G_{noimpr} + G_{neutral}$, where $G_{neutral}$ is the number of neutral mutations in the sense defined in previous paragraphs. G_{noimpr} represents the candidate solutions in which at least a single gene was changed but the fitness value remained unchanged. Note that $G_{neutral} = 0$ in the second implementation because no neutral mutations are allowed.

The evaluation of both variants of the mutation operator is shown in Fig. 2. The performance is expressed as the ratio $G_{impr}/(G_{valid} - G_{neutral})$ calculated at the end of each 15-minute evolutionary run, averaged over all fifteen runs. Despite the stochastic nature of evolutionary algorithm which leads to some variances (see the error bars in Fig. 2 showing the magnitude of standard deviation), we can conclude that the performance of both implementations is almost identical. In average, 2.34 % of valid generations were produced when the neutral mutations were enabled and 2.42 % for the opposite case. For 75 benchmarks, the variant with disabled neutral mutations performs approx. (30 ± 35) % better in average. The performance was worsened in 25 cases by approx. (9 ± 10) % in average.

According to the obtained results, we can conclude that it has no advantage to support neutral mutations in this scenario (i.e. if the goal is to minimize the

[2] A PC equipped with Intel Xeon X5670 (24 cores, 2.93 GHz, 12 MB cache), 32 GB RAM and 64-bit CentOS Linux was used.

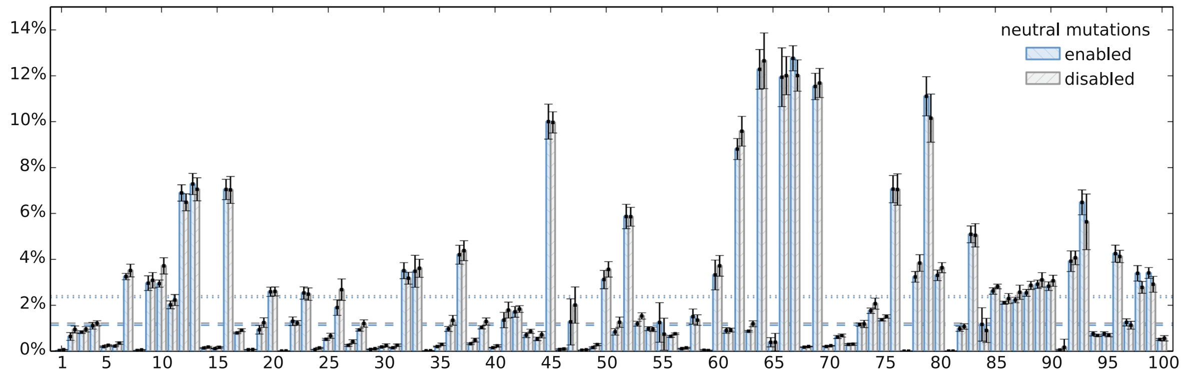


Fig. 2. The mean number of generations that enabled an improvement of fitness value when the neutral mutations were enabled (disabled). It is expressed as the ratio $G_{impr}/(G_{valid} - G_{neutral})$. The mean value obtained as an average over all benchmarks is represented by dotted line whereas the median value is depicted by dash-line.

number of gates in a fully functional circuit). In fact, the neutral mutations have a negative impact on overall performance because the probability of mutation of an active gene decreases with the increasing number of inactive genes. Even if the neutral mutations are detected and the corresponding candidate solutions do not enter the time-consuming fitness evaluation procedure, the performance of the evolutionary optimizer deteriorates as the circuit is reduced because a great portion of neutral mutations is generated.

Looking at the results shown in Fig. 2, we can identify that the performance of the mutation operator is very sensitive to the optimized circuit. One can admit that this issue could be related to the impossibility to improve the number of gates of a given benchmark circuit, but this is certainly not the case. It can be easily shown that the utilized circuits are not optimal if the number of gates is considered. Taking into account that the ratio between G_{valid} and G is approx. 0.5 % in average (see Fig. 3), there are circuits for which the mutation

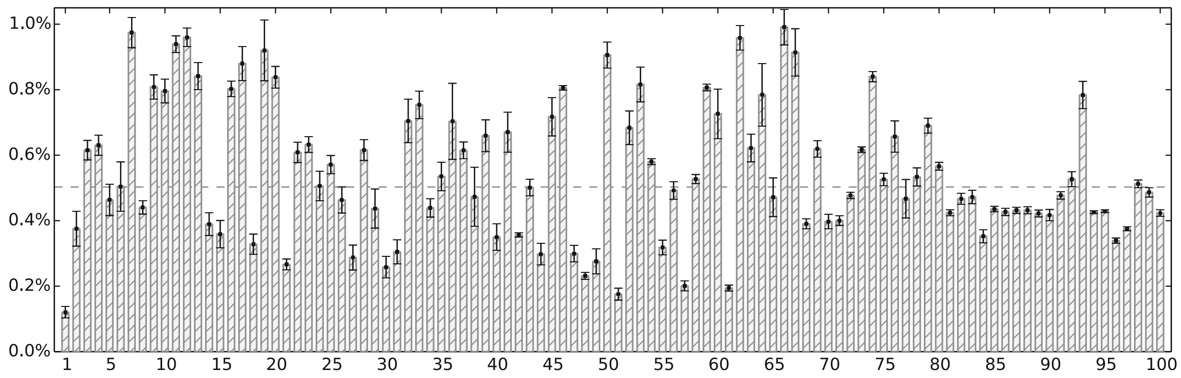


Fig. 3. The number of generations in which a valid candidate solution was produced, represented as a ratio $G_{valid}/(G_{valid} + G_{invalid})$. The results are obtained from the second implementation, where the neutral mutations are disabled. The median value is shown using a dash-line.

operator performs very poorly. Less than 0.007 % of the total number of generations enabled the improvement of the fitness value for one half of the benchmark circuits. On the other hand, there are instances showing a significantly better convergence, e.g. more than 0.12 % of the total number of generations leading to the improvement of the fitness value were produced in the case of circuit 66.

Unfortunately, there is no obvious relation between the circuit complexity (as defined in Sect. 4.1) and performance of the mutation operator. Thus, we believe that the performance of the mutation operator is in a close relation with the internal structure of an optimized circuit. Hence there are two possibilities how to improve the performance of the evolutionary optimizer. We can (a) increase the number of generations that can be evaluated within a time period and/or (b) to design a new mutation operator with better performance.

4.3 Efficiency of the Proposed Approach

To determine the value of σ_{fail} and its dependency on N_V, three experiments were performed. A 64-bit parallel simulator which is able to calculate response to 64 input combinations in a single pass was utilized. The simulator was enabled to use (a) a single pass ($N_V = 64$), (b) up to 16 passes ($N_V = 1024$), and (c) up to 32 passes ($N_V = 2048$) to disprove the equivalence. Only the cone of influence determined according to the points of mutation enters the simulator. The experimental setup and CGP parameters were the same as described in previous section. The mutation operator with suppressed neutral mutations was employed.

The obtained results are shown in Fig. 4. The value of σ_{fail} was calculated at the end of fifteen 15 min evolutionary runs. The median value of N_V can be approximated by the exponential trendline $\overline{\sigma}_{fail} \approx 3.2693 {N_V}^{-0.611}$ with R-squared equal to 0.9955. It means that σ_{fail} noticeably decreases at the beginning (i.e. for small N_V) and then, as N_V increases, the yield is smaller and smaller. In most cases, σ_{fail} is lower than 0.1 even if a single pass is used. However, there are cases with surprisingly high ratio of σ_{fail} that remains above 50 %

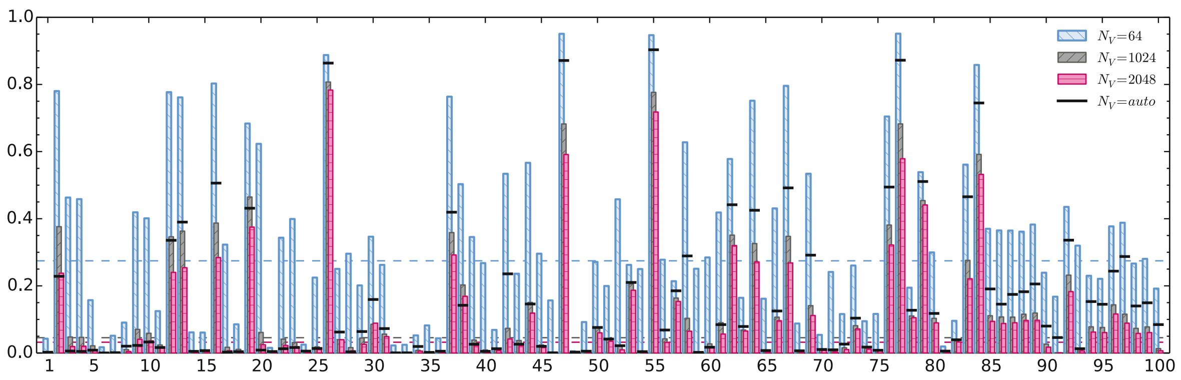


Fig. 4. Fail-rate σ_{fail} of simulation-based equivalence checking shown for various number of randomly generated test vectors (N_V) that are utilized by the circuit simulator to disprove functional equivalence between candidate solution and its parent.

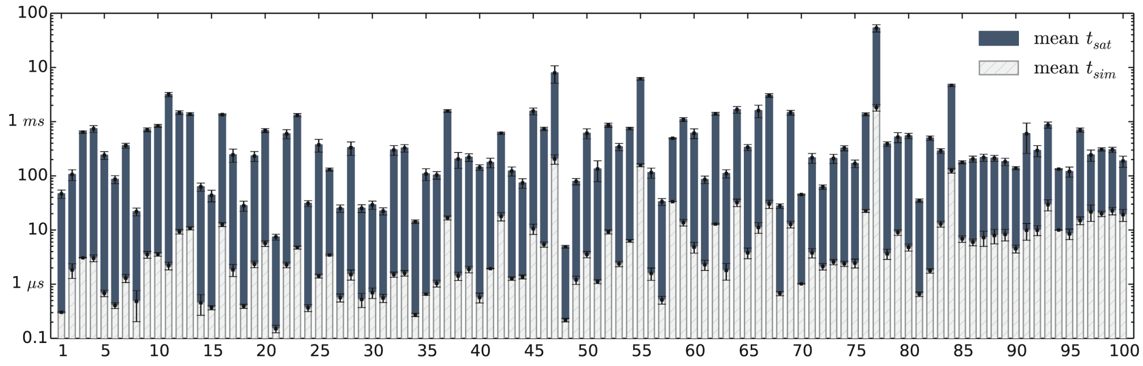


Fig. 5. Average time needed to perform equivalence checking using (a) SAT solver (see t_{sat}) and (b) simulator with a single pass (see t_{sim}).

even if 2048 randomly generated input combinations were utilized (see benchmarks 26, 47, 55, 77 and 84). Considering the parameters of those circuits (see Fig. 1), we suppose that this issue is probably related to the high number of utilized gates which may contribute to a fault masking effect.

The σ_{fail} corresponding to the number of test vectors that are used to maximize value of Eq. 1 is represented by lines labeled as $N_V = auto$ in Fig. 4. We can observe that less than 16 passes (i.e. less than 1024 test vectors) were used in most cases. These instances can easily be identified by comparing the value of σ_{fail} for $N_V = auto$ and $N_V = 1024$; the lower number of test vector implies higher σ_{fail}. Unfortunately, the ratio t_{sat}/t_{sim} remains very low for the five benchmarks discussed in previous paragraph (see Fig. 5). Hence only a few test vectors can be utilized which results in the fact that the fail-rate remains very high. Thus only a negligible speedup is expected in these cases.

The speedup of the proposed method combining SAT solver with simulator is given in Fig. 6. The speedup is calculated using the number of candidate solutions that can be evaluated within 15 min. The number of test vectors was determined adaptively during the evolution as follows. At the beginning of the evolution, a

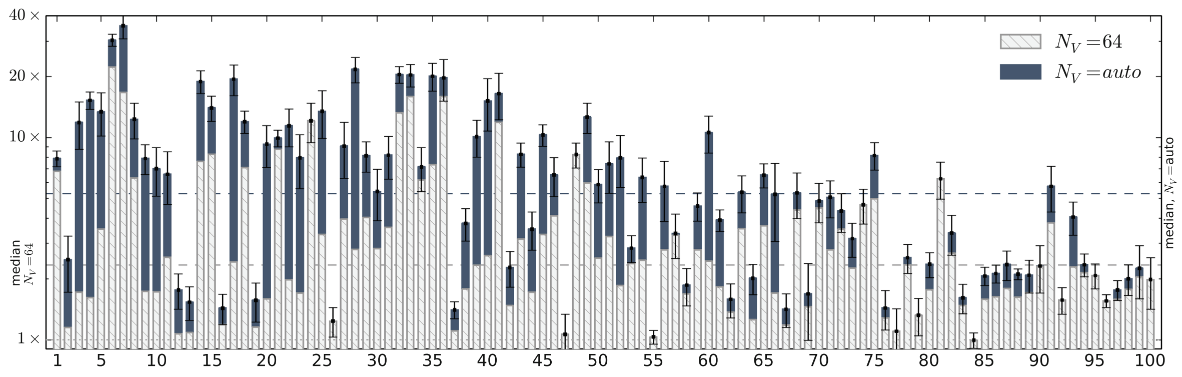


Fig. 6. Speedup of the proposed method which combines SAT-based and simulation-based equivalence checking in the fitness function. For more than 50 benchmark circuits, adaptive setting of the number of test vectors (see $N_V = auto$) increased the speedup approx. twice compared to a single-pass simulation (i.e. 64 test vectors). Note that the y-axis has a logarithmic scale.

single pass (i.e. 64 test vectors) is utilized. Then, the number of passes doubles every 10 s until a decrease in the performance is detected. Finally, the best value is determined and used. The number of test vectors is adaptively modified during evolution if there exists a different value which provides better performance.

According to the obtained results, the achieved speedup is higher than 5.28 for half of the benchmark circuits. The performance of the implementation which utilizes the adaptive number of test vectors is approximately two times higher compared to the implementation with fixed number of test vectors whose speedup factor is approx. 2.34. This finding can be considered as a very positive result since the introduction of the simulator can remarkably improve the performance of the evolutionary optimizer.

Similarly to our previous findings regarding σ_{fail}, the value of speedup noticeably varies across the benchmarks. There are cases for which the speedup factor exceeded 30. On the other hand, nearly no improvement was obtained for benchmarks 26, 47, 55, 77, and 84. According to our expectation, the speedup is close to 1.0 in these cases.

We analyzed the obtained results and identified that there is a relation between σ_{fail} and speedup. If $\sigma_{fail} \geq 0.05$, the higher σ_{fail} implies a lower speedup. However, this relation does not hold for $\sigma_{fail} < 0.05$ where the speedup varies in one order independently on the value of σ_{fail}. In addition to that, we can observe decreasing of the t_{sat}/t_{sim} ratio as the complexity of a benchmark circuit increases. Even if t_{sat} remains relative stable across the benchmarks (see Fig. 5), t_{sim} increases with the increasing complexity. The ratio t_{sat}/t_{sim} was decreasing from approx. 350 for less complex circuits to 10 for the most complex circuit. As a consequence of that, a relative small number of test vectors should be used in the simulator.

4.4 Performance of the Circuit Optimizer

The impact of the proposed method on the quality of optimization is shown in Fig. 7. The implementation which utilizes SAT solver and circuit simulator with

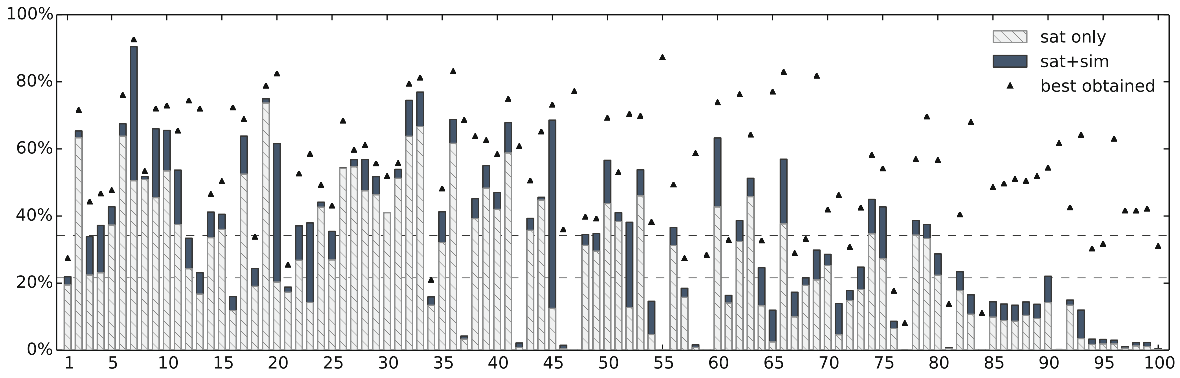


Fig. 7. Reduction of the benchmark circuits (relative to the original size) obtained after 15 min of the optimization is shown for (a) sat-based optimizer and (b) the proposed approach which combines SAT solver with simulator. The best results obtained from a 24-hour evolutionary optimization are denoted by triangles.

adaptive number of test vectors is compared against the SAT-based implementation introduced in [8]. No neutral mutations were enabled. The experimental setup is the same as used in previous section.

In all cases, the combination of a SAT solver and circuit simulator brought an improvement. The size was reduced by 13 % in average. Still, there are cases showing a very slow convergence caused mainly by the time consuming evaluation. If we compare average G_{valid} of the four aforementioned benchmarks (26, 47, 55, 77, and 84) with G_{valid} of the rest of the benchmarks, we can observe that the value is two orders of a magnitude lower. This explains why nearly no improvement was achieved within 15 min in these cases.

5 Conclusion

We introduced a new approach to the evolutionary optimization of large digital circuits which exploits the combination of a circuit simulator and a formal verification. Due to the usage of a simulator with adaptive number of test vectors, the time of evaluation was significantly reduced for 100 complex benchmark circuits in comparison with a method published in [8]. In the worst case, the time of evaluation remains the same.

In addition to that, we investigated the role of neutral mutations that are believed to be an important part of CGP. According to the obtained results, we have concluded that it has no advantage to support neutral mutations for circuit optimization (i.e. in the case that the number of gates is minimized for a fully functional circuit). This can be understood as an important result not only from theoretical but also from practical point of view because the neutral mutations in fact have negative impact on the performance of the evolutionary optimization. Our findings related to the role of neutrality correspond with observations on the evolutionary design of parity circuits [1].

The performance of the proposed method was evaluated on an extensive set of real-world benchmark circuits having tens to hundreds of inputs and consisting of hundreds to thousands of gates. For more than half of the benchmark circuits, approximately five times higher number of evaluations was performed within the same time period compared to the approach that utilizes only a formal approach. While the latter method was able to reduce the circuits by 21 % in average, the proposed method is able to reduce the circuits by 34 % using the same amount of time. Considering the fact that the runtime of the optimization process was 15 min, the obtained results are very encouraging.

We demonstrated that the circuit optimization conducted by CGP is applicable on complex real-world digital circuits. However, we simultaneously shown that there are instances for which the proposed method can bring only a marginal or none improvement in the performance. Our method is based on the assumption that evolutionary-based approach generates a large number of invalid candidate solutions that can be detected very quickly by means of applying a few test vectors on the inputs (i.e. that the time consuming formal verification can be replaced with a faster simulation-based approach). While this assumption is valid and an enormous number of invalid candidate solutions are generated during evolution, there exist circuits that are hard for the simulation-based verification. We believe that the evolutionary-based approach requires to generate a large number of candidate solutions to compensate the poor performance of the mutation operator. We observed that at least $5 \cdot 10^4$ valid candidate solutions were generated within 15 min for problem instances exhibiting a reasonable convergence. Unfortunately, approx. two orders of a magnitude (i.e. 10^6) candidate solutions have to be generated to obtain $5 \cdot 10^4$ valid candidate solutions.

One of the possibilities how to substantially improve performance of the evolutionary optimization is to orient the future research towards improving of the mutation's operator performance. Another option is to replace the randomly generated test vectors with a smart selection of test vectors which can quickly detect the inequivalence. One of the possibilities is to build a database of test vectors using the counter examples that are produced by a SAT solver during verification.

Acknowledgments. This work was supported by the Czech science foundation project 14-04197S.

References

1. Collins, M.: Finding needles in haystacks is harder with neutrality. Genet. Program. Evolvable Mach. **7**(2), 131–144 (2006)
2. Harding, S., Miller, J.F., Banzhaf, W.: Self modifying Cartesian genetic programming: parity. In: 2009 IEEE Congress on Evolutionary Computation, pp. 285–292. IEEE Press (2009)
3. Miller, J.F.: Cartesian Genetic Programming. Springer, Heidelberg (2011)
4. Shanthi, A.P., Parthasarathi, R.: Practical and scalable evolution of digital circuits. Appl. Soft Comput. **9**(2), 618–624 (2009)
5. Stomeo, E., Kalganova, T., Lambert, C.: Generalized disjunction decomposition for evolvable hardware. IEEE Trans. Syst. Man Cybern. Part B **36**(5), 1024–1043 (2006)
6. Vasicek, Z., Sekanina, L.: Hardware accelerators for Cartesian genetic programming. In: O'Neill, M., Vanneschi, L., Gustafson, S., Esparcia Alcázar, A.I., De Falco, I., Della Cioppa, A., Tarantino, E. (eds.) EuroGP 2008. LNCS, vol. 4971, pp. 230–241. Springer, Heidelberg (2008)
7. Vasicek, Z., Sekanina, L.: Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. Genet. Program. Evolvable Mach. **12**(3), 305–327 (2011)
8. Vasicek, Z., Sekanina, L.: A global postsynthesis optimization method for combinational circuits. In: Proceedings of the Design, Automation and Test in Europe, DATE, pp. 1525–1528. IEEE Computer Society (2011)
9. Walker, J.A., Miller, J.F.: The automatic acquisition, evolution and re-use of modules in Cartesian genetic programming. IEEE Trans. Evol. Comput. **12**(4), 397–417 (2008)

Appendix C

Evolutionary Approach to Approximate Digital Circuits Design

VASICEK, Zdenek and SEKANINA, Lukas. “Evolutionary Approach to Approximate Digital Circuits Design”. In: *IEEE Transactions on Evolutionary Computation* 19.3 (2015), pp. 432–444.

IF=5.908 (2015), contribution of the author of the thesis: 50%

Referenced on pages: 26, 28, 29

Evolutionary Approach to Approximate Digital Circuits Design

Zdenek Vasicek and Lukas Sekanina, *Senior Member, IEEE*

***Abstract*—In approximate computing, the requirement of perfect functional behavior can be relaxed because some applications are inherently error resilient. Approximate circuits, which fall into the approximate computing paradigm, are designed in such a way that they do not fully implement the logic behavior given by the specification and, hence, their accuracy can be exchanged for lower area, delay or power consumption. In order to automate the design process, we propose to evolve approximate digital circuits that show a minimal error for a supplied amount of resources. The design process, which is based on Cartesian genetic programming (CGP), can be repeated many times in order to obtain various tradeoffs between the accuracy and area. A heuristic seeding mechanism is introduced to CGP, which allows for improving not only the quality of evolved circuits, but also reducing the time of evolution. The efficiency of the proposed method is evaluated for the gate as well as the functional level evolution. In particular, approximate multipliers and median circuits that show very good parameters in comparison with other available implementations were constructed by means of the proposed method.**

***Index Terms*—Approximate computing, Cartesian genetic programming (CGP), digital circuits, population seeding.**

I. Introduction

APPROXIMATE computing is a new design paradigm emerging as a response to the never-ending need for performance and energy efficiency of computing systems [1]. It exploits the fact that the requirement of perfect functional behavior (i.e., accuracy) can be relaxed because some applications are inherently error resilient. The errors are not recognizable as human perception capabilities are limited (e.g., in multimedia applications), no golden solution is available for validation of results (e.g., in data mining applications), or users are willing to accept some inaccuracies (e.g., when the battery of a mobile phone is almost depleted, but at least a basic functionality is still requested). Therefore, this accuracy can be used as a design metric, traded for area, delay, throughput, or power consumption.

In approximate computing systems, approximations can be introduced at all design levels, starting from the circuit via the architecture and operating system to programming language. Examples of applications in which the principles of approximate computing are utilized range from inaccurate arithmetic circuits (e.g., adders [2], multipliers [3]) via high-level processing blocks (e.g., image compression [3], discrete cosine transform, finite and infinite impulse response filters [4]) to general purpose approximate computing machines [5] and programming languages [6]. The circuits that are intentionally designed in such a way that the specification is not met in terms of functionality and some savings are expected in terms of energy, performance, or area are called approximate circuits.

Manuscript received July 26, 2013; revised May 30, 2014 and June 25, 2014; accepted June 27, 2014. Date of publication July 8, 2014; date of current version May 27, 2015. This work was supported in part by IT4Innovations Centre of Excellence under Grant CZ.1.05/1.1.00/02.0070 and in part by Brno University of Technology under Grant FIT-S-14-2297.

The authors are with the Faculty of Information Technology, IT4Innovations Centre of Excellence, Brno University of Technology, Brno 61266, Czech Republic (e-mail: vasicek@fit.vutbr.cz; sekanina@fit.vutbr.cz).

Digital Object Identifier 10.1109/TEVC.2014.2336175

Approximate computing as a field is in an early stage of development and without an established methodology. Approximate circuits have initially been constructed manually, by removing those parts of existing fully functional designs that did not contribute to the result significantly [3]. The current trend is to create general design methods (such as SALSA [4] and SASIMI [7]) that are capable of constructing approximate circuits that never exceed a predefined error. These error-oriented approaches, however, represent only one of the possible approaches in order to approximate circuits design.

Evolutionary circuit design was successful in the task of designing a specific class of electronic circuits which has been documented in numerous survey articles (see [8] and [9]). The aim of this paper is to show that the approximate circuit design methodology based on principles of evolutionary design can produce efficient and competitive approximate gate-level as well as functional level combinational circuits. Because of the nature of approximate circuits (in fact, partially working circuits are sought) and principles of evolutionary circuit design (evolutionary-based improving of partially working circuits), we expect a synergy effect that could lead to establishing an evolutionary design as a competitive design method for approximate circuits.

In our previous work, we took advantage of the fact that the evolutionary design always provides a partially working solution even when resources needed for constructing a fully functional solution are not available [10]. It has to be noted that conventional methods do not usually provide any result when allocated resources are insufficient. As power consumption is often highly correlated with occupied resources, we can evolve a partially working circuit using constrained resources and assume that the circuit's power consumption will be reduced.

This idea is further elaborated as follows. Let n be the (minimum) number of gates required to implement a given logic circuit. The approximate circuit is created by means of randomly seeded Cartesian genetic programming (CGP) whose objective is to minimize a given error function and which can use up to m gates ($m < n$). If various other approximations are requested, CGP is executed multiple times with a gradually reduced amount of available gates. The user thus obtains a set of approximate combinational circuits, each of which typically exhibits different tradeoffs between the functionality and the number of gates. The proposed design approach can be considered as an area-oriented method because the user can control the used area (and so power consumption) more comfortably than by means of the error-oriented methods. Another important contribution of this article is a new method of seeding the initial population of CGP, which enables us to significantly reduce the time of evolution.

In order to demonstrate a wider applicability of our approach, the proposed method will be evaluated for gate-level as well as functional-level circuits. It should be noted that systematic methods have only been introduced for the bit (gate) level design of approximate circuits. Hence two case studies will be reported: the design of approximate combinational parallel multipliers (the gate level) and the design of a median computing circuit (the functional level). We will study the tradeoff between the correctness, area, and power consumption for 2-bit, 3-bit, and 4-bit multipliers. These small multipliers will be used as building blocks for larger multipliers, and again, the correctness will be traded for power consumption and area. The median computing circuit is a key component for median filters in image processing. It is expected that approximate median circuits can lead to a significant area reduction while the error of filtering remains small. In summary, the key contributions of this article are as follows.

1) We propose a new methodology for approximate circuit design that exploits the area-oriented design approach and CGP seeded by heuristically created approximate circuits.
2) We propose to extend the concept of approximate circuit evolution from the gate level to the functional level.
3) We present novel implementations of approximate combinational multipliers created by CGP. These multipliers show very good parameters in comparison with similar multipliers reported in the literature.
4) We present novel implementations of approximate median circuits created by CGP.

The rest of the paper is organized as follows. Section II surveys relevant research in areas of approximate circuits and evolutionary circuit design. The proposed design methodology is introduced in Section III. An experimental framework is presented together with obtained results in Section IV. After discussing the impact of this paper, the conclusion is given in Section V.

II. Related Work

Only a few papers on evolutionary circuit design have directly or indirectly addressed the problem of approximate circuit design. Before introducing them in Section II-B we will give an overview of current (conventional) approximate circuit design techniques in Section II-A.

A. Approximate Circuits: Overview

Power consumption reduction is one of the key challenges of the current chip design industry. Conventional approaches to power reduction of digital circuits are applied at all design levels, starting from the architecture via the circuit to the technology [11]. Further reductions can be obtained by approximating the original circuit function by a new one whose implementation is more energy efficient. The requirement on functional equivalence between the specification and implementation is thus relaxed in order to minimize energy consumption, accelerate computations, or reduce the area on a chip. The concept of approximate circuits is similar to probabilistic circuits that take into account the importance of bits of the circuit's output with respect to the complexity of their implementation [12]. However, approximate computing does not involve assumptions on the stochastic nature of any underlying processes implementing the system [1].

The next subsections will present basic design techniques (over-scaling and functional approximation), systematic design methodologies, and error metrics used in approximate circuit designs.

1) Over-Scaling: In the case of over-scaling, circuits are designed to be working perfectly under a normal environment. However, their energy consumption can be reduced by voltage over-scaling (i.e., using deliberately lower power supply voltage in which the circuit is known to occasionally produce erroneous outputs). Similarly, performance can be increased when the circuit is over-clocked. Timing-induced errors are due to the fact that some paths in the circuit fail to meet the delay constraints. The combination of scaling the supply voltage and clock frequency is known as dynamic voltage scaling.

2) Functional Approximation: Functional approximation means that the circuit is designed in such a way that it does not fully implement the logic behavior given by the specification. A simple method is to reduce the precision of computations in the case of arithmetic circuits by ignoring the least significant bits. However, only insignificant area savings can be obtained for some key circuits such as multipliers. Other methods adopt logic synthesis scenarios in which implementations that satisfy the specification almost perfectly are sought, but the amount of resources is significantly reduced (see [2] and [7]).

For example, a two-bit multiplier was manually constructed, which consists of five gates only and exhibits a delay of $2d$, where d is a unit delay. Its output is correct for 15 out of 16 possible inputs. A usual conventional solution requires eight gates and exhibits a delay of $3d$. This approximate multiplier has been used in larger approximate multipliers and then employed in approximate image processing applications [3].

3) Systematic Design Methodologies: As the manual redesign is not a universal and efficient method, systematic methods to synthesize approximate circuits are currently being developed.

The systematic methodology for automatic logic synthesis of approximate circuits (SALSA) starts with an RT-level description of the exact version of the circuit and an error constraint that specifies the type and amount of error that the implementation can exhibit [4]. The methodology introduces the so-called Q-function, which takes the outputs from both the original and approximate circuits and decides if the quality constraints are satisfied. The Q-function outputs a single Boolean value. The SALSA algorithm attempts to modify the approximate circuit with the goal of keeping the output of the Q-function unchanged.

Another systematic approach, substitute-and-simplify (SASIMI), tries to identify signal pairs in the circuit that exhibit the same value with a high probability, and substitutes one for the other [7]. These substitutions introduce functional approximations. Unused logic can be eliminated from the circuit, which results in area and power savings. The method is combined with technology-level optimizations such as downsizing of gates (i.e., creating smaller than normally sized gates to reduce power consumption, in exchange for increased delay) on critical paths and voltage over-scaling, which results in additional significant area and power savings.

SASIMI and SALSA are very new methods and, unfortunately, are not currently available to the public.

4) Error Metrics: The above methods are error-oriented in the sense that all logic optimizations leading to an approximate solution are constrained by a predefined error criterion. The error can be expressed by various metrics such as worst case error, average error, and error probability [13]. The design process has to be repeated when a new error criterion is established.

B. Evolutionary Circuit Design

Recent surveys on evolutionary circuit design (see [8] and [9]) clearly demonstrate that although some evolved implementations of target circuits can be considered as innovative, the evolutionary design approach fails in producing useful implementations of complex circuits. In order to at least partially eliminate this disadvantage, various approaches have been proposed to improve the problem representation and genetic operators (such as functional level representations [14], [15], decomposition [16], and developmental encodings [17]) and accelerate the fitness computation (such as partial evaluation [18], formal functional equivalence checking [19], and phenotype precompilation [20]).

1) Previous Works Related on Approximate Circuits: There are some examples of evolutionary circuit design that could be considered as approximate circuit design. For example, Miller evolved finite impulse response filters at the gate level where functionality was traded for area [21]. In fault-tolerant applications, if a critical number of elements is damaged, the original function cannot fully be recovered; however, a partial functionality can be obtained by means of evolutionary design. This concept has been surveyed in [22]. In another research, Kneiper *et al.* investigated the robustness of evolved classifiers [23]. A classifier system was reported, which is able to cope with changing resources at run-time. During optimization, the number of pattern matching elements was modified and its influence on classification accuracy was studied (i.e., there is a tradeoff between the classification accuracy and area).

Thompson's famous evolutionary design of a tone discriminator circuit in the XC6216 field-programmable gate array (FPGA) belongs to this class of applications too. Thompson's evolutionary algorithm discovered a tone discriminator requiring significantly fewer resources than usual solutions would occupy in the same FPGA [24]. Though the evolved discriminator was fully functional, its robustness was limited. Higher sensitivity to fluctuations in the environment (external temperature, power supply voltage) and dependability on a particular piece of FPGA were reported. Hence we can observe a tradeoff between the robustness and the amount of resources in the FPGA.

All of these approaches and applications have something in common with approximate circuits. None of them, however, has fully exploited the capability of evolutionary design as a systematic method for an approximate circuit design.

2) Direct Evolution of Approximate Circuits: Finally, this section summarizes our previous work on evolutionary design of approximate circuits.

In [10] we evolved approximate implementations of small combinational circuits (3-bit and 4-bit adders and single output circuits) using randomly seeded CGP operating at the gate level. In order to provide solutions for every possible number of gates, CGP was repeatedly executed with gradually reduced resources available for implementation. The objective was to minimize the mean absolute error with respect to a fully functional circuit. Because the utilized power estimation algorithm (which is embedded into the SIS tool [25]) is very time consuming, it has not been included in the fitness function directly. Power consumption was calculated at the end of evolution for the best evolved approximate circuits.

An inherently multiobjective approach to evolutionary design of approximate multiplierless multiple constant multipliers (MCMs) was proposed in [26]. Three design objectives—accuracy, area, and delay—were optimized by multiobjective CGP, where the area was inexpensively estimated as the number of utilized components and delay as the number of components along the longest path between the input and the output.

Both approaches utilized randomly generated initial populations, which led to relatively time consuming evolutionary runs. Seeding the initial population by suitable pregenerated designs is one contribution of our work reported in the following sections. Another feature is that for circuits from papers [10], [26], we could check in the fitness function their responses for all possible input combinations, which is impossible for complex circuits such as median circuits.

III. Proposed Method

After emphasizing key features of the current approach to approximate circuit design, this section introduces the overall idea of the proposed method, the utilized evolutionary algorithm, and the heuristic population-seeding procedure.

A. Initial Considerations

Existing systematic approximate circuit synthesis methods (such as [4] and [7]) always begin with a fully functional circuit C and a given quality constraint (acceptable error) e. Then C undergoes the approximating procedure and an approximate circuit C_1 is generated. It is ensured that the predefined error e is not exceeded by C_1. As the acceptable error (and a corresponding power consumption reduction) can be difficult to define for a given application in advance, the design process is usually repeated for several error values $e_2, e_3, \dots, e_k$, yielding approximate circuits $C_2, C_3, \dots, C_k$. The solution that exhibits the most suitable tradeoff between design objectives is then the resulting approximate circuit. However, the area, power consumption, and delay are not directly under the control of the approximating procedure.

This is inherently a multiobjective circuit design problem that could be solved by a suitable multiobjective evolutionary algorithm (MOEA); for example, algorithms reported for evolution of conventional circuits in [26] and [27] are based on NSGA-II [28]. It is expected that MOEAs will have difficulty with delivering really compact approximate circuits for complex problem instances because of the following reasons.

1) Evolutionary design of non-trivial combinational circuits (e.g., 4-bit multipliers) from scratch is a difficult problem. Only a small fraction of runs usually produce a working circuit because corresponding fitness landscapes are very rugged [29].
2) It is even harder to evolve a working circuit (e.g., 4-bit multiplier) that is better than a conventional design according to a chosen criterion (i.e., the number of gates in our case) [30].
3) A reasonably reliable estimate of power consumption, which is important for building trustworthy Pareto fronts in MOEA, can be very time consuming for complex circuits. For example, while the evaluation of a candidate 4-bit multiplier takes 35 μs, power consumption simulation by SIS requires 0.59 s (average numbers calculated on a 3 GHz processor are given).

Another difficulty lies in the scalability problem of the evolutionary circuit design. In this paper, we adopted two approaches: 1) complex approximate median circuits are evolved by means of the functional-level evolution and 2) in the case of gate-level circuits, we focus on arithmetic circuits and adopt the approach introduced in [3] in which relatively small approximate circuits are used as building blocks of complex approximate circuits. In our case, these small approximate circuits are evolved by CGP.

B. Approximate Circuit Evolution

The main features of the proposed area-oriented method, which address the mentioned problems, are as follows.

1) The direct control of the resulting area (and possibly power consumption) could be very useful for some application scenarios (e.g., computing with the minimum error for a given power budget in a mobile phone). Hence the proposed method generates approximate circuits as a function of the area rather than the error. This area-oriented approach cannot be accomplished by conventional circuit design tools because they do not provide any solution when available resources are insufficient.
2) The proposed method works as follows. Let us suppose that P is a procedure capable of creating an approximate version of a fully functional circuit C, which consists of n components (gates). P is employed to construct an approximate circuit C_1 using m_1 components with the aim of minimizing the predefined error criterion. This approximation exhibits the error e_1. Similar to error-oriented methods, such as SALSA and SASIMI, the design procedure can be repeated; however, here it is for various number of gates (not for various errors) in order to obtain different tradeoffs among design objectives. Approximate circuits $C_2, C_3, \dots, C_k$ are then constructed by P wherein $m_2, m_3, \dots, m_k$ gates are supplied; m_k is the number of gates in the smallest required approximation of C. It is expected that the resulting errors are $e_1 \leq e_2 \leq \cdots \leq e_k$. If m is successively $n-1, n-2, \dots, 2$, and 1, an approximate circuit is constructed for every possible number of gates.
3) In order to implement P, from available evolutionary circuit design methods we choose a single-objective CGP that enables the gate as well as functional level evolution [31]. Multiple runs of CGP are performed for a given amount of resources in order to find a circuit that exhibits the smallest possible error. Multiobjective NSGA-II-based CGP [26] will be used for comparative purposes in Section IV.
4) The following features of the proposed method enable us to accelerate the whole design process.
    a) The initial population is seeded by approximate circuits (created according to Section III-E) in order to find much better solutions than a randomly seeded CGP.
    b) Power consumption is computed only for selected best circuits at the end of CGP runs.
    c) Fitness evaluation exploits the idea of parallel simulation of candidate circuits and circuit translation to the binary machine code [20].
    d) Multiple runs are executed on a computer cluster (p runs on p processors).

C. Cartesian Genetic Programming

CGP and its various versions are probably the most popular methods for the evolutionary circuit design [30], [31]. In this paper, we utilize the standard CGP for combinational circuit evolution with a few modifications, as explained in the following paragraphs.

1) Circuit Representation in the Chromosome: A candidate circuit is modeled by means of an array of processing nodes arranged in n_c columns and n_r rows. The processing elements can be either elementary gates or functional level components such as adders, comparators, and shifters. The $n_c.n_r$ product is constrained by the maximum number of available nodes in the case of approximate circuit evolution.

The set of functions implemented by processing elements will be denoted Γ. The circuit utilizes n_i primary inputs and n_o primary outputs. All signals are defined over b bits, where $b = 1$ for the gate level evolution.

Primary inputs and processing node outputs are labeled $0, 1, \dots, n_i - 1$ and $n_i, n_i + 1, \dots, n_i + n_c.n_r - 1$, respectively.

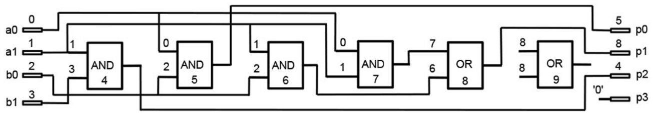


Fig. 1. Candidate 2-bit multiplier, with inputs $b_1b_0a_1a_0$ and outputs $p_3p_2p_1p_0$, represented by CGP with parameters: $n_i = n_o = 4$, $n_c = 6$, $n_r = 1$, $l = 4$, $\Gamma = \{0^{AND}, 1^{OR}\}$. Chromosome: 1, 3, $\underline{0}$; 0, 2, $\underline{0}$; 1, 2, $\underline{0}$; 0, 1, $\underline{0}$; 7, 6, $\underline{1}$; 8, 8, $\underline{1}$; 5, 8, 4, '0'.

Each node input can be connected either to the output of a gate placed in the previous l columns or to one of the primary circuit inputs. A candidate solution consisting of two-input nodes is represented in the chromosome by $n_c.n_r$ triplets $(x_1, x_2, \underline{\psi})$ determining for each processing node its function $\underline{\psi}$, and addresses of nodes x_1 and x_2 which its inputs are connected to. The last part of the chromosome contains n_o integers specifying either the nodes, to which the primary outputs are connected, or logic constants ('0' and '1'), which can be directly connected to the primary output. The support of logic constants at the primary outputs is crucial for evolving some approximate circuits.

In order to illustrate the CGP encoding in Fig. 1, we choose the approximate 5-gate multiplier discussed in Section II-A2. One important feature of CGP is that not all gates have to be included in the phenotype (e.g., gate 9). The CGP encoding is redundant; according to some studies [32], this enables us to improve the quality of the search.

D. Fitness Function

The goal of evolution is to maximize the functionality of approximate circuits whose size is constrained by the $n_c.n_r$ product. The fitness is then defined as error to be minimized as

$$f = \sum_{j=1}^{K} |y(j) - t(j)| \tag{1}$$

where y is candidate circuit's n_o-bit response and t is target response. The number of fitness cases is $K = 2^{n_i}$, because we have to evaluate circuit responses for all possible combinations of operands for arithmetic circuits. This definition of the fitness function is preferred over the Hamming distance based function because a better performance has been reported in in [10].

In the functional level evolution, the design problem is often understood as a symbolic regression problem. Then, K is the number of fitness cases in the training set.

1) Search Algorithm: We will use the $(1+\lambda)$ search method as recommended in [31].

1) The initial population of the size $1 + \lambda$ is created.
2) The fitness function f is called for each candidate circuit.
3) The highest-scored candidate circuit is selected as the new parent. It has to be noted that the previous parent α is never selected as the new parent if there are more individuals with fitness $f(\alpha)$ and $f(\alpha)$ is the best fitness value in a given population [31].
4) By applying a point mutation, λ offspring individuals are generated from the parent. In this type of mutation, h genes (integers) undergo a mutation.
5) Steps 2–4 are repeated until the termination condition is not satisfied.

E. Heuristic Population Seeding

Let C be a fully functional circuit consisting of n two-input gates. Let us suppose that CGP has to minimize the error (f) and only up to $n - 1$ gates can be utilized. The proposed heuristic for seeding the initial population is based on a local search and works as follows.

Every single gate of C is independently replaced by a wire connection (the upper input is connected to the output of the gate), which results in n approximate circuits consisting of $n - 1$ gates. The fitness values are then calculated for all n circuits. The whole procedure is repeated, but now the lower input is connected to the output for all the gates. In total, $2n$ new approximate versions of C, each containing $n - 1$ gates, are obtained. The circuit producing the smallest error is taken as the seed for CGP.

A natural extension of this heuristic for a circuit in which n gates have to be reduced to $n-k$ gates consists of: 1) a random selection of k gates and their replacement by wire connections; 2) calculating the fitness value of the modified circuit; and 3) repeating steps 1) and 2) N times (where N is a suitable constant) and outputting the circuit with the best fitness value. This approach is suitable for complex circuits (thousands of gates or more) in which modifying all the gates could be very time consuming.

F. Embedding the Heuristic Into CGP

Providing a single approximate circuit is not usually the most valuable output of approximate circuit design methods. Designers are looking for various tradeoffs among the design objectives. In order to find approximate circuits for every possible number of gates, the proposed approximate circuit design flow will call CGP several times. We have developed two approaches for embedding the heuristic into CGP in order to obtain approximate circuits containing $n - 1, n - 2, \ldots, 2, 1$ gates. Together with the random population seeding, we thus propose and compare the following three scenarios for seeding the initial populations of CGP.

1) RS: All initial populations are randomly generated.
2) HS1: Heuristic seeding, according to Section III-E, in which the best result of CGP containing m gates is used by the heuristic to build a new seed containing $m - 1$ gates. Applying HS1 means that each CGP run is, therefore, interleaved by a single run of the heuristic procedure removing just one gate from the best evolved solution.
3) HS2: Heuristic seeding, according to Section III-E, in which the heuristic is applied iteratively on its previous result in order to build a set of seeds containing $n - 1$, $n - 2, \ldots, 1$ gates. This means that all requested seeds are firstly generated by the heuristic and independent CGP runs are then initialized using the created seeds.

The initial, fully functional solution, with which the heuristics HS1 and HS2 begin, is a conventional implementation of target circuits.

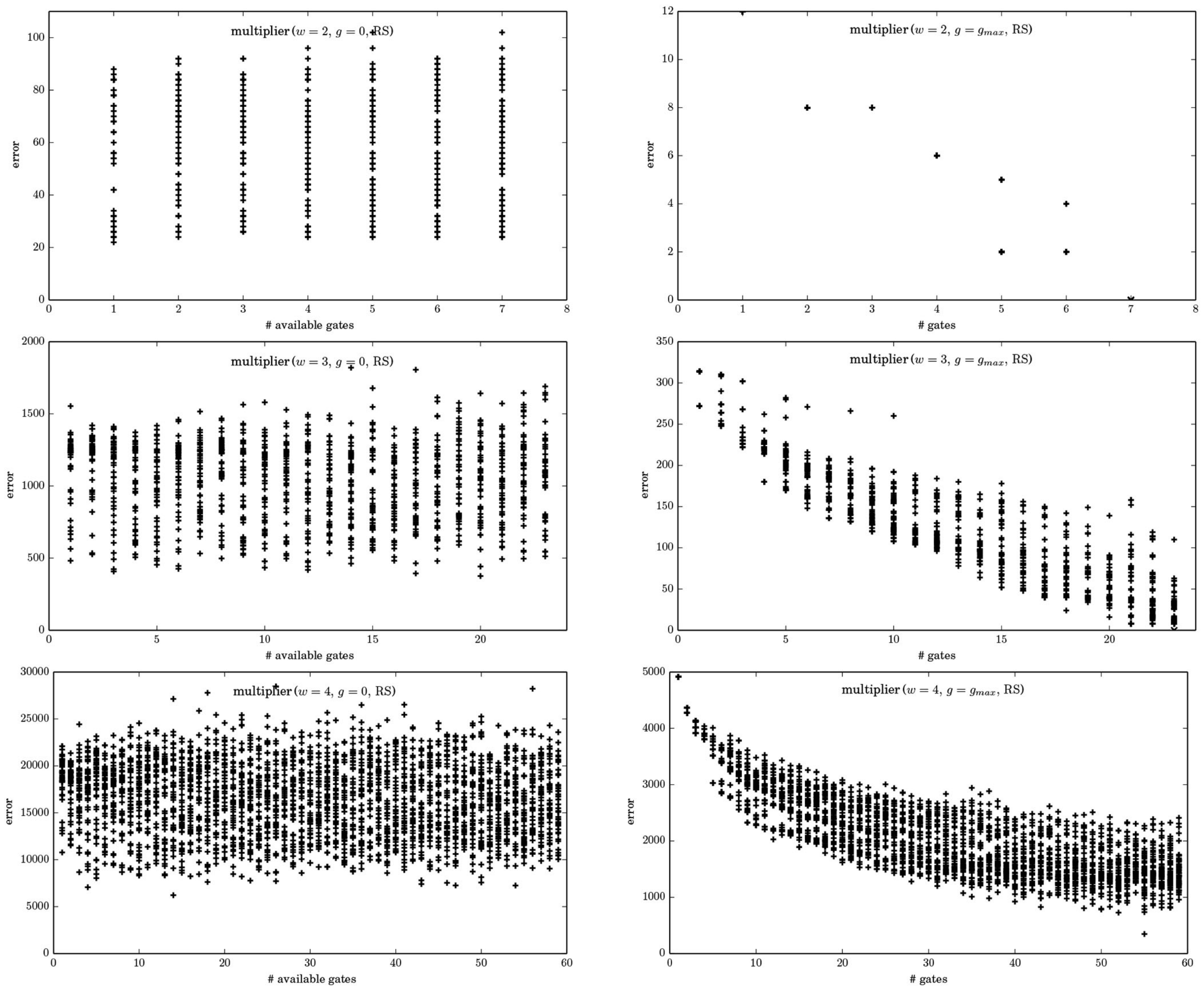


Fig. 2. Error of the randomly generated seeds (left column) and error of the evolved solutions (right column) for 2-bit, 3-bit and 4-bit approximate multiplier in the RS scenario.

TABLE I
RELATIVE ERROR ϵ_{mrt} [%] FOR VARIOUS BIT WIDTHS w AND DIFFERENT NUMBER OF CGP COLUMNS n_c FOR TWO SELECTED ARITHMETIC CIRCUITS (200 INDEPENDENT RUNS)

multiplier			adder		
w	$n_c = 1$	$n_c = n_{bst}$	w	$n_c = 1$	$n_c = n_{bst}$
2	30.440	23.702	2	18.566	20.477
3	28.943	23.877	3	18.671	22.431
4	28.434	24.090	4	19.467	22.439
5	28.540	25.246	5	19.856	23.100
6	28.874	25.531	6	20.774	23.145
7	29.339	25.628	7	21.609	23.780
8	29.498	25.426	8	22.453	24.063

IV. EXPERIMENTAL RESULTS

Several papers have addressed the evolutionary design of small combinational parallel w-bit multipliers with the goal of minimizing the number of gates (see [30] and [33]). This task is considered as a very difficult benchmark for evolutionary circuit design methods; much more difficult than the evolution of adders, multiplexers, or parity circuits. Hence results competitive with conventional synthesis algorithms were reported for up to 4-bit multipliers. This section extends these results by considering approximate versions of the multiplier circuits. Moreover, it presents a comparison of the proposed single objective CGP with MOEA. The second case study deals with the synthesis and optimization of approximate median circuits with nine inputs (9-median, for short) and 25 inputs (25-median) working over 8 bits. Results will be reported for every possible number of gates (components) in order to show all available tradeoffs.

A. Approximate Multipliers

The goal of CGP is to design a multiplier showing the lowest possible error for a given number of gates. The error is expressed according to Eq. 1. The CGP parameters are initialized as $n_r = 1$, $l = n_c$, $\lambda = 4$, $h = 5\%$, and $\Gamma = \{$BUF, NOT, AND, OR, XOR, NAND, NOR, XNOR$\}$, where BUF stands

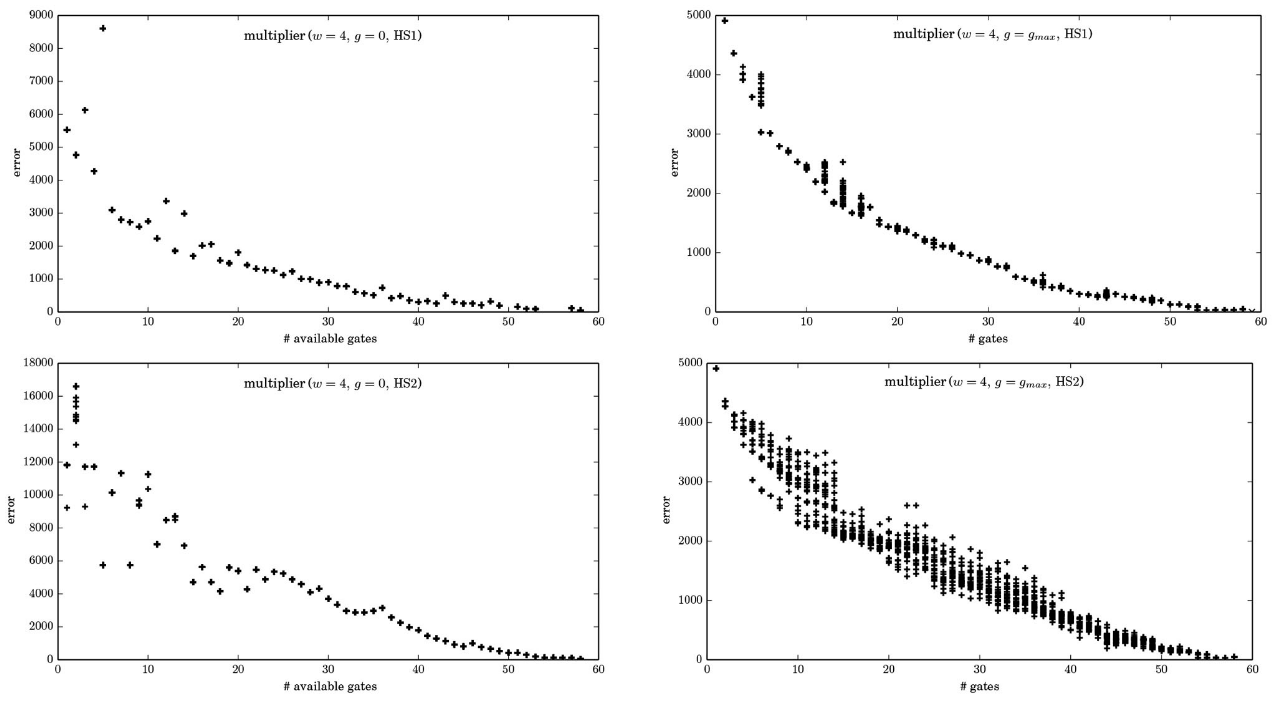


Fig. 3. Error of the seeds (left column) and error of the evolved solutions (right column) for 4-bit approximate multiplier in HS1 (top) and HS2 (bottom) scenarios.

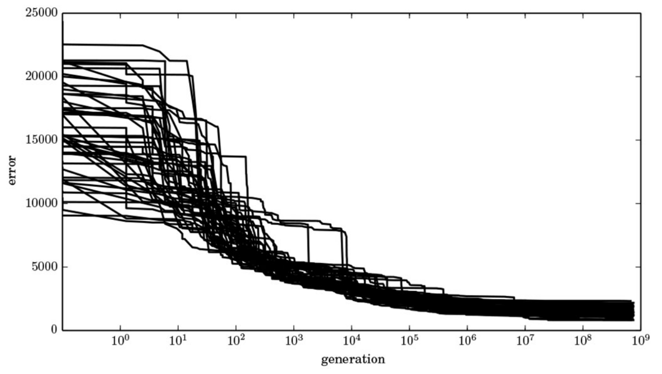


Fig. 4. Convergence curves for the best 4-bit multipliers in all 50 evolutionary runs ($n_c = 58$, RS strategy).

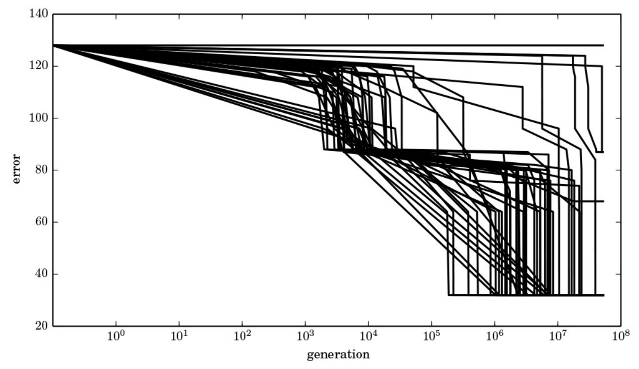


Fig. 5. Convergence curves for the best 4-bit multipliers in all 50 evolutionary runs ($n_c = 58$, HS1 strategy).

for an identity function. The setting of the CGP parameters is based on experiments conducted in our previous research [10]. The evolutionary algorithm stops when the predefined number of generations $g_{\max}$ is exhausted. All the experiments were performed on a cluster of computation nodes equipped with Intel Xeon processors running at 3 GHz.

CGP, seeded by the RS strategy, is applied as follows. Let n_{bst} be the number of two-input gates required to implement a conventional fully functional multiplier. All experiments were conducted for $n_{bst} = 7$, 23, and 59, corresponding to the 2-bit, 3-bit, and 4-bit multiplier constructed according to the conventional Ripple-Carry-Array multipliers. For each w-bit multiplier, we performed n_{bst} independent experiments consisting of 50 independent CGP runs each. The parameter $n_c = n_{bst}, n_{bst} - 1, \ldots, 1$ is used in these experiments. The initial population is always randomly generated. The maximum number of generations is limited to $g_{\max} = 800 \cdot 10^6$, $500 \cdot 10^6$ and $350 \cdot 10^6$ for the 4-bit, 3-bit, and 2-bit multipliers (which is consistent with [30]), corresponding to a single evolutionary run of 24 h, 3 h, and 50 min, respectively.

In the case of the HS1 and HS2 strategies, all the evolutionary runs of the first experiment (when $n_c = n_{bst} - 1$) are seeded with the same initial circuit obtained from a conventional solution by removing exactly one gate. Seeding the initial population means that the number of generations can be reduced (see below). Hence we chose $g_{\max} = 200 \cdot 10^6$, $100 \cdot 10^6$, and $100 \cdot 10^6$ for 4-bit, 3-bit, and 2-bit multipliers, respectively. The corresponding runtime of a single CGP run is 2 h, 30 min, and 30 min, respectively.

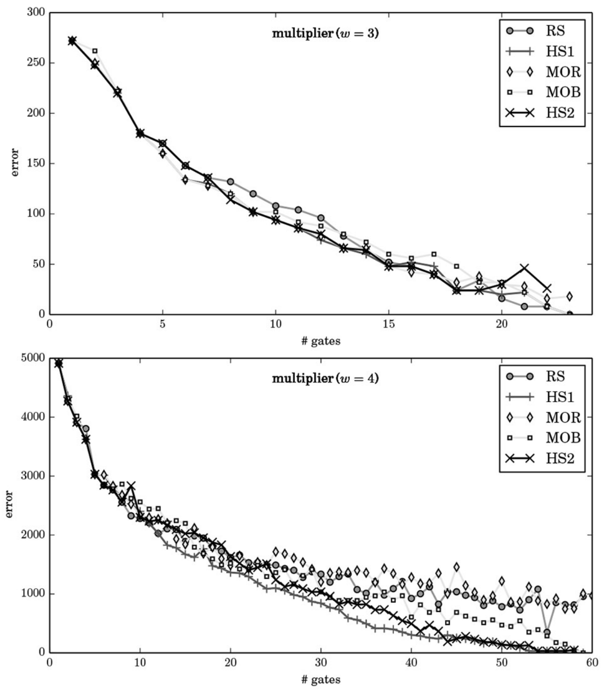


Fig. 6. Error of the best 3-bit (top) and 4-bit (bottom) approximate multipliers obtained by the proposed seeding strategies and in the multiobjective optimization scenario (MOR, MOB).

TABLE II
PARAMETERS OF THE BEST FULLY FUNCTIONAL MULTIPLIERS

		power [uW]			area [-]			delay [ns]		
w	n_c	best	worst	mean	best	worst	mean	best	worst	mean
2	7	44.5	65.3	51.5 $\pm$ 5.2	8.3	10.7	9.4 $\pm$ 0.8	4.8	13.5	8.4 $\pm$ 1.8
3	23	220.4	248.9	235.9 $\pm$ 8.4	32.3	34.6	33.1 $\pm$ 0.7	20.9	26.8	24.3 $\pm$ 1.4
4	59	790.1	1425.4	1119.6 $\pm$ 193.3	83.9	87.5	86.2 $\pm$ 1.3	47.8	55.0	50.5 $\pm$ 2.5

1) Random Seeding: Fig. 2 depicts fitness values of the randomly generated seeds and resulting fitness values at the end of evolution for all approximate multipliers in all runs. The column on the left in Fig. 2 shows that the fitness values of seeds are distributed similarly for all problem instances, independent of the number of gates. The fitness values of evolved circuits (the right column) are one order of the magnitude smaller than in the case of the seeding circuits. However, the errors are still relatively high, especially for the 4-bit multiplier. With decreasing amount of resources, the spread of fitness values becomes smaller.

One can observe that the mean fitness f_{mean} of the initial seed (calculated over all runs) is practically independent of the number of available gates for a given multiplier. In additional experiments, we analyzed this phenomenon in detail for various multipliers and adders. Table I gives the mean relative error

$$\epsilon_{mrt} = \frac{f_{mean}}{2^{n_i}(2^{n_o} - 1)} \tag{2}$$

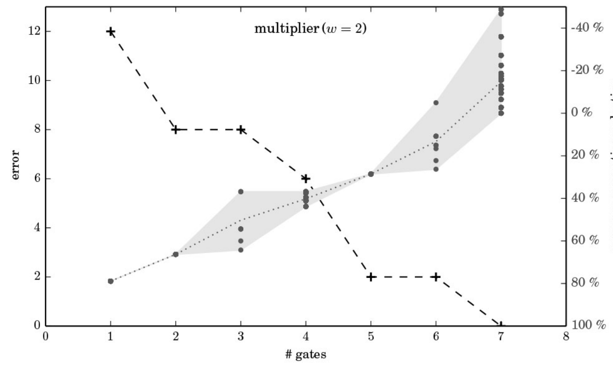


Fig. 7. Power consumption and error of the best evolved approximate 2-bit multipliers for a given number of gates. The mean power reduction is shown as a dotted line.

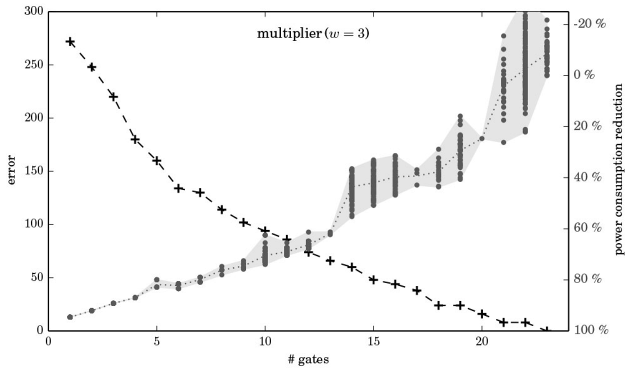


Fig. 8. Power consumption and error of the best evolved approximate 3-bit multipliers for a given number of gates.

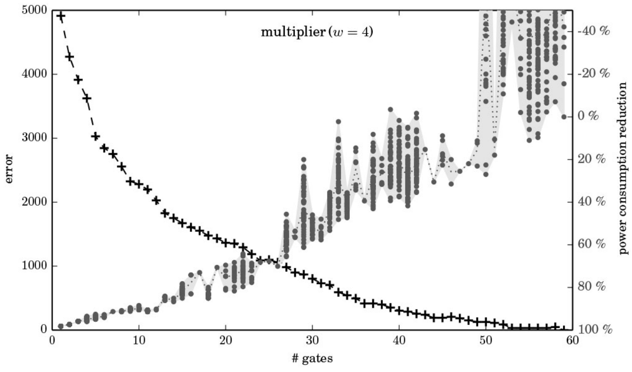


Fig. 9. Power consumption and error of the best evolved approximate 4-bit multipliers for a given number of gates.

of randomly generated circuits consisting of one gate ($n_c = 1$) and n_{bst} gates. It seems that $\epsilon_{mrt} \approx 25\%$ is a reasonable error estimation, not only for multipliers, but also for other approximate arithmetic circuits such as adders that are randomly generated using the proposed method, independent of the number of used gates. This is an important experimental outcome that should help to establish the initial error of any approximation of small combinational circuits performed by means of CGP.

2) Heuristic Seeding: Because the HS1 strategy starts with already preoptimized circuits, it can provide seeds that are

TABLE III
PARAMETERS OF MANUALLY CREATED AND EVOLVED APPROXIMATE MULTIPLIERS FOR LARGER BIT-WIDTHS (w)

Code	w	gates	worst err.	error prob.	average err.	area	delay [ns]
M2	2 [3]	5	13.33%	0.06	0.83%	6.7	5.3
C4	4	**47**	19.61%	0.19	**1.23%**	71.2	38.9
C8	8	276	22.05%	0.47	1.38%	427.4	93.5
C16	16	1288	22.22%	0.81	1.39%	2006.5	184.3

Code	w	gates	worst err.	error prob.	average err.	area	delay [ns]
E4a	4	**47**	3.92%	0.17	0.28%	66.2	34.4
E8a	8	276	4.41%	0.45	0.32%	407.3	87.8
E16a	16	1288	4.44%	0.81	0.32%	1926.3	178.7

Code	w	gates	worst err.	error prob.	average err.	area	delay [ns]
E4b	4	30	7.06%	0.77	**1.23%**	41.9	27.4
E8b	8	208	7.94%	0.98	1.28%	310.2	81.4
E16b	16	1016	8.00%	0.99	1.29%	1537.9	172.2

TABLE IV
RELATIVE ERROR DEVIATION OF TEN EVOLVED MEDIAN CIRCUITS FOR VARIOUS NUMBERS OF TEST VECTORS

	# test vectors						
problem	10	10^2	10^3	10^4	10^5	10^6	10^7
9-median	62.82%	13.24%	3.98%	0.98%	0.48%	0.15%	0.01%
25-median	46.41%	10.22%	2.28%	1.11%	0.30%	0.10%	0.01%

very close to resulting circuits (Fig. 3, above). Contrasted to a very large spread of error values in RS (Fig. 2, right column), it can be seen in HS1 that the CGP runs often converge to one or two fitness values (errors). This is valuable for practice because it means that a single run almost always provides a high-quality solution. The quality of seeding by HS2 is 2–4 times worse as all seeds are generated before the CGP is employed and no intermediate results from CGP can influence the HS2 procedure (the y-axis in Fig. 3, bottom left). The CGP runs converge to several solutions with different fitness values (errors). However, in both cases the error of the generated seeds is significantly lower than the error of the randomly generated seeds (see last row of Figs. 2 and 3).

3) Convergence Curves: Figs. 4 and 5 show convergence curves of all runs in the case where the 4-bit multiplier can utilize 58 gates ($n_{bst} = 59$). Random seeding leads to long convergence times (the best fitness value f stagnates after 10^4 generations) and relatively high errors (see the y-axis of Fig. 4). The HS1 strategy starts with error f=128 and ends with error f=32 in most cases (see the y-axis of Fig. 5). The average error at the end of evolution seeded by RS is approximately 50 times higher than in the case of the HS1 strategy.

4) Overall Comparison: Fig. 6 compares the best solutions obtained in scenarios RS, HS1, and HS2 for 3-bit and 4-bit approximate multipliers. We also included the best results obtained from 50 independent runs of MOEA, which was seeded randomly (MOR) and, in another series of 50 runs, using conventional implementations of multipliers (MOB).

The utilized MOEA implements NSGA-II according to [26] and employs a 50-member population. In order to allow the same number of evaluations as in the proposed CGP, $g_{\max} = 40 \cdot 10^6$ and $64 \cdot 10^6$ for 3-bit and 4-bit multipliers, respectively, in the case of random seeding. The number of generations was decreased to $g_{\max} = 8 \cdot 10^6$ and $16 \cdot 10^6$ in the case of seeding by conventional implementations.

While performance of all the methods is similar on the 3-bit multiplier, HS1 and HS2 seeding strategies clearly outperform RS and both MOEAs in terms of quality of results on the 4-bit multiplier. The gap is significant, especially when 60–90% gates remain in the circuit, which is a typical situation in practice.

Another improvement is in terms of time: RS requires 15 times more generations to reach a solution of the same quality as HS1 and HS2.

A detailed analysis of the best evolved approximate circuits revealed that a circuit containing k gates can exhibit a higher error than a circuit containing $k - 1$ gates (see, for example, the small peak in the fitness function for circuits containing 16 gates and 17 gates in Fig. 6, HS1, $w = 4$). In practice, the circuit containing 16 gates should be taken, even if 17 gates are allowed. There are two explanations for this behavior. Either the evolutionary algorithm did not find a better solution for 17 gates under our setup or a better solution for 17 gates does not exist at all.

5) Power Consumption Versus Error Versus Area: Using the SIS software [25], we calculated dynamic power consumption and delay for the best fully functional conventional as well as evolved multipliers (Table II), which will be serving as reference solutions in the following comparisons. The calculations are valid for the MCNC library [25], $V_{dd} = 5$ V and 20 MHz. The relative area of the used gates is: INV-A 0.67, BUF 0.0, NAND2 and NOR2 1.00, AND2 and OR2 1.33, XOR2 2.00, XNOR2 1.66. The sum of relative areas of gates connected into a particular circuit will be denoted area and will represent the total circuit area relatively to the area of a single NAND gate in the following text.

Power consumption and error of the best evolved approximate 2-bit multipliers are analyzed for a given number of gates in Fig. 7. Power consumption is given relatively to the best conventional solution from Table II. The 7-gate implementations are fully functional. It makes no sense to choose a 6-gate (3-gate, respectively) implementation because the same error can be obtained using a 5-gate (2-gate, respectively) implementation. The evolved 5-gate solution (error = 2) is identical (in terms of structure as well as parameters) with the approximate 2-bit multiplier discovered manually in [3] (see Section II-A2). Contrasted to 7-, 6-, 4-, and 3-gate implementations, there is only one (we believe that truly optimal)

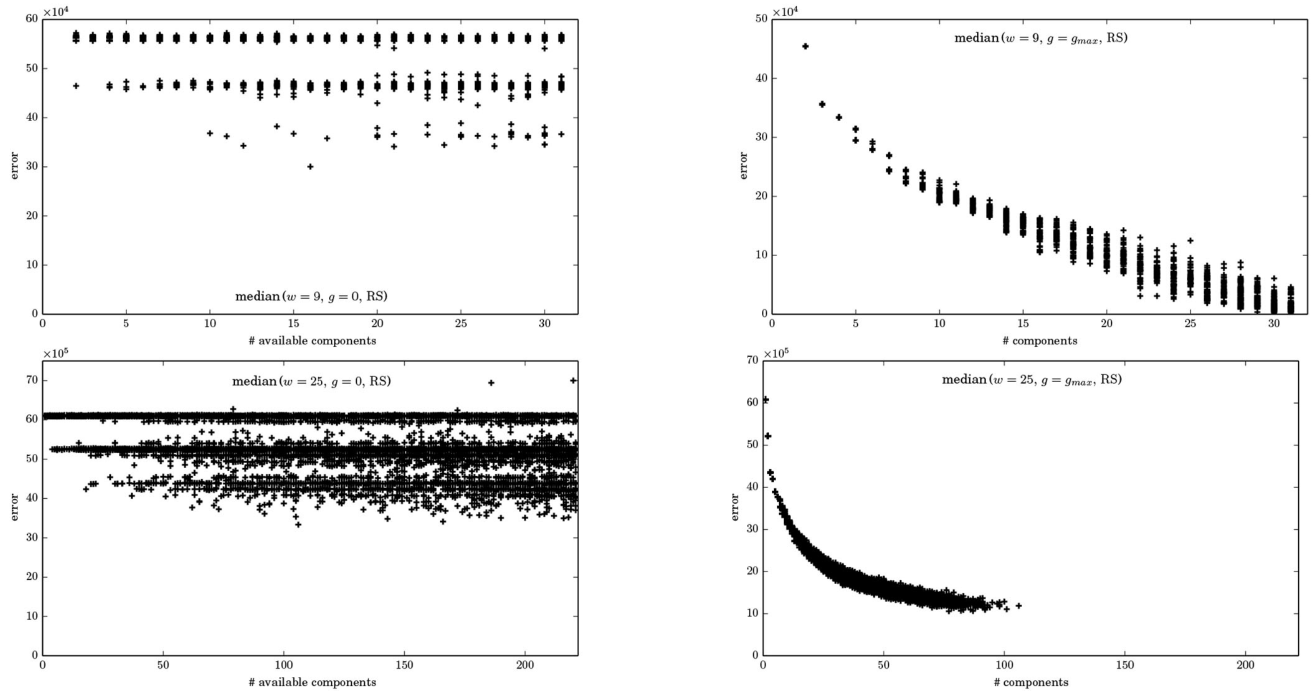


Fig. 10. Error of the seeds (left column) and error of the evolved solutions (right column) for 9-input (top) and 25-input (bottom) median circuits.

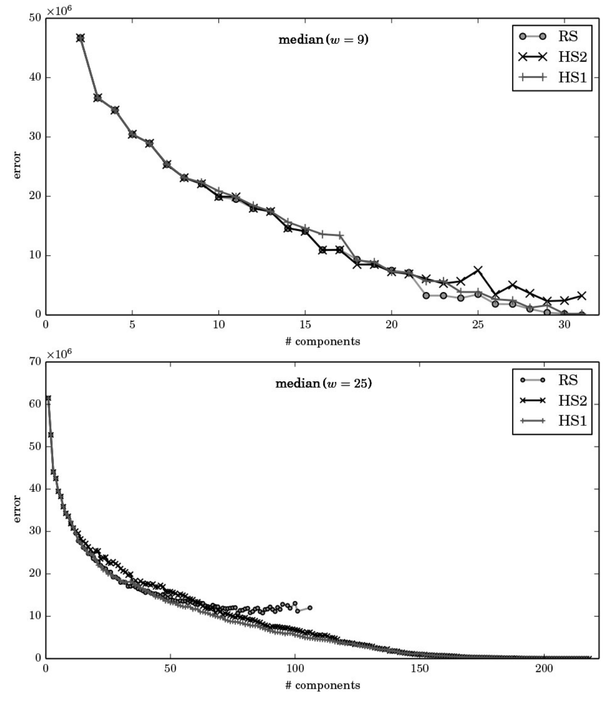


Fig. 11. Error of the best 9-input (top) and 25-input (bottom) approximate median circuits obtained by the proposed strategies.

unique solution in terms of power consumption and error composed of 5 gates.

Figs. 8 and 9 show power consumption and error of the best evolved 3-bit and 4-bit approximate multipliers. A general observation is that the amount of different implementations (and spread of power consumption) decreases with reducing available resources.

TABLE V
PARAMETERS OF THE BEST CONVENTIONAL MEDIAN CIRCUITS

		power [mW]			area [-]			delay [ns]		
w	n_c	best	worst	mean	best	worst	mean	best	worst	mean
9	31	10.8	12.9	12.6	2314.2	2836.7	2750.8	285.9	429.7	295.1
25	221	72.4	72.4	72.4	16497.7	16497.7	16497.7	539.5	539.5	539.5

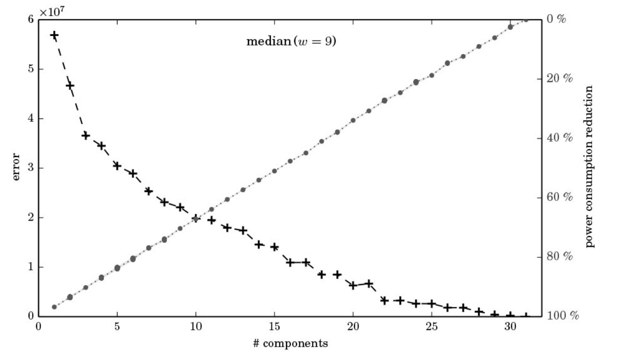


Fig. 12. Power consumption and error of the best evolved approximate 9-median for a given number of components.

6) Comparison With Other Approximate Multipliers: We rediscovered the manually created 2-bit approximate multiplier consisting of five gates (it is denoted M2 in Table III) [3]. Contrasted to the manual design we were able to find very good approximate 4-bit multipliers using CGP (see E4a and E4b in Table III). In order to demonstrate the quality of the evolved solutions, we composed (by the method introduced in [3]) larger approximate multipliers (4-bit, 8-bit, and 16-bit) using M2, E4a, and E4b. The approximate multipliers E4a and

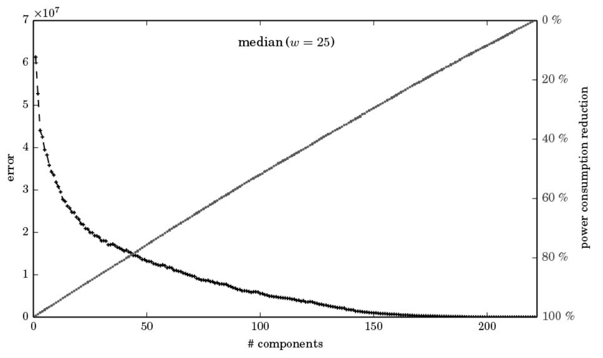


Fig. 13. Power consumption and error of the best evolved approximate 25-median for a given number of components.

E4b were included in the table because they match the number of gates (47 in the case of E4a) and the average error (1.23% in the case of E4b) of the 4-bit approximate multiplier (C4) composed of M2 multipliers.

Table III gives the resulting area and error of the chosen approximate multipliers. The errors are given relative to the corresponding maximum values. The maximum value of the worst case error as well as average error is equal to $e_{\max} = 2^{2w} - 1$. The average error is the total error (as defined in Eq. 1) averaged over all 2^{2w} inputs.

Approximate multipliers composed of evolved approximate 4-bit multipliers show a better tradeoff between the area and error than approximate multipliers composed of M2. For example, the 8-bit multiplier (E8a) composed of evolved 4-bit multipliers E4a exhibits the average error 0.32%, while the average error of the 8-bit multiplier (C8) composed of M2 is 1.38%. Moreover, the worst case error of E8a is 5 times lower. Both 8-bit multipliers, however, consists of 276 gates. A more compact implementation E8b (208 gates) shows an average error of 1.28%, which is even better than C8 can provide.

Our results are hardly comparable with the SASIMI method, because SASIMI employs a different technology library and applies various technology-dependent operations such as downsizing of gates, which allows for an additional area reduction. For example, an 8-bit multiplier initially consisting of 1055 gates was processed by SASIMI, which resulted in a 37% area reduction (it roughly corresponding to an approximate multiplier consisting of 664 gates) and the average error of 0.32% [7]. For the same average error, our 8-bit approximate multiplier E8a consists of 276 gates only. It thus exhibits a reduction of 13% of gates in comparison with a different initial implementation containing 319 gates.

B. Approximate Median Circuits

As it is intractable to evaluate all possible input combinations (256^9 and 256^{25} vectors) for candidate median circuits, we randomly generated 10^4 training vectors for the 9-median circuit and 10^5 vectors for the 25-median circuit. These values were selected according to Table IV, which shows the average deviation of the error if a certain number of randomly generated test vectors is applied to evaluate the quality of these circuits. In order to eliminate the dependency on a certain solution, ten evolved median circuits utilizing 50% of the resources were used. Each circuit was evaluated using a set of ten different randomly generated test vectors. It can be seen that if we apply at least 10^4 test vectors multiple times to evaluate the error of the 9-median circuit, the obtained deviation is less than 1%.

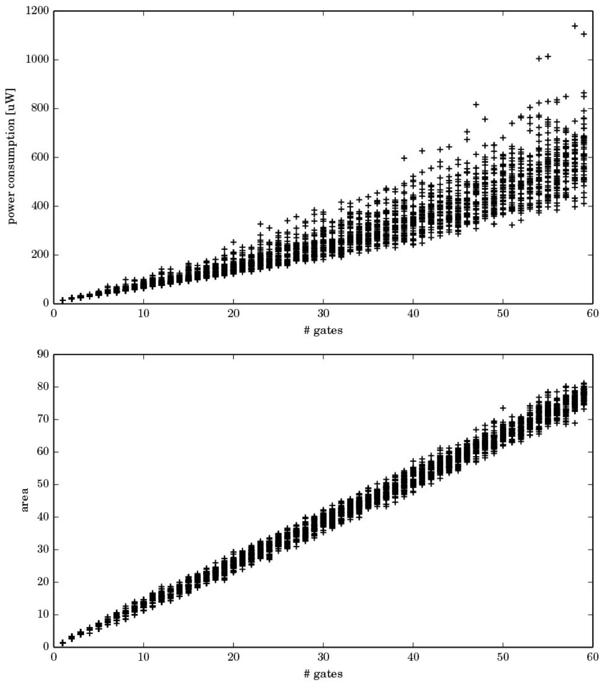


Fig. 14. Relation between the number of gates and power consumption (top) and delay (bottom) for all evolved approximate 4-bit multipliers.

CGP operates with parameters $n_r = 1$, $l = n_c$, $\lambda = 4$, $h = 5\%$, and $\Gamma = \{\text{BUF, MIN, MAX}\}$. All components and connections are defined over 8 bits. Fully functional implementations with $n_{bst} = 37$ for the 9-median and $n_{bst} = 221$ for the 25-median were constructed using the bitonic sorter algorithm [34]. The number of generations of the RS-based CGP is limited by $g_{\max} = 3\cdot 10^6$ for the 9-median and $g_{\max} = 300\cdot 10^3$ for the 25-median which corresponds to 3 h CGP runs in both cases. CGP exploiting HS1 and HS2 utilized only 1/3 of the previously mentioned time budget. Each CGP run is repeated 50 times.

1) Role of Seeding: The randomly seeded CGP led to fully functional solutions for the 9-median while no correct solution was discovered for the 25-median. It seems that solving the 25-median design problem from scratch is impossible for any evolutionary algorithm based on a direct encoding. Although CGP could utilize up to $n_{bst} = 221$ components, the most complex circuits only use 106 components (Fig. 10). In order to investigate this phenomenon, we conducted an another experiment and seeded CGP by randomly created circuits that utilize all 221 components, but most were disconnected in the course of evolution, thus reaching 106 components again.

Fig. 10 shows the fitness values of all randomly created circuits and the resulting evolved approximate median circuits. Compared to the multiplier problem, the error values of the randomly generated circuits are not distributed uniformly.

The effect of RS, HS1, and HS2 seeding strategies is compared in Fig. 11, which gives the error of the best evolved solution for a given number of components. All strategies perform almost identically for the 9-median when the number of components is lower than 20. RS is clearly outperformed by HS1 for the 25-median. We can again observe the situation in which a circuit containing k components exhibits a higher error than a circuit containing $k-1$ components.

2) Best Approximate Median Circuits: Power consumption, area, and delay of the best evolved fully functional solutions are summarized in Table V. These circuits are composed of 8-bit subcomponents: minimum and maximum. Each is represented as a netlist containing the gates presented in Section IV-A5. All the circuit parameters were obtained from the SIS tool. Power consumption is given for 320 000 randomly generated input vectors.

Figs. 12 and 13 show power consumption and error of the best evolved approximate median circuits. The error is calculated using 10^6 test vectors. Power consumption is given relatively to the fully functional circuit showing the lowest power consumption.

Median circuits are very good examples of circuits for which it makes sense to introduce their approximate versions. The mean error remains relatively low, even for significant reductions of available gates. Hence, significant improvements in energy consumption are obtained.

V. Conclusion

The proposed method and experimental results can be interpreted from several points of view. First the proposed method is a new systematic method for the design of approximate circuits. Its main contribution lies in the area rather than the error-oriented approach to approximate circuit design, which enables the user to comfortably control the used resources. It is useful, for example, when an image filter has to be approximated because the conventional implementation does not fit in the available space on a chip. The method works at the logic level and no special technology-oriented techniques (such as downsizing of gates) were considered during our experimental evaluation. Second, because the method is based on evolutionary computation, it can naturally provide more tradeoffs than current methods based on manual modifications of existing designs or reusing conventional synthesis tools. Third, we evolved new approximate implementations of key circuits (multipliers, median computing circuits) that can immediately be used in various applications. These circuits, together with approximate adders and other approximate combinational circuits presented in our previous work [10], demonstrate that the CGP-based method is suitable for approximate circuits design. Fourth, we have shown that the proposed method can easily be extended from the gate to the functional level evolution. Conventional methods (such as SALSA and SASIMI) work at the bit level only and hence they cannot be applied to directly approximate circuits such as the median. Fifth, we provided detailed analyses of selected features of CGP, particularly the population seeding, which had not been done before.

Our initial assumption that the power consumption is highly correlated with area (see Section III-A) and that the proposed methodology can be based on reducing the number of gates was positively confirmed. Fig. 14 shows the dependence for evolved multipliers. By considering the area (amount of gates) only, without calculating power consumption for every candidate circuit, we obtained very good approximations in a relatively short time.

Experimental results confirmed the superiority of heuristic seeding of the initial population over random seeding. The benefits are not only in improving the quality of evolved circuits, but also in reducing the time of optimization. Another advantage is that each run of CGP seeded by the HS1 strategy provides a high-quality solution. For more complex problem instances (such as the 25-median), the randomly seeded standard CGP did not provide satisfactory results. A suitable seeding approach thus remains a method to overcome this limitation.

The execution time is certainly the most critical disadvantage of the proposed method. It mainly depends on the number of inputs and size of the circuit. When a suitable seeding of the initial population is available, then CGP runtimes are typically in the order of tens of minutes on a common desktop computer. As no execution times were reported for SASIMI, we give the execution times of SALSA that, on a server with an AMD Opteron 6176 (2.29 GHz) processor, ranged from 4 min to 2.5 h depending on the circuit complexity.

Our future research will focus on applying selected approaches (such as incremental evolution, functional equivalence checking) introduced to eliminate the scalability problems of evolutionary circuit design to evolutionary design methods intended for approximate circuits. We believe that the notion of approximate computing offers new applications for genetic programming (such as a design of underdesigned software for embedded systems) that should be explored in future research.

References

[1] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proc. 18th IEEE Eur. Test Symp.*, Avignon, France, 2013, pp. 1–6.

[2] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 32, no. 1, pp. 124–137, Jan. 2013.

[3] P. Kulkarni, P. Gupta, and M. D. Ercegovac, "Trading accuracy for power in a multiplier architecture," *J. Low Power Electron.*, vol. 7, no. 4, pp. 490–501, 2011.

[4] S. Venkataramani, A. Sabne, V. J. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: Systematic logic synthesis of approximate circuits," in *Proc. 49th Annu. Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2012, pp. 796–801.

[5] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Proc. 2012 45th Annu. IEEE/ACM Int. Symp. Microarchitect.*, Vancouver, BC, Canada, pp. 449–460.

[6] A. Sampson *et al.*, "EnerJ: Approximate data types for safe and general low-power computation," in *Proc. 32nd ACM SIGPLAN Conf. Program. Lang. Design Implement.*, San Jose, CA, USA, 2011, pp. 164–174.

[7] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *Design Autom. Test Europe (DATE)*, Grenoble, France, 2013, pp. 1367–1372.

[8] J. D. Lohn and G. S. Hornby, "Evolvable hardware: Using evolutionary computation to design and optimize hardware systems," *IEEE Comput. Intell. Mag.*, vol. 1, no. 1, pp. 19–27, Feb. 2006.

[9] P. C. Haddow and A. M. Tyrrell, "Challenges of evolvable hardware: Past, present and the path to a promising future," *Genet. Program. Evolvable Mach.*, vol. 12, no. 3, pp. 183–215, 2011.

[10] L. Sekanina and Z. Vasicek, "Approximate circuits by means of evolvable hardware," in *Proc. 2013 IEEE Int. Conf. Evolvable Syst.*, pp. 21–28.

[11] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 195–237, 2005.

[12] N. R. Shanbhag, R. A. Abdallah, R. Kumar, and D. L. Jones, "Stochastic computation," in *Proc. 47th Annu. Design Autom. Conf. (DAC)*, Anaheim, CA, USA, 2010, pp. 859–867.

[13] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "MACACO: Modeling and analysis of circuits for approximate computing," in *Proc. 2011 IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, pp. 667–673.

[14] M. Murakawa *et al.*, "Evolvable hardware at function level," in *Parallel Problem Solving from Nature—PPSN IV*, LNCS 1141. Berlin, Germany: Springer Verlag, 1996, pp. 62–71.

[15] A. P. Shanthi and R. Parthasarathi, "Practical and scalable evolution of digital circuits," *Appl. Soft Comput.*, vol. 9, no. 2, pp. 618–624, 2009.

[16] E. Stomeo, T. Kalganova, and C. Lambert, "Generalized disjunction decomposition for evolvable hardware," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 36, no. 5, pp. 1024–1043, Oct. 2006.

[17] T. Gordon, "Exploiting development to enhance the scalability of hardware evolution," Ph.D. dissertation, Dept. Comput. Sci., Univ. College London, London, U.K., 2005.

[18] K. Imamura, J. A. Foster, and A. W. Krings, "The test vector problem and limitations to evolving digital circuits," in *Proc. 2nd NASA/DoD Workshop Evolvable Hardware*, Palo Alto, CA, USA, 2000, pp. 75–79.

[19] Z. Vasicek and L. Sekanina, "Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware," *Genet. Program. Evolvable Mach.*, vol. 12, no. 3, pp. 305–327, 2011.

[20] Z. Vasicek and K. Slany, "Efficient phenotype evaluation in Cartesian genetic programming," in *Proc. 15th Eur. Conf. Genet. Program.*, Malaga, Spain, 2012, pp. 266–278.

[21] J. F. Miller, "On the filtering properties of evolved gate arrays," in *Proc. 1st NASA-DoD Workshop Evolvable Hardw.*, Pasadena, CA, USA, 1999, pp. 2–11.

[22] G. Greenwood and A. M. Tyrrell, *Introduction to Evolvable Hardware*. Piscataway, NJ, USA: IEEE Press, 2007.

[23] T. Knieper, P. Kaufmann, K. Glette, M. Platzner, and J. Torresen, "Coping with resource fluctuations: The run-time reconfigurable functional unit row classifier architecture," in *Proc. 9th Int. Conf. Evolvable Syst.*, LNCS 6274. York, U.K., 2010, pp. 250–261.

[24] A. Thompson, P. Layzell, and S. Zebulum, "Explorations in design space: Unconventional electronics design through artificial evolution," *IEEE Trans. Evol. Comput.*, vol. 3, no. 3, pp. 167–196, Sep. 1999.

[25] E. M. Sentovich, "SIS: A system for sequential circuit synthesis," EECS Dept., Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/ERL M92/41, 1992.

[26] J. Petrlik and L. Sekanina, "Multiobjective evolution of approximate multiple constant multipliers," in *Proc. IEEE Int. Symp. Design Diagnostics Electron. Circuits Syst.*, Karlovy Vary, Czech Republic, 2013, pp. 116–119.

[27] J. Hilder, J. Walker, and A. Tyrrell, "Use of a multi-objective fitness function to improve Cartesian genetic programming circuits," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst.*, Anaheim, CA, USA, 2010, pp. 179–185.

[28] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[29] J. F. Miller, D. Job, and V. K. Vassilev, "Principles in the evolutionary design of digital circuits—Part II," *Genet. Program. Evolvable Mach.*, vol. 1, no. 3, pp. 259–288, 2000.

[30] J. F. Miller, D. Job, and V. K. Vassilev, "Principles in the evolutionary design of digital circuits—Part I," *Genet. Program. Evolvable Mach.*, vol. 1, no. 1, pp. 8–35, 2000.

[31] J. F. Miller, *Cartesian Genetic Programming*. New York, NY, USA: Springer-Verlag, 2011.

[32] J. F. Miller and S. L. Smith, "Redundancy and computational efficiency in Cartesian genetic programming," *IEEE Trans. Evol. Comput.*, vol. 10, no. 2, pp. 167–174, Apr. 2006.

[33] S. Zhao and L. Jiao, "Multi-objective evolutionary design and knowledge discovery of logic circuits based on an adaptive genetic algorithm," *Genet. Program. Evolvable Mach.*, vol. 7, no. 3, pp. 195–210, 2006.

[34] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching*, 2nd ed. Reading, MA, USA: Addison Wesley, 1998.

Zdenek Vasicek received the M.Sc. and Ph.D. degrees in electrical engineering and computer science from the Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic, in 2006 and 2012.

He is an Assistant Professor with the Faculty of Information Technology, Brno University of Technology. His research interests include evolutionary design and optimization of complex digital circuits and systems. He has authored or co-authored over 30 conference/journal papers focused on evolvable hardware and hardware design.

Mr. Vasicek received numerous awards for his research in evolvable hardware, including the Joseph Fourier Award in 2011 for research in computer science and engineering.

Lukas Sekanina (M'02–SM'12) received the M.Eng. and Ph.D. degrees from Brno University of Technology, Brno, Czech Republic, in 1999 and 2002, respectively.

He is a Full Professor with the Faculty of Information Technology, Brno University of Technology. His research interests include evolutionary design and evolvable hardware. He was a Visiting Lecturer with Pennsylvania State University, The Behrend College, PA, USA, and a Visiting Researcher with the University of Oslo, Oslo, Norway, in 2001. He has co-authored over 150 papers, mainly on evolvable hardware.

He received a Fulbright Scholarship to work with NASA Jet Propulsion Laboratory, Pasadena, CA, USA, in 2004. He was an Associate Editor of IEEE TRANSACTIONS OF EVOLUTIONARY COMPUTATION and an Editorial Board Member of *Genetic Programming and Evolvable Machines* and *International Journal of Innovative Computing and Applications*.

Appendix D

Evolutionary Design of Complex Approximate Combinational Circuits

VASICEK, Zdenek and SEKANINA, Lukas. “Evolutionary Design of Complex Approximate Combinational Circuits”. In: *Genetic Programming and Evolvable Machines* 17.2 (2016), pp. 169–192.

IF=1.143 (2015), contribution of the author of the thesis: 60%

Referenced on pages: 18, 37, 39, 40

Genet Program Evolvable Mach (2016) 17:169–192
DOI 10.1007/s10710-015-9257-1

Evolutionary design of complex approximate combinational circuits

Zdenek Vasicek[1] · Lukas Sekanina[1]

Received: 30 June 2015 / Revised: 13 November 2015 / Published online: 12 December 2015

Abstract Functional approximation is one of the methods allowing designers to approximate circuits at the level of logic behavior. By introducing a suitable functional approximation, power consumption, area or delay of a circuit can be reduced if some errors are acceptable in a particular application. As the error quantification is usually based on an arithmetic error metric in existing approximation methods, these methods are primarily suitable for the approximation of arithmetic and signal processing circuits. This paper deals with the approximation of general logic (such as pattern matching circuits and complex encoders) in which no additional information is usually available to establish a suitable error metric and hence the error of approximation is expressed in terms of Hamming distance between the output values produced by a candidate approximate circuit and the accurate circuit. We propose a circuit approximation method based on Cartesian genetic programming in which gate-level circuits are internally represented using directed acyclic graphs. In order to eliminate the well-known scalability problems of evolutionary circuit design, the error of approximation is determined by binary decision diagrams. The method is analyzed in terms of computational time and quality of approximation. It is able to deliver detailed Pareto fronts showing various compromises between the area, delay and error. Results are presented for 16 circuits (with 27–50 inputs) that are too complex to be approximated by means of existing evolutionary circuit design methods.

✉ Lukas Sekanina
sekanina@fit.vutbr.cz

Zdenek Vasicek
vasicek@fit.vutbr.cz

[1] Faculty of Information Technology, IT4Innovations Centre of Excellence, Brno University of Technology, Brno, Czech Republic

Keywords Approximate circuit · Cartesian genetic programming · Binary decision diagram · Fitness function

1 Introduction

Reducing of energy consumption in integrated circuits is one of the key challenges of current chip design industry [4]. Hence, various approaches to energy consumption reduction have been developed. Energy consumption reduction can be tackled at different system levels (such as circuit, architecture, operating system, and software) with significantly different methodologies. One of them is *approximate computing* trying to exploit the error resilience which is displayed by many applications [11]. If one can relax the precision constraints, or tolerate some errors, hardware and software can be simplified and work with less energy. Suitable applications for approximate computing were identified in the areas of multimedia, database search, fault tolerant systems and others. They exploit the fact that human users, as major consumers of data outputs, have limited perception capabilities and no golden solution is usually available for validation of results [6]. An open question is how to automate the approximation of circuits and software in order to obtain desired quality of service (i.e. an acceptable error) and optimize available resources.

The *functional approximation* is one of methods allowing designers to approximate circuits at the level of logic behavior [40]. The idea behind the functional approximation is that a less complex function than the original one is implemented and used, providing that the error is acceptable and power consumption, area on the chip or other parameters are improved adequately. The approximations are obtained by a heuristic procedure which modifies the original, accurate circuit. Applying genetic programming as a heuristic method for circuit approximation has already led to finding high-quality compromises between key circuit parameters, see, for example [35, 36].

As the vast majority of approximation methods employ an arithmetic error metric, these methods are primarily suitable for the approximation of arithmetic circuits (adders, multipliers) and digital signal processing circuits. This paper deals with the approximation of general logic in which no additional information is usually available to establish a suitable error metric. Introducing approximations to general logic could be dangerous in many cases (e.g. for controllers), but there is still an important class of circuits (such as combinational logic of pattern matching circuits or complex encoders) in which the error can safely be exchanged for reducing the energy consumption or the area on a chip. For example, see an approximate pattern matching circuit optimized for fast classification of application protocols in high-speed networks [9]. In these cases, the error of approximation has to be expressed using a more general function, for example, as the average Hamming distance between the output values produced by a candidate approximate circuit and the accurate circuit.

The current literature describes various approaches to the digital circuit approximation. Regarding the methodological and evaluation approaches, two scenarios are dominating:

1. Ad hoc methods employed for the approximation of a (single) particular circuit. For example, see the approaches proposed to approximate multipliers [17] and adders [10].
2. Design automation methods developed for the approximation of a class of circuits (for example, SALSA [40], SASIMI [39] and ABACUS [23]).

In the first scenario, a lot of knowledge about a particular circuit and its typical utilization can be incorporated into the approximation method. However, it is difficult to apply the method for approximation of other circuits. In the second scenario, the approximations are performed using the same procedure for all problem instances of a given class. Approximate circuits showing different compromises between considered circuit parameters (area, delay, power consumption and errors of different types) are generated and presented to the user whose responsibility is to choose the most suitable approximate circuit for a given application. A detailed analysis of the impact of the approximation procedure on circuit parameters that were not considered during the approximation is also left on the user.

The goal of this work is to propose and evaluate an automated circuit approximation method (scenario 2) in which the error is expressed in terms of the average Hamming distance. We opted for the evolutionary approach based on genetic programming because it was capable of delivering high quality approximations in our previous work [35, 36]. In our method, gate-level circuits are evolved using Cartesian genetic programming (CGP) and internally represented using directed acyclic graphs. The method is thus suitable for approximation of combinational circuits, i.e. digital circuits in which the output values only depend on current input values. In the case of sequential circuits containing memory elements, the proposed method can be applied to a combinational part of the circuit.

The evolutionary circuit design methods in which candidate circuits are evaluated by checking their responses for all possible input combinations are not scalable. The main reason is that the evaluation time grows exponentially with the number of inputs. A naïve approach to evolve approximate circuits would be to identify a suitable subset of all possible input vectors, establish the fitness value using this subset and evolve a circuit showing a good trade-off between the error (for this subset) and the number of gates (or area). However, as it is reasonable to evaluate only up to about 2^{20} test vectors for each candidate circuit in a single CGP run on a common desktop computer [34], the resulting error would be extremely unreliable for circuits with, for example, 30 primary inputs.

In order to overcome this problem, we propose to determine the error of approximation by an equivalence checking algorithm operating over binary decision diagrams (BDD) representing the candidate approximate circuit and the accurate circuit. The main advantage of BDDs is that the Hamming distance can be determined in linear time with respect to the BDD size. Converting a candidate circuit to BDD and performing the functionality comparison against the accurate circuit, expressed again as BDD, can be performed relatively quickly for many circuits relevant to practice. The proposed method is analyzed in terms of computational time and quality of approximation.

The method is evaluated using 16 benchmark combinational circuits which are difficult for the previous evolutionary approximation methods, because they have too many primary inputs (27–50 inputs) and gates. Pareto fronts showing obtained trade offs between the error, area and delay are also reported. Another contribution of our work is that it is focused on general (i.e. non-arithmetic) approximate circuits which has not been done before.

The rest of the paper is organized as follows. Section 2 summarizes relevant work in the areas of functional approximation and digital circuit evolution. The principles of BDD are defined in Sect. 3. The proposed method based on CGP is introduced in Sect. 4. The experimental setup, benchmark circuits and results of evolutionary design are presented in Sect. 5. Conclusions are given in Sect. 6.

2 Related work

This section briefly surveys conventional approaches introduced for functional approximation and evolutionary design methods developed for the design of common and approximate digital circuits. The survey is primarily focused on combinational circuits as no other circuits are relevant for this paper.

2.1 Functional approximation

The goal of functional approximation is to modify a given logic circuit in such a way that obtained error is minimal and key circuit parameters (such as delay, area and power consumption) are improved with respect to the original logic circuit. The approximations have been conducted manually or using systematic algorithmic methods. The *manual approximation* methods whose example results are approximate multipliers presented in [17] have recently been replaced by fully automated systematic methods in order to increase the design productivity as well as the quality and complexity of circuits that can be approximated.

The *systematic design automation methods* (such as SALSA [40], SASIMI [39] and ABACUS [23]) produce Pareto fronts showing various compromise solutions with respect to the optimized parameters (error, delay, and power consumption). It allows the user to select the best compromise solution for a given application.

A typical automated method starts with a fully functional circuit which is modified by means of a problem specific heuristic in order to improve key circuit parameters, and keep the error within predefined bounds. The Pareto front is obtained from multiple runs of a single-objective approximation (heuristic) algorithm initialized using different parameters (for example, five target errors are considered). Parameters of resulting approximate circuits are obtained by means of professional design tools. As only tens to hundreds of design alternatives are generated, the resulting solutions do not cover the whole Pareto front and they are typically centered around a few dominant design alternatives (e.g. [23]). The available literature describing these methods does not present any detailed analyses of resulting Pareto fronts, i.e. it is unknown whether and how much the obtained results can be improved if, for example, more execution time were invested.

The key issue seems to be an efficient and reliable evaluation of candidate approximate circuits. Various error functions have been used, for example, worst error, relative error, average error magnitude, and error probability. While these errors can be computed for small circuits by analyzing circuit responses for all possible input vectors, formal methods have to be introduced to determine the error of complex arithmetic circuits. For example, an auxiliary circuit is constructed which instantiates the candidate approximate circuit and the accurate (golden) circuit and compares their outputs to quantify the error for any given input. In order to check whether a predefined worst error is violated by the candidate approximate circuit, Boolean satisfiability (SAT) solver is employed [41]. However, for example, no method capable of establishing the average error using a SAT solver has been proposed up to now.

Contrasted to the methods precisely calculating the error (which were described in the previous paragraph), the error of approximation is also often estimated using training data sets. This is typical for image and signal processing components (filters, classifiers) because suitable training data are usually available and calculating the exact error is intractable because of the overall complexity of these components [23].

Fault tolerant systems are another natural class of applications of approximate computing. Redundant circuits which are present in such systems can be approximated in order obtain a good trade off between dependability parameters and power consumption or area on the chip [25].

2.2 Evolutionary circuit design

The idea of evolvable hardware and digital circuit evolution was introduced by Higuchi et al. [12], in which the evolution of a six-input multiplexer using a circuit simulator was presented. Thompson reported first circuits evolved directly in the hardware in 1996 [29].

A significant development of evolutionary circuit design is connected to Cartesian genetic programming, a branch of genetic programming whose problem representation was inspired by digital circuits. In CGP, candidate circuits are encoded as arrays of integers and evolved using a simple search strategy. The standard CGP, its extensions (such as self-modifying CGP) and typical applications have been surveyed in a monograph [19]. Miller et al. demonstrated that CGP can improve results (in terms of the number of gates and delay) of conventional circuit synthesis and optimization algorithms in the case of small arithmetic circuits. A 4-b multiplier was the most complex circuit evolved in this category [38].

After the year 2000, various digital (predominately combinational) circuits were evolved. These circuits can be classified into two categories—completely specified and incompletely specified circuits. Completely specified circuits are arithmetic circuits and general logic circuits in which a perfect response is requested for every legal input vector. On the other hand, incompletely specified circuits are used in applications such as classification, filtering, hashing and prediction in which the correctness can only be verified using a subset of all possible input vectors.

In comparison with conventional methods, the evolutionary design method is less scalable. It has several reasons. First, long chromosomes are needed to represent complex circuits, and consequently, huge search spaces have to be explored in which it is difficult to find useful designs. Second, the evaluation of complex circuits is very time consuming. In a typical approach, 2^n input vectors are applied (and simulated) to calculate the fitness of an n-input combinational circuit. In current practice, the maximum complexity of evolved circuits is low (about 20 inputs and 100 gates).

Several methods have been proposed to increase the complexity of circuits that can be obtained using evolutionary algorithms (EA). Functional level evolution [22] and decomposition methods [28, 30] enabled to reduce the search space. Combining functional level evolution with decomposition led to another increment in the complexity of evolved circuits [27]. Regarding the completely defined circuits, examples of the most complex circuits evolved so far are 22-b parity [24], 9-b adder [14] and 5-b multiplier [14]. The most complex circuit evolved using decomposition is a 135-b multiplexer which was obtained with a learning classifier system operating with complex building blocks. The correctness of resulting circuits was, however, estimated using simulation and manual inspection because it was impossible to get responses for all 2^{135} input vectors [15].

More promising results have been obtained by methods which try to reduce the fitness evaluation time using formal approaches in the fitness function.

In order to minimize the number of gates in fully functional circuits produced by well-tuned common synthesis and optimization tools, Vasicek and Sekanina [32] proposed to replace the circuit simulation by functional equivalence checking algorithms. For each candidate circuit and its parent, a SAT problem instance was created and solved using a SAT solver. If both circuits are functionally equivalent, the fitness of the candidate circuit is defined as the number of gates (with the aim to minimize them); otherwise, the candidate circuit is discarded. This approach led to a significant reduction in gate count for circuits having hundreds of inputs and containing thousands of gates [31], which is unreachable by the state of the art logic synthesis tools such as ABC [21]. The most complex circuit optimized using this method contains 16,158 gates and has 2176 inputs and 2136 outputs [31].

The SAT-based method is applicable only if a fully functional circuit is available. If a circuit has to be evolved from scratch (i.e. when no structural information about the circuit is provided, but responses are defined for all possible input combinations), Vasicek and Sekanina [34] combined CGP with BDD and developed a tool which allowed for evolving circuits with tens of inputs. The BDDs in the fitness function enable to effectively determine the Hamming distance between the output vectors of two circuits for many important problem instances (see Sect. 3). A 28-input circuit was successfully evolved from scratch without any kind of decomposition technique. In addition to that, the obtained circuit had less gates (a

57 % reduction) than the result of a conventional optimization conducted by the state-of-the-art tool.

2.3 Evolutionary circuit approximation

The use of evolutionary algorithms for functional approximation was surveyed in [26]. Employing evolutionary algorithms seems to be natural with respect to the goal of the approximation task. Small modifications introduced in the progress of evolution via genetic operators to a population of circuits and the principle of the survival of the fittest naturally lead to discovering such circuits which show very good compromises between the error and area (power consumption). Available evolutionary approximation methods employ CGP, which can operate either as a single-objective or multi-objective evolutionary optimizer. Within a given time which is available for the design, the single objective CGP provided more compact circuits than its multi-objective version [13, 35, 36].

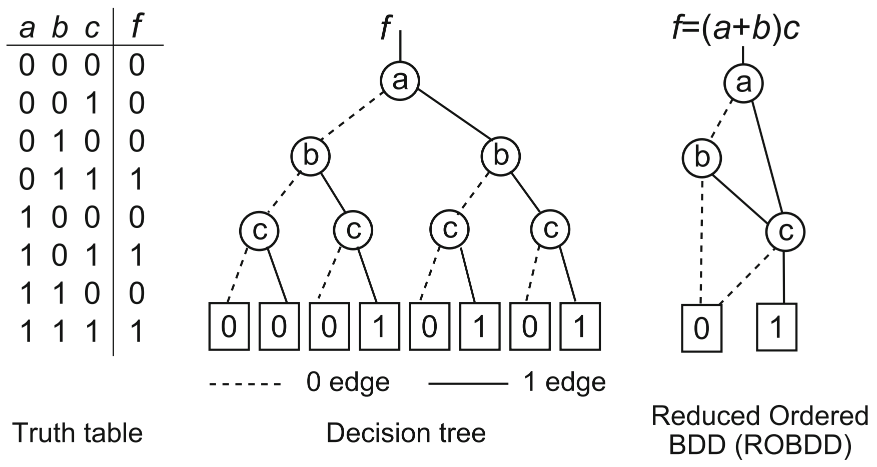
Because of the scalability problems, the evolutionary approach allowed obtaining only relatively small approximate combinational circuits and arithmetic circuits (up to 8-b adders and multipliers [33, 35] when seeded by conventional implementations). More complex circuits, such as a 25-input median circuit, were then approximated by an evolutionary algorithm estimating the error of approximation using a small subset (10^5 vectors) of all possible input vectors (10^{60}) [36]. However, the method evaluating candidate circuits using a subset of input vectors is not applicable to approximate arithmetic circuits and other circuits that we treated as completely defined in Sect. 2.2.

In order to approximate complex circuits (belonging to the class of completely defined circuits) using the Hamming distance as a metric, we will use BDDs in the fitness function. This idea was initially proposed in our paper [37], but without a detailed experimental evaluation.

3 Binary decision diagrams in circuit design

3.1 Binary decision diagrams

A BDD is one of possible representations of logic functions. A BDD is a directed acyclic graph with one root, non-terminal nodes and two terminal nodes that are referred to ‘0’ and ‘1’. Each non-terminal node is labeled by a primary input variable x_i. If $x_i = 0$ then the outgoing zero-edge is taken; if $x_i = 1$ then the outgoing one-edge is taken. By tracing a path from the root to terminal node ‘1’ one obtains an assignment to input variables for which the function is evaluated to 1. An ordered binary decision diagram (OBDD) is a BDD where variables occur along every path from the root to a terminal node in strictly ascending order, with regard to fixed ordering. A reduced ordered binary decision diagram (ROBDD) is an OBDD where each node represents a unique logic function, i.e. it contains neither isomorphic subgraphs nor nodes with isomorphic descendants. Figure 1 shows a



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	1

Fig. 1 Logic function $f = ac + bc$ expressed using truth table, BDD and ROBDD

Boolean function represented by truth table and corresponding BDD and ROBDD. Two or more logic functions can be represented by a single ROBDD (i.e. there are several root nodes) in which some subgraphs are shared by some of the functions.

ROBDDs are important because they are canonical, i.e. if two logic functions have the same truth table, their ROBDDs are isomorphic. Unfortunately, the size of ROBDD (i.e. the number of non-terminal nodes) for a given function is very sensitive to the chosen variable order; in some cases it is linear, in other cases is exponential with respect to the number of inputs [5]. Moreover, multipliers are known for their exponential memory requirements for any variable ordering [1]. In order to optimize the size of ROBDD, various minimization algorithms were proposed [5]. The most efficient method is sifting, an iterative algorithm which is based on finding the optimum position of each variable assuming that all other variables remain fixed.

BDDs and evolutionary computing have been combined in the past. For example, variable ordering of an BDD was optimized by EA [2], and an EA that learns heuristics for BDD minimization was proposed in [3]. Detailed survey is available in [5].

3.2 Operations over BDDs

ROBDDs are equipped with several operations. Let us mention two basic operations that are relevant to our paper: *apply* and *Sat-Count*.

The *apply*(op, f, g) operation enables to construct a ROBDD from existing ROBDDs. It takes a binary operator op and two ROBDDs f and g as arguments and returns a ROBDD corresponding with the result of f op g [18]. In fact, *apply* is a complex operation which can remove some nodes, add new nodes, and rearrange existing nodes to guarantee that the resulting BDD is a ROBDD.

The *Sat-Count* operation computes the number of input assignments for which f is evaluated to ‘1’, i.e. it determines the number of elements in the so-called *Onset* of f. *Sat-Count* can be performed in time $O(|F|)$, where F is a ROBDD corresponding to f, just by following the leftmost path in F that leads to a non-zero terminal. It means that the *Sat-Count* operator can be implemented in such a way that the number of input assignments for which $f = 1$ is obtained with linear time with respect to the size of ROBDD constructed for f. This is a very important feature in the context of evolutionary circuit design. Obtaining the same result using simulation requires $2^n|C|$ steps for n-input circuit C containing $|C|$ gates. On the other hand, it has to be noted that the worst case time complexity of BDD construction is exponential (see [5]), but it is not usually the case of circuits used in practice.

Several libraries have been developed to effectively construct (RO)BDDs and perform operations over them. In this work, Buddy package is employed [18].

3.3 Hamming distance using BDDs

BDDs are often used to decide whether two combinational circuits are functionally equivalent. Let us suppose that both circuits have k inputs denoted $x_1 \ldots x_k$ and m outputs denoted $y_1 \ldots y_m$ and $y'_1 \ldots y'_m$, respectively. Corresponding primary inputs of both circuits are aligned and corresponding primary outputs y_i and y'_i are connected using the XOR gates. The goal is to obtain one (auxiliary) circuit with k primary inputs $x_1 \ldots x_k$ and m primary outputs $z_1 \ldots z_m$, $z_i = y_i$ XOR y'_i. In order to disprove the equivalence, it is then sufficient to identify at least one output z_i whose $Onset(z_i)$ is not empty, i.e. to find an input assignment x for which the corresponding outputs y_i and y'_i provide different values. An example is given in Fig. 2 where two circuits C_A and C_B with four inputs and two outputs are checked for Boolean equivalence. Because y_2 and y'_2 capture the same Boolean function, the ROBDD constructed for z_2 consists of a single pointer to the zero node. The outputs y_1 and y'_1, however, represent different Boolean functions. The ROBDD constructed for z_1 thus consists of non-zero number of nodes and there exists at least one path from the root node determined by pointer z_1 to the node 1.

The auxiliary circuit used to perform the combinational equivalence checking can be applied to determine the Hamming distance between truth tables of circuit

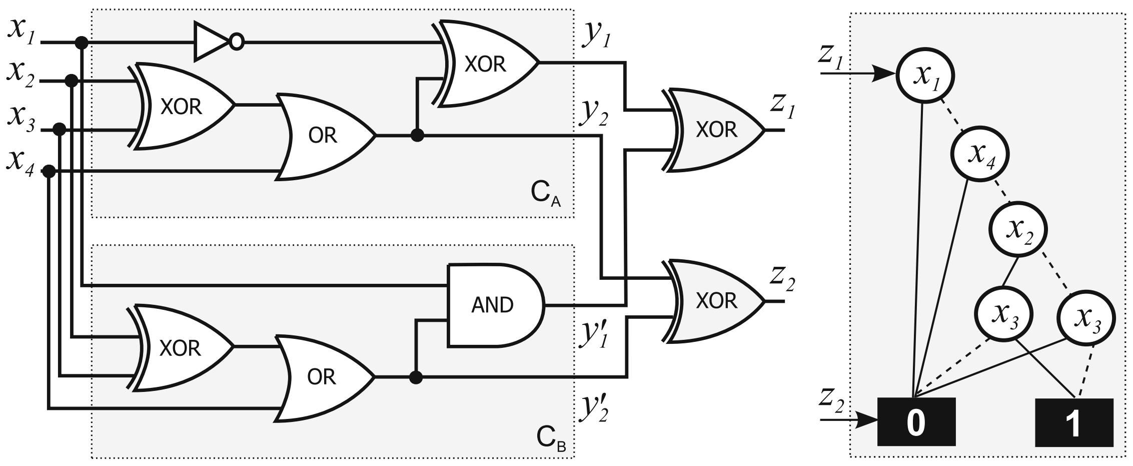


Fig. 2 Auxiliary circuit used to perform equivalence checking of two combinational circuits C_A and C_B (*left*) and ROBDD constructed for z_1 and z_2 (*right*)

C_A and C_B. The Hamming distance can be obtained by applying the *Sat-Count* operation on every output z_i and counting up all the results. In the example shown in Fig. 2, *Sat-Count* will return 2 for z_1 and 0 for z_2, i.e. the Hamming distance is $0 + 2 = 2$. It can easily be checked that if $x \in \{0000, 0110\}$, the circuits provide different output values. The Hamming distance is obtained in linear time with respect to the number of outputs.

4 Proposed method

The proposed method is based on the standard CGP [19]. The main contribution of this work is redefining the fitness calculation procedure in such a way that it can handle circuits with tens to hundreds of inputs, and showing how various compromises between the error, area and delay can be found.

Because many candidate approximate circuits have to be generated and evaluated during a typical CGP run, it is impossible to evaluate everyone using a professional design tool. Hence circuit parameters are estimated. This strategy was validated in [35].

It is assumed that the specification (i.e. an accurate circuit behavior) is given in a form of ROBDD (let us denote it σ). If not, a corresponding ROBDD is created from the accurate circuit using the *apply* operator as described in Sect. 3.2.

4.1 Circuit representation

A gate-level n_i-input/n_o-output circuit is represented using a directed acyclic graph which is encoded in a 1D array consisting of n_c gates. The number of rows, which is one of CGP parameters, is set to $n_r = 1$. This graph is internally stored using a string of integers, the so-called chromosome. The set of available logic functions is denoted Γ. The primary inputs are labeled $0 \ldots n_i - 1$ and the gates are labeled $n_i, n_i + 1, \ldots, n_i + n_c - 1$. For each gate, the chromosome contains three integers—two labels specifying where the gate inputs are connected to and a code of function in Γ. The last part of the chromosome contains n_o integers specifying either the nodes where the primary outputs are connected to or logic constants (‘0’ and ‘1’) which can directly be connected to the primary outputs. Example is given in Fig. 3.

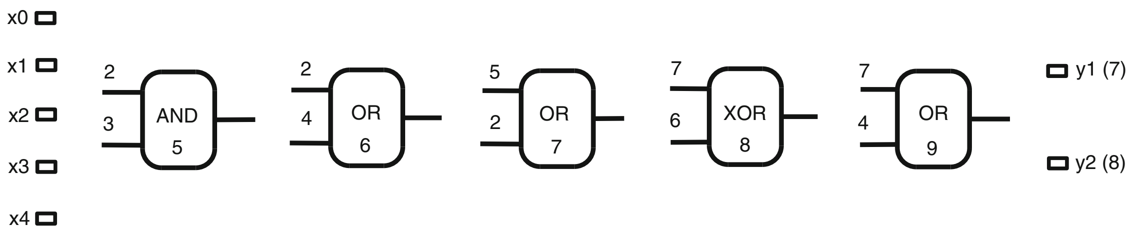


Fig. 3 Example of a circuit in CGP with parameters: $n_i = 5, n_o = 2,\ n_c = 5,\ \Gamma = \{0^{\text{and}}, 1^{\text{or}}, 2^{\text{xor}}\}$. Chromosome: 2, 3, $\underline{0}$; 2, 4, $\underline{1}$; 5, 2, $\underline{1}$; 7, 6, $\underline{2}$; 7, 4, $\underline{1}$; 7, 8. Gate 9 is not used. Its logic behavior is: $y_1 = (x_2 \text{ and } x_3) \text{ or } x_2$; $y_2 = y_1 \text{ xor } (x_2 \text{ or } x_4)$

The chromosome size is $3n_c + n_o$ genes (integers) if two-input gates are used. The main feature of this encoding is that the size of the chromosome is constant for a given n_i, n_o and n_c. However, the size of circuits represented by this chromosome is variable as some gates can remain disconnected. The gates which are included into the circuit after reading the chromosome are called the active gates.

4.2 From chromosome to ROBDD

Let circuit A be a candidate circuit represented using CGP. A new ROBDD, α, which is functionally equivalent with A has to be constructed. In order to do so, the number of BDD variables is defined firstly. Then, the *apply* function is called for every active gate N_j of circuit A. It consumes the logic function performed by N_j and two operands of N_j which are interpreted as pointers to appropriate ROBDD nodes or input variables. The function yields a pointer to a new ROBDD which represents the Boolean function at the output of gate N_j. Depending on the logic function of N_j, one or several new ROBDD nodes are thus included into α by means of one call of *apply*. The active nodes of A have to be processed from left to right in order to construct the ROBDD correctly.

4.3 Design objectives

There are three design objectives considered in this work: functionality (error), delay, and area.

4.3.1 Functionality

Circuit functionality is evaluated at the level of ROBDD and measured as the Hamming distance between the output bits generated by α and σ for all possible input combinations. The procedure follows the principle described and illustrated (Fig. 2) in Sect. 3.3. In particular, corresponding outputs of α and σ are connected to a set of exclusive-or gates, i.e. $z_i = y_i^\alpha$ xor y_i^σ for $i = 1, 2, \ldots, n_o$. By means of the *Sat-Count* operation, one can obtain the number of assignments b_i to the inputs which evaluate z_i to 1. Finally, the Hamming distance between α and σ, i.e. the fitness (functionality) of A, is the sum of b_i (see Eq. 1).

In order to accelerate this procedure, the ROBDD construction is optimized. We exploit the fact that the accurate circuit and candidate circuit (which was, in fact, created by a sequence of mutations over the accurate circuit) contain some identical subcircuits which can be removed for purposes of the Hamming distance calculation. ROBDD is then constructed using only those subcircuits which are not present in both circuits, i.e. the size of ROBDD is reduced.

4.3.2 Delay

In order to estimate the electrical parameters of circuit A, the area and delay are calculated using the parameters defined in the liberty timing file available for a given semiconductor technology. Delay of a gate is modeled as a function of its input transition time and capacitive load on the output of the gate. Delay of the whole circuit is determined as delay along the longest path.

4.3.3 Relative area

The area of a candidate circuit is calculated relatively to the area of a nand gate. The following gates are considered in Γ: *and, or, xor, nand, nor, xnor, buf, inv*, with corresponding relative areas 1.333, 1.333, 2, 1, 1, 2, 1.333, and 0.667. It is assumed that power consumption is highly correlated with the area and hence it is sufficient to optimize for the area as proposed in [35].

4.4 Search method

The search method follows the standard CGP approach [19]. The initial population is seeded by the accurate circuit. In order to generate a new population, λ offspring individuals are created by a point mutation operator modifying h genes of the parent individual. The parent is either the accurate circuit (in the first generation) or the best circuit of the previous generation (in remaining generations).

One mutation can affect either the gate function, gate input connection, or primary output connection. A mutation is called neutral if it does not affect the circuit's fitness. If a mutation hits a non-used part of the chromosome, it is detected and the circuit is not evaluated in terms of functionality because it has the same fitness as its parent. Otherwise, the error is calculated. The best individual of current population serves as the parent of new population. The process is repeated until a given number of generations (or evaluations) is not exhausted.

The role of mutation is significant in CGP (see detailed analysis in [8, 20]). A series of neutral mutations can accumulate useful circuit structures in the part of the chromosome which is not currently used. One adaptive mutation can then connect these structures with active gates which could lead to discovering new useful circuits. It has to be noted that the mutation operates over chromosomes (not at the level of BDDs).

In order to construct Pareto front, we follow the approach in which a single-objective CGP (utilizing a linear aggregation of objectives) is executed multiple times with different target errors e_i ($e_i > 0$). It is assumed that Pareto front has to be constructed for v different errors $e_1 \ldots e_v$ (each expressed as a percentage of the average Hamming distance). An obvious criticism of this approach is that some solutions are never obtained and a classic multi-objective EA such as NSGA-II has to be used. Despite the fact that some hybridizations of NSGA-II and CGP have been proposed [13, 16], our previous studies in the area of evolutionary circuit approximation have shown that single-objective approaches provide better results [35, 36].

We propose a two-stage procedure for evolving an approximate circuit showing target error e_i using a single-objective CGP.

The first stage starts with a fully functional solution which is always available in practice. The goal is to gradually modify the accurate circuit and produce an approximate circuit showing desired error e_i providing that a 5 % difference is tolerated with respect to e_i (tolerating a small error is acceptable; otherwise the search could easily stuck in a local extreme). The 5% error tolerance means that if, for example, $e_i = 0.3$ % then we accept all circuits showing the error 0.285...0.315 %. The fitness function fit_1 used in the first stage is thus solely based on the average Hamming distance (see Sect. 4.3.1),

$$fit_1 = Error(A) = \frac{\sum_{i=1}^{n_o} b_i}{2^{n_i}}. \tag{1}$$

In the second stage, which begins after obtaining a circuit with the target error, the fitness function reflects not only the error, but also the area and delay. Each objective is normalized to the interval $\langle 0, 1 \rangle$ and multiplied with weights w_e, w_a and w_d, respectively ($w_e + w_a + w_d = 1$). Then,

$$fit_2(A) = w_e Error(A) + w_a Area(A) + w_d Delay(A). \tag{2}$$

It is requested that the *Error* remains within 5 % tolerance with respect to e_i. Candidate circuits violating this hard constraint are discarded.

5 Results

This section firstly introduces benchmark circuits and CGP parameters used in all experiments. In order to check whether CGP can improve the results of conventional optimization and simultaneously obtain high-quality fully functional circuits which will be good starting points for subsequent approximations, CGP was employed to optimize the parameters of original (accurate) benchmark circuits. All results of approximations are represented as points in figures showing the objective space. Pareto fronts are constructed using the best obtained solutions.

5.1 Benchmarks

In order to evaluate the proposed approximation method which employs the Hamming distance to determine the error, we selected different types of combinational circuits from LGSynth, ITC and ISCAS libraries [7]. Even some arithmetic circuits (e.g. c3540) which should be approximated under an arithmetic error metric are included. The chosen circuits are difficult for the standard CGP, because $n_i > 25$ and more than 150 gates are involved [19].

At the beginning, all benchmark circuits were optimized using BDS [42] to get a reference solution from a "conventional" state-of-the art logic optimizer. Table 1 gives the number of primary inputs (n_i) and primary outputs (n_o), and then the number of gates (n_g) and delay (in terms of logic levels) after the optimization conducted by BDS. Table 1 also gives parameters of corresponding BDDs which serve as reference implementations used for Hamming distance calculations. The BDD size (see column |*BDD*|), obtained using [18], ranges from 321 nodes (itc_b10) to more than one million nodes (c3540). Because the size of BDD used as a

Table 1 Parameters of benchmark circuits

Circuit	Circuit parameters				Reference circuit			
	n_i	n_o	Gates	Levels	$\|BDD\|$	$\|BDD_{opt}\|$	$gain_{opt}$ (%)	t_{opt}
apex1	45	45	823	15	7073	1344	81	2.5
c1355	41	32	186	11	148,003	38,481	74	4.3
c3540	50	22	868	27	1,014,533	30,436	97	57.9
c432	36	7	159	25	167,300	1673	99	9.5
clmb[a]	46	33	641	19	6966	627	91	2.3
itc_b05[a]	34	56	427	24	18,788	1691	91	1.3
itc_b07[a]	49	49	312	25	11,055	995	91	3.1
itc_b10[a]	27	17	166	10	321	222	31	1.1
itc_b11[a]	37	31	421	18	1552	652	58	1.3
s1238[a]	31	31	483	18	1822	729	60	1.2
s635[a]	34	33	151	10	394	134	66	0.7
signet	39	8	630	17	11,471	1606	86	1.3
too_large	38	3	771	18	3508	807	77	3.3
x1dn	27	6	164	18	896	260	71	0.9
x6dn	39	5	318	14	3685	258	93	1.7
x9dn	27	7	168	20	484	218	55	0.5

The circuits that originally come from sequential benchmarks (i.e. represent a combinational subcircuit of a sequential circuit) are marked by superscript a

reference influences how fast the Hamming distance calculation and the whole evolutionary design can be, it is beneficial to optimize the BDD. The operations over small and optimized BDDs will then be performed faster than over original BDDs. Hence, we applyied *sifting* minimization algorithm to reduce the size of BDD. The results are reported in column $|BDD_{opt}|$. The improvement due sifting, which is 75.4 % on average, is given in the $gain_{opt}$ column. The last column t_{opt} is the time spent in the sifting procedure (in s). As it can be seen, the optimization of the variable order is able to ensure significant savings in the number of BDD nodes required to represent a reference circuit for a small cost of runtime. Note that this optimization is performed just once, before CGP is executed.

5.2 CGP setup

As the purpose of this paper is not to perform a detailed analysis of the CGP parameters setting, we used CGP with parameters that are usually reported in the literature. According to [26], the weights are chosen to be $w_e = 0.12$, $w_a = 0.5$, $w_d = 0.38$ and the CGP setup is: $\lambda = 4$, $h = 5$, Γ as defined in Sect. 4.3.3, and n_c is the number of gates in a particular benchmark circuit (according to Table 1). The experiments were conducted on a 64-b Linux machine running on Intel Xeon X5670 CPU (2.93 GHz, 12 MB cache) equipped with 32 GB RAM. CGP is implemented as a single-thread application.

5.3 Optimization of accurate circuits

First, CGP was employed to optimize the original benchmark circuits, i.e. no errors tolerated, $Error(A) = 0$. This step was performed because it has been known that a significant area reduction can be obtained by means of CGP [32].

A single CGP run was terminated after 30 min which seems to be a good compromise between requirements of practitioners expecting short optimization times and resources-demanding CGP. The number of generations was not specified as the termination criterion because the benchmark circuits have significantly different properties and, for example, different time is needed to process corresponding BDDs.

Table 2 gives parameters of the best and average circuit (obtained out of ten independent runs) with respect to parameters of original benchmarks: fitness (fit_2), the generation in which the best circuit was reached, and time to obtain the best circuit (runtime). It can be seen that the evolutionary optimization can lead in many cases to a significant delay, area (and, consequently, energy) reduction without introducing any error into the circuit. For example, an 80 % area improvement is reported for too_large benchmark circuit with respect to BDS. This particular circuit is hard for conventional optimization methods. Moreover, the parameters of the average circuits (determined as a median of all the runs) are close to the parameters of the best obtained circuit. C3540 and C1315 circuits represent the only exception where nearly none improvement is reported. A single gate was removed during optimization in both cases. Because the runtimes are close to the time limit, the majority of the benchmark circuits would be probably improved if more time is available to the evolution.

5.4 Evolutionary approximation

The circuits evolved in Sect. 5.3 will be used as (reference) accurate circuits in Pareto fronts. CGP-based approximations are performed from these accurate circuits for errors from $e_1 = 0.1$ to $e_9 = 0.9$ % given in terms of the Hamming distance. Results are presented from ten independent 30 min runs (one run one thread).

Firstly, we analyzed the first stage of the approximation. We calculated the time required to get an approximate circuit showing desired error e_i providing that an accurate circuit is used as a seed. In most cases, <1 s is required to find such a circuit. This corresponds with hundreds to few thousands of evaluated generations. Table 3 summarizes the cases in which more than 1 s is needed. The most difficult cases are c1355, c3540 and c432, and in particular 0.1–0.51 % error in case of c432, where some of the CGP runs spent more than 5 min. This was expected for c3540 and c1355 circuits because they are large and their BDDs are complex. A possible explanation for C432 is that the target error is too small. This findings is based on the fact that the number of evaluated generations is high (more than 15,000 genenerations) and that the achieved reduction in the area is low compared to the other benchmarks (see Fig. 4). We can, however, conclude that desired approximations can be reached relatively quickly if the other objectives are not taken into account. The chosen search strategy seems to be very efficient in this task despite

Table 2 Parameters of the best and average accurate circuits obtained using CGP

Circuit	Best obtained circuit						Average circuit (median)				
	fit$_2$	Generation	Runtime (min)	Improvement			*fit*$_2$	Runtime (min)	Improvement		
				Gates (%)	Area (%)	Delay (%)			Gates (%)	Area (%)	Delay (%)
apex1	0.96	1.6×10^6	28	3	3	5	0.96	26	2	3	5
c1355	1.00	7.6×10^3	28	<1	<1	0	1.00	21	0	0	<1
c3540	1.00	4.9×10^3	11	<1	<1	0	1.00	0	0	0	0
c432	0.97	63.4×10^3	18	2	3	2	0.97	17	1	2	3
clmb	0.65	2.9×10^6	29	36	36	26	0.70	29	31	32	20
itc_b05	0.94	0.5×10^6	29	5	6	1	0.95	28	4	5	1
itc_b07	0.97	0.4×10^6	7	<1	1	7	0.97	8	<1	1	6
itc_b10	0.91	3.7×10^6	13	7	6	18	0.91	21	7	6	16
itc_b11	0.96	2.5×10^6	26	4	4	1	0.97	29	3	3	1
s1238	0.85	2.8×10^6	28	13	15	12	0.86	29	13	14	13
s635	0.94	10.0×10^6	25	3	2	24	0.96	24	4	3	6
signet	0.58	1.2×10^6	29	43	45	30	0.62	28	41	42	18
too_large	0.30	1.0×10^6	29	79	80	28	0.33	25	76	77	26
x1dn	0.51	4.7×10^6	20	48	50	45	0.63	27	32	36	39
x6dn	0.90	2.9×10^6	25	11	12	3	0.92	26	8	9	3
x9dn	0.80	3.5×10^6	24	20	23	8	0.82	15	20	20	8

Table 3 Time (in s) spent in the first stage of the optimization

Circuit	$e_1 = 0.1$ %			$e_3 = 0.3$ %			$e_5 = 0.5$ %			$e_7 = 0.7$ %			$e_9 = 0.9$ %		
	t_{Q1}	t_{med}	t_{Q3}	t_{Q1}	t_{med}	t_{Q3}	t_{Q1}	t_{med}	t_{Q3}	t_{Q1}	t_{med}	t_{Q3}	t_{Q1}	t_{med}	t_{Q3}
c1355	9.2	27	82	32	58	100	12	48	383	12	19	30	3.2	3.2	3.2
c3540	5.9	86	168	17	70	145	7.2	17	36	18	24	49	7.7	29	56
c432	10	33	305	24	41	478	5.8	173	492	4.8	50	223	16	33	201
itc_b05	<1	<1	<1	<1	<1	2.2	<1	<1	1.6	<1	<1	1.3	<1	<1	2.3
itc_b07	<1	<1	1.3	<1	<1	1.3	<1	<1	<1	<1	<1	<1	<1	<1	<1
x1dn	<1	<1	<1	<1	<1	<1	<1	<1	4.1	<1	<1	<1	<1	<1	<1

The median value (t_{med}), the lower bound (t_{Q1}) and upper bound (t_{Q3}) of the interquartile range are calculated using all runs

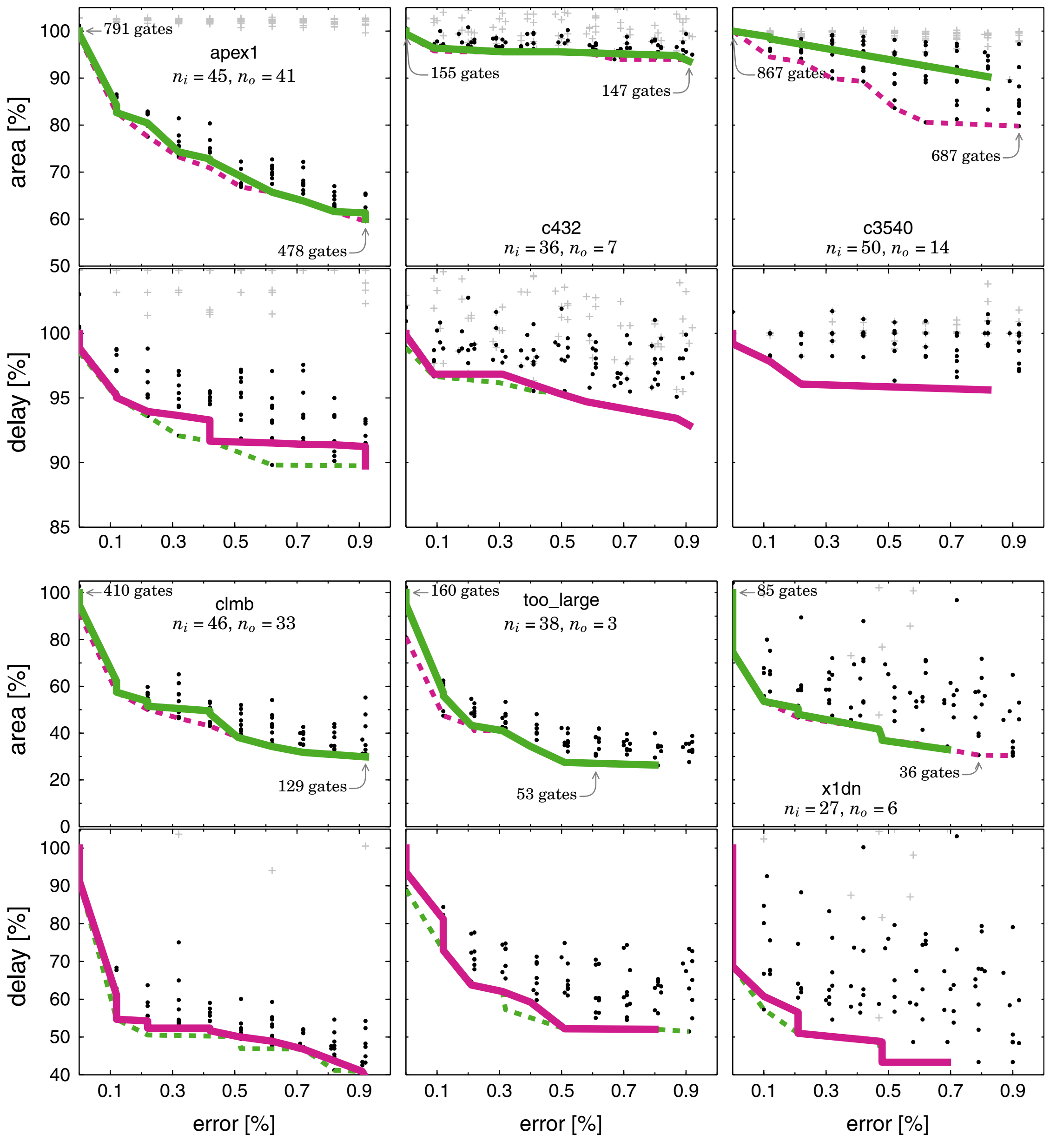


Fig. 4 Pareto fronts for benchmarks apex1, c432, c3540, clmb, too_large and x1dn

the fact that finding a circuit exhibiting a required error is in general a nontrivial task.

Next we measured the time needed to calculate the Hamming distance between two circuits with respect to a given error. This time (expressed as a median value) summarized from all the experiments is given in Table 4. This value influences how many candidate solutions can be evaluated within a period of time in average. The lower value, the higher number of evaluated candidate solutions. Stages 1 (searching for a circuit with a given error) and 2 (optimizing the area, and delay of the circuit) are handled separately. In most cases, the time is less than a few milliseconds. There are two cases (c1355, c3540) in which few 100s (1000s for c1355) of milliseconds are required to determine the Hamming distance. A consequence is that fewer generations can be produced within a given time for this circuit. This effect is clearly visible in Fig. 5 where the resulting circuit approximations are mostly far from the optimum (see black dots for errors higher than 0.4 %).

If we compare the mean time needed to determine the Hamming distance in the first and second stage (see last five columns of Table 4 showing the ratio between the first and second stage), it is evident that more evaluations per second can be performed in the second stage of the optimization. The reason is that BDDs are in average smaller than in the first stage. However, it can be also seen that the time needed to evaluate the Hamming distance of c1355 benchmark circuit increases with the increasing error. Thousands of milliseconds are needed in this particular case.

The resulting Pareto fronts are displayed in Figs. 4, 5, 6 and 7 (solid lines). For each circuit, two plots are presented: the best obtained area versus error and delay versus error, relatively to the fully functional circuit from Table 2 labeled by 100 %. The result of a single 30-min CGP run consisting of two stages is shown using a black dot. The plus symbol (+) indicates the results of the first stage. In several cases, the + symbols are not visible because they are outside the plotted areas. However, for example, the plot for s635 clearly shows that in most cases the first stage produced circuits within about 80–100 % in the area axis (corresponding to <1 s in average according to Table 3) while the second stage led in remaining 29.9 min (on average) to a significant improvement (about 30 % in the area axis).

Figures 4, 5, 6 and 7 contain the best compromises from independent CGP runs in which all circuits parameters (error, area and delay) are optimized together. For some applications it is interesting to know the best compromises for two objectives only (error vs. delay and area vs. delay). These compromises are plotted as additional Pareto fronts (dashed line). The number of gates is explicitly given for the biggest and smallest circuits.

For example, by increasing the error, the area was reduced by 70 % in the case of clmb circuit providing that the delay is adequately reduced. A general observation is that an improvement in delay is smaller than in area when the error is increasing. The reason is that fully functional benchmark circuits exhibit a relatively small delay and hence there is a little space to improve it. Two interesting cases are c432 and c3540 because their area requirements are high even for reduced accuracy. One reason could be that errors <1 % are too small to get a reasonable approximation.

Table 4 Time (in ms) required to calculate the Hamming distance in stage 1 (t_{med}, t_{Q3}) and stage 2 (t^*_{med})

Circuit	$e_1 = 0.1$ %		$e_3 = 0.3$ %		$e_5 = 0.5$ %		$e_7 = 0.7$ %		$e_9 = 0.9$ %		t_{med}/t^*_{med}				
	t_{med}	t_{Q3}	t_{med}	t_{Q3}	t_{med}	t_{Q3}	t_{med}	t_{Q3}	t_{med}	t_{Q3}	e_1	e_3	e_5	e_7	e_9
apex1	1.3	1.5	1.2	1.5	1.2	1.4	1.4	1.9	1.4	1.6	1.4	1.3	1.8	1.7	2.0
c1355	198	215	278	389	640	3083	583	704	665	665	1.5	4.6	0.3	0.1	0.1
c3540	162	200	173	192	155	161	195	215	197	216	1.0	1.1	1.1	1.3	1.3
c432	16	17	16	17	19	20	16	18	17	19	1.3	1.2	1.3	1.2	1.5
clmb	0.7	1.1	1.2	1.6	1.3	2.1	1.5	2.5	1.1	1.4	3.7	6.0	6.7	14	10
itc_b05	5.2	5.5	4.5	5.8	8.6	11	5.5	7.9	5.2	5.8	2.3	2.4	5.7	3.2	4.4
itc_b07	1.2	1.6	1.2	1.6	1.3	1.9	1.2	1.3	1.4	1.7	2.0	1.7	2.1	2.3	2.5
itc_b10	0.2	0.3	0.2	0.2	0.2	0.2	0.2	0.3	0.2	0.2	1.0	1.0	0.8	1.0	1.0
itc_b11	0.6	0.7	0.6	0.8	0.6	0.9	0.5	0.6	0.7	0.8	1.1	1.1	1.2	1.1	1.4
s1238	0.7	0.7	0.8	0.9	0.7	0.9	0.7	1.2	0.7	0.8	1.7	2.0	2.4	2.4	2.2
s635	0.2	0.3	0.1	0.3	0.2	0.3	0.3	0.8	0.2	0.3	2.2	1.3	2.2	2.6	2.0
signet	1.9	2.6	2.2	2.5	2.8	3.6	2.7	4.0	2.2	2.7	1.5	1.8	2.3	2.4	2.5
too_large	7.9	8.8	8.5	11	8.7	10	12	17	8.4	11	22	28	28	41	42
x1dn	0.4	0.6	0.3	0.4	0.3	0.4	0.8	1.1	0.3	0.6	2.7	3.1	3.5	5.1	3.5
x6dn	0.2	0.4	0.3	0.4	0.3	0.4	0.4	0.5	0.2	0.3	0.6	0.9	1.0	1.1	1.1
x9dn	0.5	0.7	0.5	0.5	0.3	0.5	0.6	1.2	0.4	0.5	1.6	1.8	1.7	3.2	3.9

The median value (t_{med}, t^*_{med}) and upper bound (t_{Q3}) of the interquartile range are calculated from run-time of all fitness function calls

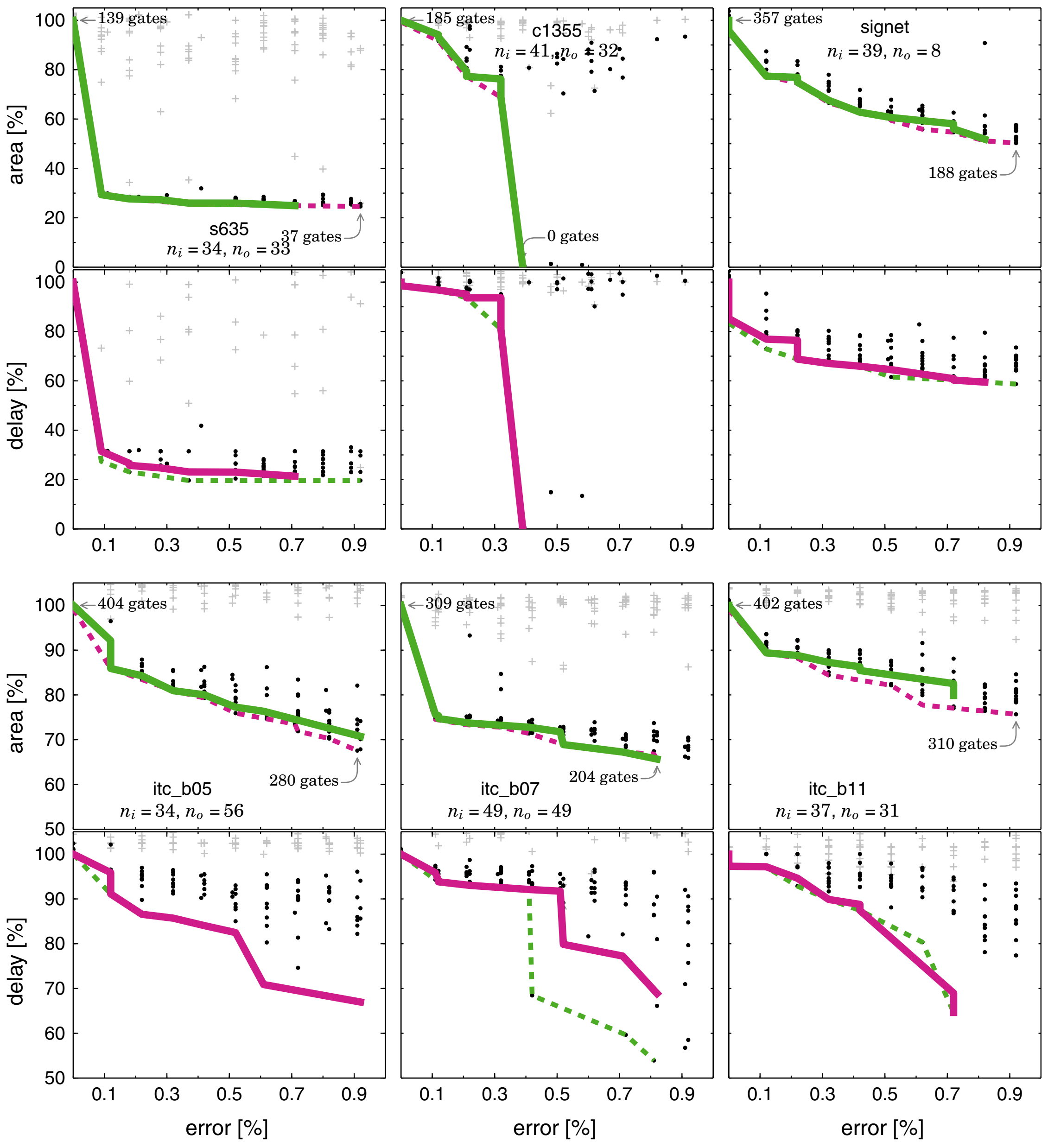


Fig. 5 Pareto fronts for benchmarks s635, c1355, signet, itc_b05, itc_b07 and itc_b11

Another reason could be that more generations are required to reduce the area. Hence we tried to prolong the evolution six times. However, no significant changes in the Pareto front have been observed (not shown in the paper). On the other hand, 0.4 % seems to be a huge error for c1355 because it allowed CGP to remove almost all gates from the circuit.

In some cases (e.g. apex1, c3540, s635), the independent CGP runs led to very similar results for a given e_i (see the black dots). In other cases (e.g. itc_b05, x1dn, c1355), the spread in the area and delay is quite large. This indicates that circuits have different structural properties and that their selection to our benchmark set is justified.

Figure 7 shows a detailed analysis of some of the discovered approximations for itc_b10 benchmark. For each of five target errors, three evolved circuits were

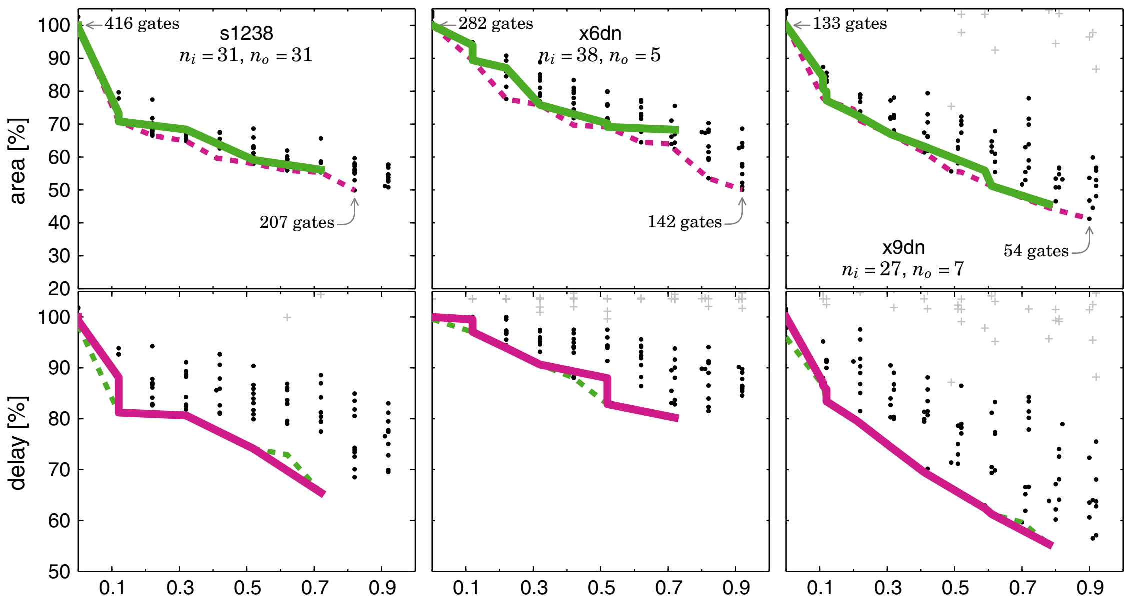


Fig. 6 Pareto fronts for benchmarks s1238, x6dn, and x9dn

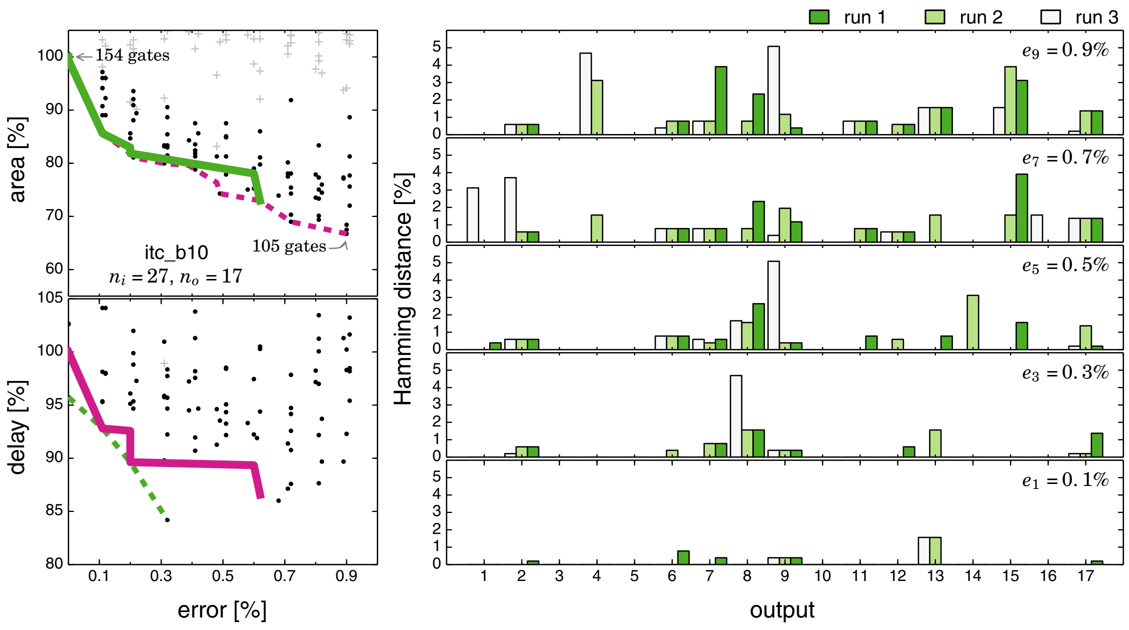


Fig. 7 Pareto front for benchmark itc_b10 (*left*) and the detailed analysis (Hamming distance of each output) of three evolved approximations for e_1, e_3, e_5, e_7 and e_9 (*right*)

chosen and their Hamming distances were calculated independently for all 17 outputs. It can be seen that the obtained solutions have in general different properties. The same solution was obtained only in the case of $e_1 = 0.1$ %, where run 2 and run 3 discovered a circuit with error in two outputs. The first run produced a completely different circuit. Although five output signals are affected by error in this case, the worst-case difference (Hamming distance) is not worse than 0.7 %.

Finally, we raised a question whether it is better to intensively optimize accurate circuits or introduce approximations in order to reduce the area. Table 5 shows how

Table 5 The number of gates when subsequent optimizations are applied

Method	BDS	CGP			
Error	0 %	0 %	0.1 %	0.5 %	0.9 %
clmb	641 (100 %)	410 (63 %)	250 (39 %)	167 (26 %)	129 (20 %)
s1238	483 (100 %)	416 (86 %)	298 (61 %)	241 (49 %)	213 (44 %)
signet	630 (100 %)	357 (56 %)	288 (45 %)	223 (35 %)	188 (29 %)
too_large	771 (100 %)	160 (20 %)	94 (12 %)	55 (7 %)	56 (7 %)
x1dn	164 (100 %)	85 (51 %)	61 (37 %)	44 (26 %)	36 (21 %)
x6dn	318 (100 %)	282 (88 %)	254 (79 %)	196 (61 %)	142 (44 %)
x9dn	168 (100 %)	133 (79 %)	103 (61 %)	72 (42 %)	54 (32 %)

subsequent optimizations reduced the number of gates in circuits displaying at least 10 % area improvement with respect to BDS. For example, CGP-based optimization of too_large circuit caused that 80 % gates were removed without any impact on the accuracy. A subsequent approximation (error = 0.1 %) removed only 8 % gates. It turns out that the impact of a proper logic optimization conducted in the standard scenario (no errors are allowed) can be, in fact, higher than when the approximations are introduced.

6 Conclusions

In this paper, we proposed a new CGP-based method which allowed us to approximate non-trivial combinational circuits. Employing a BDD package in the fitness function enabled to reduce the fitness evaluation time, which is the most contributing component to the total time of evolution. The error was expressed in terms of the Hamming distance—the error measure which can be applied for general logic approximation. Pareto fronts show reasonable tradeoffs between key circuit parameters which one would expect for combinational approximate circuits. Unfortunately, no results compatible with our scenario are available in the literature for comparison.

Our method consists of two stages. In the first stage, a circuit showing desired error is evolved from a fully functional solution. As the initial approximation is performed in order of seconds (10s–100s of seconds in the case of more complex circuits), the user thus quickly obtains a circuit with desired functionality. Additional optimizations of the area and delay are then performed in the second stage which can be terminated when a suitable tradeoff is reached. This approach allowed us to find high quality solutions in a relatively short time, which is important for practice. It was also shown that a significant area reduction can be obtained just by enabling the evolutionary optimization of the accurate circuit after performing its usual conventional optimization (Table 2).

Despite the fact that by means of BDDs we were able to approximate relatively complex circuits (10s of inputs, 100s of gates), the usage of BDDs represents an inherent weakness of the method. As we pointed out in Sect. 3, BDDs can grow exponentially for some functions. Hence it is important to find a more suitable formal model and corresponding algorithms which will allow us to further extend the class of circuits that can be approximated by CGP.

In our future work, we also plan to combine our method with a truly multiobjective evolutionary algorithm in order to obtain a Pareto front in a single run. We will evaluate if the overhead associated with the multiobjective optimizer can lead to results which are competitive with the obtained ones.

Acknowledgments This work was supported by the Czech science foundation Project 14-04197S.

References

1. R.E. Bryant, On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. IEEE Trans. Comput. **40**(2), 205–213 (1991)
2. R. Drechsler, B. Becker, N. Göckel, Genetic algorithm for variable ordering of obdds. IEE Proc. Comput. Digit. Tech. **143**(6), 364–368 (1996)
3. R. Drechsler, N. Göckel, B. Becker, in *Learning Heuristics for OBDD Minimization by Evolutionary Algorithms*. Parallel Problem Solving from Nature—PPSN IV. Lecture Notes in Computer Science, vol. 1141 (Springer, Berlin, 1996), pp. 730–739
4. M. Duranton, K. DeBosschere, A. Cohen, J. Maebe, H. Munk, in *Hipeac Vision 2015*. Technical report (HiPEAC Network of Excellence, 2015). https://www.hipeac.net/publications/vision/
5. R. Ebendt, G. Fey, R. Drechsler, *Advanced BDD Optimization* (Springer, Berlin, 2000)
6. H. Esmaeilzadeh, A. Sampson, L. Ceze, D. Burger, Neural acceleration for general-purpose approximate programs. Commun. ACM **58**(1), 105–115 (2015)
7. P. Fiser, *Collection of Digital Design Benchmarks* (Czech Technical University in Prague, Prague). http://ddd.fit.cvut.cz/prj/Benchmarks
8. B.W. Goldman, W.F. Punch, Analysis of cartesian genetic programming's evolutionary mechanisms. IEEE Trans. Evol. Comput. **19**(3), 359–373 (2015)
9. D. Grochol, L. Sekanina, M. Zadnik, J. Korenek, V. Kosar, Evolutionary circuit design for fast FPGA-based classification of network application protocols. Appl. Soft Comput. **38**, 933–941 (2016). doi:10.1016/j.asoc.2015.09.046
10. V. Gupta, D. Mohapatra, A. Raghunathan, K. Roy, Low-power digital signal processing using approximate adders. IEEE Trans. CAD Integr. Circuits Syst. **32**(1), 124–137 (2013)
11. J. Han, M. Orshansky, in *Approximate Computing: An Emerging Paradigm for Energy-Efficient Design*. Proceedings of the 18th IEEE European Test Symposium (IEEE, 2013), pp. 1–6
12. T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, T. Furuya, in *Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine*. Proceedings of the 2nd International Conference on Simulated Adaptive Behaviour (MIT Press, Cambridge, 1993), pp. 417–424
13. R. Hrbacek, in *Parallel Multi-objective Evolutionary Design of Approximate Circuits*. Proceedings of the 2015 Conference on Genetic and Evolutionary Computation (GECCO '15) (ACM, 2015), pp. 687–694
14. R. Hrbacek, L. Sekanina, in *Towards Highly Optimized Cartesian Genetic Programming: From Sequential Via SIMD and Thread to Massive Parallel Implementation*. Proceedings of the Conference on Genetic and Evolutionary Computation (ACM, 2014), pp. 1015–1022
15. M. Iqbal, W.N. Browne, M. Zhang, Reusing building blocks of extracted knowledge to solve complex, large-scale Boolean problems. IEEE Trans. Evol. Comput. **18**(4), 465–480 (2014)
16. P. Kaufmann, T. Knieper, M. Platzner, in *A Novel Hybrid Evolutionary Strategy and Its Periodization with Multi-objective Genetic Optimizers*. IEEE Congress on Evolutionary Computation (CEC) (IEEE, 2010), pp. 1–8
17. P. Kulkarni, P. Gupta, M.D. Ercegovac, Trading accuracy for power in a multiplier architecture. J. Low Power Electron. **7**(4), 490–501 (2011)
18. J. Lind-Nielsen, H. Cohen, in *BuDDy—A Binary Decision Diagram Package*. http://sourceforge.net/projects/buddy/
19. J.F. Miller, *Cartesian Genetic Programming* (Springer, Berlin, 2011)
20. J.F. Miller, S.L. Smith, Redundancy and computational efficiency in cartesian genetic programming. IEEE Trans. Evol. Comput. **10**(2), 167–174 (2006)

21. A. Mishchenko, in *ABC: A System for Sequential Synthesis and Verification* (Berkeley Logic Synthesis and Verification Group, University of California, Berkeley, CA, US, 2012). http://www.eecs.berkeley.edu/~alanmi/abc/
22. M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, T. Higuchi, in *Evolvable Hardware at Function Level*. Parallel Problem Solving from Nature (PPSN IV). LNCS, vol. 1141 (Springer, Berlin, 1996), pp. 62–71
23. K. Nepal, Y. Li, R.I. Bahar, S. Reda, in *Abacus: A Technique for Automated Behavioral Synthesis of Approximate Computing Circuits*. Proceedings of the Conference on Design, Automation and Test in Europe (DATE '14) (EDA Consortium, 2014), pp. 1–6
24. R. Poli, J. Page, Solving high-order Boolean parity problems with smooth uniform crossover, sub-machine code GP and demes. Genet. Program. Evol. Mach. **1**(1–2), 37–56 (2000)
25. A. Sanchez-Clemente, L. Entrena, M. Garcia-Valderas, in *Error Masking with Approximate Logic Circuits Using Dynamic Probability Estimations*. 20th International On-Line Testing Symposium (IOLTS) (IEEE, 2014), pp. 134–139
26. L. Sekanina, Z. Vasicek, in *Evolutionary Computing in Approximate Circuit Design and Optimization*. 1st Workshop on Approximate Computing (WAPCO 2015) (2015), pp. 1–6
27. A.P. Shanthi, R. Parthasarathi, Practical and scalable evolution of digital circuits. Appl. Soft Comput. **9**(2), 618–624 (2009)
28. E. Stomeo, T. Kalganova, C. Lambert, Generalized disjunction decomposition for evolvable hardware. IEEE Trans. Syst. Man Cybern. Part B **36**(5), 1024–1043 (2006)
29. A. Thompson, P. Layzell, S. Zebulum, Explorations in design space: unconventional electronics design through artificial evolution. IEEE Trans. Evol. Comput. **3**(3), 167–196 (1999)
30. J. Torresen, A scalable approach to evolvable hardware. Genet. Program. Evol. Mach. **3**(3), 259–282 (2002)
31. Z. Vasicek, in Cartesian GP in *Optimization of Combinational Circuits with Hundreds of Inputs and Thousands of Gates*. Proceedings of the 18th European Conference on Genetic Programming—EuroGP, LCNS no. 9025 (Springer, Berlin, 2015), pp. 139–150
32. Z. Vasicek, L. Sekanina, Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. Genet. Program. Evol. Mach. **12**(3), 305–327 (2011)
33. Z. Vasicek, L. Sekanina, in *Evolutionary Design of Approximate Multipliers Under Different Error Metrics*. IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (IEEE, 2014), pp. 135–140
34. Z. Vasicek, L. Sekanina, in *How to Evolve Complex Combinational Circuits from Scratch?* IEEE International Conference on Evolvable Systems Proceedings (IEEE, 2014), pp. 133–140
35. Z. Vasicek, L. Sekanina, in *Circuit Approximation Using single- and Multi-objective Cartesian GP*. Proceedings of the 18th European Conference on Genetic Programming— (EuroGP), LNCS no. 9025 (Springer, Berlin, 2015), pp. 217–229
36. Z. Vasicek, L. Sekanina, Evolutionary approach to approximate digital circuits design. IEEE Trans. Evol. Comput. **19**(3), 432–444 (2015)
37. Z. Vasicek, L. Sekanina, in *Evolutionary Approximation of Complex Digital Circuits*. Genetic and Evolutionary Computing Conference (ACM, 2015), pp. 1505–1506
38. V. Vassilev, D. Job, J.F. Miller, in *Towards the Automatic Design of More Efficient Digital Circuits*. Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware (IEEE Computer Society, Los Alamitos, 2000), pp. 151–160
39. S. Venkataramani, K. Roy, A. Raghunathan, in *Substitute-and-Simplify: A Unified Design Paradigm for Approximate and Quality Configurable Circuits*. Design, Automation and Test in Europe (DATE '13) (EDA Consortium, San Jose, 2013), pp. 1367–1372
40. S. Venkataramani, A. Sabne, V.J. Kozhikkottu, K. Roy, A. Raghunathan, in *SALSA: Systematic Logic Synthesis of Approximate Circuits*. The 49th Annual Design Automation Conference (DAC '12) (ACM, 2012), pp. 796–801
41. R. Venkatesan, A. Agarwal, K. Roy, A. Raghunathan, in *MACACO: Modeling and Analysis of Circuits for Approximate Computing*. IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (IEEE, 2011), pp. 667–673
42. C. Yang, M. Ciesielski, BDS: a BDD-based logic optimization system. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **21**(7), 866–876 (2002)

Appendix E

Trading between Quality and Non-functional Properties of Median Filter in Embedded Systems

VASICEK, Zdenek and MRAZEK, Vojtech. “Trading between Quality and Non-functional Properties of Median Filter in Embedded Systems”. In: *Genetic Programming and Evolvable Machines* 2016.3 (2016), pp. 1–38.

IF=1.143 (2015), contribution of the author of the thesis: 50%

Referenced on pages: 41, 43

Genet Program Evolvable Mach
DOI 10.1007/s10710-016-9275-7

Trading between quality and non-functional properties of median filter in embedded systems

Zdenek Vasicek[1] · Vojtech Mrazek[1]

Received: 20 December 2015 / Revised: 6 July 2016

Abstract Genetic improvement has been used to improve functional and non-functional properties of software. In this paper, we propose a new approach that applies a genetic programming (GP)-based genetic improvement to trade between functional and non-functional properties of existing software. The paper investigates possibilities and opportunities for improving non-functional parameters such as execution time, code size, or power consumption of median functions implemented using comparator networks. In general, it is impossible to improve non-functional parameters of the median function without accepting occasional errors in results because optimal implementations are available. In order to address this issue, we proposed a method providing suitable compromises between accuracy, execution time and power consumption. Traditionally, a randomly generated set of test vectors is employed so as to assess the quality of GP individuals. We demonstrated that such an approach may produce biased solutions if the test vectors are generated inappropriately. In order to measure the accuracy of determining a median value and avoid such a bias, we propose and formally analyze new quality metrics which are based on the positional error calculated using the permutation principle introduced in this paper. It is shown that the proposed method enables the discovery of solutions which show a significant improvement in execution time, power consumption, or size with respect to the accurate median function while keeping errors at a moderate level. Non-functional properties of the discovered solutions are estimated using data sets and validated by physical measurements on physical microcontrollers. The benefits of the evolved implementations are demonstrated on two real-

✉ Zdenek Vasicek
vasicek@fit.vutbr.cz

Vojtech Mrazek
imrazek@fit.vutbr.cz

[1] Faculty of Information Technology, IT4Innovations Centre of Excellence, Brno University of Technology, Brno, Czech Republic

Published online: 19 July 2016

world problems—sensor data processing and image processing. It is concluded that data processing software modules offer a great opportunity for genetic improvement. The results revealed that it is not even necessary to determine the median value exactly in many cases which helps to reduce power consumption or increase performance. The discovered implementations of accurate, as well as approximate median functions, are available as C functions for download and can be employed in a custom application (http://www.fit.vutbr.cz/research/groups/ehw/median).

Keywords Genetic programming · Genetic improvement · Cartesian genetic programming · Median function · Comparison network · Permutation principle · Median filter

1 Introduction

Genetic programming (GP) has traditionally been used to evolve entirely new expressions or functions to solve a particular problem which is usually specified by a training data set [23]. With the development of search based software engineering, GP has been applied to repair errors in software and assist in numerous tasks of software engineering [10]. The successful applications of GP in search based software engineering have attracted more and more researchers which has resulted in the establishment of a new research direction called *Genetic improvement* (GI). The Genetic improvement of software is defined as the application of evolutionary and search-based optimization methods with the aim of improving *functional* and/or *non-functional* properties of existing software [38].

The number of lines of code, execution time, memory usage, or power consumption represent the typical *non-functional* properties of software. These properties can be improved, for example, by replacing the existing code fragments by newly evolved code fragments that are semantically equivalent. The improvement of existing software by optimizing its non-functional properties was first addressed by White et al. [38]. Eight different target functions were considered in the paper. The authors showed that GP was able to discover code optimization tricks that are probably unreachable by current compilers. These tricks enabled slight improvements in the execution time of the chosen functions. Recently, Cody-Kenny et al. [3] demonstrated that GP was able to reduce the number of instructions for various manually constructed off-the-shelf implementations of a sort and prefix-code programs written in Java. Genetic Improvement has also been used to evolve an improved version of C++ code using automated code transplantation [22]. The authors evolved a faster version of Boolean satisfiability solver MiniSAT which is specialized for solving a particular problem known as Combinatorial interaction testing. Finally, the reduction of energy consumption of non-trivial programs was addressed in [26]. The authors introduced a system which can further optimize the low level Intel X86 code generated by optimizing compilers. While the previous examples have dealt with non-functional improvements, Langdon and Harman showed that GP can, in addition to non-functional parameters, improve the functionality of existing code [15]. The authors demonstrated that GP was able to automatically improve behaviour (i.e. accuracy) of a widely-used DNA sequencing system consisting of 50,000 lines of C++ code.

A similar research direction has been explored in the field of approximate computing which is a promising approach to obtain energy-efficient computer systems. In constrast to Genetic improvements that preserve the code functionality, approximate computing exploits the fact that many applications are error resilient and do not require a perfect output to be produced. Hence a suitable compromise is sought between the error (quality), power consumption and performance. The approximations can be introduced at the level of hardware as well as software. An approximate solution is typically obtained by a heuristic procedure that modifies the original implementation. For instance, artificial neural networks were used to approximate software modules [7] in order to accelerate computations and reduce power consumption. In addition, search-based methods were allowed to approximate hardware components [21, 36]. Using GP in the context of approximate computing has been reported for digital circuit approximation [29, 34].

In this paper, we deal with GP-based improvements of non-functional properties of programs (C functions) that are intended for low-cost microcontrollers. As we seek significant improvements mainly of power consumption and execution time, we consider the approximate computing scenario and accept some errors in the outputs. The function to be approximated is the median filter which is crucial in signal processing, image processing and sensor data processing. The goal of the GP-based search strategy is to improve the existing, in most cases even functionaly optimal, median implementations and find programs showing suitable compromises between the accuracy, execution time and power consumption for various median functions when implemented on a microcontroller. This paper develops our previous results that were presented at the first GI workshop [20]. In contrast to our previously published work, a more efficient GI method based on a two-stage optimization process is introduced. The quality of the candidate solutions is measured as a distance between the candidate program and a fully-working median implementation. A generalized version of the zero–one principle [14] denoted as *permutation principle* is used to determine this distance. The permutation principle, first introduced in this paper, is formally proven in Sect. 4.2. Compared to the previous results, the quality of the obtained solutions is improved significantly. In addition, the benefit of GI approach is demonstrated using two real-world problems that are typically handled by embedded systems—sensor data processing and image processing. The obtained results are evaluated using data sets and by physical measurements on physical microcontrollers.

In the context of this work, one can observe that the evolutionary design and/or optimization of an (accurate) median outputting program has been carried out by GP only rarely [27]. However, a considerable number of research papers were devoted to the design and optimization of sorting algorithms (e.g., [1, 38]) and sorting networks (e.g. [11, 12, 28, 33]), which are useful structures when the median value has to be obtained. As checking whether a specification (i.e. an original code) and a candidate solution are semantically equivalent is time consuming, exact equivalence checking is not performed in the fitness function. The fitness is usually based on evaluating candidate solutions using a training data set and subsequent testing at the end of evolution using other data sets. According to the zero–one principle, the training data are typically restricted to binary input vectors. A genetic improvement of two different implementations of Bubble-Sort algorithm was demonstrated in [38], where GP enabled the discovery of code optimization tricks probably unreachable by current compilers. These tricks enabled slightly improved execution time of the chosen sorting functions.

The rest of the paper is organized as follows. Section 2 briefly surveys genetic improvement and its relation to the approximate computing. Then an overview of the key areas related to this paper is given in Sect. 3. In particular, the median function and possibilities for the improvement are discussed. In Sect. 4, the permutation principle is introduced. The proposed method is described in Sect. 5. Section 6 introduces the experimental setup. The results are presented and analyzed in Sect. 7. Then, the obtained medians are applied to solve real-world problems. A detailed discussion is given in Sect. 8. Finally, Sect. 9 concludes the paper.

2 From genetic improvement to approximate computing

In contrast to approximate computing that has been developed to improve energy efficiency and performance for the cost of accuracy, GI has always kept the code functionality identical with the original software. In approximate computing, software and hardware is approximated (i.e. simplified with respect to fully accurate implementations) in order to reduce power consumption or increase performance. As a consequence, errors can emerge during computations. In many cases errors can be tolerated because human perception capabilities are limited, no golden solution is available for validation of results, or users are willing to accept some inaccuracies. Therefore, the error (accuracy of computations) can be used as a design metric and traded for area on a chip, delay, throughput, or power consumption.

One way to reduce energy consumption is by allowing timing errors by voltage over scaling or frequency over clocking. Another approximation technique, which is relevant for this paper, is *functional approximation*. The idea of functional approximation is to implement a slightly different function to the original one, provided that the error is acceptable and the non-functional parameters are improved adequately. Functional approximation can be conducted at the level of software as well as hardware.

After introducing several approximate circuits that were created manually [9], researchers started to develop more efficient systematic semi-automatic and fully-automatic methods. EnerJ [24], an extension of Java that adds approximate data types, represents one of the semi-automatic methods. Using these types, the system automatically maps approximate variables to low-power storage, uses low-power operations, and even applies more energy-efficient algorithms provided by the programmer. Axilog is a set of language annotations that provide the necessary syntax and semantics for an approximate hardware design and reuse in Verilog [39]. Axilog enables the designer to relax the accuracy requirements in certain parts of the design, while keeping the critical parts strictly precise. In contrast to fully-automatic methods, an approximate solution is typically obtained by a heuristic procedure that modifies the original implementation. For example, artificial neural networks were proposed in [7] to learn to behave like a general-purpose code written in an imperative language. The trained network then replaced the original code. There are also general search-based methods that allows us to approximate hardware components [21, 36].

While the approximate problem has already been addressed by GP community [30, 34], GI has been used to improve functional and non-functional properties of software so far. However, by having the fitness function of GP-based GI permit errors, one can easily obtain approximate solutions. Applying the GI methodology for approximate computing (particularly for approximate software) seems to be straightforward. The main outcomes would be to obtain better trade-offs among key system parameters (note that the search-based methods are not constrained by various assumptions of mathematically rigorous methodologies) and reducing the optimization time with respect to commonly used solvers such as ILP. The key advantage is that the GI systems can be constructed as multi-objective (i.e. they provide a Pareto front showing the best trade-offs among the error, speed, memory usage, energy consumption, network loading, etc.) at the end of each run.

3 Background

In this section, we give an overview of the key areas related to this paper, especially the median function, construction of median networks and possibilities for the improvement of median functions. The section is concluded with problem formulation.

3.1 Median of a data set

Given a finite sequence of data samples, the median is defined as a value separating the higher half of data samples from the lower half. The median is of central importance in robust statistics [16], as it is the statistic that is the most resistant to outliers that could be presented in a given sequence. Contrasted to the mean, the median is a robust measure of central tendency. The main feature of the robust methods is their high efficiency in a neighbourhood of the assumed statistical model which is widely exploited in signal processing where the median is usually employed to filter the measured data.

There exists two basic approaches to determine the median of a given sequence. A straightforward and naïve approach is to employ a generic sorting algorithm, for example, the most popular and efficient quicksort algorithm. Implementations of sorting algorithms are very compact and robust, however, the execution time needed to determine the median value may vary with the values of the elements in a particular input sequence. This kind of nondeterminism may be problematic in real-time applications intended for microcontrollers having limited computing power. In addition, the sorting of the whole input sequence generates an substantial overhead. In order to eliminate the overhead, a more efficient in-place algorithm known as Quick select can be applied [14].

An alternative way of calculating the median value is to use a *median network*. The median network is a kind of sorting network whose concept is deeply elaborated in [14]. A sorting network is defined as a sequence of elementary compare-swap operations that sorts all input sequences. The sequence of comparators is fixed and depends only on the number of elements to be sorted, not on the values of the elements. Similarly, a median network is a sequence of elementary compare-swap operations that calculates the median for all input sequences. A compare-swap operation of two elements (a, b) compares a and b and exchanges (if it is necessary) the elements in order to obtain a sorted sequence.

3.2 Construction of median networks

A sorting network with n inputs and n outputs can be constructed using an instance of the sorting algorithm which is operating over a sequence of n items. The only condition is that the algorithm must be data independent. Bitonic-sorting and Batcher's odd-even merge sorting are examples of such algorithms.

A median network can be constructed from a sorting network by removing the useless compare-swap operations (i.e. operations that do not contribute to the output value). Aside from this, an optimal sequence of compare-swap operations is known for some median networks [4, 31]. Generally, the direct design of the median and sorting networks is a nontrivial task, especially for larger values of N.

In order to illustrate the difference among the results produced by various algorithms, let us suppose that we need to construct a 9-median network (i.e. a median network for $n = 9$ inputs). When Bitonic-sorting algoritm is used, we obtain a sequence of 23 operations. The Batcher's odd-even merge sorting produces the median network consisting of 22 operations (see Fig. 1a). The optimal 9-median

```
dtype median9_22(dtype *din)
{
  CS(0,1); CS(3,4); CS(5,6);
  CS(7,8); CS(0,2); CS(5,7);
  CS(6,8); CS(0,3); CS(1,2);
  CS(6,7); CS(0,5); CS(1,4);
  CS(2,3); CS(1,2); CS(3,4);
  CS(1,6); CS(2,7); CS(3,8);
  CS(4,5); CS(2,4); CS(3,6);
  CS(3,4)
  return din[4];
}
```

(a)

```
dtype median9_19(dtype *din)
{
  CS(1,2); CS(4,5); CS(7,8);
  CS(0,1); CS(3,4); CS(6,7);
  CS(0,3); CS(1,2); CS(4,5);
  CS(7,8); CS(3,6); CS(4,7);
  CS(5,8); CS(1,4); CS(2,5);
  CS(4,7); CS(4,2); CS(6,4);
  CS(4,2)

  return din[4];
}
```

(b)

Fig. 1 Two instances of a median network for 9 inputs. The compare-swap operation is implemented using macro CS(a, b) which assigns the lower value to din[a] and higher value to din[b]. One of the possible implementations of CS is the following one:

$$\textbf{if}\ (\mathrm{z}\quad \mathrm{din}[\mathrm{b}] - \mathrm{din}[\mathrm{a}]) < 0\ \textbf{then}\ \mathrm{din}[\mathrm{a}] \leftarrow \mathrm{din}[\mathrm{a}] + \mathrm{z};$$
$$\mathrm{din}[\mathrm{b}] \leftarrow \mathrm{din}[\mathrm{b}] - \mathrm{z};\ \textbf{end}$$

a Median constructed using Batcher's odd-even merge sorting. **b** Optimal implementation of 9-input median [4] network consists, however, of 19 operations (see Fig. 1b). The corresponding codes in C language are shown in Fig. 1. Note that various implementations can be utilized. If there is a requirement to preserve the input data samples (i.e. to avoid the usage of in-place computations), two temporary variables have to be associated with each output of the CS operation.

Alternativelly, each compare-swap operation can be replaced by two basic operations—minimum and maximum. This allows us to furthermore decrease the total number of required instructions because not every output value calculated by the compare-swap element is subsequently utilized. For example, the last operation shown in Fig. 1b calculates din[2] and din[4]. It is evident that it makes no sense to determine the value of din[2] because only din[4] is returned at the end. The representation based on the minimum/maximum operations enables us to reduce the code size of the 9-median shown in Fig. 1b by 21 % provided that the minimum and maximum macros share the code required to determine the relation between both input values.

3.3 Power-aware improvement of median networks

It is clear that the performance as well as power consumption of a particular median network implementation directly depends on the number of operations a given median network consists of. The higher number of operations results in a higher power consumption as well as a longer execution time. This relation can easily be revealed, for example, by inspecting the implementations shown in Fig. 1. The median networks shown consist of a fixed number of operations. Each operation is executed in the same number of clock cycles on average if it is measured at the level of machine code instructions.

Decreasing the number of operations represents the only way to improve the performance and power consumption. Unfortunately, a reduction of the number of operations is not possible without accepting some errors in the outputs produced by a median function. In other words, we have to search for a sequence of compare-swap operations that are capable of approximating the median. Let us call such a sequence a comparator network.

Two possible approaches can be applied to achieve our goal. One way to obtain an improved median network with a reduced number of operations is to construct a comparator network completely from scratch. It means to employ a variant of GP (e.g. linear GP, cartesian GP, etc.) and evolve programs satisfying the required quality as well as target size constraints (i.e. consisting of the required number of operations). The other possibility is to apply evolutionary techniques to reduce the number of operations of already existing median networks provided that the quality is maximized. In this scenario, a fully working median network used as a starting point is gradually modified according to the genetic improvement methodology. At the end of this process, a comparator network of the highest possible quality consisting of the required number of operations is expected. The question is which of these approaches performs better.

It seems to be natural to employ the first approach, however, due to the limited scalability of the evolutionary design, this approach seems to be extremely inefficient. As shown in [34], randomly seeded GP discovered fully functional solutions for the 9-median, however, no correct solution was discovered for the 25-median. While the evolutionary design of a 9-median is a relatively simple problem, a 25-median consisting of more than 200 operations seems to be outside of the range of possibilities of the evolutionary design approach. If a median consisting of more than 100 operations is required, then direct evolution is unable to accomplish the goal. The authors claimed that solving the larger instances from scratch seems to be impossible for any evolutionary algorithm based on direct encoding.

3.4 Problem formulation

Given an existing median network N, i.e. a sequence of compare-swap operations of length n, and the target number of compare-swap operations m, find an alternative sequence of compare-swap operations M of length m s that this sequence maximizes the functional objective (quality) and minimizes the non-functional objectives.

In general, the problem can be understood as a single-objective as well as a multiple-objective optimization problem. The number of operations and time of execution and power consumption represent the typical software-related non-functional objectives. In addition, the number of stages required to determine the output value can be considered. This parameter is, however, important only when the comparator networks are intended for hardware implementation.

4 The quality of the improved median networks

In this section, we give an overview of approaches that enable us to assess the quality of partially working software and hardware. Then we discuss how to determine the quality of the improved median networks. We introduce and prove the permutation principle which gives a clue on how to determine the quality of median functions efficiently. In order to measure the distance between an original and improved version of the median, a problem-specific quality metric is proposed.

4.1 Common quality metrics

Various approaches to evaluate the quality of partially working software and hardware have been proposed in the literature.

The *error probability* (error rate) and *Hamming distance* represent metrics typically used to measure the quality of digital circuits. The error rate (Hamming distance) is defined as the percentage of input vectors (bits of output) for which the approximate output differs from the original one. In general, 2^{wn} input combinations exist for an n-input median network operating with elements encoded using w-bit integers. Clearly, it is intractable to evaluate all possible input combinations, however, the number of input combinations can substantially be reduced by applying the zero–one principle. The zero–one principle states that if a sorting network with n inputs sorts all 2^n input sequences of 0’s and 1’s into a nondecreasing order, it will sort any arbitrary sequence of n elements into a nondecreasing order [14]. As a consequence, 2^n input combinations are sufficient to determine the error rate. Unfortunately, it seems to be difficult to apply this metric in practice. For example, there can exist a candidate implementation slightly modifying one half of the output values, but still providing good performance if used, for example, in image filtering.

The *average error magnitude* is another metric which is used for determining the quality of arithmetic circuits, not only in the field of evolutionary design, but also in the approximate computing. The average error magnitude is defined as the sum of absolute differences in magnitude between the original and approximate circuits, averaged over all inputs. Two complex issues are, however, connected with this parameter when used to evaluate the quality of a median network. Firstly, it is not possible to apply the zero–one principle in this case. As a consequence, we are unable to determine the exact value of this metric. In practice, we have to use a subset of all possible input combinations which helps us to estimate the value of the average error magnitude. The selection of the input vectors must be done carefully because it influences the precision of the estimate. Secondly, the averaging may hide situations in which completely wrong results are returned.

The common problem of the previously discussed generic metrics is that they do not reflect the quality of selecting the median value. In order to investigate the impact of the approximations on the quality of obtained results, regardless of the values of the input items, we introduce a new problem-specific metric. Let us recall that the median of a finite list of numbers can be found by arranging all the numbers from the lowest value to the highest value and picking the middle one. In other words, the median of a finite list of numbers consisting of $2k+1$ items is equal to the $(k+1)$th lowest value. The most important property of the median functions implemented in accordance with Sect. 3.2 is that the output always equals one of the input values. Let the output value equal to the jth lowest value. To describe the quality of an approximate median function, we can introduce *distance error* defined as the distance of the item chosen as the output value (i.e. jth lowest value) from the median (i.e. $(k+1)$th lowest value) calculated as $|j-k+1|$. Two additional metrics can be inferred from the distance error: *average distance error* defined as the sum of error distances averaged over all input combinations producing an invalid output value and *worst case distance error* defined as the maximal distance error calculated over all input combinations.

Note that it is not necessary to investigate all possible input combinations in practice. The *permutation principle* introduced in Sect. 4.2 permits one to substantially reduce the total number of input combinations that has to be investigated for a given comparator network in order to precisely determine the properties of the network. According to the permutation principle, the aforementioned distance errors can be determined using the permutations of a set S consisting of $2k+1$ different values. To determine the quality, we propose to use a set $S = \{-k, -k+1, \ldots, 0, \ldots, k-1, k\}$. The set S consists of $2k+1$ successive integers starting at the value $-k$. This particular arrangements enable to calculate the average distance error in the same way as the average error magnitude. This is possible because the median of S is equal to zero and the distance between jth lowest item (i.e. the value $j-(k+1)$) and $(k+1)$th lowest item (i.e. median of S)

is equal to $j-(k+1)$. Compared to the process of determining the average error magnitude, however, a substantially lower number of input combinations is required to be processed by a candidate median implementation.

4.2 The permutation principle

Definition 1 Let Σ be an ordered alphabet. A *comparator network* is a directed acyclic graph with n inputs and n outputs ($n \geq 2$), where each node has two inputs (x_1, x_2) and two outputs (y_1, y_2). The function of a node is defined as $y_1 = min(x_1, x_2) \wedge y_2 = max(x_1, x_2)$, where $x_1, x_2 \in \Sigma$.

Definition 2 A *sorting network* is a comparator network that monotically sorts every input sequence.

Definition 3 Let $A = (a_1, \ldots, a_n)$ be a sequence of n different elements, $A \in \Sigma^*$. Let δ_A: $\Sigma^* \rightarrow \mathbb{N}$ be a mapping which assigns each element $a_i \in A$ the position of this element in the sorted variant of A. Let δ_A be defined as follows:

$$\begin{aligned}
&\delta_A(x) = 0 \Leftrightarrow \quad \forall a \in A{:}x < a \\
&\delta_A(x) = |A| - 1 \Leftrightarrow \quad \forall a \in A{:}x > a \\
&\forall\ 1 \leq i, j \leq n{:}a_i < a_j \Leftrightarrow \delta_A(a) < \delta_A(b)
\end{aligned}$$

For simplicity, let δ(A) denote the sequence $(\delta_A(a_1), \delta_A(a_2), \ldots, \delta_A(a_n))$.

Lemma 1 ([14]) *Let N be a sorting network with n inputs that transforms a sequence* $A = (a_1, a_2, a_3, \ldots, a_n)$ *to a sequence* $B = (b_1, b_2, b_3, \ldots, b_n)$. *If a monotonic mapping f is applied to the sequence A, the network N transforms a sequence* $A' = (f(a_1), f(a_2),\ f(a_3), \ldots, f(a_n))$ to $B' = (f(b_1), f(b_2),\ f(b_3), \ldots, f(b_n))$.

Theorem 1 *Let N be a comparator network with n inputs. Let S be a set consisting of n distinct values. If every permutation of a set S is sorted by N, then every arbitrary sequence is sorted by N.*

Proof Suppose $A = (a_1, \ldots, a_n)$ is an arbitrary sequence which is not sorted by N. This means $N(A) = B = (b_1, \ldots, b_n)$ is unsorted, i.e. there is a position k such that $b_{k+1} < b_k$. Clearly, mapping δ_A is monotonic. By applying Lemma 1 and δ_A, the following holds $\delta_A(b_{k+1}) < \delta_A(b_k)$, i.e. $\delta(B) = \delta(N(A))$ is unsorted. This means that $N(\delta(A))$ is unsorted or, in other words, that the sequence $\delta(A)$ is not sorted by the comparator network N.

We have shown that, if there is an arbitrary sequence A that is not sorted by N, then there is a sequence $\delta(A)$, i.e. a sequence of $(0, \ldots, n-1)$ values, that is not sorted by N. Equivalently, if there is no $(0, \ldots, n-1)$-sequence that is not sorted by N, then there can be no sequence A whatsoever that is not sorted by N. Equivalently again, if all $(0, \ldots, n-1)$-sequences are sorted by N, then all arbitrary sequences are sorted by N.

Clearly, there exists a bijection between all permutations of S and all $(0, \ldots, n-1)$-sequences as follows from the definition of S. In particular, δ_S

ensures the bijective mapping. This means that if all permutations of S are sorted by N, then all arbitrary sequences are sorted by N. $\square$

4.3 The permutation principle and distance error

The permutation principle introduced in the previous section can be employed to determine the distance between an arbitrary comparator network (e.g. partially working sorting network) and a sorting network as follows.

Theorem 2 *Let C be a comparator network, and N be a sorting network, both with n inputs. Let A be an arbitrary sequence $A = (a_1, \ldots, a_n)$. Let $D = C(\delta(A)) - N(\delta(A)) = (d_1, \ldots, d_n)$ be a mapping which assigns each element a_i a number d_i. Then, d_i is error expressed as the number of positions that are required to shift a_i to the right in sequence $C(\delta(A))$ to obtain sorted variant of sequence A.*

Proof Let $B = N(A)$ and $B' = C(A)$. For each element $b'_k \in B'$ holds that difference between a correct position and position of b'_k in a sorted variant of sequence A is equal to $d_k = \delta_A(b'_k) - k$. As N is a sorting network, it holds that $\delta_A(b_k) = k$ which implies that d_k can be expressed as $d_k = \delta_A(b'_k) - k = \delta_A(b'_k) - \delta_A(b_k)$. By applying Lemma 1, it holds that $D = C(\delta(A)) - N(\delta(A))$. $\square$

Definition 4 Let A and B be two sequences of elements. Let $\sim_\delta$ denote an equivalence relation on the set of all sequences defined as follows:

$$A \sim_\delta B \Leftrightarrow |A| = |B| \wedge \delta(A) = \delta(B)$$

To conclude this part, let us give a simple example which illustrates the principle of determining the position error for a comparator network with 4 inputs and 4 outputs and two chosen sequences A and B.

Example 1 Let A and B be two sequences consisting of 4 items defined as $A = (25, 14, 36, 8)$, $B = (16, 12, 20, 2)$. Let N denotes the sorting network and C be a comparator network both with 4 inputs and 4 outputs, where C is defined as follows: $C(a_1, a_2, a_3, a_4) = (min(a_1, a_2), max(a_1, a_2), a_3, a_4)$.

According to the Definition 3, $\delta(A) = (2, 1, 3, 0)$ and $N(\delta_A(A)) = (0, 1, 2, 3)$ which follows from $N(A) = (8, 14, 25, 36)$ where $N(A)$ denotes the sorted sequence A. As the output of C is equal to $C(A) = (14, 25, 36, 8)$, the $C(\delta(A)) = \delta(C(A)) = (1, 2, 3, 0)$. To calculate the positional differences $D_C(A)$, we apply Theorem 2 which yields the following result $D_C(A) = C(\delta(A)) - N(\delta(A)) = (1, 2, 3, 0) - (0, 1, 2, 3) = (1, 1, 1, -3)$. The result can be interpreted in such a way that each of the first three elements of the partially sorted sequence $C(A)$ should be shifted one position to the right and the last element should be shifted three positions to the left. If all the shifts are applied, we obtain a sorted sequence.

The same sequence of steps applied to B yields $D_{C(B)} = C(\delta(B)) - N(\delta(B)) = (1, 2, 3, 0) - (0, 1, 2, 3) = (1, 1, 1, -3)$. It reveals that $D_{C(B)} = D_{C(A)}$, i.e. the same sequence as for A was obtained. It means that we have applied the sequence within the same equivalence class, i.e. $A \sim_{\delta} B$. This fact can be easily checked by comparing the output of $\delta(A)$ and $\delta(B)$. It holds that $\delta(B) = \delta(A)$.

4.4 Final remarks

In general, there exist 2^{wn} input combinations that can be processed by an n-input comparator network operating at w-bits. We have shown that it is sufficient to reduce the number of the possible input combinations to $n!$ to prove the validity of a sorting network due to the existence of permutation principle (see Theorem 1). According to the zero–one principle, the validity of a sorting network can, however, be checked using 2^n binary vectors. As it can easily be checked, the 2^n is for $n \geq 4$ lower than $n!$, hence it seems that the proposed permutation principle does not offer an advantage. However, the problem of zero–one principle is that the binary vectors cannot probably be used to evaluate the quality of a comparator network. The reason is that we are not able to distinguish which value comes from what input (there are only two values—0's and 1's). To address this problem, Theorem 2 helps to determine the so called position error (distance error) which can be used as a basis of an error metric.

The impact of the introduced permutation principle can be seen from theoretical as well as practical point of view. From the theoretical point of view, it was proven that we can use this principle to evaluate the quality of candidate solutions without loss of generality (i.e. it is not necessary to evaluate responses for all w-bit input combinations). The permutation principle significantly reduces the number input vectors that have to be applied to obtain the fitness. In particular, 362, 880 vectors instead of 256^9 vectors are sufficient to precisely determine quality of a 9-input comparator network operating at 8-bits. From the practical point of view, the permutation principle (if properly applied) extremely simplifies the evaluation of candidate solutions because the response of a comparison network (i.e. output value) is equal to the distance from the median value. It means that we can avoid precomputing and storing of a training dataset.

Example 1 illustrates that no additional information about an investigated network is obtained when we try to check some property of a comparator network using sequences belonging to the same equivalence classes. This may happen when randomly generated test cases are used to determine this property. In fact, the randomly generated input sequenced may introduce a bias when used to evaluate quality of a comparison network. The probability of occurrence such cases is relatively high, because only the relation among the values within a generated sequence is important (i.e. not values themselves). Let us give an example. We created 10^6 test vectors consisting of nine randomly generated 8-bit values. Then, we calculated the number of covered equivalence classes according to Definition 4. It revealed that only 337,751 out of 362,880 (i.e. 93 %) of all possible equivalence classes were covered despite the fact that we generated approximately three times more test vectors than the number of equivalence classes. It means that there is many test vectors belonging to the same equivalence class.

The permutation principle and the obtained conclusions can directly be applied not only to comparator networks discussed in this section but also to comparator networks with a single output. In other words, the permutation principle can be used to assess the quality of partially working median as well as sorting networks.

5 The proposed method

In order to search for solutions with some improved level of a non-functional property, we must be able to quantify that property. In our case, the execution time and power consumption are considered. To estimate these non-functional properties, we can use the number of operations of which the median function consists of. This is a fairly reliable high-level estimate not only of execution time but also of power consumption. A more detailed simulation employing an accurate simulator would be necessary in general, however, our programs are designed as a sequence of min and max operations. It is supposed that each operation is transformed by compiler to a sequence of instructions that requires exactly the same number of clock cycles to perform this operation. In addition to that it also reflects the nature of modern embedded systems (e.g. ARM) whose instruction set predominantly consists of instructions that can be executed within one clock cycle.

In this paper, the task is formulated as a single objective optimization problem where the number of operations $\widetilde{n}$ represents a constraint specified by designer. Because both considered non-fuctional objectives linearly depend on this constraint, it is not necessary to include these objectives in the fitness function. This represents the main advantage of the constraint-oriented approach. In addition to this, the constrain-oriented approach is relevant to practice where the designers usually wants to achieve a particular power reduction in order to improve the performance of the whole embedded system.

To achieve our goal, we propose to use cartesian GP (CGP) in its linear form [18]. The linear form seems to be preferred approach compared to the traditional form of CGP representing the solved problems using two-dimensional array of nodes.

5.1 Representation of comparator networks

Each comparator network with n inputs can be represented using a directed acyclic graph consisting of k nodes. In order to encode such a graph, we can map the nodes to a 1D array of N nodes ($N \geq k$) that can be encoded using cartesian GP representation as follows. The number of rows, which is one of CGP parameters, is set to $n_r = 1$. The number of columns n_c is equal to the number of nodes, i.e. $n_c = N$.

The 1D array of nodes can be encoded using a string of integers, the so-called chromosome. The inputs are labelled $(0, 1, \ldots, n-1)$ and the nodes are labelled $(n, n+1, \ldots, n+n_c-1)$. Each node has two inputs and is encoded in the chromosome using three integers—two labels specifying where the node inputs are connected to and a single label specifying the function of the node. Finally, the chromosome contains a single integer specifying the label of a node where the output is connected. The chromosome consists of $3n_c + 1$ integers (i.e. genes).

The main feature of CGP is that n_c as well as n_r (i.e. the total number of nodes N) are constant during evolution. It means that the size of the chromosome is constant because it depends only on n_c. On the other hand, the size of graph represented by this chromosome is variable as some nodes may become inactive. The nodes which do not contribute to the output value are called the inactive nodes. Example is given in Fig. 2 where only 4 out of 5 nodes are active.

Each node can act as minimum or maximum function. The inputs of a node can be connected either to the output of a node placed in previous l columns or to one of the input variables. This restriction ensures that no feedbacks are allowed. The output can be connected to output of any node. The l-back parameter will be unrestricted, i.e. $l = n_c$.

5.2 Quality of candidate solutions

The computational effort of EA directly depends on the number of test cases that are used for fitness evaluation of the GP individuals. Even if we apply the permutation principle which substantially reduces the number of all possible test cases that have to be investigated, we cannot run all of these due to time constraints. Thus, the number of test cases is fixed and specified at the beginning of the evolution. Based on the preliminary experiments and in accordance with observations related to the minimum number of test vectors required to evaluate candidate solutions [34], we determine the number of test cases as $T = 10^3 \times n^2$, where n is the number of input variables. The number of test cases is chosen in such a way that we are able to relatively precisely estimate the quality of the individuals while the time of fitness evaluation remains reasonable. Surprisingly, this simplification does not have any significant effect in practice provided that a reasonable number of unique permutations is used. The example given in Fig. 3 demonstrates how the number of test cases influences the shape of distribution of distance error for an approximate version of 25-input median. If more than 10^4 test vectors are used, the distributions are almost identical.

The first generated test case is the sequence $S = (-k, -k+1, \ldots, 0, \ldots, k-1, k)$ where $n = 2k + 1$. The next test case is obtained by swapping two randomly chosen items. This step ensures that a permutation of S is obtained. Note that the process of

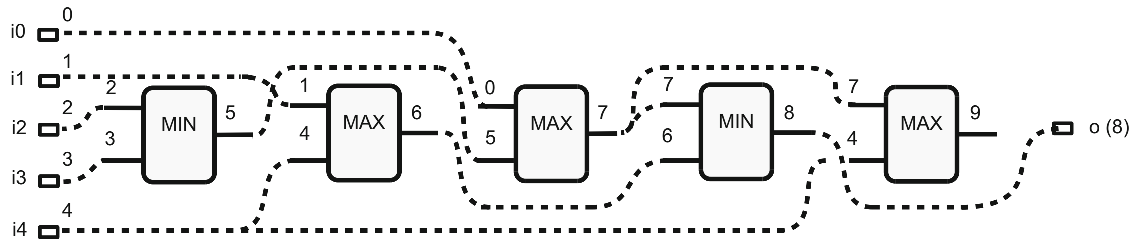


Fig. 2 Example of a 5-input comparator network encoded using cartesian GP with parameters: $k = 5$, $n_c = 5$, $l = 4$. Chromosome: 2, 3, min; 1, 4, max; 0, 5, max; 7, 6, min; 7, 4, max; 8. Node 9 is not used. The behaviour of the encoded comparator network is defined as: $o = \min(\max(i_0, \min(i_2, i_3)), \max(i_1, i_4))$

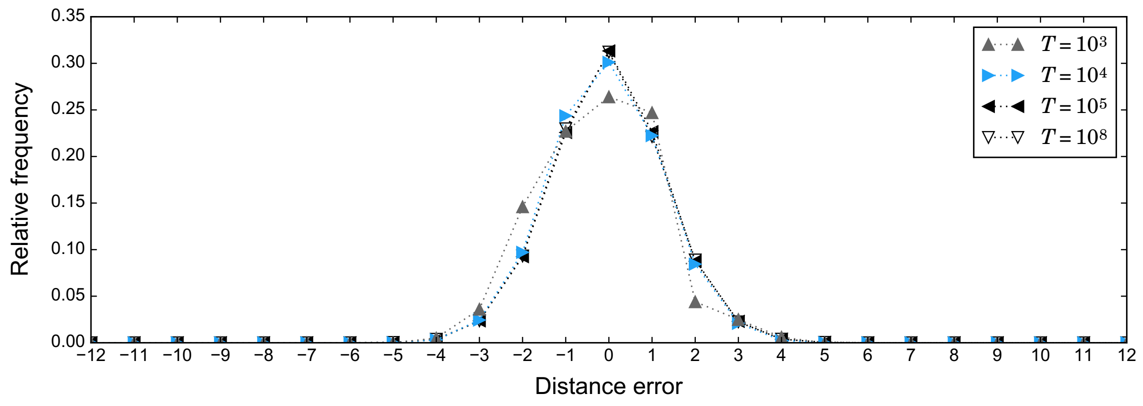


Fig. 3 Distribution of distance error for an approximate version of 25-input median as a function of the number of test cases T

generating the test cases have to be deterministic because we have to guarantee that exactly the same fitness score is obtained for individuals that represent the same behaviour. In order to satisfy this requirement, a separate random generator is used to perform the random exchanges. This generator is reinitialized with the same seed whenever we begin to generate the permutations.

The quality of the GP individuals is determined as follows. For each test case, the chromosome is interpreted. This step requires to successively determine the value at the output of each node. Finally, the response of a comparator network encoded by the individual is calculated. Because the permutations of a set proposed in Sect. 4.1 are applied to the inputs, the obtained response equals to the distance error determined for a given test case. In order to prefer the implementations with the lowest worst case distance error, we propose to calculate a histogram of distance errors and summarize the obtained results as follows:

$$q(C) = h(C,0) - \sum_{i=-k}^{k} h(C,i)i^2, \tag{1}$$

where $q(C)$ denotes the quality of a comparator network C and $h(C, i)$ represents the number of occurrences of a case for which the distance error equals to i, formally:

$$h(C,i) = \sum_{t \in T} \begin{cases} 1, & \text{ifC(t) = i.} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

where $C(t)$ denotes the response of the comparator network C obtained for a test case $t \in T$ and T is a set of considered permutations of S. There are two reasons for including i^2. Firstly, only positive numbers are summed. Secondly, a natural weight is provided in order to emphasize the most important part of the histogram.

5.3 Search method

The search method follows the standard CGP approach [18], i.e. the evolutionary strategy $1 + \lambda$ is applied. The initial population is seeded by an existing median network. In order to generate a new population, λ offspring individuals are created by a point mutation operator modifying h genes of the parent individual. The best individual of the current population (i.e. the parent individual together with λ offspring) serves as the parent of new population. The process is repeated until a given number of generations is not exhausted.

One mutation can alter either the function of a node, node input connection, or output connection. If a mutation hits a non-active node, this is detected and the candidate solution is not evaluated in terms of functionality because it has the same fitness as its parent. Mutations that do not affect the fitness score are called neutral and seem to be important in CGP because a series of neutral mutations can accumulate useful structures in the part of the chromosome which is not currently active (see detailed analysis in [8, 19]). In order to support this kind of neutrality, neutral mutants always replace their parent in CGP. One adaptive mutation can then connect these structures with active nodes which could lead to discovering new useful implementations.

In order to obtain an approximate version of median function M, we propose to apply a two-stage procedure. At the beginning, the designer specifies the target reduction that should be achieved, e.g. 15 %. The specified value is internally understood as the number of operations L of the approximate median function. In our example, L is equal to 85 % of the number of operations in the original median function.

The first stage starts with a fully functional solution. As has been discussed in Sect. 3.2, the initial solution can always be obtained in practice. In this stage, the goal is to gradually modify the initial sequence consisting of minimum and maximum operations and produce a reduced sequence of length L providing that a 5 % difference is tolerated with respect to L (tolerating a small deviance in the number of operations is acceptable; otherwise the search could easily stuck in a local extreme). The fitness function fit_1 used in the first stage is thus solely based on the number of operations

$$fit_1(C) = |C|, \tag{3}$$

where C denotes a candidate solution implemented using $|C|$ operations.

In the second stage, which begins after obtaining an implementation consisting of the target number of operations, the fitness function reflects not only the size, but also the quality:

$$fit_2(C) = \begin{cases} q(C), & \text{if } 0.95\text{L} \le |\text{C}| \le 1.05\text{L}. \\ -\infty, & \text{otherwise.} \end{cases} \tag{4}$$

It is requested that the number of operations remains within 5 % tolerance with respect to L. Candidate circuits violating this hard constraint are discarded.

The proposed two-stage method eliminates the problem with seeding of initial population which may be considered as a limitation of the resource-oriented method [20]. The advantage of our method is that we do not need to implement a heuristics for generating the initial solution consisting of L operations. Instead, a fully working median function obtained by pruning a sorting-network is used as the start point. In addition to this, it was demonstrated that the randomly seeded CGP was unable to produce reasonable solutions when the complexity of the problem to be solved increases. The role of seeding was investigated for example in [34]. The benefits of the two-stage method are not only in improving the quality of evolved circuits, but also in reducing the time of evolution.

6 Experimental setup

In order to evaluate the performance of the proposed approach, i.e. the ability to improve the considered non-functional parameters of the existing median functions, namely time of execution and power consumption, we have chosen four instances of the median filter that are common in practice. The results of optimization for 9-median, 11-median, 13-median, and 25-median will be reported. While the 9-input and 25-input medians are typically employed in image processing, the 9-input, 11-input and 13-input medians represent instances used to filter data coming from sensors. We did not consider the lower number of inputs because there is nearly no potential for improvement due to the small code complexity.

As previously mentioned, the designer has to specify the target reduction that ought to be achieved by reducing the number of instructions. Eight to eleven design points (i.e. different values of L) were considered for each problem. We carried out 100 repetitions of CGP at each design point to evaluate the variation in the output caused by the random seed. In total, 4000 experimental runs were performed. To be able to evaluate all runs in a reasonable time, the number of generations was limited to $g_{max} = 1 \times 10^4$. The number of generations is based on the initial experiments and represents a compromise between the ability to demonstrate the advantage of the genetic improvement in the solving of the chosen problem and the amount of required computational resources. If the objective is to find the best possible implementation for a certain design point, we recommend to increase the number of generations. In order to improve the efficiency of the fitness function, approach proposed in [35] was employed.

The following settings was used for the search strategy: Twenty offspring individuals are generated from the parent (i.e. $\lambda = 20$) using the mutation operator that modifies up to 5 % of the chromosome genes. The number of columns n_c is fixed for each design point and is initialized according to the number of operations of the original median function. In the case of 9-input and 25-input median, the optimal implementations consisting of the minimal number of compare-swap elements were used from [4]. Each compare-swap element was replaced by minimum and maximum operation and the worthless operations were removed as mentioned in Sect. 3.2. The obtained sequence of minimum and maximum operations was used as a starting point for seeding the evolutionary algorithm. In remaining cases, the initial fully working median networks were derived from a 25-input median network by reducing the number of inputs and removing redundant operations. We have verified that this approach produces more compact median networks compared to the results obtained using the approach employing a sorting algorithm. The parameters of the initial networks are summarized in Table 1.

Table 1 Parameters of the fully working median functions used to seed the evolution and the range in which the design points are sampled

Parameter	9-Median	11-Median	13-Median	25-Median
Number of compare-swaps elements	19	33	43	99
Number of min/max operations	30	56	74	174
Number of min operations	15 (50 %)	28 (50 %)	37 (50 %)	87 (50 %)
Number of max operations	15 (50 %)	28 (50 %)	37 (50 %)	87 (50 %)
Minimum value of *L*	8	8	10	50
Maximum value of *L*	30	56	74	174
Number of design points	11	8	8	13

7 Results

The results of the evolution are summarized in Fig. 4. For each problem and each design point, the normalized fitness score is given. This score is calculated according to Eq. 4, however, the results are normalized by the total number of test

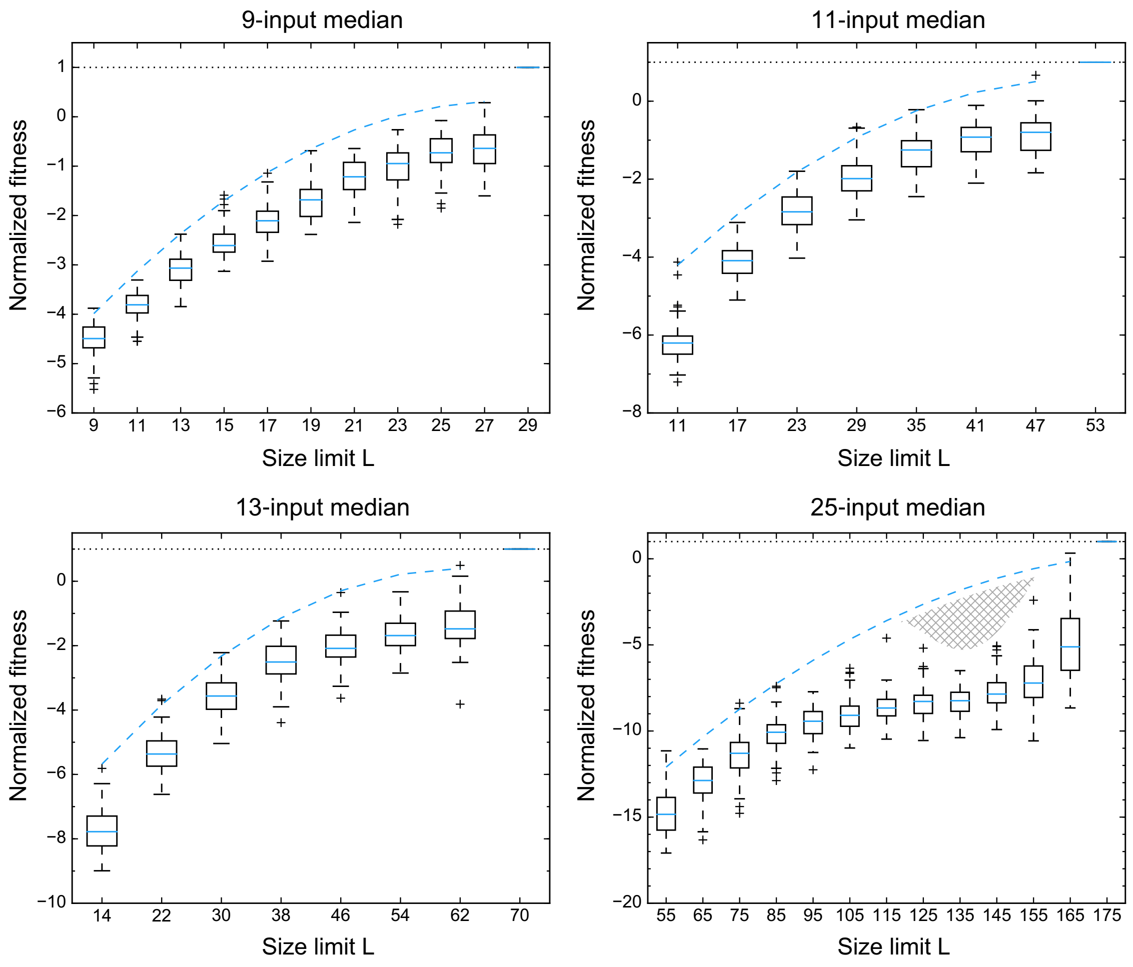


Fig. 4 The fitness score of the evolved comparator networks (approximate median function) based on 100 experimental runs performed for each design point. The *dash line* represents target Pareto frontier

cases. The interpretation of the y-axis is as follows. While the fully working median functions represented by the fittest solutions have their fitness score equal to one, the solutions of lower quality have assigned the fitness score lower than one.

For each problem, we sampled the design space equidistantly to be able to construct the Pareto frontier which helps us to discuss the performance of the method. As mentioned earlier, the maximum value of L is bounded by the size of the initial solution. Conversely, it makes no sense to explore situations where L is lower than the number of input variables because it means that some inputs will not be involved in computation. In the case of 25-input median, we restricted the lower bound even more because it would be computationally expensive to perform evolution for all cases. According to the measurements, 8.8 ms are required in average to calculate fit_2 for 9-input median. This time, however, increases up to 368.3 ms in the case of 25-input median. The experiments were conducted on a 64-bit Linux machine running on Intel Xeon X5670 CPU (2.93 GHz, 12 MB cache) equipped with 32 GB RAM.

Interestingly, compared to the resource-oriented method [34], our method is extremely efficient if the time required to obtain an implementation consisting of L operations is considered. According to the experiment, the average duration of the first stage is less than 10 ms in the case of 9-median and less than 373 ms in the case of 25-median.

The obtained results given in Fig. 4 are presented using boxplots which illustrate distribution of the normalized fitness calculated independently for each considered design point. As it can be seen, the variance of the fitness score is quite low for each design point. Taking into account that the number of generations was relatively low, these results demonstrate the robustness and stability of our method. The only exception is the 25-input median where we can see higher variance primarily at both extremes of L. In order to analyse this situation more thoroughly, we created a target pareto frontier (see dashed lines in Fig. 4) representing the goal of evolution. This Pareto frontier was obtained by interpolation of the fittest implementations obtained for 9-input, 11-input and 13-input median. The obtained regression models were generalized and projected backward to the plots. In most cases, we were able to find solutions that are very close to this imaginary pareto frontier. Unfortunately, in the case of 25-median (see Fig. 4, bottom right), we can see that there are cases in which the fitness score of the obtained results is far from the expected one. This is evident especially for cases where L is between 125 and 155. To investigate the reason of this gap, we tried to prolong the time of evolution for few of these cases and we discovered that this problem is caused by the insufficient number of generations. In order to obtain better results, it would be necessary to increase n_g adequately (at least by two orders of magnitude).

Whilst the initial implementations of 9-input and 25-input median networks remained unchanged, which was in fact expected as it is believed that the corresponding sequences of compare-swap elements are optimal, the evolution discovered improved versions of 11-input fully functional median function consisting of 50 operations and improved version of 13-input fully functional median counting 66 operations which yields 11 % reduction in both cases.

A more detailed analysis of the quality of discovered solutions is shown in Fig. 5 where we present histograms of the error distribution for each problem. The histograms are created using the best solutions obtained from all experimental runs. It means that for each design point, the fittest solution was identified and chosen. The quality is expressed in terms of the distance error. The histogram of distance errors are calculated for each discovered solution using 1000 times more permutations compared to the number of permutations utilized to determine fit_2; this enables to obtain precise results exhibiting the error in the order of 10^{-3}.

Let us discuss, for example, the results for 11-input median (see Fig. 5, top right). If we reduce the number of operations by 12 % (i.e. to 44 operations), the output value is determined correctly in more than 93 % all possible cases. In the rest of the cases (i.e. 6 %), the output value is determined incorrectly as the 4th lowest item of a sorted list of numbers. In less than 0.9 % of cases, the 6th lowest item is returned. Because the median value corresponds with 5th lowest item, the distance between median and output value is equal to 1 in both cases. If the number of operations is reduced to 20 (60 %), the worst case error increases to 2. According to the distribution of errors, this error, which is caused by outputting 3th or 7th lowest item, occurs in 3.6 % of all input cases only. The remaining 47.2 % erroneous outputs are caused by selecting 4th or 6th lowest item.

An interesting feature of the discovered solutions is the asymmetric distribution of the errors. This is more evident if we look at the histogram for 25-input

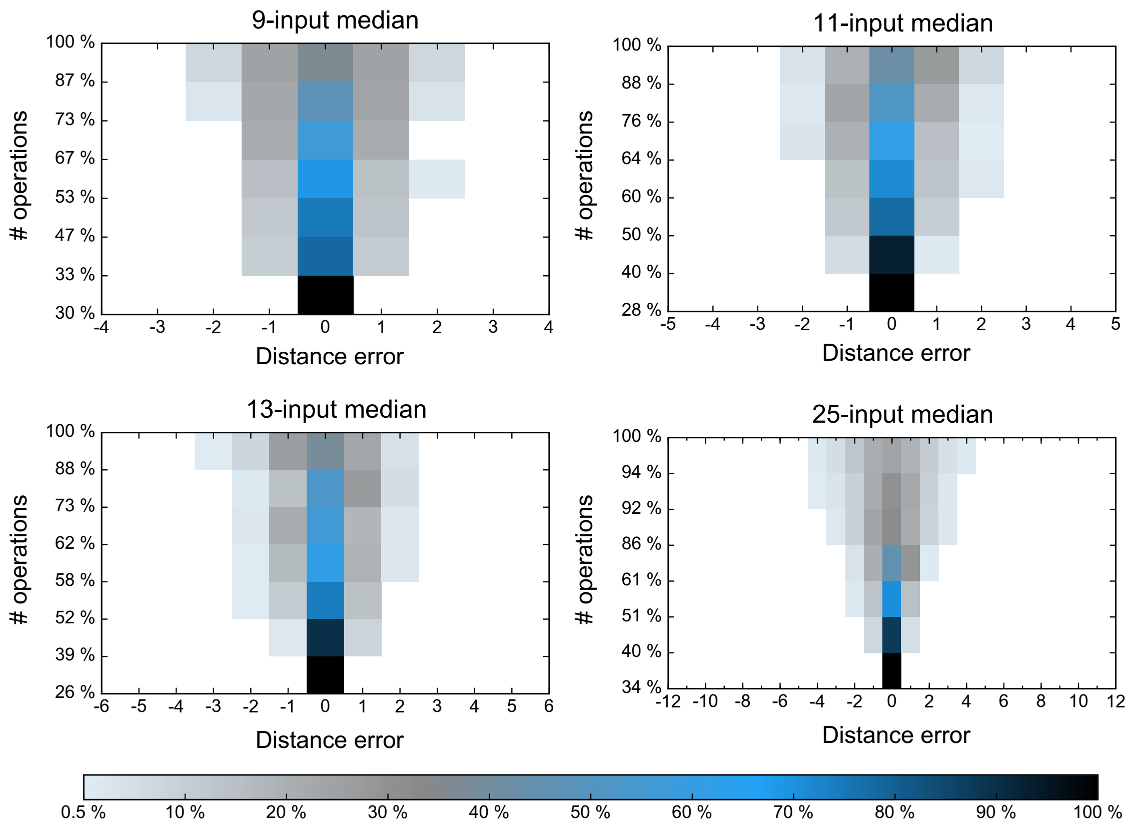


Fig. 5 The quality of the best discovered solutions consisting of a different number of operations expressed in terms of the distance error. The zero error means that the 5th, 6th, 7th, and 13th lowest item of input sequence was returned for 9-input, 11-input, 13-input, and 25-input median respectively

Table 2 Parameters of the improved implementations of 9-input median

No. of operations	Achieved improvement (%)	Error probability(%)	Distance error		
			Mean	Left/right	Worst-case
9	70.00	68.11	0.871 ± 0.702	−2	2
10	66.67	63.36	0.776 ± 0.677	−2	2
14	53.33	53.12	0.591 ± 0.601	−2	2
16	46.67	42.85	0.428 ± 0.495	−1	1
20	33.33	30.93	0.321 ± 0.491	−1	2
22	26.67	25.26	0.253 ± 0.434	−1	1
26	13.33	21.53	0.215 ± 0.411	−1	1

comparator network consisting of 150, i.e. 86 %, operations (see Fig. 5, bottom right). While the 4th lowest item is returned in 20 % of cases, the 6th lowest item is returned in more than 29 % of cases. We have not investigated the exact reason because it does not represent a real problem, however, it is worth noting that we have obtained many solutions with the same fitness score and it may happen that there is a solution with the same or slightly lower fitness score having a symmetric distribution of errors.

We can conclude that the obtained reduced median networks are of high quality. Even in the extreme case, where approximately 50 % of operations are removed, the error is not worse than one position for 9-input median, 2 positions for 11-input median, and 3 positions for 13-input and 25-input median respectively. The 25-input median consisting of more than 170 operations offers the largest possibilities for improvement. We can remove more than 25 % operations without a significant decrease in the quality. For more than 75 % of all possible input combinations, the median value or the values next to the median are returned.

To have a notion of properties of the discussed error metrics, Table 2 reports the error probability, mean distance error, and left and right worst case distance error for 9-input median. It can be seen that as the number of operations decreases, the error probability as well as the distance error are increasing. The mean value increases, however, it is not easy to a priori specify the required target value. The same is valid even for the error probability which gives the number of invalid output values, i.e. the amount of cases in which a value different from median was returned. Looking at the results shown in Fig. 5, it can easily be revealed that the issue with the mean value is that the distribution of errors is not the Gaussian distribution, especially for cases with a small reduction of the number of operations.

8 Improved medians in real embedded systems

Because the medians are typically employed to solve some real problem, we take the best discovered approximated median filters whose quality was discussed in the previous section and evaluated their performance in two different real-world problems—processing of data acquired by sensor devices, and removing of noise in image data.

For each case study, the problem is briefly introduced first. Then, non-functional parameters of evolved as well as commonly used implementations are analysed and discussed. Finally, the impact of the approximate medians on quality and performance is evaluated providing that the approximate medians are employed as the main component which process data.

Four microcontrollers were chosen to evaluate the non-functional parameters of the evolved median functions. The microcontrollers were programmed using the complied C codes of discovered implementations discussed in the previous sections. Two non-functional parameters were measured: (a) the time that each microcontroller spends in a routine which computes the median value, and (b) energy consumed by the microcontroller to execute this routine.

A specific program was implemented, compiled and executed by the microcontrollers to perform the measurements. The program is designed as follows. Firstly, an input vector consisting of *n* integers is randomly initialized and fed to the routines calculating the median. Note that *n* is equal to the number of inputs of median. Then, an infinite loop is executed, which contains calling of the routine calculating the median value followed by a code modifying a randomly chosen value of the input vector to another value. Passing one iteration of the loop is indicated by inverting the logic value on a given pin. The execution time is then obtained using an oscilloscope by monitoring the period of the signal on the pin. The average execution time is reported.

In order to precisely determine an average energy needed to calculate the median value, all unused peripheral devices are switched off. Only those external components remain used which are necessary for program execution. Energy consumption was measured using Agilent N6705B DC Power Analyzer displaying the error lower than 0.025 % for voltage as well as current measurements.

8.1 Microcontrollers used for testing

In order to evaluate the non-functional parameters, we have chosen the following common-off-the-shelf microcontrollers available in our lab: 8-bit microcontroller of Microchip PIC family with code name PIC16F628A, 16-bit PIC24F08KA102, low-power 16-bit microcontroller MSP430F2617 from Texas Instruments and 32-bit ARM-based microcontroller STM32F100RB produced by STMicroelectronics. The goal is to present results for various architectures because there typically exist variations in the performance caused by different instruction sets on the one side and different internal architecture on the other side. To be able to interpret the obtained results, the main features of the microcontrollers are briefly discussed in this section.

The 8-bit PIC equipped with 3.5 kB of FLASH and 224 B of RAM is optimized for low-cost applications. Hence, a simple accumulator architecture without a stack is used. The instruction set consists of 35 instructions encoded using a 14-bit wide instruction word. The two-stage instruction pipeline allows all instructions to be executed in a single cycle, except for program branches. The chosen chip has an internal oscillator running at 4 MHz and consuming about 10 nA in the sleep mode and about 565 μA in the active mode. Note that these values were measured when all the peripherals were deactivated.

The 16-bit PIC represents a class of microcontrollers with a register architecture consisting of 16 general-purpose 16-bit registers and 7 special registers. The instructions are encoded using a 24-bit instruction word with a variable length of the opcode field. The chosen chip contains 8 kB of FLASH memory, 1.5 kB of RAM memory and employs an internal oscillator running at 8 MHz. The instructions require from 1 to 3 clock cycles and are executed at 4 MHz. Our chip consumes about 4 mA in the active mode and 25 nA in the sleep mode.

The MSP430F2 is a 16-bit ultralow-power RISC microcontroller with register architecture optimized for processing data from sensor devices. The chosen CPU consists of 16 registers, is equipped with 92 kB of FLASH memory and 4kB of RAM, and can operate at 16 MHz. The calibrated digitally controlled internal oscillator can be configured to generate up to 8 MHz signal for system clock. The instruction set consists of 51 instructions with three formats and seven address modes. In contrast with PIC, there are instructions that enable to access two memory operands. The instructions require from 1 to 6 cycles to be executed. The instructions working with registers require a single clock cycle, the instructions addressing memory require 3 or 6 (when two memory accesses are required) cycles. The chip consumes 365 μA in the active mode at 1 MHz and 500 nA in the standby mode. In order to exploit the low-power capabilities, we configure the internal oscillator to operate at 1 MHz.

The STM32F100RB incorporates a high-performance RISC ARM Cortex M3 core offering twelve 32-bit general-purpose registers. This core builds on the ARMv7-M architecture and shows higher computational power compared to the aforementioned chips. For example, a single-cycle multiplication and a hardware division are supported. STM32 is equipped with 128 kB of FLASH memory, 8 kB of RAM and operates at 24 MHz. The maximum current consumption in the sleep mode is approx. 3.8 mA. When the peripherals are enabled, the current increases to 9.6 mA. The current in active mode ranges from 10 to 150 mA depending on the state of peripherals.

8.2 Evolved code on different microcontrollers

The process of obtaining C code from a chromosome is straightforward. Every active node, starting from one with the lowest index, corresponds with a single line of code containing a call of *min* or *max* function whose operands are taken from the input sequence or the outputs of preceding operations.

The *min* and *max* functions are defined as two macros outputting the minimal and maximal value for two operands. The compiler is then able to unroll the code and optimize it in terms of register assignment and overall performance.

8.3 Processing data from sensor devices with approximated median filters

When we look at signals coming from various devices such as A/D converters, temperature sensors, or accelerometers, the data are noisy even in a perfect environment. In a real situation, where the accelerometers are, for example, used to stabilize various flight vehicles, the situation is even worse because of various vibrations caused by motors or propellers that are for example out of balance. When such a sensor acts as a central element controlling a process, it is necessary to remove the noise so as to prevent unwanted behaviour.

There are many filters that can be applied to smooth the measured data, for example, a variant of *low-pass filter*. The filter tries to keep the low frequency data while removing the high frequency noise (i.e. spikes). A low-pass filter usually is implemented in a situation where a limited number of computational resources are available because its implementation is simple. It can be implemented as an exponentially weighted moving average $x_{t+1} = \alpha y_t + (1 - \alpha)x_t$ where y_t represents data measured at time t and x_t the output value obtained at time t. Alternatively, a more robust Kalman filter may be used [13]. In contrast to the low-pass filter which has a fixed parameter α, Kalman filter is an adaptive estimator which minimizes the mean square error of the estimated parameters according to the previous state and actual measured value. Given only the mean and standard deviation of noise, the Kalman filter is the best linear estimator.

Unfortunately, there are two issues connected with the usage of linear filters. The first problem is that the data is being delayed by the filter which is a feature of linear filters when they are set to have a strong filtering effect. The second issue is that the filtered signal does not seem to follow the original measured data very well. To avoid the delay and provide results of high quality, we can employ an instance of median filter to smooth the measured data.

A relatively small number of samples are sufficient to be able to filter the measured data and remove the outliers. To demonstrate the benefits of the discovered approximations, we will apply the evolved 11-input and 13-input medians to filter the outliers presented in a signal captured by an accelerometer sensor. The obtained non-functional parameters are summarized in Tables 3 and 4. As the non-functional parameters are manually evaluated on real systems, only some of the Pareto dominant discovered solutions are investigated. It should be noted that only the number of operations and the quality defined by Eq. 1 was considered during construction of the Pareto set. We have implemented and measured not only the evolved solutions, but also three common approaches to determine median value—the *quicksort* algorithm, *quickselect* algorithm and the so called *running median*. While *quicksort* represents a sorting algorithm, the *quickselect* is a selection algorithm which is able to find the *k*th smallest element in an unordered list [4]. The quickselect uses the same overall approach as quicksort, however, it only recurses into one side of the input sequence which reduces the average complexity. The *running median* attempts to minimize processing time by maintaining a data list that is sorted from the smallest value to the largest value [25]. When a new sample is submitted, it replaces the oldest sample. The new sample is then shifted in the sorted list to bring it to the correct location.

Firstly, let us discuss size of the machine code of the complied C codes. If we compare the amount of bytes occupied by median networks and common approaches such as quicksort, quickselect and running median, we can easily

Table 3 Non-functional parameters of accurate (emphasized) and approximated implementations of 11-input median function measured on different MCUs

Impl.	Machine code size [B]				Execution time [µs]				Consumed energy [nWs]			
	STM32	PIC24	PIC16	TI430	STM32	PIC24	PIC16	TI430	STM32	PIC24	PIC16	TI430
14-ops	118	324	328	156	4.0	88	341	310	120	604	682	249
20-ops	158	441	452	224	4.6	108	450	356	140	745	900	286
25-ops	206	549	567	276	5.5	127	552	375	169	878	1105	301
30-ops	232	648	684	318	5.9	144	659	410	179	995	1318	329
32-ops	254	696	845	342	6.2	153	791	420	188	1057	1582	337
38-ops	294	819	1065	400	6.8	175	982	450	207	1208	1964	361
44-ops	328	900	1200	434	7.5	187	1105	465	230	1290	2210	373
50-ops	378	1032	1320	472	8.6	210	1220	480	261	1449	2440	385
qsort	128	333	–	196	40.5	958	–	1515	1235	6610	–	1217
qselect	212	849	607	276	17.5	488	2910	705	535	3367	5820	566
running	236	729	412	344	14.2	274	785	690	435	1887	1570	554

An implementation labelled as *n*-ops denotes evolved comparator network consisting of *n* operations

Table 4 Non-functional parameters of accurate (emphasized) and approximated implementations of 13-input median function measured on different MCUs

Impl.	Machine code size [B]				Execution time [µs]				Consumed energy [nWs]			
	STM32	PIC24	PIC16	TI430	STM32	PIC24	PIC16	TI430	STM32	PIC24	PIC16	TI430
17-ops	138	378	387	192	4.4	96	375	335	135	666	750	269
26-ops	214	588	594	288	5.6	136	587	390	172	935	1174	313
34-ops	288	747	922	402	7.0	163	880	470	215	1125	1760	377
38-ops	330	825	1054	434	8.0	176	980	480	244	1218	1960	385
41-ops	332	894	1144	516	8.0	190	1058	530	245	1309	2115	426
48-ops	398	1020	1306	574	8.8	210	1205	565	270	1452	2410	454
58-ops	478	1209	1590	642	9.5	242	1690	585	290	1672	3380	470
66-ops	496	1353	1854	666	10.2	266	1690	560	311	1835	3380	450
qsort	128	333	–	196	51.6	1178	–	1800	1574	8128	–	1445
qselect	212	849	607	276	21.4	610	4060	800	652	4212	8120	642
running	236	732	394	344	13.1	552	1100	750	400	3809	2200	602

determine that these implementations are more compact compared to the accurate median filter implemented using the median network consisting of 50 min/max operations for the 11-input median and 66 operations for the 13-input median. The size of the quicksort routine is equal to the size of the 11-input approximate median consisting of 14 operations. To sum up, quicksort is the most compact algorithm. Nevertheless, it is interesting to note that PIC16 does not allow one to execute the quicksort algorithm because its implementation relies on the recursion which cannot fit the in-memory emulated stack. The implementation of the running median occupies approximately a 1.8 times higher number of bytes on average compared to the quicksort. Quickselect consumes a bit more except for STM32 and MSP430 where the algorithm requires a lower number of bytes to be implemented.

The number of operations of the approximate median functions correlates with the machine code size. There is only one exception. The 13-input comparator network consisting of 38 operations implemented on STM32 exhibits nearly no reduction compared to the code consisting of 41 operations. It seems that some optimization tricks were discovered by the ARM compiler.

If we compare the size of machine code across all considered microcontrollers, the ARM architecture has an extremely efficient mechanism of instruction encoding. In addition, it revealed that the ARM compiler contains a very effective optimization engine. Similarly, the code generated by the MSP430 GCC compiler is very compact compared to the code for PIC microcontrollers. It is worth noting that it is extremely important to enable GCC optimizations. Otherwise, not only the size of the machine code, but also computation time increases by 60 % on average without changing a line of C code. When we take into account the fact that MSP430 is equipped with an exceptionally large FLASH memory (see Sect. 8.1), it seems to be a very powerful low-cost microcontroller.

The average execution time and average energy consumption measured for various implementations of 11-input and 13-input accurate as well as approximate median functions are given in the second and third part of Tables 3 and 4. Let us first discuss the parameters of the accurate implementations. During the measurements, it turned out that the energy consumption pattern remains almost invariant because all approximations use identical sequences of instructions. Consumed energy thus mainly depends on the execution time which is shorter when more aggressive approximations are applied. The average power consumption, when an accurate median is calculated, is 0.8 mW for MSP430, 2 mW for PIC16, 6.9 mW for PIC24 and 30.5 mW for ARM. In the case of the 11-input median function implemented on PIC24, the median network is 4.5 times faster than the quicksort algorithm and the consumed energy was reduced from 6610 to 1449 nWs (i.e. by 78 %). The median network is 4.7 times faster than quicksort on the STM32 and the energy was also reduced by 78 %. A little bit worse situation is at MSP430. The median network is 3.1 times faster than quicksort, but its energy consumption decreases by 68 %. Similar results were obtained for the 13-input median. The median network implemented on PIC24 is 4.4 times faster than the quicksort algorithm and the consumed energy was reduced by 77 %. At STM32, the quicksort algorithm exhibits 5 times worse execution time and an 80 % higher energy consumption compared to the median calculated using 66 min/max operations. At

MSP430, the median network is executed 3.2 times faster than quicksort. While there is a relative large performance gain of median networks compared to the quicksort algorithm, the execution time of running median is comparable with the median networks. The best improvement is achieved at STM32 where the implementations of median networks are executed 1.7 times faster than running median. For the rest, the gain varies around 1.4 on average.

Let us now move on to the execution time and energy consumption of the evolved approximate median functions. At first glance, it is evident that the execution time decreases with the decreasing number of operations. The situation is, however, a little bit complicated here. Let us compare the execution time of, for example, an 11-input accurate median network consisting of 50 operations and an 11-input reduced network consisting of 25 operations. While the number of operations is reduced by 50 %, the execution time is adequately decreased only at PIC16, where a 54 % improvement was achieved. STM32 and PIC microcontrollers exhibit improvement which is less than 39 %. In the case of MSP430, only a 22 % reduction was achieved. In order to better understand this phenomenon, we have to firstly investigate the dependence between the number of min/max operations and the number of generated instructions for MSP430. The implementation of an accurate median network consists of 219 instructions and the reduced median network consists of 122 instructions which leads to a 44 % improvement. Unfortunately, the difference between the improvement at the level of instructions (44 %) and improvement at the level of operations (50 %) is relatively small. In order to determine the root source of such a discrepancy, it is necessary to perform an analysis at the level of a machine code. It has been revealed that two different mechanisms were used to implement the min/max operations. Some operations are implemented using indirect addressing, other operations are optimized and consists of instructions only manipulated with registers. This makes a huge difference in the number of clock cycles required to execute a single min/max operation. Some operations are evaluated within 5 cycles, others require up to 11 clock cycles. Despite this finding, there is linear dependence between the energy consumption and time of execution and longer times imply a higher energy demand.

Despite the fact that STM32 has the largest current consumption in active mode, it provides the best results from the perspective of energy consumption. Even if the MSP430 is declared as an ultralow-power microcontroller, it requires about a 1.7 times higher amount of energy to execute the same code. It is necessary to note, however, that higher energy consumption is in close relation with the time of execution which is more than 60 times higher compared to STM32. Compared to PIC16 and PIC24, MSP430 consumes from 3 to 6 times lower energy to accomplish the same task. On the other hand, PIC24 is able to produce about five times more results within the same period of time at the cost of 30 % higher energy consumption compared to MSP430. In order to avoid misinterpretation, it is worth noting that MSP430 operates at 1 MHz while PIC24 operates at 4 MHz. When we increase the frequency to 4 MHz, the time of execution decreases four times with no additional cost (the energy consumption remains at the same level).

Figure 6 gives an example of real data filtered by various implementations of 11-input and 13-input median filters. The data were obtained from an accelerometer

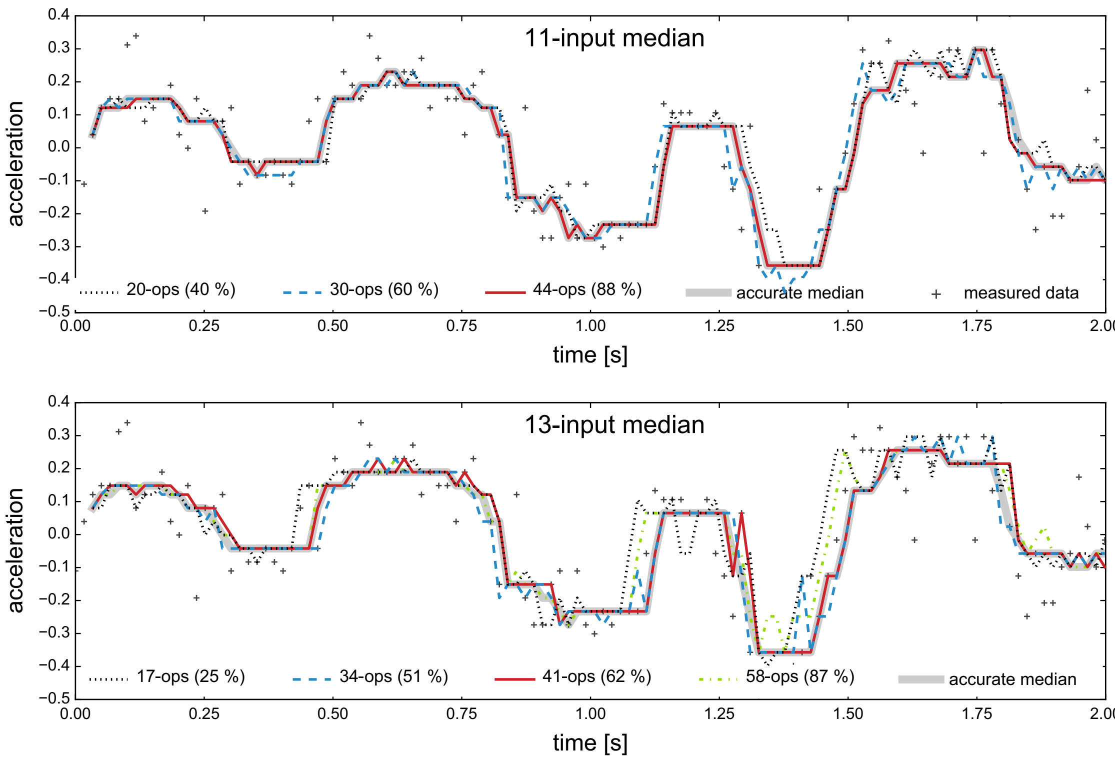


Fig. 6 Example of data filtered using accurate as well as approximate versions of 11-input and 13-input median filter. Note that only some of the *measured points* are shown because of readability

whose output signal was sampled at 8 kHz. Taking into account the sample rate, the considered accurate median filters introduce a delay not worse than 1.7 ms which represents a reasonable value. When six operations (12 %) are removed from the 11-input median network, the resulting approximate median produces an output that is nearly similar to the output of accurate implementation. There are only neglible differences that do not prevent us from applying this inaccurate implementation in an embedded application to filter the outliers and save energy. In the case of implementation at STM32, for example, we can reduce the consumed energy by 11.8 % by introducing the approximated median network consisting of 44 operations.

Interestingly, the median network which consists of 20 operations (60 %) produces a signal which is very similar to the output of an accurate median, despite the fact that the measured signal is very noisy. It seems that the output is of a better visual quality compared to the output of a network having 30 operations. In contrast to the output of an accurate median, there are some small oscillations around 1.7 seconds caused by the oscillations in a signal coming from the accelerometer. Nevertheless, the trend in data is reliably followed. In case these small differences do not represent a real problem for a target application, it is worth implementing the improved median network which is able to offer a 40 % reduction of power consumption on the one hand, its approximately 1.8 times faster execution time on the other hand.

The approximate versions of the 13-input median also performs very well. Only small differences are observable when a median network with 38 % removed operations is used. Compared to a commonly used running median, we obtain a solution which has 38 % lower power consumption when implemented on a STM32 microcontroller. Interestingly, the approximate median networks which consists of 17 and 58 operations exhibit lower delay compared to the fully working 13-input median. It can be seen that the filtered data appears to be shifted to the left when these filters are used.

It can be concluded that the observations on real examples are consistent with conclusions given in Sect. 7. As the quality of an approximate median defined by Eq. 1 decreases, the amount of inaccuracies in the output signal increases. The processing of the sensor data seems to be an application with great potential for genetic improvement. As was previously shown, we are able to significantly improve energy consumed by the filters for a cost of small differences in the output data. In fact, any of the presented approximations can be used to filter the input signal because no golden solution is available for the validation of the obtained outputs. The filtration is typically used to avoid high variances in output signal (i.e. to reduce sensitivity of the output signal to the outliers). In this sense, we can employ approximate medians consisting of 50 % (or even less) operations to accomplish this task because they are able to sufficiently remove the outliers.

8.4 Median in image processing

The processes of acquiring, transmitting and storing images in computer systems are not always ideal and hence some pixels or groups of pixels can be corrupted. Hence, noise elimination is a typical low level image processing task. In many applications, the noise elimination has to be implemented by non-linear functions because the noise contained in the images is inherently non-linear [6]. A typical representative of non-linear noise is a shot noise which manifests itself by setting some individual pixels to a random value. Median-based non-linear filters play a prominent role among the filters utilized to suppress the shot noise [2]. Traditionally, a simple median filter applied to every pixel of the input image is employed. In advanced image filters (e.g. switching filters [32]) the filtering function is only applied if a noise detector, implemented typically using a median, detects some noise.

The image filters operate with pixel values in the neighbourhood of the centre pixel. The process of filtration is based on a sliding window, a square window of an odd size $(2k+1)$, that moves along the image. More formally, let I be an image consisting of $m \times n$ pixels $x(i,j) \in I$, where $1 \leq i \leq m, 1 \leq j \leq n$. Then, each pixel of the filtered image I' is calculated as $y(m,n) = \mathrm{median}(W_I(m,n))$, where $W_I(m,n) = \{x(m+i, n+j) \in I \mid -k < i,j < k\}$ is a sliding window function. It is evident that the median value is calculated using $(2k+1)^2$ pixels. The typical sliding windows employed in image processing consists of 3×3 and 5×5 pixels which corresponds with 9-input and 25-input filtering functions.

The measured non-functional parameters of various implementations of 9-input accurate as well as approximate median filters are summarized in Table 5. Apart from the evolved implementations, two common approaches to determine a median value are evaluated—the quicksort algorithm and the quickselect algorithm. The

Table 5 Non-functional parameters of accurate (emphasized) and approximated implementations of 9-input median function measured on different MCUs

Impl.	Machine code size [B]				Execution time [µs]				Consumed energy [nWs]			
	STM32	PIC24	PIC16	TI430	STM32	PIC24	PIC16	TI430	STM32	PIC24	PIC16	TI430
9-ops	78	204	207	96	3.2	65	228	274	97	450	457	220
10-ops	84	234	238	108	3.3	71	256	280	102	492	512	225
14-ops	112	315	324	156	3.9	86	338	310	118	590	675	249
16-ops	126	372	376	176	4.1	96	386	324	126	666	771	260
20-ops	158	441	454	208	4.6	108	452	340	141	745	905	273
22-ops	180	495	502	234	5.0	118	506	360	151	818	1012	289
26-ops	208	573	586	280	5.4	132	576	388	165	909	1153	312
30-ops	240	645	676	330	6.4	144	650	412	196	994	1299	331
qsort	128	333	–	196	26.8	830	–	1325	816	5727	–	1064
qselect	212	849	607	272	15.3	466	2255	690	467	3219	4510	554

running median discussed in the previous section is not applicable in this case because more than one value has to be removed/added between two subsequent processing windows. The discussion that has been given for the implementation of the 11-input median and its variants is also valid for the 9-input alternative whose parameters are included in Table 5. There is nearly a linear dependency between the number of operations used to approximate the median value and the execution time as well as power consumption.

The results for the 25-input median and its alternative implementations are given in Table 6. In contrast with the 13-input approximate medians, the difference between the improvement achieved at the level of operations and improvement at the level of instructions does not exceed 5 %. Similarly, the time of execution decreases linearly with a decreasing number of operations with one exception—implementation compiled for MSP430 which suffers from issues observed also for 13-input approximate medians. There is an 18 % difference between the reduction at the level of instructions and the reduction of execution time (see the execution time for 174-ops and 60-ops implementations). Since the response of other architectures to a reduced number of operations is as expected, it may suggest that there may be a problem with the quality of the compiled code. We did not analyse this problem in detail as it is outside the scope of this paper.

The chosen problem nicely demonstrates the overhead of median networks implemented in the software. The accurate median function implemented using 174 operations occupies ten times more bytes than the quicksort algorithm. Even if we remove half of the operations, the machine code is more than six times larger. This is the price that must be sacrificed for greater speed of the algorithm based on a median network. As regards the execution time, the median can be calculated 70 % faster when the median network which consists of 174 operations is used instead of the quicksort algorithm and 31 % faster when compared to the quickselect

Table 6 Non-functional parameters of accurate (emphasized) and approximated implementations of 25-input median function measured on different MCUs

Impl.	Machine code size [B]			Execution time [μs]			Consumed energy [nWs]		
	STM32	PIC24	TI430	STM32	PIC24	TI430	STM32	PIC24	TI430
60-ops	502	1302	742	10.9	262	665	333	1808	534
70-ops	596	1527	912	12.3	303	785	375	2091	630
88-ops	796	1887	1180	16.4	366	955	501	2525	767
107-ops	920	2250	1438	18.4	428	1100	562	2953	883
150-ops	1264	3015	1688	23.9	554	1130	727	3823	907
160-ops	1378	3195	1818	24.6	584	1200	751	4030	964
164-ops	1454	3255	1826	26.0	596	1240	793	4109	996
174-ops	1524	3423	1864	27.6	619	1270	841	4271	1020
qsort	128	333	196	104.0	2430	2610	3172	16,767	2096
qselect	212	849	276	39.1	1040	1685	1194	7176	1353

Note that PIC16 is not included in this table due to small amount of available RAM memory algorithm. The 25-median implemented using 150 operations enables us to reduce the energy by more than 10 %. According to the distribution of errors shown in Fig. 5, this implementation provides an output of high quality with a low percentage of erroneous outputs that are close to the median value.

In order to evaluate the filtering quality as well as robustness of the evolved approximate medians, the medians were employed as median filters and evaluated using 25 randomly selected test images (384x256 pixels) from [17] that were corrupted by random valued shot noise. Because the removal of random valued shot noise represents a difficult problem, it usually is used to compare the performance of various median filters [5]. Considering the fact that a sliding window is used, more than two million test cases were in fact used for quality assessment. There exists several approaches to measure the quality of filtered images. The structural similarity index (SSIM) represents probably the most advanced approach which attempts to quantify the visibility of errors (differences) between a distorted image and a reference image[37].

Boxplots of the structural similarity index calculated for 9-input and 25-input accurate as well as approximate median networks used as image filters are given in Fig. 7. As is evident from the results, there is a relatively large variance in the similarity index of accurate as well as inaccurate median filters. The index of similarity for images produced by accurate an 9-input median filter is 88.6 % in average. The average similarity index decreases with the decreasing number of operations. Interestingly, it decreases very slightly without any radical change in variance. When we reduce the number of operation to 16 (53 %), the average similarity index decreases to 87.5 %. The results suggest that it is possible to use an approximate median network consisting of half the number of operations instead of an accurate median. The impact on quality of the filtered images is negligible.

Figure 8 illustrates filtering capabilities of various filters on an image corrupted by random valued shot noise where 10 % of the pixels are affected. The output of the median filter and approximate median filter is visually indistinguishable. Nearly all of the noisy pixels were successfully detected and removed, even for a median with 16 operations. When we reduce the number of operations to 14, a few noisy pixels remain in the filtered image. This behaviour corresponds with the distribution

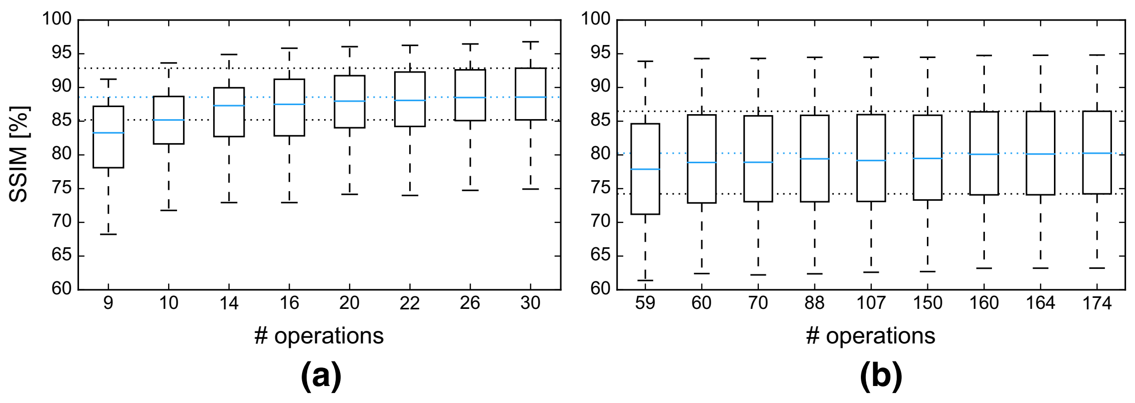


Fig. 7 *Boxplots* illustrating the distribution of structural similarity index for evolved median networks calculated using a set of test images corrupted by 10 % random valued shot noise. **a** 9-input median. **b** 25-Input median

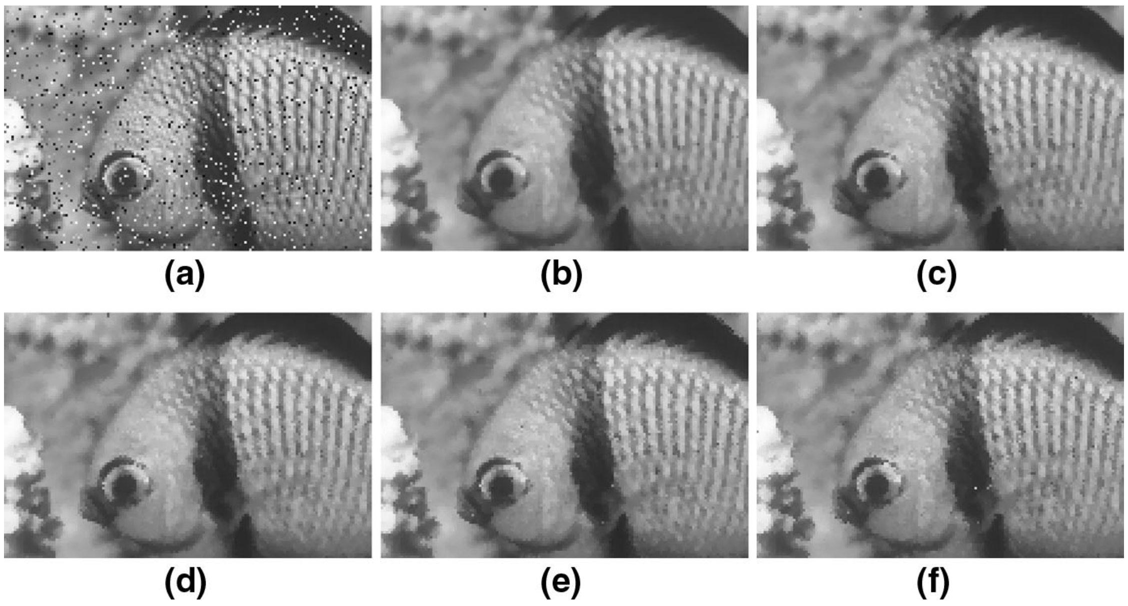


Fig. 8 Detail of an image **a** corrupted by 10 % random valued shot noise filtered by **b** 9-input accurate median filter and approximated median filters consisting of **c** 22 (73 %) operations, **d** 16 (53 %), **e** 14 (46 %) and **f** 10 (33 %) operations

of errors shown in Fig. 5 and a detailed analysis provided in Table 2. The 9-input approximate median with 16 operations exhibits the worst-case distance error equal to one, while the 14-ops implementation has the worst-case distance error equal to two.

If we compare the distribution of the similarity index for a 9-input and 25-input median filter, it is evident that the 25-input median filters provide results of lower quality. The similarity index of the accurate implementation consisting of 174 operations is equal to 80.3 %. The reason is obvious. Increasing the size of the filtering window allows for the common median filter to remove a great deal of noisy pixels, however, because the standard median filters modify almost all pixels, images become smudged and less detailed. Nevertheless, this fact does not prevent the employment of the 25-input median filter as a robust noise detector. Interestingly, there is only a small degradation in quality of the reduced 25-input median filters. When we remove 50 % of operations, the similarity index decreases to 79.4 % on average. This approximation yields a 40 % reduction in power consumption when implemented on STM32 microcontroller.

Similar conclusions may be inferred even if we use the peak signal-to-noise ratio (PSNR) which represents another commonly used quality metric. In contrast to structural similarity, PSNR does not respect a psycho-visual model of the human optical system. While PSNR of the images filtered by the accurate 9-input median filter is equal to 29.4 dB in average, PSNR of the images obtained by the 14-ops (9-ops) filter drops by 1.3 dB (3.5 dB). The PSNR of the images filtered by the accurate 25-input median filter is equal to 25.9 dB. When the number of operations is reduced to 59, the PSNR only decreases by 0.7 dB.

The results demonstrate how robust the evolved implementations are and that there is great space for improvement of the non-functional parameters in practice. In most cases, it is not even necessary to exactly determine the median value which helps us to reduce the power consumption or increase the performance (i.e. speed) of a given piece of software.

9 Conclusions

In this paper, we presented a new approach to improve non-functional properties of software. In particular, we concentrated on improvements in the execution time and power consumption of various instances of the median function. In general, it is impossible to improve non-functional parameters of the median function without accepting occasional errors in results since optimal implementations of typical instances are available. In order to address this problem, we adopted the approximate computing scenario which allows us to accept some errors in the outputs.

In approximate computing, software and hardware is approximated, i.e. simplified with respect to fully accurate implementations, in order to reduce power consumption or increase performance. As a consequence, errors can emerge during computations which is tolerable in many real applications. When an approximation should be introduced, the common approach is to remove the less significant bits and reduce data widths. This paper shows that the approximation conducted at the level of function (algorithm) that are based on EA is able to deliver significantly better results.

The median is implemented using a sequence of elementary operations that forms a median network. The task is formulated as a single objective optimization problem where the number of operations represents constraints specified by the designer. The constrains-oriented approach is relevant to practice where the designers usually wants to achieve a particular power reduction in order to improve the performance of the whole embedded system. The method is based on cartesian GP and exploits the fact that GP is able to find a good trade-off between the error and number of operations, even if the number of operations is intentionally constrained.

In order to avoid problem with seeding (only fully functional implementations of various instances of median filter exist), we proposed to apply a two-stage procedure. The first stage starts with a fully functional median network and gradually reduces the number of employed operations in order to satisfy constraints given by a designer. As soon as a satisfactory candidate solution is found, the second stage responsible for maximizing the quality of partially working implementations is used.

The accuracy of determining a median value is measured by means of a problem-specific quality metric. The proposed metric is based on the positional error calculated using the permutation principle introduced in this paper. The impact of the permutation principle was discussed from a theoretical as well as a practical point of view. Firstly, the permutation principle helps us to reduce the computational complexity of the fitness evaluation. Secondly, the permutation principle enables one to construct a metric approaching the quality of selecting the median value and, what is important, which can be efficiently calculated. Finally, the permutation principle helps to understand how to avoid biased solutions that may be produced when we generate test vectors used to determine the fitness score inappropriately (randomly). In order to understand this phenomenon, it is necessary to realize that the median value is determined according to a set of values (i.e. the ordering of input values is completely ignored). It was illustrated and discussed that it is necessary to generate test vectors from different equivalence classes so as to avoid any bias.

The problem of trading between quality and non-functional parameters was demonstrated in four different instances of the median function that are typically employed in practice. The performance of the best discovered approximated median filters was evaluated in two real-world problems—sensor data processing and image processing. The non-functional parameters were measured for four microcontrollers so as to avoid misleading conclusions. The results confirmed that median functions are very good examples of functions for which it makes sense to introduce their approximate versions. When the approximate medians are employed in a particular application, the output quality remains relatively high, even for significant reductions of the number of operations. Hence significant improvements in energy consumption can be obtained.

Even though the permutation principle as well as the proposed error metric are problem specific, this paper demonstrated the ability of GI to provide competitive solutions for a chosen real-world problem from the area of approximate computing. This opens a complete new application area for GI. The ability to deliver partially working solutions seems to be natural for evolutionary techniques. Hence the approximate computing seems to have a great potential for these techniques.

There are several directions for future research. Execution time and power consumption are two possible non-functional criteria that can be optimized. There are additional criteria such as delay that need to be considered, especially if median networks would be implemented in the hardware. Despite the fact that the proposed permutation principle helps to significantly improve the time required to determine the fitness value, the test based approach used to calculate the fitness score represents a bottleneck of the whole framework. Unfortunately, this is a general problem of all generate-and-test-based evolutionary approaches. As a consequence of that, only a subset of all possible permutations was used for quality assessment. This simplification introduces two issues. Firstly, it means that we are not able to guarantee the worst-case error unless all the input permutations are tested. Secondly, it may happen that the quality of a given network is worse than determined. Suprisingly, the experiments revealed that our simplification does not have any significant effect in practice. We are convinced, however, that these issues can be completely eliminated by introducing a formal method based on BDDs to the fitness function.

Acknowledgments This work was supported by the Czech science foundation project 14-04197S—Advanced Methods for Evolutionary Design of Complex Digital Circuits.

References

1. A. Agapitos, S.M. Lucas, Evolving efficient recursive sorting algorithms, in *IEEE Congress on Evolutionary Computation*, pp. 2677–2684 (2006)
2. R.H. Chan, C.W. Ho, M. Nikolova, Salt-and-pepper noise removal by median-type noise detectors and edge-preserving regularization. IEEE Trans. Image Process. **14**, 1479–1485 (2005)
3. B. Cody-Kenny, E.G. Lopez, S. Barrett, locoGP: improving performance by genetic programming java source code, in *Genetic Improvement 2015 Workshop*, ed. by W.B. Langdon, J. Petke, D.R. White (ACM, Madrid, 2015), pp. 811–818
4. N. Devillard, *Fast Median Search: An ANSI C Implementation* (1998). http://ndevilla.free.fr/median/median.pdf
5. Y. Dong, A new directional weighted median filter for removal of random-valued impulse noise. IEEE Signal Process. Lett. **14**(3), 193–196 (2007)
6. E.R. Dougherty, J.T. Astola, (eds.) *Nonlinear Filters for Image Processing. SPIE/IEEE Series on Imaging Science and Engineering*. SPIE/IEEE (1999)
7. H. Esmaeilzadeh, A. Sampson, L. Ceze, D. Burger, Neural acceleration for general-purpose approximate programs. Commun. ACM **58**(1), 105–115 (2014)
8. B.W. Goldman, W.F. Punch, Analysis of cartesian genetic programming's evolutionary mechanisms. IEEE Trans. Evol. Comput. **19**(3), 359–373 (2015)
9. J. Han, M. Orshansky, Approximate computing: An emerging paradigm for energy-efficient design, in *Proceedings of the 18th IEEE European Test Symposium*, pp. 1–6. IEEE (2013)
10. M. Harman, B.J. Jones, Search-based software engineering. Inf. Softw. Technol. **43**, 833–839 (2001)
11. W.D. Hillis, Co-evolving parasites improve simulated evolution as an optimization procedure. Phys. D **42**(1–3), 228–234 (1990)
12. H. Juille, Evolution of non-deterministic incremental algorithms as a new approach for search in state spaces, in *Genetic Algorithms: Proceedings of the 6th International Conference (ICGA95)*, ed. by L. Eshelman (Morgan Kaufmann, Pittsburgh, PA, USA, 1995), pp. 351–358
13. R.E. Kalman, A new approach to linear filtering and prediction problems. Trans. ASME J. Basic Eng. **82**(Series D), 35–45 (1960)
14. D.E. Knuth, *The Art of Computer Programming*, vol. 3, 2nd edn. (Sorting and Searching. Addison Wesley Longman Publishing Co., Inc, Redwood City, 1998)
15. W.B. Langdon, M. Harman, Optimizing existing software with genetic programming. IEEE Trans. Evol. Comput. **19**(1), 118–135 (2015)
16. R. Maronna, D. Martin, V. Yohai, *Robust Statistics: Theory and Methods, Wiley Series in Probability and Statistics* (Wiley, New Jersey, 2006)
17. D. Martin, C. Fowlkes, D. Tal, J. Malik, A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics, in *Proceedings of the 8th International Conference Computer Vision*, vol. 2, pp. 416–423 (2001)
18. J.F. Miller, *Cartesian Genetic Programming* (Springer, Berlin, 2011)
19. J.F. Miller, S.L. Smith, Redundancy and computational efficiency in cartesian genetic programming. IEEE Trans. Evol. Comput. **10**(2), 167–174 (2006)
20. V. Mrazek, Z. Vasicek, L. Sekanina, Evolutionary approximation of software for embedded systems: Median function, in *Genetic Improvement 2015 Workshop*, ed. by W.B. Langdon, J. Petke, D.R. White (ACM, Madrid, 2015), pp. 795–801
21. K. Nepal, Y. Li, R.I. Bahar, S. Reda, Abacus: A technique for automated behavioral synthesis of approximate computing circuits, in *Proceedings of the Conference on Design, Automation and Test in Europe, DATE '14*, pp. 1–6. EDA Consortium (2014)
22. J. Petke, M. Harman, W.B. Langdon, W. Weimer, Using genetic improvement and code transplants to specialise a C++ program to a problem class, in *17th European Conference on Genetic Programming, LNCS*, vol. 8599, ed. by Miguel Nicolau, et al. (Springer, Granada, Spain, 2014), pp. 137–149
23. R. Poli, W.B. Langdon, N.F. McPhee, *A Field Guide to Genetic Programming*.Published via http://lulu.com and http://www.gp-field-guide.org.uk (2008)
24. A. Sampson, W. Dietl, E. Fortuna, Gnanapragasam, D., Ceze, L., Grossman, D.: Enerj: Approximate data types for safe and general low-power computation, in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 164–174. ACM (2011)

25. P. Schmidt, Simple median filter library designed for the arduino platform (2014). https://github.com/daPhoosa/MedianFilter
26. E. Schulte, J. Dorn, S. Harding, S. Forrest, W. Weimer, Post-compiler software optimization for reducing energy, in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'14 (ACM, Salt Lake City, 2014), pp. 639–652
27. L. Sekanina, Evolutionary design space exploration for median circuits, in *Applications of Evolutionary Computing, LNCS 3005*, pp. 240–249. Springer (2004)
28. L. Sekanina, M. Bidlo, Evolutionary design of arbitrarily large sorting networks using development. Genet. Progr. Evolv. Mach. **6**(3), 319–347 (2005)
29. L. Sekanina, Z. Vasicek, Approximate circuits by means of evolvable hardware. in *Proceedings of the 2013 IEEE Symposium Series on Computational Intelligence (SSCI), 2013 IEEE International Conference on Evolvable Systems*, pp. 21–28. IEEE CIS (2013)
30. P. Sitthi-Amorn, N. Modly, W. Weimer, J. Lawrence, Genetic programming for shader simplification. ACM Trans. Gr. **30**(6), 152:1–152:12 (2011)
31. J.L. Smith, Implementing median filters in xc4000e fpgas. XCell **23**(1), 16 (1996)
32. T. Sun, Y. Neuvo, Detail-preserving median based filters in image processing. Pattern Recognit. Lett. **16**, 341–347 (1994)
33. V.K. Valsalam, R. Miikkulainen, Using symmetry and evolutionary search to minimize sorting networks. J. Mach. Learn. Res. **14**(1), 303–331 (2013)
34. Z. Vasicek, L. Sekanina, Evolutionary approach to approximate digital circuits design. IEEE Trans. Evol. Comput. **19**(3), 432–444 (2015)
35. Z. Vasicek, K. Slany, Efficient phenotype evaluation in cartesian genetic programming, in *Proceedings of the 15th European Conference on Genetic Programming, LNCS 7244*, pp. 266–278. Springer Verlag (2012)
36. S. Venkataramani, A. Sabne, V.J. Kozhikkottu, K. Roy, A. Raghunathan, Salsa: systematic logic synthesis of approximate circuits, in *The 49th Annual Design Automation Conference 2012*, DAC '12, pp. 796–801. ACM (2012)
37. Z. Wang, A. Bovik, H. Sheikh, E. Simoncelli, Image quality assessment: from error visibility to structural similarity. IEEE Trans. Image Process. **13**(4), 600–612 (2004)
38. D.R. White, A. Arcuri, A. John, Evolutionary improvement of programs. IEEE Trans. Evol. Comput. **15**(4), 515–538 (2011)
39. A. Yazdanbakhsh, D. Mahajan, B. Thwaites, J. Park, A. Nagendrakumar, S. Sethuraman, K. Ramkrishnan, N. Ravindran, R. Jariwala, A. Rahimi, H. Esmailzadeh, K. Bazargan, Axilog: Language support for approximate hardware design, in *Design, Automation and Test in Europe, DATE'15*, pp. 1–6. EDA Consortium (2015)

Appendix F

Automatic Design of Arbitrary-Size Approximate Sorting Networks with Error Guarantee

MRAZEK, Vojtech and VASICEK, Zdenek. “Automatic Design of Arbitrary-Size Approximate Sorting Networks with Error Guarantee”. In: *26rd International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2016.* Bremen, DE: IEEE Computer Society, 2016, (to appear).

acceptance rate: 34%, contribution of the author of the thesis: 50%

Referenced on pages: 43, 44

Automatic Design of Arbitrary-Size Approximate Sorting Networks with Error Guarantee

Vojtech Mrazek
Brno University of Technology,
Faculty of Information Technology,
Centre of Excellence IT4Innovations
Email: imrazek@fit.vutbr.cz

Zdenek Vasicek
Brno University of Technology,
Faculty of Information Technology,
Centre of Excellence IT4Innovations
Email: vasicek@fit.vutbr.cz

***Abstract*—Despite the fact that hardware sorters offer great performance, they become expensive as the number of inputs increases. In order to address the problem of high-performance and power-efficient computing, we propose a scalable method for construction of power-efficient sorting networks suitable for hardware implementation. The proposed approach exploits the error resilience which is present in many real-world applications such as digital signal processing, biological computing and large-scale scientific computing. The method is based on recursive construction of large sorting networks using smaller instances of approximate sorting networks. The design process is tunable and enables to achieve desired tradeoffs between the accuracy and power consumption or implementation cost. A search-based design method is used to obtain approximate sorting networks. To measure and analyze the accuracy of approximate networks, three data-independent quality metrics are proposed. Namely, guarantee of error probability, worst-case error and error distribution are discussed. A significant improvement in the implementation cost and power consumption was obtained. For example, 20% reduction in power consumption was achieved by introducing a small error in 256-input sorter. The difference in rank is proved to be not worse than 2 with probability at least 99%. In addition to that, it is guaranteed that the worst-case difference is equal to 6.**

I. Introduction

Sorting is one of the most fundamental operations that is widely used in many applications in computer science including digital signal processing, biological computing and large-scale scientific computing [1].

The hardware sorters are typically employed to improve the performance of applications operating over big data sequences. The sorters can be used either to sort a given sequence [2], [3] or to compute quantiles [4], [5]. These operations represent a typical task performed in database systems, machine learning or business intelligence to distill summary information from huge data sets [4]. In these areas, FPGA-based systems have become popular due to their inherent ability to achieve various trade-offs between throughput and power consumption [1], [3].

On the other hand, the sorting is employed in solving completely different problems. Switching networks, multi-access memories and multiprocessors can be implemented using hardware sorters [6]. In addition to that, sigma-delta digital modulators and various sorter-based arithmetic circuits such as adders, exponential, hyperbolic and logarithmic functions have been proposed recently [7].

The hardware sorters can be classified to two main categories – linear sorters and sorting networks [2]. While linear sorters process one element at a time, sorting networks operate in parallel over the input elements. As a consequence of that, the hardware implementation of linear sorters is usually compact but it fails to scale in performance. On the contrary, sorting networks offer great performance but they become expensive as the number of inputs increases.

Several techniques have been proposed to reduce the area and power consumption of sorting networks. For example, Zuluaga et. al. [2] proposed a domain-specific language and compiler that automatically generates hardware implementations of sorting networks with reduced area optimized for latency or throughput. The area reduction was achieved by reusing the common parts of sorting networks. Chen et. al. [1] introduced a concept of streaming permutation network that was obtained by folding the Clos network. The permutation network was used to construct a high-throughput and a low cost architecture. Compared to [2], significantly better memory as well as energy efficiency was achieved.

Although a lot of effort has been put into the improvement of cost of sorters, it has been demonstrated that many applications from signal processing, computer vision and machine learning exhibit an inherent tolerance to errors in computation [8]. As the power consumption become a critical factor for digital designs, inexact or approximate computing seems to be a viable approach to reduce consumption of many real-world systems and improve the overall efficiency of computers.

Despite the fact that many papers have been published in the field of approximate computing, there is no paper that explicitly addressed the problem of trading the quality of hardware sorters for power efficiency even if there is potential for doing that. The only paper that addressed the problem of approximate sorting was introduced by Leighton and Plaxton in early nineties [9]. The authors theoretically proved existence of an n-input sorting circuit of depth $7.44 \log n$ that sorts all but superpolynomially small fraction of the all possible input permutations. Unfortunately, the hardware implementation remains impractical due to the fact that there is a trade-off between the value of the multiplicative constant and the success probability, and a significant increase in the constant is required for practical instance sizes [9].

A. Our contributions elaborated in this paper

We introduce a scalable method for a construction of arbitrary-size approximate sorting networks. In order to build a large approximate sorting network, smaller instances of approximate or accurate sorting networks are employed. The principle of construction based on recursive bitonic algorithm is inherently tunable to the level of accuracy required for a target application because various approximate as well as accurate sorting networks can be combined together. This gives us the opportunity to obtain approximate sorting networks exhibiting various trade-offs between quality and implementation cost.

In order to design small approximate sorting networks having up to 32 inputs, a systematic search-based design method is proposed. The method works in such a way that it starts with a known architecture of accurate sorting network (generated using bitonic algorithm) that is subsequently optimized (i.e. reduced) to meet the target constraints while introducing a minimal error. The constraints specified by designer can include target implementation cost or target power reduction. The obtained approximate networks can either be applied to construct a larger network or employed autonomously.

Traditionally, a randomly generated set of test vectors is applied to assess the quality of an approximate circuit. This approach, unfortunately, provides no guarantee on the error and make it difficult to predict the behavior of an approximate circuit under different conditions (e.g. when different data-width is used or data with different input distribution are processed). In order to address this problem, we have introduced a method that is able to formally prove and guarantee worst-case error. In addition to that, error distribution can be calculated. Both metrics are based on an extension of zero-one and permutation principle.

II. Sorting networks

The concept of sorting networks was originally studied in 1950s by Armstrong, Nelson and O'Connor and deeply elaborated in 1960s by Knuth [10]. Sorting network is defined as a network consisting of a sequence of elementary operations denoted as *compare-and-swap* (CS) operations that sorts all input sequences. A compare-and-swap operation of two elements, a and b, compares a and b and exchanges (if it is necessary) the elements in order to obtain sorted sequence (a', b'), i.e. $a' = \min(a, b)$, $b' = \max(a, b)$. In hardware, CS is implemented using two multiplexers that are controlled by means of a comparator that determines the maximum of the two (see Figure. 1a).

The sequence of compare-swap operations executed by a sorting network depends only on the number of elements to be sorted, not on the values of the elements. It means that the sequence of comparisons is fixed. This fact represents the main advantage of sorting networks because such a structure can be efficiently implemented using a parallel pipelined hardware architecture. Compared to the linear sorters, the sorting networks do not require to implement a control logic.

A. Representation of sorting networks

The sorting networks are composed solely of wires and comparators. In fact, each comparator implements a two-input sorter. In order to represent sorting networks efficiently, Knuth introduced a notation consisting of vertical segments and horizontal wires [10]. Each vertical segment connecting the two elements being compared represents a single CS operation with arrow determining the larger value (see Figure 1b). A horizontal wire represents an element of input sequence and transmits values from place to place. The unsorted elements (inputs) appear on the left and the sorted sequence is obtained on the right, with the smaller input element appearing on the top output and the larger input element appearing on the bottom output. All comparisons that can be performed in parallel represents a single stage.

B. Construction of sorting networks

Sorting networks can be generated from basic sorting algorithms such as bubble or insertion sort. Both algorithms provide structurally equivalent architectures [3]. Unfortunately, networks generated by these approaches are inefficient like their algorithmic counterparts and consist of many comparator elements.

In order to improve the efficacy, various algorithms have been proposed in literature [10]. The Batcher odd-even merge-sort and bitonic sort represent two simple, yet most efficient algorithms. These algorithms produce sorting networks of the same asymptotic complexity $O(n \log^2 n)$ and the same depth $O(\log^2 n)$ which makes them efficient for parallel implementation. Although the bitonic sorters contain a little bit more comparators, they hardware implementation is the preferred one because all signal paths are of the same length. In addition to that, the same number of comparisons is used in each stage.

Bitonic sorting algorithm is based on repeatedly merging two bitonic sequences to form a larger bitonic sequence. A sequence is bitonic if it can be split to two parts such that the first part is monotically increasing and the second part monotically decreasing, or it can be circularly shifted to become so. The merging operation representing a key step of algorithm is called bitonic merge. The input to this operation is a pair of sequences that are sorted in opposite directions, one in ascending order and the other in descending order, so that together they form a bitonic sequence. Bitonic merge takes this bitonic sequence and from it forms a single sorted sequence. A

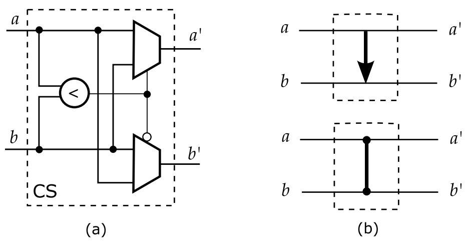


Fig. 1. Compare-and-swap operation: (a) hardware implementation, (b) two equivalent schematic representations using Knuth's notation.

complete sorter can be constructed from small bitonic sorters by successively bitonic sorting and merging smaller sequences into larger sequences until we have a bitonic sequence of size n.

In order to reduce high implementation cost, various modifications of bitonic sorting algorithm have been proposed. Stone [11] suggested to employ the perfect shuffle enabling the reuse of $n/2$ processing elements. This approach substantially improved the implementation cost but $\log^2 n$ cycles are required to obtain the sorted sequence. In [12], Lee et al. have improved the time complexity of Stone's algorithm to $\frac{1}{2}\log n(\log n + 1)$ introducing an additional logic. The latest modification has been proposed by Chen et.al. [1]. The authors employed a streaming permutation network based on Clos network which is programmable and performs all the data permutations in the bitonic sorting network.

C. Optimal sorting networks

Although a complete and deep theory has been developed around sorting networks, nobody has discovered a sorting algorithm producing the optimal (i.e. minimal) sequence of comparison operators. Even if the best known sorting algorithms such as Bitonic sorting exhibit an optimal asymptotic complexity, there is a large constant factor hidden in the asymptotic bound. The optimal sequence of comparison operators is known only for some instances. The construction of optimal sorting networks is extremely difficult problem even for small number of inputs. For long time, nobody was able to prove the optimality of sorting networks introduced more than 40 years ago by Knuth in [10]. Recently, Bundala and Zavodny proposed a method that is able to construct optimal-depth networks for $n \leq 16$ in reasonable time [13]. Then, Ehlers et al. proved existence of optimal-depth sorting network for $n = 17$ and discovered faster networks for 17, 19 and 20 inputs than the previously known best ones [14]. In [15], Codish et al. proved the optimality of 9-input and 10-input sorting network consisting of 25 and 29 comparators found by Floyd and Waksman in the seventies.

III. Quality of approximate sorters

One of the main issues in the approximate computing is the assessment of quality of approximations. The most popular measure for quality is the mean squared difference between the specification and the output of the approximate circuit that is estimated using a set of test vectors.

The problem of the general quality measures is that they do not assess the quality of sorting process. What's worse, the obtained result depends on a particular set of test vectors. In addition to that, there is no guarantee on the error (e.g. the worst-case error) because only a fraction of all possible input vector was used. In order to address these problems, three data-independent quality measures are introduced in this section.

Let us recall the basic properties of the accurate as well as approximate sorting networks, i.e. comparison networks in general. Let $C(x_1, \ldots, x_n)$ be a comparison network with n inputs, $x_i \in A$ and A be a totally ordered set of elements. It is guaranteed by construction that each comparison network produces a permutation of the input sequence. It means that there exists one to one mapping between the values obtained at the output of comparison network and the values at the input, so no new value can arise during the exchanging performed by compare-and-swap elements. Formally, $C : \pi(x_1, \ldots, x_n) \to \pi(x_1, \ldots, x_n)$. Hence, every approximate sorting network must produce a partially ordered output for at least one input sequence. In such a case, there must exist at least two outputs that are returning an invalid value.

In general, 2^{wn} input combinations exist to evaluate an n-input sorting network operating with elements encoded using w-bit integers. Clearly, it is intractable to evaluate all possible input combinations, however, the number of input combinations can substantially be reduced by applying the zero-one principle [10] and permutation principle [16].

A. Error probability

According to zero-one principle, 2^n binary sequences are sufficient to determine the error rate. Let $C(\mathbf{x})[i]$ denote value of i-th output ($1 \leq i \leq n$) of an n-input n-output comparison network C. Let $E_i \subseteq \{0,1\}^n$ be the set of all possible input assignments $\mathbf{x} \in \{0,1\}^n$ for whose an invalid output value is produced, where

$$\begin{aligned} E_i = \quad & \{\mathbf{x} : C(\mathbf{x})[i] = 0 \wedge (x_1 + \ldots + x_n) > (n-i)\} \cup \\ & \{\mathbf{x} : C(\mathbf{x})[i] = 1 \wedge (x_1 + \ldots + x_n) \leq (n-i)\} \end{aligned} \tag{1}$$

Then, $e_i = 2^{-n} \cdot |E_i|$ is the probability that an invalid value is obtained at the i-th output of C. The overall accuracy, i.e. the relative number of correct responses, is equal to

$$accuracy = 1 - \frac{1}{n}\sum_{i=1}^{n} e_i \tag{2}$$

Although it is impractical to determine the size of E_i explicitly by enumerating the assignments satisfying the condition given in Equation 1 (the number of input assignments grows exponentially with the increasing number of inputs n), the number of such assignments, i.e. $|E_i|$, can be calculated easily using a Constaint satisfaction problem (CSP) solver or Binary-Decision Diagrams (BDDs) even for large instances.

Note that the error probability has to be evaluated carefully in practice because there can exist a network with high error rate, but still providing good performance because nearly sorted sequences are produced in most cases.

B. Approximation guarantees

A sorting network can be understood as a structure that computes n quantiles in parallel. The first quantile represents the minimum and the last quantile represents the maximum. Then, we can investigate the difference in rank between the true quantile produced by the accurate sorting network and that of the output produced by the approximate network.

Let us give a simple example. Let $\mathbf{x} = (1, 4, 3, 0, 2)$ be an input sequence, S be sorting network and C be an approximate sorting network producing output $C(\mathbf{x}) = (0, 2, 1, 3, 4)$. It is clear that C returned a partially sorted sequence because the second and third items are invalid, i.e. $C(\mathbf{x})[2] \neq S(\mathbf{x})[2]$ and $C(\mathbf{x})[3] \neq S(\mathbf{x})[3]$. The second output returned the third lowest item of $\mathbf{x}$ (i.e. $C(\mathbf{x})[2] = S(\mathbf{x})[3]$) and the third output returned the second lowest item (i.e. $C(\mathbf{x})[3] = S(\mathbf{x})[2]$). In both cases, the difference in rank is equal to one.

Zero-one principle and CSP solver can be employed to perform formal worst-case error analysis efficiently. Since asymmetric difference in rank may occur at some outputs, it seems to be reasonable to investigate the left (δ^L) and right (δ^R) worst-case distances separately. Firstly, let us define two predicates

$$\begin{aligned} P_L(\mathbf{x}, i, d) &: C(\mathbf{x})[i] = 0 \wedge (x_1 + \dots + x_n) = n - i + d \\ P_R(\mathbf{x}, i, d) &: C(\mathbf{x})[i] = 1 \wedge (x_1 + \dots + x_n) = n - i - d + 1 \end{aligned} \tag{3}$$

The problem of the worst-case error analysis can be formulated using Pseudo-Boolean CSP as follows. For each output $i \in \{1, \dots, n\}$ find maximal $\delta \in \{0, \dots, n - i - 1\}$ such that $\exists \mathbf{x} \in \{0, 1\}^n : P_L(\mathbf{x}, i, \delta)$. Then, $\delta^L[i] = \delta$ is the left worst-case distance for i-th output. Similarly, the right bound $\delta^R[i]$ can be determined using $P_R(\mathbf{x}, i, \delta)$ instead. Note that it is beneficial to use binary search algorithm to maximize δ because it significantly reduces the number of CSP queries.

Knowledge of the worst-case error alone will not suffice since its probability of occurrence could be negligible. To address this problem, we propose a technique to obtain an error distribution that would provide information about probability of occurrence of errors of different distances. Although it is possible to determine the true error distribution (by counting the number of input assignments that satisfy P_L and P_R, similarly as it was discussed in the previous section), it is practically sufficient and computationally significantly faster to estimate the error distribution. In order to do that, we can adopt permutation principle introduced in [16] and employed to determine the distance between an arbitrary comparator network (i.e. approximate sorting network) and a sorting network. The permutation principle states that it is sufficient to prove response to the permutations of a set consisting of n distinct elements to precisely determine quality of an arbitrary comparison network.

Let us give an example for $n = 3$. Let S be sorting network and C be approximate sorting network consisting of two compare-and-swap operation. Let the first comparator be connected to the first and second horizontal wire, the second comparator be connected to the second and third horizontal wire. In our case, $A = \{1, 2, 3\}$ which gives us six possible permutations that have to be be considered, i.e. $|\pi(A)| = 6$. We can easily determine that $C(\mathbf{x}) = S(\mathbf{x})$ iff $\mathbf{x} \in \pi(A) \setminus \{(2, 3, 1), (3, 2, 1)\}$, i.e. for 4 out of 6 cases. In the remaining two cases, the output of C equals to $(2, 1, 3)$. It means that the first as well as the second output produce erroneous value whose difference in rank is equal to one in both directions (left and right). The results can be summarized using a matrix $\mathcal{H}$ which captures the number of input assignments that cause error at output i whose difference in rank is equal to j. Note that $j = 0$ means that correct response was obtained. The number of correct responses for each output is given in the main diagonal. For example, $h_{3,3} = 1$ because the third output produces always correct result; $h_{1,1} = 4/6$ because there are two cases for that an incorrect response is returned by the first output. The complete $\mathcal{H}(C)$ is as follows:

$$\mathcal{H}(C) = \frac{1}{6} \begin{pmatrix} 4 & 2 & 0 \\ 2 & 4 & 0 \\ 0 & 0 & 6 \end{pmatrix} \tag{4}$$

Interestingly, a relative small subset of all possible permutations is required in practice to obtain a reasonable estimate of error distribution. For example, only 10000 out of more than 10^{77} vectors are required in average to obtain an estimate exhibiting 0.3% relative error (in worst-case) compared to the true error distribution $\mathcal{H}$ computed using BDDs for $n = 256$.

IV. Construction of approximate sorters

In order to construct approximate sorting networks, we propose to modify the Bitonic sorting algorithm as follows.

Algorithm 1: Approximate bitonic sorting

```
Input: unsorted sequence $X$, direction $dir \in \{\uparrow, \downarrow\}$
Output: sorted sequence $X$
1  Function sort(dir, X)
2      if $|X| = 1$ then
3          return X;
4      else if $|X| = 2^B$ then
5          return b-sort(dir, X);
6      else
7          $h \leftarrow |X| \div 2$;
8          $a \leftarrow \mathrm{sort}(\uparrow, (x_0, \dots, x_{h-1}))$;
9          $b \leftarrow \mathrm{sort}(\downarrow, (x_h, \dots, x_{2h-1}))$;
10         return merge(dir, $(a_0, \dots, a_{h-1}, b_0, \dots, b_{h-1})$);
11 Function merge(dir, X)
12     if $|X| = 1$ then
13         return X;
14     else if $|X| = 2^M$ then
15         return b-merge(dir, X);
16     else
17         $h \leftarrow |X| \div 2$;
18         for i = 0 to $h - 1$ do
19             if $x_i > x_{(h+i)} \Leftrightarrow dir =\uparrow$ then
20                 swap $x_i$, $x_{(h+i)}$
21         $a \leftarrow \mathrm{merge}(dir, (x_0, \dots, x_{h-1}))$;
22         $b \leftarrow \mathrm{merge}(dir, (x_h, \dots, x_{2h-1}))$;
23         return $(a_0, \dots, a_{h-1}, b_0, \dots, b_{h-1})$;
```

The algorithm consists of two parts – sorting and merging. The input sequence is successively divided into two halves

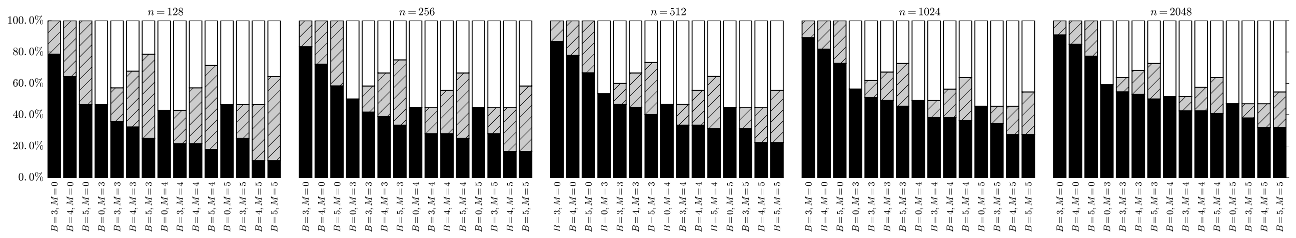


Fig. 2. The relative number of compare-and-swap elements occupied by 2^M-input b-mergers (see □) and 2^B-input b-sorters (see ▨) compared to the total number of compare-and-swap elements for various number of inputs n, and selected values of B and M.

until two elements remain. Then, these halves are successively merged until a single sorted sequence is obtained. Two subcircuits can be identified in the resulting sorter – let us call them b-sorters and b-mergers. Our goal is to replace the 2^B-input b-sorters and 2^M-input b-mergers with their approximate versions (see Figure 3). Such a substitution reduces not only the total number of comparisons but may also decrease quality of the sorter. Hence a reasonable trade-off needs to be identified. The designer has the possibility to tune both parameters because various values of B and M can be chosen. In addition to that, approximate networks (i.e. b-sorters and b-mergers) of different quality can be employed.

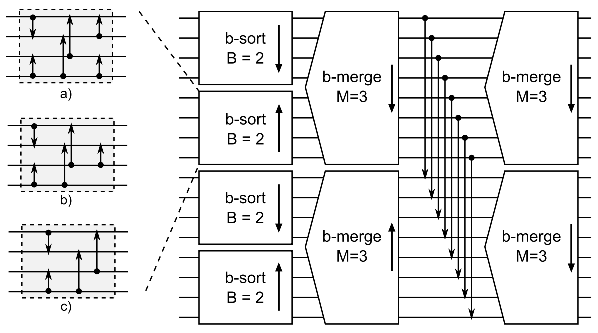


Fig. 3. Sub-circuits (b-sorters and b-mergers) in the 16-input sorter generated using Bitonic algorithm for $B = 2$ and $M = 3$. In addition to that, eight compare-and-swap operations are required. Each b-sorter can be replaced with various 4-input comparison networks, sorter (a) or some approximation (b,c).

As it is non-trivial to predict what values yields the best trade-offs, Figure 2 shows the distribution of compare-and-swap operations for sorting networks having from $n = 128$ to $n = 2048$ inputs. The total number of CSs consists of three groups – the CSs that are required to implement 2^B-input b-sorters, the CSs that are required to implement 2^M-input b-mergers and the remaining ones. The distribution shows the possible area reduction that can be achieved by choosing various values of B and M The upper-bound is limited by the number of comparators that can't be approximated. For $n = 128$, $B = 5$ and $M = 0$, for example, 27% reduction in the total number of CSs can be achieved when the 32-input b-sorters are replaced with their approximate versions occupying half of the resources. Such a reduction is possible because the b-sorters comprise 54% of the total number of CSs.

A. Design of approximate b-sorters and b-mergers

As evident, there is a great potential for improvement in the implementation costs as well as power consumption of sorters especially when larger values of B and M are employed. The only problem is how to obtain high-quality approximations of b-mergers and b-sorters blocks.

The problem of finding an approximate network can be formulated as a constrained optimization problem where the goal is to find a comparison network C exhibiting maximum quality for a target number of compare-swap operations. In order to solve this problem efficiently, we propose to employ Cartesian genetic programming (CGP) [17]. CGP is easy to implement, it can easily handle constraints, it is naturally multi-objective and high-quality approximate circuits have already been obtained in literature [18].

We propose to apply a two-stage procedure. At the beginning, the designer specifies the target reduction that should be achieved. The first stage starts with an exact and accurate solution (i.e. b-sorter or b-merger). The goal is to gradually modify the initial solution and obtain a reduced network of the target cost. In the second stage, which begins as soon as the target reduction was achieved, the search method reflects not only the implementation cost, but also the quality. The second stage aims to improve the quality as much as possible.

CGP employs a simple population-oriented search method. The λ candidate solutions that forms the population are generated from the parental solution (i.e. the best individual discovered so far) using a mutation operator slightly modifying the candidate solutions (up to h randomly chosen genes are modified). Then, the candidate solutions are evaluated and each member receives the so-called fitness score. The highest-scored individual becomes a new parent of the next population. The fitness function $F(C)$ summarize the quality of candidate solutions into a single value and is defined as follows:

$$F(C) = -\begin{cases} \infty & \text{if constraint is violated} \\ \sum_{i,j=1}^{n} h_{i,j}(i-j)^2 & \text{otherwise,} \end{cases} \tag{5}$$

where $h_{i,j}$ is an element of the error matrix $\mathcal{H}(C)$ calculated for a candidate comparison network C. The constraint is represented by the target number of compare-swap operation. The fitness function is designed in such a way that the individuals of higher quality receive higher score. In addition to that,

TABLE I
PARAMETERS OF ACCURATE AND FIVE APPROXIMATE 16-INPUT SORTERS

Impl.	**CSs**	**Depth**	**Quality indicators**						**FPGA Synthesis results**			**ASIC Synthesis results**	
	N	D	*accuracy*	Δ_{avg}	Δ_{95}	Δ_{99}	Δ^L	Δ^R	#LUTs	#REGs	Power (W)	Area (μm^2)	Power (mW)
C1	60 (100%)	10	100%	0.00	0	0	0	0	769 (100%)	1120 (100%)	0.24 (100%)	68945 (100%)	1.22 (100%)
C2	49 (81%)	10	49%	0.63	2	3	6	6	621 (81%)	1112 (99%)	0.23 (97%)	58863 (85%)	1.09 (89%)
C3	39 (65%)	10	30%	1.13	3	4	8	8	525 (68%)	1104 (99%)	0.22 (92%)	49698 (72%)	0.96 (79%)
C4	29 (48%)	8	20%	1.70	4	6	9	9	445 (58%)	784 (70%)	0.20 (86%)	37851 (55%)	0.75 (62%)
C5	24 (40%)	5	17%	2.05	5	7	10	10	289 (38%)	640 (57%)	0.20 (82%)	29247 (42%)	0.54 (45%)
C6	19 (31%)	5	16%	2.31	6	8	11	11	253 (33%)	592 (53%)	0.19 (80%)	24664 (36%)	0.48 (40%)

it promotes solutions having narrower error distribution. The fitness score of an accurate b-sorter (b-merger) is equal to zero.

The candidate solutions consists of 2-input elements that can act as compare-and-swap operation or simple buffer. The following encoding of candidate solutions is proposed. Each element is encoded using a triplet consisting of three integers (s, l, f). The first integer $s \in \{1, \dots, n\}$ defines the index of a horizontal wire where the first input is connected to. The second integer $l \in \{1, \dots, n-s\}$ determines the length of the corresponding vertical segment. The value, in fact, indirectly encodes the index of horizontal wire where the second input is connected to. Finally, the last integer $f \in \{0^{noop}, 1^{\uparrow}, 2^{\downarrow}\}$ determines the operation of the encoded element. In case that f=0, the element is treated as empty operation and is ignored. Otherwise, a compare-and-swap operation with a given direction is utilized. ASAP scheduling is employed to obtain corresponding comparison network. The proposed encoding guarantee that a valid comparison network is always captured. The number of triplets is defined by the initial solution and remains fixed during the whole search process. The mutation operator modifies values of up to h randomly chosen integers.

The sorting network shown in Figure 3a can be encoded, for example, using 8 triplets as (0,1,$1^{\downarrow}$) (2,1,$2^{\uparrow}$) (1,3,0^{noop}) (1,2,$2^{\uparrow}$) (0,2,$2^{\uparrow}$) (1,3,$1^{\downarrow}$) (2,3,0^{noop}) (2,1,$2^{\uparrow}$). Note that there are two inactive elements (encoded by 3rd and 7th triplet) that do not have any impact on the structure of resulting network.

V. EXPERIMENTAL RESULTS

Firstly, we approximated small sorting networks. In particular, 8, 16 and 32 inputs were considered. The number of inputs was chosen in accordance with the analysis shown in Figure 2 and corresponds with $B = 3, 4, 5$. The search algorithm uses population consisting of $\lambda = 20$ individuals. Up to $h = 5$ integers are modified by mutation operator. The fitness function is calculated using 64×10^4, 256×10^4 and 1024×10^4 randomly generated permutations for $n = 8, 16, 32$, respectively. The optimization process is terminated either when no improvement in fitness score is achieved within the last 15 minutes or the maximum amount of time (2 hours) is exhausted. In case of 8-input (16-input) sorters, the optimization was initialized with the optimal known sorter consisting of 19 (60) compare-and-swap operations. The reference 32-input sorter was obtained using Bitonic sorting algorithm. Ten design points (i.e. design constraint) are considered for each problem instance. The goal is to find approximate sorters consisting of about 95%, 90%, 80%, 70%, 60%, 50%, 40%, 30% and 20% compare-and-swap operations compared to the number of operations of the initial solutions. In addition to that, it is requested that the depth of the approximate sorters is not worse than the depth of the initial accurate sorter.

In total, 30 independent experimental runs were performed and more than $3 \cdot 9 \cdot 30 = 810$ unique approximate solutions were discovered. The Pareto-optimal solutions were identified and implemented as fully streaming pipeline architectures in Virtex-7 FPGA XC7VX330T using Xilinx Vivado and as 45nm VLSI circuits using Cadence Encounter. Then, we conducted the post-place-and-route power analysis performed at 250 MHz (900 MHz for ASIC). Switching activity analysis was employed to improve the accuracy of power estimation.

Due to the limited space, let us discuss the results obtained for 16-input instance. Table I summarizes the parameters of the accurate (C1) and five chosen Pareto-optimal approximate (C2-C6) sorters. Namely, it contains the number of compare-and-swap operations (N), depth (D), quality indicators and sythesis results for FPGA and VLSI. The quality indicators include the accuracy and five differences in rank – mean (Δ_{avg}), 0.95-quantile (Δ_{95}), 0.99-quantile (Δ_{99}) and both worst-cases (Δ^L and Δ^R); all determined over all outputs. The percentages in parentheses indicate the ratio of the value in that column compared to the accurate sorter. As evident, the accuracy gradually decreases with the increasing amount of removed compare-and-swap operations. The same observation is also valid for the implementation cost in ASIC as well as FPGA (see the number of LUT tables and the number of registers). As expected, the implementation cost correlates with N in both cases. A slightly different situation is in the case of power consumption. Because the 16-input sorters are relative small circuits, static part of power consumption dominates in FPGA the dynamic one. As a consequence of that, only 20% improvement in power consumption was achieved for implementation C6 despite more than 50% reduction in implementation cost. No such discrepancy is observable for sorters implemented as ASIC.

Figure 4 depicts the quality matrices of the discovered implementations. Contrasted to the quality indicators, the quality matrices helps to better understand the quality of approximations. The interpretation of $\mathcal{H}$ is as follows. All possible input sequences are correctly sorted in the case of exact sorter (implementation C1). In all cases, the i-th output

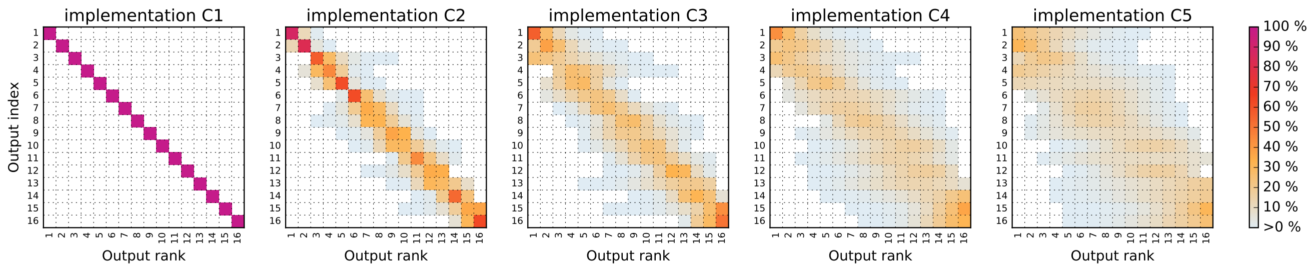


Fig. 4. Quality matrices $\mathcal{H}$ of accurate (C1) and four approximate (C2–C5) 16-input sorting networks

returned the i-th smallest element. It means that only main diagonal contains non-zero values.

Let us discuss the implementation C2 consisting of 49 compare-and-swap operations, i.e. about 18% less compared to the exact sorter. According to the eight column of $\mathcal{H}$, the eight smallest element (i.e. the element with rank 8) is returned by the eight output with probability 33%. This element, however, can be for some input sequence returned also by the output seven (or nine) in 29.3% (17.9%) cases. It means that the difference in one rank occurs in 47.2% in practice. Difference in two ranks (elements is returned by 6th or 10th output) occurs in 19.8% cases. According to the first row of $\mathcal{H}$, it can be determined that the minimum is correctly identified in more than 87.4% cases. In the rest of the cases, the second or third smallest element is returned with prob. 11.8% and 0.8%, respectively. We can conclude that this approximation is of a high quality despite the fact that the worst-case error is equal to 6. According to the quality matrix and Δ_{99}, the difference in rank is not worse than 3 for more than 99% input sequences.

The same principle was applied to approximate 8-input, 16-input and 32-input b-mergers. The initial accurate b-mergers were extracted from the sorters generated using Bitonic sorting algorithm. The experimental setup was kept the same as in the case of sorters. Only the test vectors utilized in the fitness functions are generated in different way. Since two sorted sequences are expected at the input of each merger, it is not necessary to test all the input permutations. Only $\binom{n}{\frac{n}{2}}$ input sequences are required to exhaustively evaluate the quality.

A. Construction of large approximate sorters

The approximate sorters and mergers discovered in the previous experiments were used to construct large approximate sorters for $n = 256, 512, 1024$ and 2048 inputs. The sorters were constructed according to the Algorithm 1. The following parameters were considered: $B = \{0, 3, 4, 5\}$ and $M = \{0, 3, 4, 5\}$. To simplify the problem, ten Pareto-optimal implementations of various sorter (merger) instances were chosen to act as b-sorter (b-merger). In total, 121 architectures were generated and analyzed for each n.

The results for $n = 256$ are summarized in Table II. In addition to the parameters discussed earlier, the value of B and M parameter and the number of compare-and-swap operations required by b-sorters and b-mergers are included. While the whole range of B was utilized, smaller values of M are preferred. It seems that the inaccuracy introduced into the larger mergers has a negative impact on the overall quality. Hence, the majority of architectures employ a variant of 8-input merger (i.e. M=3). The b-sorters represent 10% to 31% the total number of CSs. Compared to the b-sorters, b-mergers occupy significantly larger portion of the CSs especially when the accuracy is higher than 50%. The implementation cost as well as power consumption decreases with decreasing N linearly.

Interestingly, it seems to be nontrivial to predict the quality of the constructed approximate network according to the qual-

TABLE II
PARAMETERS OF ACCURATE AND ELEVEN APPROXIMATE 256-INPUT SORTERS

Impl.	**Width**		**Compare-swap operations**			**Depth**	**Quality indicators**						**FPGA Synthesis results**		
	B	M	*sorters*	*mergers*	N	D	*accuracy*	Δ_{avg}	Δ_{95}	Δ_{99}	Δ^L	Δ^R	#LUTs	#REGs	Power (W)
S1	—	—	0%	0%	4608 (100%)	36	100%	0.00	0	0	0	0	55297 (100%)	73728 (100%)	5.997 (100%)
S2_C2	4	4	19%	50%	4112 (89%)	36	92%	0.08	1	1	3	3	49857 (90%)	72320 (98%)	5.433 (91%)
S3	5	4	31%	40%	3880 (84%)	36	85%	0.16	1	1	6	6	47713 (86%)	70784 (96%)	5.154 (86%)
S4	3	3	11%	36%	3584 (78%)	35	60%	0.44	1	2	6	4	44033 (80%)	69376 (94%)	4.826 (80%)
S5	5	3	24%	29%	3288 (71%)	35	47%	0.93	3	6	9	10	41985 (76%)	67328 (91%)	4.505 (75%)
S6	5	3	25%	27%	3192 (69%)	38	32%	1.24	4	6	10	11	41449 (75%)	67272 (91%)	4.424 (74%)
S7_C6	4	3	10%	33%	3120 (68%)	33	28%	1.42	4	6	15	13	38977 (70%)	64256 (87%)	4.238 (71%)
S8	5	3	22%	26%	2936 (64%)	35	20%	1.78	5	7	14	17	38497 (70%)	62400 (85%)	4.012 (67%)
S9_C5	4	3	14%	19%	2688 (58%)	27	18%	2.05	6	8	20	16	32491 (59%)	50924 (69%)	3.601 (60%)
S10_C5	4	3	15%	15%	2560 (56%)	27	12%	2.42	6	9	20	18	30991 (56%)	46610 (63%)	3.553 (59%)
S11	5	3	14%	17%	2232 (48%)	23	9%	3.70	9	13	32	28	27133 (49%)	43358 (59%)	3.024 (50%)
S12	5	3	15%	13%	2136 (46%)	23	7%	4.27	10	13	32	29	26717 (48%)	40509 (55%)	2.897 (48%)

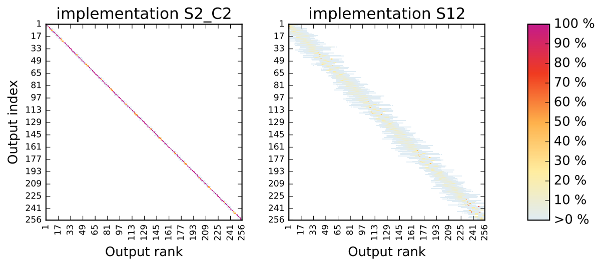


Fig. 5. Quality matrices of two approximate 256-input sorters

ity of the small b-sorters. For example, b-sorters in S2_C2 are implemented using approximate sorters C2. While C2 exhibits only 49% accuracy, S2_C2 provides the correct response for 92% of all possible input sequences. Not only the accuracy was improved, but also the worst-case error is significantly lower (see Δ^L and Δ^R). The same effect is observable also for S7_C6. On the contrary, similar quality is achieved for S9_C5 and S10_C5. These findings suggest that if a b-merger of higher quality is applied, the quality of b-sorter could be improved significantly.

The quality matrices for two chosen implementations are shown in Figure 5. It can be concluded, that the large approximate networks are of high quality even when the number of CSs was reduced significantly. In particular, architecture S2_C2 achieves 10% reduction in power consumption with hardly visible error. In at least 99.9%, the difference in rank is not worse than 1. Implementation S4 exhibiting 20% reduction in power consumption guarantees that the difference is not worse than 2 (5) with probability at least 99% (99.9%), see Δ_{99}. Finally, even implementation S12 could be safely used in many non-critical applications (see Figure 5).

VI. Conclusion and remarks

We addressed the problem of design of approximate sorting networks suitable for hardware implementation exhibiting trade-off between the quality and power consumption. Intuitively, it seems to be sufficient to successively remove the first stages of sorting networks. Unfortunately, our initial experiments revealed that this approach yields non-optimal solutions. Hence, we proposed a scalable method for construction of approximate sorting networks exhibiting trade-off between the quality and power consumption. The method is based on recursive construction of large sorting networks using smaller instances of approximate sorting networks that are designed using a search-based design method.

Many approximate circuits have been proposed in recent years. The correctness, however, is typically guaranteed for precise data and only some estimation is promised for the approximate data [8]. The designers are then reluctant to use such circuits. The strength of our method is that the quality of the approximate networks is guaranteed and formally proved for arbitrary data widths.

Although the power consumption was optimized indirectly by reducing the number of compare-and-swap operations, we have experimentally confirmed that a significant improvement in power consumption can be achieved for sorters implemented not only in FPGAs but also as VLSI circuits. Naturally, the discovered sorters can be employed to improve the computation efficiency of algorithms running on CPUs and GPUs.

Probably due to the lack of a formal apparatus for analysis of approximation guarantees, no survey devoted to the approximate sorters and their applications has been published up to now. Let us mention two applications offering a great space for lowering the computational effort (e.g., energy consumption). Firstly, the small approximate sorters can directly be employed to improve the power consumption of many sorter-based arithmetic circuits or network arbiters [7]. On the other hand, our method can be adopted to produce large networks for power-efficient approximate processing of massive quantile queries, a problem with many real-world applications [5]. In order to do that, the fitness function should reflect the quality of some outputs only.

Acknowledgment

This research was supported by the Czech science foundation project GA16-17538S and Brno University of Technology project FIT-S-14-2297.

References

[1] R. Chen, S. Siriyal, and V. Prasanna, "Energy and memory efficient mapping of bitonic sorting on FPGA," in *Proceedings of the 2015 ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: ACM, 2015, pp. 240–249.

[2] M. Zuluaga, P. A. Milder, and M. Püschel, "Computer generation of streaming sorting networks," in *Design Automation Conference*, 2012, pp. 1245–1253.

[3] R. Mueller, J. Teubner, and G. Alonso, "Sorting networks on FPGAs," *The VLDB Journal*, vol. 21, no. 1, pp. 1–23, 2012.

[4] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, "Approximate medians and other quantiles in one pass and with limited memory," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*. New York, USA: ACM, 1998, pp. 426–435.

[5] X. Lin, J. Xu *et al.*, "Approximate processing of massive continuous quantile queries over high-speed data streams," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 5, pp. 683–698, 2006.

[6] K. E. Batcher, "Sorting networks and their applications," in *Proc. of the spring Joint Comp. Conf.*, ser. AFIPS '68. New York, USA: ACM, 1968, pp. 307–314.

[7] H. Fujisaka, T. Kamio, C. J. Ahn *et al.*, "Sorter-based arithmetic circuits for sigma-delta domain signal processing – part I: Addition, approximate transcendental functions, and log-domain operations," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 59, no. 9, pp. 1952–1965, Sept 2012.

[8] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, 2016.

[9] T. Leighton and C. G. Plaxton, "A (fairly) simple circuit that (usually) sorts," in *Proc. 31st Annu. Symp. Foundations of Computer Science*, 1990, pp. 264–274.

[10] D. E. Knuth, *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1998.

[11] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. 20, no. 2, pp. 153–161, Feb. 1971.

[12] J.-D. Lee and K. E. Batcher, "Minimizing communication in the bitonic sort," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 5, pp. 459–474, May 2000.

[13] D. Bundala and J. Závodný, "Optimal sorting networks," in *Proc. of the Language and Automata Theory and Applications: 8th International Conference*. Cham: Springer International Publishing, 2014, pp. 236–247.

[14] T. Ehlers and M. Müller, "New bounds on optimal sorting networks," in *Proc. Evolving Computability: 11th Conference on Computability in Europe, CiE 2015*. Cham: Springer Int. Publishing, 2015, pp. 167–176.

[15] M. Codish, L. Cruz-Filipe, M. Frank, and P. Schneider-Kamp, "Twenty-five comparators is optimal when sorting nine inputs (and twenty-nine for ten)," in *26th IEEE Int. Conf. Tools with Artificial Intelligence, ICTAI 2014*, pp. 186–193.

[16] Z. Vasicek and V. Mrazek, "Trading between quality and non-functional properties of median filter in embedded systems," *Genetic Programming and Evolvable Machines*, 2016 *(to be published)*.

[17] J. F. Miller, *Cartesian Genetic Programming*. Springer-Verlag, 2011.

[18] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Trans. Evol. Comput.*, vol. 19, no. 3, pp. 432–444, June 2015.

Appendix G

Design of Power-Efficient Approximate Multipliers for Approximate Artificial Neural Networks

MRAZEK, Vojtech, SARWAR, Shakib Syed, SEKANINA, Lukas, VASICEK, Zdenek, and ROY, Kaushik. “Design of Power-Efficient Approximate Multipliers for Approximate Artificial Neural Networks”. In: *Proceedings of the 35th IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Austin, TX, US, 2016. to appear in.

acceptance rate: 25% (2015), contribution of the author of the thesis: 10%

Referenced on pages: 32

Design of Power-Efficient Approximate Multipliers for Approximate Artificial Neural Networks

Vojtech Mrazek[1] imrazek@fit.vutbr.cz
Syed Shakib Sarwar[2] sarwar@purdue.edu
Lukas Sekanina[1] sekanina@fit.vutbr.cz
Zdenek Vasicek[1] vasicek@fit.vutbr.cz
Kaushik Roy[2] kaushik@purdue.edu

[1]Faculty of Information Technology, Centre of Excellence IT4Innovations
Brno University of Technology
Brno, Czech Republic

[2]School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA

ABSTRACT

Artificial neural networks (NN) have shown a significant promise in difficult tasks like image classification or speech recognition. Even well-optimized hardware implementations of digital NNs show significant power consumption. It is mainly due to non-uniform pipeline structures and inherent redundancy of numerous arithmetic operations that have to be performed to produce each single output vector. This paper provides a methodology for the design of well-optimized power-efficient NNs with a uniform structure suitable for hardware implementation. An error resilience analysis was performed in order to determine key constraints for the design of approximate multipliers that are employed in the resulting structure of NN. By means of a search based approximation method, approximate multipliers showing desired tradeoffs between the accuracy and implementation cost were created. Resulting approximate NNs, containing the approximate multipliers, were evaluated using standard benchmarks (MNIST dataset) and a real-world classification problem of Street-View House Numbers. Significant improvement in power efficiency was obtained in both cases with respect to regular NNs. In some cases, 91% power reduction of multiplication led to classification accuracy degradation of less than 2.80%. Moreover, the paper showed the capability of the back propagation learning algorithm to adapt with NNs containing the approximate multipliers.

Categories and Subject Descriptors

B.6.3 [**Hardware**]: Logic Design—*Automatic synthesis*; I.2.6 [**Computing Methodologies**]: Artificial Intelligence

ICCAD '16, November 07-10, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4466-1/16/11...$15.00
DOI: http://dx.doi.org/10.1145/2966986.2967021

1. INTRODUCTION

Recent advances in artificial intelligence methods and a huge amount of computing resources available on a single chip have led to a renewed interest in efficient implementations of complex neuromorphic systems based on artificial neural networks (NNs). Implementing complex NNs in low power embedded systems requires careful optimization strategies at various levels including neurons, interconnects, learning algorithms, data storage and memory access. This work is focused on reducing power consumption of computations performed in neurons, which is of the same importance as optimizing the data storage and memory access [7].

Inexact or approximate computing has been adopted in recent years as a viable approach to reduce power consumption and improve the overall efficiency of computers. In approximate computing, circuits are not implemented exactly according to the specification, but they are simplified in order to reduce power consumption or increase operation frequency. It is assumed that the errors occurring in simplified circuits are acceptable, which is typical for error resilient application domains such as multimedia, classification and data mining. Applications based on NNs have proven to be highly error resilient [2].

This paper provides a methodology for the design of well-optimized power-efficient NNs that have a uniform structure (i.e. all nodes are identical in all layers) which is thus suitable for hardware implementation. An error resilience analysis is performed in order to determine key constraints for the design of approximate multipliers that are employed in the resulting structure of NN. In order to avoid a manual approximation of accurate multipliers, systematic methods capable of performing approximations have been introduced recently [21, 20, 14]. These methods typically start with a gate-level description of the accurate circuit and an error constraint that specifies the type of error that can be accepted. The approximation algorithm is typically constructed as a design space exploration algorithm directly approximating some parts of the circuit [11] or the whole circuit [18]. The search is guided by an error metric such as the average error magnitude or maximum arithmetic error.

In addition to developing highly-optimized power efficient

NNs, an automated design space exploration method is proposed. The method is capable to design approximate multipliers in such a way that the resulting multipliers satisfy not only a given error, but also a set of other application-specific constraints.

2. ARTIFICIAL NEURAL NETWORKS

In machine learning, artificial neural networks are a family of models inspired by biological neural networks. A typical artificial neural network consists of an input layer of neurons, several hidden layers of neurons and an output layer of neurons.

2.1 Artificial Neuron

A typical structure of neuron is as follows [4, 22]. The output h_i of neuron i is defined as $h_i = \sigma(\sum_{j=1}^{N} w_{ij}x_j - \theta)$, where $\sigma(\cdot)$ is an activation function, N is the number of inputs of the neuron, w_{ij} is weight of the link, x_j is the j-th input and θ is a threshold or bias. The purpose of the activation function is (in addition to introducing non-linearity into NN) to map the resulting values into the interval $(-1, 1)$ or $(0, 1)$. The activation can be a threshold function, semi-linear or non-linear function. A common example of the non-linear function, which is used in this work, is sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$.

2.2 Architecture and learning

The NNs are classified into feed-forward neural networks (FNNs), recurrent neural networks (RNN) and their combination. In RNNs, there is at least one feedback connection. The earliest and the simplest architecture is the perception model which utilizes just one layer of output neurons that are connected with all the inputs. The extended version, the multilayer perception model (MLP), uses one or more layers (a.k.a. hidden layers) of neurons between the input and output layers. In the hidden layer, each neuron is directly linked to the outputs of the previous layer. An important contribution to the state of the art in NNs has been the development of large-scale NNs such as the convolutional NNs introduced by LeCun [9], where more types of layers (e.g. convolutional layers) are employed. Another type of layers is the average pooling layer which is used for weighted subsampling. Nowadays there are many different application-specific layers intended for, e.g., image classification [8], segmentation [1], speech processing [6] etc.

Learning is the most important capability of neural networks. It is performed by an algorithm that iteratively updates the synapses (weights) and other parameters of neural network. Determining the most suitable parameters and weights of NN can be viewed as a complex nonlinear optimization problem. Learning methods are usually divided into supervised, unsupervised, reinforcement, and evolutionary methods [4]. The most popular algorithms for supervised learning, which we will employ, are the least mean squares method and back propagation algorithm [4].

2.3 Approximations in NNs

As neural networks are inherently error-resilient, various approaches have been proposed to approximate them [13].

Venkataramani et al. [19] proposed a methodology of identifying error-resilient neurons based on the backpropagation gradients. For the error-resilient neurons, an approximation using precision modification and piecewise-linear approximation of activation function was applied to create an approximate neural network. Since training is by itself an error-healing process, after creating the approximate version, the NN is retrained. They also proposed a neuromorphic processing engine platform to determine the best tradeoff between the precision and energy.

Zhang et al. [24] used a different approach for critical neuron identification. A neuron is considered as critical, if small jitters on the neuron's computation introduce large output quality degradation; otherwise, the neuron is resilient. They presented a theoretical approach for finding the critical neurons. The least critical neurons are candidates for approximation. Due to the tight interconnection between the neurons, the ranking of candidate neurons is updated after approximation of each neuron. Hence, an iterative algorithm for the criticality ranking and approximation was developed. Three approximation strategies were used – precision scaling, memory access skipping and approximating the multiplier circuits.

Du et al. [5] proposed an inexact Neural Network accelerator showing that it is possible to use inexact multipliers in NNs. The multipliers were designed using an inexact logic minimization algorithm [11]. For small fully connected neural networks, their strategies were able to find good configurations. They exploited the fact that the output layer has a small number of neurons and since there is no synaptic weight after these neurons, lowering the errors through retraining is difficult [13].

Judd et al. [7] showed that computations and memory accesses significantly contribute to power consumption. Hence they used bit-precise weights reduction in standard multiplication and reduced memory accesses of standard memories in their implementation for GPUs and ASIC.

Power consumption of the synaptic weight memory was optimized by Srinivasan et al. [17] who applied a conventional 6T SRAM that is known to be susceptible to bit-cell failures due to voltage over-scaling. A significance driven hybrid 8T-6T SRAM was proposed wherein the sensitive MSBs are stored in robust 8T bit-cells. The memory access power reduction was exchanged for a small loss in the classification accuracy.

Sarwar et al. [16] introduced approximate multipliers based on alphabet-set multiplication. The weights were divided into parts having 4 bits. Multiplication by each 4-bit part of the weight was implemented by shifting a precomputed input value and followed by summation. Authors showed that reducing the set of precomputed values has a significant impact on power consumption and a small impact on the total accuracy of neural network. This architecture is suitable for an efficient hardware implementation because the resulting NN shows a uniform structure and each neuron has the same architecture.

In summary, the first four approaches presented in this section have shown that it is possible to approximate some neurons. The resulting NNs can be characterized as non-uniform NNs. However, for an efficient VLSI implementation and for implementing a general-purpose NN (not an application specific one), all (or almost all) neurons have to be uniform. Moreover, the selected components were approximated manually and independently of a target NN. It was also shown that not only multiplication but also the memory access has a significant impact on the total power consumption.

2.4 Approximate multipliers in NNs

Since NN contains hundreds of thousands multiplications, it seems to be useful to introduce approximate multipliers to reduce power consumption. In order to determine the impact of inexact multiplication on NNs' accuracy, the following sensitivity analysis has been carried out.

A non-trivial MLP network (1 hidden layer, 100 hidden neurons) trained for recognizing handwritten numbers of MNIST dataset (described in Section 4.2.1) was chosen as our benchmark problem and evaluated using *DeepLearn-Toolbox*.[1]. Its accurate implementation shows the classification accuracy 94.16% when precise 8-bit multipliers are used.

To emulate imprecise multiplication, a jitter function $\Delta : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ is introduced. Let the output of inexact multiplier m be defined as $m(a,b) = a \cdot b + \Delta(a,b)$. To ensure that the relative worst-case error of 8-bit multiplier m is 5.2%, the range of the jitter function Δ is bounded by ± 852, calculated as $5.2\% \cdot 2^{2 \cdot 7}$. Note that this worst-case error was chosen according to approximate multipliers proposed in [18].

When function m is used instead of accurate multiplication and no retraining is applied, the classification accuracy of the network decreased to 10.77%. A detailed analysis revealed that there are more than 80% cases where one of the input operands of multiplication is zero. The random jitter then provides a non-zero output value and this error is accumulated. Hence we hypothesized that the multiplication must be accurate if at least one of the operands is zero.

To investigate this hypothesis, we re-defined the approximate multiplier m to m', where:

$$m'(a,b) = \begin{cases} m(a,b) & \text{if } a \cdot b \neq 0 \\ 0 & \text{otherwise} \end{cases} \qquad (1)$$

Now the original NN which employs approximate multipliers m' exhibits the classification accuracy of 94.20%. Although the impact of approximate multipliers on the accuracy is application-specific, this benchmark showed that it is necessary to have the accurate multiplication by 0. Figure 1 shows the absolute difference between outputs of the same neurons in the case that approximate multipliers provide (a) inexact and (b) exact multiplication by 0.

3. PROPOSED DESIGN METHOD

The proposed method is based on uniform NNs that utilize approximate multipliers. In this section, we will define feasible approximate multipliers, describe a design space exploration search method for obtaining the feasible multipliers, and introduce the overall methodology for NN approximation.

3.1 Constraints and cost function

A digital combinational circuit with n inputs and m outputs computes a completely-specified Boolean function $\mathcal{F} : B^n \rightarrow B^m$, $B = \{0,1\}$, that maps n-input Boolean vector $x = \langle x_1, x_2, \dots x_n \rangle$ to an m-output Boolean vector $y = \langle y_1, y_2, \dots y_m \rangle$ with associated hardware cost. Let n-bit accurate multiplier be represented by a function $\mathcal{M} : B^n \times B^n \rightarrow B^{n+n}$ and let $\delta : B^m \rightarrow \mathbb{N}$ assign a natural number to an m-bit Boolean vector.

[1]https://github.com/rasmusbergpalm/DeepLearnToolbox

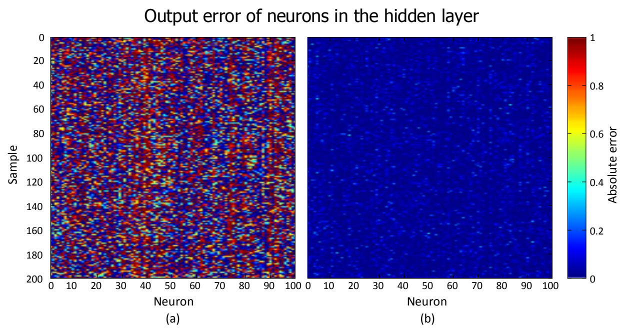


Figure 1: The error of the output neurons in the approximate NN in comparison with the original NN. The approximate NN utilizes approximate multipliers showing a 5.2% error and (a) inaccurate and (b) accurate multiplication by zero

The error metric is defined as maximal relative error ε, i.e. it is requested that the maximal arithmetic error of multiplication for each combination of operands is lower than ε on the whole output range (which is $0 \dots (2^{2n} - 1)$). This error ε will be one of the input parameters of the algorithm designing approximate multiplies.

A candidate approximate multiplier $\mathcal{M}'$ is a *feasible* solution is two conditions hold. (i) The error is acceptable:

$$\forall_{(a,b) \in B^n \times B^n} : |\delta(\mathcal{M}(a,b)) - \delta(\mathcal{M}'(a,b))| \leq \varepsilon \cdot (2^{2n} - 1). \quad (2)$$

and (ii) multiplication by 0 is accurate:

$$\forall_{a \in B^n} : \begin{array}{l} \mathcal{M}(a, \{0\}^n) = \mathcal{M}'(a, \{0\}^n) \quad \wedge \\ \mathcal{M}(\{0\}^n, a) = \mathcal{M}'(\{0\}^n, a). \end{array} \qquad (3)$$

In the approximation process, the implementation cost of multiplier $S_{\mathcal{M}'}$ will be estimated as the number of used gates. The number of two-input gates is a sufficient metric because the circuits are relatively simple (as it will be seen in Section 5). The number of used gates $S_{\mathcal{M}'}$ is determined recursively as follows: (1) the gate is used if its output is connected to output of the circuit; (2) the gate is connected if its output is connected to an input of any used gate.

The cost function for the approximation process is defined as

$$\mathbf{C}_{\mathcal{M}'} = \begin{cases} S_{\mathcal{M}'} & \text{if constraints (2) and (3) are met} \\ \infty & \text{otherwise} \end{cases} . \qquad (4)$$

3.2 Approximate multiplier design

In order to approximate an accurate multiplier, various approaches have been proposed. In this work, we employ Cartesian Genetic Programming (CGP) [12] because it can easily handle constraints given on candidate circuits, the method is naturally multi-objective and high-quality approximate circuits have already been obtained with it [18].

The standard CGP is a branch of genetic programming which represents candidate designs using directed acyclic graphs [12]. A candidate circuit is modeled using a 2D array of programmable nodes with n_c columns and n_r rows. In our case, the nodes will be 2-input Boolean functions, where Γ is the set of available functions. The circuit utilizes n_i primary inputs and n_o primary outputs. Feedback connections are not enabled.

The primary inputs and the outputs of the nodes are labeled $0, 1 \dots n_c \cdot n_r + n_i - 1$ and considered as addresses which the node inputs can be connected to. A candidate solution is represented in the so-called chromosome (which is, in fact, a netlist) by $n_r \cdot n_c$ triplets (x_1, x_2, ψ) determining for each node its function ψ ($\psi \in \Gamma$) and input connections. The last part of the chromosome contains n_o integers specifying the nodes where the primary outputs are connected to. While the chromosome size s is constant $s = n_c n_r (n_a + 1) + n_o$, the circuit size is variable and measured as the number of active (i.e. used) nodes. See an example in Fig. 2. The set of valid chromosomes (netlists) represents the whole search space.

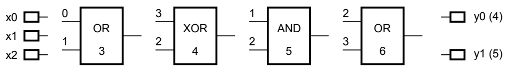


Figure 2: Example of a circuit in CGP with parameters: $n_i = 3$, $n_o = 2$, $n_c = 4$, $n_r = 1$, $\Gamma = \{0^{and}, 1^{or}, 2^{xor}\}$. Chromosome: 0, 1, $\underline{1}$; 3, 2, $\underline{2}$; 1, 2, $\underline{0}$; 2, 3, $\underline{1}$; 4, 5. Gate 6 is not used. Logic behavior of the circuit is:
$y_0 = ((x_0 \text{ or } x_1) \text{ xor } x_2)$; $y_1 = x_1 \text{ and } x_2$.

CGP employs a simple search method . In our case, the initial population P of CGP contains one of various implementations of accurate multipliers and a few circuits generated using mutation of the accurate multiplier. Creating the accurate multiplier in the initial population is trivial as there is a one-to-one mapping between multiplier netlists and CGP chromosomes. The next step consists in the evaluation of candidate circuits using the fitness function. Each member of P then receives the so-called fitness score and the highest-scored individual becomes a new parent of the next population. From this parent, λ candidate solutions are generated using mutation. The termination criterion is given by the number of iterations.

Despite many attempts to propose a suitable crossover operator to CGP, the mutation is still used as the crucial genetic operator. The mutation operator modifies up to h randomly chosen genes (integers) of the chromosome. Their new values are generated randomly, but it is checked whether the new values are valid. One mutation can affect either the gate function, gate input connection, or primary output connection.

In order to approximate multipliers, the fitness is defined as $fitness(\mathcal{M}') = -\mathbb{C}_{\mathcal{M}'}$ and $\Gamma = \{$NAND, NOR, XNOR, AND,OR, XOR, NOT, identity$\}$.

3.3 Evaluation platform

This section describes the evaluation platform used for simulations of the proposed approximate NNs. The software framework is based on C++ project *tiny-cnn*[2]. We have implemented two new types of layers to NNs: the approximate fully connected layer and approximate convolution layer. In the software simulation, the approximate multiplication was realized using a lookup table. The framework uses weights and inputs with double floating point precision. We rounded them to the fixed point representation in the range $\langle -1, 1 \rangle$. All numbers are unsigned, the sign is determined after the computation.

[2]https://github.com/nyanp/tiny-cnn

We have also synthetised multipliers for neural network. The multipliers were implemented at the Register-Transfer Level (RTL) in Verilog and mapped to the IBM 45nm technology using Synopsys Design Compiler Ultra. The hardware multiplication unit utilizes a combinational approximate unsigned multiplier circuit and logic for the sign extension. We have utilized the one's complement method which is easy to calculate ($4n$ XOR gates), but provides lower accuracy w.r.t. the two's complement method (extra three one-subtractors) used in standard applications. The framework can estimate energy consumption and area under iso-speed conditions. The clock frequency for 8 bit neurons is 3 GHz and 2.5 GHz for 12 bit neurons.

There are equal count of multiplications and additions and one activation function in the neuron computational model. Since the count of operations is big (tens or hundreds) and the multiplication consumes significantly more energy than addition, the multiplication is the most consuming part and power reduction of this part significantly contributes to the overall power consumption reduction.

3.4 Overall design methodology

Finally, the overall methodology for design of approximate multipliers that will be used in approximate NNs is presented in Figure 3. The inputs to the methodology are the accurate neural network (with accuracy J), training and testing data, quality constraint Q, accurate multiplier and initial error ε. The whole procedure is as follows. The CGP algorithm is utilized for creating a set of approximate multipliers from the accurate one. The approximate multipliers are used in the pretrained network. In order to achieve the best quality results, the network is retrained. The NN implementation showing the best accuracy K is selected. The accuracy K is checked if it meets the quality constraint Q w.r.t. accurate neural network with accuracy J. If the con-

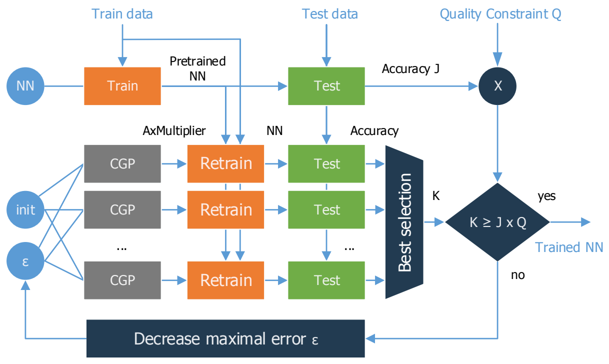


Figure 3: Overview of approximate multiplier design for approximate NNs

straint is not met, the relative maximal approximation error ε is decreased and next iteration is performed. Due to nondeterministic generation of approximate multipliers by CGP, it is necessary to generate several approximate multipliers and then re-evaluate the accuracy of NN.

4. EXPERIMENTAL SETUP

The goal of the experiments is to investigate the impact of proposed approximation methods on the accuracy and power consumption of NNs. This section provides the experimental setup and benchmark problems description.

4.1 CGP configuration

CGP will be used to design 7 bit and 11 bit unsigned multipliers. The sign extension, i.e. 8 bit and 12 bit-width multipliers, will be designed manually using the one's complement method. The maximum target arithmetic error ε of approximate multipliers is taken from the set {0.5%, 1%, 2%, 5%, 10%, 15%, 20%}. We did not employ arithmetic error beyond 20% for approximate multipliers since the classification accuracy drops significantly. The approximation process starts with accurate multipliers (Ripple Carry Array, Carry Save Array with RCA and CSA adders, Wallace Tree with RCA, CSA and CLA adders [23]) which constitute the so-called initial population. Considering two bit widths, 7 target errors and 6 types of initial multiplier architectures, there are 84 initial configurations in total.

4.2 NN Accuracy analysis

The accurate multipliers are replaced with candidate approximate multipliers in the NN which is then retrained in the supervised learning scenario. Two types of NN and two classification datasets are utilized for the accuracy analysis.

4.2.1 Handwritten numbers

The first dataset is MNIST (Mixed National Institute of Standards and Technology) database of handwritten numbers [10] which consists of two sets of data. The first one is the training data set containing 60,000 28×28 images and their labels. The second one contains 10,000 test pairs. The digits are normalized and centered in fixed-sized images. The dataset is very popular for quantifying the accuracy of classification methods. It was shown that neural networks are able to provide the error rate as low as 0.27% using convolutional networks [3]. In this case, we used a MLP network with 28×28 input neurons, 300 neurons in the hidden layer and 10 output neurons whose outputs are interpreted as the probability of each of 10 target classes $(0 - 9)$.

4.2.2 House numbers

The second dataset is SVHN (Street View House Numbers) which is obtained from house numbers in Google Street View images [15]. The images come from a significantly harder, unsolved, real-world environment. The dataset contains 73,257 digits for training and 26,032 digits for testing. Each digit is represented as a pair of 32×32 RGB image and label. While MLP does not provide good accuracy in this case, LeNet-6 (a 6 layer NN in Figure 4 [9]) is able to classify the images with a very small error. The network consumes a 32×32 grayscale image as an input. In order to reduce the complexity, we transformed original RGB images to grayscale using an equation $Y = 0.299R + 0.587G + 0.114B$.

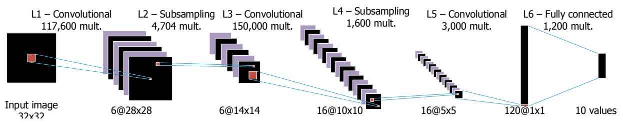

L1 – Convolutional 117,600 mult.
L2 – Subsampling 4,704 mult.
L3 – Convolutional 150,000 mult.
L4 – Subsampling 1,600 mult.
L5 – Convolutional 3,000 mult.
L6 – Fully connected 1,200 mult.
Input image 32x32
6@28x28
6@14x14
16@10x10
16@5x5
120@1x1
10 values

Figure 4: LeNet structure, where L1, L3, L5 and L6 contain approximate multipliers, i.e. 98 % of multiplications are approximated.

Layers L1, L3 and L5 perform the convolution. The L3 employs a special table that indicates which feature map from 6 previous maps is used for generating each of 16 output feature maps. Last layer (L6) connects all 120 values with each neuron of the output layer. Convolutional and fully connected layers represent 98 % of all multiplications performed in the network. Hence, the approximation was applied only for this layers. Layers L2 and L4 perform a subsampling by weighted average, but this process was not approximated because it has a small impact on power consumption.

5. RESULTS

The first part of this section is devoted to the results of the proposed CGP-based approximation of multipliers. The second part deals with approximate NNs. We also report detailed parameters of approximate multipliers.

5.1 Multiplier approximation with CGP

CGP is used with settings given in Section 3.2. Five circuits ($\lambda = 5$) are evaluated in each iteration and new circuits are created by modifying just 1 integer in the chromosome ($h = 1$) of the parent circuit. CGP operates with $n_c \times n_r = 900$ and $n_c \times n_r = 300$ (respectively) nodes for 11-bit and 7-bit multiplier (respectively). The evaluation of a candidate 11 bit approximate multiplier requires evaluation of 256x more test vectors than for the 7 bit multiplier (2^{22} vs. 2^{14}). Hence, the maximal time for CGP was set to 120 minutes for 11 bit multipliers and 30 minutes for 7 bit multipliers. CGP performed 1,343 (and 122,773) iterations on average for 11 (and 7) bit multiplier. The setting of CGP corresponds with typical values used in the literature [12, 18].

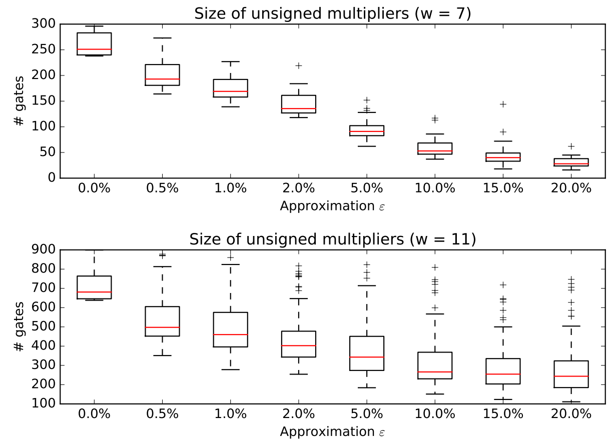


Figure 5: The number of gates in approximate multipliers

Figure 5 gives the number of gates in approximate multipliers as boxplots showing the results from 60 independent runs for a given error ε. If the error is zero only 6 values are presented which corresponds with gate counts in our accurate multipliers. In addition to obtaining many different tradeoffs between the error and the number of gates, the proposed method guarantees the exact multiplication by zero in all approximate multipliers. The spread in obtained gate counts is high especially for the 11-bit multipliers. Please note that the approximate multipliers do not prolong delay of the original accurate multipliers.

5.2 Approximate NNs

For constructing the approximate NNs, each of designed approximate multipliers was utilized. In total, we thus obtained 2×852 NNs (LeNet6 and MLP with (28×28)-100-10 layers) using 852 approximate multipliers which were subsequently extended to signed versions using the one's complement. The accuracy of approximate NNs is presented in Figure 6 for a pretrained network (column *initial*) and then for 5 and 10 retrains, respectively, using the backpropagation algorithm. Each boxplot represents 60 multipliers, e.g. in MNIST $w = 8, \varepsilon = 15\%$, there is one multiplier leading to the accuracy 20% and another to 97% in the initial placement in pretrained neural network.

Because it is infeasible to estimate power consumption of each of 852 circuits, we did a precise power analysis in the following way. We have selected circuits for each error ε and bit-width w that have one of top three best accuracies for SVHN. Then we selected a circuits from the top three sets that provide the best tradeoff between MNIST accuracy and the number of used gates. The accuracy of NNs utilizing the selected approximate multipliers is shown in Figure 7. The accuracy is normalized w.r.t. a circuit with the same bit-width and $\varepsilon = 0\%$. We have followed the alphabet-reduced

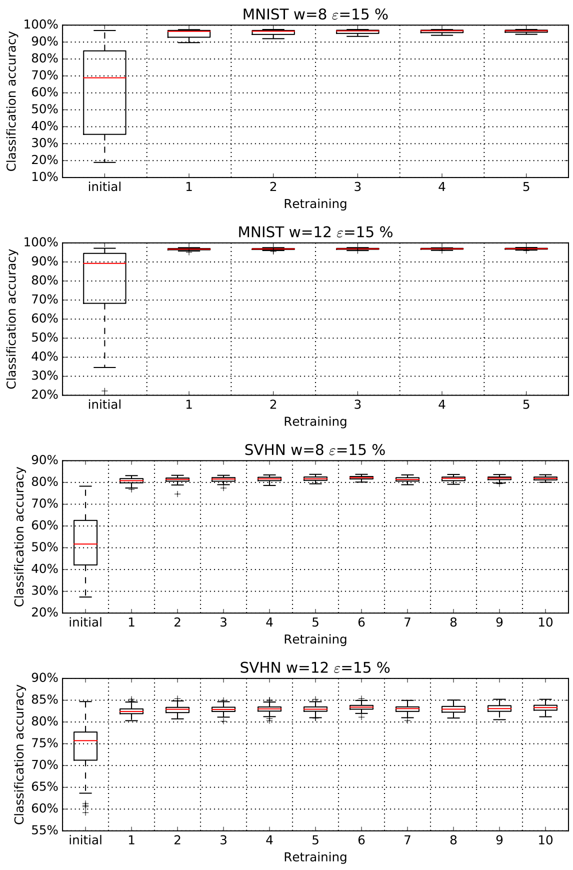


Figure 6: Accuracy of NNs for several configurations during the retraining process. The data shows statistical information for all designed multipliers with selected w and ε.

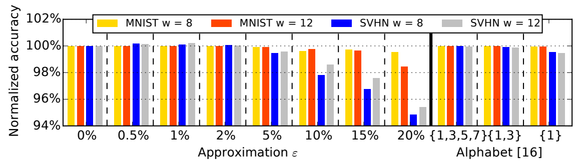


Figure 7: Normalized accuracy of NNs utilizing the best approximate multipliers developed by the proposed method for a given ε and its comparison with [16]. For each configuration, the accuracy is normalized w.r.t. NN employing accurate multipliers ($\varepsilon = 0$).

approach proposed in [16] and perform the simulation. The reduced alphabet $\{1\}$ enables to employ just 40 out of 256 weights for $w = 8$ and 200 out of 4,096 weights for $w = 12$. It can be seen that results from [16] are very similar for the proposed approach when $\varepsilon = 5\%$.

Error ε	**Power** μW	**Area** μm	**Accuracy SVHN**	**Accuracy MNIST**
0 %	250.0	440.0	87.00	97.67
0.5 %	201.0	367.7	87.15	97.66
1 %	175.0	316.6	87.08	97.68
2 %	107.0	218.3	87.07	97.65
5 %	58.6	129.9	86.54	97.58
10 %	45.2	109.5	85.11	97.31
15 %	22.3	63.2	84.20	97.42
20 %	22.9	65.8	82.52	97.22

(a)

Error ε	**Power** μW	**Area** μm	**Accuracy SVHN**	**Accuracy MNIST**
0 %	831.0	1175.0	87.04	97.70
0.5 %	417.0	664.9	87.15	97.69
1 %	475.0	720.8	87.22	97.71
2 %	284.0	523.8	87.06	97.71
5 %	247.0	483.0	86.68	97.61
10 %	125.0	285.0	85.81	97.48
15 %	115.0	262.4	84.95	97.38
20 %	111.0	252.5	83.06	96.18

(b)

Table 1: Power consumption and area of (a) 8-bit and (b) 12-bit sign-extended approximate multipliers and the absolute accuracy of NNs utilizing these multipliers.

Table 1 gives power consumption of selected approximate multipliers (in IBM 45nm process) and the accuracy of NNs that are utilizing these multipliers in two classification tasks. In comparison with the original NNs (which utilize the accurate multiplication), one can observe that approximate NN ($w = 8, \varepsilon = 10\%$) provides 81.9% power reduction of multiplication process while its accuracy decreases by 1.89% for SVHN and 0.36% for MNIST. If the error of multiplication remains below 20% the accuracy is only slightly decreased (in some cases it is even improved, but the improvement is negligible) for the MNIST problem. The NN trained for the SVHN dataset is more sensitive to approximations. The reason is that SVHN is a significantly harder classification problem than MNIST, because SVHN contains natural scene images with a high variability. However, the accuracy degradation of NN is around 1% if $\varepsilon \leq 5\%$. And finally, for example, 91% multiplier power reduction ($w = 8, \varepsilon = 15\%$) corresponds with the accuracy degradation of NN less than 2.80%.

To summarise the results, it was shown in Section 3.3 that the multiplication has a significant impact on total power consumption of calculation. When the calculation of LeNet consumes approximately 44% [7], 91% reduction of multiplication power leads to a significant total power consumption reduction of the NN.

6. CONCLUSION

This paper provided a methodology for the design of power-efficient NNs with approximate multipliers. An analysis of error resiliency of neural networks showed the feasibility of using the proposed multipliers to achieve trade-off between classification accuracy versus energy consumption. By means of CGP, approximate multipliers were designed to achieve the desired tradeoffs between the accuracy and implementation cost. Resulting approximate NNs, containing the approximate multipliers, were evaluated using standard benchmarks (MNIST dataset) and a real-world classification problem of Street-View House Numbers (SVHN). A significant improvement in power efficiency was obtained compared to the exact (or original) NNs. In some cases, 91% power reduction of multiplication was obtained with classification accuracy degradation less than 2.80% for SVHN dataset.

7. ACKNOWLEDGMENTS

This work was supported by the Czech science foundation project 14-04197S and by The Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science - LQ1602. Syed Shakib Sarwar and Kaushik Roy's research were funded in part by National Science Foundation.

8. REFERENCES

[1] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected CRFs. *CoRR*, abs/1412.7062, 2014.

[2] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *DAC '13*, pages 113:1–113:9, 2013.

[3] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Convolutional neural network committees for handwritten character classification. In *ICDAR*, 2011.

[4] K.-L. Du and M. Swamy. *Neural Networks in a Softcomputing Framework*. Springer London, 2006.

[5] Z. Du, K. Palem, A. Lingamneni, O. Temam, Y. Chen, and C. Wu. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. In *ASP-DAC '14*, 2014.

[6] G. Hinton, L. Deng, D. Yu, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov 2012.

[7] P. Judd, J. Albericio, T. H. Hetherington, T. M. Aamodt, N. D. E. Jerger, R. Urtasun, and A. Moshovos. Reduced-precision strategies for bounded memory in deep neural nets. In *HiPEAC WAPCO '16*, 2016.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *the IEEE*, 86(11):2278–2324, Nov 1998.

[10] Y. LeCun, C. Cortes, and C. J. Burges. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[11] A. Lingamneni, A. Basu, C. Enz, K. V. Palem, and C. Piguet. Improving energy gains of inexact dsp hardware through reciprocative error compensation. In *DAC '13*, 2013.

[12] J. F. Miller. *Cartesian Genetic Programming*. Springer-Verlag, 2011.

[13] S. Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4), Mar. 2016.

[14] K. Nepal, Y. Li, R. I. Bahar, and S. Reda. Abacus: A technique for automated behavioral synthesis of approximate computing circuits. In *DATE '14*, 2014.

[15] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop 2011*, 2011.

[16] S. S. Sarwar, S. Venkataramani, A. Raghunathan, and K. Roy. Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing. In *DATE '16*, pages 145–150, 2016.

[17] G. Srinivasan, P. Wijesinghe, S. S. Sarwar, A. Jaiswal, and K. Roy. Significance driven hybrid 8T-6T SRAM for energy-efficient synaptic storage in artificial neural networks. In *DATE '16*, pages 151–156, 2016.

[18] Z. Vasicek and L. Sekanina. Evolutionary approach to approximate digital circuits design. *IEEE Tr. on Evolutionary Computation*, 19(3):432–444, 2015.

[19] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan. Axnn: Energy-efficient neuromorphic systems using approximate computing. In *ISLPED '15*, 2014.

[20] S. Venkataramani, K. Roy, and A. Raghunathan. Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits. In *DATE'13*, 2013.

[21] S. Venkataramani, A. Sabne, V. J. Kozhikkottu, K. Roy, and A. Raghunathan. Salsa: systematic logic synthesis of approximate circuits. In *DAC '12*, pages 796–801.

[22] S.-C. Wang. *Interdisciplinary Computing in Java Programming*, chapter Artificial Neural Network, pages 81–100. Springer US, Boston, MA, 2003.

[23] N. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, USA, 4th edition, 2010.

[24] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu. Approxann: An approximate computing framework for artificial neural network. In *DATE '15*, 2015.