BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

POLYMORPHIC CIRCUITS SYNTHESIS AND OPTIMIZATION

SYNTÉZA A OPTIMALIZACE POLYMORFNÍCH OBVODŮ

EXTENDED ABSTRACT OF A PHD THESIS
ROZŠÍŘENÝ ABSTRAKT DISERTAČNÍ PRÁCE

AUTHOR Ing. ADAM CRHA
AUTOR PRÁCE

SUPERVISOR Doc. Ing. RICHARD RŮŽIČKA, Ph.D. MBA.
ŠKOLITEL

BRNO 2020

Abstract

This thesis deals with synthesis and optimization methods of polymorphic circuits. Ordinary and multi-functional synthesis and optimization methods are discussed. The main objective of this thesis is to introduce novel methodologies for scalable synthesis of multi-functional digital circuits. Despite the fact that several approaches have been proposed during recent years, those are applicable for small-scale circuits only or are based on various evolution-inspired techniques. Obviously, scalable synthesis methodology for complex multi-functional circuits does not exist yet. The proposed methodology is based on And-Inverter Graphs (AIGs) with built-in extension for multi-functional circuits where the employment of rewriting techniques reduces the area by sharing common resources of two different input circuits. Experiments performed on publicly available benchmark circuits demonstrate significant optimization achievements.

Abstrakt

Tato práce se zabývá metodami logické syntézy a optimalizací pro polymorfní obvody. V práci jsou jak diskutovány existující metody pro konvenční obvody, tak i představeny nové metody, aplikovatelné na polymorfní elektroniku. Hlavním přínosem práce je představení nových metod optimalizace a logické syntézy pro polymorfní obvody. Přesto, že v minulých letech byly představeny metody pro návrh polymorfních obvodů, jsou tyto metody založené na evolučních technikách nebo nejsou dobře škálovatelné. Z toho vyplývá, že stále neexistuje stabilní metodika pro návrh složitějších polymorfních obvodů. Tato práce představuje zejména reprezentaci polymorgních obvodů a metodiku pro jejich návrh založenou na And-Inverter grafech. Na polymorfní obvody reprezentované pomocí AIG je možné aplikovat známé techniky jako například přepisování [rewriting]. Nasazením techniky přepisování na polymorfní AIG získáme obvod, obsahující polymorfní prvky uvnitř obvodu, a je možné dosáhnout značných úspor prostředků, které mohou být sdíleny mezi dvěma funkcemi současně. Ověření návrhové metodiky pro polymorfní obvody bylo provedeno nad sadou veřejně dostupných obvodů, čímž je demonstrována efektivita metodiky.

Keywords

Polymorphic electronics, polymorphic circuit, logic synthesis, logic optimizations, AIG, PAIG.

Klíčová slova

Polymorfní elektronika, polymorfní obvod, syntéza číslicových obvodů, optimalizace číslicových obvodů, AIG, PAIG.

Reference

CRHA, Adam. *Polymorphic circuits
synthesis and optimization.* Brno, 2020. Extended abstract of a PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Doc. Ing. Richard Růžička, Ph.D. MBA.

Rozšířený abstrakt

Rozvoj číslicové techniky v šedesátých letech minulého století otevíral vědě nový, neprozkoumaný prostor. Nová technologie nabízela nové otázky, na které nebyly známé odpovědi a bylo nutné poznávat a pozorovat chování nových materiálů, ze kterých byla vyrobena logická hradla. S rostoucí integrací byly také kladeny požadavky na vhodné nástroje, pomocí kterých se z hradel navrhovaly složitější číslicové obvody. Nyní, po více než 60 letech existence číslicových obvodů, lze předpokládat existenci efektivních návrhových nástrojů, jejichž vývoj se stabilizoval a dnes už nedochází k tak bouřlivému rozvoji, jako v počátcích číslicové techniky.

Ano, toto tvrzení je zcela pravdivé, hovoříme-li o běžných číslicových obvodech. Avšak v roce 2001 představil A. Stoica moderní pojem "polymorfní elektronika", čímž otevřel další nepříliš prozkoumanou vědeckou oblast [51]. Jde o vícefunkční číslicové obvody, u kterých změna funkce není vyvolána přepínačem nebo rekonfigurací, jak je tomu známo u konvenční elektroniky. Namísto toho je změna funkce vyvolána uvnitř číslicového obvodu v závislosti na externím prostředí (teplota, světlo, ...) [50]. Objev polymorfní elektroniky s sebou přinesl nové technologie a otázky týkající se efektivního návrhu polymorfních obvodů.

Materiály, které dříve byly považovány za nestabilní a tudíž nepoužitelné, nacházejí uplatnění právě v polymorfní elektronice. Je možné sledovat značný pokrok ve vývoji grafenu, křemíkových nanotrubiček a organických materiálů [44] [40]. Jedná se tak o velmi mladou vědeckou disciplínu nabízející mnoho disertabilních témat.

Bohužel, konvenční návrhové metody a algoritmy nejsou dobře použitelné pro návrh polymorfních obvodů. Metody syntézy pro návrh polymorfních obvodů jsou mnohem složitější než metody syntézy konvenční elektroniky. Touto problematikou se již zabývalo několik výzkumníku, avšak dosud objevené syntézní metody nejsou natolik efektivní jako metody pro návrh konvenční elektroniky. Tato situace vyžaduje výzkum a vývoj nových, lepších a efektivnějších návrhových metod pro polymorfní obvody. Největší přínos polymorfní elektroniky je spatřován ve sdílení prostředků realizovaných funkcí v co největší možné míře. Je snahou objevovat metody, které budou generovat polymorfní obvody splňující tento předpoklad. Syntézní algoritmy pracují s obvody, nejčastěji reprezentovanými pravdivostní tabulkou, logickým výrazem, či binárním rozhodovacím diagramem. Výstupem by měla být co nejjednodušší reprezentace obvodu.

Cílem této práce je obecně představit polymorfní elektroniku a její otevřené problémy, návrhové techniky konvenčních obvodů a současné návrhové techniky polymorfních obvodů. Práce představuje tři techniky sloužící k návrhu polymorfních obvodů.

Jedním z hlavních přínosů práce je představení nové reprezentace polymorfních obvodů PAIG, díky které je možné reprezentovat polymorfní obvody v And-Inverter grafu. Na tuto novou reprezentaci je možné aplikovat již existující optimalizační metody, známé jako strukturální hashování či přepisování [rewriting], ale i další. Právě rewriting byl přizpůsoben tak, aby jej bylo možné spustit na reprezentaci PAIG za účelem optimalizace výsledného polymorfního obvodu a propagace polymorfních prvků do nitra obvodu. Práce prezentuje výsledky vykazující efektivitu metodiky a navrhuje další rozšíření.

Polymorphic circuits synthesis and optimization

Declaration

I declare that this thesis was prepared as an original author's work under the supervision of doc. Richard Růžička Ph.D., MBA. I declare that all relevant information sources used for this thesis are properly cited.

. . . .
Adam Crha
July 15, 2020

Acknowledgements

I would like to thank doc. Ing. Richard Růžička, Ph.D., MBA for his support and supervision of this thesis. Next, I would like to thank Ing. Václav Šimek for collaboration on publication activities. Big thanks belongs to my wife Tereza, who supported me, although I have been spending evenings with research instead of spending evenings with her. I'm thanking to my parents who supported me during whole Ph.D. study and also to my cousin, Luděk Bryan, who has been motivating me not only during Ph.D. study.

Contents

Chapter 1

Introduction

Reconfigurability as a phenomenon in the world of digital circuits brings more efficient ways to implement certain applications, opens new possibilities and also allows new applications of electronics. As a matter of fact, it makes hardware more flexible. Flexibility is one of the features that make software so popular as a way to implement various systems. But a wide range of applications still needs to be implemented in hardware. So the hardware reconfiguration is (and will be henceforward) very important for significant number of applications.

Typical implementation of the hardware reconfiguration consists of a field of reconfigurable elements, a controller, and memory that serves as a storage for different configurations [6]. The field of reconfigurable elements usually assumes various granularity levels - from coarse-grained elements like functional units or data processing units on RT level to transistor-level fine-grained field of elements. This allows not only the classic reconfiguration scheme (the hardware changes its structure and behavior according to the configurations prepared beforehand), but also effective implementation of so-called evolvable hardware (new configurations are being created as a direct response to actual circumstances) [34].

Another (and quite different) concept of hardware reconfiguration was proposed by Stoica et al. under the term „Polymorphic Electronics“ [51]. In this concept polymorphic circuits have a permanent structure (interconnections are fixed) and each element (or selected group of elements) of the circuit is sensitive to certain environmental factors (temperature, variation of supply voltage, etc.). Then, the function of a polymorphic circuit changes instantaneously in accordance with those specific factors. If these elements are efficiently implemented and the synthesis of the circuit is properly done, the resulting circuit will be highly efficient. Let me also note that due to the multi-functional nature of individual elements, synthesis of polymorphic circuits is much more complex than synthesis of an ordinary digital circuit.

1.1 Research motivation

It is possible to identify two main issues, which still significantly hinder more extensive adoption of polymorphic electronics as a technique for reconfigurable circuits. The first one results from a lack of suitable polymorphic components on all levels of synthesis. As the majority of polymorphic circuits have been designed on a gate level, the most-wanted polymorphic components are naturally polymorphic gates. Several useful polymorphic gates were proposed during the last decade [45] and some prospective sets of multi-functional gates are emerging even today [42]. The second issue is dealing with multi-functional circuit synthesis using those polymorphic components. As the problem of polymorphic circuit synthesis is relatively hard to address in a conventional way, many of the previously devised polymorphic circuits have been synthesized using evolutionary principles (EA, CGP etc.). Time needed to evolve a result grows dramatically with complexity of a circuit and probability of obtaining reasonable and efficient implementation drops at the same time.

1.2 Thesis organization

Brief introduction to digital circuit design is reviewed in the following section 1.3. The section explains integrated circuits design flow from system specification to physical device. The design flow is demonstrated on a well known Y-chart, where logic synthesis phase takes place. Consequently, the logic synthesis phase is discussed.

Chapter 2 describes state of the art of currently known synthesis methods for polymorphic circuits.

The main contribution of this thesis is presented in chapter 3. In the beginning, a novel, multi-level representation for polymorphic circuits is introduced. The innovative representation is an extension of AIG, in order to add capability to handle polymorphic circuits. The chapter continues with a proposal of polymorphic-AIG (PAIG) rewriting of polymorphic circuits. Its aim is to optimize a PAIG network.

Major experiments related to PAIG rewriting are described in chapter 4 in detail. The chapter consists of four sections, where the first (section 4.1) optimizes one desired circuit with polymorphic behavior. The second section (section 4.2) focuses on optimization of two independent circuits in polymorphic mode. The second experiment The third experiment (section 4.3) compares PAIG rewriting that allows KL-cuts to PAIG rewriting which permits K-cuts only. The last one (section 4.4) compares the PAIG rewriting with the most famous synthesis method PolyBDD.

Conclusion, thesis contributions and suggestions for future research are discussed in chapter 5.

1.3 Digital circuits design background

In general, an electronic device is a composition of basic electronic components such as resistors, capacitors, inductors, diodes, and transistors interconnected with wires. Interconnection of mentioned components with wires creates an electronic circuit with an ability to perform simple or complex operations such as computation, signal amplifying, data transfer, etc. Electronics can be divided into these groups: digital electronics, analog electronics and mixed electronics, which is mix of both previously mentioned [3].

Digital electronics is a subset of electronics, that operates on digital signals. A highlight of digital electronics in comparison to analog electronics is that digital signals can be transmitted without degradation caused by noise. For example, it is possible to reconstruct an audio signal transmitted as a sequence of ones and zeros without any damage, assuming that noise is not strong enough to prevent recognition of the zeros and ones in the sequence [26]

Electrical signals appearing in digital circuits are discrete and represent logic values. These values represent information that is usually further processed. In most of cases, binary logic is applied: One voltage level (typically positive value) represents logical '1', another voltage level (usually zero voltage) represents logical '0'. Digital circuits are built from logic gates (gates are built from transistors usually) and these gates offer functions of boolean algebra, such as AND, NAND, OR, NOR, XOR, XNOR etc. Combination and interconnection of these elementary gates can represent combinatorial digital circuit. [25]

Digital circuits can be divided in two groups: combinatorial and sequential circuits. Outputs of combinatorial digital circuits depend on and only on values attached to circuit inputs. Sequential digital circuits compute an output value based on values attached to circuit inputs and also on internal state of the sequential circuit. It suggests, the sequential circuits are enriched with memory, which can keep an internal state of a sequential circuit. For the purposes of this thesis, only combinatorial circuits will be discussed in the further text.

Nowadays, nearly all the computing machines are internally based on some variant of a digital circuit. From a formal point of view, its composition can be described in a straightforward way through the following definition [46] below, where its depicted as a variant of acyclic graph:

Definition 1. *Digital circuit*
Let K be a set of functional blocks (e.g. logic gates), and let G is an acyclic graph $G = (V, E)$. Then, a digital circuit is $C = (V, E, \varphi)$, where

- *V is set of nodes (I/O ports of logic gates),*
- *$E = \{(a, b)|a, b \in V\}$ is a set of edges (interconnections),*
- *φ denotes a projection that assigns to each vertex from V a component from the set K, $\varphi : V \to K$.*

The definition 1 describes a structural description of a digital circuit on logical level in Y-chart. The term Y-chart is explained a few paragraphs below. The structural description is used for the purposes of this thesis.

In its most simple valid composition, a digital circuit may consist of a single logic gate or similar fundamental element. It's necessary to point out that in a real situation a circuit would be comprised of potentially high number of mutually interconnected logic blocks (or other substantial parts for its flawless operation).

It involves a term VLSI - Very Large Scale Integration, which means a process of creating an integrated circuit by combining millions of logic gates onto a single integrated circuit. VLSI began in the 1970s when integrated circuits were widely expanded, enabling the development of complex semiconductor technologies. It is good to mention that microprocessors and memory chips are created by VLSI process. Before the introduction of VLSI technology, most integrated circuits had a limited set of functions they could perform. An electronic circuit might consist of a CPU, ROM, RAM and other glue logic. VLSI lets integrated circuits designers add all of these into one chip.

For development of integrated circuits, a Y-chart (also known as Gajski-Kuhn chart) is mostly used. Y-chart was developed in 1983 by Daniel Gajski and Robert Kuhn. The chart, visible in figure 1.1, represents the hardware development view as three domains that are depicted as three axises (behavioral, structural and physical) and looks like an Y. Along these axises, the abstraction levels describe the degree of abstraction. The outer shells are generalizations, the inner ones refinements of the same subject. The system level describes the most abstract layer, such as processors or SoC's (System-On-Chip). Register Transfer

Level describes digital circuits using registers (e.g. adder), logic level works with gates and circuit level operates with transistors.

Physical axis binds the structure to silicon. It specifies a Printed Circuit Board (PCB) layout or integrated circuit layout [41].

Behavioral axis reflects how a desired circuit should respond to a given input vector. Behavior may be specified by truth tables, Boolean equations, algorithms or any hardware description languages (HDLs) [41].

Structural axis describes how components are interconnected to perform a desired function. This representation uses a list of components and their interconnections [41].

The thesis content can be put to logic level, where gates are representatives of structural axis and Boolean expressions of behavioral axis.

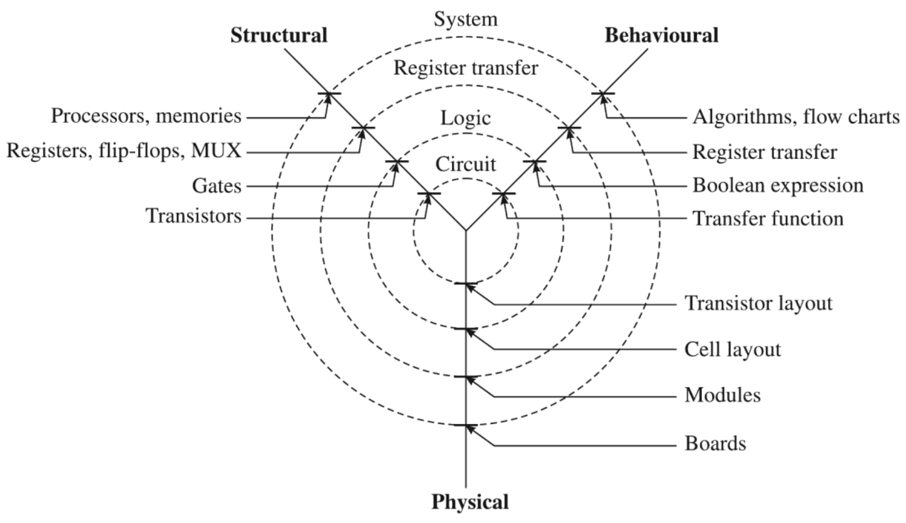

Figure 1.1: Y-chart [41].

1.3.1 Logic synthesis and optimization

On the basis given by Y-chart, a design of digital circuits is following the flow from outer shell inwards of Y-chart. To reach a systematic design of VLSI circuits, IC (Integrated circuit) design flow comes out from the Y-chart. IC design flow uses a limited set of digital logic gates (a cell library), and the process can be divided into five parts: *System specification*, *High-Level synthesis*, *Logic synthesis*, *Physical synthesis* and *Tape out*.

- *System specification* simply describes the functional and non-functional requirements posed on a system element.
- *High-Level synthesis* makes a transformation at an architectural level, transforming an algorithmic description into an RTL.
- *Logic synthesis* performs a transformation from a behavioral circuit description into a netlist of logic gates of target technology [33].
- *Physical synthesis* (also known as Low-Level Synthesis) is responsible for transformation of a netlist, obtained from logic synthesis, into a set of geometric shapes and layers to be manufactured.

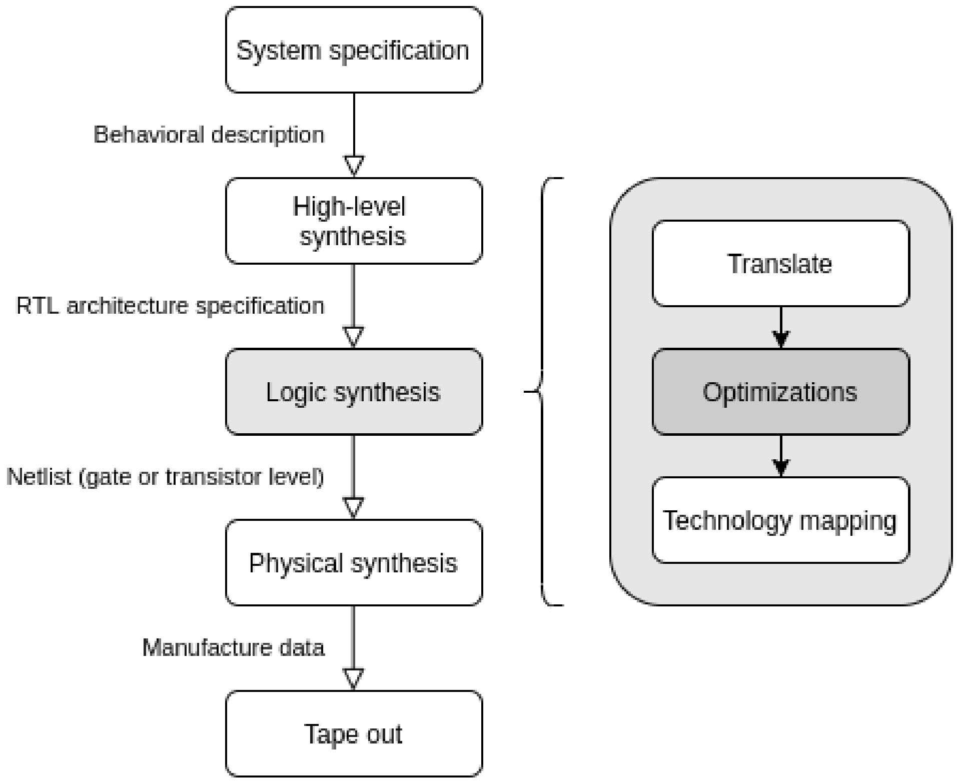

Figure 1.2: Design flow.

- *Tape out* is the final result of the design process for integrated circuits or printed circuit boards before they are sent for manufacturing.

A design flow is clearly illustrated in figure 1.2. The figure covers all mentioned abstraction layers with processes, that are applied in digital circuit design. The *logic synthesis* block is highlighted and expanded, because the block is essential for this thesis. Logic synthesis flow can be also divided into at least three parts: Translation, optimization and technology mapping:

- *Translation*
  A Register Transfer circuit description is transferred to input format of synthesis tool like PLA (Programmable Logic Array), BLIF (Berkeley Logic Interchange Format), Aiger, etc.

- *Optimization*
  An input is optimized by a synthesis tool. Synthesis tool produces an optimized result/circuit description.

- *Technology mapping*
  An optimized description is mapped onto target technology, where target technology elements are specified by a mapping library. The result is a *Netlist* on the gate level of target technology.

The term **logic synthesis** is a very important topic in EDA (Electronic Design Automation) area [27]. The logic synthesis is a transformation process of a circuit behavioral description into an optimized gate-level representation, i.e. a netlist of gates for a target technology. Main goals of logic synthesis are optimizations, typically area, delay and power optimizations, where these steps are common for two-level and multi-level representations and also for ordinary and multi-functional circuits. The logic synthesis methods attempt to minimize the number of required components, power consumption and delay of signal delivery.

Chapter 2

Polymorphic circuit synthesis and optimization

Synthesis methods of ordinary digital circuits have to solve a problem an interconnection and nodes placement of graph G, searching just for one particular function F. If a suitable canonical form[1] of F is found, a structure of G can be easily inferred from it. For polymorphic circuits, this approach tends to exhibit higher complexity. The reason is that just one graph has to cover several functions from an existing set $\phi = \{F_1, \ldots, F_n\}$, that are requested from a given circuit, and the demand of multi-functional operation has to be fulfilled in the same time. The task to find the same form for all the functions F_1 to F_n (with different elementary functions on the same position) is, therefore, not trivial at all.

Contemporary the polymorphic circuit design takes place mostly at a gate level. During the course of numerous experiments carried out within the field of polymorphic circuits it became obvious that circuits designed solely with polymorphic gates are less useful than in situation, which involves both polymorphic and conventional elements. It should be noted that the number of conventional gates typically exceeds the number of polymorphic gates of a target circuit currently. In many cases it is also sufficient to use a single polymorphic gate type, if such gate executes logically complete functions (e.g. NAND / NOR). If a wider selection of polymorphic gates is available, it could ultimately lead to better solution. However, the overall complexity of the problem could increase (in state space) [46] significantly.

2.1 Existing polymorphic design and optimization methods

Recent advances in the domain of multi-functional circuits have brought into being several approaches how to handle the synthesis process of polymorphic circuits performed at a gate level. However, all of the methods presented up to now fail to comply in some measure with the general requirements shared by common applications (e.g., resulting size, propagation delay, operating frequency, runtime duration, etc.), which is particularly apparent for demanding synthesis tasks.

[1] **Canonical representation**: Representation is unique for a particular function and variable order.

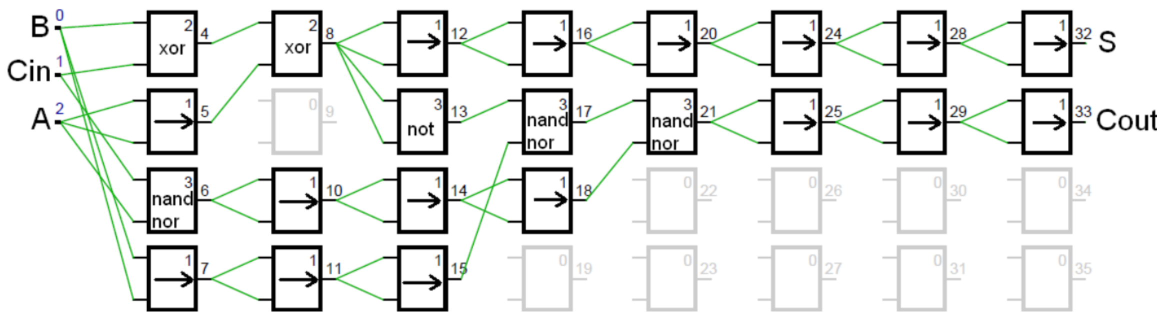

Figure 2.1: Self-checking polymorphic adder in REPOMO32 designed by CGP [48].

2.1.1 Ad - hoc: common sense design

Ad hoc approach is regarded as a circuit design method without using any explicit formal design techniques or supplementary tools. Only fundamental knowledge and an experienced designer is expected. This method allows to construct only relatively small circuits. The method is therefore not the right choice for bigger circuits [46].

2.1.2 Evolutionary design

Regardless of the existing drawbacks visited in parent section, utilization of evolutionary algorithms and techniques still plays an important role in connection with the optimization of multi-functional circuits. In fact, such methods have been a natural choice ever since the invention of polymorphic electronics. Especially in situations when certain awareness of the expected result exist, it becomes less clear how a particular objective was achieved. However, it is not an exception to obtain decent solutions.

CGP (Cartesian Genetic Programming) [35, 34] is considered to be one of several field-proven methods generating satisfactory results. The design of polymorphic circuits using CGP is almost the same compared to the conventional CGP design of circuits except the fact that it involves extended fitness function. The difference lies only in the fitness function. It is necessary to ensure that correctness of a circuit is evaluated for all functions / modes that the circuit has to perform. However, scalability becomes the major issue for really complex circuits due to possible explosion of state space, which needs to be searched [46]. Number of evolutionary-based techniques is capable of providing very efficient polymorphic circuit solutions even from scratch [47, 30]. Nevertheless, it has been found that usage of such methods makes sense for small problems only (up to 15 inputs [21]). Figure 2.1 shows an polymorphic adder designed by CGP.

Unfortunately, a process and result of various evolutionary techniques and optimization schemes derived from them is hard to predict in advance and get firmly under control. Another important problem is scalability aspect. Despite that, the evolutionary design is the most effective approach today. The proposed algorithms are capable to find many solutions, which may not be satisfying at the beginning, but the algorithm can generate successively better solutions. New solutions are derived as long as the previous ones do not achieve a perfect match with the requested functionality specified by, for example, the truth table. This approach may be conceived in a such way the fulfillment of even several parameters may be demanded.

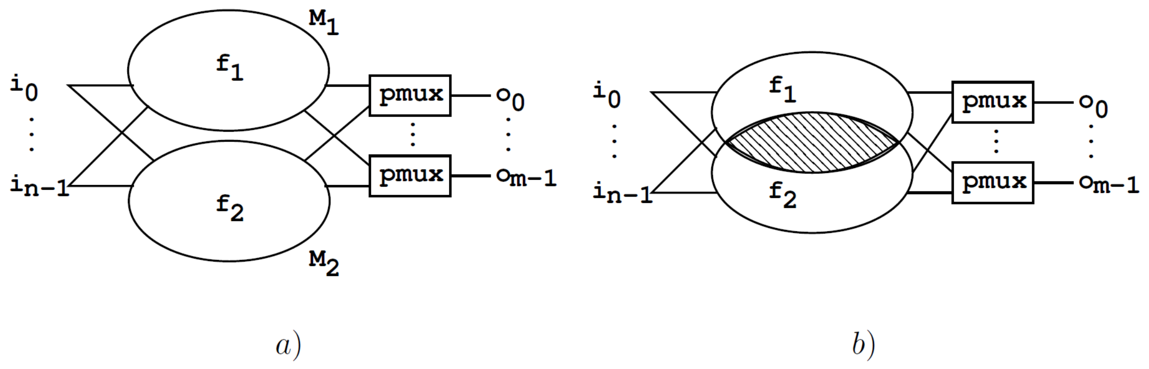

Figure 2.2: Multiplexing of conventional circuits by means of using polymorphic multiplexers: a) independent modules, and b) sharing gates between modules [20].

2.1.3 Polymorphic multiplexing

A simple and straightforward design method of multi-functional circuits is called polymorphic multiplexing - switching functions with respect to an environment state [20]. This technique was designed by Gajda and Sekanina [21]. This is a rather simple method that strives to adhere to the principles of conventional circuit design. In short, the principle is this: Every function that the polymorphic circuit will perform is designed in a conventional way using standard CMOS-based blocks. The output of each proposed circuit is connected to a so-called polymorphic multiplexer, that finally performs selection of a given input, depending on environmental conditions.

Polymorphic multiplexing can be explained formally as follows: Let's have two conventional digital circuits M_1 and M_2 implementing two different logic functions f_1 and f_2 (see a) on figure 2.2). Both circuits may be optimized using ABC tool [1] and have the same number of inputs PI and outputs PO. Outputs are connected using polymorphic multiplexers. In the first mode, primary output 0 of circuit M_1 is propagated to a common output o_0, primary output 0 of circuit M_2 is propagated to a common output o_0 in the second mode. In fact, the intended sharing of common parts is not achieved without additional steps of the synthesis process.

This initial approach is not very efficient in terms of occupied area (no sharing of similar circuit parts), which is in a direct contrast to expected benefits of using polymorphic electronics [46] [20]. A closer analysis of conventional circuits, which are supposed to be merged together using polymorphic principles, reveals the fact that it is possible to share common parts of the participating circuits. In fact, the original version of polymorphic multiplexing method can be slightly optimized with this assumption in mind. Figure 2.2 shows closer view at the principals of polymorphic multiplexing method.

2.1.4 PolyBDD

Gajda [20] suggested a method for synthesis of polymorphic circuits using binary decision diagrams (BDD). This method is called PolyBDD. It uses a concept of so-called multi-terminal BDD, which is an extension of binary decision diagrams with the terminal nodes of the diagram containing integer values. The PolyBDD method is using these values to represent a possible relation between input variables and a relevant output. In case of polymorphic circuit expected to implement two different functions working in two allowable

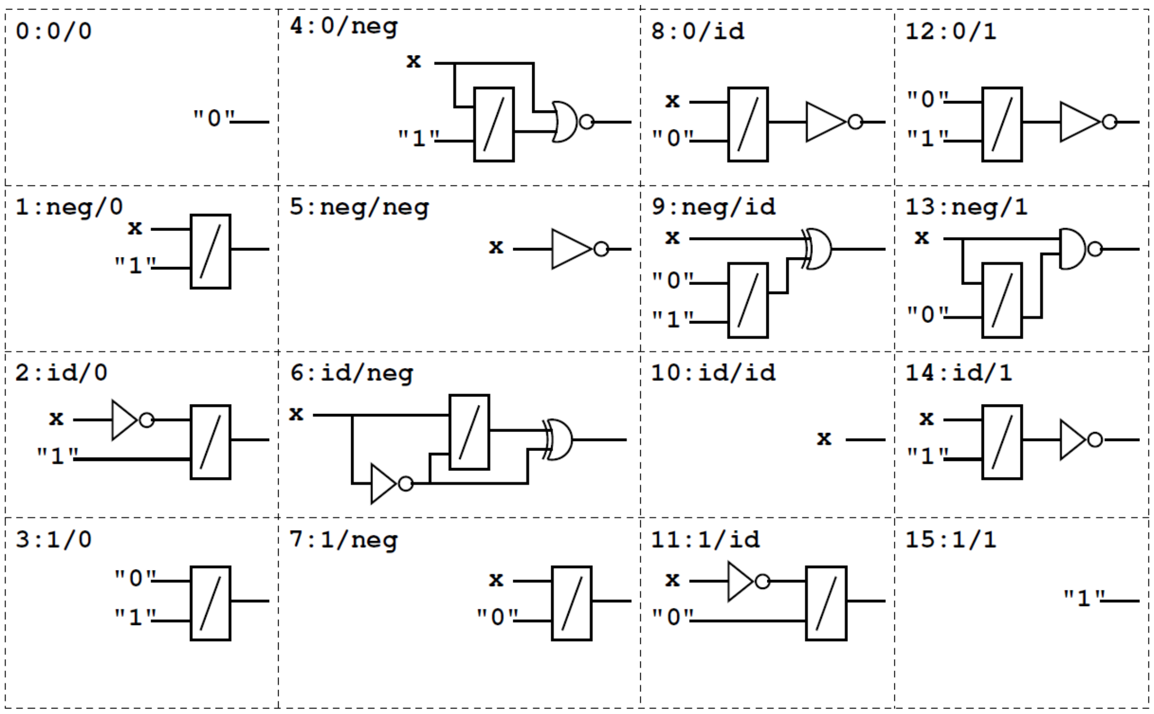

Figure 2.3: Conversion table for the transformation procedure of PolyBDD into a polymorphic circuit [20].

modes while using polymorphic gates, it yields 16 possible combinations. Values in terminal nodes of MTBDD tree will be therefore integers from an interval <0-15> (figure 2.3). Detailed explanation of internal principles of PolyBDD method can be found in [20] and [21]. Principal drawbacks of this method lie in relatively sparse exploitation of polymorphic gates, i.e. these are practically used only in a role of input/output switches. Further optimizations relies on evolutionary optimization of polymorphic circuit.

2.1.5 Recent work on logic synthesis of polymorphic circuits

Some further work on polymorphic circuit logic synthesis has been done in recent years. This section summaries interesting published papers related to polymorphic logic synthesis topic.

Evolutionary design of polymorphic circuits with the improved evolutionary repair [58]

In 2013, a Chinese research team tried to improve evolutionary synthesis of polymorphic circuits by including *Repair algorithm* into evolutionary synthesis process [58]. Evolutionary synthesis algorithms are the most usable techniques for design of polymorphic circuits, but they still face scalability problems. The most complex polymorphic circuit designed evolutionary is a sorter/multiplier with 6 primary inputs and 6 primary outputs. Thus, designing more complex polymorphic circuits is still the biggest obstacle in the case of use of evolutionary algorithms.

The team published the *Repair algorithm* for evolutionary synthesis algorithms in order to accelerate the evolution process and overcome *Stall effect* in 2012 [57]. *Stall effect* is a state when the best fitness of the evolutionary population is not increasing or increases slowly. It may take several generations to find a population that will have progressive fitness again. The repair algorithm is deployed when stalling state is detected. It repairs incomplete, best individual to a target circuit directly. This repair technique removes time wasted in stall effect.

Authors applied the repair algorithm onto evolutionary design of polymorphic circuits in order to make evolutionary techniques more scalable.

Evolutionary Design of Polymorphic Circuits with Weighted Sum [30]

Sekanina et. al. experimentally applied a CGP design of logic circuits. The fitness is computed for the whole polymorphic circuit without respect to complexity of each desired function [47]. A Chinese research team extended CGP evolutionary approach applying weighted sum, that helps to increase the success ratio and decrease the evolutionary generations in 2007 [30]. Without loss of generality, authors expect that a polymorphic circuit can perform two independent logic functions in two independent modes. Both circuit (performing function 1 or function 2) can be regarded as a traditional digital circuit. Circuits in mode 1 or mode 2 may have different complexity. Thus, a characteristics of polymorphic circuits (to be evolved) should be taken into account for generating the expected polymorphic circuit. Authors involve weighted sums W_1 and W_2 in computation of fitness function of desired polymorphic circuit in the following form: $F = W_1 * F_1 + W_2 * F_2$, where W_1 W_2 are weight coefficients, and F_1 F_2 are common fitness for circuits in desired modes. For detailed description, see experiments presented in [30]. Unfortunately, experiments are presented on quite small circuits only.

In 2015, the team extended the weighted sum approach with periodical weight adjustment, changing weight factor are changed periodically according to the sinusoid function, expressed as: $W_i = sin(2\pi t_i) + 1$, where $1 \leq i \leq n$. Authors present improvement in comparison to initial weighted sums work, thus, experiments are presented on small circuits only. For more details, visit [31].

Design Methods for Polymorphic Combinational Logic Circuits based on the Bi-Decomposition Approach [29]

The newest paper related to design and optimization of polymorphic circuits is dealing with Bi-Decomposition approach [29]. Authors apply Bi-Decomposition, known from traditional design of logic circuits [49, 39]. The work promises an algorithm for design of relatively large circuits with gate-efficient implementation and utilization of polymorphic gates in contrast to PolyBDD, where the lesser utilization of polymorphic gates is criticized. Experiments report number of used gates and utilization of polymorphic gates. This paper looks very promising, however, it is currently available on arXiv[2] only without admitting reviews.

[2] arXiv is a free distribution service and an open archive for scholarly articles.

Chapter 3

Proposed multi-level design and optimization method

In fact, a specification of logic function itself can appear in several, mutually different forms. An elaborate discussion on five of the most common description using two-level arrangement can be found in [55]. These cases are mostly focused on various representations of truth table forms together with disjunctive/conjunctive notation. For the sake of completeness it is important to point out that synthesis and minimization techniques in digital circuit domain are based extensively on multi-level representations as well [17, 23, 24], especially due to reasonable compromise between compact representation and efficient manipulation. Probably one of the most illustrative examples here is tied with AIG as a widely adopted scheme in logic optimizations.

Those minimization and synthesis techniques could be, as a matter of fact, roughly classified as two-level or multi-level oriented. In case of two-level methods the final circuit composition is delivered as a logic expressions in conjunctive or disjunctive notation. This approach then leads to the situation when input signals will only pass through two logic gates at most. On the other hand, multi-level techniques are generating so called nested expressions with the resulting data path (or interconnection of the individual gates) spanning even far more than two circuit elements within the final circuit arrangement.

The previous chapter discussed two-level optimizations of polymorphic circuits. Presented optimization techniques are applicable to two-level circuit representations. Literals are inputs and it is assumed that a final circuit is represented in the same way as a represented function. It leads to significant number of multi-input AND gates and one big OR gate. This fact may be a motivation to focus multi-level optimization methods. Multi-level representations are more realistic in the most cases.

In order to develop a method for multi-level logic optimization, a valid multi-level representation of logic circuit for an optimization algorithm is necessary. Multi-level circuit representations for ordinary logic circuit already exist, such as BDDs, AIGs or factored forms. It is assumed that a node can perform an arbitrary function and a number of literals can be significantly reduced. Unfortunately, similar descriptions for polymorphic circuits are missing.

It is a motivation to propose a multi-level representation of polymorphic circuits. This representation shall prepare a basis for optimization processes. Last years, AIG's are very popular representation useful for optimizations of ordinary circuits. Popularity of AIG's has led to a focus the AIG representation and hence PAIG (Polymorphic And-Inverter Graph), which is described in following section, was introduced.

3.1 PAIG - An extension of AIG for polymorphic circuits

The problem outlined above, which is being pursued by this thesis, represents a challenge that stands out particularly in a situation when the circuit complexity is reaching beyond the boundary of more than just a few tens of gates. A key obstacle here is given by the fact that standard methods for representation of a typical digital circuit fail to adequately capture the specifics within the polymorphic electronics domain, which in turn renders their performance quite unsatisfactory. However, a solution leading ultimately to an inception of a novel format for representation of polymorphic circuits. The novel format could be identified in integration of corresponding extension into the foundation of conventional schemes like e.g. And-Inverter Graphs [37] or Binary Decision Diagrams [2].

Hence, the need to design a novel format for transparent representation of polymorphic circuits, which satisfies the requirements of easy manipulation in terms of synthesis and optimization tasks, clearly emanates from those aspects.

3.1.1 Elements of And-Inverter Graph scheme

One of the most ubiquitous schemes used for representation of a conventional digital circuit is known as And-Inverter graph (AIG). In fact, its core principle is based on an acyclic network of nodes and edges, where a node is two-input AND gate and an edge behaves like a negation of the logic signal passing through it between nodes. A significant advantage of using AIG concept for representation of a digital circuits is undoubtedly given by the fact that its foundation is relying on well-established graph theory. Hence this observation suggests a possibility to apply various operations that are generally known from graph theory also in case of AIGs. The continuous exploration and refinement of AIG with regard to its origins of theoretical background, which has been done already for more than a decade, brought a couple of advanced operations, which turn out to be very useful for digital circuits optimization.

Most of these operations have originated from research activities of Alan Mischenko [37]. For the sake of clarity, let's mention just some of them:

- Balancing: reduction of the depth.
- Structural hashing: detection of an isomorphic subgraphs.
- Functional reduction: detection of an isomorphic subfunctions in a graph.
- Rewriting: identification of ineffective parts and their replacement.
- and a rich set of additional operations.

And-Inverter graph offers modern, effective and transparent representation for necessary minimization operations requested by logic synthesis. For this reason, I decided to use AIG 's as a base for a new unique representation format of polymorphic circuits.

3.1.2 Toolset for operations with AIGs

The term AIGER denotes a format and, in the same time, also set of utilities for And-Inverter Graphs processing, which was developed at Johannes Kepler University in Linz. AIGER has been presented to the general audience at the Alpine Verification Meeting 2006 in Ascona [4]. The main idea was to provide a simple, compact file format for a model checking purposes. In fact, a specification of AIGER format [5] is available in two, slightly different versions: an ASCII and a binary. Each version is conceived with the aim to accommodate somewhat different purpose.

The ASCII version is the format of choice when it comes to AIG circuit representation, which needs to be saved for further reading and editing by a human circuit designer. It is simple to generate and it less constrained in comparison to the binary format. The binary version is, in fact, a compressed version of ASCII variant. Binary format saves data and is unreadable without corresponding AIGER reader. ASCII format requires more disk space, but it is readable by human.

Further within the context of this contribution the ASCII format will be considered only, especially because it gives better means for explaining the fundamentals of novel approach to the representation of polymorphic circuits.

Every circuit description file compliant with AIGER format should begin with one-line header, where the exact version of AIGER is given:

- „aig“ - binary identifier
- „aag“ - ascii identifier

This identifier is followed by 5 unsigned integers M I L O A which denote the following items respectively:

- M - maximum variable index
- I - number of inputs
- L - number of latches
- O - number of outputs
- A - number of ands

After this sequence of integer identifiers, the file format simply continues with an AIG description of a circuit structure. Now, each object (input, output, and, latch) has a unique numeric identifier. In order to keep a tidy arrangement, inputs are described first - each input (unique numeric identifier) is specified on a dedicated line:

```
<input identifier>
```

After the specification of all relevant input, latches can be placed. However, latches are not supported at the moment for polymorphic circuits, because this work focuses to combinatorial circuits only:

```
<latch input> <latch output>
```

Outputs must be specified in the same way as inputs:

```
<output identifier>
```

Finally, there still remains the need to specify the inner portion of AIG circuit representation consisting of AND-based nodes and their mutual links based on inverters:

```
<node identifier> <left leaf> <right leaf>
```

The following box contains an example of AIGER format in ASCII encoding [5]. The example is based on the previous instructions:

```
aag 6 2 0 2 2
2 #input 0
4 #input 1
8  #output 0
7  #output 1
6 2 4  #and node
8 3 5  #and node
```

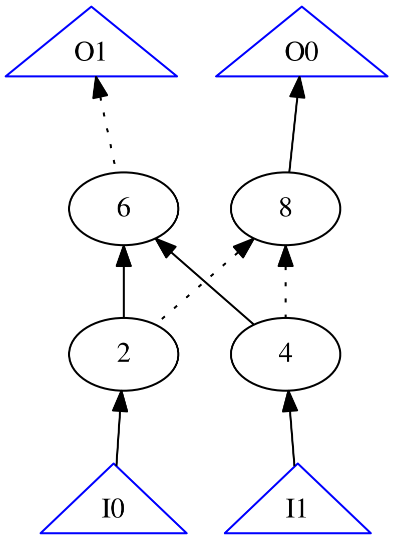

Figure 3.1: Representation of circuitry that contains two combinatorial functions, NAND - output O1 and NOR - output O2, using AIG paradigm.

It is possible to notice on figure 3.1 and example of AIG description given above, all object identifiers are **even**. It is required by internal implementation of AIGER interconnections, where even number is a wire. An inverting edge (dotted) is specified by **odd** object identifier (+1). For example an inverting edge from node 8 to node 2 will be noted as follows[1]:

```
8 3
```

3.1.3 Newly proposed AIG format for polymorphic circuits: PAIG

As it was concisely demonstrated in the previous section, AIGER and other widespread conventional tools and techniques do not offer, if any at all, an immediate support for the representation of polymorphic circuits. Hence, the need to design a novel format for

[1]Graphical representation of edges differs from commonly used notation. Arrows lead from leafs to roots.

transparent representation of polymorphic circuits, which satisfies the requirements of easy manipulation in terms of synthesis and optimization tasks.

Technical details of PAIG

The intention to develop new format for representation of polymorphic circuits was primarily motivated by the constraints and severe limitations of the available conventional methods. In addition, there was also an objective to get a compact format which may facilitate the tasks of synthesis and further optimization of circuit structure. A key decision was to preserve backward compatibility to AIGER format due to its widespread acceptance and relative simplicity.

With the aim to keep the complex nature of polymorphic circuit synthesis at a reasonable level, a number of permissible modes for each node within AIG representation was limited to the number of two. It means that the final polymorphic circuit can ultimately work in just two different operating modes, where an actual mode is switched by a state of the environment. In general, a resulting behaviour of the circuit built in AIG can be affected only by two aspects:

1. Interconnection.
2. Edges (wire or inverter).

One of the possible ways how to enhance the capabilities of AIGs involves definition of new edge types. Thanks to the polymorphism it is possible to change behaviour of gates and also inverters. AIG contains only AND gates, however, any other function can be built from AND gates and their appropriate interconnection. This idea has resulted into the extended variety of edge types - from the initial two types to total of four types now:

1. Normal edge - wire.
2. Inverted edge - inverter.
3. Polymorphic edge 1:
   - In mode 1 - wire,
   - in mode 2 - inverter.
4. Polymorphic edge 2:
   - In mode 1 - inverter,
   - in mode 2 - wire.

Black solid line represents a normal wire. Black dotted line represents an inverter. New polymorphic edges are denoted as a double solid line in the case of polymorphic edge 1 and as dotted double line in the case of polymorphic edge 2. See figure 3.2 for examples illustrating those types of edges. Finally, those four types of edges enable design arbitrary polymorphic circuit with two operating modes.

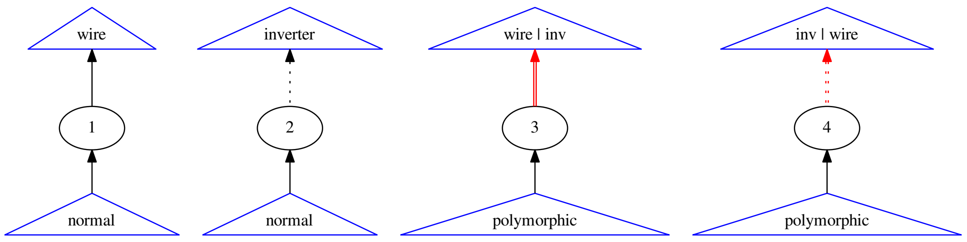

Figure 3.2: Graphical representation of edge types. Edge types from the left: wire, inverter, polymorphic wire, polymorphic inverter. Polymorphic wire is working as a normal wire in mode 1, while in mode 2 it assumes the function of inverter. Polymorphic inverter of is shown on the right side. In mode 1 it provides the functionality of inverter. Then, in mode 2, its behavior resembles wire.

At the beginning it is necessary to inform an AIGER parser about an intention to use the extended format. Format identifier in header must contain string „paag“. Then, the extension for polymorphic circuits will be correctly recognized:

- „paig“ - binary identifier for polymorphic AIGER (not supported yet),
- „paag“ - ascii identifier for polymorphic AIGER.

The ordinary AIGER format works with unsigned integers only. Even reference indexes are treated as „wires“ and odd are being seen as „inverters“. Extending AIGER to work with **signed integers** is necessary for the support of new edge types - polymorphic edges. Ordinary edges are staying unchanged, while the polymorphic edges have negative prefix before their object index. Following example highlights the proposed extension:

- Polymorphic edge 1 (mode 1 = wire, mode 2 = inverter) will be noted as **negative even** integer.
- Polymorphic edge 2 (mode 1 = inverter, mode 2 = wire) will be noted as **negative odd** integer.

```
paag 4 4 0 4 0
2 #input 0
4 #input 1
6 #input 2
8 #input 3
2 #output 0
5 #output 1
-6 #output 2
-9 #output 3
```

3.1.4 New constructions offered by PAIG extension

AIG extension of polymorphic edges has brought new constructions that may appear in a network. These constructions may bring new, more effective interconnections, but in other hand they require an attention during AIG manipulation.

One of the new construction that does not make sense in AIG, but in PAIG has significant importance, is connection of both edges from node a A to a node B. This construction has no meaning in AIG, because it propagates constant value, but in PAIG, it may change output dependently on polymorphic state. It is possible to insert a polymorphism into a circuit easily and this construction is also applicable inside a network. See figures 3.3 for better understanding.

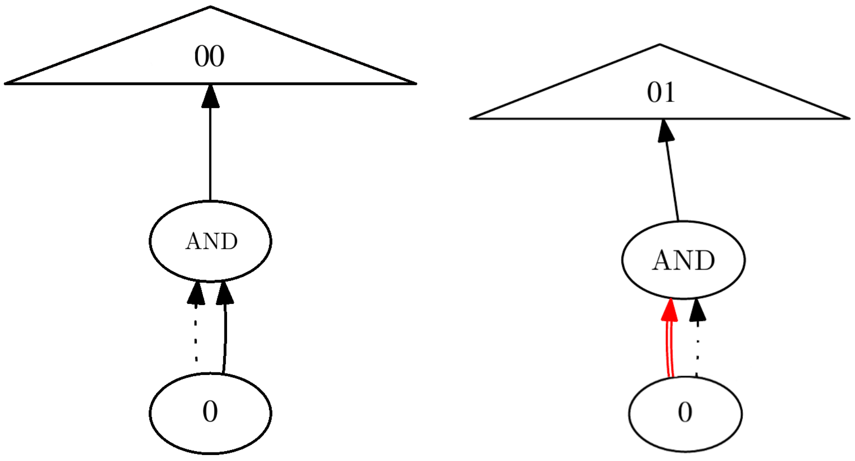

Figure 3.3: The left network shows an interconnection useless in AIG. However, the right network shows the same interconnection with polymorphic edge, that has significant usability in proposed PAIG networks.

This construction is further used for transformation of random primary input to virtual polymorphic input in performed experiments and evaluation. These constructions can appear inside a PAIG network also.

3.1.5 Experiments and demonstration

no.	Description		
#	Description	Circuit 1	Circuit 2
1	2-bit ALU	Logic ALU	Arithmetic ALU
2	2-bit Adder	SUM	Carry
3	Cellular tr.function	Rule 30	Rule 100
4	GRAY/BCD Coder	Gray	BCD
5	Self-checking adder	Carry	Carry

Table 3.1: Description of experimental circuits for PAIG extensions.

For the purpose of demonstrating properties of the newly proposed format for polymorphic circuits representation five different experiments were prepared in total (table 3.1). These experiments clearly show the efficiency of polymorphic circuits handling using new PAIG/PAAG format in comparison to the conventional solution based on standard AIGER format without additional modifications. Selection of benchmark circuits was random in order to clearly demonstrate the concept of PAIG representation.

no.	Conventional solution			Polymorphic solution		Improvement	
#	Circuit 1	Circuit 2	SUM	AIG	PAIG	Conv. vs PAIG.	AIG vs PAIG
	[ANDs]	[ANDs]	[ANDs]	[ANDs]	[ANDs]	[%]	[%]
1	9	7	16	18	10	44.44	37.50
2	4	6	10	13	7	46.15	30.00
3	4	4	8	13	6	53.85	25.00
4	16	7	23	26	16	38.46	30.43
5	4	4	8	8	4	50.00	50.00

Table 3.2: Comparison results of conventional AIG representations and PAIG representations.

Experiment no.1: Logic/Arithmetic ALU

The first experiment is dealing with a combination of two different ALU structures. The first ALU should be working in a logic mode and the second ALU should be working in an arithmetic mode. Conventional AIG implementation of these two ALUs is shown in figure 3.4. In this case, the logic operation mode of the ALU requires 9 AND gates and the arithmetic ALU consumes only 7 AND gates. See table 3.2 stating the overall number of used gates.

If an addition of polymorphism feature (switching of operating modes) is conceived using conventional AIG, then a mode switching is achieved thanks to an additional, dedicated virtual input labelled Mode. Let's call this solution „virtual polymorphism“. Virtual polymorphism of ALU requires 18 AND gates. It is possible to see a graphical representation of this circuit variant in figure 3.5 on the left.

If the PAIG/PAAG format is used, it is possible to reach significant savings of resources. See figure 3.5 on the right. It shows a polymorphic ALU working in both logic and arithmetic mode, in which an operating mode is depends on a state of a target operating environment. This implementation is built upon the exploitation of new polymorphic edge types.

Experiment no.2: 2-bit adder

The second experiment is comprising a polymorphic adder, where Carry and Sum bits are switched in polymorphic way. In the first mode, adder calculates SUM, in the second mode, adder calculates carry. An adder circuit representation using conventional AIG scheme requires 10 AND gates. If the PAIG novel format is used, common resources are shared and whole the adder implementation requires only 7 AND gates. See table 3.2 for details.

Experiment no.3: Cellular transition functions

Another application of polymorphic circuits is demonstrated as a transition function of cellular automaton. This experiment executes two transition functions: Rule 30 and Rule 110. Rules are switched with regard to the operating conditions. Two conventional functions build from AIGs require 8 gates in total. Polymorphic AIG solution saves 2 gates while still preserving the original functionality. See table 3.2 for details.

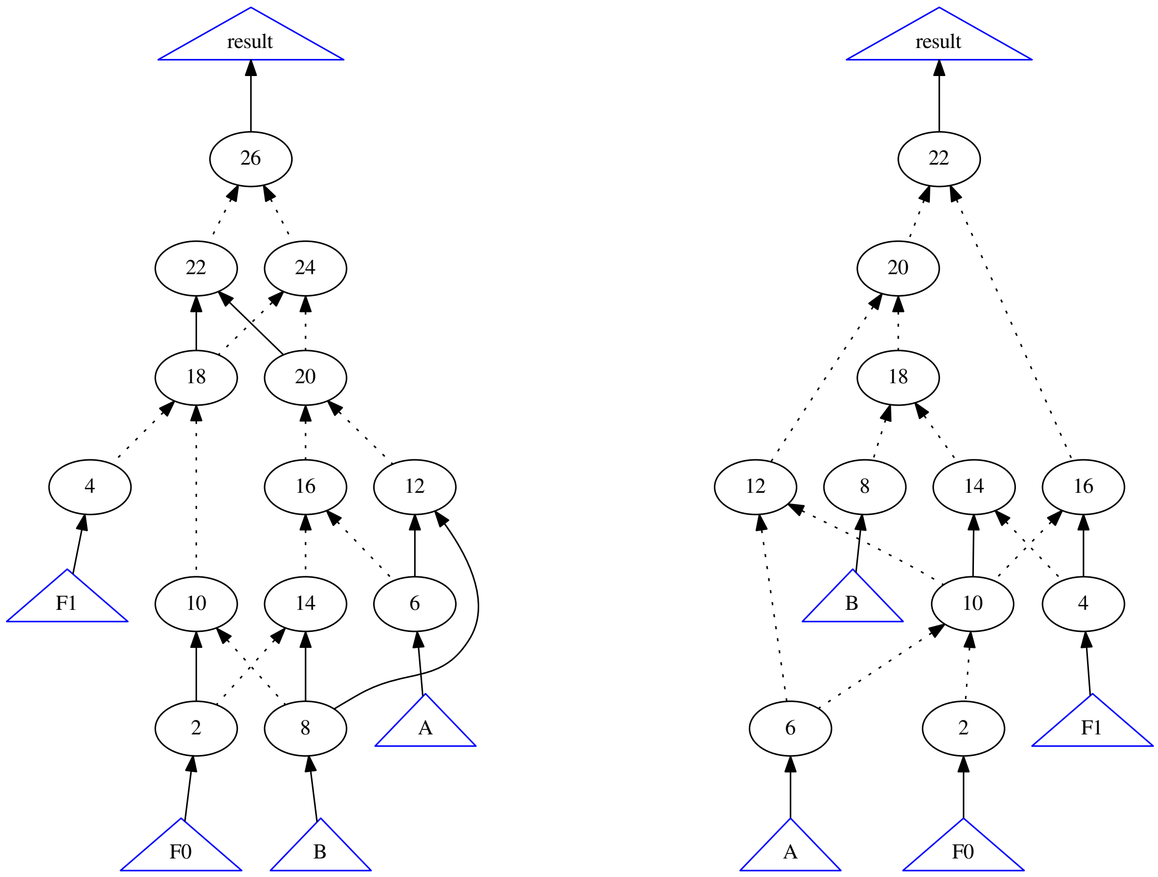

Figure 3.4: There is a conventional AIG of logic ALU on the left. There is a conventional AIG of arithmetic ALU on the right.

Experiment no.4: Gray/BCD decoder

This experiment combines two 4-input and 4-output circuits - Gray coder and BCD coder. Polymorphism switches between gray coding and BCD coding. A conventional solution of both circuits requires 23 AND gates in total. Virtual polymorphism needs 26 AND gates and a PAIG solution consumes only 16 AND gates. An improvement of the polymorphic solution reaches 38.46% in comparison to the conventional solution.

Experiment no.5: 2-Bit self-checking adder

The last experiment takes aim on a special kind of 2-bit adder with self-checking ability. This idea originated at Faculty of Information Technology, Brno University of Technology [45] as an illustrative example of polymorphic circuits application. Adder works equally in both polymorphic modes. An additional feature is hidden in fault detection mechanism. The adder calculates carry in the first mode. In the second mode, the adder performs the same - carry. If the carry is equal in both modes, no fault is present. But, if the carry is different in operational modes with respect the same inputs, an fault is signalized.

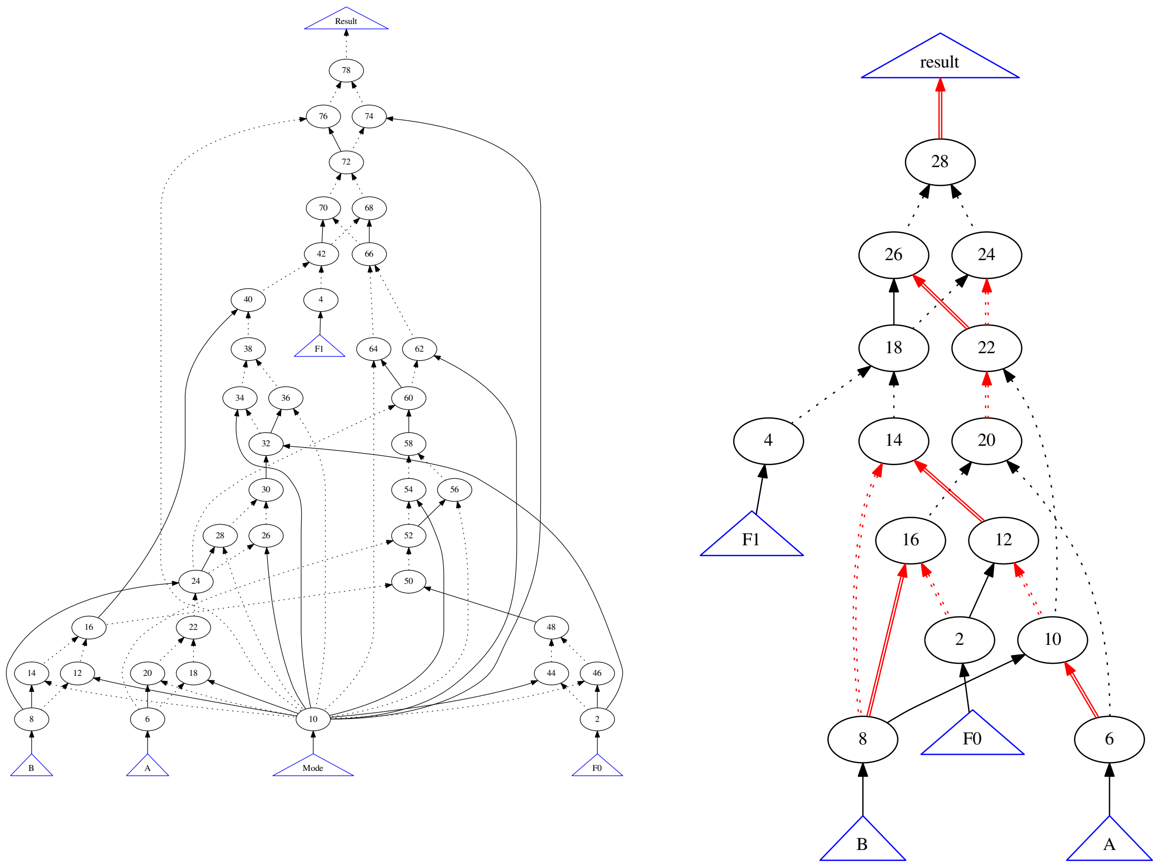

Figure 3.5: There is a conventional AIG of virtual polymorphic logic/arithmetic ALU on the left, in which the mode is controlled by virtual input labeled as MODE. There is a polymorphic AIG (PAIG) version of logic/arithmetic ALU on the right. Is possible to note significant savings of AND gates.

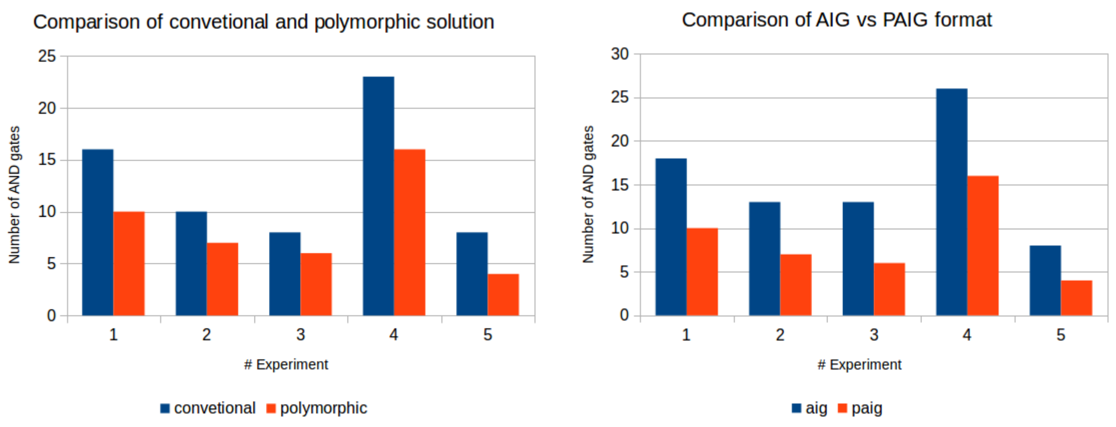

Figure 3.6: Left graph shows a comparison between two circuits synthesized as two separate circuits and polymorphic solution using PAIG. Right graph shows a comparison between AIG and PAIG, where both are representing a polymorphic circuit.

PAIG extension evaluation

The table 3.2 summarizes results of all conducted experiments outlined in the previous section. The table is separated into two main columns: Conventional solution and polymorphic solution. The conventional solution contains number of AND gates used with conventional technology and AIG representation. The polymorphic solution contains number of ANDs required by polymorphic technology. An AIG column contains number of ANDs required in case of using virtual polymorphism (basic AIG representation). A column PAIG contains number of used ANDs in PAIG/PAAG representation. The third column shows an improvement between conventional solution vs polymorphic solution in a percentage ratio and also the comparison of virtualized polymorphism to PAIG/PAAG representation. The results demonstrate, that the proposed PAIG/PAAG representation scheme can provide average saving of 34.59% AND gates (see figure 3.6 for details).

This evaluation presents the novel PAIG/PAAG representation on relatively small circuits. A main intention of the evaluation is clear explanation of PAIG/PAAG representation principles. More complex circuits are evaluated later, in chapter 4.

3.1.6 PAIG extension summary

A novel format for representation of polymorphic circuits using AIG was proposed. It is an extension of AIGER format, which is fully supported in well known tools like ABC and others. A few experiments show that the novel format seems to be very effective approach how to represent polymorphic circuits, including very complex variants (complex variants discussed in chapter 4).

3.2 Polymorphic AIG Rewriting

In comparison to conventional circuit design, designing of polymorphic circuit is much more complex. It is mainly caused by an ability to change behavior of building elements, while interconnection remains the same. Many designing methods have been proposed (see section 2.1), but most of them are two-level, or they have a scalability problem. Two-level polymorphic design methods were already published [12, 11], but a scalable, multi-level and straight methodology is still missing. Considering the AIG as a very popular concept for conventional circuit design, I decided to create a polymorphic circuit design methodology based on AIG.

In order to synthesize and optimize multi-functional circuits by means of using a scalable methodology based upon a formal foundation, a polymorphic-AIG (PAIG) rewriting is proposed as a modification of original AIG Rewriting [37] and PAIG extension. Original AIG Rewriting is described in [37] in detail. However, it is not applicable for synthesis and optimization of polymorphic circuits without further modifications of the former algorithm.

Basic idea of the original rewriting technique is based on replacement of AIG sub-graphs by optimal, smaller sub-graphs in order to reduce total number of gates. The algorithm proceeds iteratively from leaves to roots of an AIG and if a better solution is found, replacement is applied. Sub-graphs which are investigated have typically 4-inputs (K-feasibility K=4) and L outputs (L=1).

PAIG-based rewriting approach uses rewriting technique with a slightly different idea in mind. As it was mentioned in previous subsections, main intention of multi-functional circuits is to take an advantage of sharing common logic of two completely different functions. At first, it is necessary to insert polymorphism (multi-functionality) into a circuit that is going to be synthesized. Two options open up here. The first option, an usual procedure implements two different functions in a conventional manner, so the polymorphic multiplexing is used as a seed of polymorphism. Polymorphic multiplexers are connected to primary outputs with respect to functions that are being switched. In PAIG, I designed two variants of polymorphic multiplexer, which are depicted in figure 3.7. Let's note that both variants consist of 3 AND gates. Adding polymorphic multiplexers creates full-fledged

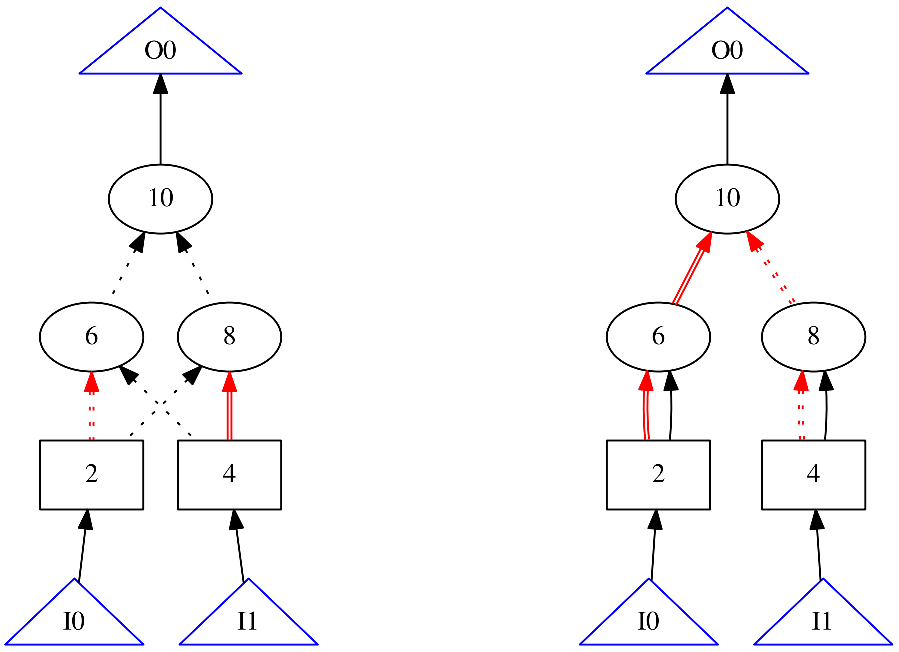

Figure 3.7: Two variants of polymorphic multiplexer represented in PAIG network.

polymorphic circuit with two functions. The second option is conversion of conventional circuit into polymorphic circuit by removing one primary input and making the primary input polymorphically driven.

However, sharing of common resources is not reached in either options, in contrary to the main objective here. So here comes the right moment for unleashing PAIG rewriting algorithm for optimization of polymorphic circuit in order to share common logic resources. In comparison to the original AIG rewriting, a few modification have been applied, such as support of PAIG representation, modified cut enumeration and generating optimal subcircuits. All these modifications are described in detail in the following sections.

3.2.1 PAIG rewriting algorithm

For optimization and synthesis of polymorphic circuits, a rewriting algorithm has been chosen for his popularity, quality and efficiency in conventional synthesis. Rewriting algorithm is well described in [37], where it was introduced. In the recap, rewriting technique goes through all nodes in a network and all cuts of each node. Then, each cut is analyzed and replaced by its optimal implementation (note that all optimum 222 NPN-classes are pre-computed). The cut having the best gain of nodes is replaced.

Algorithm 1 PAIG rewriting algorithm

```
1: procedure REWRITE($AIG, use\_zero\_gain$)
2:     for each node N in AIG in the reverse topological order do
3:         $C_{best}$ = NULL;
4:         gain_max = 0;
5:         for each 4-input poly cut C of node N do
6:             F = simulate_cut(C);
7:             $C_{optimal}$ = Generate_optimal_subgraph(F);
8:             $AIG_{sand}$ = copy(AIG);
9:             replace_cut($AIG_{sand}$, $C_{optimal}$);
10:            $AIG_{sand}$ = structural_hashing($AIG_{sand}$);
11:            gain = AIG_num_gates - $AIG_{sand}$_num_gates;
12:            if ((gain > gain_max) ||
13:          ( (gain_max == 0) && (use_zero_gain) )
14:           ) then
15:                $C_{best} = C_{optimal}$;
16:                gain_max = gain;
17:                polyedges_max = polyedges(C);
18:        if ($C_{best}! = NULL$) then
19:            replace cut(AIG, $C_{best}$);
20:    return AIG;
```

Pseudo-algorithm of PAIG rewriting procedure is denoted in Algorithm 1. The Algorithm 1 describes a single rewrite iteration through a circuit C. For maximum efficiency of a synthesis process, it is recommended to run more than one iteration, until zero gain is achieved - this principle is valid for original rewriting also.

In comparison to the original AIG rewriting is iteration through AND nodes in the *reverse topological* order. The reverse iteration from roots to leaves will ensure propagation of polymorphism deeper into a network. To achieve maximum expansion of polymorphism into a network, in contrast to ordinary rewriting, PAIG rewriting is processing each node unless the gain of cut or whole network is negative.

PAIG rewriting algorithm is a key element behind the first synthesis and optimization method targeting multi-functional circuits which does not involve usage of any heuristic aspects at its core. All steps in the algorithm are strictly defined and the optimization process is fully controlled in comparison to evolutionary optimizations. The algorithm ensures polymorphism propagation deeper into the circuit structure and enables sharing of common sources of both desired functions.

3.2.2 Cut enumeration in PAIG

Cut enumeration completely follows the known process of construction and recursive cut enumeration. DAG cuts and tree cuts are considered with respect to DAG-aware rewriting.

It is quite typical for a conventional rewriting procedure that 4-input (k = 4) cuts are commonly used. If k = 3, the chance to find a replacement with reasonable potential to improve the circuit representation is reduced significantly. If k = 5, a number of subgraphs grows rapidly instead. A 4-input polymorphic cut capable to switch between two functions require an additional 5th input. The additional input ensures the function switching and for purposes of this thesis the input is named *virtual polymorphic input.* The virtual polymorphic input in conjunction with 4-input cuts for the purpose of optimizing polymorphic circuits makes up altogether 5-input cuts. It is possible to observe a direct influence on the expansion of state-space comprising all permissible cuts.

Cut enumeration produces a set of all k-feasible cuts assigned to each node. Cut enumeration starts at leaves and continues in topological order to the root of AIG. Each node contains at least trivial cut. Each set of cuts of node n is computed as Cartesian product of two previous cut sets of nodes a and b. Formal notation of cut sets computation of node n is following [38]:

$$\phi(n) = \{\{n\}\} \cup \{u \cup v | u \in \phi(a), u \in \phi(b), |u \cup v| \leq k\}$$

Cartesian product of two sets creates a new cut set of node n, while keeping only K-feasible cuts.

3.2.3 Optimal circuit generator - MinCirc

Based on AIG rewriting principles, the rewriting replaces non-optimal subgraphs by optimal. These optimal subgraphs must be somehow available. There are two options here: to have a precomputed library of optimal subgraphs or compute optimal subgraphs during runtime. For 4-input subgraphs and usage of NPN-class, we must have available 222 optimal structures, that are used for the ordinary rewriting algorithm (4-input cuts are used). Unfortunately, polymorphism adds additional virtual input - a polymorphic control wire, hence the polymorphic cut enumeration searches for 5-input cuts instead. Due to very high number of 5-input cuts (4-real input, 1-virtual input), a number of possible solutions is growing up at an extremely fast pace and, thus, it is rendering the option to keep all the pre-computed optimal cuts structures unrealistic. It simply becomes necessary to compute optimum cuts on-line.

Nan Li and Elena Dubrova proposed a technique for AIG rewriting using 5 input cuts. 5-input NPN equivalence classes circuits counts 616126, this number of graphs is too big to precompute and store. Authors experimentally discovered that only 2749 classes appear in all IWLS 2005 benchmarks [7]. Further they picked 1185 classes of 2749 with more than 20 occurrences and they generated best circuits for representative functions [28].

To generate an optimum circuit is $\sum_2 -complete$ problem [53] and this problem worries researchers since 1970's. Some methods have been introduced, for example methods based on ILP (Integer Linear Programming) or SAT based approaches (listed in [18]).

Although the generation process of optimum circuit implementation is well-mastered process in case of conventional circuits, no such approach hadn't existed for polymorphic circuits until introduction a MinCirc tool [18]. The MinCirc tool is a tool for generation of optimum circuits including polymorphic circuits using proposed PAIG format. MinCirc is mainly used for an on-line computation of *optimum sub-graphs* in PAIG rewriting. Once the MinCirc produces optimum subgraphs (structures) for particular function, the optimum structure is chosen and deployed on the basis of achieved gain of overall network.

Because the generation process of optimum functions is essential for the rewriting algorithm, let us look under-hood the MinCirc. The initial version of MinCirc was mainly designed for generating optimum circuits with XOR gates [19], while later was extended for polymorphic circuits. MinCirc extension for polymorphic circuits exploits a property that polymorphism can be viewed as an additional primary input. Thus, a polymorphic stimulus P is introduced and enables (switches) between polymorphic modes. Basically, polymorphic stimulus P is a virtual polymorphic input mentioned in previous section 3.2.2.

An example of formula describing a polymorphic gate implementing a function AND/OR, is following:

$$F = \bar{P}(a * b) + P(a + b)$$

where P is the polymorphic stimulus (virtual polymorphic input) and a, b are primary inputs of gate.

Algorithm

To understand problematics of generation of optimum circuits, this short section briefs MinCirc algorithm, based on [18].

The problem of optimum circuit generation is solved by its reduction to a *decision CNF-SAT problem* [22]. These and similar problems belong to disparate complexity classes of polynomial hierarchy, the reduction is not polynomial. The optimization problem is elegantly limited by a simple trick applied to a decision problem : „Does there exists an n-node implementation of a given k-input function?“. Initial value of $n = 1$. If the answer is „no“, procedure is repeated with incremented n. The algorithm repeats until answer „yes“ is obtained. Then, it is a solution of the original problem.

The algorithm outlined by a pseudo-code in the Algorithm 2. The algorithm input is a truth table of the intended function and the output is an optimal structure. The key procedure in algorithm is *Generate_CNF()*. Detailed description of CNF generation can be found in [18] as well as experimental results.

Algorithm 2 MinCirc algorithm [18]

```
1: procedure GENERATE_OPTIMUM_STRUCTURE(truth_table f, int k)
2:     n = 1;
3:     do
4:         CNF = Generate_CNF(f, k, n);
5:         Sol = SAT_Solve(CNF);
6:         if (Sol.unsat) n++;
7:     while (Sol.unsat);
```

The MinCirc tool also offer to configure some parameters, such as required delay (depth), gate cost, etc. The MinCirc is deployed in PAIG rewriting for generating optimum sub-circuits and completely covers *Genereate_optimal_subgraph(F)* in PAIG rewriting Algorithm 1.

3.2.4 Cut replacing

The original AIG algorithm performs *dereferencing* and *referencing* nodes during replacement and structural hashing is immediately applied per each node change. PAIG rewriting, implemented in the PAIG tool, removes nodes matching an inspected cut from a network and adds new nodes into a network corresponding to optimal sub-graph (generated by MinCirc). So modified network is structurally re-hashed at once. It may affect the performance of PAIG algorithm. Nevertheless, the complexity of original AIG rewriting algorithm is well-known. Cut replacement implementation can be improved later by referring the original AIG rewriting.

Chapter 4

Evaluation of multi-level polymorphic design and optimization method

In order to prove and evaluate the proposed PAIG rewriting algorithm, a set of experiments has been prepared for thorough evaluation. Experiments are divided into two groups, where deployment of polymorphic circuits make a sense. The first group of experiments rests in conversion of conventional circuits to polymorphic, and so converting one primary input to a virtual polymorphic input. The experiments may prove that circuits composed of polymorphic components may be solved more effective.

The second group of experiments consist in a joining of two different circuits and switching their output function using polymorphism. The main idea is to share common resources of two completely different circuit using polymorphism and demonstrate that polymorphic circuits saves resources.

Both kinds of experiments are working with the same batch of 21 combinatorial circuits from a publicly available benchmark set *LGSynth91* [56]. Table 4.1 summaries the properties of combinatorial circuits used for experiments. Circuits in a set are chosen with a various number of AND nodes, high number of primary inputs and outputs in order to thoroughly evaluate proposed algorithm.

All the experiments presented in this section are successively performed in accordance to the synthesis flow previously discussed in Section 3.2.

In order to put the proposed solution and obtained results presented in this contribution into a proper context, an important aspect behind the experimental work takes aim at providing an illustrative comparison in terms of synthesis efficiency between polymorphic-based rewriting approaches against other convenient optimization methods exhibiting scalable properties. However, it is necessary to take into account an important fact that no easily scalable methodology for synthesis and optimization of polymorphic circuits has been reported up to the date.

4.1 Conversion of primary input to virtual polymorphic input

Experiment that demonstrates applicability of PAIG rewriting on conventional circuits is presented in this section. Conventional circuit is modified into a polymorphic circuit in the

Table 4.1: List of combinatorial circuits used for both kinds of experiments.

Index	Circuit name	Primary Inputs	Primary Outputs	AND nodes
1	cht.aig	47	36	185
2	apex1.aig	45	45	2604
3	apex6.aig	135	99	659
4	apex7.aig	49	37	221
5	lal.aig	26	19	109
6	c8.aig	28	18	169
7	misex2.aig	25	18	119
8	misex3.aig	14	14	1549
9	misex3c.aig	14	14	721
10	pcler8.aig	27	17	71
11	my_adder.aig	33	17	176
12	ttt2.aig	24	21	218
13	C499.aig	41	32	400
14	C1355.aig	41	32	504
15	seq.aig	41	35	2411
16	count.aig	35	16	127
17	unreq.aig	36	16	112
18	pdc.aig	16	40	1621
19	vda.aig	17	39	924
20	k2.aig	45	45	1998
21	rot.aig	135	107	550

following way: a random primary input is converted to a polymorphic driven one on the basis of environmental state.

The main objective of the proposed experiment is to produce an optimized structures of polymorphic circuits.

4.1.1 Specification of benchmark set

For demonstration purposes, all 21 polymorphic circuits has been selected from the table 4.1 for thorough evaluation, where circuit properties are also summarized. Each test case C consists of one combinatorial circuit from a publicly available benchmark set *LGSynth91* (listed in table 4.1). The set has been selected with various number of inputs PI and outputs PO. Selection of the individual test circuits used during the consecutive experimental evaluation did not reflect any further consideration or properties (e.g. signal propagation delay or interconnection complexity of a given circuit structure) than the aspect explicitly mentioned above. However, it could be interesting to assess the proposed approach behavior also from this standpoint during some of the future research activities.

Each test circuit C has number of primary inputs PI^C and number of primary outputs PO^C. One of PI^C is removed and substituted by virtual polymorphic input P in initial circuit, that cause the PI^C is controlled by the basis of environmental state. See figure 4.1 for more details of the conversion.

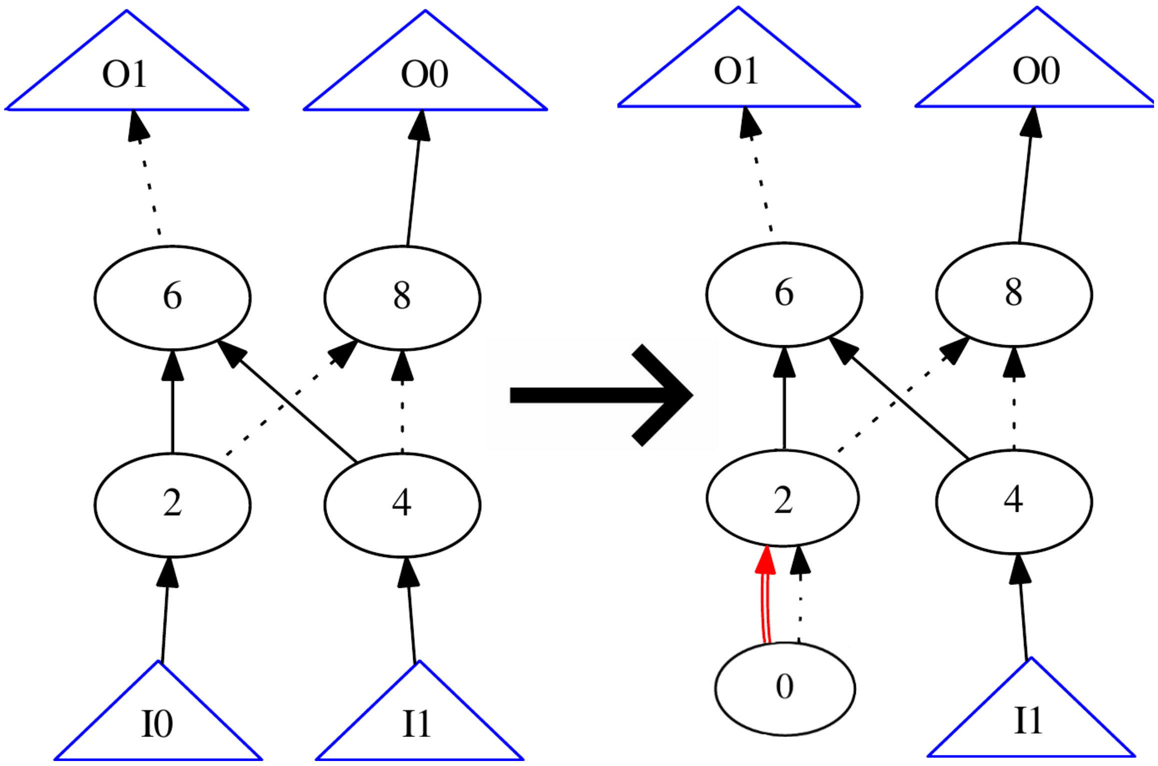

Figure 4.1: Example of primary input *I0* conversion to virtual polymorphic input.

4.1.2 Results analysis

Details on results obtained thanks to the polymorphic-oriented modification of conventional rewriting process are shown in table 4.2. The table itself is further divided into four sections, where the first one identifies the individual circuits from a given benchmark set. Then, the second one outlines the average results after optimization by PAIG rewriting that led to an observable circuit improvement. The third section reports an optimization results obtained with ABC tool and finally, the fourth one depicts the improvement PAIG rewriting comparison against ABC tool results. All results in the table 4.2 are averaged values of permutation over all primary inputs.

A column *num of rewrites* in second section reports average number of applied sub-circuit replacements per one iteration (one iteration = one rewrite command). A column *AND nodes before* represents a number of AND nodes of initial circuit. A column *AND nodes after* denotes the resulting number of AND gates required by a given target circuit once the synthesis process is finished. Analogically, a column *gain* reports a number of saved AND nodes and it is computed as subtraction of number ands after from number ands before. Both columns P_{edges} denotes the overall number of polymorphic edges used in that circuit. A column *avg cuts per it* reports average number of all found cuts per one iteration (one rewrite command). Next column *rewrite iterations* denotes a number of called „rewrite“ command to reach the best optimization. *Improvement PAIG* shows the details on a percentage improvement against the initial circuit. *Time* column contains an assessment of the elapsed time of the whole synthesis process including time for generation of optimal sub-circuit.

For the purpose of drawing a relevant comparison a contribution of standard ABC tool was used for an optimization of combinatorial circuits from every circuit under optimization given in table 4.1. With the aim to make optimization results more relevant, the optimization is performed especially for each primary input PI^C of each initial circuit and results are averaged. Thus, results report average values of optimization for all primary inputs.

The third section reports briefly maximal optimization reached by ABC tool, whereas several iterations has been applied too. The best optimization using ABC tool was achieved no later than in the in fourth iteration of the ABC „rewrite“ command. In summary, the optimization flow exploiting the conventional ABC-based rewriting methodology has resulted into the average reported improvement of 17.50% against the initial benchmark circuits arrangement. The last section contains only one column, reporting the difference of improvement of PAIG optimization against to ABC tool optimization.

An improvement or gain expressed in terms of AND nodes saving achieved by PAIG throughout all the experiments is reaching 21.02% in average. Further details mentioned here within the second section of table 4.2 give an overview of the proposed approach characteristics in situation when the objective was to achieve the best possible refinement of polymorphic circuits structure from the specified benchmark set. Efficiency of the PAIG-based rewriting algorithm is enumerated in column *Improvement PAIG*. The best derived solution (averaged per all primary inputs) has 34.76% improvement, which confirms the ability to design competitive multi-functional circuits.

The table 4.2 provides also a closer insight into the performance and runtime behavior (which is indeed a significant aspect when it comes to optimization of complex circuits) of the proposed approach in case of the chosen benchmark set, when the on-line computation of optimum sub-circuits (replacement cuts) is taken into account.

On-line optimum cut computation is a difficult task, which is managed by MinCirc tool within the proposed synthesis flow. In order to minimize the impact of that particular property exhibited by MinCirc, already generated PAIG graphs are reused without even launching the tool. The MinCirc produces optimum sub-graphs in a quite fast manner for the majority of functions (combinatorial circuits handed in to the synthesis process), however, some of functions are too difficult to be resolved in an acceptable time.

Therefore, a 5 seconds timeout period for MinCirc tool has been chosen. It is necessary to specify the timeout period in a cautious manner because its overly constrained value prevents the generation of good sub-graphs having an importance for circuit improvement. Tens of optimal subgraphs are usually generated in this period. On the contrary, too generous timeout significantly extends the overall duration of synthesis process. If the MinCirc runtime period expires for a particular function, the function is noted as difficult and MinCirc skips the function in the future with the aim to reduce the necessary synthesis time. Unfortunately, such timeout could potentially lead to non-deterministic behavior on a different computing platforms than the one actually used in that case. To get rid of this drawback, the future research activities could explore the possibility of creating the database of precomputed optimum circuits. The algorithm is expected to run just a few seconds with precomputed optimum cuts and effective PAIG code implementation. The experiments were performed using a workstation equipped with Intel(R) Core(TM) i7 CPU 920 processor.

Table 4.2: Optimization results for polymorphic rewriting.

average per inputs	num of rewrites	AND nodes before	AND nodes after	AND nodes gain	P$_{edges}$ before	P$_{edges}$ after	avg cuts per it	rewrite iterations	improvement PAIG [%]	time [s]	improvement ABC [%]	PAIG against to ABC [%]
cht	115,33	185	143,50	41,50	4,35	109,81	11 094,63	2,06	22,43	14,45	20,00	2,43
apex1	3 558,91	2604	2 105,56	498,44	41,53	394,02	1 308 036,86	5,40	19,14	6 861,38	19,12	0,02
apex6	392,48	659	621,80	37,20	3,71	55,20	61 165,19	4,98	5,64	306,56	2,88	2,76
apex7	168,54	221	192,33	28,67	3,50	37,00	11 582,29	3,42	12,97	305,05	9,50	3,47
lal	93,11	109	72,11	36,89	3,81	16,48	2 391,04	4,30	33,84	251,93	31,19	2,65
c8	327,31	169	110,21	58,79	6,52	59,62	7 904,00	6,28	34,79	221,14	28,99	5,79
misex2	86,58	119	99,08	19,92	4,81	22,31	2 946,73	3,54	16,74	187,57	19,33	-2,59
misex3	4 387,07	1549	1 240,40	308,60	81,80	440,13	677 783,33	8,13	19,92	12 319,53	18,40	1,52
misex3c	1 409,27	721	597,80	123,20	42,20	225,20	155 692,93	5,47	17,09	3 584,50	15,26	1,83
pcler8	325,89	169	110,25	58,75	6,14	59,57	7 910,18	6,25	34,76	3,65	9,86	24,90
my_adder	127,50	176	123,00	53,00	3,53	62,29	8 190,06	2,62	30,11	32,42	25,57	4,55
ttt2	207,32	218	164,60	53,40	7,56	45,76	9 597,48	4,12	24,50	633,52	23,85	0,64
C499	1 319,93	400	381,69	18,31	5,14	296,17	135 858,07	4,14	4,58	1 330,76	2,00	2,58
C1355	1 397,21	504	381,50	122,50	5,14	290,64	138 408,69	4,40	24,31	2 399,39	15,08	9,23
seq	3 145,46	2411	1 912,44	498,56	43,76	297,61	917 020,68	6,00	20,68	7 056,94	20,32	0,36
count	131,92	127	108,67	18,33	3,22	51,03	7 882,17	2,44	14,44	61,37	11,81	2,62
unreg	120,76	112	108,00	4,00	3,51	78,59	8 772,11	2,00	3,57	9,99	0,00	3,57
pdc	4 158,53	1621	1 076,18	544,82	75,82	365,59	549 684,82	8,71	33,61	13 569,84	29,61	4,00
vda	2 694,53	924	694,47	229,53	29,00	220,76	222 492,59	7,65	24,84	10 845,27	25,97	-1,13
k2	1 337,81	1998	1 384,84	613,16	6,19	67,62	284 169,27	7,32	30,69	3 128,60	29,93	0,76
rot	332,07	550	480,32	69,68	3,21	66,61	63 168,34	3,30	12,67	327,31	8,91	3,76
Average	1 230,36	740,2857143	576,61	163,68	18,31	155,34	218 654,83	4,88	21,02	3 021,48	17,50	3,51

4.2 Switching between two different functions

This section describes experimental results that demonstrate usability of PAIG rewriting for joining two different circuits into one polymorphic circuit, where their outputs are multiplexed on the basis of environmental state.

A main objective of the proposed approach is to produce an optimized structure of polymorphic circuits while mutual, non-conflict sharing of common resources between two initial circuits (input of the synthesis toolkit based on the proposed PAIG rewriting technique) is naturally ensured.

4.2.1 Specification of benchmark set

For experimental purposes, 15 polymorphic circuits has been selected from the the table 4.1 for thorough evaluation. Circuit properties are summarized in the table 4.3. Each test case C consists of two circuits A and B (listed in the table 4.1), where both of them have a similar number of inputs PI and outputs PO. Choice of individual test circuits or their mutual combination into a target circuit pair C used during the consecutive experimental evaluation did not reflect any further consideration or properties (e.g. signal propagation delay or interconnection complexity of a given circuit structure) than the aspect explicitly mentioned above.

Each test circuit C has a number of primary inputs $PI^C = max(PI^A, PI^B)$ and a number of primary outputs $PO^C = max(PO^A, PO^B)$. Since the primary inputs of circuit C are shared, the primary outputs $min(PO^A, PO^B)$ are connected using polymorphic multiplexers. See figure 3.7 for more details on implementation of polymorphic multiplexers. Remaining outputs are assumed to have permanent/constant function. Column $AND^{A,B,C}$ denotes the number of two-input AND nodes used in a given circuit. In case of circuit C, the initial number of AND nodes is following: $AND^C = AND^A + AND^B + AND^{pmux} * min(PO^A, PO^B)$, where $AND^{pmux} = 3$.

4.2.2 Results analysis

This subsection is divided into two parts: Results collected during the benchmark circuits processing by ABC tool and results collected during the benchmark circuits processing by PAIG tool.

ABC results

Similarly as previous experiment, for the purpose of drawing a relevant comparison, was the contribution of standard ABC tool used for optimizing A and B combinatorial circuits from every test case C given in table 4.3. Then, the optimized variants of both A and B combinatorial circuits are switched accordingly through the polymorphic multiplexer.

Results collected during the benchmark circuits processing by ABC tool are shown in a table 4.4). The table itself is further divided into three large sections, where the first one identifies the individual circuits from a given benchmark set. Then, the second one outlines the results after the first iteration that led to an observable circuit improvement and finally, the third one depicts the results when the synthesis process reached the best optimization level.

Each section of the table also provides the details on total number of AND nodes required by a given benchmark circuit C in three different situations (no optimization took place,

Table 4.3: Combination of combinatorial circuits used for evaluation PAIG rewriting by joining two different circuits. Polymorphic multiplexers are connected to primary outputs of both circuits. Thus P_{edges} represents number of polymorphic edges in polymorphic circuit to be optimized.

polymorphic circuit C	circuit A	circuit B	AND nodes	P_{edges}
C1	cht.aig	apex7.aig	514	144
C2	lal.aig	c8.aig	332	72
C3	misex2.aig	c8.aig	342	72
C4	pcler8.aig	c8.aig	291	68
C5	my_adder.aig	count.aig	351	64
C6	misex2.aig	lal.aig	282	72
C7	ttt2.aig	lal.aig	384	76
C8	ttt2.aig	misex2.aig	391	72
C9	lal.aig	pcler8.aig	231	68
C10	C499.aig	C1355.aig	1000	128
C11	count.aig	unreq.aig	287	64
C12	my_adder.aig	unreg.aig	336	64
C13	pdc.aig	vda.aig	2662	156
C14	apex1.aig	k2.aig	4737	180
C15	misex3.aig	misex3c.aig	2312	56

after the first iteration of ABC processing with some improvement, the best optimization result accomplished). Results shown in table 4.4 were obtained in the following way:

- optimized circuits were joined by polymorphic multiplexers connected to primary outputs in order to create polymorphic circuit,
- a rewrite command was issued on the input circuits A and B until any improvement at all,
- a final number of AND nodes has been counted (*column A+B+pmux*) and compared to the situation with initial circuits (*column Impr.*).

In summary, the optimization flow exploiting the conventional ABC-based rewriting methodology has resulted into the average reported improvement of 17.83% against the initial benchmark circuits arrangement.

PAIG results

Results achieved by the polymorphic rewriting are shown in table 4.5. The table itself is further divided into three large sections. The first one identifies the individual circuits from a given benchmark set. Then, the second one outlines the results after the first iteration that led to an observable circuit improvement and finally, the third one depicts the results when the synthesis process reached the best optimization level. A column AND_C in second and third section simply denotes the resulting number of AND gates required by a given target circuit once the synthesis process is finished. P_{edges} denotes the overall number of polymorphic edges used in that circuit. *Rwrts* shows the number of sub-circuit replacements. *Gain* reports number of saved AND gates and *Impr.* shows the details on a percentage improvement against the initial circuit. *Runtime* column contains an assessment of the elapsed time in case of the first iteration that brought an observable improvement (second section of the table) and also of the whole synthesis process (third section of the table).

It is important to explicitly mention the fact that so called KL-cuts were enabled during the first iteration in order to get rid of the polymorphic multiplexers occurrence at the primary outputs. In general, KL-cuts offer an easier way how to find a proper cut (a sub-circuit eligible to be replaced by its optimized version) with polymorphic edges.

The table 4.5 provides a closer insight into the performance of the PAIG rewriting in case of the chosen benchmark set, when precomputed optimum cuts are taken into account. An improvement or gain expressed in terms of AND nodes saving achieved throughout all the experiments is reaching 23.00% in average after the first iteration with multi-output cuts option enabled. Further details mentioned here within the third section of table 4.5 give an overview of the proposed approach characteristics in situation when the objective was to achieve the best possible refinement of polymorphic circuits structure from the specified benchmark set. Efficiency of the PAIG-based rewriting algorithm is enumerated in column *Impr.* The best derived solution has 48.11% improvement, which confirms the ability of the proposed synthesis method to design multi-functional circuits while simultaneously trying to employ the principle of common resources sharing. Finally, it is possible to notice an average improvement of 25.95% across the whole benchmark set in comparison to the initial circuit C_n.

Table 4.4: Optimization results for conventional ABC rewriting.

Initial circuit			First rewrite iteration [AND nodes]					Rewrite iterating to the best [AND nodes]				
circuit C	A+B	A+B+pmux	A	B	A+B	A+B+mux	Impr [%]	A	B	A+B	A+B+pmux	Impr [%]
C1	406	514	148	201	349	457	11.09	148	200	348	456	11.28
C2	278	332	86	128	214	268	19.28	75	120	195	249	25.00
C3	288	342	100	128	228	282	17.54	96	120	216	270	21.05
C4	240	291	64	128	192	243	16.49	64	120	184	235	19.24
C5	303	351	131	112	243	291	17.09	131	112	243	291	17.09
C6	228	282	100	86	186	240	14.89	96	75	171	225	20.21
C7	327	384	179	86	265	322	16.15	166	75	241	298	22.40
C8	337	391	179	100	279	333	14.83	166	96	262	316	19.18
C9	180	231	86	64	150	201	12.99	75	64	139	190	17.75
C10	904	1000	394	444	838	934	6.60	392	428	820	916	8.40
C11	239	287	112	112	224	272	5.23	112	112	224	272	5.23
C12	288	336	131	112	243	291	13.39	131	112	243	291	13.39
C13	2545	2662	1172	712	1884	2001	24.83	1141	684	1825	1942	27.05
C14	4602	4737	2123	1474	3597	3732	21.22	2106	1400	3506	3641	23.14
C15	2270	2312	1278	616	1894	1936	16.26	1264	611	1875	1917	17.08
Avg:							15.19					17.83

Table 4.5: Optimization results for polymorphic rewriting.

Initial circuits	First rewrite iteration						Rewrite iterating to the best					
circuit C	AND_C	P_{edges}	Rwrts	Gain	Runtime [h:m:s]	Impr [%]	AND_C	P_{edges}	Iters	Gain	Runtime [h:m:s]	Impr [%]
C1	399	310	240	115	00:05:53	22.37	395	318	4	119	00:06:45	23.15
C2	239	166	145	93	00:02:14	28.01	235	178	4	97	00:03:10	29.22
C3	260	193	150	82	00:02:10	23.98	256	198	3	86	00:03:15	25.15
C4	223	158	128	68	00:02:02	23.37	151	151	3	140	00:02:42	48.11
C5	274	112	141	77	00:00:06	21.94	257	146	4	94	00:00:07	26.78
C6	228	137	130	54	00:02:30	19.15	224	144	4	58	00:03:35	20.57
C7	314	158	160	70	00:06:14	18.23	304	177	4	80	00:08:10	20.83
C8	308	123	156	83	00:06:34	21.23	300	133	4	91	00:08:19	23.27
C9	193	104	104	38	00:00:36	16.45	188	110	3	43	00:00:47	18.61
C10	711	580	759	289	00:02:04	28.90	709	507	3	291	00:05:35	29.10
C11	225	147	158	62	00:01:15	21.60	225	147	2	62	00:01:31	21.60
C12	241	160	207	95	00:01:21	28.27	241	160	2	95	00:01:27	28.27
C13	1979	258	773	683	01:49:11	25.66	1938	288	5	724	01:54:46	27.20
C14	3573	399	1329	1164	02:47:13	24.57	3536	423	6	1201	03:03:47	25.35
C15	1819	170	410	493	00:33:10	21.32	1803	163	5	509	00:36:29	22.02
Avg:						23.00						25.95

Experimental measurements have revealed that the largest chunk of computational time is consumed by PAIG code execution caused by ineffective implementation. But, from logic point of view, rewriting technique is very fast, while the code is written well. For example, ABC code is pretty well optimized and optimum cuts are stored in effective way. The experiments were performed using a workstation equipped with Intel(R) Core(TM) i7 CPU 920 processor.

For evaluation and comparison of PAIG rewriting algorithm efficiency with ABC conventional rewriting, it is possible to analyse in detail tables 4.4 and 4.5 respectively. In this way it is possible to recognize a significant advantage behind PAIG rewriting algorithm when it comes to the optimization of polymorphic circuits. A closer look will reveal the fact that the algorithm is able to achieve about 8.12% better optimization of polymorphic circuits than the conventional approach based on the utilization of ABC rewriting.

4.3 KL-cuts influence on PAIG rewriting

K-cuts are an efficient representatives of a region of an AIG, where the region has one output. Lets imagine a multiple output region. Such region would have to be covered by count of K-cuts. KL-cuts are novelty introduced in [32], that covers a multiple output region. It is supposed, that KL-cuts may help to find a more cuts in an AIG and thus offer to PAIG rewriting more paths for optimization of polymorphic circuits.

For this kind of experiment, a hypothesis was set: *Forced deployment of cuts with zero contribution to the optimization of a circuit structure during replacement stage allows to propagate polymorphism deeper into the circuit. Then, the utilization of KL-cuts can help to generate more comprehensive pool of cuts with the possibility to achieve improvement > 0 and, thus, perform synthesis of polymorphic circuits in terms of better area results.*

In this approach, KL-cuts are generated in the same way as K-cuts. However, KL-cuts are not dropped during the generation process. Generation process of KL-cuts does not conceal any other difficulty instead of expansive growth of generated solutions.

Despite the fact that the generation process of KL-cuts is quite straightforward, some complications still do emerge during the consecutive replacement process.

For the purpose of examining the influence of KL-cuts on the optimization efficiency of polymorphic circuits, the implementation of PAIG optimization tool was used. This section provides detailed overview of the experimental results obtained with this tool on set of benchmark circuits.

4.3.1 Specification of benchmark set

Benchmark set *(LGSynth91 [56])* has been chosen as the starting point for subsequent evaluation of the proposed PAIG-based rewriting scheme using KL-cuts at its core. The experiments were performed using 15 pairs of similar conventional circuits from that benchmark set. Detailed overview of selected circuits properties is provided in table 4.3. See section 4.2 for more details, because this experiment use the same benchmark set and initial circuit setup as experiments called „Switching between two different functions".

The experimental part of my contribution presented in this chapter is closely related to the objective to confirm or deny the hypothesis formulated in section 4.3 that the utilization of PAIG-based rewriting with KL-cuts for the purpose of polymorphic circuit synthesis tasks is expected to deliver better results in terms of resulting area optimization. Thus all the experiments are first conceived as a comparison of polymorphic rewriting algorithm with

K-cuts to the variant which, on the contrary, involves KL-cuts. Second, the comparison is shown as an area improvement for both variants expressed in percentage value, where the attention is also given to the possible state space expansion in case of KL-cuts.

The main objective of polymorphic-aware rewriting is to produce an optimized structure of polymorphic circuits while mutual, non-conflict sharing of common logic resources between two initial circuits (the desired functions to be performed by the resulting circuitry) is accomplished.

Experiments are performed in two stages, with the assistance of PAIG tool implemented in C language. First of all, synthesis of circuits from the benchmark set is performed with the K-cuts. During the second round of experiments, the utilization of KL-cuts is allowed. Details of the initial circuits configuration are given in table 4.3.

4.3.2 Results analysis

Obtained results are depicted in table 4.6. The organization of the table summarizing the results is conceived with two main sections. Each of them is dedicated to a separate round of experiments in case of K-cuts and KL-cuts.

Each section in table 4.6 has the following sub-columns: Column *Iters* denotes how many iterations of polymorphic rewrite were used for a particular circuit Cn until it becomes resilient to further attempts of its optimization. Column *Tot.rewrites* counts the number of performed rewrites (replacements) as a sum of all iterations. Column *Ands* denotes a number of all gates (nodes) within the optimized circuit. *Gain* column contains information about number of saved gates in comparison to the initial circuit pair. *Impr.* column denotes how much area has been saved (number of nodes) using polymorphic rewrite algorithm in percentage value. This description is applicable for both experimental stages (first one with K-cuts and second one with KL-cuts enabled).

In addition, it is possible to notice also a third main section entitled *Influence* within table 4.6. Its sole purpose is to provide a comparison between using K-cuts and KL-cuts. A column *Impr.* in this section gives an account of the area improvement while KL-cuts usage is enabled in contrast to the situation with KL-cuts considered as prohibited. In other words, *Impr.* column illustrates the effect of KL-cuts onto the quality of resulting solution produced by the polymorphic rewriting optimization algorithm. At last, column *Growth* shows space explosion of investigated cuts in the case of enabled KL-cuts.

As it becomes apparent from a closer inspection of table 4.6, KL-cuts are undoubtedly helping to get more optimized polymorphic circuit structure in most of the situations, when the average improvement is reaching the level of 4.39%. The maximum value of the obtained area improvement of polymorphic circuit optimization using KL-cuts is 31.10 % against K-cuts rewriting for the circuit variant identified as *C10*. Whereas in the case of circuits *C11* and *C12* the utilization of KL-cuts did not have any impact on optimization at all. Please, refer figure 4.2, where both variants are graphically compared.

However, although the exploitation of KL-cuts brings only positive area optimization result, another important aspect deserves a further attention - explosion of cut set range (set of cuts to be investigated). State space explosion comprising different variants of cuts can be observed in the last column of table 4.6 and it is also depicted in figure 4.3. As it can be clearly seen, number of investigated cuts grows to 172.07% in average, where the maximum observed value of the state space growth is 459.64%.

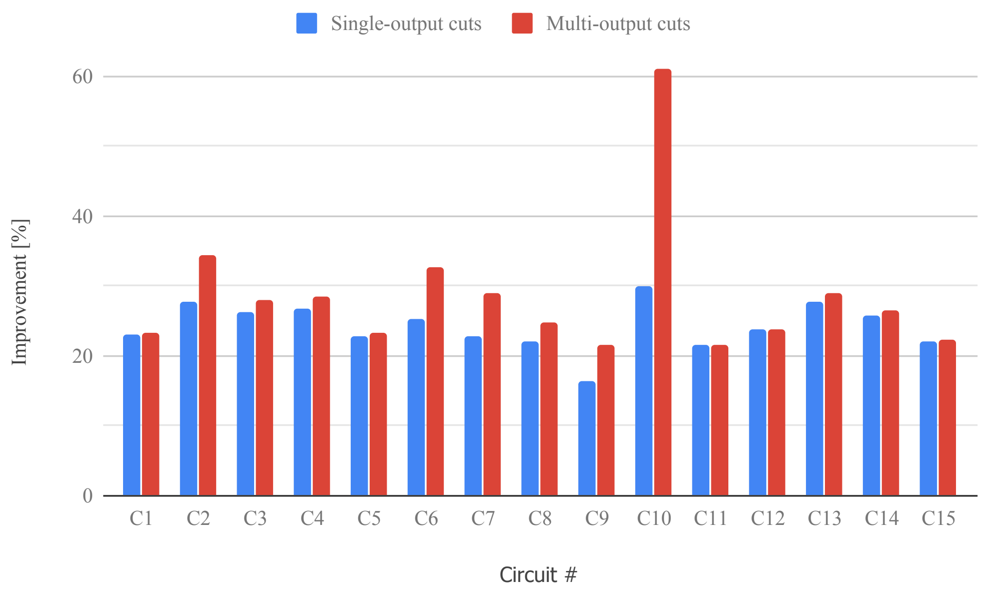

Figure 4.2: Graph shows the improvement in case of optimized polymorphic circuits: blue bars denote percentage improvement of circuits with K-cuts only and red bars denote percentage improvement of circuits with KL-cuts allowed.

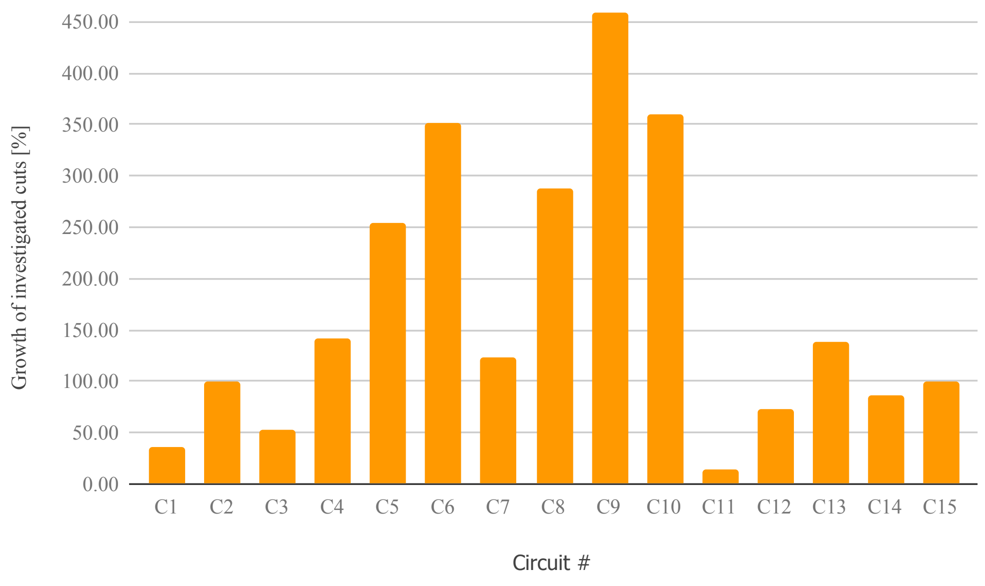

Figure 4.3: Graph reflects percentage growth of the number of investigated cuts for all chosen test circuit pairs when KL-cuts were allowed.

Table 4.6: Results of polymorphic circuit synthesis.

	K-cuts						KL-cuts						Influence	
circuit C	Iters	Tot.cuts	Tot.rewrites	Ands	Gain	Impr. [%]	Iters	Tot.cuts	Tot.rewrites	Ands	Gain	Impr. [%]	Impr. [%]	Growth [%]
C1	5	368410	180	395	119	23.15	5	503860	216	394	120	23.35	0.19	36.77
C2	8	193296	151	240	92	27.71	13	386958	280	218	114	34.34	6.63	100.19
C3	7	205296	162	252	90	26.32	9	315270	219	246	96	28.07	1.75	53.57
C4	5	127190	100	213	78	26.80	11	308011	236	208	83	28.52	1.72	142.17
C5	4	227904	128	271	80	22.79	8	806760	398	269	82	23.36	0.57	253.99
C6	11	161073	168	211	71	25.18	19	728137	622	190	92	32.62	7.45	352.05
C7	13	423280	342	296	88	22.92	16	941728	550	273	111	28.91	5.99	122.48
C8	10	293330	215	305	86	21.99	19	1139221	654	294	97	24.81	2.81	288.38
C9	4	67604	70	193	38	16.45	14	378336	334	181	50	21.65	5.19	459.64
C10	5	2438025	242	700	300	30.00	16	11207584	1489	389	611	61.10	31.10	359.70
C11	4	158692	130	225	62	21.60	4	180348	132	225	62	21.60	0.00	13.65
C12	4	211668	128	256	80	23.81	4	367064	191	256	80	23.81	0.00	73.41
C13	18	14216436	1359	1923	739	27.76	24	33897768	2389	1889	773	29.04	1.28	138.44
C14	19	41769448	2249	3516	1221	25.78	20	78111460	3209	3481	1256	26.51	0.74	87.01
C15	8	4593200	480	1803	509	22.02	14	9168838	838	1794	518	22.40	0.39	99.62
Average:						**24.29**						**28.67**	**4.39**	**172.07**

4.3.3 Related summary

This subsection was dealing with the investigation of a hypothesis that the usage of KL-cuts may lead to an improvement of polymorphic circuit optimization. The proposed approach is employed in a close conjunction with the PAIG-based rewriting algorithm. Especially due to the number of modifications that were introduced in case of polymorphic rewriting itself (e.g. forced replacement due to propagation of polymorphic edges deeper into the circuits structure) in comparison to the conventional rewriting variant, the hypothesis has been successfully confirmed. The obtained results clearly demonstrate a positive impact of the KL-cuts scheme, yet the improvement reach beyond the level of just a few percent.

4.4 Comparison of PAIG rewriting to PolyBDD

In order to compare PAIG rewriting results to the most famous methodology for polymorphic circuit design PolyBDD [20], the PAIG rewriting was applied to the same circuits that Gajda reports in his thesis.

Mr. Gajda has selected a number of test circuits for evaluation of PolyBDD method. Each polymorphic circuit is composed of two circuits performing independent functions. Gajda has used only polymorphic gates of type NAND/NOR and AND, OR, XOR, NAND, NOR, inverter and multiplexer gates. Unfortunately, his results report just overall numbers of required gates after synthesis by PolyBDD method, regardless of the price of used gates (especially XOR and multiplexer). An optimum implementation of XOR gate and 2-way multiplexer is consisting of 3 two-input AND gates. In table 4.7 in a column *PolyBDD*, numbers of required gates are reported, includes expensive XOR and MUX gates after PolyBDD synthesis. Figure 4.4 shows a PolyBDD structure (a) and corresponding polymorphic circuit (b) (note that circuit is composed of multiplexers mainly). The number of multiplexers in non-reduced BDD grows with power of 2 of primary inputs.

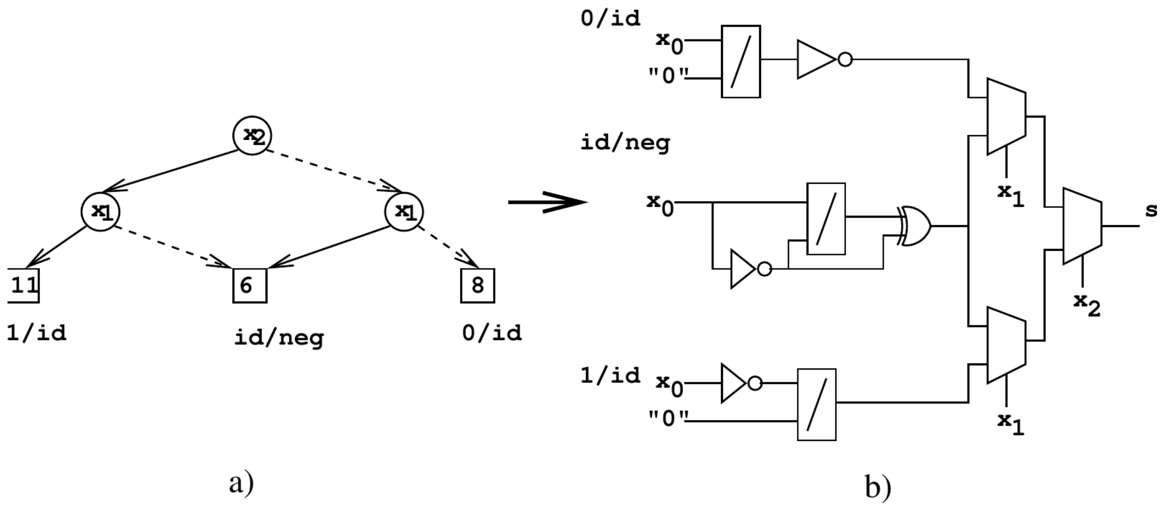

Figure 4.4: PolyBDD circuit (a) and corresponding polymorphic circuit (b) [20].

The proposed PAIG rewriting approach can handle only AND gates, stemming from the nature of AIG. Thus AIG cannot represent complex gates such as *XOR gate* and *multiplexer* natively. Despite that, one node type is not a disadvantage. It allows effective simple optimizations. Simple AIG structures are mappable to complex target technology during technology mapping process. However, different metrics are appearing and comparison of PAIG to PolyBDD may be inaccurate.

Table 4.7 reports a comparison of PAIG to PolyBDD method. As it is outlined above, metrics of methods are different. The column *Circuit* denotes a name of a desired polymorphic circuit. M/S is Majority/Sorter, M/P is Majority/Parity and xA/xB is multiplier by A and B. The column *Inputs* contains number of primary inputs, analogically the column *Outputs* contain number of primary outputs. The column *PolyBDD* contains numbers of used gates after PolyBDD synthesis considering these types of gates: *{NAND/NOR polymorphic gate, AND, OR, XOR, NAND, NOR, 2-way multiplexer, inverter}*.

The next column *PolyBDD Evo.* contains a number of gates required after evolutionary optimization of PolyBDD synthesized circuits. Evolutionary optimization of PolyBDD circuit is performed by CGP, where are two-input elements only. Thus evolutionary results do not reflect complex multiplexers, that are often placed in PolyBDD circuits (see figure 4.4). Unfortunately, complex XOR gates are included, which may be a disadvantage for PAIG comparison. It is especially visible in the case of Majority/Parity circuits, that are mainly composed of XOR gates. However, the PAIG is still competitive, although XOR gate costs three 2-input AND gates.

The fifth column reports results of PAIG rewriting synthesis and values denotes a number of used 2-input AND gates. Analogically, the sixth column reports a number of required 2-input gates using conventional AIG (results from ABC tool).

A reader can compare efficiency of PAIG with PolyBDD by focusing columns *PolyBDD Evo.* and *PAIG*. *PolyBDD Evo.* includes complex XOR gates, that are not possible to express in PAIG structure natively.

Circuit	Inputs	Outputs	PolyBDD	PolyBDD Evo.	PAIG	AIG (abc)
M/S4	4	4	31	45	19	27
M/S5	5	5	50	71	43	49
M/S6	6	6	94	131	88	97
M/S7	7	7	150	212	162	170
M/S8	8	8	269	375	311	321
M/S9	9	9	428	697	602	614
M/P7	7	1	31	41	42	43
M/P9	9	1	41	60	60	61
M/P11	11	1	59	81	86	91
M/P13	13	1	73	114	114	115
x67/x127	7	14	228	274	187	215
x131/x251	8	16	430	547	441	468
x257/x509	9	18	348	410	279	310
x521/x1021	10	20	905	1028	865	894

Table 4.7: Comparison of PolyBDD and PAIG rewriting method.

Chapter 5

Conclusion

Logic synthesis and optimizations techniques are still popular topics despite the fact that they've been researched for at least 50 years. The need for further investigation of these topics is mainly related to growing complexity of digital circuits. Therefore, more effective and scalable synthesis methods are required. New research areas may be opened by emerging technologies or applications, such as multi-functional or polymorphic electronics. The concept of polymorphic electronics was introduced in 2001 [52], almost 20 years ago, and a non-evolutionary, scalable optimization technique still does not exist. Mentioned situation gave me an opportunity to start research of scalable synthesis and optimization methods for polymorphic circuits.

The main goal of this thesis was to propose an effective, scalable method for synthesis and optimization of multi-functional circuits. Initial research started with two-level design methods. The first proposed method, using only NAND/NOR gates, is well applicable to small circuits deployable to REPOMO32. The second method is based on boolean division and kerneling, which is suitable for detection of common parts of two desired circuits. Ongoing research has followed up on multi-level methods and brought new „PAIG" representation for polymorphic circuits in And-Inverter Graphs, which was very useful for further optimization. Since the polymorphic representation was designed, the AIG rewriting technique was adapted to work with the new PAIG representation. The whole synthesis and optimization process of polymorphic circuits clearly gives promising results.

5.1 Thesis contribution

The thesis contribution was partially mentioned in the previous paragraph. The thesis presents a proposal of synthesis and optimization methods for polymorphic circuits.

The first approach, using NAND/NOR gates and dealing with the issues of multi-functional logic circuits synthesis, was introduced in the thesis. The proposed synthesis method was based on a formal Boolean representation of corresponding input functions. Its main advantage can be recognized in its simplicity and an employment of boolean minimization techniques, which is in a direct contrast to existing solutions, predominantly based on heuristic approaches. Despite some constraints of the proposed approach, that were identified during the theoretical analysis and subsequent experiments, the method was successfully applied to real functions specified by the truth table. The obtained results clearly suggest benefits of the proposed approach in comparison the the conventional techniques. It is safe to say that further improvements can be achieved, especially when new types of polymorphic circuit components based on the emerging materials will be prepared [9, 8, 43].

The second milestone was an introduction of kerneling based synthesis method. The method searches common parts of two desired circuits that were randomly generated. The obtained results indicate that it's possible to achieve around 27% improvement, especially in comparison to the synthesis tool called Espresso. Next experiment was performed on real-life complex circuits. Especially, in one case it was possible to achieve almost 40% gates saving. An average improvement on benchmark MCNC circuits is about 20% [54, 13]. Results were also published in Journal of Electrical Engineering [10]

In order to increase the synthesis efficiency of polymorphic circuits, a research was focused to an applicability of AIG and structural hashing for better identification of circuit parts that can be shared between two functions subjected to the synthesis process.

As a result, the novel polymorphic AIG representation format for polymorphic circuits was introduced (section 3.1). It is an extension of AIGER format, which is fully supported in well known tools like ABC. A few experiments showed that the novel format can be very effective representation of polymorphic circuits for future, more complex synthesis processes [14].

An innovative scalable methodology (section 3.2), called PAIG rewriting scheme, capable of synthesis of multi-functional circuits was introduced. The methodology is inspired by an existing rewriting algorithm [36, 37] that was used mainly for optimization tasks of conventional digital logic circuits. The PAIG rewriting offers strictly rigid, algorithm-based and scalable methodology, which is capable of producing valid results in a finite, predictable amount of time. More precisely, a defined background of the proposed approach to predict accurately the amount of time needed to obtain an acceptable solution. The PAIG rewriting method also contrasts to the state-space exploration (searching for the valid solution) involving, for example, various evolution-inspired techniques [15, 16].

The obtained experimental results indicate significant contribution in the field of synthesis of multi-functional circuits, that could potentially help to increase adoption of the polymorphic circuits for various application scenarios within the domain of multi-functional digital circuits. Research activities behind this contribution, including AIG extension for polymorphic circuits, open a new path for the synthesis of polymorphic circuits and create a stable basis for further research. This contribution may move the areas of synthesis and optimization polymorphic circuits forward significantly.

5.2 Future work

This thesis presents innovative design methods for multi-functional circuits. The thesis prepares a basis for a new direction in logic synthesis of multi-functional circuits in research area. Possible ways to continue this research are outlined in the following paragraphs:

A focus was given to verification of proposed principles mainly instead of development of optimized tools. Thus, a C code implementation of PAIG tool in order to speed up handling a graph may be the first task on the agenda.

Then, future work should be focused on exploration of the most frequent cuts, and preparation of an on-line available cut library in order to reduce the burden of time-consuming need to generate all the sub-graphs in an on-line manner, as proposed in [28].

Further focus on development of the proposed scheme should be aimed to technology mapping issues, i.e. translation from PAIG network structure to building components of a target technology. PAIG network can represent conventional AND gates, wires, inverters and polymorphic wires only, but real circuits are not expressed entirely in this way. Real circuits are composed of more complex gates, such as XOR or even polymorphic complex gates and thus it is supposed that a mapping to a target polymorphic technology may further shrink an area of desired polymorphic circuits.

List of Figures

List of Tables

Bibliography

[1] Berkeley Logic Syntehsis and Verification Group: ABC: A System for sequential synthesis and verification.
Retrieved from: https://people.eecs.berkeley.edu/~alanmi/abc/

[2] Akers, S. B.: Binary Decision Diagrams. *IEEE Transactions on Computers*. vol. C-27, no. 6. June 1978: pp. 509–516. ISSN 0018-9340.

[3] Alexander, C.; Sadiku, M.: *Fundamentals of Electric Circuits*. McGraw Hill Higher Education. fourth edition. 2008. ISBN 9780071284417.

[4] Armin, B.: Aiger format. In *Proceedings on Alpine Verification Meeting*. 2006. pp. 186–197.

[5] Biere, A.; Heljanko, K.; Wieringa, S.: AIGER 1.9 And Beyond. Technical report. FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria. 2011.

[6] Bobda, C.: *Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications*. The address: Springer. first edition. 2007. ISBN 978-1-4020-6088-5.

[7] Christoph, A.: IWLS 2005 Benchmarks.
Retrieved from: http://iwls.org/iwls2005/benchmarks.html

[8] Crha, A.; Růžička, R.; Šimek, V.: On the Synthesis of Multifunctional Logic Circuits. In *Abstracts Proceedings of International FLASH Conference*. Faculty of Electrical Engineering and Communication BUT. 2015. ISBN 978-80-214-5270-1. pp. 52–53.

[9] Crha, A.; Růžička, R.; Šimek, V.: Synthesis Methodology of Polymorphic Circuits Using Polymorphic NAND/NOR Gates. In *Proceedings on UKSim-AMSS 17th International Conference on Computer Modelling ans Simulation*. IEEE Computer Society. 2015. ISBN 978-1-4799-8713-9. pp. 612–617.

[10] Crha, A.; Růžička, R.; Šimek, V.: Novel Approach to Synthesis of Logic Circuits Based on Multifunctional Components. *Journal of Electrical Engineering*. vol. 67, no. 1. 2016: pp. 29–35. ISSN 1339-309X.

[11] Crha, A.; Růžička, R.; Šimek, V.: Toward Efficient Synthesis Method of Multifunctional Logic Circuits. In *Proceedings of the 27th International Conference on Microelectronics (ICM 2015)*. IEEE Computer Society. 2015. ISBN 978-1-4673-8759-0. pp. 21–24.

[12] Crha, A.; Růžička, R.; Šimek, V.: Synthesis Methodology of Polymorphic Circuits Using Polymorphic NAND/NOR Gates. In *Proceedings on UKSim-AMSS 17th International Conference on Computer Modelling ans Simulation*. IEEE Computer Society. 2015. ISBN 978-1-4799-8713-9. pp. 612–617.

[13] Crha, A.; Šimek, V.; Růžička, R.: Synthesis tool for design of complex polymorphic circuits. In *2017 12th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE Circuits and Systems Society. 2017. ISBN 978-1-5090-6376-5. pp. 149–154.

[14] Crha, A.; Šimek, V.; Růžička, R.: Towards novel format for representation of polymorphic circuits. In *13th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE Circuits and Systems Society. 2018. ISBN 978-1-5386-5290-9. pp. 1–2.

[15] Crha, A.; Šimek, V.; Růžička, R.: PAIG Rewriting: The Way to Scalable Multifunctional Digital Circuits Synthesis. In *2019 22nd Euromicro Conference on Digital System Design (DSD)*. Institute of Electrical and Electronics Engineers. 2019. ISBN 978-1-7281-2861-0. pp. 335–342.

[16] Crha, A.; Šimek, V.; Růžička, R.: KL-cuts influence on optimization of polymorphic circuits based on PAIG rewriting. In *2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. Institute of Electrical and Electronics Engineers. 2020. ISBN 978-1-7281-9938-2. pp. 1–6.

[17] Darringer, J.; Jr, W.; Berman, C.; et al.: Logic Synthesis Through Local Transformations. *IBM Journal of Research and Development*. vol. 25. 07 1981: pp. 272–280.

[18] Fiser, P.; Simek, V.: Optimum polymorphic circuits synthesis method. In *2018 13th International Conference on Design Technology of Integrated Systems In Nanoscale Era (DTIS)*. 04 2018. pp. 1–6.

[19] Fišer, P.; Háleček, I.; Schmidt, J.: SAT-Based Generation of Optimum Function Implementations with XOR Gates. In *2017 Euromicro Conference on Digital System Design (DSD)*. 2017. pp. 163–170.

[20] Gajda, Z.: Evolutionary Approach to Synthesis and Optimization of Ordinary and Polymorphic Circuits . In *PhD thesis*. UPSY FIT VUT Brno. 2011.

[21] Gajda, Z.; Sekanina, L.: On Evolutionary Synthesis of Compact Polymorphic Combinational Circuits. *Journal of Multiple-Valued Logic and Soft Computing*. vol. 17, no. 6. 2011: pp. 607–631. ISSN 1542-3980.
Retrieved from: https://www.fit.vut.cz/research/publication/9621

[22] Garey, M. R.; Johnson, D. S.: *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman and Co.. 1990. ISBN 0716710455.

[23] Hachtel, G.; Somenzi, F.: *Logic Synthesis and Verification Algorithms*. 01 2006. ISBN 978-0-387-31004-6.

[24] Hassoun, S.; Sasao, T.: *Logic Synthesis and Verification*. 01 2002. ISBN 978-1-4613-5253-2.

[25] Hayes, J. P.: *Introduction to Digital Logic Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.. first edition. 1993. ISBN 0201154617.

[26] Horowitz, P.; Hill, W.: *The Art of Electronics*. New York, NY, USA: Cambridge University Press. 1989. ISBN 0-521-37095-7.

[27] Lavagno, L.; Martin, G.; Scheffer, L.: *Electronic Design Automation for Integrated Circuits Handbook - 2 Volume Set*. USA: CRC Press, Inc.. 2006. ISBN 0849330963.

[28] Li, N.; Dubrova, E.: AIG rewriting using 5-input cuts. In *2011 IEEE 29th International Conference on Computer Design (ICCD)*. 2011. pp. 429–430.

[29] Li, Z.; Luo, W.; Yue, L.; et al.: Design Methods for Polymorphic Combinational Logic Circuits based on the Bi_Decomposition Approach. *CoRR*. 2017.

[30] Liang, H.; Luo, W.; Wang, X.: Designing Polymorphic Circuits with Evolutionary Algorithm Based on Weighted Sum Method. 2007: pp. 331–342.

[31] Liang, H.; Xie, R.; Chen, L.: Designing Polymorphic Circuits with Periodical Weight Adjustment. In *2015 IEEE Symposium Series on Computational Intelligence*. Dec 2015. ISSN null. pp. 1499–1505.

[32] Martinello, O.; Marques, F. S.; Ribas, R. P.; et al.: KL-Cuts: A new approach for logic synthesis targeting multiple output blocks. In *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*. 2010. pp. 777–782.

[33] Micheli, G. D.: *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education. first edition. 1994. ISBN 0070163332.

[34] Miller, J.; Job, D.; K. Vassilev, V.: Principles in the Evolutionary Design of Digital Circuits—Part I. *Genetic Programming and Evolvable Machines*. vol. 1. 04 2000: pp. 7–35.

[35] Miller, J. F.: *Cartesian Genetic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg. 2011. ISBN 978-3-642-17310-3. pp. 17–34.

[36] Mishchenko, A.; Brayton, R. K.: Scalable Logic Synthesis using a Simple Circuit Structure. In *Proceedings on the IWLS 2006*. 2006.

[37] Mishchenko, A.; Chatterjee, S.; Brayton, R.: DAG-aware AIG rewriting: a fresh look at combinational logic synthesis. In *2006 43rd ACM/IEEE Design Automation Conference*. July 2006. ISSN 0738-100X. pp. 532–535.

[38] Mishchenko, A.; Chatterjee, S.; Brayton, R. K.: Improvements to Technology Mapping for LUT-Based FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. vol. 26, no. 2. Feb 2007: pp. 240–253. ISSN 0278-0070.

[39] Mishchenko, A.; Steinbach, B.; Perkowski, M.: An algorithm for bi-decomposition of logic functions. In *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*. June 2001. ISSN 0738-100X. pp. 103–108.

[40] Morris, J.; Iniewski, K.: *Nanoelectronic Device Applications Handbook*. Devices, Circuits, and Systems. Taylor & Francis. 2013. ISBN 9781466565234.

[41] N, C.; Manjunath, k.: *VLSI CAD*. PHI Learning Pvt. Ltd.. ISBN 9788120342866.

[42] Nevoral, J.; Růžička, R.; Mrázek, V.: Evolutionary Design of Polymorphic Gates Using Ambipolar Transistors. In *2016 IEEE Symposium Series on Computational Intelligence*. Institute of Electrical and Electronics Engineers. 2016. ISBN 978-1-5090-4239-5. pp. 1–8.

[43] Nevoral, J.; Šimek, V.; Růžička, R.: PoLibSi: Path Towards Intrinsically Reconfigurable Components. In *2019 22nd Euromicro Conference on Digital System Design (DSD)*. Institute of Electrical and Electronics Engineers. 2019. ISBN 978-1-72812-861-0. pp. 328–334.

[44] Raza, H.: Graphene Nanoelectronics: Metrology, Synthesis, Properties and Applications. . 2012.

[45] Růžička, R.; Sekanina, L.; Prokop, R.: Physical Demonstration of Polymorphic Self-checking Circuits. In *Proc. of the 14th IEEE Int. On-Line Testing Symposium*. IEEE Computer Society. 2008. ISBN 978-0-7695-3264-6. pp. 31–36.

[46] Růžička, R.: Polymorphic electronics. In *Habilitation thesis*. DCSY FIT Brno University of Technology. 2011. page 118.

[47] Sekanina, L.: Evolutionary Design of Gate-Level Polymorphic Digital Circuits. vol. 3449. 2005: pp. 185–194.

[48] Sekanina, L.; Růžička, R.; Vašíček, Z.; et al.: REPOMO32 - New Reconfigurable Polymorphic Integrated Circuit for Adaptive Hardware. In *Proc. of the 2009 IEEE Symposium Series on Computational Intelligence - Workshop on Evolvable and Adaptive Hardware*. IEEE Computational Intelligence Society. 2009. ISBN 978-1-4244-2755-0. pp. 39–46.

[49] Steinbach, B.; Lang, C.: Exploiting Functional Properties of Boolean Functions for Optimal Multi-Level Design by Bi-Decomposition. *Artificial Intelligence Review*. vol. 20, no. 3. Dec 2003: pp. 319–360. ISSN 1573-7462.

[50] Stoica, A.; Zebulum, R.: Multifunctional logic gate controlled by temperature. In *NASA Tech Briefs*. California Institute of Technology. 2005. page 18. NPO-30795.

[51] Stoica, A.; Zebulum, R.; Keymeulen, D.: Polymorphic Electronics. In *Evolvable Systems: From Biology to Hardware*. Springer Berlin Heidelberg. 2001. ISBN 978-3-540-45443-4. pp. 291–302.

[52] Stoica, A.; Zebulum, R. S.; Keymeulen, D.: Polymorphic Electronics. In *Proceedings of the 4th International Conference on Evolvable Systems: From Biology to Hardware*. ICES '01. Berlin, Heidelberg: Springer-Verlag. 2001. ISBN 3-540-42671-X. pp. 291–302.

[53] Umans, C.: The Minimum Equivalent DNF Problem and Shortest Implicants. *Journal of Computer and System Sciences*. vol. 63, no. 4. 2001: pp. 597 – 611. ISSN 0022-0000.

[54] Šimek, V.; Růžička, R.; Crha, A.: Toward Efficient Synthesis Method of Multifunctional Logic Circuits. In *Proceedings of the 27th International Conference on Microelectronics (ICM 2015)*. IEEE Computer Society. 2015. ISBN 978-1-4673-8759-0. pp. 21–24.

[55] Wakerly, J.: *Digital Design: Principles and Practices*. Prentice Hall Series in Computer Engineering. Prentice Hall. 1994. ISBN 9780132114592.

[56] Yang, S.: Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0. Microelectronics Center of North Carolina (MCNC). jan 1991.

[57] Zhang, X.; Luo, W.: Evolutionary repair for evolutionary design of combinational logic circuits. *2012 IEEE Congress on Evolutionary Computation*. 2012: pp. 1–8.

[58] Zhang, X.; Luo, W.: Evolutionary design of polymorphic circuits with the improved evolutionary repair. In *2013 IEEE Congress on Evolutionary Computation*. June 2013. ISSN 1941-0026. pp. 2192–2200.

Curriculum Vitae

Education

Ph.D. of Computer Science and Engineering Faculty of Information Technology, Brno University of Technology Supervisor: doc. Ing. Richard Růžička, Ph.D., MBA	2013 – present
Master of Computer and Embedded Systems Faculty of Information Technology, Brno University of Technology Supervisor: Ing. Václav Šimek	2011 – 2013
Bachelor of Information Technology Faculty of Information Technology, Brno University of Technology Supervisor: Ing. Václav Šimek	2008 – 2013
High school Střední průmyslová škola elektrotechnická Kounicova 16, Brno	2004 - 2008

Visited conferences

2020
DDECS (Virtual symposium)

2019
Euromicro DSD (Prague, CZ)

2018
DTIS (Taormina, IT), PESW (Prague, CZ)

2017
DTIS (Palma de Mallorca, ES)

2016
DAC (Austin TX, USA)

2015
UKSim (Cambridge, GB), ICM (Casablanca, MAR)

2014-2016
PAD (CZ)

Projects

2020
Design, Optimization and Evaluation of Application Specific Computer Systems, BUT, FIT-S-20-6309, 2020-2022

2017
Advanced parallel and embedded computer systems, BUT, FIT-S-17-3994, 2017-2019

2015
Innovation of technical equipment for multilayer printed circuit boards, BUT, FEKT/FIT-J-15-2832, 2015
Multifunctional Circuit Components Based on Hybrid Organic Electronics, BUT, FCH/FIT-J-15-2652, 2015-2016

2014
Architecture of parallel and embedded computer systems, BUT, FIT-S-14-2297, 2014-2016
Unconventional Design Techniques for Intrinsic Reconfiguration of Digital Circuits: From Materials to Implementation, MŠMT CR, LD14055, 2014-2017

Teaching

Microprocessors and Embedded Systems - *labs*
Personal Computers - *labs*
Practical Aspects of Software Design - *labs*
Personal Computers Architecture - *labs*
Review of bachelor and master thesis
Supervising of bachelor and master thesis:

- Laser Plotter for Cutting Steel Plates, Master Thesis: Dokulil Marek, Ing.
- Testing Framework for Automatic Tests of MCUXpresso Config Tools, Bachelor Thesis: Dubovský Tomáš, Bc.
- Automatic Regression Tests for SDK and CMSIS, Bachelor Thesis: Svoboda Tomáš
- The Automatic Update Mechanism of Software Packages in Cloud Distribution System, Bachelor Thesis: Willaschek Tomáš, Bc.
- Racing Motorcycle On-Board Computer, Master Thesis: Dokulil Marek, Bc.

Work experience

2014 – 2015
HW/SW developer at Identcode s.r.o.

2016 – present
Software engineer at NXP Semiconductors

Publications

2020	
Crha Adam, Šimek Václav, Růžička Richard: Vliv řezů s více výstupy na optimalizaci polymorfních obvodů s využitím techniky PAIG přepisování, In: 2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), Novi Sad, IEEE, 2020, p. 1-6, ISBN 978-1-7281-9938-2	50 %
2019	
Crha Adam, Šimek Václav, Růžička Richard: PAIG Rewriting: The Way to Scalable Multifunctional Digital Circuits Synthesis, In: 2019 22nd Euromicro Conference on Digital System Design (DSD), Kallithea, Chalkidiki, IEEE, 2019, p. 335-342, ISBN 978-1-7281-2861-0	60 %
2018	
Crha Adam, Šimek Václav, Růžička Richard: Towards novel format for representation of polymorphic circuits, In: 13th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS), Taormina, IEEE CAS, 2018, p. 1-2, ISBN 978-1-5386-5290-9	55 %
2017	
Crha Adam, Šimek Václav, Růžička Richard: Synthesis tool for design of complex polymorphic circuits, In: 2017 12th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS), Palma de Mallorca, IEEE CAS, 2017, p. 149-154, ISBN 978-1-5090-6376-5	65 %
Šimek Václav, Nevoral Jan, Crha Adam, Růžička Richard: Towards Design Flow for Space-Efficient Implementation of Polymorphic Circuits Based on Ambipolar Components, In: ElectroScope, roč. 11, č. 1, Plzeň, CZ, p. 1-10, ISSN 1802-4564	20 %
2016	
Crha Adam: Polymorfní elektronika a metody syntézy, In: Počítačové architektury a diagnostika Česko-slovenský seminář pro studenty doktorského studia, Brno, FIT VUT, 2016, p. 93-97, ISBN 978-80-214-5376-0	100 %
Crha Adam, Růžička Richard, Šimek Václav: Novel Approach to Synthesis of Logic Circuits Based on Multifunctional Components, In: Journal of Electrical Engineering, roč. 67, č. 1, Berlin, DE, p. 29-35, ISSN 1339-309X	40 %
Otáhal Alexandr, Šimek Václav, Crha Adam, Růžička Richard, Szendiuch Ivan: Innovative Methods in Activation Process of Through-hole Plating, In: Periodica Polytechnica Electrical Engineering and Computer Science, roč. 60, č. 4, Budapest, HU, p. 217-222, ISSN 2064-5279	10 %

Otáhal Alexandr, Šimek Václav, Crha Adam, Růžička Richard, Szendiuch Ivan: Innovative Methods for Through Holes Plating, In: 2016 39th International Spring Seminar on Electronics Technology (ISSE), Plzeň, IEEE CS, 2016, p. 48-52, ISBN 978-1-5090-1389-0,	10 %
Šimek Václav, Stříteský Stanislav, Řezníček Michal, Crha Adam, Růžička Richard: Towards Implementation of Logic Circuits Based on Intrinsically Reconfigurable Organic Transistors, In: Proceedings of the 6th Electronics System-Integration Technology Conference, Grenoble, IEEE, 2016, p. 1-6, ISBN 978-1-5090-1401-9	5 %
Tesař Radek, Šimek Václav, Růžička Richard, Crha Adam: Design of Polymorphic Operators for Efficient Synthesis of Multifunctional Circuits, In: Journal of Computer and Communications, roč. 4, č. 15, Wuhan, CN, p. 151-159, ISSN 2327-5227	5 %
Šimek Václav, Tesař Radek, Růžička Richard, Crha Adam: Modelling and Physical Implementation of Ambipolar Components Based on Organic Materials, In: Proceedings of the 28th International Conference on Microelectronics (ICM 2016), Cairo, IEEE CAS, 2016, p. 341-344, ISBN 978-1-5090-5721-4	5 %
2015	
Crha Adam, Růžička Richard, Šimek Václav: Synthesis Methodology of Polymorphic Circuits Using Polymorphic NAND/NOR Gates, In: Proceedings on UKSim-AMSS 17th International Conference on Computer Modelling ans Simulation, Cambridge, IEEE CS, 2015, p. 612-617, ISBN 978-1-4799-8713-9	50 %
Šimek Václav, Růžička Richard, Crha Adam: Toward Efficient Synthesis Method of Multifunctional Logic Circuits, In: Proceedings of the 27th International Conference on Microelectronics (ICM 2015), Casablanca, IEEE CS, 2015, p. 21-24, ISBN 978-1-4673-8759-0	40 %
Crha Adam, Růžička Richard, Šimek Václav: On the Synthesis of Multifunctional Logic Circuits, In: Abstracts Proceedings of International FLASH Conference, Brno, FEKT VUT, 2015, p. 52-53, ISBN 978-80-214-5270-1	40 %
Šimek Václav, Růžička Richard, Crha Adam, Řezníček Michal, Buršík Martin: Reconfigurable Digital Circuits Based on Chip Expander with Integrated Temperature Regulation, In: Journal of Computer and Communications, roč. 3, č. 11, Wuhan, CN, p. 169-175, ISSN 2327-5227	30 %
2014	
Šimek Václav, Růžička Richard, Crha Adam, Tesař Radek: Implementation of a Cellular Automaton with Globally Switchable Rules, In: 11th International Conference on Cellular Automata for Research and Industry, ACRI 2014, Cham, Springer Science+Business Media, 2014, p. 378-387, ISBN 978-3-319-11519-1,	10 %
Tesař Radek, Šimek Václav, Růžička Richard, Crha Adam: Polymorphic Electronics Based on Ambipolar OFETs, In: EDS 2014 IMAPS CS International Conference Proceedings, Brno, VUT v Brně, 2014, p. 106-111, ISBN 978-80-214-4985-5,	20 %