Inversion Optimization in Majority Inverter Graph with Minority Operations

A Novel Approach

MS(Research) Thesis

By

Umar Aalam

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

June 2020

Inversion Optimization in Majority Inverter Graph with Minority Operations

A Novel Approach

A THESIS

Submitted in fulfillment of the

requirements for the award of the degree

of

Master of Science (Research)

By

Umar Aalam

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

June 2020

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **Inversion Optimization in Majority Inverter Graph with Minority Operations** in the fulfillment of the requirements for the award of the degree of **MASTER OF SCIENCE(RESEARCH)** and submitted in the **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the time period from July 2018 to June 2020 under the supervision of Dr. Bodhisatwa Mazumdar, Assistant Professor, Indian Institute of Technology Indore, India and Dr. Neminath Hubballi, Associate Professor Indian Institute of Technology Indore, India.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Umar Aalam
29-06-2020

Signature of the Student with Date
(Umar Aalam)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Bodhisatwa Mazumdar 14/10/2020

Signature of Thesis Supervisor with Date
(Dr. Bodhisatwa Mazumdar)

14-10-2020

Signature of Thesis Supervisor with Date
(Dr. Neminath Hubballi)

Umar Aalam has successfully given his MS(Research) Oral Examination held on 14/10/2020

Bodhisatwa Mazumdar 14/10/2020

14/10/2020

Signature of Chairperson, OEB with date

14-10-2020

Signature(s) of Thesis Supervisor(s) with date

14-10-2020

Signature of Convener, DPGC with date

Somnath Dey 14/10/2020

Signature of Head of Discipline with date

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my heartfelt gratitude to a number of persons who in one or the other way contributed by making this time as learnable, enjoyable, and bearable. At first, I would like to thank my supervisors, **Dr. Bodhisatwa Mazumdar**, **Dr. Neminath Hubballi**, who were a constant source of inspiration during my work. Without their constant guidance and research directions, this research work could not be completed. Their continuous support and encouragement have motivated me to remain streamlined in my research work.

I am thankful to **Dr. Abhishek Srivastava**, **Dr. Manish Kumar Goyal** and **Dr. Ram Bilas Pachori**, my research progress committee members, took some valuable time to evaluate my progress all these years. Their valuable comments and suggestions helped me to improve my work at various stages. I am also grateful to the HOD of Computer Science **Dr. Surya Prakash** for his help and support.

My sincere acknowledgment and respect to **Prof. Neelesh Kumar Jain**, Director(Officiating), Indian Institute of Technology Indore for providing me the opportunity to explore my research capabilities at Indian Institute of Technology Indore.

I want to express my heartfelt respect to my parents for their love, care, and support they have provided to me throughout my life. Special thanks to my elder brother Mr. Zafar Alam who is my constant source of inspiration. This thesis would not have been possible without the help of their support and encouragement. I would like to appreciate the fine company of my dearest colleagues and friends, especially Nazish Kalam, Rabindra Patel, Akshay Thakare, Shivam, Neha, Shubham, Pratik, Siddharth, Anagha, Pararthi, Sadaf Ali.

I am also grateful to the institute staffs for their unfailing support and assistance and the Ministry of Electronics and Information Technology for funding the MS(Research). Finally, I am thankful to all who directly or indirectly contributed, helped, and supported me. To sign off, I write a quote by Albert Einstein:
"If you can't explain it simply, you don't understand it well enough."

Umar Aalam

To my family and friends

Abstract

On account of their simplicity, homogeneous logic representations have drawn attention in the domain of logic synthesis. In such representations, combinational circuits are often modeled as Directed Acyclic Graph (DAG) wherein terminal nodes represent primary inputs and other nodes represent same Boolean logic operation. The edges in DAGs can be complemented where the nodes have a complemented output, which renders a functionally universal representation. Such homogeneous logic representations simplify optimization algorithms significantly, hence, assist in efficient implementation.

This thesis performs an analysis into the inversion optimization of a homogeneous logic representation, referred to as *majority inverter graph* (MIG) using *minority* operations. MIG is a logic representation structure that along with its algebraic properties and Boolean transformation methods synthesizes circuits with significant optimization in terms of size, depth, and switching activity. In this work, we first propose rules and properties for logic representations with *minority* and *inversion* operations referred to as mIG logic synthesis. Subsequently, we propose synthesis techniques based on *minority*, *majority*, and *inversion* operations, referred to as mMIG synthesis. We propose an algorithm based on the rules and properties to achieve an optimization in terms of area reduction and inversion reduction in mMIG-based circuit synthesis. We present comparison of resource consumption in mMIG logic synthesis with existing MIG synthesis techniques, and present some important observations towards inverter optimization.

These propsed methods have been tested on various standard benchmark circuits, such as EPFL benchmark suites, ISCAS'85 and also on lightweight cryptographic block ciphers and cryptoprimitives. Analysis of the implementation results demonstrates that the proposed $mMIG$ structure outperforms the existing MIG structure mainly in term of reduction in inversion count which can have major impact in emerging technologies such as *quantum dot cellular automata* (QCA) and *spin wave devices* (SWDs), and inversion intensive circuits such as RAM address-decoding circuits.

Keywords: Minority logic, Inversion, Boolean algebra, Majority logic, Inversion Optimization.

LIST OF PUBLICATIONS

1. "*mMIG: Inversion Optimization in Majority Inverter Graph with Minority Operations*", under revision in Elsevier VLSI Integration.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The present day advancement of digital electronic circuits is substantially attributed to the strong coupling between *electronic design automation* (EDA) tools and *complementary metal oxide semiconductor* (CMOS) technology. Over the years, circuits with higher density and performance enhancement were enabled due to the continuous downscaling CMOS technology [3]. To progress at the same rate, present day EDA tools are challenged to handle both ever-increasing circuit topologies with growing functionality and emerging device models with increasing complexity.

Present day EDA tools are fine tuned for CMOS device technology; the underlying optimization methodologies are tuned to CMOS logic primitives, such as NAND-NOR or AND-Inverter instances. Such logic primitives are considered to be the ultimate building blocks over the next decade [4], however, literature does not demonstrate any evidence that are the optimal basis for any EDA software. In the timeline of EDA, the design metrics have repeatedly shifted along with technology changes, such as from cost of transistors to need for speed, and a subsequent shift to reduced power and energy consumption. An efficient way to implement an optimal hardware design depends on the way logic functions are represented in EDA tools.

In present day world, relentless evolution of electronic devices and products is threatened by physical limitations of conventional CMOS technology to further scale down transistor size. To satisfy futuristic constraints and sustain Moore's prediction, recent research is focusing its attention on emerging nanotechnologies as candidates

for replacing traditional CMOS. Nanotechnologies, however, often carry different logic primitives, for instance, *majority* and XOR, for which contemporary logic synthesis techniques are not fully satisfactory from multiple performance metrics. We attempt to address the following evolving question: "*Can present day logic synthesis techniques demonstrate better performance metrics if based on new, different, logic primitives?*". In this thesis we address the performance metrics of inverter minimization, which incur a huge overhead in designs, such as RAM address decoding circuit.

1.1 Logic Synthesis in EDA

Logic synthesis is a key component in EDA tools, as logic functions are extracted from high level programming languages (for e.g. C, C++), and *hardware description languages* (HDLs) (for e.g. Verilog/VHDL). Optimization of such extracted logic functions is crucial to achieve efficient implementations. Due to the vastness of problem size and plurality of choices, present day logic optimization has been a daunting task, which has necessitated design automation. Over the years, logic synthesis has progressed through combining theoretical results and engineering practices. The idea of design optimality has been obscured by factors, such as circuit complexity, delay, and power consumption. In absence of precise objective function of the mentioned factors and prevalence of large design space, tool flows have largely been dependent on heuristics. Present day logic synthesis techniques aim to push the limits of performance throughput upwards, and power consmption downwards. These factors have been compounded by other metrics, such as, testability, reliability, and area reduction.

Typical EDA tools constitute front-end and back-end phases that comprise algorithms and methods used to design complex electronics systems. As shown in Figure 1.1, taking a *high level description* (HDL) of an electronic circuit design, an EDA flow proceeds through several logic abstractions and generates an implementation comprising primitives from the technology library [1]. The main steps comprising the flow are *high level synthesis*, *logic synthesis*, and *physical design*. *High level synthesis* transforms a programming language description or HDL into *register transfer language* (RTL)

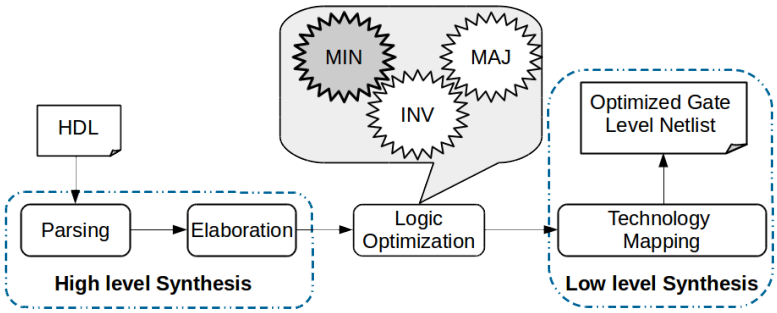

Figure 1.1: Logic Optimization in front-end and back-end phases of EDA design flow.

netlist. The *logic synthesis* performs optimization and maps a circuit from RTL netlist into *standard cells* (ASICs) or *look up tables* (FPGAs).

The *low level synthesis* phase performs *place* and *route* operations wherein physical resources are assigned to the mapped logical elements followed by *routing* which interconnects the placed logic elements. The focus of this work is on logic optimization in the logic synthesis phase. Conventional primitive gates used in logic synthesis are AND, OR, and INV. Present day logic optimization methods exploit local functionality of primitive gates and the topology over which the circuit is described. For instance, two-level AND-OR logic circuits, also called *sum-o-products* (SOPs) work on optimizing the cubes (inherently AND functions) and their sum (OR function) [5].

In multi-level logic representations, logic gates exhibit an unbounded functionality, implying that each element can represent an arbitrary logic function. However, these logic elements are represented as SOP polynomials, which are factorized into AND/ORs [6]. However, present day research focus on new compact representations of logic functions, such as *majority* and *inverter* operations [2], which are combined together with powerful manipulation and combination techniques. The state-of-the-art design tools, while performing optimization in terms of area or depth reduction of the circuit, make extensive use of such *homogeneous logic circuits*.

1.2 Motivation

Present day EDA tools encounter challenges that are harder than ever as design sizes in CMOS technology approach the ultimately achievable limit. Analysis of new logic primitives can enhance the potential of logic synthesis and formal verification tools that presently exist in CMOS technology. New logic primitives can enable such tools reach points in design space that were not accessible earlier [7].

Further, research of new logic primitives is necessitated due to the emergence of new nanotechnologies that offer enhanced functionality over conventional (CMOS) switches [8]. In logic optimization, the circuit representation is manipulated to minimize a certain performance metric, such as, area (number of nodes or elements), logic depth (maximum number of levels), or interconnections (number of nets or wires), etc. In the EDA timeline, the standard logic representation has moved across various schemes, such as, sum-of-product (SOP) form, product-of-sums (POS) form, directed acyclic graph (DAG) representation [9]. Most of the recent technologies such as *Spin Wave Devices* (SWD) [10], *Quantum-dot Cellular Automata* (QCA) [11], and resistive random access memories (RRAM) [12] are based on majority logic and present scope of optimal implementation. In fault-tolerant computing paradigm, the cost models illustrate that majority gate can be implemented at the same cost as AND/OR gate [13]. In other words, the present day EDA tools necessitate emerging logic synthesis techniques and abstractions to attain chosen performance metrics [14]. An approach towards synthesis of RRAM-based logic circuits using MIGs was proposed in [15]. In [16], the memristive behavior of RRAM is captured as a majority based logic operation for efficient synthesis of logic-in-memory circuits and systems.

In [2], a logic representation structure called *majority inverter graph* (MIG) was proposed which comprises three-input majority nodes and complemented edges, thus forming a directed acyclic graph structure. MIG-based logic synthesis methods have demonstrated superior performance efficiency for both standard CMOS and emerging technologies [2]. The work proposed Boolean algebra involving majority and inversion operations that led to strong optimization depicted by reduction in average delay, area

and, power over multiple academic and industrial benchmark circuits. In addition, the MIG synthesis technique and the optimizer (MIGHty) employs inherent properties of MIGs, such as bit error masking in circuit optimizations. For arithmetic benchmark circuits, the synthesis technique enables 16% depth reduction in LUT-6 circuits compared to implementations in ABC synthesis tool. With the advent of majority gates with larger fan-in in emerging technologies, the work in [17] proposed novel majority logic synthesis flow for majority gates with arbitrary number of inputs. Optimization methods of MIGs in sequential circuits was first examined in [18]. The work demonstrated an improvement of 9% and 38% in area and delay performance metrics as compared to ABC tool.

In MIGs, inversion operations are represented as complemented edges (CEs). Emerging technologies that are built using only majority and inverter operations demonstrate that minimization of inverters plays an important role as area and delay cost metrics depend heavily on the number of inversion operations in the circuit [19, 20]. In [20], minimization of complemented edges relied on recursive application of inverter propagation axiom to move complemented edges on the inputs to the respective output. A set of transformation rules without affecting the depth and the size of MIG was applied on all majority nodes of the circuit. The work demonstrated how the minimization of complemented edges affected circuit performances in Spin Wave Devices (SWD) and Quantum Cellular Automata (QCA) based technology.

In this work, we demonstrate how minority operation along with majority and inverter graph (mMIG) representation for a Boolean function can aid to MIG based logic synthesis for improved reduction in inverter count. We first propose Boolean algebra for minority inverter graph (mIG) and mMIG synthesized circuits, followed by multiple case studies of performance improvement for MIG synthesized circuits with minority operations on design and implementation of basic functional blocks that form the core of datapath elements in ISCAS'85 benchmark circuits, EPFL benchmark circuits, and circuits embedded in crypto-processor designs.

1.3 Background

In automated EDA design flow, multilevel logic synthesis plays an important role towards finding Boolean functions with better quality while considering different cost functions. The cost functions comprise number of logic gates, logic depth, and switching activity targeted towards better area, performance, and energy, respectively [21]. To address this issue, efficient representation and optimization of Boolean functions are key features [7]. For instance, compact logic representations aim to contain minimal number of primitive elements, such as, literals, sum-of-product terms, and gates to have a small hardware footprint requiring as few library elements as possible.

Conventional logic representation forms comprise two level forms, such as, product-of-sums (POS) and sum-of-product (SOP) forms. Over the years, such representation forms have been augmented by efficient heuristic and exact optimization algorithms. With rapid progress in VLSI technology platforms, standard logic representations moved towards directed acyclic graph (DAG) representations where nodes represent logic gates, and directed edges represent the wires connecting the nodes.

The work in this work is based on [2] that depicted how majority and inversion operations through a set of transformation methods can be used to derive MIG algebraic optimization methods. The set of transformation methods were based on a set of axioms and derived theorems of Boolean algebra, such as commutativity, associativity, distributivity, and inverter propagation. The MIG representation used other inherent properties of majority function, such as bit-error masking, to further optimize the synthesized circuits. The experimental results demonstrated that the circuit optimized MIG representations could achieve depth reduction of up to 7% in circuits implemented in LUT-6 technology by the ABC tool [22]. The transformations also reduced circuit size as well as power activity when compared to similar AIG optimization. In addition to reducing the average delay, area and power in standard CMOS circuits, MIG based circuit synthesis has also explored multiple majority operator-based nanotechnologies [23]. This has led to a renewed interest in circuits synthesized in majority operations followed by optimization [24, 25], producing competitive results in CMOS

ASICs and Field Programmable Gate Arrays (FPGAs).

For beyond-CMOS technologies, such as TFET and SpinFETs, physical implementation of inverter is more expensive than majority operation, which motivates us to aim to reduce inverter count in circuit implementations [24]. Furthermore, emerging technologies, which employ majority and inversion operations for circuit representations, area and delay cost of the respective implementations have significant dependence on the inverter count in the circuit [20]. Hence, in such technologies, minimization of number of inverters is a big challenge. The work in [20] used inverter propagation across two and higher levels to reduce the overall inverter count. However, these methods have limitations where the number of inverters in the smaller subtrees are small. Circuits that have predominantly NAND or NOR logic gates over a large number of smaller subtrees around leaf nodes can be one such instance. **In this work, we demonstrate how an optimization method based on minority, majority operations and interconnecting complemented edges (CEs) can lead to a significant reduction in the number of inverter count.**

We first demonstrate the derived theorems of Boolean algebra for mIG circuit representations, followed by various subclasses of derived theorems. This is followed by case studies of various combinatorial circuits used in various applications. However, we state here that our optimizations assume that the standard cell library implementation of minority and majority operations consume equivalent amount of resources in terms of area and delay. Furthermore, in present-day FPGA platforms, such as SPARTAN-6 device family, an LUT6 (six-input lookup table) hardware primitive enables implementation of eight majority, or eight minority gates, or a combination of eight majority/minority gates in the same LUT, as both majority and minority logic mapping consume the same amount of resources when they are mapped to an LUT implementation.

1.4 Thesis Contributions

A brief overview of our research contributions is provided below, and more details are available in the later chapters.

Contribution I:

We propose a set of rules for Boolean algebra in minority inverter graph (`mIG`) and minority-majority-inverter graph (`mMIG`) representation of a circuit.

Contribution II: We have explored the use of proposed rules. For each set of rules, we demonstrate the class of rules where `mIG` and `mMIG` representations yield better performance as compared to `MIG` representation in terms of reducing the count of inversion operations (demonstrated as a reduction in `CEs` in the circuit graph).

Contribution III:

We have proposed novel `mMIG` logic optimization algorithms for size and depth optimization of the circuit, which comprises reshaping algorithm, inverter propagation algorithm, etc.

Contribution IV:

We present case studies of various combinatorial logic circuits in EPFL, ISCAS'85 benchmark circuits, and lightweight cryptographic primitives, which demonstrate minority operation along with majority and inversion operations yield better resource optimization in terms of reduction in `CEs` as compared to only majority and inversion operations.

Contribution V:

We further demonstrate circuit instances which, when implemented with minority, majority, and inversion operations (we refer to it as mMIG synthesized circuits) yield reduced cost in terms of inverter count (and hence reduced net delay in the critical path), and smaller signal switching power as compared to MIG circuit representations.

Contribution VI:

We propose the logic synthesis optimization flow in `mMIG` synthesis as shown in Figure 1.2. Considering an input circuit in `MIG` synthesis, the optimization steps for

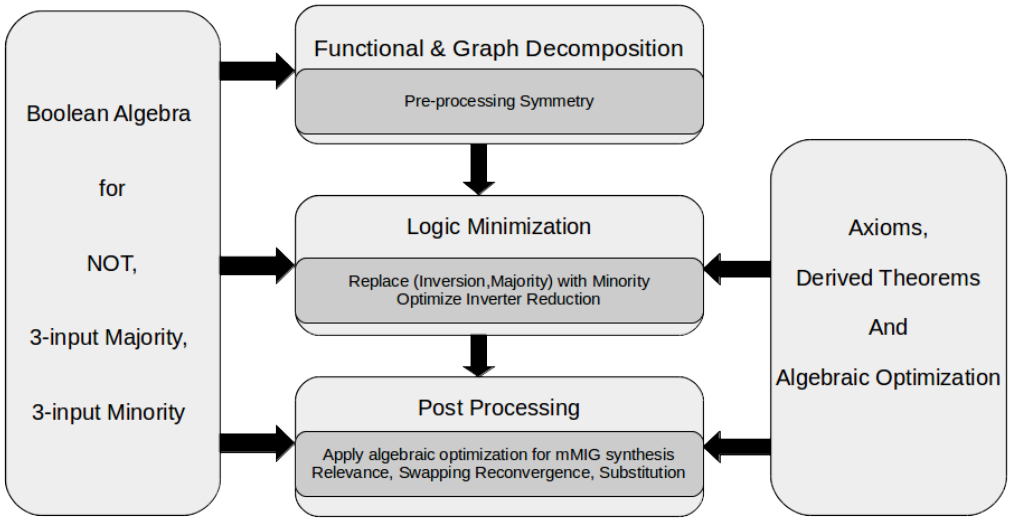

Figure 1.2: Logic synthesis optimization flow involving mMIG based Boolean algebra.

`mMIG` synthesis proceeds through the following three phases:

(i) *Functional and Graph Decomposition*: The MIG topology is partitioned into logic cones as determined by the outputs of the circuit, and substitute majority nodes driving a complemented edge (`CE`) with a minority node. Each logic cone is further split into smaller partitions which comprise only connected majority nodes, only connected minority nodes, and groups comprising interconnected majority and minority nodes.

(ii) *Logic Minimization*: In this phase, we apply size optimization of each sub-partition by reducing the complementary edges (`CEs`) through inverter propagation and replacement of majority node and immediate inversion with minority gate, wherever possible in each partition.

(iii) *Post-Processing*: We apply a set of axioms and proposed derived theorems, such as, *relevance*, *reconvergence swapping*, and *substitution* on the `mMIG` logic obtained from the logic optimization phase to achieve reduction in both, complemented edges (`CEs`) and size.

1.5 Organization of the Thesis

This thesis is organized into five chapters. A summary of each chapter is provided below:

Chapter 1 (Introduction)

This chapter describes the background knowledge of logic synthesis techniques, the motivation of our work, and the contribution of this thesis.

Chapter 2 (Literature Survey and Research Methodology)

This chapter provides a detailed literature survey and a summary of various logic representation techniques and data structure in the logic synthesis technique.

Chapter 3 (Reducing Inversion Overhead with Minority Operations)

In this chapter, we propose a set of transformative algebraic rules for *minority inverter graph* (`mIG`), representation properties, and Boolean algebra corresponding to minority logic data structure. Subsequently, we present a *minority-majority inverter graph* (`mMIG`), its representation properties, and transformative algebraic rules followed by the mMIG logic optimization algorithm, which majorly comprises reshaping algorithm, INV propagation algorithm, mMIG size optimization algorithm.

Chapter 4 (Experimental Results: Case Study and Comparison)

In this chapter, we have shown a comparative assessment of the logic optimization algorithm that we proposed for mMIG implementations with corresponding MIG implementations. It contains experimental results of standard combinational circuits from EPFL benchmark suites, ISCAS'85 benchmark suites and combinational circuits that form the core of certain cryptographic primitives in both private and public key cryptography.

Chapter 5 (Conclusions and Future Work)

This chapter briefly describes the contribution of this thesis and the possible future directions of our work.

Chapter 2

Literature Survey

Since its origin, the overall problem of logic synthesis has been centered around finding the "best implementation of a logic circuit". The term "best" often depends on the goals, such as circuit complexity, delay, and/or power consumption for combinational logic synthesis. The initial approach to address this problem dwelt upon *sum-of-product* (SOP) representations, which focused on reducing the cardinality of product terms called *implicants* or *logic covers*. Typical representations of SOPs, such as *programmable logic arrays* (PLAs) had rectangular shapes with rows corresponding to product terms. Hence, reducing the count of product terms imply area reduction as well. The first logic synthesis algorithm, *Quine-McCluskey* algorithm solves for exact minimization of logic cover [26]. With usage of advanced data structures, the algorithm solved most benchmarks. To reduce computing time, several heuristic minimization approaches of two level forms were preferable over exact minimization [1]. Such an instance is ESPRESSO, which yielded irredundant covers with near-optimal size [27].

On similar lines, logic synthesis research has been extensively devoted to *exclusive-sum-of-product* (ESOP) minimization [28]. Similar to ESPRESSO, single-output or multiple-output functions are optimized as an intermediate step of logic synthesis flow. In the domain of quantum computing circuits, ESOP minimization is important as ESOP forms can be transformed into cascaded Toffoli gates providing a reversible logic solution. In 1980s, with the advent of CMOS technology and semicustom design

styles, scientific research emerged in contemporary logic synthesis.

The initial problem comprised transforming logic functions into "best" interconnections of primitives of technology library. As a result, many solution efforts partition synthesis into technology-independent phase, wherein interconnections between blocks are minimized irrespective of the underlying technology. This was followed by a technology mapping step where primitives from library are chosen. With ever-changing technology specifications, recent efforts addressing the problem, "*What should be the maximum dimension of a logic block to achieve an optimum realization subjected to specific constraints?*". Optimality in this case may imply minimum area, i.e., sum of area consumed by selected library cells [29, 30, 31], or minimum delay [32, 33], which is the critical path delay of the circuit, which can be computed once the cells are chosen and verified after physical design phase. The optimality problem can also be defined in terms of satisfiability (`SAT`). With availability of increasingly powerful computer resources, increasingly larger optimal circuits (targeting area or delay) can be computed. However, given the large design space, the main issue is the practicality of such an approach for very large circuits. Such present-day circuits comprise millions of `NAND`-equivalent gates. Nevertheless, *divide-and-conquer* approach is applied by decomposing logic networks into logic blocks, and logic blocks are synthesized using exact methods.

In addition, the "optimal" implementations of functional blocks are put up as library primitives; the primitives can be instantiated by logic synthesis algorithms at runtime. This design flow helps restraining from possible blow-up in size of Boolean function representations, hence forms underlying input to optimization algorithms. In the next section, we present logic representations that are used to capture efficiently logic circuits as interconnection of blocks and assist heuristic-based and exact method-based optimization methods.

2.1 Logic Representation

One of the most important factors that determine efficient hardware implementations is the way that logic functions are represented in EDA tools. The two important factors that determine the structure of logic representation are as follows:

(i) Logic representations aim to comprise least number of *primitive elements*, for e.g., literals, sum-of-product terms, nodes in a circuit graph in order to have small hardware footprint and require as less number of library elements as possible.

(ii) Logic representations must be simple enough to have smaller rules for optimization.

Typical data structures, such as *two-level representations* [1] or *binary decision diagrams* (BDDs) [34, 35], are chosen for representations depending upon the trade-off between compactness and ease of optimization.

In the initial years of EDA, the standard logic representation form was the *sum-of-products* (SOP) form, i.e, a disjunction (OR) of conjunctions (AND) of literals [36]. This form was used in *programmable logic array* (PLA) technology wherein the functionality was implemented as SOP [37]. Other *two-level forms* that were analysed comprise *product-of-sums* (POS) and EX-SOP [28]. Two-level logic exhibits compactness for small logic functions only, beyond which it is hard to efficiently map to silicon. Nevertheless, two-level logic representation has been efficient heuristic and exact optimization algorithms.

With advent of technology scaling and emerging VLSI technologies, standard logic representation shifted from two-level logic to *directed acyclic graphs* (DAGs). In DAG-based representation, nodes denote logic functions of gates, and directed edges represent wires connecting the nodes. The nods can have representations in SOPs which leverage the efficiency of two-level forms. In general, each node is assigned a typical function while an edge can be a regular or a complemented one. A prominent example of DAG where each node realize the same function is *binary decision diagram* (BDDs) [38]. In a BDD, a node is a 2×1 multiplexer. Another instance of such a

DAG is the *AND-OR-Inverter Graph* (AOIG), wherein nodes represent two-input ANDs. Circuit topologies in AIG can be optimized by using conventional Boolean algebra and derived theorems. In AIGs, local transformations render small resource footprint through various logic optimization rules. Hence, AOIGs have become one of the present representation standards for logic synthesis.

Another node functionality that possess enhanced expressiveness and have more varied optimization strategies as compared to conventional AOIGs, hence, leading to superior synthesis results are the *majority* nodes. Such homogeneous structures enable *algebraic rewriting*, which reshape circuit partitions to reduce the number of nodes and levels [39]. A circuit designer can build up a database of precomputed circuit topologies of a function. For a particular topology, the designer can compute the corresponding function, and ascertain whether replacing the topology with a precomputed topology leads to an improvement.

2.2 Logic Optimization

Logic optimization entails transforming a logic representation to minimize a a performance metrics. Some usual performance metrics comprise size (number of nodes), depth (maximum number of levels), interconnections (number of edges/complemented edges). Such optimization algorithms are strongly coupled to the data structures on which they are run. For instance, minimization of two-level SOPs can be achieved by reducing the number of terms. Such minimization is achieved by the program ESPRESSO [27], which incorporates both, an optimal implementation of *Quine Mc-Cluskey* algorithm [26] for *exact minimization*, and fast near-optimal heuristics [40]. Further, minimization through *exact method* in BDDs was attained in [41] and through heuristics.

Over the decades, various approaches have been taken towards logic optimization, which comprise *algebraic methods* that are based on polynomial algebra [42, 43], *algebraic rewriting* based on algebraic axioms and theorems [39], and *Boolean methods* based on Boolean algebra [44, 45, 46]. In all these approaches, heuristics are applied

to select the variant and sequence of transformations.

As this thesis deals with DAG based data structures, we mention certain aspects of DAG-based logic optimization methods. Typically such optimizations are classified into two groups: *algebraic methods* which are fast and *Boolean methods* which are slow but may achieve better results [47]. Conventional algebraic methods assume that the DAG nodes are transformed into SOP form and consider them as polynomial expressions [1, 48]. The algebraic operations are applied on each DAG node until further optimization is not feasible. The basic algebraic operations comprise *extraction*, *decomposition*, *factoring*, *substitution*, and *balancing* [47, 43].

In comparison to algebraic methods, Boolean methods consider the functions on the basis of their Boolean nature and *don't cares* to achieve a better solution [46, 49, 50]. *Functional decomposition* is another Boolean method which represents the original function by smaller component functions [51]. A more scalable approach to functional decomposition is BDD decomposition that can be applied recursively while exploiting optimization opportunities, which are not achievable by algebraic methods [52, 53, 54]. It is to mention that main hindrance in developing Boolean algorithms is due to unlimited design space of choices.

Logic optimization results demonstrate that DAG nodes in data structure, such as, AOIG, when subjected to one type of logic transformations, such as, *balancing*, *refactoring*, and *rewriting* have reinforced improved results [39, 48, 55]. In *algebraic rewriting*, AIG subgraphs are replaced with equivalent AIG implementations through a combination of *balancing* and *refactoring* logic transformation steps. Repeated application of local and powerful transformation steps often lead to reduction in depth levels and area. However, researchers observed that Boolean methods can still yield better performance as compared to AIG optimization in many circuit topologies instances [56].

In this thesis, the work is based on the basic operations of *majority* (MAJ) and *inversion* (INV) nodes. We use the term inversion operation and *complemented edge* (CE) interchangeably. As shown in Figure 2.1, the *majority function* is at the center of Boolean function classification [1].

We state some existing Boolean an algebraic optimization techniques of the data

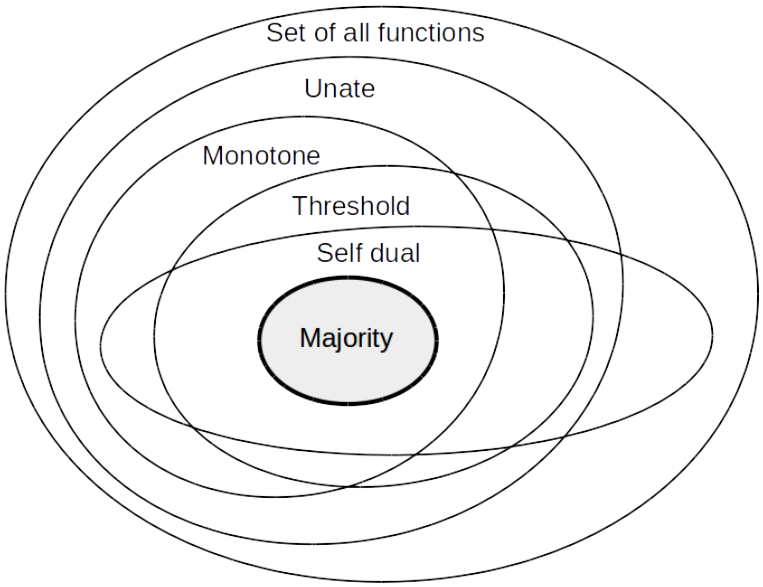

Figure 2.1: Majority function in the center of Boolean function classification [1].

structure MIG and how such techniques enable reaching to new points in the design space.

2.3 Preliminaries

In this thesis, we consider the set of Boolean functions, $\mathbb{B}^n \mapsto \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$ denotes the set of binary Boolean values. The conjunction (AND) and disjunction (OR) operators are represented by the symbols $\wedge$ and $\vee$, respectively. The inversion (INV) operator is denoted by $'$ operator and 0/1 represent teh false/true logic values. The standard Boolean algebra operates on the set $(\mathbb{B}, \wedge, \vee, ')$. Boolean algebra comprises the operations *identity*, *commutativity*, *associativity*, *distributivity*, and *complement* axioms over $\wedge$, $\vee$, and $'$ operators.

The axiomatization of Boolean algebra defined by the logic arguments and axioms are valid (*soundness*) and provable (*completeness*) [57]. A logic circuit is a *directed acyclic graph* (DAG), wherein nodes refer to logic functions, and directed edges denote interconnections between the nodes. The direction of edges in such DAGs indicate the flow of computation from the inputs to the outputs. In this thesis, we refer to

logic function and logic circuit interchangeably. A logic circuit is *homogeneous* if each logic node represents the same logic function, and edges can possess a regular or complemented attribute. The *depth* of a node is defined as the length of the longest path from any primary input to that node. The *depth* of the logic circuit is the largest depth among all nodes. The size of a logic circuit is the number of nodes in the circuit.

The majority function is a logic function which falls in the class of *self-dual* function as shown in Figure 2.1. A logic function $f(a, b, \ldots, c)$ is self-dual if $f = f'(a', b', \ldots, c')$. By the property of *complementation*, *self-dual* function can also be formulated as $f' = f(a', b', \ldots, c')$. Hence, the *majority function* defined as $f(a, b, c) = (a \wedge b) \vee (b \wedge c) \vee (c \wedge a)$ is a self-dual function [1].

2.3.1 Majority Function

The *majority function* M on n inputs (n being odd) returns the truth value taken by more than half of the inputs. The function M returns the truth value *one* if k number of input variables ($k \geq \lceil \frac{n}{2} \rceil$) are equal to *one*. For instance, the three-input majority function can be represented as $M(x, y, z) = (x \wedge y) \vee (y \wedge z) \vee (x \wedge z)$. Moreover, $(x \vee y) \wedge (y \vee z) \wedge (x \vee z)$ is also a valid representation for M. Based on majority and inversion operations, *majority-inverter-graph* (`MIG`) data structure have been proposed in [2]. In the next section, we revisit the representation properties and demonstrate the Boolean algebra that fits in the `MIG` data structure.

2.3.2 Majority Inverter Graphs

A *majority inverter graph* (`MIG`) is a homogeneous logic circuit, wherein each node represents a majority function with an indegree of 3; the edges in the graph can be a regular or complemented. A majority operation defined as $M(x, y, z)$ works as *conjunction* operation, `AND(x,y)` when $z = 0$, and as *disjunction* operation `OR(x,y)` when $z = 1$. An MIG representation for the logic circuit $f = (x_3 \vee (x_2 \wedge (x_1' \wedge x_0)'))$ is shown in Figure 2.2. The conventional `AOIG` circuit is first taken as the input, which comprises of `AND`, `OR`, and `INV` operations. The `INV` operation is shown as a

complemented edge (CE) in the figure. The circuit is transformed into MIG circuit by replacing the AND (OR) operations with *majority* (MAJ) nodes, wherein one input is set to *zero* (*one*) truth value.

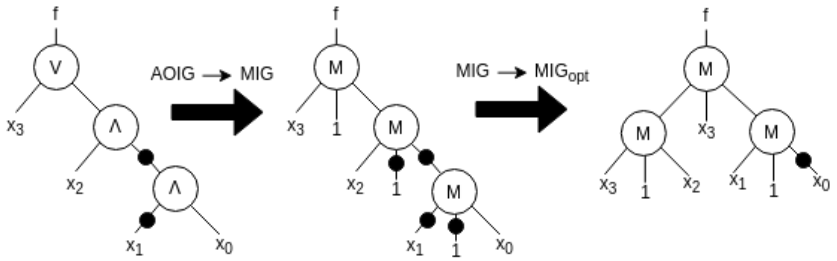

Figure 2.2: AOIG to MIG representation for the circuit, $f = (x_3 \vee (x_2 \wedge (x_1' \wedge x_0)'))$, and further MIG depth optimization. The operator $\wedge$ and $\vee$ represent AND and OR operations, respectively.

Moving a one step further in the figure, applying *distributivity* property on two lower nodes in MIG leads to depth reduction of the circuit. We next state the MIG Boolean algebra based on which logic transformations and optimizations are carried out.

2.3.3 MIG Boolean Algebra

The MIG Boolean algebra is defined over the set $(\mathbb{B}, M, ', 0, 1)$ where M is the *majority* operator. In this thesis, we use "$\bar{}$" and "$'$" notations for inversion operation interchangeably. The five primitive transformation rules, referred to as Ω, form an axiomatic system as follows,

(i) *Commutativity*, $\Omega.C$: $M(x, y, z) = M(x, z, y) = M(y, x, z)$

(ii) *Majority*, $\Omega.M$: $M(x, y, x) = M(x, x, z) = x$, $M(x, y, x') = y$

(iii) *Associativity*, $\Omega.A$: $M(x, u, M(y, u, z)) = M(y, u, M(x, u, z)) = M(z, u, M(y, u, x))$

(iv) *Distributivity*, $\Omega.D$: $M(x, y, M(u, v, z)) = M(M(x, y, u), M(x, y, v), z)$

(v) *Inverter Propagation*, $\Omega.I$: $M(x'y', z') = M'(x, y, z)$

In addition to these axioms, several other properties or theorems were derived from Ω in $(\mathbb{B}, M, ', 0, 1)$. This set of rules were referred to as Ψ and are described hereafter,

(i) *Relevance*, $\Psi.R$: $M(x, y, z) = M(x, y, z_{x/y'})$

(ii) *Complementary Associativity*, $\Psi.C$: $M(x, u, M(y, u', z)) = M(x, u, M(y, x, z))$

(iii) *Substitution*, $\Psi.S$: $M(x, y, z) = M(v, M(v', M_{v/u}(x, y, z), u), M(v', M_{v/u'}(x, y, z), u'))$

In the above set of rules, the symbol $z_{x/y}$ denotes *replacement* operation, i.e., x is replaced with y in all occurrences of z. The *relevance* rule replaces *reconvergent* variables with neighboring variables. For instance, in function $f = M(x, y, M(w, z', M(x, y, z)))$, variables x and y are *reconvergent* as they occur in both lower and upper majority operations. The *relevance* property replaces x with y' in the lower majority operation as, $f = M(x, y, M(w, z', M(y', y, z)))$, which further reduces to $M(x, y, w)$.

The *complementary associativity* rule involves variables which are present in opposite polarities as shown by variable u in the equation. The *substitution* rule applies to variable replacement in non-reconvergent case. All the rules in Ψ can be derived from Ω [2]. The set $(\mathbb{B}, M, ', 0, 1)$ along with axioms in Ω and derivable theorems in Ψ form the complete majority logic system. The primary data structure for $(\mathbb{B}, M, ', 0, 1)$ is an `MIG` and the associated logic transformation tools are axioms from Ω and derived theorems from Ψ. It is possible to transform an `MIG`-based implementation of a circuit, α, into another logically equivalent `MIG`-based circuit β, through a sequence of transformations from Ω and Ψ [2] .

The work in [2] use mathematical theory [58] to define a consistent logic optimization framework. It presented experimental evidence on the benefits predicted by the proposed theory. Results demonstrate significant depth reduction in logic circuits implemented in `MIG` as compared to state-of-the-art logic synthesis techniques, such as `AOIG`.

2.4 Significance of MIG based Synthesis

In recent years, optimization in logic synthesis have received quite an attention by researchers [21, 17, 18]. Chu et al. [21] have added exclusive-OR(XOR) operations with MIG structure and proposed two identities which shows circuit optimization in XOR-MIG, referred to as XMG, whose experimental results on EPFL benchmark suites has shown optimization in depth and size. As XOR logic gate occupies more space than minority logic so we have decided to go with the second one. On using mMIG structure to represent any XMG structure is always having added advantage in terms of especially inversion count.

[59], [60] introduced the network synthesis tool that worked for majority logic as well as minority logic. Recent nano-technologies, such as *tunneling phase logic (TPL)* and *single electron tunneling (SET)* are solely based on minority logic whereas *quantum cellular automata (QCA)* is based on majority logic. It has also shown the effectiveness of this approach compared with very popular Boolean synthesis approach. It leads to an idea to synthesize the large network using both, majority logic gate and minority gate which will cover all the recent technologies. It has compared the quality of the network of majority gates by counting the gates and number of levels.

In automated EDA design flow, multilevel logic synthesis plays an important role towards finding Boolean functions with better quality while considering different cost functions. The cost functions comprise number of logic gates, logic depth, and switching activity targeted towards better area, performance, and energy, respectively [21]. To address this issue, efficient representation and optimization of Boolean functions are key features [7]. For instance, compact logic representations aim to contain minimal number of primitive elements, such as, literals, sum-of-product terms, and gates to have a small hardware footprint requiring as few library elements as possible.

Conventional logic representation forms comprise two level forms, such as, product-of-sums (POS) and sum-of-product (POS) forms. Over the years, such representation forms have been augmented by efficient heuristic and exact optimization algorithms. With rapid progress in VLSI technology platforms, standard logic representations

moved towards directed acyclic graph (DAG) representations where nodes represent logic gates, and directed edges represent the wires connecting the nodes.

The work in this thesis is based on [2] that depicted how majority and inversion operations through a set of transformation methods can be used to derive MIG algebraic optimization methods. The set of transformation methods were based on a set of axioms and derived theorems of Boolean algebra, such as commutativity, associativity, distributivity, and inverter propagation. The MIG representation used other inherent properties of majority function, such as bit-error masking, to further optimize the synthesized circuits. The experimental results demonstrated that the circuit optimized MIG representations could achieve depth reduction of up to 7% in circuits implemented in LUT-6 technology by the ABC tool [22]. The transformations also reduced circuit size as well as power activity when compared to similar AIG optimization. In addition to reducing the average delay, area and power in standard CMOS circuits, MIG based circuit synthesis has also explored multiple majority operator-based nanotechnologies [23]. This has led to a renewed interest in circuits syntheiszed in majority operations followed by optimization [24, 25], producing competitive results in CMOS ASICs and Field Programmable Gate Arrays.

For beyond-CMOS technologies, such as *tunnel field effect transistor* (TFET) and *spin polarized tunnel field effect transistor* (Spin-FETs), physical implementation of inverter is more expensive than majority operation, which motivates us to aim to reduce inverter count in circuit implementations [24]. Furthermore, emerging technologies, which employ majority and inversion operations for circuit representations, area and delay cost of the respective implementations have significant dependence on the inverter count in the circuit [20]. Hence, in such technologies, minimization of number of inverters is a big challenge. The work in [20] used inverter propagation across two and higher levels to reduce the overall inverter count. However, these methods have limitations where the number of inverters in the smaller subtrees are small. Circuits that have predominantly NAND or NOR logic gates over a large number of smaller subtrees around leaf nodes can be one such instance. It has also been shown that the minimization cannot be applied on every node of MIG structure so in order to deal

with these issues we have proposed several algorithms.

In this thesis, we demonstrate how an optimization method based on minority, majority operations and interconnecting complemented edges (CEs) can lead to a significant reduction in the number of inverter count.

We first demonstrate the derived theorems of Boolean algebra for `mIG` circuit representations, followed by various subclasses of derived theorems. This is followed by case studies of various combinatorial circuits used in various applications. However, we state here that our optimizations assume that the standard cell library implementation of minority and majority operations consume equivalent amount of resources in terms of area and delay. Furthermore, in present-day FPGA platforms such as SPARTAN6 device family, a single LUT6 (*six-input lookup table*) hardware primitive enables implementation of eight majority, or eight minority gates, or a combination of eight majority/minority gates in the same LUT, as both majority and minority logic mapping consume the same amount of resources when they are mapped to a LUT implementation.

Chapter 3

Reducing Inversion Overhead with Minority Operations

In emerging technologies, circuits comprising *majority* (MAJ) and *complemented edges* (CEs), minimization of *inverters* plays a crucial role since area and delay costs depend on the count of CEs in such circuits. Inversion minimization has a profound impact on emerging technologies, such as *spin wave devices* (SWDs), *Quantum-dot cellular automata* (QCAs), and *resistive random access memories* (RRAMs) [20].

In this chapter, we present transformation rules and instances, wherein *minority* mode instead of *majority* node and its output CE leads to improved reduction in CEs in MIG-based circuits. Our *minority*-based circuit synthesis also aids in circuits, which comprise millions of NAND/NOR-based design of logic gates whose direct implementation using minority function requires no inversion operation. However, such reduction is not possible when such implementations are performed using only majority and inversion operations.

Transformation rules are the most effective way to achieve optimization in the graph-based logic circuits. In this thesis, several rules are introduced in *minority inverter graph* (mIG) and *minority majority inverter graph* (mMIG) synthesis to show its logical representation and properties, which helps to achieve the compact implementation of the circuit. The proposed logical transformations in mIG includes theorems derivable from Boolean algebra axioms. The proposed set of rules comprises:

(i) *associativity rules* $(\Omega'.A_{nrv}, \Omega'.A_{n^2}v)$ demonstrate the similar nature of adjacent Boolean variable with two variants, one rule between *reconvergent* and *non-convergent* variable, and another between two *non-reconvergent* variables.

(ii) *distributivity rule* $(\Omega'.D)$ for reducing three-level cascaded *minority* nodes to a two-level topology.

(iii) derived theorems that comprise *swapping reconvergence* rules $(\Psi'.SR^1_{n^2v}, \ldots, \Psi'.SR^4_{n^2v}), (\Psi'.SR^1_{nrv}, \ldots, \Psi'.SR^4_{nrv})$ where *reconvergent* Boolean variables are swapped with other *reconvergent* or *nonreconvergent* Boolean variables. This rule is important for reducing logical depth when a Boolean variable (both *reconvergent* and *non-reconvergent*) is present in a critical path.

(iv) *Relevance* rule $(\Psi'.R)$ and *Substitution* rule $(\Psi'.S)$ enlarges the circuit so that other rules and theorems when applied results in optimized circuit in terms of area and depth through iterated absorption and elimination transformations.

In `mMIG`, we apply Boolean algebraic methods in a heuristic fashion, which mainly includes operations, such as *push-up*, *push-down*, swapping Boolean variable (*reconvergent* with *reconvergent* or *nonreconvergent* with *nonreconvergent* or *reconvergent* with *nonreconvergent*), which leads to functionally equivalent optimal structure comprising less number of inverters.

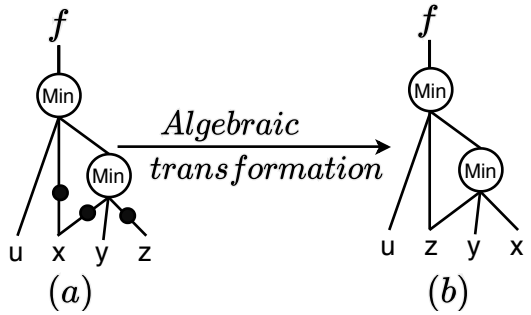

Figure 3.1: An example to demonstrate role of algebraic transformation in `mMIG` for inverter reduction of the function $f = m(u, \bar{x}, m(\bar{x}, y, \bar{z}))$. The transformation is obtained by swapping *reconvergent* Boolean variable x with *non-reconvergent* Boolean variable z of the bottom most node, which removes all four inverters, hence, significant area reduction as well.

The proposed set of rules for `mMIG` comprises:

(i) *Elimination rule* ($\Psi_{mMIG}.El$) removes redundant nodes of `mMIG`.

(ii) *Absorption rule* ($\Psi_{mMIG}.Ab_1, \Psi_{mMIG}.Ab_2$) absorbs or combines a *minority* node and a *majority* operation and replaces them with either a *majority* or a *minority* node.

(iii) *Swapping reconvergence rule* ($\Psi_{mMIG}.SR_1, \ldots, \Psi_{mMIG}.SR_3$) swaps *reconvergent* and *non-reconvergent* variables in both regular and complemented forms.

(iv) *Relevance rule* ($\Psi_{mMIG}.R_1, \ldots, \Psi_{mMIG}.R_8$) is applied when two *reconvergent* variables occur in two consecutive levels; as both reconvergent variables are in upper level, each occurrence of one is replaced by the complemented form of the other variable in the lower variable. This renders the third variable in lower level relevant, and hence moves to the upper level.

3.1 Introduction to Minority logic

In MIG-based circuit synthesis, the property *inverter propagation* fails to reduce the inverter count if the connectivity of CEs is small but count of CEs is significantly large. Two CEs are connected if they CEs have a common *majority* node. This small connectivity property motivates us to propose *minority* node for reducing inverter count.

Several case studies show a significant reduction in inverter count by using only *minority logic*. One such motivational instance can be seen in Section 4.4, where $XOR3$ implemented using mIG consumes less number of inverter operations as compared to that in MIG implementation. This property can be observed from the generic equation of XOR_n as we show later in our case studies in next chapter. Another such simple yet motivational circuit instances can be observed in the Figure 3.2. A minority logic circuit is a homogeneous logic network with an indegree of 3, where each node represents a minority logic function,

$$min(x, y, z) = \bar{x}\bar{y} + \bar{y}\bar{z} + \bar{x}\bar{z} \tag{3.1}$$

The operation yields a logical one output when there are two or more logical zeros at its inputs x, y, z. The minority function is *self-dual*, similar to majority function [2], which can be represented in *conjunctive normal form* (CNF) and *disjunctive normal form* (DNF) as:

$$min(x, y, z) = \bar{x}\bar{y} \vee \bar{y}\bar{z} \vee \bar{x}\bar{z} = (\bar{x} \vee \bar{y})(\bar{y} \vee \bar{z})(\bar{x} \vee \bar{z}) \tag{3.2}$$

Further, the minority function deduces to one Boolean input value if any two input Boolean variables either always take same values or hold complemented values of each other as,

$$min(x, x, z) = \bar{x} \quad min(x, \bar{x}, z) = \bar{z} \tag{3.3}$$

In terms of representation, minority logic involves only a minority operator, which

makes it a universal operator that can be used for the synthesis of any basic logical operations and can thus produce an *inverter*, an `OR` gate, or an `AND` gate. For instance, some of the operations can be derived as follows: `NAND`: $\overline{xy} = min(x, y, 0)$, `AND`: $xy = min(min(x, y, 0), 0, 1)$, `NOR`: $\overline{x + y} = min(x, y, 1)$, `OR`: $x + y = min(min(x, y, 1), 0, 1)$, and `NOT`: $\bar{x} = min(x, 0, 1)$. From the gate realizations, *minority* logic yields far more optimal implementations in case of `NAND-NOR` based standard cell library. The universal property of *minority* operator leads us to the following theorem.

Theorem 1. *Minority logic gate is a universal logic gate.*

Proof. In an analogy to the existing *majority inverter graph* (`MIG`), a minority operator is an inversion of majority operation. If all three inputs are subjected to the same input value, it operates as an *inversion* operation. Hence, a *minority* logic gate can implement both *majority* and *inversion* operations, thus reaching all points in representation space as that by an `MIG` data structure. □

We consider a logic function as $f = (\overline{x_1 + (\overline{x_2(\overline{x_3 + x_4})})})$, where $\bar{x}$ denotes an inversion operation on x and $+$ denotes a logical OR operation. Fig 3.2 represents *majority inverter graph* (`MIG`) representation followed by *minority inverter graph* (`mIG`)

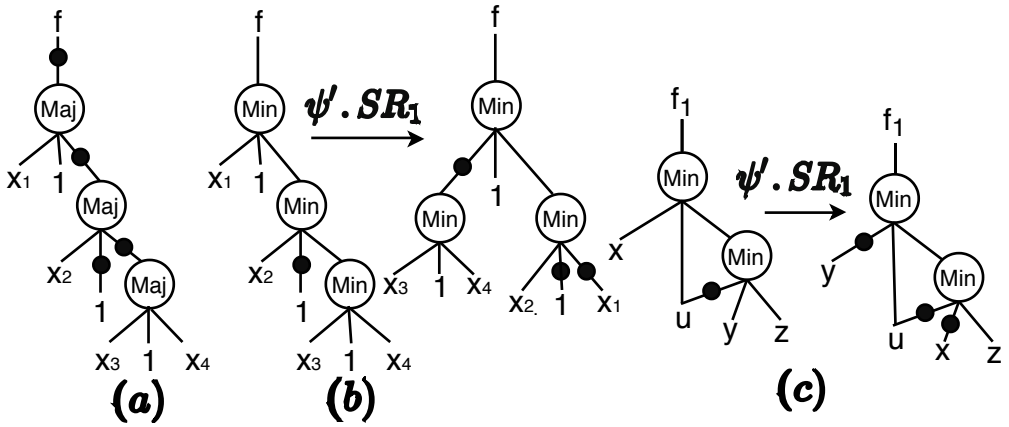

Figure 3.2: (a) `MIG` representation of $f = (x_1 + (x_2 \cdot (x_3 + x_4)')')'$ and, (b) its equivalent `mIG` representation followed by optimized version of `mIG` leading to depth optimized circuit, (c) `mIG` representations that demonstrate one form of *Swapping reconvergence* ($\psi'.SR$) to implement such optimizations. *Inversion* operations at the output of a node are represented as *complemented edges*, i.e., edges with bubbles.

for this logic function leads to a reduction of three *inversion* operations. The `mIG` representation can be further optimized for depth reduction, as shown in the figure with the `mIGOpt` version using one form of *swapping reconvergence* as an optimization rule.

In continuation, we now propose a set of Boolean algebra rules corresponding to *minority logic* data structure, algebraic optimization rules to achieve compact implementation in `mIG` synthesis. For this purpose, we have introduced a new Boolean algebra similar to [2] and propose following optimization rules.

3.2 Optimization Rules in Minority Logic

In this section, we propose transformation rules based on *minority* function in the form of `mIG` Boolean algebra and then propose derived theorems which includes variants of the primitive transformation. We state here that if the technology library contains only *minority* and *inverter* standard cells, the equivalent *minority* representation for *inversion* operation, $\bar{x} = min(x, x, x) = min(x, 0, 1)$, incurs a larger hardware footprint in implementation. Hence, we consider both minority and inversion operations in our circuit representation system.

3.2.1 `mIG` Boolean Algebra

We propose a novel Boolean algebra, defined over the set $(\mathbb{B}, m, 0, 1)$, where m is the *minority* node with three input Boolean variables. The *minority logic* has five primitive transformation rules which comprises the possible variants of these transformation rules as mentioned below.

- **Commutativity**, $\boldsymbol{\Omega'.C}$: $m(x, y, z) = m(y, x, z) = m(z, y, x)$.
- **Minority**, $\boldsymbol{\Omega'.m}$:
  if(x==y), $m(x, x, z)$=$m(y, y, z)$=$\bar{x} = \bar{y}$, else if(x==$\bar{y}$), $m(x, \bar{x}, z) = \bar{z}$.
- **Associativity**, $\boldsymbol{\Omega'.A}$: We define two variants of $\boldsymbol{\Omega'.A}$, namely, *associativity of two non-reconvergent variables* ($\boldsymbol{\Omega'.A_{n^2v}}$), and *associativity of a non-*

reconvergent variable and a neighboring reconvergent variable ($\boldsymbol{\Omega'.A_{nrv}}$) as follows,

  - $\boldsymbol{\Omega'.A_{n^2v}}$:
    $\bar{m}(x,u,\bar{m}(u,y,z))$=$\bar{m}(z,u,\bar{m}(u,y,x))$=$\bar{m}(y,u,\bar{m}(u,x,z))$.
  - $\boldsymbol{\Omega'.A_{nrv}}$: $m(\bar{x},u,m(u,y,z))$=$m(\bar{u},x,m(x,y,z))$.

- **Distributivity**, $\boldsymbol{\Omega'.D}$:
  $m(\bar{x},\bar{y},m(u,v,z))$=$m(m(x,y,u),m(x,y,v),\bar{z})$.

- **Inverter Propagation**, $\boldsymbol{\Omega'.I}$: $m'(x,y,z) = m(\bar{x},\bar{y},\bar{z})$.

These five transformation rules in Ω' depicts the behavior of minority function over the three input Boolean variables namely, x,y and z. Axiom $\Omega'.C$ shows the commutative property of the minority operator. Axiom $\Omega'.A$ defines an associative property over three input parameter. Axiom $\Omega'.D$ exhibits the distributive property of minority operator, which will be later helpful to eliminate the *minority* node in the circuit. In Axiom $\Omega'.I$, *inversion propagation* of minority operation is stated, which demonstrates inverter reduction in circuit topology when two or more inputs of minority node have `CE` or, the output and more than one input of *minority node* has `CE`.

3.3 Derived Theorems

In this subsection, we propose a set of derived theorems for `mIG` synthesis technique as Ψ'. Firstly, we mention the derived theorems in `MIG` synthesis that involved shared Boolean variables with only opposite polarities (0/1) or replacement of reconvergent variables. In `mIG` synthesis, we observe that in addition to opposite polarities, shared variables with same polarities (0/0 or 1/1) have their own respective properties. We state these properties as derived theorems in this section.

3.3.1 Swapping Reconvergence rule in mIG

In MIG, *complementary associativity* $\Psi \cdot C$ involves shared Boolean variables with opposite polarities in two consecutive levels. On the other hand, in mIG, a similar case involves swapping shared variables, defined as $\Psi'.SR$, with both opposite polarities as well as same polarities across two consecutive levels. The selection of opposite or same polarities of shared variables to be swappped depends on the circuit topology.

We define two variants of $\Psi'.SR$, namely, *swapping between non-reconvergent and reconvergent variable* ($\Psi'.SR_{nrv}$) and *swapping between two non-reconvergent variables* ($\Psi'.SR_{n^2v}$). The circuits obtained by applying both these variants are functionally equivalent. Consider the circuit instances in Figure 3.3 and Figure 3.4. If t_x, t_u, t_y, t_z are the logic delay of the circuits driving the nodes x, u, y, and z, respectively, selection of either of these variants is determined with respect to the optimal implementation of the circuit, which depends on the following conditions. If $t_u > max(t_x, t_y, t_z)$ then select $\Psi'.SR_{nrv}$, else if $t_z > max(t_x, t_u, t_y)$ then we select $\Psi'.SR_{n^2v}$ property. Upon applying the variants of the *Swapping reconvergence*, position of the reconvergent and the nonreconvergent critical variables get changed, hence, it reduces the inverter count if the propagation is from right to left or vice-versa, as observed in Figure 3.3 and Figure 3.4.

The transformation rules in the *swapping reconvergence* category are as follows:

$$\Psi'.SR^1_{n^2v} : m(x, u, m(\bar{u}, y, z)) \leftrightarrow m(\bar{z}, u, m(\bar{u}, \bar{x}, y)) \leftrightarrow m(\bar{y}, u, m(\bar{u}, \bar{x}, z))$$

Proof.

$$m(x, u, m(\bar{u}, y, z)) = m(m(\bar{x}, \bar{u}, \bar{u}), m(\bar{x}, \bar{u}, y), \bar{z})(\boldsymbol{\Omega'.D})$$
$$m(m(\bar{x}, \bar{u}, \bar{u}), m(\bar{x}, \bar{u}, y)) = m(\bar{z}, u, m(\bar{x}, \bar{u}, y))(\boldsymbol{\Omega'.m})$$

□

Depending on the inversion operation of the independent variables, $\Psi'.SR^1_{n^2v}$ trans-

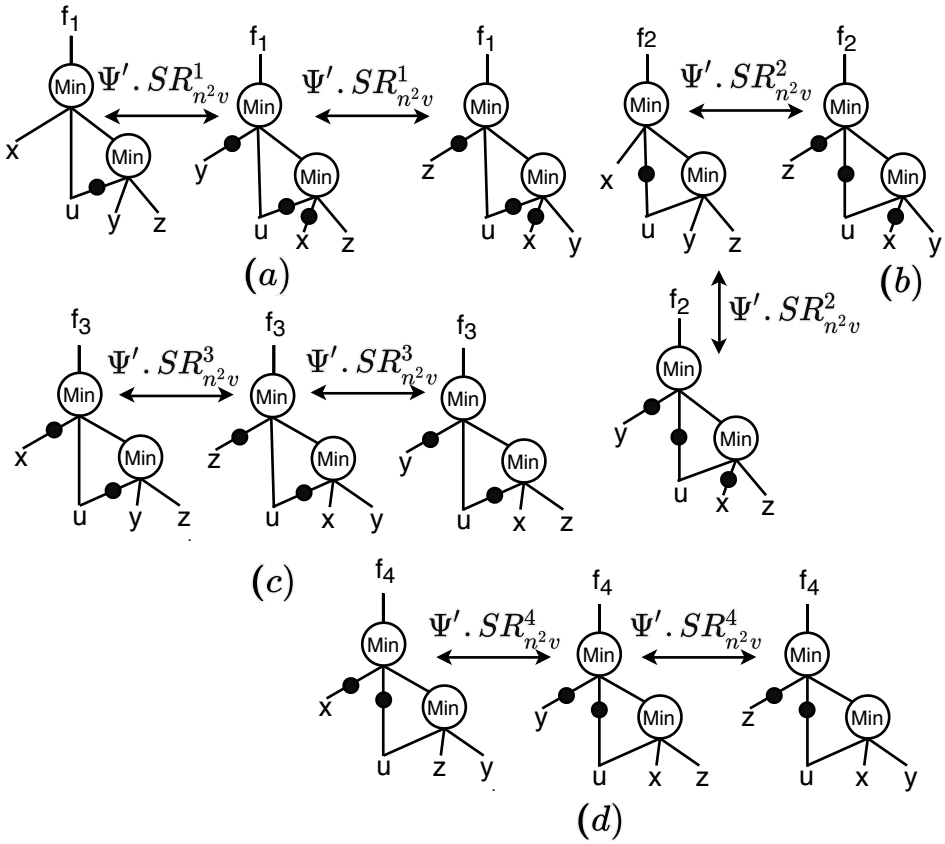

Figure 3.3: The circuits in (a), (b), (c), and (d) depict *swapping reconvergence* property($\Psi'.SR_{n^2v}$) in `mIG` structure wherein a *nonreconvergent* Boolean variable is swapped with another *nonreconvergent* variable as a result of which one nonreconvergent variable moves one level up towards the primary output.

forms to the following rules in *Swapping Reconvergence*,

$$\Psi'.SR^2_{n^2v} : m(x, \bar{u}, m(u, y, z)) \leftrightarrow m(\bar{z}, \bar{u}, m(u, \bar{x}, y)) \leftrightarrow m(\bar{y}, \bar{u}, m(u, \bar{x}, z))$$

Proof.

$$m(x, \bar{u}, m(u, y, z)) = m(m(\bar{x}, u, u), m(\bar{x}, u, y), \bar{z})(\boldsymbol{\Omega'.D})$$

$$m(m(\bar{x}, u, u), m(\bar{x}, u, y), \bar{z}) = m(\bar{z}, \bar{u}, m(u, \bar{x}, y))(\boldsymbol{\Omega'.m})$$

□

$$\Psi'.SR^3_{n^2v} : m(\bar{x}, u, m(\bar{u}, y, z)) \leftrightarrow m(\bar{z}, u, m(\bar{u}, x, y)) \leftrightarrow m(\bar{y}, u, m(\bar{u}, x, z))$$

Proof.

$$m(\bar{x}, u, m(\bar{u}, y, z)) = m(m(x, \bar{u}, \bar{u}), m(x, \bar{u}, y), \bar{z})(\boldsymbol{\Omega'.D})$$
$$m(m(x, \bar{u}, \bar{u}), m(x, \bar{u}, y), \bar{z}) = m(\bar{z}, u, m(\bar{u}, x, y))(\boldsymbol{\Omega'.m})$$

□

$$\Psi'.SR^4_{n^2v} : m(\bar{x}, \bar{u}, m(u, y, z)) \leftrightarrow m(\bar{y}, \bar{u}, m(u, x, z)) \leftrightarrow m(\bar{z}, \bar{u}, m(u, x, y))$$

Proof.

$$m(\bar{x}, \bar{u}, m(u, y, z)) = m(m(x, u, u), m(x, u, z), \bar{y})(\boldsymbol{\Omega'.D})$$
$$m(m(x, u, u), m(x, u, z), \bar{y}) = m(\bar{y}, \bar{u}, m(u, x, z))(\boldsymbol{\Omega'.m})$$

□

Further, we propose that $\Psi'.SR$ property also applies between non-reconvergent and reconvergent variables as follows,

(i) $\Psi'.SR^1_{nrv}$: $m(x, u, m(\bar{u}, y, z)) \leftrightarrow m(u, x, m(x, y, z))$.

(ii) $\Psi'.SR^2_{nrv}$: $m(x, \bar{u}, m(u, y, z)) \leftrightarrow m(\bar{u}, x, m(x, y, z))$.

(iii) $\Psi'.SR^3_{nrv}$: $m(\bar{x}, u, m(\bar{u}, y, z)) \leftrightarrow m(u, \bar{x}, m(\bar{x}, y, z))$.

(iv) $\Psi'.SR^4_{nrv}$: $m(\bar{x}, \bar{u}, m(u, y, z)) \leftrightarrow m(\bar{u}, \bar{x}, m(\bar{x}, y, z))$.

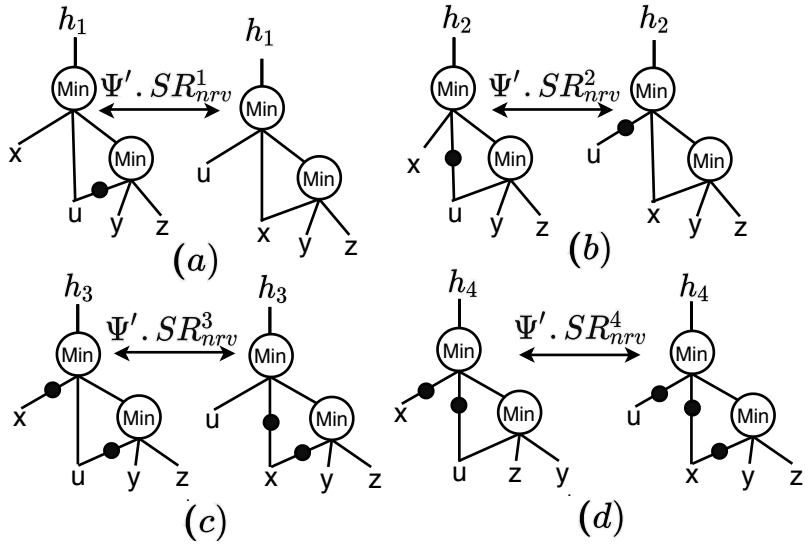

Figure 3.4: A variant of the *Swapping reconvergence property*, i.e, $\Psi'.SR_{nrv}$ where swapping of *reconvergent* (shared) Boolean variable with *nonreconvergent* Boolean variable can be observed in all four cases (a), (b), (c), and (d) as a result of which *reconvergent* variable pushed-up one level. Further, (a),(d) leads to inverter reduction.

3.3.2 Relevance rule in `mIG`

The second rule, *relevance* $\Psi'.R$, applies to the set of *reconvergent* variables, defined as $m(x, y, z) = m(x, y, z_{x/\bar{y}})$ where $x/\bar{y}$ implies replacing the *reconvergent* variable x with variable $\bar{y}$ for every occurrence of variable z.

We now demonstrate a case where the *relevance* rule in `mIG` representation has improved circuit delay optimization as compared to a case where *relevance* rule is applied in MIG representation.

Consider the logic function, $f = M(x, y, M(w, \bar{z}, M(x, y, z)))$ implemented in `MIG` representation as shown in Figure 3.5(a). Suppose the input w to f is driven by a logic function f_1. We assume that the critical path of f comprises f_1. The critical path delay of f is defined as, $t_f = t_{f_1} + 2t_M$, where t_{f_1} and t_M is the delay of f_1 and *majority* gate M, respectively. From [2], the *relevance* property $\Psi.R$ replaces x with y' in the bottom majority gate followed by $\Omega.M$. The equivalent circuit reduces to $f' = M(x, y, w)$, whose delay computes to $t'_f = t_{f_1} + t_M$, thus resulting in reduction of overall delay of f as, $\Delta t_f = t_f - t'_f$ as t_M.

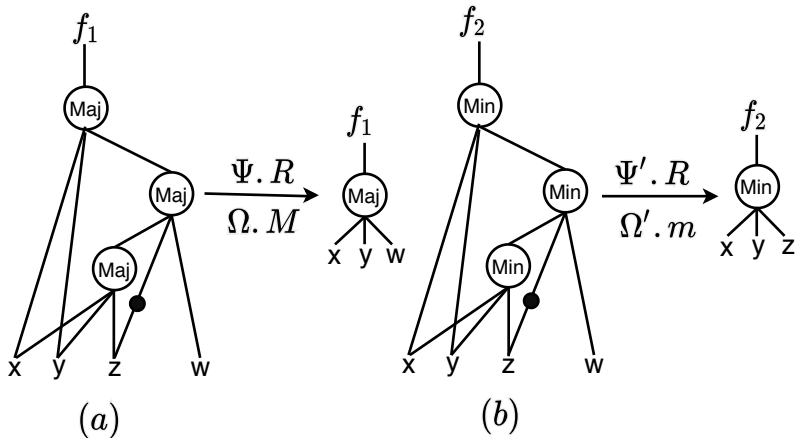

Figure 3.5: *Relevance* property depicted in (a) `MIG` representation of function $f = M(x, y, M(w, \bar{z}, M(x, y, z)))$ which takes the third input variable of middle node and puts it in the final node by applying $\Psi.R$,$\Omega.M$ respectively, (b) `mIG` representation of function $F = m(x, y, m(w, z', m(x, y, z)))$ takes the third input variable of the bottom node and puts it in the final node by applying $\Psi'.R$, and $\Omega'.m$, respectively.

Further, consider the logic function in `mIG` representation, $F = m(x, y, m(w, z', m(x, y, z)))$ shown in Figure 3.5(b). Suppose the input z to F is driven by a logic function F_1, and the critical path of F comprises F_1. The critical path delay of F is defined as $t_F = t_{F_1} + 2t_m + t_{inv}$, where t_{F_1}, t_m, and t_{inv} is the logic delay of F_1, *minority* gate m, and *inverter*, respectively. From *relevance* property $\Psi'.R$, the representation for F reduces to $F = m(x, y, z)$, and the associated critical path delay as $t'_F = t_{F_1} + t_m$. The overall reduction in critical path delay of F can be computed as $\Delta t_F = t_F - t'_F = t_{inv} + t_m$.

If the logic delay of standard cells of *minority* and a *majority* gates are considered equal, i.e., $t_m = t_M$, application of $\Psi'.R$ in *minority* gate has a better delay optimization than $\Psi.R$ in *majority* gate by a t_{inv} delay, i.e., one *inverter* delay. For a cascaded system of n stages in `mIG` and `MIG` representations, $\Psi'.R$ yields a better delay reduction over $\Psi.R$ by an amount of $n \cdot t_{inv}$.

Both `mIG` and `MIG` implementations of $F = m(x, y, m(w, z', m(x, y, z)))$ comprises three *reconvergent* variables, namely, x, y, and z. Each of these variables can have two polarities, leading to 64 possible circuit transformations.

Each such transformation can have either an `mIG` implementation or an `MIG` imple-

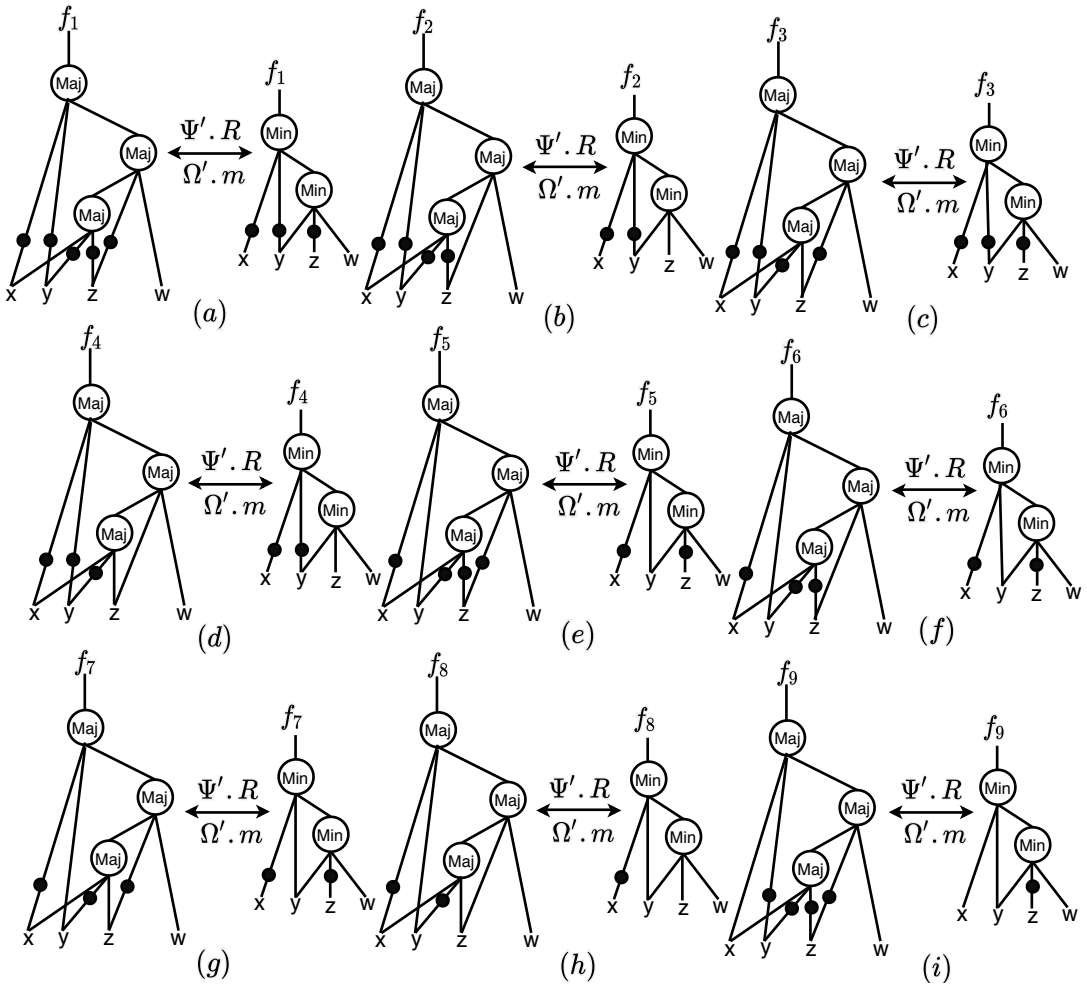

Figure 3.6: Out of 16 possible transformations for *reconvergent* variable, nine possible transformations are shown here that depicts smaller count of *complemented edges* (CE) and smaller count of *majority/minority* nodes, i.e, size optimization. The logical depth of the `mIG` network is also reduced if a left to right transformation is applied.

mentation. However, we find 16 transformations wherein `MIG` synthesis method yields reduced inverter count as compared to `mIG` synthesis. This improved performance in `mIG` is shown in Figure 3.6 and Figure 3.7. In addition, we identify that `mIG` implementation of 16 other transformations have a smaller *inverter* count as compared to the respective `MIG` implementation, which are as follows:

(i) $M(\bar{x}, \bar{y}, M(M(x, \bar{y}, \bar{z}), \bar{z}, w)) \leftrightarrow m(\bar{x}, \bar{y}, m(y, \bar{z}, w))$

(ii) $M(\bar{x}, \bar{y}, M(M(x, \bar{y}, \bar{z}), z, w)) \leftrightarrow m(\bar{x}, \bar{y}, m(y, z, w))$

(iii) $M(\bar{x}, \bar{y}, M(M(x, \bar{y}, z), \bar{z}, w) \leftrightarrow m(\bar{x}, \bar{y}, m(y, \bar{z}, w))$

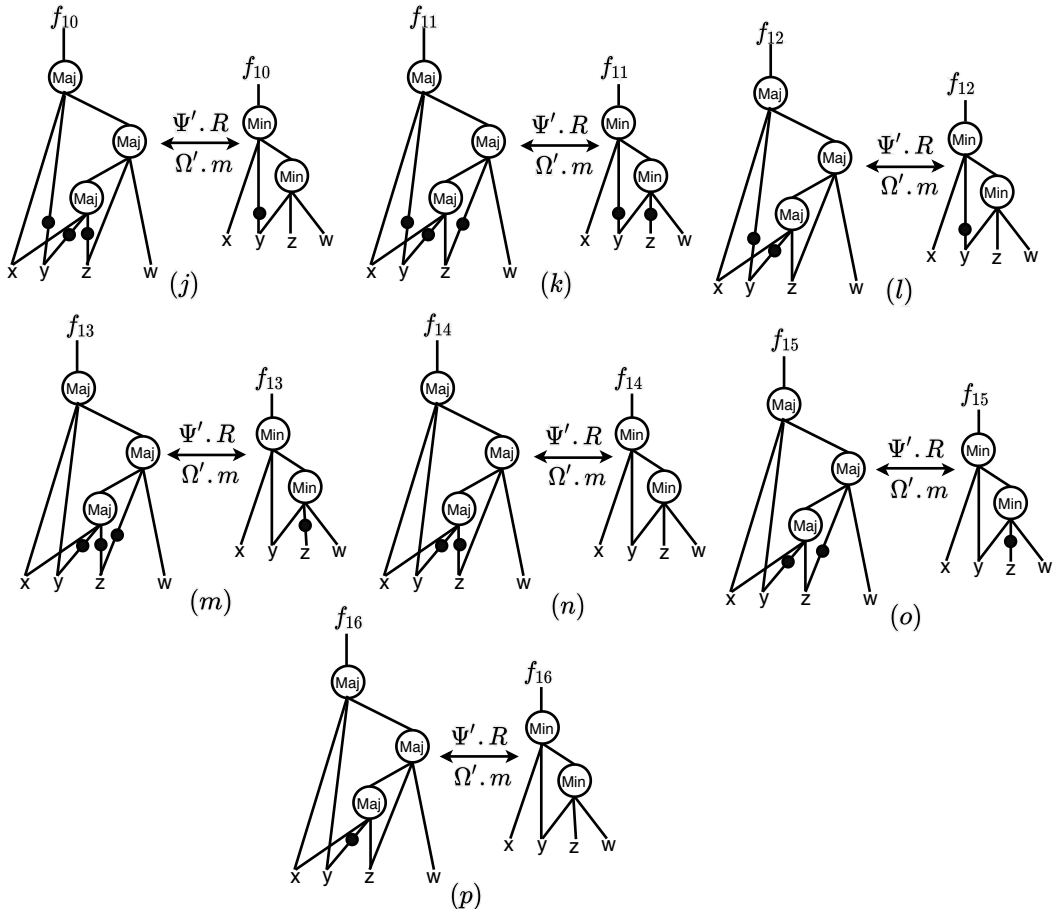

Figure 3.7: Out of 16, seven possible transformation are shown here that depicts size and depth optimization as achieved by applying *relevance* and *minority* rules.

(iv) $M(\bar{x}, \bar{y}, M(M(x, \bar{y}, z), z, w)) \leftrightarrow m(\bar{x}, \bar{y}, m(y, z, w))$

(v) $M(\bar{x}, y, M(M(x, \bar{y}, \bar{z}), \bar{z}, w)) \leftrightarrow m(\bar{x}, y, m(y, \bar{z}, w))$

(vi) $M(\bar{x}, y, M(M(x, \bar{y}, \bar{z}), z, w)) \leftrightarrow m(\bar{x}, y, m(y, z, w))$

(vii) $M(\bar{x}, y, M(M(x, \bar{y}, z), \bar{z}, w)) \leftrightarrow m(\bar{x}, y, m(y, \bar{z}, w))$

(viii) $M(\bar{x}, y, M(M(x, \bar{y}, z), z, w)) \leftrightarrow m(\bar{x}, y, m(y, z, w))$

(ix) $M(x, \bar{y}, M(M(x, \bar{y}, \bar{z}), \bar{z}, w)) \leftrightarrow m(x, \bar{y}, m(y, \bar{z}, w))$

(x) $M(x, \bar{y}, M(M(x, \bar{y}, \bar{z}), z, w)) \leftrightarrow m(x, \bar{y}, m(y, z, w))$

(xi) $M(x, \bar{y}, M(M(x, \bar{y}, z), \bar{z}, w)) \leftrightarrow m(x, \bar{y}, m(y, \bar{z}, w))$

(xii) $M(x, \bar{y}, M(M(x, \bar{y}, z), z, w)) \leftrightarrow m(x, \bar{y}, m(y, z, w))$

(xiii) $M(x, y, M(M(x, \bar{y}, \bar{z}), \bar{z}, w)) \leftrightarrow m(x, y, m(y, \bar{z}, w))$

(xiv) $M(x, y, M(M(x, \bar{y}, \bar{z}), z, w)) \leftrightarrow m(x, y, m(y, z, w))$

(xv) $M(x, y, M(M(x, \bar{y}, z), \bar{z}, w)) \leftrightarrow m(x, y, m(y, \bar{z}, w))$

(xvi) $M(x, y, M(M(x, \bar{y}, z), z, w)) \leftrightarrow m(x, y, m(y, z, w))$

There exist another mutually disjoint set of 16 transformations where in MIG transformation yields smaller *inverter* count. Hence, in some circuit topologies, mIG-based synthesis leads to smaller inversion count and in others MIG performs better.

3.3.3 Substitution rule in mIG

The third rule, *substitution*, $\Psi'.S$, is proposed as $m(x, y, z) = m(\bar{v}, m_{v/u}(x, y, z), m(\bar{u}, m_{v/\bar{u}}(x, y, z), \bar{v})$. This rule results in intermediate circuit enlargement, as a result of which primitive and algebraic transformations are applied to obtain the optimized circuit with fewer *inverter* and *minority* nodes. One such instance of $\Psi'.S$ is shown in Figure 3.8, which demonstrates that the function $f = x \oplus y \oplus z$ when implemented in mIG synthesis incurs two less *inverters* in count as compared to MIG synthesis.

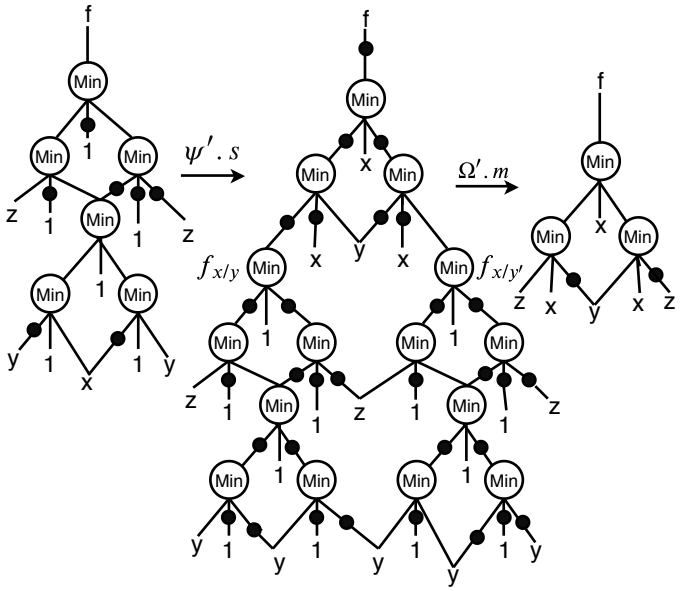

Figure 3.8: mIG implementation of $f = x \oplus y \oplus z$ and optimisation using *substitution* rule, $\Psi'.S$. The optimized implementation of f in mIG representation consumes two less *inversion* operations as compared to optimized implementation in MIG representation [2].

3.4 Introduction to minority-Majority-Inverter Graphs (mMIG)

Consider a logic topology represented algebraically as, $f = g(x, y, g(w, \bar{z}, g(x, y, z)))$, which was mentioned in Subsection 3.3.2 to motivate the need of shifting from mIG to mMIG. The function f is represented as cascaded stages of function g. The function g is represented using *majority* function as well as *minority* function. As we have seen 16 comparable cases where mIG performs better than MIG in term of resource count, there exist another mutually disjoint set of 16 transformations out of a total 32 comparable cases where MIG performs better than mIG in terms of resource count. It gives us an intuition to start looking at the topology implementation using both functions, *majority* as well as the *minority*, which we together consider as *minority-majority-inverter graph* (mMIG) synthesis approach.

Considering several case studies, which includes implementations of standard combinational circuits, lightweight cryptographic block ciphers discussed in Section 4.3, Section 4.4, Section 4.5, Section 4.6, Section 4.7,Section 4.8, we achieved significant amount of *inverter* reduction which motivated our intuition. Hence, from hereon, we focus further on the mMIG algebraic transformation methods as this is the best way to achieve optimization in the graph-based network that comprises the entire technology library. To attain improved performance, we have adopted a two-pronged approach comprising a *greedy approach* and a *divide-and-conquer* approach.

(i) *Greedy approach*: In this approach, we identify subgraph instances in MIG implementation where replacing a *majority* gate and an inverter gate at its output with a *minority* gate leads to a reduction in *inversion* count. As a result, this leads to reduced area without analyzing the complete circuit result, as shown in Algorithm 3, 4, which is present in main logic optimization in Algorithm 1.

(ii) *Divide and conquer approach*: We use this approach in Algorithm 5, where a circuit is divided into partitions based on the logic cones. These partitions are then optimized (with respect to count of inverters) on the basis of MIG, mIG, and

`mMIG` Boolean algebraic transformation rules.

In the next section, we present *minority-majority inverter graph* (`mMIG`) and its logical representation; its variaous properties are expressed using various forms of transformation rules. Furthermore, the set of rules is proposed on the constructed multilevel `mMIG` network to implement logic optimization in terms of the number of resources, i.e., size optimization, and reducing the number of levels of the mMIG network, i.e., depth optimization. Finally, we propose several algorithms for logic optimization. Fig 3.19 depicts a high-level overview of the major steps that comprise our methodology.

3.4.1 `mMIG` logic representation

An `mMIG` is a combinational logic network that comprises three-input *minority*, three-input *majority* node along with *inversion* operations represented as CEs. We consider $f_M = f(M(x, y, z))$ as a function that defined over *majority* operation taking x, y, z as input variable, f_I as an inverter function comprising only complemented edges, $f_m = f(m(u, v, w))$ as function defined over *minority* operation, and the function f_{mMIG} defined over input Boolean variable x, y, z, u, v, w are used in `mMIG` synthesis defined as,

$$f_{MIG} = f(f_M, f_I) \quad f_{mIG} = f(f_m, f_I)$$

$$f_M = xy + yz + xz \tag{3.4}$$

$$f_m = \bar{x}\bar{y} + \bar{y}\bar{z} + \bar{x}\bar{z} \tag{3.5}$$

$$f_{mMIG} = f(f_{MIG}, f_{mIG}) \tag{3.6}$$

We use terms *inverter* and *complemented edges* (`CE`s) interchangeably in the remaining thesis. In `mMIG`, edges are marked as a regular or a *complemented* attribute. In the remainder of the thesis, a three-input *majority* function or *minority* function

referred to as *majority* function or *minority* function, respectively.

Upon considering the output edges from a node, following three scenarios can arise,

Case (I): Nodes with one outgoing edge. This is a general case which is already analyzed.

Case (II): If outgoing edges of the node is two (say e_1 and e_2). There are four possible scenarios:

(i) both e_1 and e_2 are uncomplemented: In this case, the nodes are restored in their original forms.

(ii) e_1 is complemented and e_2 is uncomplemented: e_1 is at the output of minority node, and e_2 is at output of majority node with the same input value.

(iii) e_1 is uncomplemented and e_2 is complemented: This is equivalent to the previous case.

(iv) both e_1 and e_2 are complemented: It leads to following two possible cases:

- Subcase II(a): When node is majority then we can consider it as two unique minority nodes.
- Subcase II(b): When node is minority then we can consider it as two unique majority nodes.

Case (III): If outgoing edges of the node is greater than two. This will lead to a hypergraph which may require different optmization analysis to be considered.

3.5 Optimization rules in Minority Majority logic

In graph-based logic networks, the transformation rules are decisive in determining the extent of achieved optimization. The rules can take multiple forms for the same graph topology. Selection of desirable transformation is done on the basis of which transformation leads to lesser resource count, especially in the terms of *inverter* operations. We minimize the `CE`s of multilevel `mMIG` synthesized circuits using algebraic

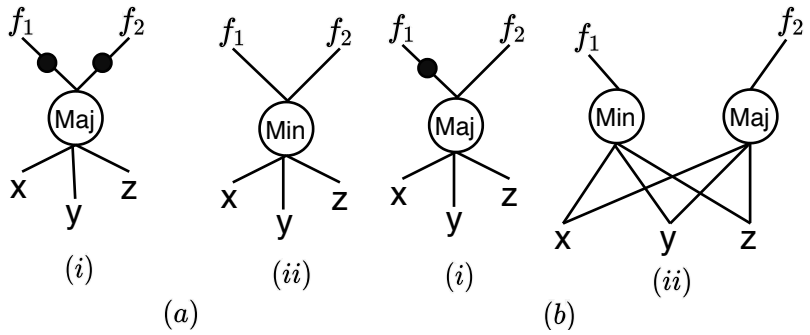

Figure 3.9: (a) A majority node when both the output edges are complemented is equivalent to a minority node with two uncomplemented output edges, (b) A majority node with two output edges, one complemented and one uncomplemented, is equivalent to a pair of majority and minority nodes driven by the same set of inputs, x, y, and z.

transformation procedure, which includes a set of *elimination*, *absorption*, and *reshaping* followed by other transformative rules. These algebraic transformation procedures in `mMIG` synthesis use following possible operations for optimization steps,

(i) The rules substitute majority nodes and the `CE`s which they drive with *minority* nodes.

(ii) The rules can push-up or push-down *reconvergent*/*nonreconvergent* Boolean variable across the consecutive levels, as a result of which the levels of the corresponding variables will change.

(iii) The rules can swap between *reconvergent* variable with a *nonreconvergent* variable at the same levels which results in graph comprising fewer number of `CE`s.

3.5.1 Inverter Reduction using Algebraic Transformation

To reduce resource count, especially in count of `CE`s of a multilevel graph-based network comprising *minority* nodes, *majority* nodes, and `CE`s, we apply algebraic transformation methods. Heuristically, we check whether using swapping operation, push-up, or push-down operations between two *reconvergent* Boolean variables, or a *reconvergent* Boolean variable with a *nonreconvergent* Boolean variable leads to a graph topology

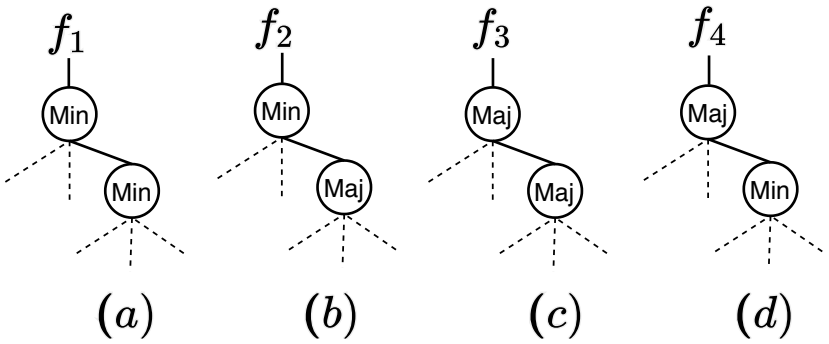

Figure 3.10: Permutable pairs of a majority and minority node are shown here where $f_1 ... f_4$ represents the intermediate output and dashed line represents the intermediate edges of the circuit.

demonstrating inverter reduction. In an iteration of applying transformations for a certain graph structure, we select that transformation greedily that leads to maximal reduction in count of `CE`s.

We first demonstrate the procedure for minimizing `CE`s after proposing the complete set of algebraic transformations for all possible classes.

Let us consider `Min` and `Maj` as representations of *minority* node and *majority* node in the mMIG logic synthesis approach, respectively. As basic topology, we consider all four possible permutations comprising a *minority* node and a *majority* node as depicted in Figure 3.10

Let us consider subcases $(a), (b)$ from Fig 3.11 in order to get the transformation rules associated with that topology. Fig 3.11(a) contains five *nonreconvergent* variables. Each of these variables can have two polarities, leading to 32 possible circuit transformations. The presence of only *nonreconvergent* variables hinders its candidacy for any transformation rule. Fig 3.11(b) is classified into (i) and (ii) based on the number of *reconvergent* variables, which contains both *reconvergent* and *nonreconvergent* variables. Based on the two possible polarities, it leads to a total of $32 + 32 = 64$ possible structures. These structures can be further classified into three different categories based on the propagation of *reconvergent* and *nonreconvergent* variables. As a result, the number of possible structures for Fig 3.11(b) is 192 $(= 64 \times 3)$. Hence, combining cases of Fig 3.11(b), Fig 3.11(d), Fig 3.11(f) and Fig 3.11(h) gives total of

768 (= 192 × 4) possible structures that comprises the scope of rule formation.

All possible circuit instances in mMIG are demonstrated in Figure 3.11. To achieve optimization, a set of rules is proposed, which is based on the observed characteristics of mMIG algebraic transformation rules, such as *swapping*, *absorbing*, *exclusion*, and *convergence*. In total, there are 768 possible transformations for mMIG synthesis which are classified based on the property, whose corresponding rules are mentioned in [61].

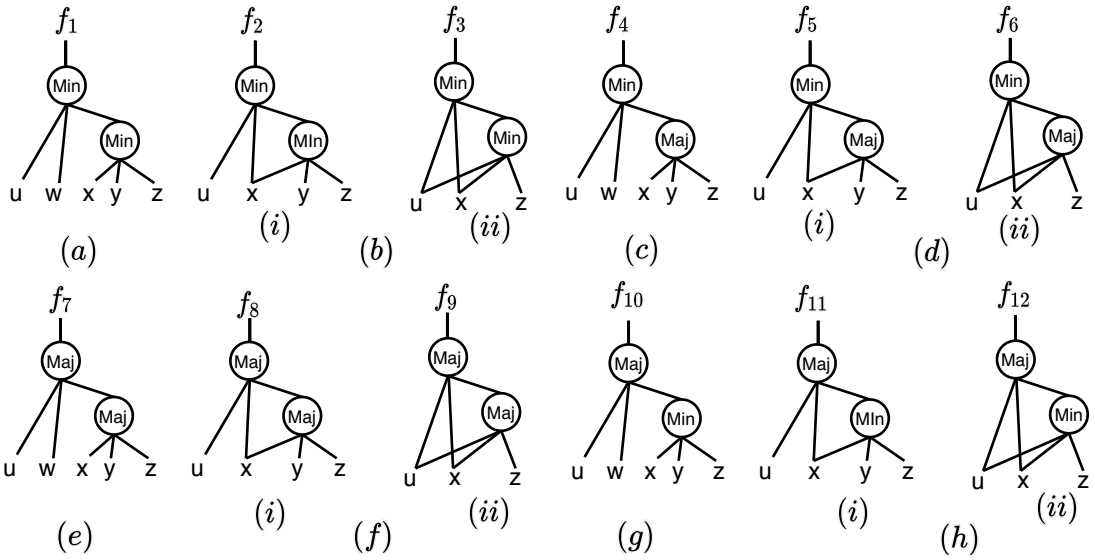

Figure 3.11: 12 basic `mMIG` topologies without inverter are classified based on the number of *reconvergent* variables (≤ 2). x, y, z, u and w are input Boolean variables where $f_1, \ldots, f_{12}$ depict the output of `mMIG` topology.

Based on the optimization class, these possible transformations are broadly classified in the form of algebraic rules. The classes include, *Elimination rule*, *Absorption rule*, *Swapping reconvergence rule*, and *Relevance rule*. The optimization class consists of several forms of rules that are yet to be assigned based on their inherent property.

Based on the above classification, transformation rules are categorized on the basis of optimization class as:

- **Class 1**: Optimization only in number of `CE`s if we follow the variable propagation either from left to right or right to left. The transformation rules in this class are as follows:

1. $m(u,x,m(x,y,z)) \leftrightarrow m(u,\bar{y},m(\bar{y},\bar{x},z)) \leftrightarrow m(u,\bar{z},m(\bar{z},y,\bar{x}))$
2. $m(u,x,m(x,y,\bar{z})) \leftrightarrow m(u,\bar{y},m(\bar{y},\bar{x},\bar{z})) \leftrightarrow m(u,\bar{y},M(y,x,z))$
3. $m(u,x,m(x,\bar{y},z)) \leftrightarrow m(u,\bar{z},m(\bar{z},\bar{y},\bar{x})) \leftrightarrow m(u,\bar{z},M(y,x,z))$
4. $m(u,x,m(\bar{x},y,\bar{z})) \leftrightarrow m(\bar{y},x,m(\bar{x},\bar{u},\bar{z})) \leftrightarrow m(\bar{y},x,M(x,u,z))$
5. $m(u,x,m(\bar{x},\bar{y},z)) \leftrightarrow m(y,x,m(\bar{x},\bar{u},z)) \leftrightarrow m(\bar{z},x,m(\bar{x},\bar{y},\bar{u}))$
6. $m(u,x,m(\bar{x},\bar{y},\bar{z})) \leftrightarrow m(y,x,m(\bar{x},\bar{u},\bar{z})) \leftrightarrow m(z,x,m(\bar{x},\bar{y},\bar{u}))$
   $\leftrightarrow m(y,x,M(x,u,z)) \leftrightarrow m(z,x,M(x,y,u))$
7. $m(u,\bar{x},m(x,y,z)) \leftrightarrow m(\bar{y},\bar{x},m(x,\bar{u},z)) \leftrightarrow m(\bar{z},\bar{x},m(x,y,\bar{u}))$
8. $m(u,\bar{x},m(x,y,\bar{z})) \leftrightarrow m(\bar{y},\bar{x},m(x,\bar{u},\bar{z})) \leftrightarrow m(z,\bar{x},m(x,y,\bar{u}))$
9. $m(u,\bar{x},m(x,\bar{y},z)) \leftrightarrow m(y,\bar{x},m(x,\bar{u},z)) \leftrightarrow m(\bar{z},\bar{x},m(x,\bar{y},\bar{u}))$
10. $m(u,\bar{x},m(\bar{x},y,\bar{z})) \leftrightarrow m(u,z,m(z,y,x)) \leftrightarrow m(u,\bar{y},m(\bar{y},x,\bar{z}))$
11. $m(u,\bar{x},m(\bar{x},\bar{y},z)) \leftrightarrow m(u,y,m(y,x,z)) \leftrightarrow m(u,\bar{z},m(\bar{z},\bar{y},x))$
12. $m(u,\bar{x},m(\bar{x},\bar{y},\bar{z})) \leftrightarrow m(u,\bar{x},M(z,y,x))$
13. $m(\bar{u},x,m(x,y,z)) \leftrightarrow m(\bar{u},\bar{y},m(\bar{y},\bar{x},z))$
14. $m(\bar{u},x,m(\bar{x},y,\bar{z})) \leftrightarrow m(z,x,m(\bar{x},y,u))$
15. $m(\bar{u},x,m(\bar{x},\bar{y},z)) \leftrightarrow m(y,x,m(\bar{x},u,z))$
16. $m(\bar{u},x,m(\bar{x},\bar{y},\bar{z})) \leftrightarrow m(y,x,m(\bar{x},u,\bar{z}))$
17. $m(\bar{u},\bar{x},m(x,y,\bar{z})) \leftrightarrow m(z,\bar{x},m(x,y,u))$
18. $m(\bar{u},\bar{x},m(x,\bar{y},z)) \leftrightarrow m(y,\bar{x},\bar{x},m(x,u,z))$
19. $m(\bar{u},\bar{x},m(x,\bar{y},\bar{z})) \leftrightarrow m(y,\bar{x},m(x,u,\bar{z}))$
20. $m(\bar{u},\bar{x},m(\bar{x},y,\bar{z})) \leftrightarrow m(\bar{u},z,m(x,y,z))$
21. $m(\bar{u},\bar{x},m(\bar{x},\bar{y},z)) \leftrightarrow m(\bar{u},y,m(x,y,z))$
22. $m(\bar{u},\bar{x},m(\bar{x},\bar{y},\bar{z})) \leftrightarrow m(\bar{u},y,m(x,y,\bar{z}))$
23. $m(u,x,M(\bar{x},\bar{y},\bar{z})) \leftrightarrow m(u,x,m(x,y,z))$.
24. $m(u,\bar{x},M(\bar{x},\bar{y},\bar{z})) \leftrightarrow m(\bar{y},\bar{x},M(\bar{x},u,\bar{z})) \leftrightarrow m(\bar{z},\bar{x},M(\bar{x},\bar{y},u))$
    $\leftrightarrow m(u,\bar{x},m(x,y,z))$
25. $m(\bar{u},x,M(\bar{x},\bar{y},\bar{z})) \leftrightarrow m(\bar{u},x,m(x,y,z))$

26. $m(\bar{u}, \bar{x}, M(\bar{x}, y, z)) \leftrightarrow m(y, \bar{x}, M(\bar{x}, \bar{u}, z)) \leftrightarrow m(z, \bar{x}, M(\bar{x}, y, \bar{u}))$
27. $m(\bar{u}, \bar{x}, M(\bar{x}, y, \bar{z})) \leftrightarrow m(y, \bar{x}, M(\bar{x}, \bar{u}, \bar{z})) \leftrightarrow m(y, \bar{x}, m(x, y, z))$
$\leftrightarrow m(\bar{z}, \bar{x}, M(\bar{x}, y, \bar{u}))$
28. $m(\bar{u}, \bar{x}, M(\bar{x}, \bar{y}, z)) \leftrightarrow m(\bar{y}, \bar{x}, M(\bar{x}, \bar{u}, z)) \leftrightarrow m(z, \bar{x}, M(\bar{x}, \bar{y}, \bar{u}))$
$\leftrightarrow m(z, \bar{x}, m(x, y, u))$
29. $m(\bar{u}, \bar{x}, M(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow m(\bar{y}, \bar{x}, M(\bar{x}, \bar{u}, \bar{z})) \leftrightarrow m(\bar{z}, \bar{x}, M(\bar{x}, \bar{y}, \bar{u}))$
$\leftrightarrow m(\bar{u}, \bar{x}, m(x, y, z)) \leftrightarrow m(\bar{y}, \bar{x}, m(x, u, z)) \leftrightarrow m(\bar{z}, \bar{x}, m(x, y, u))$
30. $M(u, x, M(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow M(u, x, m(x, y, z))$
31. $M(u, \bar{x}, M(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow M(\bar{y}, \bar{x}, M(\bar{x}, u, \bar{z})) \leftrightarrow M(\bar{z}, \bar{x}, M(\bar{x}, \bar{y}, u))$
32. $M(\bar{u}, x, M(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow M(\bar{u}, x, m(x, y, z))$
33. $M(\bar{u}, \bar{x}, M(x, \bar{y}, \bar{z})) \leftrightarrow m(u, x, M(\bar{x}, y, z))$
34. $M(\bar{u}, \bar{x}, M(\bar{x}, y, \bar{z})) \leftrightarrow m(u, x, M(x, \bar{y}, z))$
35. $M(u, \bar{x}, m(x, y, z)) \leftrightarrow M(\bar{y}, \bar{x}, m(x, \bar{u}, z)) \leftrightarrow M(\bar{z}, \bar{x}, m(x, y, \bar{u}))$
36. $M(u, m(\bar{x}, y, z)) \leftrightarrow M(\bar{y}, x, m(\bar{x}, \bar{u}, z)) \leftrightarrow M(\bar{z}, x, m(\bar{x}, y, \bar{u}))$
37. $M(\bar{u}, \bar{x}, M(\bar{x}, \bar{y}, z)) \leftrightarrow m(u, x, M(x, y, \bar{z}))$
38. $M(\bar{u}, \bar{x}, M(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow m(u, x, M(x, y, z))$

- **Class 2**: Size optimization in the terms of number of `CE`s as well as *majority* nodes or *minority* nodes, and Depth optimization. The transformation rules in this class are as follows:

1. $m(x, y, m(x, y, z)) \leftrightarrow m(x, y, \bar{z})$
2. $m(x, y, m(x, y, \bar{z})) \leftrightarrow m(x, y, z)$
3. $m(x, y, m(x, \bar{y}, z)) \leftrightarrow \bar{y}$
4. $m(x, y, m(x, \bar{y}, \bar{z})) \leftrightarrow \bar{y}$
5. $m(x, y, m(\bar{x}, y, z)) \leftrightarrow \bar{x}$
6. $m(x, y, m(\bar{x}, y, \bar{z})) \leftrightarrow \bar{x}$
7. $m(x, y, m(\bar{x}, \bar{y}, z)) \leftrightarrow m(x, y, \bar{z})$

8. $m(x, y, m(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow m(x, y, z)$
9. $m(x, \bar{y}, m(x, y, z)) \leftrightarrow y$
10. $m(x, \bar{y}, m(x, y, \bar{z})) \leftrightarrow y$
11. $m(x, \bar{y}, m(x, \bar{y}, z)) \leftrightarrow M(\bar{x}, y, z)$
12. $m(x, \bar{y}, m(x, \bar{y}, \bar{z})) \leftrightarrow m(x, \bar{y}, z)$
13. $m(x, \bar{y}, m(\bar{x}, y, z)) \leftrightarrow M(\bar{x}, y, z)$
14. $m(x, \bar{y}, m(\bar{x}, y, \bar{z})) \leftrightarrow m(x, \bar{y}, z)$
15. $m(x, \bar{y}, m(\bar{x}, \bar{y}, z)) \leftrightarrow \bar{x}$
16. $m(x, \bar{y}, m(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow \bar{x}$
17. $m(\bar{x}, y, m(x, y, z)) \leftrightarrow x$
18. $m(\bar{x}, y, m(x, y, \bar{z})) \leftrightarrow x$
19. $m(\bar{x}, y, m(x, \bar{y}, z)) \leftrightarrow M(x, \bar{y}, z)$
20. $m(\bar{x}, y, m(x, \bar{y}, \bar{z})) \leftrightarrow m(\bar{x}, y, z)$
21. $m(\bar{x}, y, m(\bar{x}, y, z)) \leftrightarrow M(x, \bar{y}, z)$
22. $m(\bar{x}, y, m(\bar{x}, y, \bar{z})) \leftrightarrow m(\bar{x}, y, z)$
23. $m(\bar{x}, y, m(\bar{x}, \bar{y}, z)) \leftrightarrow \bar{y}$
24. $m(\bar{x}, y, m(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow \bar{y}$
25. $m(\bar{x}, \bar{y}, m(x, y, z)) \leftrightarrow M(x, y, z)$
26. $m(\bar{x}, \bar{y}, m(x, y, \bar{z})) \leftrightarrow M(x, y, \bar{z})$
27. $m(\bar{x}, \bar{y}, m(x, \bar{y}, z)) \leftrightarrow x$
28. $m(\bar{x}, \bar{y}, m(y, \bar{x}, \bar{z})) \leftrightarrow y$
29. $m(\bar{x}, \bar{y}, m(\bar{x}, y, z)) \leftrightarrow y$
30. $m(\bar{x}, \bar{y}, m(\bar{x}, y, \bar{z})) \leftrightarrow y$
31. $m(\bar{x}, \bar{y}, m(\bar{x}, \bar{y}, z)) \leftrightarrow M(x, y, z)$
32. $m(\bar{x}, \bar{y}, m(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow M(x, y, \bar{z})$
33. $M(x, y, m(x, y, z)) \leftrightarrow M(x, y, \bar{z})$

34. $M(x, y, m(x, y, \bar{z})) \leftrightarrow M(x, y, z)$

35. $M(x, y, m(x, \bar{y}, z)) \leftrightarrow y$

36. $M(x, y, m(x, \bar{y}, \bar{z})) \leftrightarrow y$

37. $M(x, y, m(\bar{x}, y, z)) \leftrightarrow x$

38. $M(x, y, m(\bar{x}, y, \bar{z})) \leftrightarrow x$

39. $M(x, y, m(\bar{x}, \bar{y}, z)) \leftrightarrow M(x, y, \bar{z})$

40. $M(x, y, m(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow M(x, y, z)$

41. $M(x, \bar{y}, m(x, y, z)) \leftrightarrow \bar{y}$

42. $M(x, \bar{y}, m(x, y, \bar{z})) \leftrightarrow \bar{y}$

43. $M(x, \bar{y}, m(x, \bar{y}, z)) \leftrightarrow m(\bar{x}, y, z)$

44. $M(x, \bar{y}, m(x, \bar{y}, \bar{z})) \leftrightarrow M(x, \bar{y}, z)$

45. $M(x, \bar{y}, m(\bar{x}, y, z)) \leftrightarrow m(x, y, z)$

46. $M(x, \bar{y}, m(\bar{x}, y, \bar{z})) \leftrightarrow M(x, \bar{y}, z)$

47. $M(x, \bar{y}, m(\bar{x}, \bar{y}, z)) \leftrightarrow x$

48. $M(x, \bar{y}, m(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow x$

49. $M(\bar{x}, y, m(x, y, z)) \leftrightarrow \bar{x}$

50. $M(\bar{x}, y, m(x, y, \bar{z})) \leftrightarrow \bar{x}$

51. $M(\bar{x}, y, m(x, \bar{y}, z)) \leftrightarrow m(x, \bar{y}, z)$

52. $M(\bar{x}, y, m(x, \bar{y}, \bar{z})) \leftrightarrow M(\bar{x}, y, z)$

53. $M(\bar{x}, y, m(\bar{x}, y, z)) \leftrightarrow m(x, \bar{y}, z)$

54. $M(\bar{x}, y, m(\bar{x}, y, \bar{z})) \leftrightarrow M(\bar{x}, y, z)$

55. $M(\bar{x}, y, m(\bar{x}, \bar{y}, z)) \leftrightarrow y$

56. $M(\bar{x}, y, m(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow \bar{x}$

57. $M(\bar{x}, \bar{y}, m(x, y, z)) \leftrightarrow m(x, y, z)$

58. $M(\bar{x}, \bar{y}, m(x, y, \bar{z})) \leftrightarrow m(x, y, \bar{z})$

59. $M(\bar{x}, \bar{y}, m(x, \bar{y}, z)) \leftrightarrow \bar{x}$

60. $M(\bar{x}, \bar{y}, m(x, \bar{y}, \bar{z})) \leftrightarrow \bar{x}$

61. $M(\bar{x}, \bar{y}, m(\bar{x}, y, z)) \leftrightarrow \bar{y}$

62. $M(\bar{x}, \bar{y}, m(\bar{x}, y, \bar{z})) \leftrightarrow \bar{y}$

63. $M(\bar{x}, \bar{y}, m(\bar{x}, \bar{y}, z)) \leftrightarrow m(x, y, z)$

64. $M(\bar{x}, \bar{y}, m(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow m(x, y, \bar{z}$

65. $m(x, y, M(x, y, z)) \leftrightarrow m(x, y, z) \leftrightarrow m(x, z, M(x, y, z)) \leftrightarrow m(z, y, M(x, y, z))$

66. $m(x, y, M(x, y, \bar{z})) \leftrightarrow m(x, y, \bar{z}) \leftrightarrow m(\bar{z}, y, M(x, y, \bar{z})) \leftrightarrow m(x, \bar{z}, M(x, \bar{z}, y))$

67. $m(x, y, M(x, \bar{y}, z)) \leftrightarrow \bar{x}$

68. $m(x, y, M(x, \bar{y}, \bar{z})) \leftrightarrow \bar{x}$

69. $m(x, y, M(\bar{x}, y, z)) \leftrightarrow \bar{y}$

70. $m(x, y, M(\bar{x}, y, \bar{z})) \leftrightarrow \bar{y} \leftrightarrow m(z, y, M(\bar{z}, y, \bar{x}))$

71. $m(x, y, M(\bar{x}, \bar{y}, z)) \leftrightarrow m(x, y, z) \leftrightarrow m(z, y, M(\bar{z}, \bar{y}, x)) \leftrightarrow m(x, z, M(\bar{x}, \bar{z}, y))$

72. $m(x, y, M(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow m(x, y, z) \leftrightarrow m(z, y, M(\bar{z}, \bar{y}, x))$

73. $m(x, \bar{y}, M(x, y, z)) \leftrightarrow \bar{x}$

74. $m(x, \bar{y}, M(x, y, \bar{z})) \leftrightarrow \bar{x}$

75. $m(x, \bar{y}, M(x, \bar{y}, z)) \leftrightarrow m(x, \bar{y}, z) \leftrightarrow m(z, \bar{y}, M(z, \bar{y}, x))$

76. $m(x, \bar{y}, M(x, \bar{y}, \bar{z})) \leftrightarrow M(\bar{x}, y, z)$

77. $m(x, \bar{y}, M(\bar{x}, y, z)) \leftrightarrow m(x, \bar{y}, z)$

78. $m(x, \bar{y}, M(\bar{x}, y, \bar{z}) \leftrightarrow M(\bar{x}, y, z)$

79. $m(x, \bar{y}, M(\bar{x}, \bar{y}, z) \leftrightarrow y$

80. $m(x, \bar{y}, M(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow y$

81. $m(\bar{x}, y, M(x, y, z)) \leftrightarrow \bar{y}$

82. $m(\bar{x}, y, M(x, y, \bar{z})) \leftrightarrow \bar{y}$

83. $m(\bar{x}, y, M(x, \bar{y}, z)) \leftrightarrow m(\bar{x}, y, z)$

84. $m(\bar{x}, y, M(x, \bar{y}, \bar{z})) \leftrightarrow M(x, \bar{y}, z)$

85. $m(\bar{x}, y, M(\bar{x}, y, z)) \leftrightarrow m(\bar{x}, y, z)$

86. $m(\bar{x}, y, M(\bar{x}, y, \bar{z})) \leftrightarrow M(x, \bar{y}, z)$

87. $m(\bar{x}, y, M(\bar{x}, \bar{y}, z)) \leftrightarrow x$

88. $m(\bar{x}, y, M(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow x$

89. $m(\bar{x}, \bar{y}, M(x, y, z)) \leftrightarrow M(x, y, \bar{z})$

90. $m(\bar{x}, \bar{y}, M(x, y, \bar{z})) \leftrightarrow M(x, y, z)$

91. $m(\bar{x}, \bar{y}, M(x, \bar{y}, z)) \leftrightarrow y$

92. $m(\bar{x}, \bar{y}, M(x, \bar{y}, \bar{z})) \leftrightarrow y$

93. $m(\bar{x}, \bar{y}, M(\bar{x}, y, z)) \leftrightarrow x$

94. $m(\bar{x}, \bar{y}, M(\bar{x}, y, \bar{z})) \leftrightarrow x$

95. $m(\bar{x}, \bar{y}, M(\bar{x}, \bar{y}, z)) \leftrightarrow M(x, y, \bar{z})$

96. $m(\bar{x}, \bar{y}, M(\bar{x}, \bar{y}, \bar{z})) \leftrightarrow M(x, y, z)$

- **Class 3**: In this class, the variable in the critical path of the circuit propagates towards the output, thus leading to decrease in the depth of the circuit. The transformations in this class are as follows:

1. $m(u, x, m(x, \bar{y}, \bar{z})) \leftrightarrow m(u, y, m(y, \bar{x}, \bar{z})) \leftrightarrow m(u, z, m(z, \bar{y}, \bar{x}))$

2. $m(u, x, m(\bar{x}, y, z)) \leftrightarrow m(\bar{y}, x, m(\bar{x}, \bar{u}, z)) \leftrightarrow m(\bar{z}, x, m(\bar{x}, y, \bar{u}))$

3. $m(u, \bar{x}, m(x, \bar{y}, \bar{z})) \leftrightarrow m(y, \bar{x}, m(x, \bar{u}, \bar{z})) \leftrightarrow m(z, \bar{x}, m(x, \bar{y}, \bar{u}))$

4. $m(u, \bar{x}, m(\bar{x}, y, z)) \leftrightarrow m(u, \bar{y}, m(\bar{y}, x, z)) \leftrightarrow m(u, \bar{z}, m(\bar{z}, y, x))$

5. $m(\bar{u}, x, m(x, y, \bar{z})) \leftrightarrow m(\bar{u}, z, m(z, y, \bar{x}))$

6. $m(\bar{u}, x, m(x, \bar{y}, z)) \leftrightarrow m(\bar{u}, y, m(y, \bar{x}, z))$

7. $m(\bar{u}, x, m(x, \bar{y}, \bar{z})) \leftrightarrow m(\bar{u}, y, m(y, \bar{x}, \bar{z}))$

8. $m(\bar{u}, x, m(\bar{x}, y, z)) \leftrightarrow m(\bar{y}, x, m(\bar{x}, u, z))$

9. $m(\bar{u}, \bar{x}, m(x, y, z)) \leftrightarrow m(\bar{y}, \bar{x}, m(x, u, z))$

10. $m(\bar{u}, \bar{x}, m(\bar{x}, y, z)) \leftrightarrow m(\bar{u}, \bar{y}, m(\bar{y}, x, z))$

11. $m(u, x, M(x, y, z)) \leftrightarrow m(y, x, M(x, u, z) \leftrightarrow m(z, x, M(x, y, u))$

12. $m(u, x, M(x, y, \bar{z})) \leftrightarrow m(y, x, M(x, u, \bar{z})) \leftrightarrow m(\bar{z}, x, M(x, y, u))$

13. $m(u, x, M(x, \bar{y}, z)) \leftrightarrow m(\bar{y}, x, M(x, u, z)) \leftrightarrow m(z, x, M(x, \bar{y}, u))$

14. $m(u, x, M(x, \bar{y}, \bar{z})) \leftrightarrow m(\bar{y}, x, M(x, u, \bar{z})) \leftrightarrow m(\bar{z}, x, M(x, \bar{y}, u))$

15. $m(u, \bar{x}, M(\bar{x}, y, z)) \leftrightarrow m(y, \bar{x}, M(\bar{x}, u, z)) \leftrightarrow m(z, \bar{x}, M(\bar{x}, y, u))$

16. $m(u, \bar{x}, M(\bar{x}, y, \bar{z})) \leftrightarrow m(y, \bar{x}, M(\bar{x}, u, \bar{z})) \leftrightarrow m(\bar{z}, \bar{x}, M(\bar{x}, y, u))$

17. $m(u, \bar{x}, M(\bar{x}, \bar{y}, z)) \leftrightarrow m(\bar{y}, \bar{x}, M(\bar{x}, u, z)) \leftrightarrow m(z, \bar{x}, M(\bar{x}, \bar{y}, u))$

18. $m(\bar{u}, x, M(x, y, z)) \leftrightarrow m(y, x, M(x, \bar{u}, z)) \leftrightarrow m(z, x, M(x, y, \bar{u}))$

19. $m(\bar{u}, x, M(x, y, \bar{z})) \leftrightarrow m(\bar{z}, x, M(x, y, \bar{u})) \leftrightarrow m(y, x, M(x, \bar{u}, \bar{z}))$

20. $m(\bar{u}, x, M(x, \bar{y}, z)) \leftrightarrow m(\bar{y}, x, M(x, \bar{u}, z)) \leftrightarrow m(z, x, \bar{y}, \bar{u})$

21. $m(\bar{u}, x, M(x, \bar{y}, \bar{z})) \leftrightarrow m(\bar{y}, x, M(x, \bar{u}, \bar{z})) \leftrightarrow m(\bar{z}, x, M(x, \bar{y}, \bar{u}))$

22. $M(u, x, M(x, y, z)) \leftrightarrow M(y, x, M(x, u, z)) \leftrightarrow M(z, x, M(x, y, u))$

23. $M(u, x, M(x, y, \bar{z})) \leftrightarrow M(y, x, M(x, u, \bar{z})) \leftrightarrow M(\bar{z}, x, M(x, y, u))$

24. $M(u, x, M(x, \bar{y}, z)) \leftrightarrow M(\bar{y}, x, M(x, u, z)) \leftrightarrow M(z, x, M(x, \bar{y}, u))$

25. $M(u, x, M(x, \bar{y}, \bar{z})) \leftrightarrow M(\bar{y}, x, M(x, u, \bar{z})) \leftrightarrow M(\bar{z}, x, M(x, \bar{y}, u))$

26. $M(u, \bar{x}, M(\bar{x}, y, z)) \leftrightarrow M(y, \bar{x}, M(\bar{x}, u, z)) \leftrightarrow M(z, \bar{x}, M(\bar{x}, y, u))$

27. $M(u, \bar{x}, M(\bar{x}, y, \bar{z})) \leftrightarrow M(\bar{z}, \bar{x}, M(\bar{x}, y, u)) \leftrightarrow M(y, \bar{x}, M(\bar{x}, u, \bar{z}))$

28. $M(u, \bar{x}, M(\bar{x}, \bar{y}, z)) \leftrightarrow M(z, \bar{x}, M(\bar{x}, \bar{y}, u)) \leftrightarrow M(\bar{y}, \bar{x}, M(\bar{x}, u, z))$

29. $M(\bar{u}, x, M(x, y, z)) \leftrightarrow M(y, x, M(x, \bar{u}, z) \leftrightarrow M(z, x, M(x, y, \bar{u}))$

30. $M(\bar{u}, x, M(x, y, \bar{z})) \leftrightarrow M(y, x, M(x, \bar{u}, \bar{z})) \leftrightarrow M(\bar{z}, x, M(x, y, \bar{u}))$

31. $M(\bar{u}, x, M(x, \bar{y}, z)) \leftrightarrow M(\bar{y}, x, M(x, \bar{u}, z)) \leftrightarrow M(z, x, M(x, \bar{y}, \bar{u}))$

32. $M(\bar{u}, x, M(x, \bar{y}, \bar{z})) \leftrightarrow M(\bar{y}, x, M(x, \bar{u}, \bar{z})) \leftrightarrow M(\bar{z}, x, M(x, \bar{y}, \bar{u}))$

33. $M(\bar{u}, \bar{x}, M(\bar{x}, y, z)) \leftrightarrow M(y, \bar{x}, M(\bar{x}, \bar{u}, z)) \leftrightarrow M(z, \bar{x}, M(\bar{x}, y, \bar{u}))$

All the proposed classes of transformation rules fulfil the primary goal of reducing the `CE`s. Some classes of the rules lead to direct reduction of `CE`s, whereas others reduce `CE`s after some primitive transformation rules.

A heuristic approach for inverter reduction using algebraic transformation is proposed as:

(i) The set of transformation rules in Class 1 is applied if and only if the number of *reconvergent* variables is only one and the pairs are *minority-minority*, *minority-majority*, and *majority-minority*.

(ii) The set of transformation rules in Class 2 is applied if and only if the number of *reconvergent* is two and the pairs are *minority-minority*, *majority-minority*, and *minority majority*.

(iii) The set of transformation rules in Class 3 is applied if and only if propagation of critical either *reconvergent* or *nonreconvergent* Boolean variable does not affect any intermediate result.

A set of rules based on the observed characteristic of `mMIG` algebraic transformation rules, such as *swapping*, *absorbing*, *exclusion*, *convergence* is proposed. In total, there can be 768 possible transformations for `mMIG` synthesis as follows:

- There are possible cases which lead to either partial optimization or complete optimization either in terms of size or depth or both.
- A set of rules can be grouped and renamed based on the observed property.
- There are some cases whose equivalent structures could not be formulated.
- Not every rule leads to size optimization or depth optimization.
- These rules will cover all three partitions of `mMIG` synthesized circuit as discussed in Algorithm 5, where only connected *minority* nodes, connected *majority* nodes, and connected *majority-minority* nodes are considered for separate set of transformation rules.
- There is a similar topology that exists for most of the structure, which comprises an equal count of *majority* nodes and *minority* nodes, but lesser *inversion operations* if propagation of variables are from left to right or from right to left.

3.5.2 Elimination rule in mMIG synthesis

Elimination rules in mMIG comprise extended scenarios which can be optimized in term of the number of CEs or number of *majority* nodes/*minority* nodes. Elimination rules are shown in Figure 3.12.

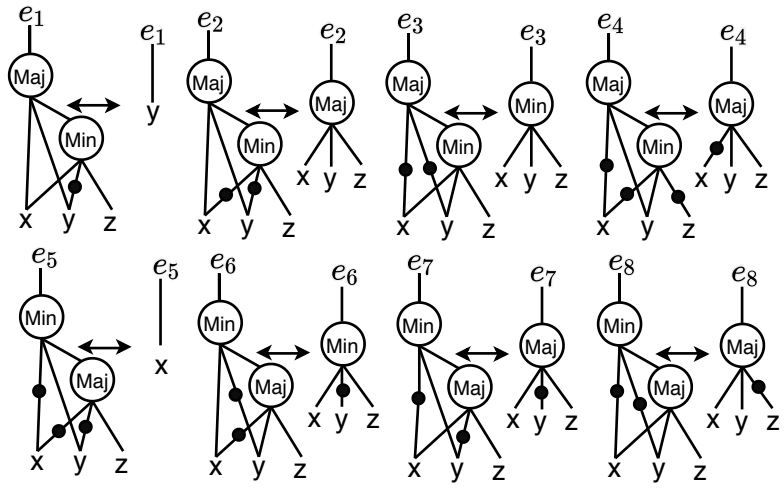

Figure 3.12: Eight instances of elimination rules are listed for each of the eight circuit outputs $e_1, \ldots, e_8$ results in compact mMIG synthesized circuit.

The elimination rules in mMIG synthesis comprise cases wherein two back-to-back cascaded *minority* and *majority* nodes are eliminated with input signals that drive the circuit. Some instances of elimination rules are, $\Psi_{mMIG}.El$ are, $M(x, m(x, x, y), z) = z$, $M(x, m(\overline{x}, \overline{x}, y), z) = x$, $m(x, M(x, x, y), z) = \overline{x}$, etc. mMIG synthesis captures such scenarios wherein a system of nodes are eliminated resulting in circuits with reduced area. The transformations, namely, *associativity*, *complementary associativity*, and *distributivity* properties solely perform elimination of only majority or only minority nodes. However, in case of connected majority-minority nodes, the proposed elimination rule reduces the number of intermediate steps of removing the redundant nodes much faster, thus achieving reduced area in smaller number of iterations.

3.5.3 Absorption rule in mMIG

We propose the following absorption rules in mMIG logic synthesis. These rules involve absorbing or combining a *minority* node and a *majority* node and replace the nodes with either a *majority* or *minority* node. The absorption property of this rule in term of *majority* or *minority* nodes is illustrated in Figure 3.13.

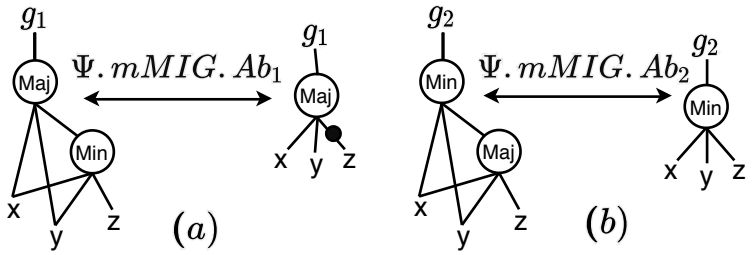

Figure 3.13: Absorption rule for replacing a connected set of *majority* and *minority* nodes with lesser number of either *majority* or *minority* nodes.

This class of transformation rules comprises the following variants:

1. $\Psi_{mMIG}.Ab_1$: When upper node is *majority* and bottom node is *minority*.

   (a) $M(x, y, m(x, y, z)) \leftrightarrow M(x, y, \bar{z})$

2. $\boldsymbol{\Psi_{mMIG}.Ab_2}$: When upper node is *minority* and bottom node is a *majority* node.

   (a) $m(x, y, M(x, y, z)) \leftrightarrow m(x, y, z)$

3.5.4 Swapping Reconvergence rule $\Psi_{mMIG}.SR$ in mMIG

We propose *swapping reconvergence* rule in mMIG synthesis as follows. It includes swapping of *reconvergent* and *nonreconvergent* Boolean variables comprising either *complemented* or *regular* variables, either from input to output or vice-versa, as per the specific application while implementing the circuit. Inverter reduction is attained by using swapping of *reconvergent* variable or *nonreconvergent* variable either with another *reconvergent* variable or *nonreconvergent* variable as depicted in the Figure 3.14. A

simple circuit instance shown in Figure 3.15 depicts depth and size optimization using multiple variants of *swapping reconvergence* property in `mMIG`.

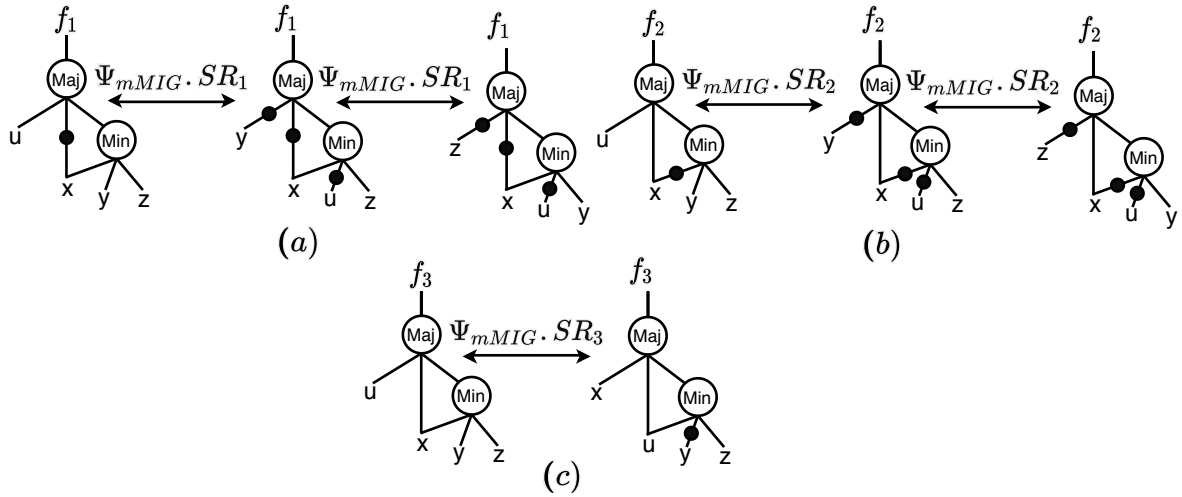

Figure 3.14: (a),(b) depicts the swapping of *nonreconvergent* Boolean variables with *nonreconvergent* Boolean variables, which reduces inverter count if propagation is from the right-hand side to the left-hand side; the same swapping rule applies in (c) also between *reconvergent* and *nonreconvergent* Boolean variable.

The transformation rules in this category are as follows:

(i) $\Psi_{mMIG}.SR_1$:$M(u,\bar{x},m(x,y,z)) \leftrightarrow M(\bar{y},\bar{x},m(x,\bar{u},z)) \leftrightarrow M(\bar{z},\bar{x},m(x,y,\bar{u}))$,

(ii) $\Psi_{mMIG}.SR_2$:$M(u,x,m(\bar{x},y,z)) \leftrightarrow M(\bar{y},x,m(\bar{x},\bar{u},z)) \leftrightarrow M(\bar{z},x,m(\bar{x},y,\bar{u}))$,

(iii) $\Psi_{mMIG}.SR_3$:$M(u,x,m(x,y,z)) \leftrightarrow M(x,u,m(u,\bar{y},\bar{z}))$

3.5.5 Relevance $\Psi_{mMIG}.R$ rule in `mMIG`

The proposed rule shows *relevance* nature of the *reconvergent* variable. It signifies the importance of specific input Boolean variables present in a critical path, which lies either at the bottom-most position or in any intermediate position. This property is convenient in case of a critical variable which is present in a critical path, and it renders `mMIG` synthesized circuit highly optimized. Ψ_{mMIG} is having added advantage over Ψ_{MIG} and Ψ_{mIG} as depicted in the Figure 3.16.

The transformation rules in this category are as follows:

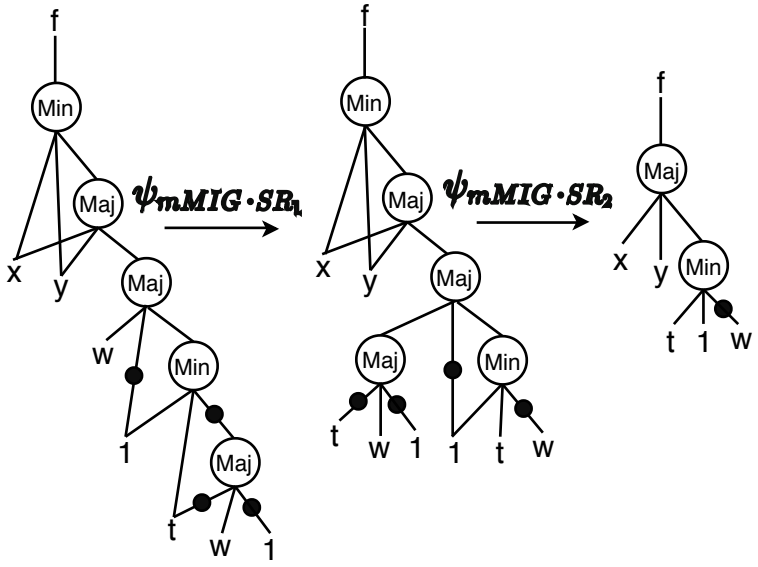

Figure 3.15: Depth and size optimization using *swapping reconvergence* rules $\Psi_{mMIG}.SR_1$ and $\Psi_{mMIG}.SR_2$ in `mMIG` circuit. Inverter operation count is reduced from two inversion operations to one inversion operation.

(i) $\Psi_{mMIG}.R_1$:$M(x, y, m(M(x, y, z), \bar{z}, w)) \leftrightarrow M(x, y, \bar{w})$

(ii) $\Psi_{mMIG}.R_2$:$M(\bar{x}, \bar{y}, m(M(\bar{x}, \bar{y}, z), \bar{z}, w)) \leftrightarrow M(\bar{x}, \bar{y}, \bar{w}) \leftrightarrow m(x, y, w)$

(iii) $\Psi_{mMIG}.R_3$:$M(\bar{x}, \bar{y}, m(M(x, y, z), \bar{z}, w)) \leftrightarrow M(\bar{x}, \bar{y}, \bar{w}) \leftrightarrow m(x, y, w)$

(iv) $\Psi_{mMIG}.R_4$:$M(x, y, m(M(\bar{x}, \bar{y}, z), \bar{z}, w)) \leftrightarrow M(x, y, z)$

(v) $\Psi_{mMIG}.R_5$:$m(x, y, M(m(x, y, z), \bar{z}, w)) \leftrightarrow M(\bar{x}, \bar{y}, z) \leftrightarrow m(x, y, \bar{z})$

(vi) $\Psi_{mMIG}.R_6$:$m(\bar{x}, \bar{y}, M(m(\bar{x}, \bar{y}, z), \bar{z}, w)) \leftrightarrow M(x, y, 0)$

(vii) $\Psi_{mMIG}.R_7$:$m(\bar{x}, \bar{y}, M(m(x, y, z), \bar{z}, w)) \leftrightarrow M(x, y, z)$

(viii) $\Psi_{mMIG}.R_8$:$m(x, y, M(m(\bar{x}, \bar{y}, z), \bar{z}, w)) \leftrightarrow M(x, y, 0)$

3.6 `mMIG` Logic Optimization Algorithm

In this section, we illustrate logic optimization algorithms in `mMIG` synthesis that renders size optimization and reduction of *inversion* operations through *elimination*,

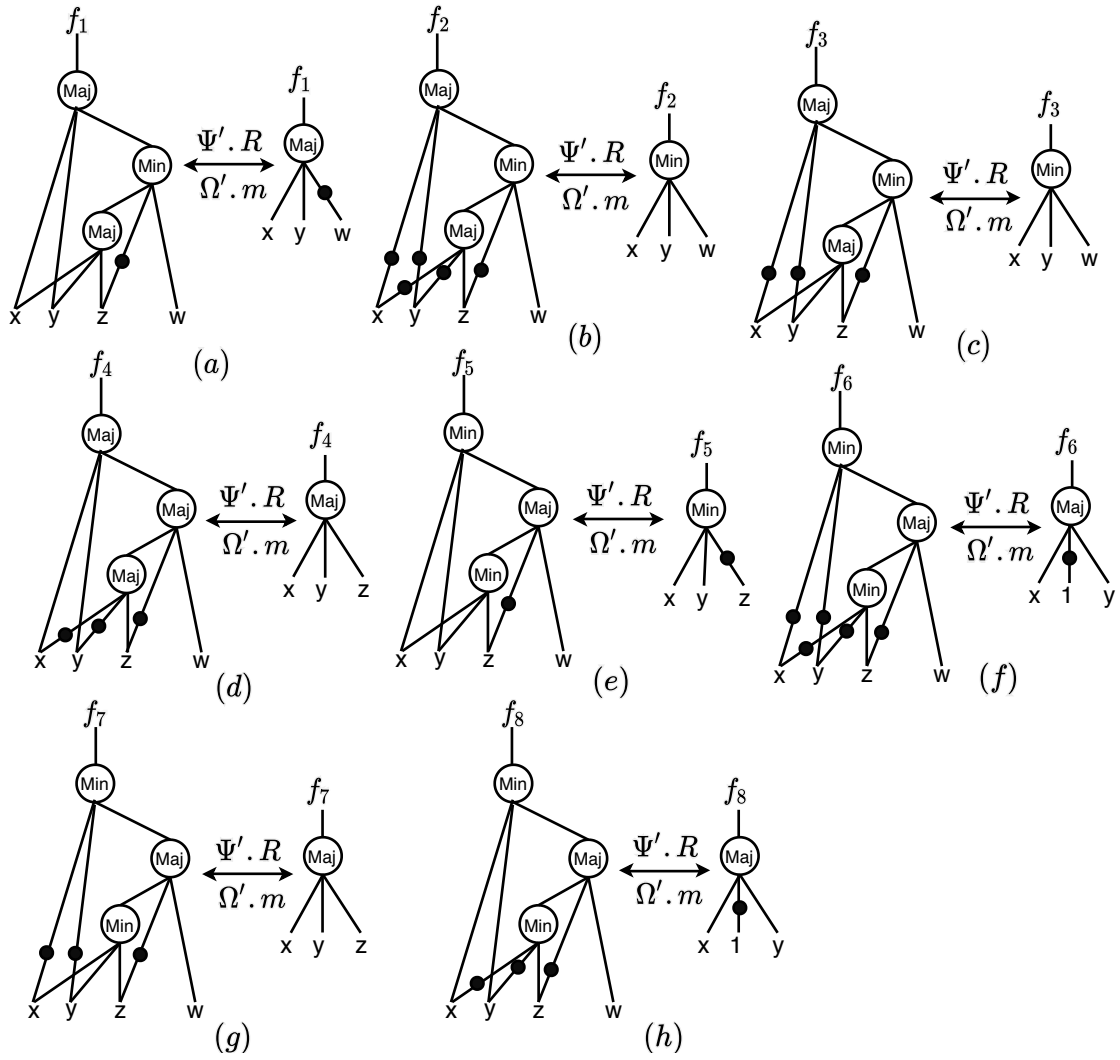

Figure 3.16: In (a),(b) and (c), replace x with y' for occurrence of z at the bottom node by using $\Psi'.R$. This is followed by application of $\Omega'.m$ which results in intermediate input Boolean critical variable to reduce to resultant *majority* node/*minority* node. In (d),(e),(f),(g) and (h), the resultant node contains the input Boolean variable that is input to the leaf node as a *majority* node or *minority* node.

absorption, *swapping reconvergence*, *relevance*, and other transformation rules in *minority* and *majority* logic synthesis methods.

The logic optimization algorithm is demonstrated in Algorithm 1. It takes the `MIG` synthesized circuit as the input which is obtained by direct conversion from AND-OR-Inverter Graph, i.e., $AOIG \rightarrow MIG$. and applies a set of optimization algorithms that comprise `XOR` optimization, `CE` reduction, *reshaping*, *inversion propagation*, and `mMIG`

size optimization algorithms. Depending on the conditions satisfied, such as existence or parity of count of XOR operations or scope of `CE` reduction, the optimization algorithms are applied to yield a logic optimized circuit comprising *majority* nodes, *minority* nodes, and `CE`s.

Algorithm 1: *mMIG Logic Optimization Algorithm*

Input: Circuit α in MIG representation
Output: Optimized circuit α' in mMIG representation
if *(XOR_operations_in(α))* **then**
  $\alpha \leftarrow XOR_n$*_Optimization(α);*
end
else if *(Check_Reduction_CE(α))* **then**
  $\alpha \leftarrow$ *Reshape(α);*
  `//` $\mathcal{O}(M)$ `where M = number of Majority nodes.`
  $\alpha \leftarrow$ *INV_propagation(α);*
  `//` $\mathcal{O}(M)$ `where M = number of Majority nodes.`
end
else
  $\alpha' \leftarrow \alpha$*;*
  return*(α');*
end
$\alpha' \leftarrow$ *mMIG_Size_Optimization(α);*
return*(α');*

3.6.1 XOR_n Optimization in mMIG

The *XOR_n optimization* step first identifies all two-input XOR gates (denoted as XOR_2) and three-input XOR (denoted as XOR_3) gates, and optimizes the corresponding `MIG` synthesized partitions with `mMIG` synthesis approach as mentioned later in Section 4.4. We further extend these transformations towards optimization of n-input XOR gates in the circuit. We demonstrate that optimal circuit implementations for even and odd values of n shall have different realizations.

Algorithm 2: *Check_Reduction_CE(α)*

Input: Circuit α in MIG representation
Output: TRUE/FALSE
for *Each majority node M in* α **do**
  if $((M(\overline{x},\overline{y},z)$——$M(\overline{x},\overline{y},\overline{z})$—— $\overline{M}(x,y,z)$——$\overline{M}(\overline{x},y,z))$ **then**
    return(TRUE);
  end
  else if $(M(\overline{x},a,\overline{z})\&\&(a == constant))$ **then**
    return(TRUE);
  end
  else if $(\overline{M}(x,a,z)\&\&(a == constant))$ **then**
    return(TRUE);
  end
  else if $(\overline{M}(x,a,b)\&\&(a == constant)\&\&(b == constant))$ **then**
    return(TRUE);
  end
  else
    return(FALSE);
  end
end

3.6.2 Check_Reduction_CE Algorithm

The *Check_Reduction_CE* algorithm, described in Algorithm 2, identifies locations in MIG synthesized circuit where the complemented edges (CEs) can be reduced. Instances of nodes where CEs can be reduced have expressions such as, $M(\overline{x},\overline{y},z)$, $M(\overline{x},\overline{y},\overline{z})$, $\overline{M}(x,y,z)$, and $\overline{M}(\overline{x},y,z)$. After identification, such instances of *majority* operations are replaced with *minority* operation as mentioned in *Reshaping* algorithm in the next section.

3.6.3 Reshaping Algorithm

The *Reshaping* algorithm replaces the identified *majority* nodes (in previous section) with larger number of CEs with *minority* nodes.

As mentioned in Algorithm 3, the *majority* nodes were identified based on the polarities at the output of *majority* node, and that of the input signals, x, y, and z as mentioned in the algorithm. Instances of the *reshaping* process are shown in the

Figure 3.17

3.6.4 INV_propagation Algorithm

The *INV_propagation* algorithm, described in Algorithm 4, enumerates all cases wherein propagation of *inversion* operation from input to output, or vice versa leads to a reduced count of CEs in mMIG synthesis.

The algorithm identifies all such cases wherein either the *majority* node is restored or substituted with a *minority* node after propagation of CE. Some instances of the inverter propagation are shown in Figure 3.18

Algorithm 3: *Reshape(α)*

Input: Circuit α in MIG representation
Output: Transformed mMIG synthesized circuit α'
for (*each majority node* $M(x, y, z)$ *in* α) **do**
  switch (M, x, y, z) **do**
    case $(M, \overline{x}, \overline{y}, z)$ **do**
      $m(x, y, \overline{z}) \leftarrow M(\overline{x}, \overline{y}, z)$
    case $(M, \overline{x}, \overline{y}, \overline{z})$ **do**
      $m(x, y, z) \leftarrow M(\overline{x}, \overline{y}, \overline{z})$
    case $(\overline{M}, x, y, z)$ **do**
      $m(x, y, z) \leftarrow \overline{M}(x, y, z)$
    case $(\overline{M}, \overline{x}, y, z)$ **do**
      $m(\overline{x}, y, z) \leftarrow \overline{M}(\overline{x}, y, z)$
    case $(M, \overline{x}, a, \overline{z})$ **do**
      $m(x, \overline{a}, z) \leftarrow M(\overline{x}, a, \overline{z})$
    case $(\overline{M}, x, a, z)$ **do**
      $m(x, a, z) \leftarrow \overline{M}(x, a, z)$
    case $(\overline{M}, x, a, b)$ **do**
      $m(x, a, b) \leftarrow \overline{M}(x, a, b)$
  end
  $\alpha' \leftarrow \alpha$;
  return(α');
end

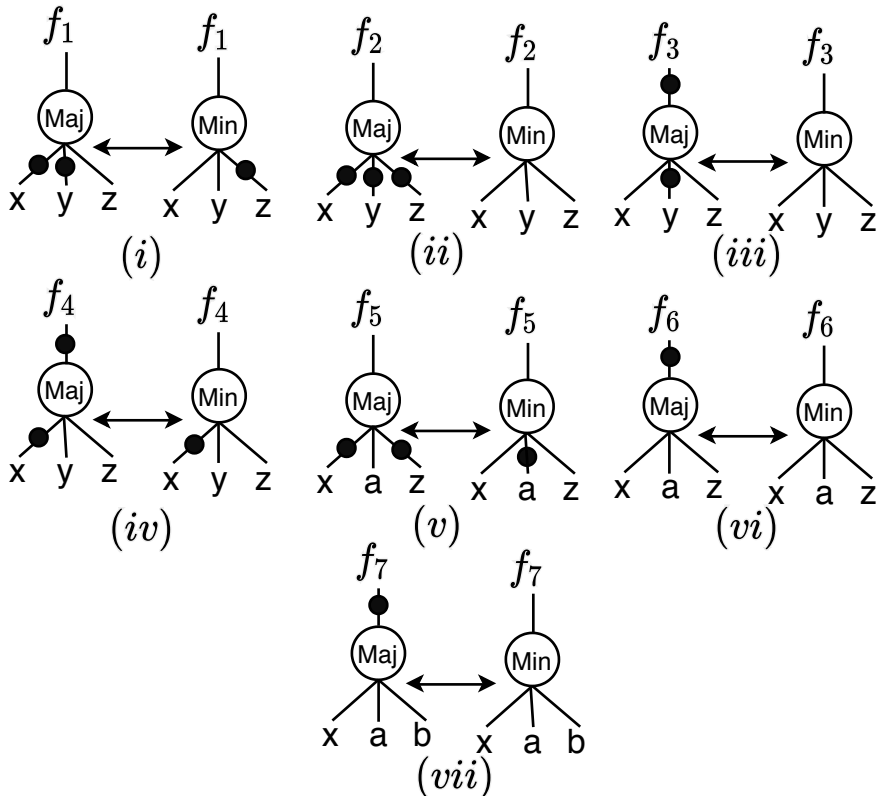

Figure 3.17: Reshaping process in Algorithm 3 on circuit α transforms circuits in $(i), \ldots, (vii)$ to attain compact implementation of the circuit, where x, y, z are input Boolean variables, a and b are constants, and $f_1, \ldots, f_7$ are output functions.

3.6.5 Filtering CEs Algorithm

We define two CEs are connected if they share a common majority node. Based on the count of connected complemented edges (CEs) and disconnected CEs, this algorithm follows two different ways,

(i) if count of disconnected CEs in the circuit is available then replacement operation is performed where it replaces majority node and CE with minority node.

(ii) if count of connected CEs in the circuit is available then it calls INV_propagation algorithm.

3.6.6 mMIG_Size_Optimization Algorithm

The *mMIG_Size_Optimization* algorithm, described in Algorithm 5, initially, splits the `mMIG` synthesized circuit α, into three partitions:

- $P_M(\alpha)$: This partition of m`MIG` synthesized circuit α, comprises a connected set of only majority nodes; the number of nodes in the set, defined as $|P_M(\alpha)| \geq 2$. The constructed partition is subjected to a set of transformation rules from [2], which comprise *majority axiom* ($\Omega.M$), *distributivity* ($\Omega.D$), *associativity* ($\Omega.A$), *complementary associativity* ($\Psi.C$), *relevance* ($\Psi.R$), and *substitution* ($\Psi.S$), which involve a set of connected majority nodes only. The sequence of rules reduces the size of the partition and count of the associated complemented edges (`CE`s).

- $P_m(\alpha)$: This partition of `mMIG` synthesized circuit α, comprises a connected set

Algorithm 4: *INV_propagation(α)*

Input: Circuit α in mMIG representation
Output: Transformed mMIG synthesized circuit α'
for (*each majority node* $M(x, y, z)$ *in* α) **do**
  switch (M, x, y, z) **do**
    case $(\overline{M}, \overline{x}, \overline{y}, z)$ **do**
      $M(x, y, \overline{z}) \leftarrow \overline{M}(\overline{x}, \overline{y}, z)$
    case $(\overline{M}, \overline{x}, \overline{y}, \overline{z})$ **do**
      $M(x, y, z) \leftarrow \overline{M}(\overline{x}, \overline{y}, \overline{z})$
    case $((\overline{M}, \overline{x}, a, \overline{z})\&\&(a == constant))$ **do**
      $M(x, \overline{a}, z) \leftarrow \overline{M}(\overline{x}, a, \overline{z})$
    case $(\overline{M}, \overline{x}, y, z)$ **do**
      $m(\overline{x}, y, z) \leftarrow \overline{M}(\overline{x}, y, z)$
    case $(M, \overline{x}, a, \overline{z})$ **do**
      $m(x, \overline{a}, z) \leftarrow M(\overline{x}, a, \overline{z})$
    case $(\overline{M}, x, a, z)$ **do**
      $m(x, a, z) \leftarrow \overline{M}(x, a, z)$
    case $(\overline{M}, x, a, b)$ **do**
      $m(x, a, b) \leftarrow \overline{M}(x, a, b)$
  end
  $\alpha' \leftarrow \alpha$;
  return(α');
end

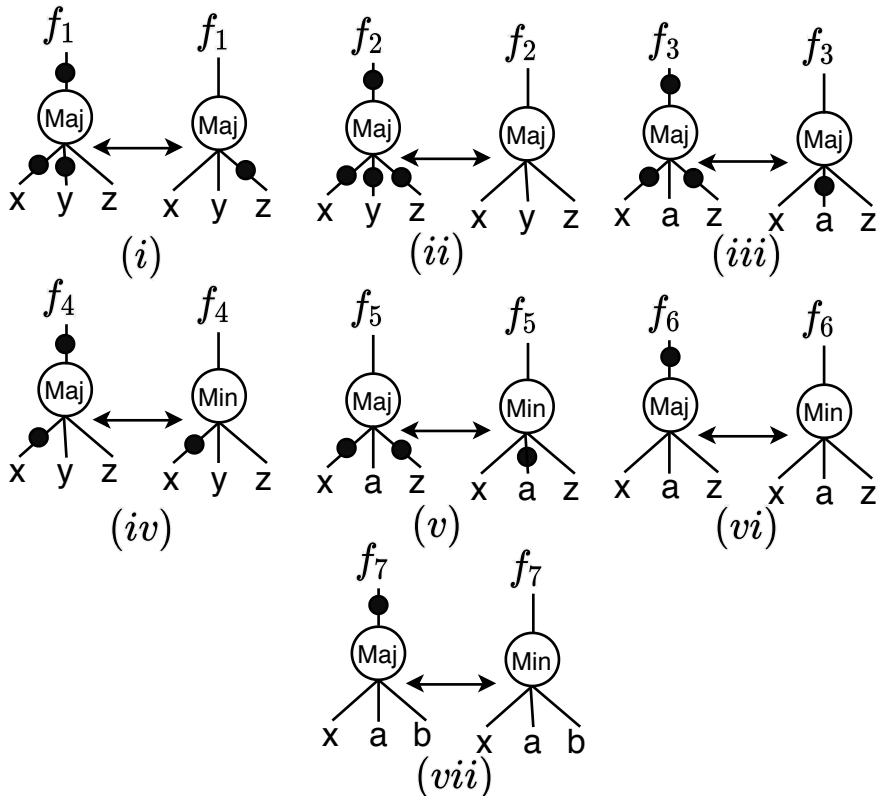

Figure 3.18: Inverter propagation process in Algorithm 4 on circuit α transforms subgraphs $(i), \ldots, (vii)$ to obtain compact implementation of the circuit, where x, y, z are input Boolean variables, a and b are constants and $f_1, \ldots, f_7$ are output functions.

of only *minority* nodes; the number of nodes in the set, defined as $|P_m(\alpha)| \geq 2$. The generated partition is subjected to a set of transformation rules proposed in Section 3.2, which comprise *associativity involving reconvergent and non-reconvergent variables* ($\Omega'.A_{n^2v}, A_{nrv}$), *swapping reconvergence* ($\Psi'.SR$), and *relevance* ($\Psi'.R$). The rules are applied on a set of connected minority nodes only in the partition $P_m(\alpha)$ in a sequence to reduce the size and depth of the partition, and count of the associated complemented edges (CEs) in the partition.

- $P_{mM}(\alpha)$: This partition defined as, $\alpha \setminus P_M(\alpha) \setminus P_m(\alpha)$, is the residual partition after computing $P_M(\alpha)$ and $P_m(\alpha)$. The partition is a connected component comprising majority nodes, minority nodes, and complementary edges (CEs). Computing a connected system or a subpartition of only majority

nodes or only minority nodes is not possible in this partition. For any P_M in this partition, defined as, $P_M(P_{mM}(\alpha))$, the cardinality of such a set satisfies, $|P_M(P_{mM}(\alpha))| \leq 1$. Similarly, for a partition comprising only minority nodes, the cardinality of the set satisfies $|P_m(P_{mM}(\alpha))| \leq 1$. A set of transformation rules from mMIG synthesis is proposed in Section 3.4. The set comprises *elimination* ($\Psi_{mMIG}.El$), *absorption* ($\Psi_{mMIG}.Ab_1, \Psi_{mMIG}.Ab_2$), *swapping reconvergence* ($\Psi_{mMIG}.SR_1, \ldots, \Psi_{mMIG}.SR_3$), and *relevance* ($\Psi_{mMIG}.R_1, \ldots, \Psi_{mMIG}.R_8$), which are applied in a specific sequence to reduce the CEs in the partition, $P_{mM}(\alpha)$.

The run-time complexities of these algorithms are mentioned in Table 3.1.

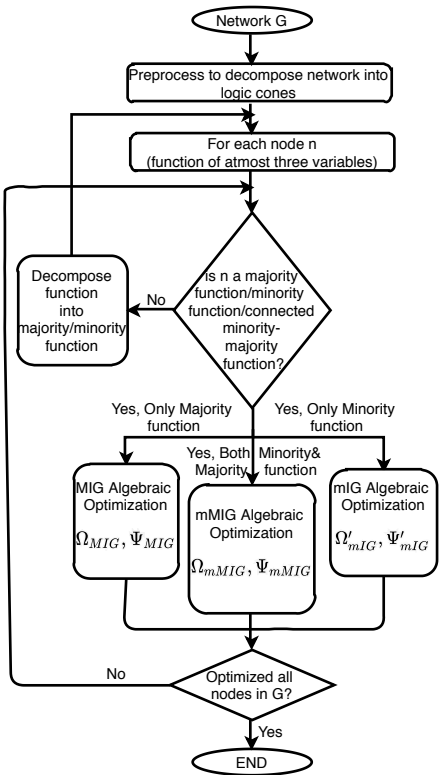

Figure 3.19: Flow diagram providing an overview of our minority-majority network optimization methodology.

Algorithm 5: *mMIG_Size_Optimization(α)*

Input: Circuit α in mMIG representation
Output: Size optimized mMIG synthesized circuit α'
Create partition $P_M(\alpha)$ comprising only $M(x, y, z)$ nodes, $|P_M(\alpha) \geq 2|$;
// Partition $P_M(\alpha) \subset \alpha$, comprising majority nodes only, # nodes ≥ 2
Create partition $P_m(\alpha)$ comprising only $m(x, y, z)$ nodes, $|P_m(\alpha) \geq 2|$;
// Partition $P_m(\alpha) \subset \alpha$, comprising minority nodes only, # nodes ≥ 2
$P_{mM}(\alpha) \leftarrow \alpha \setminus P_M(\alpha) \setminus P_m(\alpha)$;
Check partition $P_{mM}(\alpha)$ is a connected component and $\forall P_M, P_m$, s.t.,
$|P_M(P_{mM}(\alpha)) \leq 1|$ && $|P_m(P_{mM}(\alpha)) \leq 1|$; // Partition $P_{mM}(\alpha)$ comprising connected majority and minority nodes only, # nodes in each $P_M \subset P_{mM}$ and $P_m \subset P_{mM}$ is ≤ 1
for $(\alpha_1 \leftarrow P_M(\alpha))$ **do**
    $\Omega.M_{L\rightarrow R}(\alpha_1)$; // Optimizations on $P_M(\alpha)$ partition
    $\Omega.D_{R\rightarrow L}(\alpha_1)$;
    $\Omega.A(\alpha_1)$; $\Psi.C(\alpha_1)$; // From [2]
    $\Psi.R(\alpha_1)$; $\Psi.S(\alpha_1)$;
    $\Omega.M_{L\rightarrow R}(\alpha_1)$; $\Omega.D_{R\rightarrow L}(\alpha_1)$;
end
for $(\alpha_2 \leftarrow P_m(\alpha))$ **do**
    $\Omega'.m_{L\rightarrow R}(\alpha_2)$; // Optimizations on $P_m(\alpha)$ partition
    $\Omega'.D_{R\rightarrow L}(\alpha_2)$;
    $\Omega'.A_{n^2v}(\alpha_2)$; $\Omega'.A_{nrv}(\alpha_2)$;
    $\Psi'.SR^1_{n^2v}(\alpha_2)$; $\Psi'.SR^2_{n^2v}(\alpha_2)$;
    $\Psi'.SR^3_{n^2v}(\alpha_2)$; $\Psi'.SR^4_{n^2v}(\alpha_2)$;
    $\Psi'.SR^1_{nrv}(\alpha_2)$; $\Psi'.SR^2_{nrv}(\alpha_2)$;
    $\Psi'.SR^3_{nrv}(\alpha_2)$; $\Psi'.SR^4_{nrv}(\alpha_2)$;
    $\Psi'.R^(\alpha_2)$; $\Psi'.S(\alpha_2)$;
    $\Omega'.m_{L\rightarrow R}(\alpha_2)$; $\Omega'.D_{R\rightarrow L}(\alpha_2)$;
end
for $(\alpha_3 \leftarrow P_{mM}(\alpha))$ **do**
    $\Psi_{mMIG}.El(\alpha_3)$; // Optimizations on $P_{mM}(\alpha)$ partition
    $\Psi_{mMIG}.Ab_1(\alpha_3)$;
    $\Psi_{mMIG}.Ab_2(\alpha_3)$; $\Psi_{mMIG}.SR_1(\alpha_3)$;
    $\Psi_{mMIG}.SR_2(\alpha_3)$; $\Psi_{mMIG}.SR_3(\alpha_3)$;
    $\Psi_{mMIG}.R_1(\alpha_3)$; $\Psi_{mMIG}.R_2(\alpha_3)$;
    $\Psi_{mMIG}.R_3(\alpha_3)$; $\Psi_{mMIG}.R_4(\alpha_3)$;
    $\Psi_{mMIG}.R_5(\alpha_3)$; $\Psi_{mMIG}.R_6(\alpha_3)$;
    $\Psi_{mMIG}.R_7(\alpha_3)$; $\Psi_{mMIG}.R_8(\alpha_3)$;
end
$\alpha' \leftarrow \alpha_1 \cup \alpha_2 \cup \alpha_3$;
return(α');

Algorithm 6: *Filtering CEs*

Input: Circuit α in MIG representation
Output: Optimized circuit α' in mMIG representation
if *(Count_Disconnected_CEs_ In(α))* **then**
  Replacement Majority_logic and CEs_in_α $\leftarrow$ Minority_logic;
end
; `// ` $\mathcal{O}(m) * DCEs$ `where DCEs= #Disconnected CEs, m =# minority nodes`
else if *(Count_Connected_CEs_In(α))* **then**
  $\alpha \leftarrow$ INV_propagation(α);
  ; `// ` $\mathcal{O}(M) * CCEs$ `where CCEs= #Connected CEs, M =# Majority nodes`
end
return(α');

Table 3.1: Timing complexities of different components of mMIG Algorithm.

mMIG Algorithm	**Worst case time complexity**	**Termination Condition/ Remarks**
mMIG Logic Optimization Algorithm	max($\mathcal{O}(X)$, $\mathcal{O}(M^2)$, *time complexity(mMIG_Size_Optimization Algorithm)*)	Where X is the number of XOR operations, M is the total number of majority nodes in the MIG circuit.
Check_Reduction_CE Algorithm	$\mathcal{O}(M)$ where M is number of Majority nodes.	When there is no majority node satisfying the mentioned condition.
Reshaping Algorithm	$\mathcal{O}(M)$ where M is number of Majority nodes.	When there is no majority node satisfying the mentioned condition.
INV_propagation Algorithm	$\mathcal{O}(M)$ where M is number of Majority nodes.	When there is no majority node satisfying the mentioned condition.
Filtering_CE Algorithm	max($\mathcal{O}(m * DCEs)$, $\mathcal{O}(M * CCEs)$)	m and M are the number of minority, majority nodes respectively, and DCEs, CCEs is the total count of disconnected edges, connected edges. respectively.

Chapter 4

Experimental Results: Case Study and Comparison

In this section, we have shown the comparison between MIG and mMIG implementations of standard combinatorial circuits, lightweight cryptographic block ciphers and cyptoprimitives [62]. All the comparisons are shown in this paper made by counting the resources in both the structure which includes number of *majority*, *minority* and CEs. This paper mainly investigates the reduction in CEs through counting and percentage of CE reduction.

We have used two different approaches to obtain synthesized circuit with minimal number of resources especially in the term of CEs.

First approach: It includes the optimization of the optimised MIG synthesized circuit from [2]. To achieve this goal, we have written script called, *mMIGhty* tool to convert the MIG synthesized circuit to mMIG synthesized circuit using minority logic where inverters are suppressed using NAND and NOR equivalent representation. It reads files in Verilog format and writes back a Verilog description of the optimized mMIG. Suppose Boolean variable x, y and z are two input logical operations like and, or represented using &, | operators respectively. Assume $x = \bar{y}\&\bar{z}$, we have converted x into $\overline{(y|z)}$ which requires 0 inverter if it does using minority logic, result of which equivalent circuit but with lesser number of CEs.

Second Approach: Another approach comes under the complete construction of the circuit from the network graph(lets say G) whose complete flow can be observed in

the Figure 3.19, where resources are optimized based on the nature of the function. We have used connected property of CEs in above discussed both the approaches. We define two CEs are connected if they share a common majority node. Circuits with smaller connectedness restricts optimization in inverter count in MIG-based implementations. In such cases, mMIG-based implementations can lead to significant inverter reduction by capturing the inversion within minority node, wherever applicable in the circuit.

we perform a comparative assessment of the logic optimization algorithm that we proposed for `mMIG` implementations with corresponding `MIG` implementations of standard combinational circuits from IWLS'05 open core and arithmetic benchmark circuits, EPFL benchmark circuits, ISCAS'85 benchamrk circuits, and combinational circuits that form the core of certain cryptographic primitives in both private and public key cryptography. Compact implementation of such cryptographic primitives is important to meet resource-constrained requirements of lightweight cryptography [62, 63] in terms of hardware and energy footprint, hence, they have been considered for comparative evaluation of the proposed `mMIG` synthesis technique.

4.1 IWLS'05 Open Core benchmarks and larger arithmetic HDL benchmarks

In this section, we provide performance results of mMIG implementations of IWLS2005 benchmark and arithmetic benchmark circuits on Kintex UltraScale FPGA platform on 20nm VLSI technology.The IWLS 2005 benchmark suite was published by International Workshop on Logic and Synthesis (IWLS). It comprises MCNC, ISCAS and ITC'99 suites and few OpenCores along with other designs [64].

Metrics used for comparison between MIG with mMIG: In addition to inverter reduction in mMIG circuit synthesis, we consider other performance metrics, such as *signal switching power*, *resource count* in LUTs, and *total net delay in the critical path* of the synthesized circuit as follows:

(i) Reduction in inverter count reduces number of nodes in the circuit. As a result, it also decreases the total switching power of signals in the circuit. This aspect is demonstrated in Figure 4.4, wherein significant reduction in signal switching power is observed in mMIG synthesized circuits.

(ii) In the critical path, reduction in inverter count reduces the number of nets from the primary inputs to the primary outputs. Hence, it reduces the total net delay in the entire critical path. This is observed in comparison of total net delay in the critical path for MIG-synthesized and mMIG-syntheized circuits as shown in Figure 4.1.

FPGA Implementation results: In Table 4.1, we demonstrate the efficacy of mMIG synthesis technique in comparison to MIG synthesis with performance metrics stated in previous section. The results are shown on 15 benchmark circuits, which comprises eight open core IWLS'05 benchmark circuits and seven Arithmetic HDL circuits. In addition, we demonstrate comparative results of *net delay* in critical path, *logic delay*, *LUT count* and *on-chip dynamic signal power* in Figure 4.1, Figure 4.2, Figure 4.3, and Figure 4.4, respectively.

Figure 4.1 shows the improvement in net delay in mMIG synthesis as compared to MIG logic synthesis technique. There are some circuits namely, $c10$, $c12$, where it does not improve parameters due to dominant presence of AND and OR logic gates which are optimally represented in MIG than $mMIG$. Figure 4.2 demonstrates improvements in the logic delay in some circuits. In such circuits, replacing majority and inverter with minority eliminates the fan-out delay of the majority and fan-in delay of inverter. Due to this reduced fan-in and fan-out delay of the system, the overall propagation delay or logic delay of the circuit also reduces. Figure 4.3 compares the LUT count which shows significant improvement in most of the cases, result of which mMIG synthesized circuit requires lesser resource count. Figure 4.4 compares on-chip dynamic signal power which shows significant improvement. Elimination of inverter nodes in certain cases reduces the total number of signal transitions in the overall circuit, hence, the dynamic switching power of the signals also reduces.

Table 4.1: Performance Metrics comparison between MIG and mMIG implementations of IWLS'05 and Arithmetic HDL benchmark circuits on 20-nm technology node of Kintex UltraScale FPGAs.

Benchmark	#I/Os	On-Chip Power				Resources	Critical Path Parameters			
		Dynamic			Static					
Open Core IWLS'05		Signal	Logic	I/O		(CLB LUTs)	# max levels	Total delay (ns)	Logic delay (ns)	Net delay (ns)
MIG_usb_phy	113/111	1.615W (5%)	0.25W (1%)	29.1W (94%)	1.147W (3%)	82	5	9.919	1.709	8.21
mMIG_usb_phy		1.615W (5%)	0.25W (1%)	29.1W (94%)	1.167W (3%)	80	5	6.366	1.678	4.688
MIG_ss_pcm	106/98	1.375W (4%)	0.254 W (1%)	31.72W(95%)	1.336W	71	5	7.031	1.707	5.324
mMIG_ss_pcm		2.044W (4%)	0.265 W (1%)	47.896W(95%)	1.336W	68	5	5.779	1.529	4.25
MIG_sasc	133/132	2.752W(5%)	0.414W(1%)	45.56W (94%)	2.21W	127	4	7.8	1.729	6.071
mMIG_sasc		3.821W(5%)	0.445W(1%)	68.257w (94%)	2.353W	127	5	6.378	1.69	4.688
MIG_simple_spi	148/147	3.252W (6%)	0.557W (1%)	46.804W (93%)	2.376W(4%)	183	6	9.477	1.834	7.643
mMIG_simple_spi		3.168W (6%)	0.532W (1%)	46.752W (93%)	2.361W(4%)	184	6	8.821	1.729	7.092
MIG_pci_spoci_ctrl	85/76	1.275W(6%)	0.407W(2%)	22.871W (93%)	0.926W	178	8	9.308	2.151	7.157
mMIG_pci_spoci_ctrl		1.782W(6%)	0.512W(2%)	28.446W (92%)	0.886W	221	6	8.9	1.99	6.91
MIG_i2c	147/142	2.962W(7%)	0.618W (1%)	34.529W (92%)	1.482W	220	6	9.228	1.887	7.341
mMIG_i2c		3.696W(7%)	0.63W (1%)	51.74W (92%)	1.539W	219	6	8.147	1.811	6.336
MIG_spi	274/276	10.123W (11%)	4.104W (4%)	78.073W (85%)	4.39W (5%)	806	9	11.072	2.105	8.967
mMIG_spi		8.453W (11%)	3.06W (4%)	65.243W (85%)	4.39W (5%)	657	9	11.15	2.099	9.048
MIG_des_area	368/72	13.498W (19%)	4.285W (5%)	54.886W (75%)	4.391W (6%)	883	10	16.837	2.024	14.813
mMIG_des_area		12.657W (18%)	3.828W (5%)	54.946W (77%)	4.391W (6%)	789	9	16.004	1.99	14.014
Arithmetic HDLs										
MIG_hamming	200/7	34.391W(50%)	24.605W(36%)	9.439W (14%)	4.391W(6%)	517	18	17.544	3.037	14.507
mMIG_hamming		29.12W(47%)	26.116W(42%)	6.302W (11%)	1.766W(3%)	492	18	16.272	4.204	12.068
MIG_sqrt32	32/16	14.841W(43%)	12.593W(37%)	6.885W(20%)	1.291W (4%)	492	59	38.741	8.448	30.293
mMIG_sqrt32		13.932W(43%)	11.252W (35%)	6.910W (22%)	1.193W (4%)	435	59	40.202	9.424	30.778
MIG_square	64/127	96.5 W (41%)	61.5W (26%)	77.83W (33%)	4.4W (2%)	3942	18	22.045	3.547	18.5
mMIG_square		73.3 W (24%)	54.62 W (19%)	77.69 W (57%)	4.39 W (2%)	3352	17	20.112	3.553	16.56
MIG_div16	32/32	8.513W (36%)	6.912W (30%)	7.923W (34%)	0.891W	1060	41	29.659	6.682	22.977
mMIG_div16		8.322W (37%)	6.524W (29%)	7.889W (34%)	0.874W	1043	41	30.97	6.81	24.15
MIG_tv80	373/404	7.971W (13%)	2.737W (5%)	43.868W (82%)	2.768W (6%)	740	14	17.5	2.842	14.659
mMIG_tv80		8.018W (15%)	2.812W (5%)	43.857W (80%)	2.780W (5%)	754	14	16.779	2.767	14.012
MIG_revx	20/25	67.347W (46%)	68.705 W (47%)	10.755W (7%)	4.391W (3%)	1612	43	31.306	6.753	24.553
mMIG_revx		54.911W (44%)	60.486W (48%)	10.668W (8%)	4.391W(3%)	1541	43	29.023	7.071	21.952
MIG_log32	32/32	397.149W (53%)	333.7W (45%)	17.59W (2%)	4.391W (1%)	9247	62	51.424	9.289	42.135
mMIG_log32		297.396W (51%)	273.45W (46%)	17.565W (3%)	4.391W (1%)	8090	62	46.083	9.774	36.309

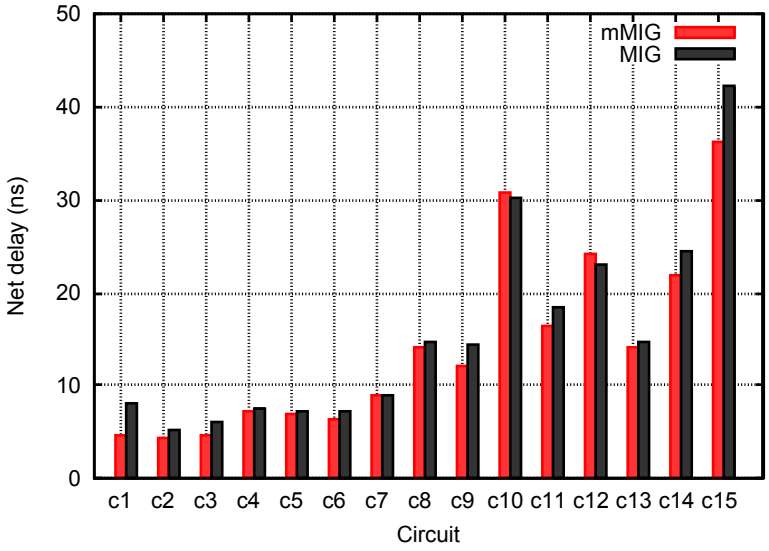

Figure 4.1: Net delays in IWLS'05 and HDL arithmetic benchmark circuits.

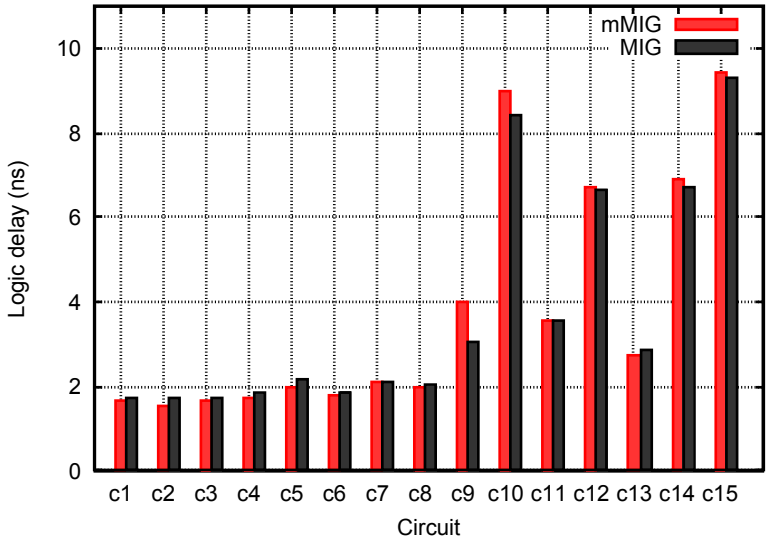

Figure 4.2: Logic delay in IWLS'05 and HDL arithmetic benchmark circuits.

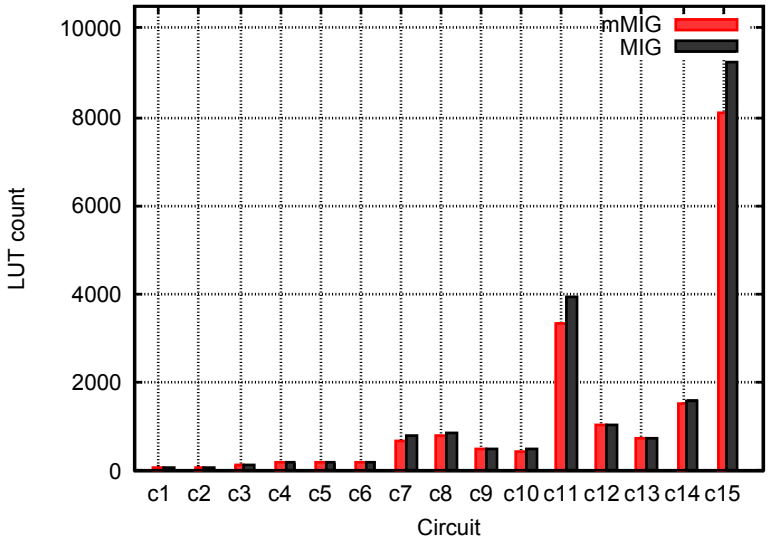

Figure 4.3: LUT count in IWLS'05 and HDL arithmetic benchmark circuits.

4.2 Random/Control Combinatorial Circuits from EPFL benchmark suite

In this section, we demonstrate comparison result in reduction of inversion operations in `mMIG` synthesis as compared to `MIG` synthesis for combinational circuits in

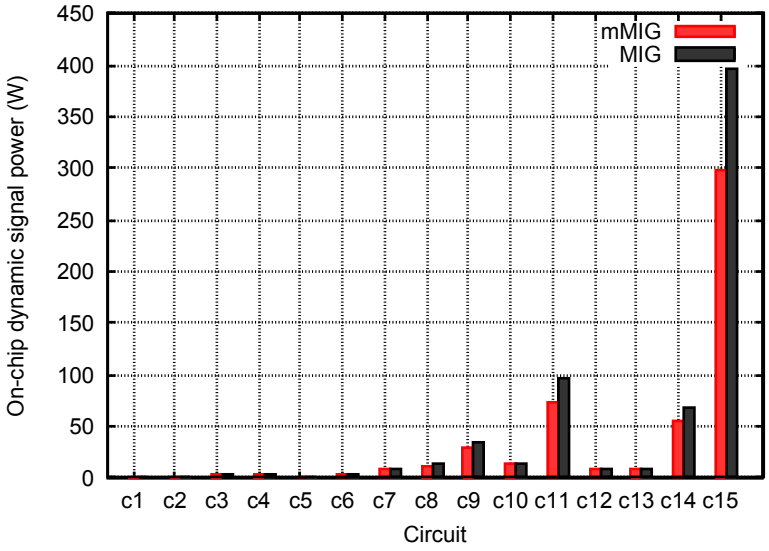

Figure 4.4: On-chip dynamic signal power in IWLS'05 and HDL arithmetic benchmark circuits.

EPFL combinational benchmark suite [65]. These random/control benchmark circuits comprise a significant amount of two-input logical AND operation. Implementation of the AND gate using only MIG creates a huge amount of additional overhead due to CEs, so this overhead is minimized using MMIG representation of the circuits whose underlying representation is shown in Figure 4.5.

4.2.1 ALU Control Unit

This benchmark circuit is a simple ALU Control Unit with seven inputs and 26 outputs. It comprises various signals that control ALU operations, register load operation, memory read/write operations, and jump instructions. The circuit comprises 174 AND operations and 10 logic levels. The resource consumption in number of *majority* nodes, complementary edges (CEs) for all outputs in bothMIG synthesis and MMIG synthesis is shown in Figure 4.6 and the results are illustrated in Table 4.2.

The MMIG synthesis has the largest optimization in cases where it replaces a *majority* node and CE at its output by a *minority* gate. From the results of percentage reduction in CEs shown in Figure 4.8, the MMIG synthesis achieves the largest reduction

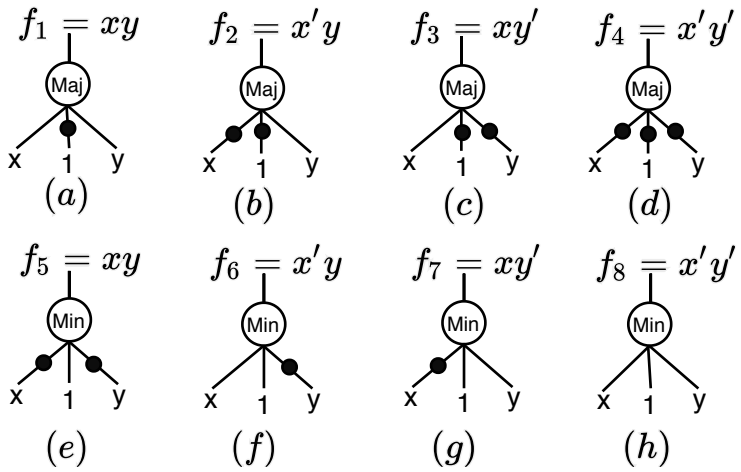

Figure 4.5: MIG representation of two-input logical AND operation considering both the polarities leads to four possible structure shown as (a),(b),(c) and (d), Similarly, (e),(f),(g) and (h) denotes the mIG representation of the same. Fundamental representation (a),(h) will be chosen over (e),(d) to nullify the impact of CEs and (b),(c),(f) and (g) are the similar representation in the view of resource count.

in CEs with 93.3% for halt output signal as compared toMIGsynthesis. However, there is one output signal, *bgez* in the circuit, in which no CEs were removed in mMIG synthesis. For other outputs, the percentage reduction values in CEs due to mMIG synthesis range from 28.6% to 87%, and average percentage reduction in CEs across all outputs to 64.1%.

4.2.2 Int to Float Converter

This EPFL benchmark circuit computes integer to floating point conversion. The input is an integer in binary format with 10 bits. The output is a floating point value with a four bit mantissa signal and three bit exponent. The resource count in number of majority nodes ($\#N_{MIG}$), number of complemented edges ($\#CE_{MIG}$) for MIG, and number of majority nodes ($\#N_{M\_mMIG}$), number of minority nodes ($\#N_{m\_mMIG}$), and number of complemented edges ($\#CE_{mMIG}$) in mMIG synthesis is shown in Figure 4.7 and the results are tabulated in Table 4.3. The results show a large reduction in count of complemented edges for all mantissa output bits in mMIG synthesis as shown in Figure 4.9. For mantissa output bits, the percentage reduction in complementary

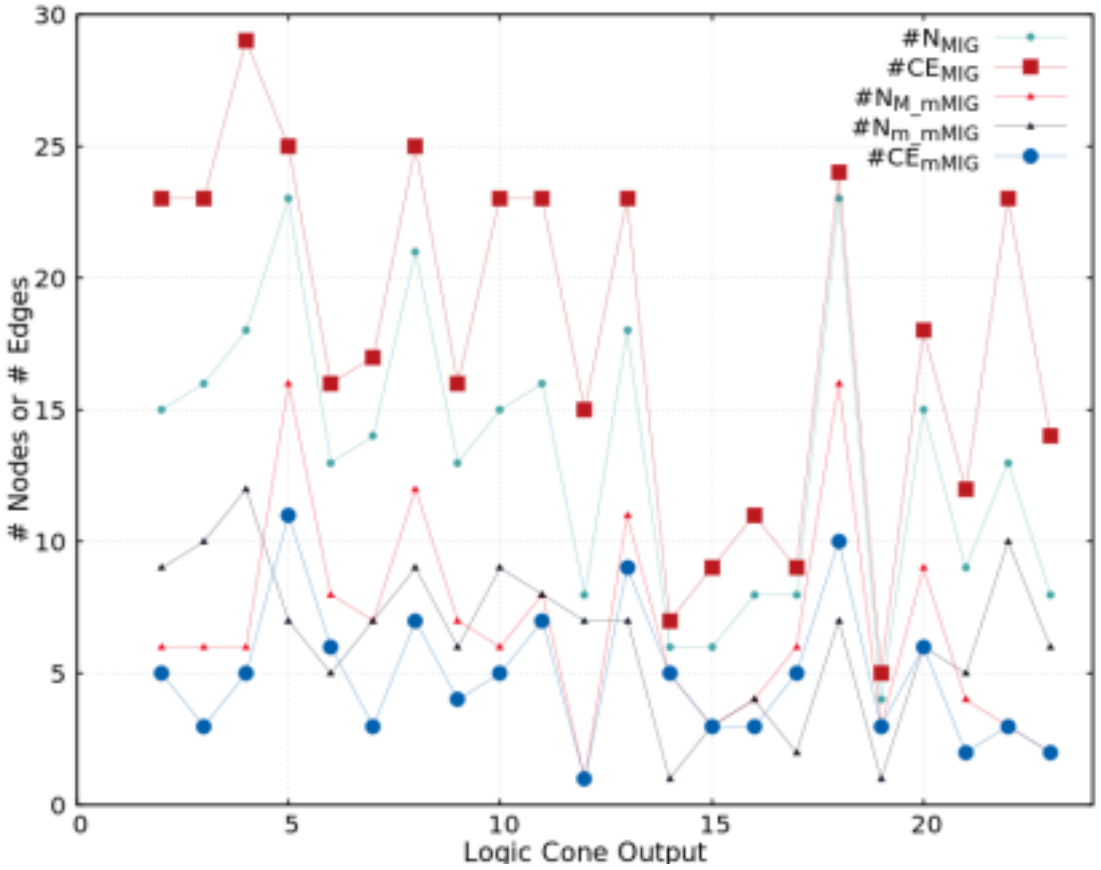

Figure 4.6: Resource count comparison of number of majority modes ($\#N_{MIG}$), complemented edges ($\#CE_{MIG}$) in `MIG` synthesized circuit and number of majority nodes ($\#N_{M\_mMIG}$), minority nodes ($\#N_{m\_mMIG}$), and complemented edges ($\#CE_{mMIG}$) in `mMIG` synthesized circuit in `ALU` Control Unit in EPFL benchmark circuit.

Table 4.2: Resource count of`MIG`and `mMIG` representation of `ALU` control unit w here $\#N_{MIG}$, $\#CE_{MIG}$, $\#N_{M\_mMIG}$, $\#N_{m\_mMIG}$, $\#CE_{mMIG}$ denotes number of *majority* nodes, number of `CE`s in `MIG` synthesized circuit, and number of *majority* nodes, number of *minority* nodes and number of `CE`s in `mMIG` synthesized circuit, respectively.

	`MIG`		`mMIG`		
Output	$\#N_{MIG}$	$\#CE_{MIG}$	$\#N_{MIG}$	$\#N_{mIG}$	$\#CE_{mMIG}$
$Sel\_reg\_dst[0]$	16	14	11	5	4
$Sel\_reg\_dst[1]$	15	23	6	9	5
$Sel\_alu\_opB[0]$	16	23	6	10	3
$Sel\_alu\_opB[1]$	18	29	6	12	5
$alu\_op[0]$	23	25	16	7	11
$alu\_op[1]$	13	16	8	5	6
$alu\_op[2]$	14	17	7	7	3
$alu\_op\_ext[0]$	21	25	12	9	7
$alu\_op\_ext[1]$	13	16	7	6	4
$alu\_op\_ext[2]$	15	23	6	9	5
$alu\_op\_ext[3]$	16	23	8	8	7
$halt$	8	15	1	7	1
reg_write	18	23	11	7	9
Sel_pc_opA	6	7	5	1	5
Sel_pc_opB	6	9	3	3	3
$beqz$	8	11	4	4	3
$bnez$	8	9	6	2	5
$bgez$	8	7	8	0	7
$bltz$	8	9	6	2	5
Cin	23	24	16	7	10
$jump$	4	5	3	1	3
$invA$	15	18	9	6	6
$invB$	9	12	4	5	2
mem_write	13	23	3	10	3
Sel_wb	8	14	2	6	2
Total	322	420	174	148	124

edges in `mMIG` synthesis as compared to `MIG` synthesis are 66.7%, 67.4%, 61.1%, and 66.7%. For exponent outputs, the respective percentage reduction values are 58.8%, 60%, and 83.3%, respectively.

Table 4.3: Resource count of`MIG`and `mMIG` representation of Int to float converter where $\#N_{MIG}$,$\#CE_{MIG}$,$\#N_{M\_mMIG}$,$\#N_{m\_mMIG}$,$\#CE_{mMIG}$ denotes number of majority nodes,number of complemented edges in `MIG` synthesized circuit and number of *majority* nodes, number of *minority* nodes and number of `CE`s in `mMIG` synthesized circuit respectively.

	MIG		mMIG		
Output	$\#N_{MIG}$	$\#CE_{MIG}$	$\#N_{MIG}$	$\#N_{mIG}$	$\#CE_{mMIG}$
$M[0]$	44	54	26	18	18
$M[1]$	71	86	42	29	28
$M[2]$	33	36	22	11	14
$M[3]$	13	12	9	4	4
$E[0]$	37	34	27	10	14
$E[1]$	27	20	21	6	8
$E[2]$	7	6	4	3	1
Total	232	248	151	81	87

4.2.3 Lookahead XY router

This EPFL benchmark circuit is a router implementation for network-on-chips. It represents a lookahead XY router which achieves low latency, high throughput communication in a network-on-chip system. The original AIG implementation comprises 257 nodes with 54 logic levels. The `mMIG` synthesis results is demonstrated in Figure 4.10 and in the Table 4.4. The router comprises 30 output ports, however, the number of active output ports is only *three*, as shown in the figure. The percentage reduction in complemented edges in `mMIG` synthesis with respect to`MIG`synthesis for these three outputs are 57.7%, 58.7%, and 60%, respectively. This large reduction in inverter count implies a significant reduction in hardware footprint as well.

Table 4.4: Resource count of `MIG` and `mMIG` representation of Lookahead XY router where $\#N_{MIG}$,$\#CE_{MIG}$,$\#N_{M\_mMIG}$,$\#N_{m\_mMIG}$,$\#CE_{mMIG}$ denotes number of *majority* nodes, number of `CE`s in `MIG` synthesized circuit and number of *majority* nodes, number of *minority* nodes and number of `CE`s in `mMIG` synthesized circuit, respectively.

	MIG		`mMIG`		
Output	$\#N_{MIG}$	$\#CE_{MIG}$	$\#N_{MIG}$	$\#N_{mIG}$	$\#CE_{mMIG}$
$Output[0]$	393	440	266	127	186
$Output[1]$	867	995	575	292	411
$Output[2]$	858	991	561	297	397
Total	2118	2426	1402	716	994

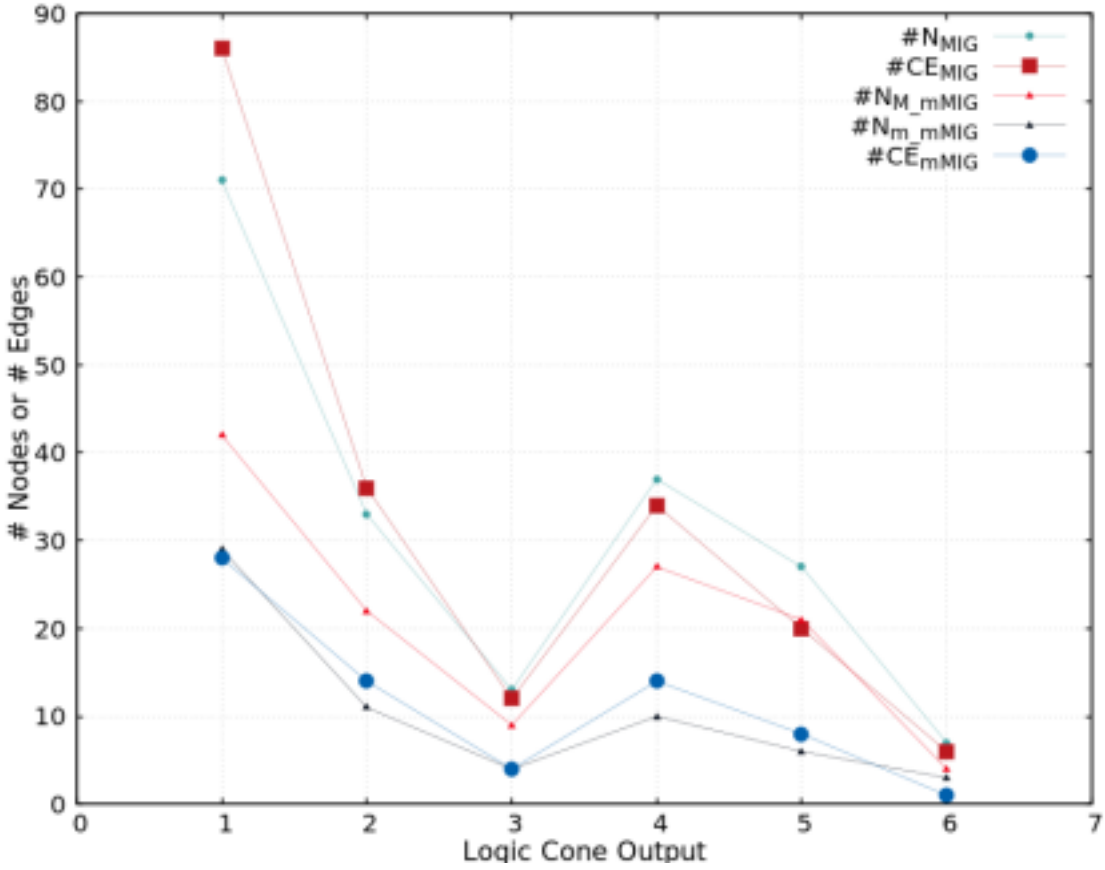

Figure 4.7: Resource count comparison of number of *majority* modes ($\#N_{MIG}$), *complemented* edges ($\#CE_{MIG}$) in `MIG` synthesized circuit and number of *majority* nodes ($\#N_{M\_mMIG}$), *minority* nodes ($\#N_{m\_mMIG}$), and *complemented edges* ($\#CE_{mMIG}$) in `mMIG` synthesized circuit in Int to float circuit in EPFL benchmark circuit.

4.3 *c*432 ISCAS'85 benchmark circuit

The *c*432 ISCAS benchmark circuit is a 27-channel interrupt controller with 36 inputs and 7 outputs, and comprises 160 gates; the outputs are mentioned with labels *N430*, *N431*, *N432*, *N223*, *N329*, *N370*, *N421*. We refer to a circuit that drives each of the seven outputs of *c*432 as a *logic cone*, (LC_i, $1 \leq i \leq 7$) of *c*432. We consider *nand2*, *nand3*, and *nand4* as the binary, ternary, and quaternary $NAND$ logic operations, respectively. We represent each logic cone as $f_i(g_1, g_2, \ldots, g_n)$, a tuple of all logic gates g_1, g_2, $\ldots$, g_n, comprising the logic cone. Further, *nor2*, *xor2*, and *and2* denote binary NOR, binary XOR, and binary AND operations, respectively. The *c*432

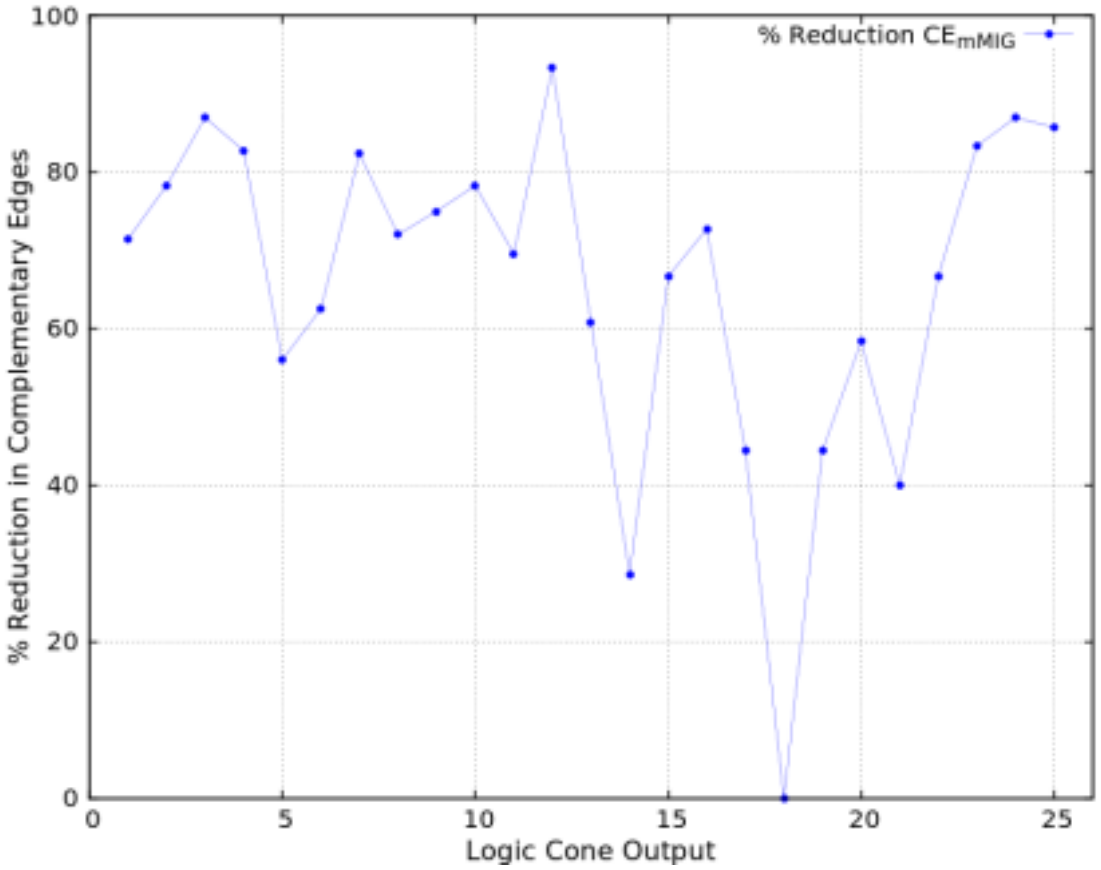

Figure 4.8: Percentage reduction in complementary edges in `mMIG` logic synthesis of EPFL benchmark circuits in `ALU` Control Unit

benchmark circuit comprises of the following seven logic cones as follows:

$\mathrm{LC}_1 : N223 = f_1(nand2, and9), LC_2 : N329 = f_2(nand2, xor2, nor2, and9), LC_3 : N370 = f_3(nand2, nor2, xor2, and9), LC_4 : N421 = f_4(nand4, nand2, xor, nor2, and8), LC_5 : N430 = f_5(nand4, nand2, xor2, nor2), LC_6 : N431 = f_6(nand4, nand2, nand3, xor2, nor2), LC_7 : N432 = f_7(nand4, nand2, xor2, nor2)$

where $f_1, \ldots, f_7$ are the Boolean functions used to compute the output of the respective logic cones. Bulky logic cones among the seven logic cone namely are LC_5, LC_6, LC_7 whose `MIG`, `mIG` and `mMIG` representation are shown in Figure 4.11, Figure 4.12, Figure 4.13 respectively.

We consider the logic cone LC_4 for inversion operation count analysis in both `MIG` and `mMIG` synthesis methods which is demonstrated in Figure 4.14

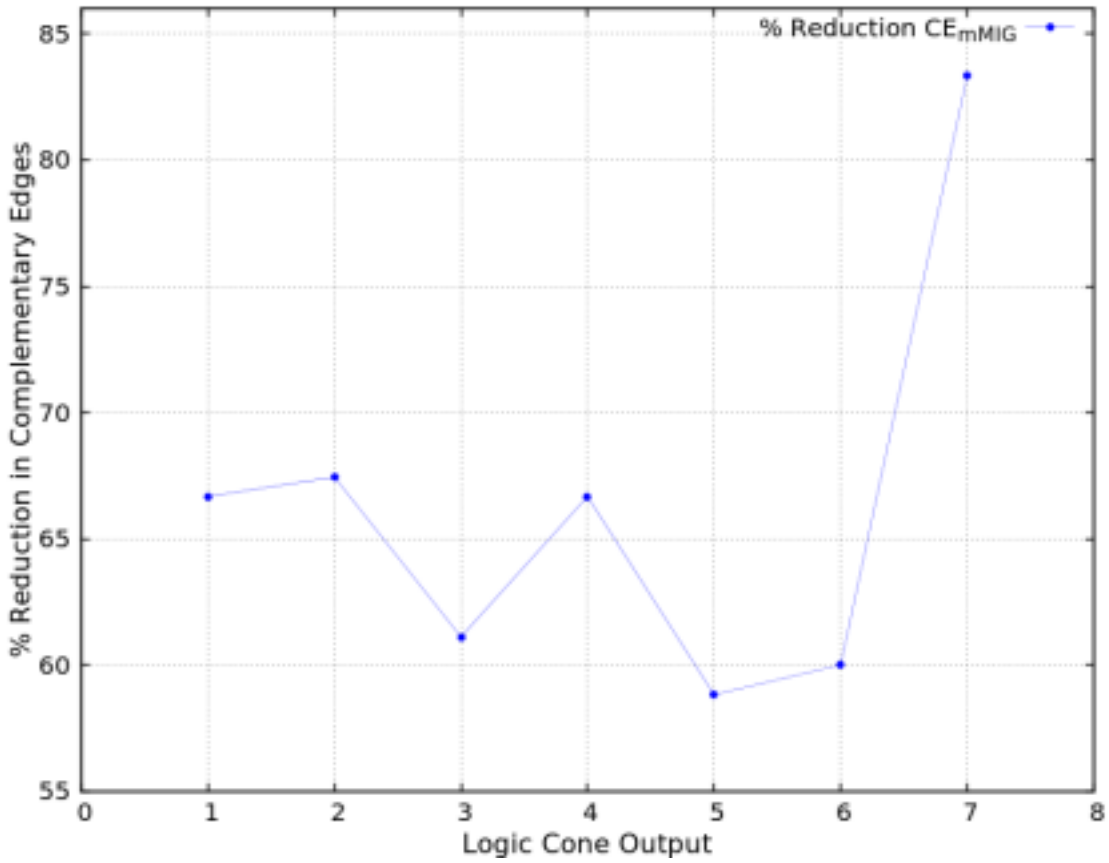

Figure 4.9: Percentage reduction in complementary edges in `mMIG` logic synthesis of EPFL benchmark circuits in Int to float converter in Random/control benchmarks.

LC_4 consists of 8865 *nand2*, 1855 *nor2*, 8 *nand4*, 819 *xor2*, and 1 *and8* operations, upon which `MIG` synthesis leads to an inversion operation count of 8865, 1855, 8, 1638 and zero, respectively. However, in `mMIG` synthesis, no inversion operations are required in any of the logic operations due to the proposed rules and derived theorems. Hence, it leads to a saving of $12,366$ inversion operations in this logic cone leading to compact implementation. Similar inversion optimizations are applicable to other logic cones of $c432$ benchmark circuit.

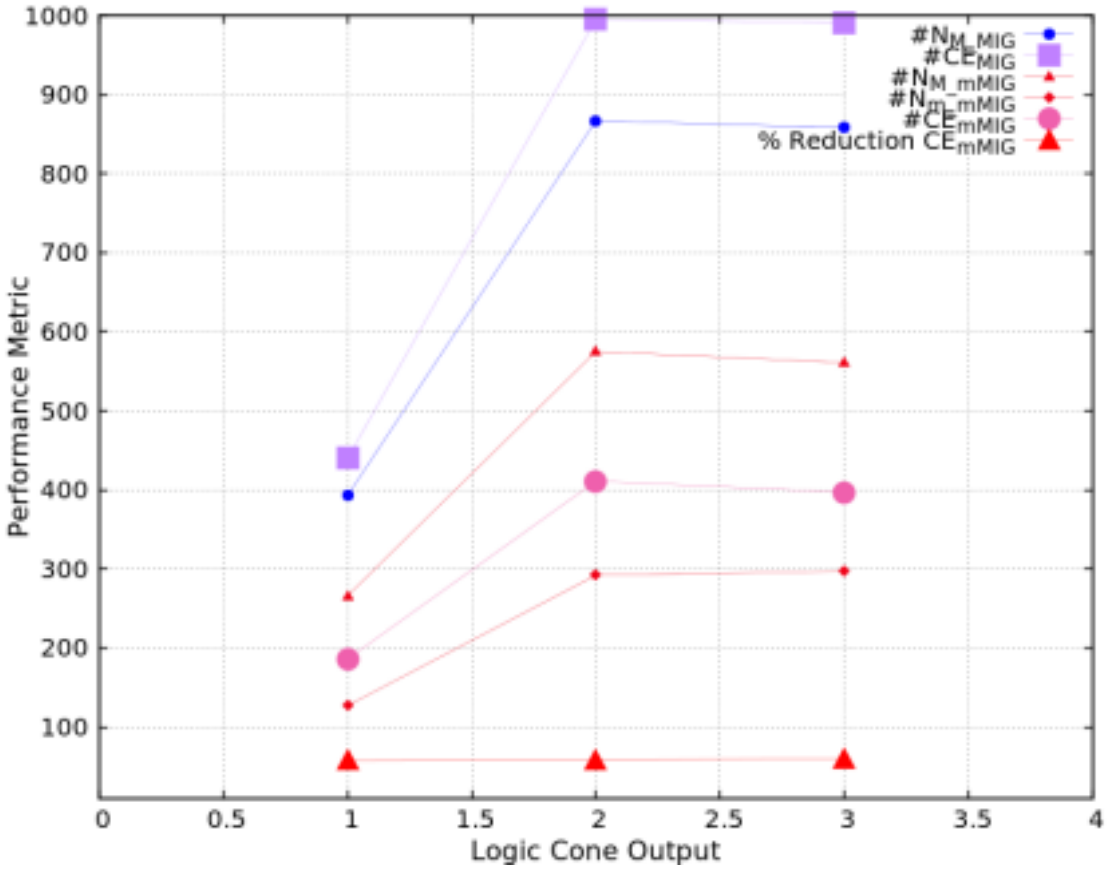

Figure 4.10: Resource count comparison of number of *majority* modes ($\#N_{MIG}$), *complemented* edges ($\#CE_{MIG}$) in `MIG` synthesized circuit and number of *majority* nodes ($\#N_{M\_mMIG}$), *minority* nodes ($\#N_{m\_mMIG}$), and `CE`s ($\#CE_{mMIG}$) in `mMIG` synthesized circuit, and percentage reduction in complemented edges in `mMIG` synthesis % Reduction CE_{mMIG} in lookahead XY router circuit in EPFL benchmark circuit.

4.4 Optimizing linear functions: $XORn$ implementations

In this section, we initially state optimized MIG and `mMIG` implementation of n-input XOR gates, referred to as $XORn$, starting with $n = 2$ followed by its detection and synthesis.

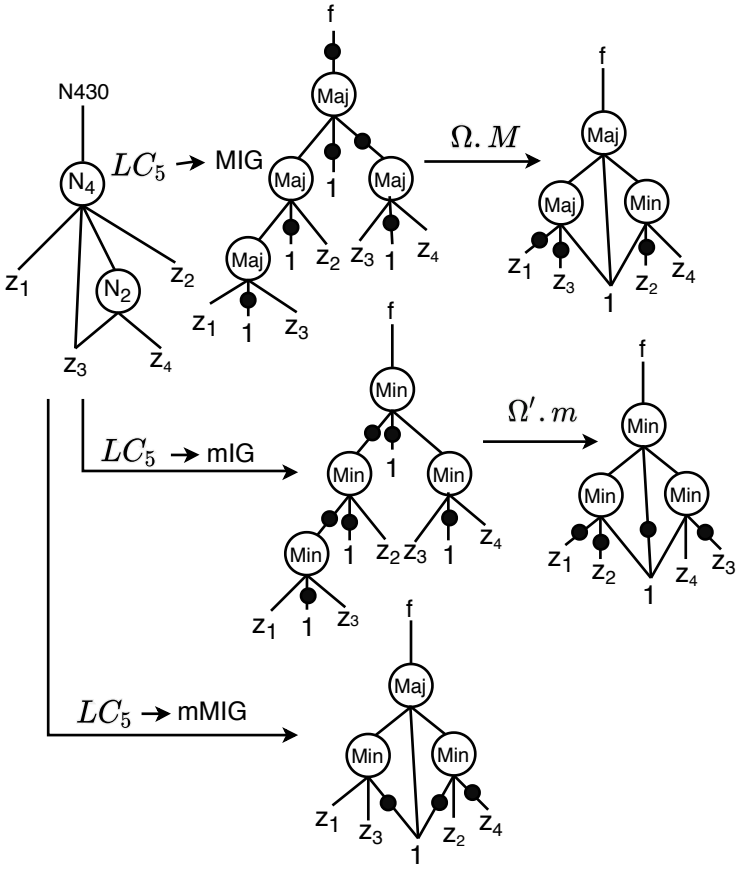

Figure 4.11: `mMIG` representation of the logic cone(LC_5) of c432(ISCAS'85 benchmark) circuit whose output ($N430$) have derived from the function f_5 = nand4(z_1, z_2, z_3, nand2(z_3, z_4)) depicts the lesser number of inverter count compared to `MIG` representation. $z_1 ... z_4$ are the intermediate value generated while deriving the output of the logic cone LC_5.

4.4.1 $XOR2$ implementation

The XOR operation involving two variables x and y can be expressed in MIG synthesis as $M(M(y, \bar{1}, \bar{x}), 1, M(x, \bar{1}, \bar{y}))$, which comprises 3 majority and 2 inversion operations shown in Figure 4.17(a). In `mMIG` synthesis, $XOR2$ can be implemented as $m(m(x, 1, y), 1, M(x, \bar{1}, y))$, which incurs 2 minority, 1 majority, and no inverter operation, leading to reduction of 2 inversion operations as shown in Figure 4.17(b).

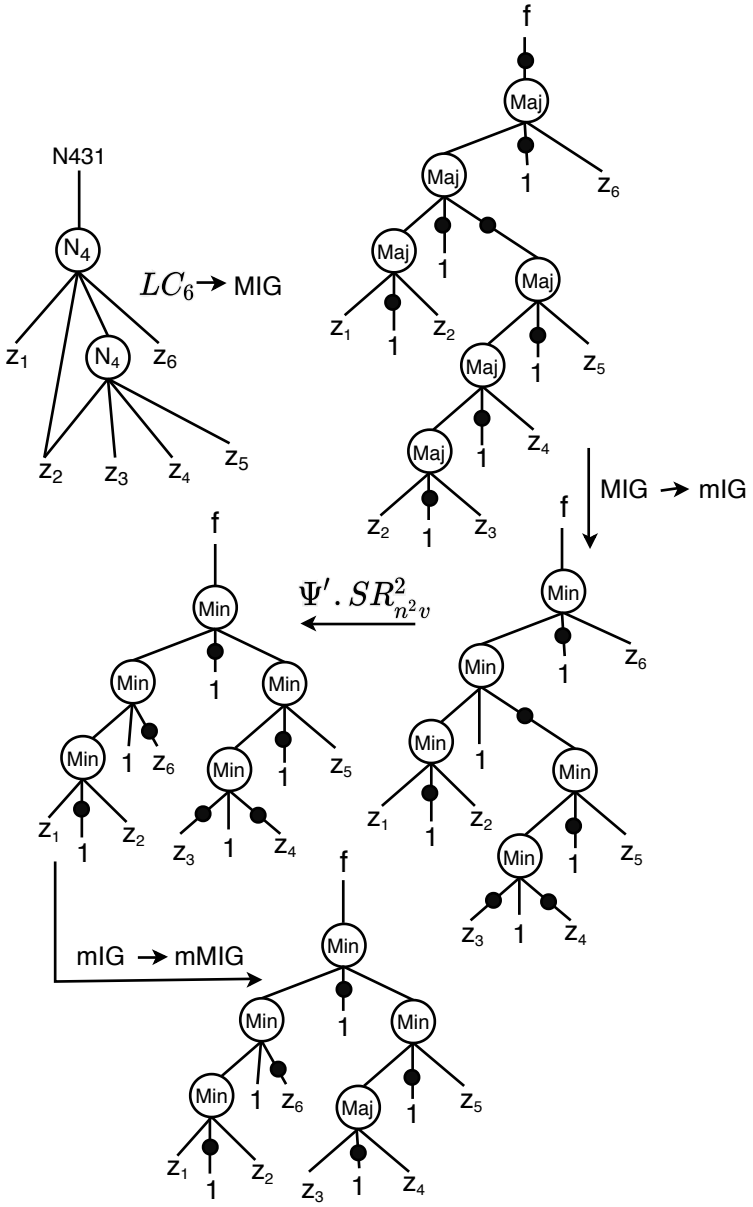

Figure 4.12: `mIG` representation of the logic cone(LC_6) of c432(ISCAS'85 benchmark) circuit whose output ($N431$) have derived from the function f_6 = nand4(z_1, z_2, z_5, nand4(z_2, z_3, z_4, z_5)) uses one variant of swapping reconvergence($\Psi'.SR^2_{n^2v}$) transformation rule decreases the level by 2 of LC_6 followed by `mMIG` representation reduces the total inverter count, result of which area is reduced compared to `MIG` representation where $z_1 ... z_5$ are the intermediate value generated while deriving the output of LC_6.

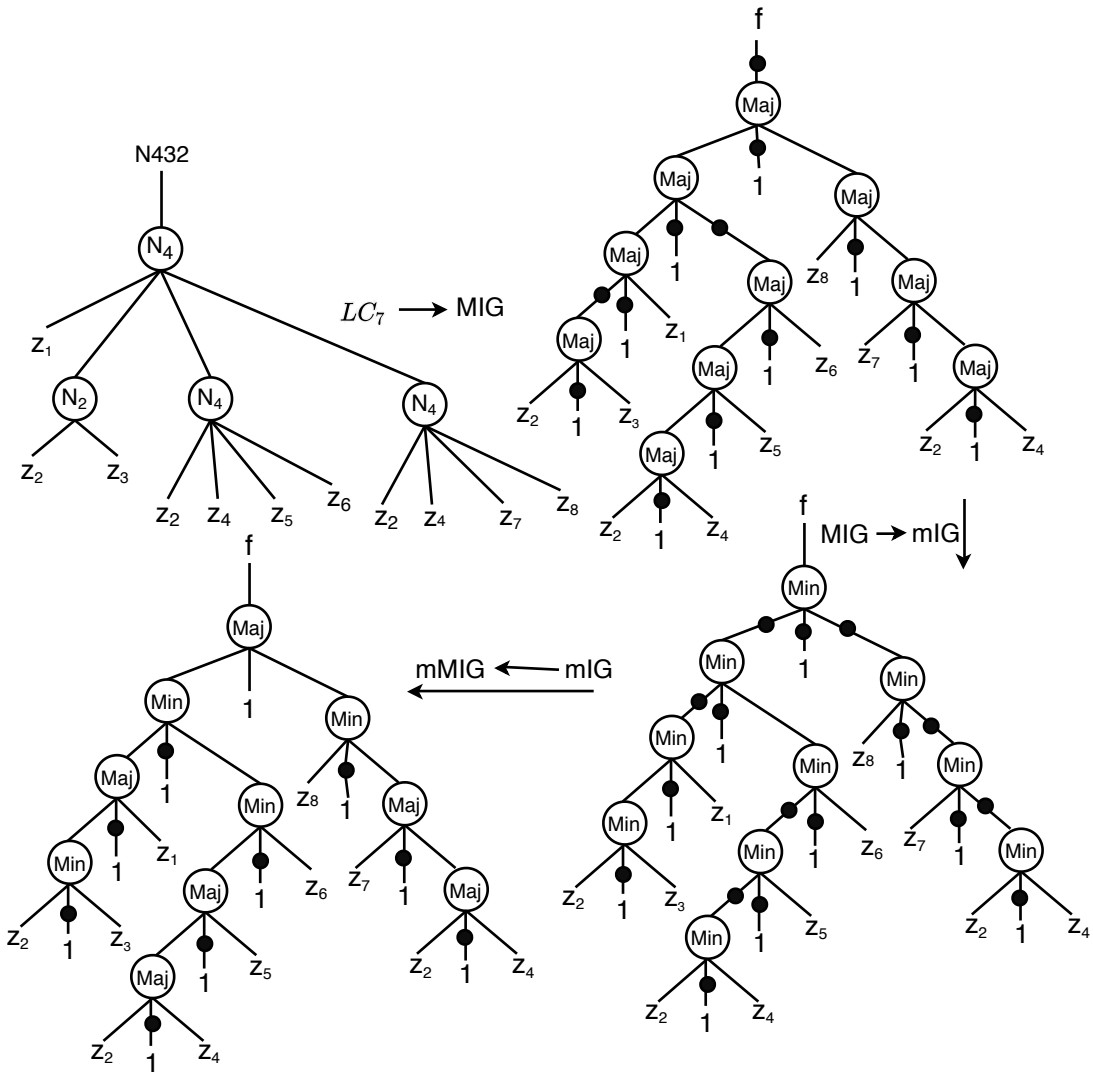

Figure 4.13: `mMIG` representation of the logic cone(LC_7) of c432(ISCAS'85 benchmark) circuit whose output ($N432$) have derived from the function $f_7 = \text{nand4}(z_1, \text{nand2}(z_2, z_3), \text{nand4}(z_2, z_4, z_5, z_6), \text{nand4}(z_2, z_4, z_7, z_8))$ shows the significant amount of inverter reduction than `MIG` implementation. $z_1 ... z_8$ are the intermediate value generated while deriving the output of LC_7.

4.4.1.1 $XOR3$ implementation

In MIG synthesis, $XOR3$ function, expressed as $x \oplus y \oplus z$ is implemented as $M(M(z, \bar{x}, y), x, M(\bar{y}, \bar{x}, \bar{z}))$ which comprises three majority and for inversion operations. Applying inversion propagation operation, reduces the overhead by two inverters. Neverthless, this is not the optimal case. Optimality is achieved using mMIG

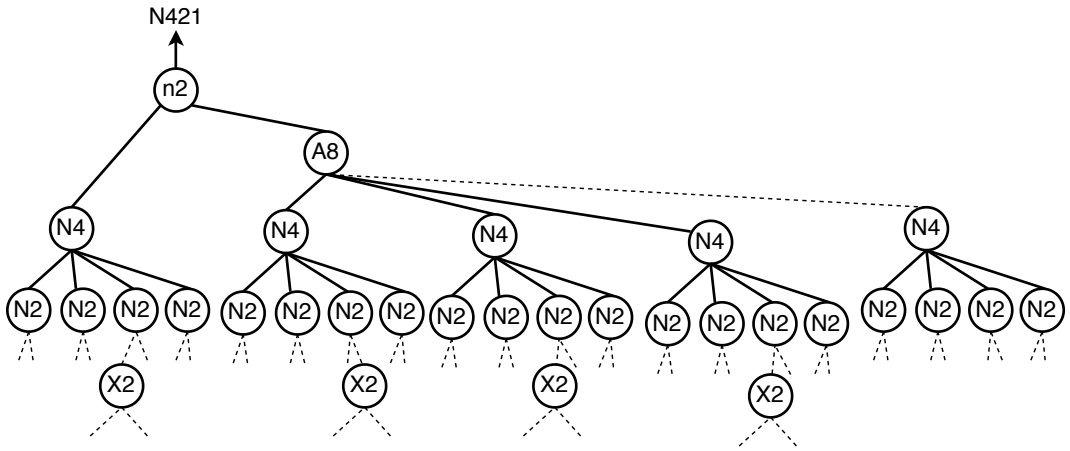

Figure 4.14: Topological representation of the logic cone LC_4 of c432(ISCAS'85 benchmark circuit) which comprises four, two and eight Boolean input logical operations such as $NAND4(N4)$,$NAND2(N2)$,$NOR2(n2)$,$XOR2(X2)$, and $AND8$ where $N4, N2, n2$ optimally represented using only minority function and $AND8$ represented using only majority function while $XOR2$ comprises majority as well as minority function, can be referred from Table 4.5 and dashed line shows the intermediate

representation of $XOR3$ as $M(M(z, \bar{x}, y), x, m(y, x, z))$ which costs two majority, one minority, and only one inversion operation as observed in Figure 4.18.

In MIG synthesis, $XOR3$ expressed as $x \oplus y \oplus z$ is implemented as $M(M(z, \bar{x}, y), x, M(\bar{y}, \bar{x}, \bar{z}))$ which comprises 3 majority and 4 inversion operations shown in Figure 4.19(a). However, in `mMIG` synthesis, $XOR3$ can be optimally expressed as $m(m(z, x, \bar{y}), x, m(y, x, \bar{z}))$, which incurs 3 minority and 2 inversion operations, leading to a reduction of 2 inversion operations shown in Figure 4.19(b).

4.4.2 Optimizing $XORn$ implementation

An n-input XOR, depending on parity of n, can be implemented with XOR3 and XOR2 as,

$$XORn = \begin{cases} (\lceil \frac{n}{2} \rceil - 1)XOR3 \text{ and } 1XOR2 & \text{if } n \text{ is even} \\ (\lceil \frac{n}{2} \rceil - 1)XOR3 & \text{if } n \text{ is odd} \end{cases}$$

Hence, from implementations of $XOR2$ and $XOR3$, we can compute the required

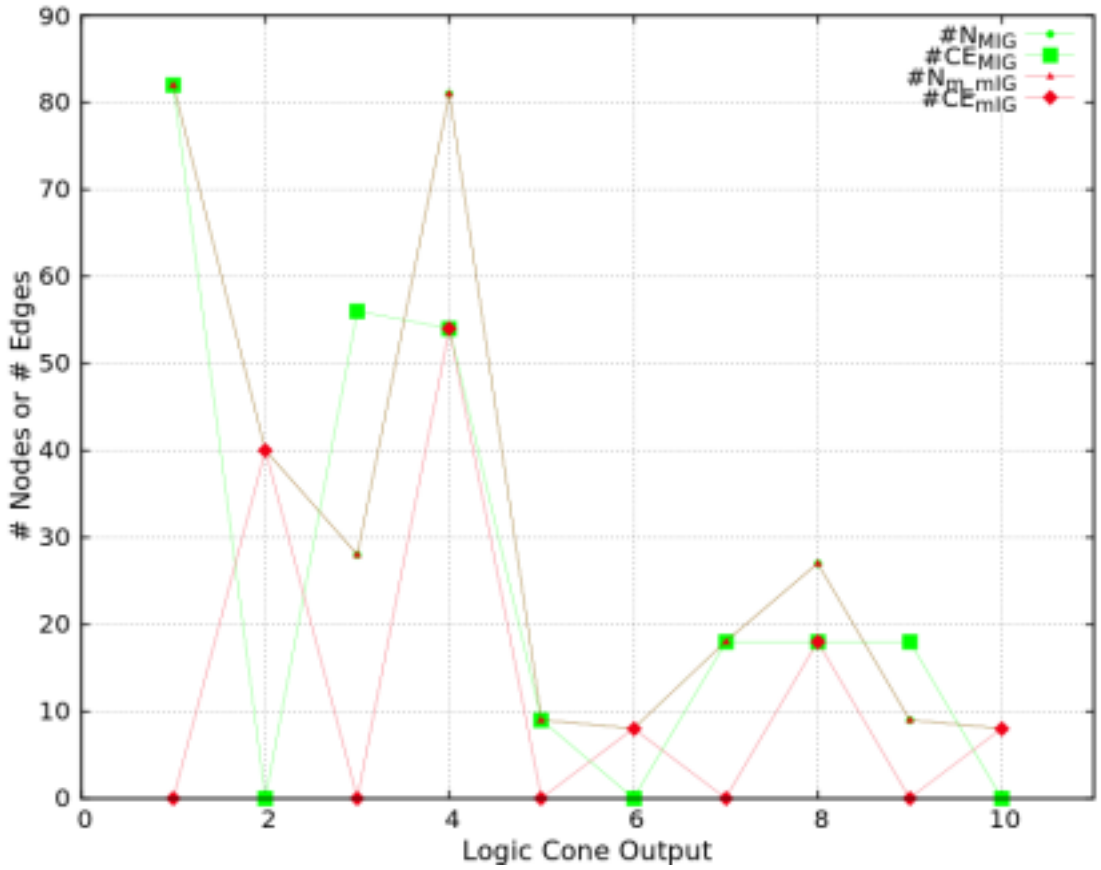

Figure 4.15: Resource count comparison of logic cones($LC_1 ... LC_3$) from the prespective of number of minority nodes ($\#N_m mIG$), complemented edges ($\#CE_{mIG}$) compared to`MIG`synthesized circuit in c432 of ISCAS'85 benchmark circuit.

resources for $XORn$ operation in MIG and `mMIG` synthesis techniques as shown in Table 4.5. For even (odd) n, `mMIG` method consumes approximately (exactly) half the number of inverter as compared to MIG method. However, we again reiterate that our assumption is that both minority and majority operation consumes same resources in area if they exist in standard cell library of the technology.

4.4.3 Detecting and Synthesizing XOR operations

Due to the complexity of exact synthesis method [24] and recent technologies whose physical implementation of inverter is more expensive than majority operation [20], it

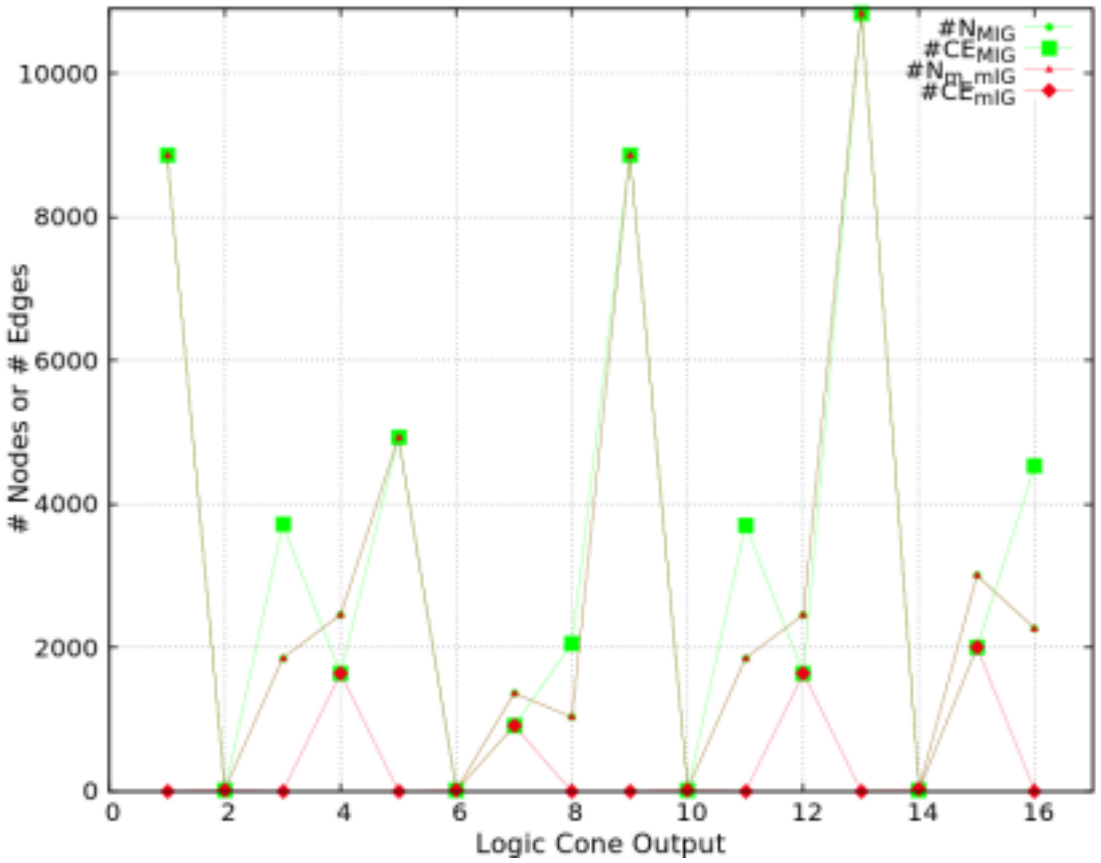

Figure 4.16: Resource count comparison of logic cones($LC_4 ... LC_7$) of mIG synthesized circuit in the term of number of minority nodes ($\#N_{m\_mIG}$), and complemented edges ($\#CE_{mIG}$) in c432 of ISCAS'85 benchmark circuit.

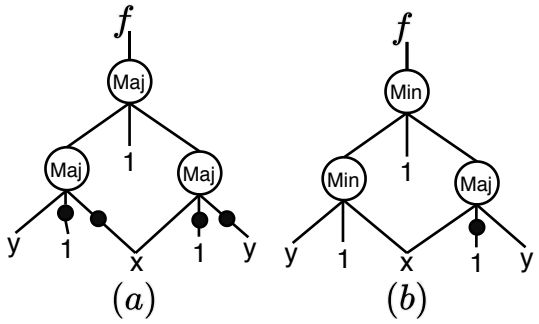

Figure 4.17: `mMIG` representation of $XOR2$ function $f = x \oplus y$ requires no CEs, result of which no additional overhead which is not true in the case of MIG representation.

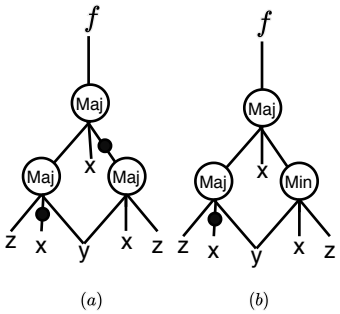

Figure 4.18: (a) Optimized MIG implementation of $f = x \oplus y \oplus z$ requires three majority nodes with two CEs, thereby incurs an additional overhead of an inverter propagation operation from [2] and two *net delay* value, whereas (b) mMIG implementation of f requires two majority nodes and one minority node with only one CE and one *net delay* value. This implies an improvement in CE count and *net delay* of single unit in XOR3 operation.

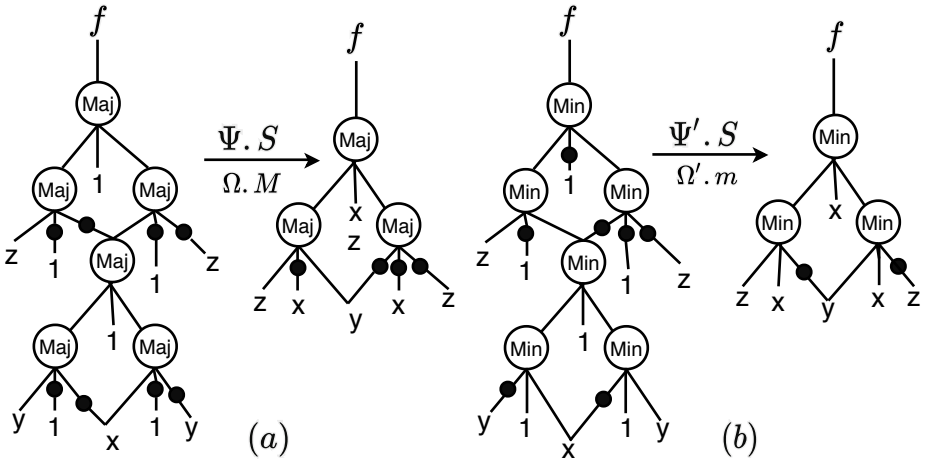

Figure 4.19: mIG representation of $XOR3$ function $f = x \oplus y \oplus z$ requires fewer inverter, which is quickly achieved upon applying operations $\Psi'.S$ followed by $\Omega'.m$ compared to MIG representation.

Table 4.5: Resource consumption for XORn implementation in MIG and `mMIG` synthesis methods. min, MAJ, and INV refer to minority, majority, and inverter operations, respectively.

Synthesis Method	MIG	mMIG
XORn (even n)	$3\lceil\frac{n}{2}\rceil$ MAJ, $(4\lceil\frac{n}{2}\rceil - 2)$ INV	$(3\lceil\frac{n}{2}\rceil - 1)$ min, 1 MAJ, $2(\lceil\frac{n}{2}\rceil - 1)$INV
XORn (odd n)	$3(\lceil\frac{n}{2}\rceil - 1)$ MAJ, $4(\lceil\frac{n}{2}\rceil - 1)$ INV	$3(\lceil\frac{n}{2}\rceil - 1)$ min, $2(\lceil\frac{n}{2}\rceil - 1)$ INV

is quite necessary to recognize all the XOR operations first and reshape it accordingly so that resultant structure consumes less number of inversion operations otherwise inverters will cause an additional overhead. Several rules are proposed in [21], which will be considered for the detection of XOR operation followed by its optimized `mMIG` representation as shown in the Table 4.5.

4.5 ARX-boxes: MARX-2 and SPECKEY

The class of ARX operations comprising modular addition, rotation, and XOR, form an important crypto-primitve that has fast performance with compact implementation in addition to resilience against timing based side channel attacks. We consider two important ARX boxes, MARX2 and SPECKEY [66]. MARX-2 is a 32-bit ARX box comprising two 8-bit modular additions, two 8-bit XOR2 operations, and left circular shift operations of $1, 2, 3, 7$ bits. SPECKEY comprises one 16-bit modular addition, 16 XOR2 additions, and left circular shift of 2 and 7 bits. The respective resource consumption of both these ARX-boxes when mapped to MIG synthesis and `mMIG` synthesis is depicted in Table 4.6.

Table 4.6: $\#N_{MIG}$: number of MIG nodes, $\#CE_{MIG}$: number of complemented edges in MIG, $\#N_{mIG}$: number of mIG nodes, $\#CE_{mMIG}$: number of complemented edges in `mMIG` in two Addition- Rotation-XOR operation (ARX) based S-boxes: MARX-2 and SPECKEY.

ARX Boxes	Operation	#Resources MIG		#Resources mMIG		
		$\#N_{MIG}$	$\#CE_{MIG}$	$\#N_{MIG}$	$\#N_{mIG}$	$\#CE_{mMIG}$
MARX-2	16 XOR2	48	32	16	32	0
	Two mod 2^8 addition	64	60	18	46	28
SPECKEY	16 XOR2	48	32	16	32	0
	One mod 2^{16} addition	64	62	17	47	30

4.6 Carry Save Adder: CSA

To reduce gate delays, a carry-save adder is introduced, which is heavily used for fast multiplication. While designing a circuit, it gives added advantage on the partial

sum(say S) and partial carry(say C) operation defined over input Boolean variable (say x, y, and z). Consider $S = x \oplus y \oplus z$ and $C = xy + yz + xz$, whose mIG implementation depicts the saving of the inverter count, as shown in Figure 4.20.

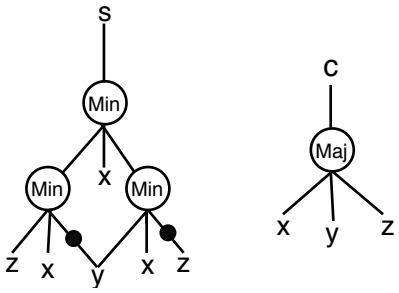

Figure 4.20: Representation of the Sum(S) as logical XOR operation on three input Boolean variables say x,y, and z in mIG as $S = m(m(z, x, \bar{y}), x, m(y, x, \bar{z}))$ which shows the savings in the inverter count compared to MIG representation, and Carry(C) as logical OR operation on all the set of two variable logical ANDed with another variable expressed in MIG representation, $C = M(x, y, z)$

4.7 SIMON Round function

We consider an optimized implementation of round function of SIMON2n, which is a lightweight Feistel block cipher with block size of $2n$. The SIMON round function [67] is an ARX construction with a balanced Feistel structure defined with the following equations: temp $\leftarrow x$, $x \leftarrow y \oplus (S(x)\&S^8(x)) \oplus k$, $y \leftarrow$ temp, where x, y are n-bit signals, and S^j is a left circular shift operation by j bits. A SIMON2n round function, for $n = 16$, incurs following resource consumption,

As the construction has an optimized implementation where a two-input AND operation implemented using one, three-input majority logic gate, a three-input XOR operation implemented using three, three-input minority logic gates, and a two-input XOR operation implemented using one majority logic gate and two, three-input minority logic gates. Therefore, for n=16, 16 XOR3 (optimally represented in MIG as 48 MAJ, 64 INV and in mMIG 48 min, 32 INV), 16 XOR2 (optimally represented in MIG as 48 MAJ, 32 INV and in mMIG 32 min, 16 INV), 16 AND2 (optimally represented in MIG as 16 MAJ and in mMIG 16 MAJ). Hence, 112 MAJ operations and 96 INV

operations are required in MIG synthesis technique, whereas in mMIG synthesis, 80 min, 32 MAJ, and 32 INV operations are required. This demonstrates that mMIG synthesis technique requires 64 INV operations lesser for a SIMON2n round. For a given n, the inverter count reduces by $4n$ in `mMIG` synthesis method.

The construction has an optimized implementation as shown in Figure 4.21 a two-input AND operation implemented using 1 three-input majority logic gate, a three-input XOR operation implemented 3 three-input minority logic gates, and a two-input XOR operation implemented using 1 majority logic gate and 2 three-input minority logic gates.

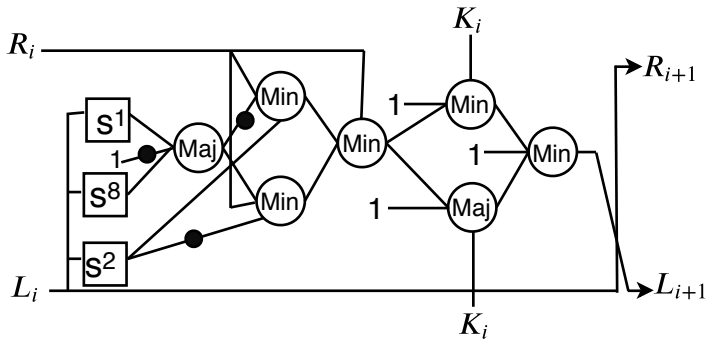

Figure 4.21: Implementation of SIMON round function using 2 majority gate,5 minority gate along with only 2 inverter(`mMIG`) where R_i, L_i denotes the right half and left half of the word respectively. S^i denotes the circular left shift by i number of bits,K_i denotes key value and the results are R_{i+1},L_{i+1} respectively in the i^{th} step of the process.

4.8 Present S-box

We consider the implementation of 4×4 substitution box (S-box), denoted as $S(x)$, in PRESENT lightweight block cipher. It comprises two cascaded 4×4 (four-bit input and four-bit output) vectorial Boolean functions, G and F, represented as $(S(x) = F(G(x)))$. Both G and F comprise four coordinate functions, namely, g_0, g_1, g_2, g_3, and f_0, f_1, f_2, f_3, respectively, as $G(x, y, z, w) = (g_3, g_2, g_1, g_0)$, and $F(x, y, z, w) = (f_3, f_2, f_1, f_0)$. The coordinate functions are defined as

$$g_0 = 1 \oplus w \oplus xy \oplus xz \oplus yz, \tag{4.1}$$

$$g_1 = 1 \oplus x \oplus z \oplus yw \oplus zw, \tag{4.2}$$

$$g_2 = 1 \oplus y \oplus z \tag{4.3}$$

$$g_3 = y \oplus z \oplus w \tag{4.4}$$

$$f_0 = z \oplus yw, \tag{4.5}$$

$$f_1 = y \oplus z \oplus xw, \tag{4.6}$$

$$f_2 = x \oplus zw \tag{4.7}$$

$$f_3 = y \oplus z \oplus w \oplus xw, \tag{4.8}$$

The above functions comprise two,three,four, and five-input logical XOR operations and two-input logical AND operations whose efficient implementation in `mMIG` includes representation of n input XOR using XOR2 and XOR3 comprising majority and minority operator depicted in table 4.5 whereas, AND operation uses only majority operator. `mMIG` representation of g_0, g_1 are demonstrated in the Figure 4.22,Figure 4.23 respectively, whereas MIG and mIG representation of $g_2, g_3, f_0, f_1, f_2, f_3$ are shown in Figure 4.24, Figure 4.25. Consider the intermediate output h_i, $0 \leq i \leq 10$, are intermediate outputs of G and F functions which are used in g_i, $0 \leq i \leq 3$, and

f_i, $0 \leq i \leq 3$ functions are represented as follows,

$$h_0 = M(x, 0, z) \tag{4.9}$$

$$h_1 = M(x, 0, y) \tag{4.10}$$

$$h_2 = m(m(h_0, \bar{w}, \bar{h_1}), \bar{w}, m(h_1, \bar{w}, \bar{h_0})) \tag{4.11}$$

$$h_3 = M(y, 0, z) \tag{4.12}$$

$$h_4 = M(y, 0, w) \tag{4.13}$$

$$h_5 = m(m(\bar{x}, z, \bar{h_4}), z, m(h_4, z, x)) \tag{4.14}$$

$$h_6 = M(w, 0, z) \tag{4.15}$$

$$h_7 = M(x, 0, w) \tag{4.16}$$

$$h_8 = M(\bar{x}, 0, w) \tag{4.17}$$

$$h_9 = M(M(h_0, w, h_1), \bar{w}, M(\bar{h_1}, w, \bar{h_0})) \tag{4.18}$$

$$h_{10} = M(M(z, x, h_4), \bar{x}, M(\bar{h_4}, x, \bar{z})) \tag{4.19}$$

Now, the `mMIG` synthesis of all the coordinate functions using the above intermediate function(h_i) are shown below,

$$g_0 = M(M(h_3, 0, \bar{h_9}), 1, M(h_9, 0, \bar{h_3})) \tag{4.20}$$

$$g_1 = M(M(h_6, 0, \bar{h_1}0), 1, M(h_1 0, 0, \bar{h_6})) \tag{4.21}$$

$$g_2 = M(M(z, 0, y), 1, M(\bar{y}, 0, \bar{z})) \tag{4.22}$$

$$g_3 = M(M(z, \bar{w}, y), w, M(\bar{y}, \bar{w}, \bar{z})) \tag{4.23}$$

$$f_0 = M(M(z, 0, \bar{h_4}), 1, M(h_4, 0, \bar{z})) \tag{4.24}$$

$$f_1 = M(M(z, \bar{y}, h_7), y, M(\bar{h_7}, \bar{y}, \bar{z})) \tag{4.25}$$

$$f_2 = M(M(x, 0, \bar{h_6}), 1, M(h_6, 0, \bar{x})) \tag{4.26}$$

$$f_3 = M(M(z, \bar{y}, M(\bar{x}, 0, w)), y, M(\bar{M}(\bar{x}, 0, w), \bar{y}, \bar{z})) \tag{4.27}$$

Table 4.7 demonstrates the count of the individual operations in the decomposition

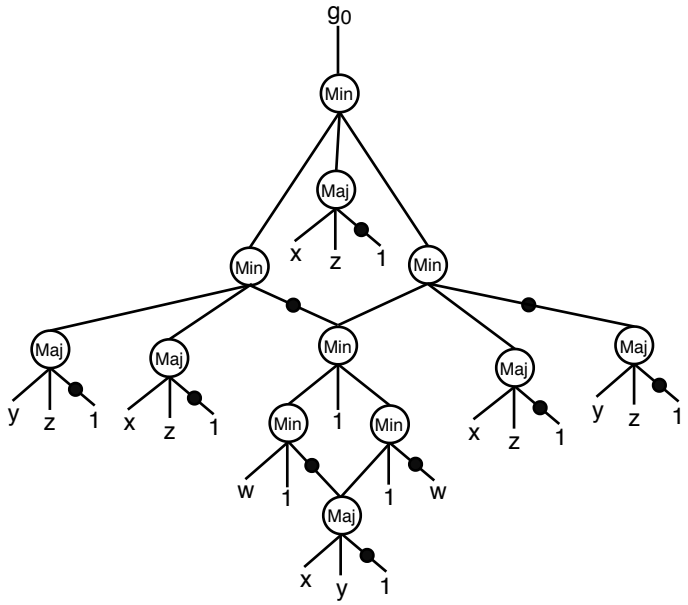

Figure 4.22: `mMIG` implementation of the coordinate function g_0 where inverters are saved due to optimized $XORn$ implementation of the circuit where w, x, y and z are the input Boolean variables.

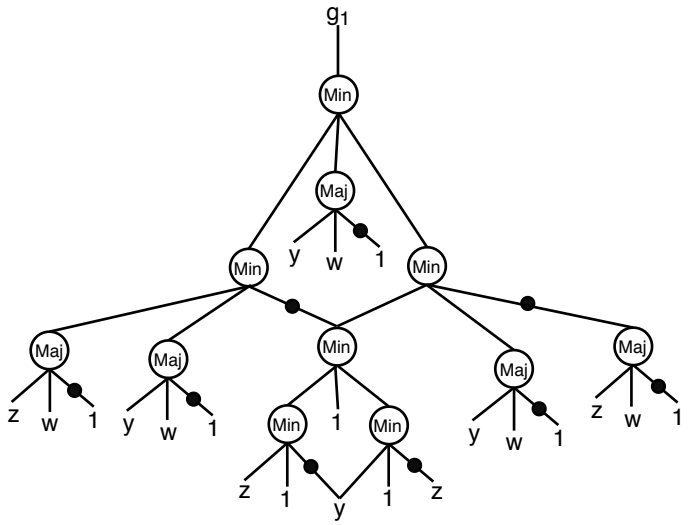

Figure 4.23: `mMIG` implementation of the coordinate function g_1 leads to reduce the inverter count due to `mMIG` implementation of $XORn$ operation where w, x, y and z are the input Boolean variables.

functions $g_0, \ldots, g_3$ and $f_0, \ldots, f_3$, which would be required in MIG synthesis and `mMIG` synthesis methods. As results show, the inverter count in `mMIG` synthesis is 15, which

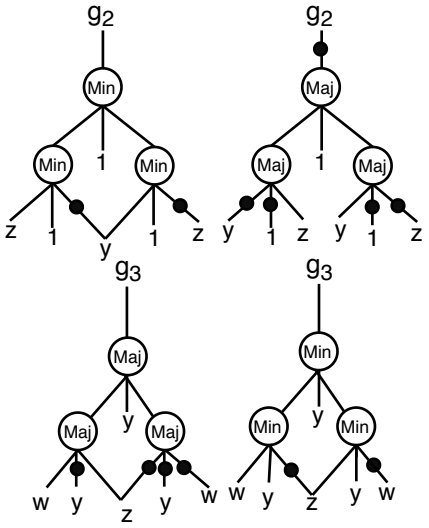

Figure 4.24: MIG and mIG representation of the coordinate function g_2 and g_3, depicts the total savings of the inverter count where w, y and z are input Boolean variables.

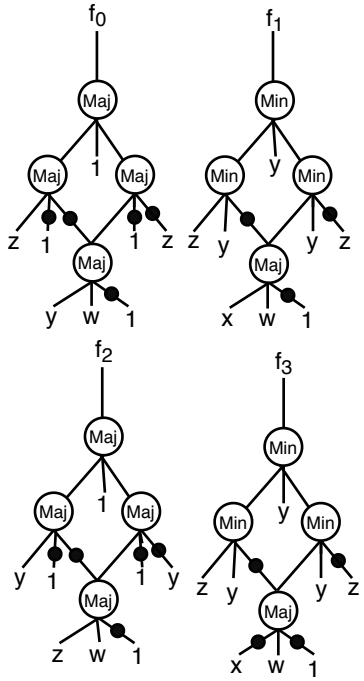

Figure 4.25: MIG and `mMIG` implementation of the coordinate function f_0, f_1, f_2 and f_3 shows the reduction in complemented edges(CEs) where w, y and z are input Boolean variables.

Table 4.7: Resource count of MIG and mMIG of distributed functions in Present S-box.N,CE,MIG,`mMIG` refer to nodes,complemented edges,MIG nodes, and `mMIG` nodes, respectively.

	Logic gates			MIG		`mMIG`		
Function	#XOR3	#XOR2	#AND2	$\#N_{MIG}$	$\#CE_{MIG}$	$\#N_{mIG}$	$\#N_{MIG}$	$\#CE_{mMIG}$
g_0	1	1	3	11	5	3	3	5
g_1	0	1	2	6	3	2	4	3
g_2	0	1	0	3	4	1	2	0
g_3	1	1	0	3	4	3	0	2
f_0	0	1	1	4	2	2	2	0
f_1	1	1	1	4	4	3	1	2
f_2	0	1	1	4	2	2	2	0
f_3	1	1	1	4	5	3	1	3
Total	4	8	9	39	29	19	15	15

is approximately half of the inverter count in MIG synthesis technique. The total number of majority and minority operations are also lesser in `mMIG` than MIG where $\#N_{MIG}$, $\#N_{mIG}$, $\#CE_{MIG}$, $\#CE_{\texttt{mMIG}}$ denotes number of MIG nodes, number of mIG nodes, number of complemented edges of the MIG and number of complemented edges of the `mMIG`, respectively.

Chapter 5

Conclusions and Future Work

In this thesis, we propose an mMIG synthesis technique, which comprises *minority* nodes in addition to pre-existing *majority* and *inversion* operations (referred to as complemented edges). Despite the soundness and completeness property of MIG-based circuit synthesis, the proposed synthesis technique aims to reduce the count of inversion operations as compared to MIG synthesis approach leading to reduced area and delay metrics of the target circuit when implemented for beyond-CMOS technologies. We demonstrate a set of Boolean algebra rules for *minority* operations, and further propose a set of transformation rules for logic optimization in mMIG synthesis.

The proposed rules when applied in a specific sequence as demonstrated in the proposed algorithms of mMIG based implementations demonstrate reduction in inverter count or CEs over a range from 57.7% to 93.3% in EPFL combinational benchmark suite, and by almost 50% in S-box of PRESENT lightweight cipher and ARX boxes of MARX-2 and SPECKEY. In addition, we demonstrate circuits that have a large run of XOR operations such as $XORn$ for a large n can be heavily optimized in terms of inversion count. We state that exploring the sequence of transformation rules can have an important implication on reduction of *inversion* operations, which remains an important future research direction of this approach.

5.1 Summmary of Research Achievements

To summarize the contributions of this thesis are as follows:

(i) **Set of rules based on `mIG`:** We proposed a set of algebraic transformation rules for Boolean algebra in *minority inverter graph*(`mIG`) in Chapter 3 of Section 3.1 mainly consists the variants of *Associativity* rule, *Swapping reconvergence* rule, *Relevance* rule, and the *substitution* rule.

(ii) **Set of rules based on `mMIG`:** We proposed a set of algebraic transformation rules for novel logic representational structure *minority-majority-inverter graph* (`mMIG`) in Chapter 3 of Section 3.4 consists the large collection of the optimization class(categorized in three class) whose main focus is to reduce the `CEs` along with the little focus on the complete size and depth optimization, *Elimination* rule, *Absorption* rule, *Swapping reconvergence* rule along with its variant, and *Relevance* rule along with its variant can be find in Section 3.5.

(iii) **Optimization algorithm for `mMIG`:** We proposed novel ***`mMIG`*** *logic optimization algorithm* in Chapter 3 of Section 3.6 to achieve size and depth optimization of the circuit mainly comprises *reshaping algorithm*, *inverter propagation algorithm* uses *Greedy approach* and ***`mMIG`*** *size optimization algorithm* uses *Divide and conquer approach.*

(iv) **Case study for `mIG` and Comparison with `MIG`:** We present case studies in Chapter 4 of Section 4.3, 4.4 shows c432(ISCAS'85 standard benchmark circuit), linear function(XOR3) respectively along with multiple circuit instances.

(v) **Performance metrics for `mMIG` and Comparison with `MIG`: We present performance metrics that can be used for comparative result analysis which includes *Net delay*, *Logic delay*, *LUT count*, *On-chip dynamic signal power*, in addition to count of CEs.**

(vi) **Case study for `mMIG` and Comparison with `MIG`:** We present several case studies in Chapter 4 of Section 4.2, 4.3, 4.1 shows standard benchmark circuits

EPFL, ISCAS'85, IWLS'05, Arithmetic HDL respectively, lightweight cryptographic primitives and combinational circuits can be find in Section 4.5, 4.8, 4.7, 4.6, 4.4 and along with this several circuit instances are used for the comparison.

(vii) **Optimization flow in `mMIG`:** We propose the logic synthesis optimization flow comprises three phases namely, *Functional and Graph Decomposition* comes under pre-processing phase where input circuit in MIG synthesis is partitioned into logic cones which is determined by the output of the circuit followed by further splitting into smaller partitions which comprises only connected majority nodes, only connected minority nodes, and group comprising interconnected *majority* and *minority* nodes. *Logic Minimization* deals with the replacement of *majority* node and immediate inversion with *minority* gate on each sub-partition and finally, *Post-Processing* phase applies a set of algebraic transformation rules, proposed derived theorems on the `mMIG` logic obtained from the logic minimization phase to achieve both reduction in `CEs` and size.

5.2 Future research directions

Despite significant progress in the topic of inverter reduction, future scope of work can be explored in several interesting research directions:

(i) Transformation rules that will assist in fault masking of `mIG`, `MIG`, `mMIG` or contributing to side-channel resilience of synthesized circuits.

(ii) Ease of detection of Hardware Trojans in `mMIG`-based circuit synthesis.

(iii) Exploring the sequence of transformation rules will have huge impact on reduction of inversion, area reduction and reduction in critical path of combinational circuits.

Bibliography

[1] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1994.

[2] Luca Gaetano Amarù, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. Majority-Inverter Graph: A New Paradigm for Logic Optimization. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 35(5):806–819, 2016.

[3] Gordon E Moore. Cramming More Components onto Integrated Circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.

[4] Mark Bohr. 14 nm Process Technology: Opening New Horizons.

[5] R. L. Rudell and A. Sangiovanni-Vincentelli. Multiple-Valued Minimization for PLA Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(5):727–750, 1987.

[6] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli. Multilevel Logic Synthesis. *Proceedings of the IEEE*, 78(2):264–300, 1990.

[7] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. Majority-inverter graph: A Novel Data-Structure and Algorithms for Efficient Logic Optimization. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2014.

[8] K. Bernstein, R. K. Cavin, W. Porod, A. Seabaugh, and J. Welser. Device and architecture outlook for beyond cmos switches. *Proceedings of the IEEE*, 98(12):2169–2184, 2010.

[9] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1st edition, 1994.

[10] Thomas Schneider, Alexander A Serga, Britta Leven, Burkard Hillebrands, Robert L Stamps, and Mikhail P Kostylev. Realization of Spin-Wave Logic Gates. *Applied Physics Letters*, 92(2):022505, 2008.

[11] Kun Kong, Yun Shang, and Ruqian Lu. An Optimized Majority Logic Synthesis Methodology for Quantum-Dot Cellular Automata. *IEEE Transactions on Nanotechnology*, 9(2):170–183, 2009.

[12] The Programmable Logic-in-Memory (PLiM) Computer, author=Gaillardon, Pierre-Emmanuel and Amarú, Luca and Siemon, Anne and Linn, Eike and Waser, Rainer and Chattopadhyay, Anupam and De Micheli, Giovanni, booktitle=2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), pages=427–432, year=2016, organization=Ieee.

[13] Mathias Soeken, Martin Roetteler, Nathan Wiebe, and Giovanni De Micheli. Design Automation and Design Space Exploration for Quantum Computers. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 470–475. IEEE, 2017.

[14] Luca Amarú, Pierre-Emmanuel Gaillardon, Subhasish Mitra, and Giovanni De Micheli. New Logic Synthesis as Nanotechnology Enabler. *Proceedings of the IEEE*, 103(11):2168–2195, 2015.

[15] Saeideh Shirinzadeh, Mathias Soeken, Pierre-Emmanuel Gaillardon, and Rolf Drechsler. Fast Logic Synthesis for RRAM-based In-Memory Computing using Majority-Inverter Graphs. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 948–953. IEEE, 2016.

[16] Saeideh Shirinzadeh, Mathias Soeken, Pierre-Emmanuel Gaillardon, and Rolf Drechsler. Logic Synthesis for Majority based In-Memory Computing. In *Ad-*

vances in Memristors, Memristive Devices and Systems, pages 425–448. Springer, 2017.

[17] Maj-n logic Synthesis for Emerging Technology, author=Neutzling, Augusto and Marranghello, Felipe S and Matos, Jody Maick and Reis, Andre and Ribas, Renato P, journal=IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, year=2019, publisher=IEEE.

[18] Walter Lau Neto, Xifan Tang, Max Austin, Luca Amaru, and Pierre-Emmanuel Gaillardon. Improving Logic Optimization in Sequential Circuits using Majority-inverter Graphs. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 224–229. IEEE, 2019.

[19] S MUROGA. Threshold logic and its application. *Wily-Interscience*, 1971.

[20] Eleonora Testa, Mathias Soeken, Odysseas Zografos, Luca Amaru, Praveen Raghavan, Rudy Lauwereins, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. Inversion Optimization in Majority-Inverter Graphs. In *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 15–20. IEEE, 2016.

[21] Zhufei Chu, Mathias Soeken, Yinshui Xia, Lunyao Wang, and Giovanni De Micheli. Structural Rewriting in XOR-Majority Graphs. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pages 663–668, 2019.

[22] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification, accessed February, 2020.

[23] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. Majority-based Synthesis for Nanotechnologies. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 499–502. IEEE, 2016.

[24] M. Soeken, L. G. Amarù, P. Gaillardon, and G. De Micheli. Exact Synthesis of Majority-Inverter Graphs and Its Applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(11):1842–1855, Nov 2017.

[25] Winston Haaswijk, Mathias Soeken, Alan Mishchenko, and Giovanni De Micheli. SAT based Exact Synthesis Using DAG Topology Families. In *2018 55Th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[26] E. J. McCluskey. Minimization of boolean functions. *The Bell System Technical Journal*, 35(6):1417–1444, 1956.

[27] Richard Rudell and Alberto Sangiovanni-Vincentelli. *Logic synthesis for VLSI design*. PhD thesis, University of California, Berkeley, 1989.

[28] Ning Song and Marek A Perkowski. Minimization of exclusive sum-of-products expressions for multiple-valued input, incompletely specified functions. *IEEE transactions on computer-aided design of integrated circuits and systems*, 15(4):385–395, 1996.

[29] Arist Kojevnikov, Alexander S Kulikov, and Grigory Yaroslavtsev. Finding efficient circuits using sat-solvers. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 32–44. Springer, 2009.

[30] Mathias Soeken, Luca Gaetano Amaru, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. Exact synthesis of majority-inverter graphs and its applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(11):1842–1855, 2017.

[31] Rolf Drechsler and Wolfgang Günther. Exact circuit synthesis. In *In Int'l Workshop on Logic Synth*, 1998.

[32] L. Amarú, M. Soeken, P. Vuillod, J. Luo, A. Mishchenko, P. Gaillardon, J. Olson, R. Brayton, and G. De Micheli. Enabling exact delay synthesis. In

2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pages 352–359, 2017.

[33] Mathias Soeken, Giovanni De Micheli, and Alan Mishchenko. Busy man's synthesis: Combinational delay optimization with sat. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 830–835. IEEE, 2017.

[34] B. Bollig and I. Wegener. Improving the variable ordering of obdds is np-complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996.

[35] Henrik Reif Andersen. An introduction to binary decision diagrams.

[36] Tsutomu Sasao. *Switching theory for logic synthesis*. Springer Science & Business Media, 2012.

[37] Richard L Rudell and Alberto Sangiovanni-Vincentelli. Multiple-valued Minimization for PLA Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(5):727–750, 1987.

[38] Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[39] A. Mishchenko, S. Chatterjee, and R. Brayton. Dag-aware aig rewriting: a fresh look at combinational logic synthesis. In *2006 43rd ACM/IEEE Design Automation Conference*, pages 532–535, 2006.

[40] Robert K Brayton, Gary D Hachtel, Curt McMullen, and Alberto Sangiovanni-Vincentelli. *Logic minimization algorithms for VLSI synthesis*, volume 2. Springer Science & Business Media, 1984.

[41] R. Drechsler, N. Drechsler, and W. Gunther. Fast exact minimization of bdd's. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(3):384–389, 2000.

[42] Anup Hosangadi, Farzan Fallah, and Ryan Kastner. Factoring and eliminating common subexpressions in polynomial expressions. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, pages 169–174. IEEE, 2004.

[43] Robert K Brayton, Richard Rudell, Alberto Sangiovanni-Vincentelli, and Albert R Wang. Mis: A multiple-level logic optimization system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(6):1062–1081, 1987.

[44] Saburo Muroga, Yahiko Kambayashi, Hung Chi Lai, and Jay Niel Culliney. The transduction method-design of logic networks based on permissible functions. *IEEE Transactions on Computers*, (10):1404–1424, 1989.

[45] Saburo Muroga. Logic synthesizers, the transduction method and its extension, sylon. In *Logic Synthesis and Optimization*, pages 59–86. Springer, 1993.

[46] Alan Mishchenko, Jin S Zhang, Subarnarekha Sinha, Jerry R Burch, Robert Brayton, and Malgorzata Chrzanowska-Jeske. Using simulation and satisfiability to compute flexibilities in boolean networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(5):743–755, 2006.

[47] Robert K Brayton, Gary D Hachtel, and Alberto L Sangiovanni-Vincentelli. Multilevel Logic Synthesis. *Proceedings of the IEEE*, 78(2):264–300, 1990.

[48] Alan Mishchenko, Robert Brayton, and Satrajit Chatterjee. Boolean factoring and decomposition of logic networks. In *2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 38–44. IEEE, 2008.

[49] Shih-Chieh Chang, Malgorzata Marek-Sadowska, and Kwang-Ting Cheng. Perturb and simplify: multilevel boolean network optimizer. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(12):1494–1504, 1996.

[50] Shih-Chieh Chang, Lukas PPP Van Ginneken, and Malgorzata Marek-Sadowska. Circuit optimization by rewiring. *IEEE Transactions on computers*, 48(9):962–970, 1999.

[51] J Paul Roth and Richard M Karp. Minimization over boolean graphs. *IBM journal of Research and Development*, 6(2):227–238, 1962.

[52] Congguang Yang and Maciej Ciesielski. Bds: A bdd-based logic optimization system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(7):866–876, 2002.

[53] Navin Vemuri, Priyank Kalla, and Russell Tessier. Bdd-based logic synthesis for lut-based fpgas. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 7(4):501–525, 2002.

[54] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. Bds-maj: A bdd-based logic synthesis tool exploiting majority logic decomposition. In *Proceedings of the 50th Annual Design Automation Conference*, pages 1–6, 2013.

[55] Robert K. Brayton and Curtis T. McMullen. The decomposition and factorization of boolean expressions. In *1982 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1982.

[56] N. Song and M. A. Perkowski. Exorcism-mv-2: minimization of exclusive sum of products expressions for multiple-valued input incompletely specified functions. In *[1993] Proceedings of the Twenty-Third International Symposium on Multiple-Valued Logic*, pages 132–137, 1993.

[57] Bjarni Jónsson and Alfred Tarski. Boolean algebras with operators. part i. *American journal of mathematics*, 73(4):891–939, 1951.

[58] Matthias Krause and Pavel Pudlák. On the computational power of depth-2 circuits with threshold and modulo gates. *Theoretical Computer Science*, 174(1-2):137–156, 1997.

[59] Rui Zhang, P. Gupta, and N. K. Jha. Synthesis of majority and minority networks and its applications to qca, tpl and set based nanotechnologies. In *18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design*, pages 229–234, 2005.

[60] Rui Zhang, Pallav Gupta, and Niraj K. Jha. Majority and minority network synthesis with application to qca-, set-, and tpl-based nanotechnologies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(7):1233–1245, July 2007.

[61] The mMIG Team. mMIG: , accessed October, 2020.

[62] Masanobu Katagi, Shiho Moriai, et al. Lightweight Cryptography for the Internet of Things. *Sony Corporation*, 2008:7–10, 2008.

[63] NIST USA. Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process, accessed February, 2020.

[64] Christoph Albrecht. Iwls 2005 benchmarks. In *International Workshop for Logic Synthesis (IWLS): http://www. iwls. org*, 2005.

[65] EPFL. *The EPFL Combinational Benchmark Suite*, 2020 (accessed May 31, 2020).

[66] Daniel Dinu, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov. SPARX: A Family of ARX-based Lightweight Block Ciphers Provably Secure Against Linear and Differential Attacks. *proceedings of ASIACRYPT'16*, pages 1–21, 2016.

[67] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. *IACR Cryptology ePrint Archive*, 2013(1):404–449, 2013.