Thesis n° 10 874

Logic Synthesis and Optimization for CMOS and Post-CMOS Technologies

Presented on 7th March 2025

School of Computer and Communication Sciences
Integrated Systems Laboratory (IC/STI)
Doctoral program in Computer and Communication Sciences

for the award of the degree of Docteur ès Sciences (PhD)

by

Dewmini Sudara MARAKKALAGE

École
polytechnique
fédérale
de Lausanne

2025

To all who have supported me. . .

Acknowledgements

Pursuing a Ph.D. has been a journey filled with challenges, growth, and invaluable support from many individuals. This is a humble attempt to express my deepest gratitude to those who have guided, supported, and inspired me throughout this endeavor.

First and foremost, I am immensely grateful to my advisor, Prof. Giovanni De Micheli, for his guidance, encouragement, and wisdom. His unwavering support, insightful discussions, and patience have played a pivotal role in shaping my research journey. I am incredibly fortunate to have had his guidance throughout my academic endeavors.

I extend my sincere appreciation to my jury committee members, Prof. Paolo Ienne, Prof. Thomas Bourgeat, Prof. Lana Josipović, and Dr. Patrick Vuillod, for their time, thoughtful feedback, and valuable insights on my research.

I am also grateful to Chantal Demont, the administrative manager of our lab, for her kindness and efficiency in handling countless administrative tasks, both within and beyond EPFL, that made my academic life smoother. My thanks also go to all the other administrative staff at EPFL for their support throughout the years.

I sincerely thank all my labmates and postdocs at the Integrated Systems Laboratory (LSI): Mingfei, Andrea, Chang, Heinz, Mathias, Fereshte, Sonia, Alessandro, Rehab, and Rassul. Their collaboration and invaluable insights have been instrumental throughout this journey.

I warmly acknowledge the Processor Architecture Laboratory (LAP), where I worked as a part-time research scholar prior to my doctoral studies. I am grateful to Chantal Schneeberger, the administrative assistant, for her invaluable help with administrative tasks, and to Andrea and Stefan for their friendly support and engaging discussions.

I deeply appreciate my coauthors and collaborators from Synopsys: Eleonora Testa, Luca Amarù, Walter Lau Neto, and Giulia Meuli, whose contributions and discussions have significantly shaped my research. Additionally, I am grateful to Alan Mishchenko from UC Berkeley and Marcel Walter and Robert Wille from TUM for their invaluable collaboration and knowledge-sharing.

Teaching has been an enriching experience, and I extend my gratitude to the professors under whom I worked as a teaching assistant: Ties Jan Henderikus Kluter and Nicolas Boumal. Their dedication to teaching has been truly inspiring.

I express my sincere gratitude to the staff of the Department of Electronics and Telecommunication Engineering at the University of Moratuwa, where I completed my bachelor's degree, especially Dr. Ranga, Dr. Pasqual, Prof. Dileeka, Dr. Thayaparan, Dr. Prathapasinghe, and Dr. Dulika, for their invaluable mentorship and guidance. I also cherish the friendships I have built with my batchmates and colleagues: Shanika, Danushi, Shehara, Dakila, Duvindu, Ishara, Ishan, Kalindu, Pasan, Poorna, Ridwan, Rushen, Thilina, Dinushani, Sachinthana, Hasantha, Kawshalya, Wageesha, Senani, Shalanika, Madushika, Ayanga, Sanduni, Kasun, Dumindu, Vikum, Janith, and Gresha. Thank you for the shared memories and unwavering support.

I would also like to acknowledge the support and encouragement of my teachers and friends from my high school, Sirimavo Bandaranaike Vidyalaya. In particular, I am grateful to my teachers: Mrs. Dharshani, Mrs. Dulini, Mrs. Samitha, Mrs. Kamalini, Mrs. Chandani, and Mrs. Hemamala for their dedicated teaching and encouragement. My heartfelt thanks also go to my high school friends, especially Prabodha, Suvini, and Lakshika, for their constant support.

My sincere appreciation goes to my Sri Lankan friends in Switzerland—Anuradha, Ruchiranga, Yasara, Wenuka, Heshani, Udaranga, Pasindu, and Lakmal—for their immense support and for making me feel at home even in a foreign land.

I am also indebted to the medical doctors who have helped me during difficult times: Dr. Dilshani, Dr. Buddhini, Dr. Sanduni, and Dr. Kelum. Their care and support have been invaluable.

I extend my sincere gratitude to the taxpayers of Sri Lanka, Switzerland, and the European Union, as well as other funding providers for EPFL research, for enabling me to pursue my education from kindergarten to Ph.D.

Last but not least, words cannot express my profound gratitude to my family. My parents, Kalyanapala Marakkalage and Wansika Julgoda Manage, have been my unwavering pillars of strength, and I am forever grateful for their unconditional love and encouragement. I extend my heartfelt gratitude to my brother, Hasala, and my sister, Wathmini, who have always been a constant source of affection and encouragement. I offer my deepest thanks to my husband, Buddhima Gamlath, for his love, constant care, and support. I am also grateful to Buddhima's parents, Amarasooriya and Vijitha, Hasala's wife, Thathsara, as well as Buddhima's brother, Kassapa, and his wife, Sasankara, for their kindness and support.

This journey would not have been possible without the collective support of everyone. I am deeply grateful for the contributions of all, whether explicitly mentioned or otherwise, and your kindness has been invaluable. Thank you, from the bottom of my heart.

February 20, 2025 Dewmini Sudara Marakkalage

Abstract

Logic synthesis is a cornerstone of *electronic design automation* (EDA), facilitating the optimization of *power*, *performance*, and *area* (PPA) in integrated circuits. As *complementary metal-oxide-semiconductor* (CMOS) technology approaches its physical limitations, achieving further improvements using conventional methods has become increasingly difficult. Simultaneously, emerging *post-CMOS* technologies offer significant potential for enhanced energy efficiency, performance, and scalability, but introduce unique constraints that are often unmet by traditional synthesis approaches. Motivated by these challenges, this thesis explores two key research directions: advanced optimization techniques for CMOS and novel synthesis methodologies tailored to post-CMOS technologies.

The first part of this thesis focuses on advancing CMOS logic synthesis through *sequential optimization*, which exploits the broader solution space afforded by the reachable states of memory elements. We introduce a scalable algorithm leveraging *sequential observability don't cares* (SODCs) to enhance *redundancy removal* and *resubstitution* via k-step *sequential induction*. Our approach achieves an average area reduction of 6.9% after *technology mapping*, with post-layout reductions of 2.89% and 1.43% in combinational and sequential areas, respectively.

The second part of this thesis addresses the specific constraints of emerging post-CMOS technologies, including fanout limits, path balancing, and planarization requirements. First, we treat *fanout-bounded synthesis* as a general problem, considering its applications also in the CMOS domain; we develop exact and heuristic algorithms for fanout-bounded synthesis, realizing an average area reduction of 11.82% compared to *state-of-the-art* (SOTA) techniques. We also demonstrate the adaptability of these methods to emerging technologies such as *adiabatic quantum-flux-parametron* (AQFP) circuits. We then propose an exact-synthesis-database-driven approach for solving the splitter and buffer insertion problem in superconducting electronics, achieving up to 40% reduction in critical path delay and a 21% decrease in area for AQFP circuits. Finally, building on the same database-driven approach, we present a general synthesis framework for emerging technologies, incorporating planarization constraints that are particularly critical for *field-coupled nanocomputing* (FCN) technologies. For FCN technology, our method reduces buffer count, crossing count, and critical path length by 84.5%, 74.5%, and 65.2%, respectively.

This thesis provides scalable and effective methodologies for optimizing circuits in both CMOS and post-CMOS domains. By advancing sequential synthesis and integrating emerging technology constraints into the synthesis process, this work establishes a foundation for the efficient design of next-generation digital systems.

Keywords: Logic synthesis, electronic design automation, CMOS, Post-CMOS, emerging technologies, sequential synthesis, superconducting electronics, AQFP, FCN, path-balancing, buffer insertion, planarization, exact synthesis.

Résumé

La synthèse logique est une pierre angulaire de l'automatisation de la conception électronique (EDA), optimisant la puissance, les performances et la surface (PPA) des circuits intégrés. À mesure que la technologie CMOS (complementary metal-oxide-semiconductor) atteint ses limites physiques, il devient de plus en plus difficile d'obtenir des améliorations avec des méthodes conventionnelles. En parallèle, les technologies post-CMOS émergentes offrent un potentiel considérable pour améliorer l'efficacité énergétique, les performances et la scalabilité, tout en introduisant des contraintes uniques souvent ignorées par les approches traditionnelles. Cette thèse explore deux axes principaux : des techniques d'optimisation avancées pour le CMOS et des méthodologies de synthèse adaptées aux technologies post-CMOS.

La première partie se concentre sur l'optimisation séquentielle pour la synthèse logique CMOS, exploitant les états atteignables des composants de mémoire. Un algorithme basé sur les "Sequential Observability Don't Cares" (SODCs) est proposé, permettant une réduction moyenne de la surface de 6,9 % après le mappage technologique et des réductions post-routage de 2,89 % et 1,43 % pour les zones combinatoires et séquentielles.

La deuxième partie traite des contraintes des technologies post-CMOS, telles que les limites de fanout, l'équilibrage des chemins et les exigences de planarisation. Nous développons des algorithmes exacts et heuristiques pour la synthèse sous contrainte de fanout, réalisant une réduction moyenne de la surface de 11,82 %. Ces méthodes sont adaptées aux circuits émergents comme l'Adiabatic Quantum-Flux-Parametron (AQFP), où une approche basée sur une base de données permet de réduire jusqu'à 40 % le délai critique et 21 % la surface. Un cadre de synthèse général est également proposé pour intégrer les contraintes de planarisation des technologies de Nanotechnologies à Couplage de Champs (FCN), réduisant les buffers, croisements et longueurs des chemins critiques de 84,5 %, 74,5 % et 65,2 %, respectivement.

Cette thèse propose des méthodologies évolutives et efficaces pour optimiser les circuits dans les domaines CMOS et post-CMOS, jetant les bases d'une conception efficace des systèmes numériques de la prochaine génération.

Mots-clés : Synthèse logique, automatisation de la conception électronique, CMOS, Post-CMOS, technologies émergentes, synthèse séquentielle, électronique supraconductrice, AQFP, FCN, équilibrage des chemins, insertion de buffers, planarisation, synthèse exacte.

Contents

Acknowledgements **i**

Abstract (English/Français) **iii**

List of Figures **xi**

List of Tables **xiii**

List of Acronyms **xv**

1 Introduction **1**
1.1 Electronic Design Automation (EDA) 2
1.2 Logic Synthesis 5
1.3 Challenges in Logic Synthesis 7
1.4 Thesis Contributions Overview 9
1.4.1 Scalable Sequential Logic Synthesis 9
1.4.2 Fanout-Bounded Logic Synthesis 10
1.4.3 Logic Synthesis for AQFP Technology 10
1.4.4 Logic Synthesis for FCN Technologies 11
1.5 Thesis Organization 13

2 Background **15**
2.1 Boolean Algebra and Functions 15
2.1.1 Boolean Algebra 15
2.1.2 Boolean Functions 16
2.1.3 Equivalence of Boolean Functions 17
2.2 Logic Representations 18
2.2.1 Truth Tables 18
2.2.2 Algebraic Expressions 19
2.2.3 Canonical Forms 19
2.2.4 Binary Decision Diagrams (BDDs) 20
2.2.5 Logic Networks 20
2.2.6 And-Inverter Graphs (AIGs) 21
2.2.7 Majority-Inverter Graphs (MIGs) 22

2.3 Logic Optimization 22
2.4 Post-CMOS / Emerging Technologies 24
2.4.1 Superconducting Electronics (SCE) 24
2.4.2 Field-Coupled Nanotechnologies (FCN) 26

3 Scalable Sequential Logic Synthesis 27
3.1 Introduction 27
3.2 Preliminaries 29
3.2.1 Sequential Logic Optimizations 29
3.2.2 Don't Cares in Logic Networks 30
3.2.3 Prior Work on Sequential Synthesis 31
3.3 Scalable Sequential Optimization 32
3.3.1 Motivation 32
3.3.2 Sequential ODCs 33
3.3.3 Framework Definition 34
3.3.4 Proposed Method 35
3.3.5 Complete Algorithm 38
3.3.6 Correctness of the Proposed Approach 40
3.3.7 Characterizing SODC-Optimizable Transformations 44
3.4 Experimental Results 49
3.4.1 Comparison of Different Configurations 49
3.4.2 Technology Mapped Results 52
3.4.3 Post Place & Route Results on Industrial Designs 53
3.5 Summary 54

4 Fanout-Bounded Logic Synthesis 57
4.1 Introduction 57
4.2 Preliminaries 60
4.2.1 And-Inverter Graphs / Majority-Inverter Graphs 60
4.2.2 Static Timing Analysis 60
4.2.3 Node Equivalence 61
4.2.4 AQFP Logic Circuits 61
4.3 Related Work 62
4.3.1 General Fanout-Bounded Synthesis 62
4.3.2 Path-Balanced Fanout-Bounded Synthesis 64
4.4 Globally Optimum General Fanout-Bounded Synthesis 65
4.5 Top-Down Heuristic Approach for General Fanout-Bounded Synthesis Problem 69
4.5.1 Improved Top-Down Approach with Over-Duplication 72
4.6 Path-Balanced Fanout-bounded Synthesis 74
4.6.1 ILP Formulation for the Global Optimum 74
4.6.2 Scalable Heuristic Approach 75
4.7 Experimental Results 77

4.7.1 Global Optimum for General Fanout-Bounded Synthesis 78
4.7.2 Heuristics for General Fanout-Bounded Synthesis 79
4.7.3 Global Optimum Splitter/Buffer Insertion for AQFP 81
4.7.4 Heuristic Splitter/Buffer Insertion for AQFP 85
4.8 Summary 85

5 Logic Synthesis for AQFP Technology 87
5.1 Introduction 87
5.2 Motivation 90
5.3 Preliminaries 92
5.3.1 Majority-Inverter Graphs (MIGs) 92
5.3.2 AQFP logic circuits 93
5.3.3 Exact synthesis 93
5.4 AQFP Resynthesis Approach 94
5.4.1 Generation of the database 94
5.4.2 Synthesis Algorithm 99
5.5 Experimental Results 101
5.6 Summary 105

6 Logic Synthesis for FCN Technologies 107
6.1 Introduction 107
6.2 Preliminaries 108
6.2.1 Beyond-CMOS Technologies with Unconventional Costs 109
6.2.2 Circuit Model 112
6.2.3 Conventional Technology Mapping 112
6.3 Proposed Methodology 113
6.3.1 Generation of Optimal Subcircuits 114
6.3.2 Rewriting Using the Exact Database 115
6.4 Experimental Evaluation 117
6.4.1 Experimental Setup 117
6.4.2 Results 119
6.5 Summary 120

7 Conclusion 121
7.1 Scalable Sequential Logic Synthesis 121
7.2 Fanout-Bounded Logic Synthesis 122
7.3 Logic Synthesis for AQFP Technology 123
7.4 Logic Synthesis for FCN Technologies 125
7.5 Final Remarks 126

Bibliography 127

Curriculum Vitae 143

List of Figures

1.1 A simplified synthesis flow of a typical digital system. 4

3.1 A combinational logic network (a) and its optimized version (b). 32
3.2 A sequential logic network (a), its optimized version (b), and its base case network (c) and the inductive case network (d) for 1-step sequential induction. 33
3.3 A sequential logic network (a), its optimized version (b), and its inductive case network (c) for 2-step sequential induction. 37
3.4 A sequential logic network with feedback (a) and its inductive case network (b) for 1-step sequential induction before applying assumptions in the first frame. 38
3.5 An example sequential circuit and its frame representation. 45
3.6 The inductive network for the sequential circuit in Figure 3.5. 45
3.7 The inductive network for the sequential network in Figure 3.5, where the decomposition of the 3-input AND is unfavorable to the proposed algorithm. 46
3.8 The inductive network for the sequential network in Figure 3.5, where the decomposition of the 3-input AND is conducive to the proposed algorithm. 47
3.9 The inductive network for the sequential network in Figure 3.5, where shadow nodes are added to reflect the assumptions on the reachable states of registers. 47

4.1 Example logic network (left) and two of its possible fanout-bounded, path-balanced versions targeting AQFP technology assuming a fanout capacity of 1 for gates and 3 for splitters. (Buffers and splitters are shown by triangles.) The version in the middle does not use any gate duplication whereas the version on the right allows gate duplication resulting in a reduction in both the overall number of logic levels as well as the total area. 62
4.2 Example logic network (left) and a possible fanout-bounded version assuming a fanout limit of 2 (right). 65
4.3 A fanout net for a node n with levels of fanouts already decided (a), two possible outcomes for the fanout net of n if the algorithm of [178] is used (b and c), and the optimum buffer tree for n (d) when $f_{\text{buff}} = f_{\text{gate}} = 3$ and $c_{\text{gate}} > c_{\text{buff}}$. 71

4.4 An intermediate step of fanout-bounded synthesis with levels decided for all nodes except n, n_1, n_2 (top), the synthesized fanout nets by the algorithm described in naive top-down approach (middle), and the synthesized fanout nets if over-duplication allowed (bottom) when $f_{\text{buff}} = f_{\text{gate}} = 3$ and $c_{\text{gate}} > c_{\text{buff}}$. 73
4.5 Two possibilities for a part of an AQFP netlist and their retimed version. 77

5.1 An example logic network with unit-delay gates (left) and its splitter-inserted, path-balanced version (right). 88
5.2 A part of a logic network with unit-delay gates (left), and its path-balanced versions (middle and right) using two choices of locally optimum splitter-trees with 1-to-2 splitters. 90
5.3 (a) A logic network with unit-delay gates, (b) its path balanced version using a balanced splitter tree, (c) an optimized version of (a) by pushing one splitter up in the hierarchy, and (d) optimum path balancing with an unbalanced splitter tree. 91
5.4 Two partial DAGs (left and middle) and a DAG (right). The partial DAG in the middle is derived from the one on the left by tying a new gate and connecting it to one uncommitted leaf in each of the existing gates. The DAG on the right is derived from the partial DAG on the left by attaching four inputs (leaf nodes) to the five uncommitted leaves. 95
5.5 Two fanout-nets with the same relative fanout levels. 96
5.6 Computing best area for a relative level configuration using dynamic programming. 97
5.7 Three different buffer-splitter configurations to realize the same DAG structure. 98

6.1 Elementary FCN devices. 109
6.2 QCA gates and wire segments. 110
6.3 Two realizations of a network computing $o_1 = a \wedge b \wedge c \wedge d$ and $o_2 = \neg a \wedge \neg b \wedge \neg c \wedge \neg d$. Inverters are denoted by dashed edges and crossing cells are denoted by a $\times$ symbol. 113

List of Tables

3.1 Comparison of the proposed method against the baseline (redundancy removal) 50
3.2 Comparison of the proposed method against the baseline (redundancy removal + resubstitution) 51
3.3 Comparison of the proposed method against the baseline (k-step induction) 52
3.4 Comparison of the proposed method against the baseline (Increased window size) 53
3.5 Results after technology mapping for OpenCores designs 54
3.6 Results after place and route for industrial designs 54

4.1 The global optimums for general fanout-bounded synthesis. 78
4.2 Results of the top-down fanout-bounded synthesis on benchmarks of [76]. 79
4.3 Results of the top-down fanout-bounded synthesis algorithm on EPFL benchmarks. 80
4.4 Results of AQFP splitter/buffer insertion without gate duplication. 82
4.5 Results of AQFP splitter/buffer insertion with gate duplication (part i). . 83
4.6 Results of AQFP splitter/buffer insertion with gate duplication (part ii). . 84
4.7 Results of scalable heuristic approach for AQFP. 84

5.1 Results for the experiment where the proposed AQFP synthesis algorithm is applied for 10 iterations under the assumption that no splitter-buffer insertion is needed for primary inputs but primary outputs need path-balancing. The *reference* column shows the optimized results from [163]. . 102
5.2 JJ utilization of majority-3 and majority-5 gates in the output of the proposed AQFP synthesis algorithm under the assumption that no splitter-buffer insertion is needed for primary inputs but primary outputs need path-balancing. 103
5.3 Results for the experiment where the proposed AQFP synthesis algorithm is applied for 10 iterations under the assumption that primary inputs need splitters to support multiple fanouts and primary outputs need path-balancing. 104

6.1 Results of the proposed technology mapping approach on the ISCAS [38] and EPFL [11] benchmark suites. 118

List of Acronyms

AIG	And-Inverter Graph
AQFP	Adiabatic Quantum-Flux-Parametron
AI	Artificial Intelligence
ASIC	Application-Specific Integrated Circuit
BDD	Binary Decision Diagram
CAD	Computer-Aided Design
CDC	Controllability Don't Care
CMOS	Complementary Metal-Oxide-Semiconductor
CODC	Compatible Observability Don't Care
DAG	Directed Acyclic Graph
EDA	Electronic Design Automation
FCN	Field-Coupled Nanocomputing
FBS	Fanout-Bounded Synthesis
HDL	Hardware Description Language
HLS	High-Level Synthesis
IC	Integrated Circuit
ILP	Integer Linear Programming
IoT	Internet of Things
JJ	Josephson Junction
LUT	Look-Up Table
MFFC	Maximum Fanout-Free Cone
MIG	Majority-Inverter Graph
NML	Nanomagnetic Logic

NPN	Negation-Permutation-Negation
ODC	Observability Don’t Care
PI	Primary Input
PO	Primary Output
POS	Product of Sums
PPA	Power, Performance, Area
QCA	Quantum-dot Cellular Automata
RI	Register Input
RO	Register Output
RSFQ	Rapid Single Flux Quantum
RTL	Register Transfer Level
SAT	Satisfiability
SCE	Superconducting Electronics
SDC	Satisfiability Don’t Care
SMT	Satisfiability Modulo Theory
SoC	System-on-Chip
SoTA	State of The Art
SODC	Sequential Observability Don’t Care
SOP	Sum of Products
SSW	Sequential SAT-Sweeping
TFI	Transitive Fan-In
TFO	Transitive Fan-Out
XAG	XOR-And Graph
XMG	XOR-Majority Graph

1 Introduction

From robotic submarines exploring the deepest trenches of the ocean to space probes venturing beyond the Solar System, electronic systems have seamlessly integrated into nearly every facet of modern technology.

The rapid evolution of electronic devices has fueled an unrelenting demand for more efficient, high-performance *integrated circuits* (ICs) capable of performing complex computations while minimizing power consumption and shrinking in size. To address these challenges, *electronic design automation* (EDA) tools have become indispensable in modern semiconductor design, providing robust frameworks for optimizing circuits in terms of *power*, *performance*, and *area* (PPA).

Among these tools, *logic synthesis* plays a pivotal role, refining digital circuit designs at the logical level by transforming initial specifications into optimized structures ready for physical design and subsequent synthesis steps. Along with other steps of the EDA flow and advancements in semiconductor technology, several decades of research in logic synthesis have been instrumental in keeping pace with the ever-increasing complexity of modern electronic systems, enabling Moore's Law to continue its course. In this thesis, we lay the groundwork for further advancements in logic synthesis, focusing on two key research directions: scalable sequential logic synthesis and synthesis for emerging computing paradigms.

Since the advent of the first integrated circuits in the 1960s, the semiconductor industry has witnessed a remarkable transformation, driven by continuous advancements in process technology and design methodologies. *Complementary metal-oxide-semiconductor* (CMOS) technology has been the dominant semiconductor technology for decades, offering a robust foundation for the development of high-performance, low-power ICs. Consequently, logic synthesis has primarily focused on optimizing circuits for CMOS technology, leveraging a wide array of algorithms and techniques to achieve superior PPA metrics.

Among existing research on logic synthesis for CMOS, the majority of work has concentrated on optimizing *combinational* logic, which forms the core of digital circuits, given that combinational logic dominates the area and power consumption of most designs. The synthesis of *sequential logic*, which deals with circuits containing memory elements, has received relatively less attention. However, despite its name, sequential logic synthesis is not just about optimizing sequential memory elements but also about optimizing the combinational logic that interacts with these memory elements, taking advantage of the fact that the set of reachable states of the memory elements is often smaller than the set of all possible states. Thus, sequential logic synthesis has the potential to significantly improve PPA metrics of digital circuits. In this thesis, we introduce a novel sequential synthesis method that leverages *sequential observability don't cares* (SODCs) to optimize sequential circuits across reachable states.

On the other hand, as CMOS technology approaches its scaling limits, emerging computing paradigms have emerged as promising alternatives. For instance, superconducting technologies such as *adiabatic quantum-flux parametron* (AQFP) and *field-coupled nanocomputing* (FCN) technologies have gained traction for their potential to deliver high-performance, ultra-low-power solutions. In many of these post-CMOS technologies, the fundamental logic gates differ significantly from those of CMOS. Moreover, they often come with substantial interconnect overhead, arising from unique physical constraints such as tight fanout bounds, the need for specialized components to support multiple fanouts, precise signal synchronization requirements between gates, and the need for special handling of wire crossings. As a result, traditional CMOS-centric synthesis methods are not directly applicable to these technologies. Even when adapted, these methods often yield suboptimal results due to the lack of consideration of interconnect costs. In this thesis, we introduce novel synthesis methods focusing on post-CMOS technologies that account for the unique constraints of these paradigms, achieving significant PPA improvements.

In the following sections, we provide an overview of the EDA and the role of logic synthesis within this domain. We then introduce the contributions of this thesis, highlighting the key research directions and the novel methods developed to address the challenges in sequential logic synthesis and synthesis for emerging computing paradigms. We conclude this chapter by outlining the organization of the subsequent chapters.

1.1 Electronic Design Automation (EDA)

Electronic design automation (EDA) refers to the collection of software tools and methodologies used to design, verify, and produce electronic systems and integrated circuits (ICs). In the modern electronics industry, EDA plays a critical role in managing the complexity of advanced circuit designs, thus reducing time-to-market and development costs. As technology continues to scale, the reliance on EDA tools has grown exponentially, making them indispensable in every stage of the design process.

The EDA market has seen significant growth in recent years, driven by the increasing demand for semiconductors, the rise of AI and IoT applications, and advancements in manufacturing processes. Major players in the EDA industry, including Synopsys, Cadence, and Siemens EDA, are continually innovating to support the design of next-generation technologies. The market is also witnessing the emergence of niche players and open-source tools, diversifying the ecosystem. The EDA industry was valued at USD 10.40 billion in 2019, and it is expected to grow up to USD 15.38 billion by 2025 [105].

The history of EDA dates back to the 1960s and 1970s, when early *computer-aided design* (CAD) tools such as schematic capture programs, simulation software, and layout editors were developed, forming the foundation for modern EDA tools. The 1980s brought a major breakthrough with *hardware description languages* (HDLs) like VHDL and Verilog, enabling abstract and efficient circuit representations. This advancement facilitated the modeling, verification, and synthesis of complex systems.

During the 1990s and 2000s, EDA tools evolved to include comprehensive automation, such as synthesis, place-and-route, and verification, significantly enhancing design efficiency and handling the increasing complexity of ICs. Modern EDA tools are highly sophisticated, offering extensive features for designing *system-on-chip* (SoC) and *application-specific integrated circuits* (ASICs) including some recent GPUs and CPUs having well over a hundred billion transistors [1]. Academic research on EDA has also been pivotal, contributing open-source tools (e.g., ABC [34], Yosys [174]) and libraries (e.g., EPFL Logic Synthesis Libraries [154]). These efforts complement industrial advancements, fostering innovation and progress in EDA technologies.

The EDA flow represents the sequence of processes and tools used to design an IC, from its initial specification to final tape-out. The typical EDA flow for digital IC design consists of several stages; a simplified version of a typical EDA flow is shown in Fig. 1.1 where each step addresses specific aspects of the design process. Starting with the design specification, the flow progresses through high-level/behavioral synthesis, architectural synthesis, logic synthesis, and physical synthesis to generate a fabrication-ready layout. Each stage involves a series of optimizations, transformations, and verifications to ensure that the design meets the desired specifications and constraints.

We remark, however, that a modern EDA flow is much more complex and non-linear than the simplified flow shown in Fig. 1.1. For instance, the flow may involve multiple iterations between different stages, feedback loops, etc. to optimize the design.

Design Specification: The high-level requirements and functionality of the circuit are defined. This stage involves understanding the system's purpose, target applications, and performance goals.

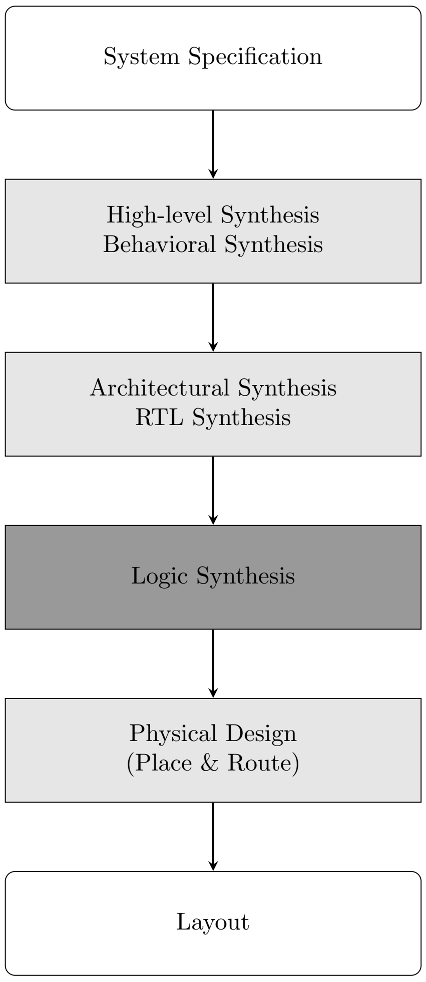


Figure 1.1: A simplified synthesis flow of a typical digital system.

High-level Synthesis/Behavioral Synthesis: *High-Level Synthesis* (HLS), sometimes also referred to as Behavioral Synthesis, converts a high-level description of a system's behavior, often written in languages like C, C++, or SystemC, into a *register-transfer-level* (RTL) representation. This process involves:

- operation scheduling, where the order of operations is determined to optimize factors such as latency and throughput,
- allocation of resources such as functional units, registers, and memories, and
- generation of control logic to manage the flow of data and operations execution.

In short, HLS translates abstract specifications into a format ready for hardware implementation, while performing some initial PPA optimizations.

Architectural Synthesis: This step transforms an RTL description, which defines the behavior of a circuit in terms of data transfer between registers and combinational logic, into a gate-level netlist. This step also performs some optimizations to reduce circuit complexity and meet timing constraints, and it includes resource mapping to translate RTL components like adders and multiplexers into standard logic gates. The output of RTL synthesis is a technology-independent gate-level design.

Logic Synthesis: In logic synthesis, the gate-level netlist generated during RTL synthesis is extensively optimized to further improve PPA using both technology-independent and technology-specific optimizations. Transforming the optimized technology-independent netlist into a technology-specific netlist and subsequent technology-dependent optimization are often referred to as technology mapping. A more detailed discussion on logic synthesis is provided in Section 1.2.

Physical Synthesis: Physical synthesis, consisting of place & route, converts the technology-mapped netlist into a complete physical layout, including floorplanning, placement, clock tree synthesis, and routing. This stage ensures that the design meets the constraints for manufacturing, signal integrity requirements, and timing closure goals. Physical design is crucial for preparing the design for fabrication and ensuring that the layout adheres to design constraints.

In each stage of the EDA flow, verification is a critical component that ensures the design meets all functional, timing, power, and reliability specifications. An integral part of verification is equivalence checking, which ensures that the synthesized netlist at the end of each stage is functionally equivalent to the original RTL design.

The contributions of this thesis focus on the logic synthesis stage of the EDA flow. In the next section, we delve into the details of logic synthesis and technology mapping, highlighting the challenges and opportunities in these areas.

1.2 Logic Synthesis

Logic synthesis and technology mapping are critical steps in the EDA process, serving as a bridge between high-level design specifications and physical implementation. These steps ensure that a digital design adheres to PPA constraints while meeting the requirements of the target manufacturing technology.

Broadly, logic synthesis involves finding optimized representations of Boolean functions. The definition of *optimized* depends on the chosen criteria and representation framework. For instance, one logic synthesis problem involves finding the *sum-of-products* (SOP) expression with the fewest literals for a given Boolean function. Another example is identifying the smallest depth or size circuit consisting of 2-input AND gates and inverters for a Boolean function.

The origins of logic synthesis predate the EDA industry itself. While the roots trace back to George Boole's foundational work on Boolean algebra in the 1800s, the field began to take a formal shape with Claude Shannon's seminal work [147] in the 1930s, which demonstrated the use of Boolean algebra for representing and analyzing switching circuits. Classic Boolean optimization techniques, such as Karnaugh Maps [85] and the Quine-McCluskey methods [116, 136], were introduced in the 1950s. Over time, increasingly sophisticated synthesis algorithms were developed, evolving the field into a mature domain with a rich set of methodologies and tools.

Many logic synthesis problems are computationally intractable, with no known efficient algorithms to solve them optimally. As a result, a variety of heuristic-based techniques have been developed to optimize logic representations. These approaches often combine algebraic, Boolean, and exact optimization techniques to achieve high-quality results.

In digital circuit optimization, logic synthesis can be categorized into two main types: *combinational logic synthesis* and *sequential logic synthesis*.

Combinational logic synthesis focuses on optimizing circuits while treating memory elements, such as flip-flops and latches, as black boxes. In this approach, registers are treated as independent pairs of primary inputs and outputs, assuming no dependencies among them. **Sequential logic synthesis** exploits dependencies among registers to optimize not only the sequential elements but also the combinational logic portion of the design. This can significantly enhance PPA metrics by taking advantage of the sequential nature of the circuit.

Logic synthesis typically consists of two main stages: *technology-independent optimization* and *technology mapping*.

Technology-independent optimization: This stage optimizes the logic representation of the design without considering the target technology. While simple logic functions can be represented as truth tables or Boolean expressions, such representations do not scale well for large designs. Therefore, synthesis tools often use graphical representations, such as *directed acyclic graphs* (DAGs), to represent logic functions efficiently. A detailed discussion of various logic representations is presented in Section 2.2. Common optimization objectives during this stage include minimizing the number of gates and circuit depth, which correlate to area and delay, respectively. Techniques such as rewriting, refactoring, and resubstitution, with varying levels of complexity, are applied to optimize these metrics [118].

Technology mapping: This stage involves mapping the technology-independent, optimized logic representation onto a target technology library and performing technology-specific optimizations. Technology mapping converts the logic representation into a technology-specific netlist, where gates are selected from the target library [49, 50, 75, 84, 94]. During this stage, more accurate technology-specific models for area, delay, and power are used compared to the technology-independent optimization stage.

Overall, logic synthesis is a critical step in the EDA flow. In today's technological landscape, logic synthesis is a key enabler for achieving high-performance, low-power designs at a lower cost, even as the technology itself is reaching its downscaling limits. In general, logic synthesis is relevant not only in traditional EDA, but also in other application fields such as cryptography [161, 177] and quantum computing [117].

1.3 Challenges in Logic Synthesis

The work of this thesis is inspired by two key challenges in logic synthesis and technology mapping:

1. Extend the advanced combinational logic synthesis techniques to sequential logic synthesis.
2. Develop synthesis methods tailored for unique constraints of emerging computing paradigms.

Enhancing Sequential Logic Synthesis

Many existing logic synthesis techniques have been tailored for combinational logic synthesis; though sequential logic synthesis has also been studied, there is still room for improvement. Recently, there has been a growing interest in exploring sequential logic synthesis techniques as the ever more complex designs require more sophisticated optimizations to keep up with the growing demands for performance, power, and area efficiency. To this end, a notable recent development is the introduction of sequential SAT-sweeping (SSW) [125], which is an extension of combinational SAT-sweeping [93, 121, 122] to sequential circuits.

A natural question is whether there are other advanced combinational logic synthesis techniques that can potentially be extended to sequential logic synthesis to uncover more optimization opportunities. One promising candidate is the use of *observability don't cares* (ODCs). In combinational logic synthesis, ODCs have been shown to provide significant optimization opportunities by identifying redundant logic. The key idea behind ODCs is to focus on internal wires that are not observable at the primary outputs under certain input conditions and to simplify the logic based on these conditions.

Effectively extending the ODC-based optimizations to sequential logic is challenging as the synthesis algorithms have to simultaneously work on two tasks:

1. argue about potential reachable state combinations and
2. handle dependencies among ODC-based simplifications.

In this thesis, we introduce a novel synthesis algorithm that successfully executes these tasks, leveraging *sequential observability don't cares* (SODCs) to optimize sequential circuits across reachable states.

Synthesis for Post-CMOS Technologies

With the advent of CMOS technology in the 1980s, which has since dominated the semiconductor industry, logic synthesis has primarily focused on optimizing circuits for CMOS. The internal logic representations employed in synthesis tools, as well as their associated optimization metrics, have been heavily influenced by the characteristics of CMOS technology. For instance, graphical representations of logic consisting of AND gates and inverters have been widely adopted in synthesis tools, as these gates form the basic building blocks of CMOS circuits. Optimization metrics in this domain have typically focused on minimizing the number of gates and the circuit depth (i.e., the number of gates in the longest input-to-output path), which directly correlate to the area and delay of the synthesized circuits.

In emerging technologies, however, as discussed in Section 2.4, several key differences render traditional CMOS-centric synthesis methods less effective. For example, in many superconducting technologies—promising post-CMOS alternatives—the fundamental logic gate is often the majority gate rather than AND or OR gates. Consequently, logic optimized for CMOS may not translate efficiently to these technologies.

Additionally, emerging technologies often incur significant interconnect overhead due to various physical constraints. These include:

- **Tight fanout bounds:** Multi-fanout nets require additional hardware, such as splitters, to support multiple fanouts.
- **Balanced logic paths:** Stringent requirements for path balancing necessitate increased buffering.
- **Special handling of wire crossings:** Certain technologies require additional resources to manage wire crossings effectively.

These constraints necessitate additional synthesis and optimization stages to properly account for interconnect requirements and optimize interconnect resources. As a result, traditional CMOS-centric synthesis methods are not readily applicable to these technologies.

In this thesis, we propose novel synthesis techniques to address these constraints at the logic synthesis stage, enabling efficient optimization tailored to the unique characteristics of emerging post-CMOS technologies.

1.4 Thesis Contributions Overview

In this section, we provide an overview of the contributions of this thesis. The first part of the thesis focuses on scalable sequential logic synthesis, while the second part addresses logic synthesis for emerging computing paradigms.

1.4.1 Scalable Sequential Logic Synthesis

Sequential logic synthesis offers powerful optimizations for circuits with memory elements by exploiting the reachable states and transition behavior of registers. Unlike combinational synthesis, which optimizes based on stateless inputs and outputs, sequential synthesis considers the inherent statefulness of circuits, allowing for potentially significant improvements in PPA. This work presents a novel method for don't-care-based sequential logic synthesis, using an advanced algorithm that leverages sequential observability don't cares (SODCs) to perform redundancy removal and resubstitution under sequential conditions.

The proposed approach builds on the concept of sequential k-step induction, where both a base case and an inductive case network are optimized simultaneously. By examining dependencies within the SODCs—where certain inputs or states are not critical for the circuit's observable output—this method optimizes the circuit with an enhanced focus on the sequentially reachable states. Unlike previous approaches, this method incorporates SODC dependencies directly, without limiting the solution space to *compatible ODC* (CODC) simplifications, which often miss optimization opportunities. The k-step induction model enables scalability by applying windowing, thus ensuring manageable SAT problem sizes for larger designs.

Experimental results demonstrate that this approach achieves superior PPA metrics when compared to existing sequential synthesis methods. Implemented in an industrial setting, this method achieved an average area reduction of 6.9% after technology mapping, with additional reductions of 2.89% in combinational area and 1.43% in sequential area in post-place-and-route designs. This high-efficiency approach to sequential synthesis showcases its potential for substantial area savings in practical designs, with verified correctness through advanced sequential verification tools.

By integrating this method within a full EDA flow, this work addresses one of the key challenges in sequential synthesis—balancing scalability with the complexity of sequential dependencies. The results underline the benefits of considering don't-care conditions that include sequential reachability, providing an effective framework for scalable, high-quality synthesis in sequential circuits.

1.4.2 Fanout-Bounded Logic Synthesis

Fanout-bounded synthesis is a design approach tailored to meet the specific constraints of emerging technologies like AQFP and FCN, where traditional CMOS-based synthesis methods are insufficient. In these post-CMOS paradigms, constraints on fanout and path balancing are crucial due to the distinct operational mechanisms that require precise control over signal propagation. For instance, AQFP circuits cannot drive multiple fanouts directly from a single gate; instead, splitters are required to distribute the signal, adding substantial overhead to both area and delay. To accommodate these requirements, fanout-bounded synthesis focuses on minimizing the use of splitters and buffers while adhering to a bounded fanout constraint, ensuring that each gate's fanout does not exceed a specified limit.

Taking a rigorous approach to the generic fanout-bounded synthesis problem, this work formulates the problem as an Integer Linear Programming (ILP) model, in order to minimize the total area under a fixed delay constraint. The ILP model incorporates variables representing the number of gate copies and buffers associated with each gate and each logic level and introduces constraints to ensure that each level has sufficient fanout capacity to drive the fanouts in subsequent levels. By solving the ILP with a commercial solver, we obtain optimum solutions for benchmark circuits up to a few hundred gates in size. We also extend the ILP to the path-balanced setting, targeting the AQFP technology, demonstrating the versatility of our approach.

For scalability, the ILP-based fanout-bounded synthesis is complemented by heuristic approaches, which reduce computation time while achieving significantly better results as compared to the state-of-the-art. Our heuristic employs a top-down approach, initially favoring the addition of buffers over duplicating gates to avoid unnecessary area increases. Critical paths are prioritized in the synthesis process to ensure that timing requirements are met without excessive buffering. Additionally, path-balancing heuristics are applied to synchronize the arrival of signals at each gate, a requirement in technologies like AQFP, where clocked operations depend on precise timing alignment across fanins.

Experimental results on benchmark circuits demonstrate that fanout-bounded synthesis methods can achieve significant area reductions.

1.4.3 Logic Synthesis for AQFP Technology

Adiabatic Quantum-Flux Parametron (AQFP) technology is a promising family of superconducting electronic circuits known for its exceptional energy efficiency. AQFP circuits operate with extremely low power consumption, making them highly suitable for applications where energy efficiency is paramount. However, AQFP technology imposes specific design constraints, including limited fanout capacity, clocked operations, and a need for synchronized signal paths. To address these constraints effectively, AQFP synthesis requires specialized optimization techniques that consider both the placement of logic gates and the arrangement of buffers and splitters necessary for clock synchronization. We propose an exact database-driven synthesis approach for AQFP circuits, leveraging precomputed optimal configurations that take into account both logic and interconnect resource requirements.

The exact database technique for AQFP synthesis consists of two primary stages. In the first stage, a database of minimum-area AQFP structures is generated through enumeration of all possible graph structures in a size-bounded manner. This process is performed only once and is independent of specific input networks, making it a reusable asset for optimizing multiple designs. The database contains precomputed structures for each unique 4-input Boolean function (up to some notion of equivalence), with optimized layouts for different input arrival-time patterns. By considering both area (measured in terms of Josephson Junctions) and delay (measured by the number of levels in the critical path), the database captures the best possible configurations for minimizing resource usage. This approach contrasts with traditional methods, which often treat buffer and splitter insertion as a post-synthesis step, thereby missing optimization opportunities that could be realized through simultaneous synthesis of logic and path-balancing resources.

In the second stage, the synthesis algorithm maps the input logic network to a Look-Up Table (LUT) representation and then replaces each LUT with an optimized structure from the exact database in topological order. This topological rewriting ensures that each logic block is replaced by its best equivalent from the database based on area or delay considerations. By integrating buffer and splitter insertion directly into the synthesis process, this method minimizes resource usage for both logic functions and necessary path-balancing components. The exact database technique also supports both majority-3 and majority-5 gates, leveraging AQFP's gate-level capabilities for more complex logic functionalities.

Experimental evaluations highlight the effectiveness of this database-driven synthesis approach. Compared to conventional AQFP synthesis methods, the database-based technique achieved significant improvements, reducing critical path delay by over 40% and area by approximately 21%. These results underscore the advantages of simultaneous optimization of logic and path-balancing resources for AQFP circuits in particular, and for similar post-CMOS technologies in general.

1.4.4 Logic Synthesis for FCN Technologies

Technology mapping for beyond-CMOS circuitry, such as FCN and similar emerging technologies, presents unique challenges that are not effectively addressed by conventional CMOS synthesis methods. In CMOS-based synthesis, technology mapping typically focuses on minimizing the number of gates or nodes in a logic network, assuming that reducing these elements will optimize area, delay, and power consumption. However, in beyond-CMOS technologies, additional design constraints emerge that can drastically affect layout quality. These constraints include the need for path balancing, signal synchronization, and the minimization of interconnect elements such as buffers, splitters, and crossings. Addressing these requirements calls for a technology mapping approach that optimizes for unconventional cost functions directly relevant to the characteristics of beyond-CMOS technologies.

In beyond-CMOS circuits, interconnects often carry a substantial portion of the layout costs, as they may involve specialized components like planarizing crossings and fanout-branching splitters. For example, in FCN, which encompasses technologies like Quantum-dot Cellular Automata (QCA) and Nanomagnetic Logic (NML), each gate or wire segment contributes uniformly to area and delay due to strict design grid layouts. Additionally, crossing cells, which enable wire paths to intersect without interference, can be costly and difficult to fabricate, necessitating an approach that minimizes their usage. Similarly, path-balancing buffers are required to ensure signal synchronization, as delays from imbalanced paths can lead to incorrect logic states. Therefore, technology mapping for beyond-CMOS circuitry must consider all these factors holistically to achieve an efficient layout.

The proposed technology mapping approach for beyond-CMOS circuitry with unconventional cost functions begins by generating a library of optimized subcircuits, which includes not only basic logic gates but also special cells like buffers, splitters, and crossings. These subcircuits are optimized according to the specific requirements of the target beyond-CMOS technology, ensuring that each cell configuration minimizes area, delay, and interconnect costs. The mapping process then uses this library to transform the logic network into a layout that is functionally equivalent to the original design while incorporating the necessary buffers, splitters, and crossings to meet design constraints. This mapping algorithm optimizes the arrangement of these cells, placing priority on reducing the critical path length and minimizing the number of costly interconnect elements.

Experimental evaluations of this mapping technique on benchmark circuits demonstrate significant improvements in layout quality compared to traditional mapping approaches. The proposed technology mapping algorithm achieves reductions in buffer count by over 84%, crossing count by 74.5%, and critical path length by 65.2% on average compared to state-of-the-art physical design algorithms adapted for FCN. These results underscore the effectiveness of incorporating unconventional cost functions into the mapping process, showing that layout quality in beyond-CMOS technologies can be greatly enhanced by directly addressing the specific challenges of interconnect management and path synchronization. This approach not only yields more efficient designs but also supports the broader adoption of emerging, energy-efficient computing paradigms by making them more practical and scalable for large-scale applications.

1.5 Thesis Organization

The remainder of this thesis is organized into six chapters, each addressing distinct aspects of the research and contributing to the overarching goals of scalable sequential logic synthesis and synthesis for emerging computing paradigms. Experimental results are discussed in Chapters 3 through 6 to validate the proposed methods.

Chapter 2: Background

This chapter provides the foundational concepts and terminology necessary for understanding the contributions of this thesis. Topics include Boolean algebra, logic representations, and optimization techniques. Additionally, the chapter gives an overview of emerging technologies, such as AQFP and FCN, which serve as the basis for post-CMOS synthesis approaches discussed later.

Chapter 3: Scalable Sequential Logic Synthesis

This chapter introduces the concept of *sequential observability don't cares* (SODCs), a powerful tool for optimizing sequential circuits by leveraging don't-care conditions that arise from the reachability of states. The chapter presents a novel synthesis method that utilizes SODCs to optimize sequential circuits while preserving correctness. Detailed proofs are provided, along with experimental results showcasing significant improvements in power, performance, and area (PPA) metrics compared to existing methods.

Chapter 4: Fanout-Bounded Logic Synthesis

This chapter addresses the challenges of fanout-bounded synthesis, which is crucial for technologies with tight fanout constraints, such as AQFP and FCN. The chapter formalizes the problem and presents both an exact Integer Linear Programming (ILP) formulation and heuristic approaches to optimize logic networks while adhering to fanout limits. The chapter also discusses adaptations of these methods for the path-balanced setting, where precise signal synchronization is required. Experimental results demonstrate the effectiveness of the proposed techniques in minimizing area and delay.

Chapter 5: Logic Synthesis for AQFP Technology

This chapter focuses on a two-stage exact-database-based synthesis method specifically designed for AQFP technology. The approach combines precomputed optimal configurations for small circuit blocks with an efficient mapping algorithm to produce highly optimized circuits. By integrating logic optimization with path-balancing resources such as splitters and buffers, this method achieves significant reductions in area and critical path delay. Experimental results highlight the advantages of this technique, showcasing improvements over traditional AQFP synthesis methods.

Chapter 6: Logic Synthesis for FCN Technologies

This chapter presents a technology mapping algorithm tailored to the unique constraints of Field-Coupled Nanocomputing (FCN) technologies. The method addresses key challenges such as minimizing the use of costly interconnect elements (e.g., buffers, crossings) and achieving path balancing. By leveraging a library of optimized subcircuits and applying advanced mapping techniques, the proposed algorithm delivers substantial improvements in layout quality and efficiency. Experimental results demonstrate reductions in interconnect costs and critical path lengths, validating the approach.

Chapter 7: Conclusion

The final chapter summarizes the key contributions of this thesis and discusses potential future research directions. It highlights the advancements made in scalable sequential logic synthesis, particularly through the introduction of SODC-based methods, and in synthesis for emerging computing paradigms, focusing on fanout-bounded and post-CMOS technologies. Suggestions for extending the work to broader applications and other emerging paradigms are also presented.

2 Background

This chapter introduces the key concepts, notations, and terminology necessary for a comprehensive understanding of this thesis. It also provides an overview of post-CMOS technologies, their technology-specific constraints, and the challenges they pose to traditional logic synthesis techniques.

2.1 Boolean Algebra and Functions

Boolean algebra is a branch of mathematics that deals with variables having two distinct values, typically denoted as `0` (false) and `1` (true). The origins of Boolean algebra can be traced back to the work of George Boole [29] in the mid-19th century, who developed a formal system for logical reasoning based on binary values. Boolean algebra forms the foundation of digital logic and binary systems, providing a rigorous mathematical framework for representing and manipulating logical expressions.

2.1.1 Boolean Algebra

Axiomatically, any set $\mathbb{B}$, binary operations $(\wedge, \vee)$, and a unary operation $(\neg)$ that satisfy the following axioms for any $a, b, c \in \mathbb{B}$ form a Boolean algebra:

Associativity:	$(a \vee b) \vee c = a \vee (b \vee c), \quad (a \wedge b) \wedge c = a \wedge (b \wedge c),$
Commutativity:	$a \vee b = b \vee a, \quad a \wedge b = b \wedge a,$
Absorption:	$a \vee (a \wedge b) = a, \quad a \wedge (a \vee b) = a,$
Identity:	$a \vee 0 = a, \quad a \wedge 1 = a,$
Distributivity:	$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c), \quad a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c),$

$$\text{Complement:} \qquad a \vee \neg a = 1, \quad a \wedge \neg a = 0.$$

In the context of digital logic, Boolean algebra typically refers to the two-value algebra with the set $\mathbb{B} = \{0, 1\}$, where the operators $\wedge$ (logical AND), $\vee$ (logical OR), and $\neg$ (logical NOT) are defined as follows:

$$\text{Logical AND:} \qquad 0 \wedge 0 = 0, \quad 0 \wedge 1 = 0, \quad 1 \wedge 0 = 0, \quad 1 \wedge 1 = 1,$$

$$\text{Logical OR:} \qquad 0 \vee 0 = 0, \quad 0 \vee 1 = 1, \quad 1 \vee 0 = 1, \quad 1 \vee 1 = 1,$$

$$\text{Logical NOT:} \qquad \neg 0 = 1, \quad \neg 1 = 0.$$

We often use the notation a' to denote the logical NOT of a, i.e., $a' = \neg a$. Similarly, $+$ and $\cdot$ are often used to denote logical OR and AND operations, respectively. For brevity, logical AND is frequently written without the operator symbol, e.g., $a \wedge b = ab$. For instance, the expression $(a \wedge b) \vee \neg c$ can equivalently be written as $ab + c'$.

Logical AND and OR operations can be generalized to multiple inputs:

$$\wedge(a_1, a_2, \ldots, a_n) = a_1 \wedge a_2 \wedge \ldots \wedge a_n, \quad \vee(a_1, a_2, \ldots, a_n) = a_1 \vee a_2 \vee \ldots \vee a_n.$$

The order in which the binary operators are applied does not matter due to the commutative and associative properties of Boolean algebra.

The logical XOR (exclusive OR) operation, denoted by $\oplus$, is defined as:

$$a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b) = ab' + a'b.$$

The XOR operation is also referred to as modulo-2 addition, as it is equivalent to addition modulo 2. It also is commutative and associative, i.e., $a \oplus b = b \oplus a$ and $a \oplus (b \oplus c) = (a \oplus b) \oplus c$, and can be extended to multiple inputs:

$$\oplus(a_1, a_2, \ldots, a_n) = a_1 \oplus a_2 \oplus \ldots \oplus a_n.$$

More details on Boolean functions and operations can be found in [39, 118].

2.1.2 Boolean Functions

Boolean functions are mathematical expressions that describe relationships between binary variables, mapping an n-tuple of binary inputs to a single binary output. Formally, a Boolean function f is defined as:

$$f : \{0, 1\}^n \rightarrow \{0, 1\},$$

where n represents the number of input variables. The function computes a binary output for every combination of its inputs.

Multi-output Boolean functions extend the concept of single-output Boolean functions by mapping binary input variables to multiple binary outputs. These functions are fundamental in digital design, as they naturally model combinational circuits with multiple outputs, such as decoders, arithmetic units, and control logic. A multi-output Boolean function f is defined as:

$$f : \{0,1\}^n \to \{0,1\}^m,$$

where n is the number of input variables and m is the number of output variables. The function computes m binary outputs for every combination of its n binary inputs. For example, if $n = 3$ and $m = 2$, the function $f(a, b, c)$ produces two outputs, f_1 and f_2, each defined as a single-output Boolean function:

$$f(a, b, c) = (f_1(a, b, c), f_2(a, b, c)).$$

The set of input variables of a Boolean function is called the *support* of the function.

2.1.3 Equivalence of Boolean Functions

Two Boolean functions defined on the same support are considered equivalent if they produce the same output for every possible input. However, there are other relaxed notions of equivalence commonly used in practice, defined based on three types of transformations:

Input Permutations: Two functions are considered equivalent under input permutations if one can be transformed into the other by permuting the input variables. For example, the functions $f(x, y) = \neg x \wedge y$ and $g(x, y) = x \wedge \neg y$ are equivalent under input permutations since $f(x, y) = g(y, x)$.

Input Negations: Two functions are considered equivalent under input negations if one can be transformed into the other by negating one or more input variables. For example, the functions $f(x, y) = x \wedge y$ and $g(x, y) = \neg x \wedge y$ are equivalent under input negations since $f(x, y) = g(\neg x, y)$.

Output Negation: Two functions are considered equivalent under output negation if one can be transformed into the other by negating their output. For example, the functions $f(x, y) = x \wedge y$ and $g(x, y) = \neg(x \wedge y)$ are equivalent under output negation since $f(x, y) = \neg g(x, y)$.

Two functions are considered NPN equivalent (short for Negation-Permutation-Negation) if one can be transformed into the other using a combination of input negations, input permutations, and output negation [23].

NPN equivalence is an equivalence relation, as it satisfies the properties of reflexivity (a function is NPN-equivalent to itself), symmetry (if f is NPN-equivalent to g, then g is NPN-equivalent to f), and transitivity (if f is NPN-equivalent to g and g is NPN-equivalent to h, then f is NPN-equivalent to h). Thus, all functions defined on the same support can be partitioned into NPN equivalence classes. There exists algorithms to efficiently enumerate NPN classes for Boolean functions [78, 132, 152].

NPN equivalence is a powerful concept in logic synthesis, as it reduces the number of distinct functions that need to be considered in many synthesis tasks. For instance, while there are $2^{2^4} = 65536$ different 4-input Boolean functions, there are only 222 distinct 4-input NPN classes. Often, synthesis databases are constructed by considering only one representative function from each NPN class for a fixed number of inputs (e.g., 4-input functions).

2.2 Logic Representations

A logic representation is a method used to describe a Boolean function in a structured and systematic manner. A given Boolean function can be represented in various forms, including truth tables, algebraic expressions, binary decision diagrams, and logic networks.

2.2.1 Truth Tables

A truth table is a tabular representation of a Boolean function, listing all possible input combinations and their corresponding output values. For example, consider the Boolean function $f(x, y, z) = x(y + z) + y'z$. Its truth table representation is as follows:

x	y	z	$f(x, y, z)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

By fixing the order of input variables, truth tables can be compactly represented as a binary or hexadecimal string, where the i-th least significant bit corresponds to the output value of the i-th row. For example, the truth table above can be encoded as `b'11100010` in binary or `0xe2` in hexadecimal.

Truth tables are straightforward and easy to understand, but their size grows exponentially with the number of input variables, making them impractical for functions with many inputs. A function with n Boolean inputs has 2^n different input combinations, requiring $O(2^n)$ space to store the truth table. For a multi-output function with n inputs and m outputs, the truth table contains m columns, each of which can be represented as a 2^n-bit binary/hexadecimal string.

2.2.2 Algebraic Expressions

Boolean functions can be expressed algebraically using logical AND, OR, and NOT operations. For example, the function $f(x, y, z) = x(y + z) + y'z$ discussed earlier is represented as an algebraic expression.

Boolean functions can often have multiple equivalent algebraic expressions. For example, the following algebraic expressions are equivalent representations of $f(x, y, z)$:

$$f(x,y,z) \;=\; x(y+z)+y'z \;=\; xy+z(x+y') \;=\; xy+z(x+y').$$

Boolean algebraic properties can be used to manipulate such expressions and derive their equivalent forms.

Due to multiple equivalent representations, it is not straightforward to compare two algebraic expressions for equivalence. This has led to the development of canonical forms, which provide a systematic and unique representation for any Boolean function.

2.2.3 Canonical Forms

A canonical algebraic form is a special case of algebraic expressions where the expression is written in a standard form. Two common canonical forms are the Product-of-Sums (POS) and Sum-of-Products (SOP) forms:

POS: Represents the function as a conjunction (AND) of maxterms, where each maxterm is a disjunction (OR) of literals (variables or their negations). For example, $f(x, y, z) = (x + y + z)(x' + y + z')(x' + y + z)(x + y' + z')$ is a POS representation.

SOP: Represents the function as a disjunction (OR) of minterms, where each minterm is a conjunction (AND) of literals. For example, $f(x, y, z) = xyz' + xy'z + xyz' +$

xyz is an SOP representation.

Canonical forms provide a unique representation for any Boolean function and are useful in certain logic synthesis techniques. For example, the logic minimization tool Espresso [36, 142] uses the SOP form to minimize Boolean functions. However, for complex functions with hundreds of inputs, canonical forms can still be impractical due to their exponential size.

2.2.4 Binary Decision Diagrams (BDDs)

A Binary Decision Diagram (BDD) [5, 96] is a directed acyclic graph (DAG) representation of a Boolean function, where each node represents a Boolean variable and each edge represents a variable assignment. The leaf nodes are labeled with the constants 0 or 1, representing the function's output. Ordered Binary Decision Diagrams (OBDDs) are a variant of BDDs that enforce a fixed variable order, ensuring a unique and canonical form for a given function with respect to the variable order. A special case of OBDDs is Reduced Ordered Binary Decision Diagrams (ROBDDs) [40], which eliminate redundant nodes and merge identical subgraphs to further reduce the size of the BDD.

A BDD can be constructed using the so-called Shannon decomposition rule:

$$f(x_1, x_2, \ldots, x_n) = x_1 f(1, x_2, \ldots, x_n) + x_1' f(0, x_2, \ldots, x_n),$$

which is used to recursively split the function based on the value of a variable.

While the worst-case complexity of BDDs is still exponential in the number of variables, they can provide compact representations for many practical functions, as many real-world functions exhibit shared structures that BDDs can exploit. BDDs are used in logic synthesis tools such as CUDD [155] and ABC [34] for efficient manipulation of Boolean functions in various optimization and verification tasks.

2.2.5 Logic Networks

A logic network is a multi-level representation of a Boolean function. It represents the Boolean function as a network of interconnected logic gates, where each gate performs a specific Boolean operation.

Logic networks are highly versatile representations. In general, a gate in a logic network can correspond to any arbitrary logic function. For example, a technology-mapped circuit may be represented as a logic network where nodes represent standard cells from the target technology library, and edges represent connections between cells. In technology-independent logic synthesis, however, synthesis tools typically use network representations where the gates correspond to simple logic functions such as AND, OR, and NOT gates.

Mathematically, a logic network can be represented as a directed acyclic graph (DAG), where nodes represent logic gates and edges represent connections between gates. Logic networks sometimes omit explicit NOT gates; instead, the edges are annotated with a binary flag denoting the presence of an implicit inverter along the edge.

The terminal nodes with no incoming edges (sources) represent the primary inputs (PIs), and the terminal nodes with no outgoing edges (sinks) represent the primary outputs (POs). A logic network may also represent a sequential digital circuit, where sources may represent register outputs (ROs) and sinks may represent register inputs (RIs). The corresponding RI/RO pairs are usually stored in a separate data structure along with the respective initial values of the registers.

The *fanins* of a node n refer to the set of nodes that drive n, i.e., the nodes that have directed edges to n. Similarly, the *fanouts* of a node n refer to the set of nodes driven by n, i.e., the nodes that have directed edges from n. The *transitive fanin* (TFI) cone of a node n is the set of all nodes from which n is reachable via a directed path. Analogously, the *transitive fanout* (TFO) cone is the set of all nodes reachable from n via a directed path.

Two of the most common types of logic networks used in logic synthesis are *And-Inverter Graphs* (AIGs) and *Majority-Inverter Graphs* (MIGs). These are homogeneous networks, meaning that all nodes in the network are of the same type (up to possible fan-in inversions denoted by edge annotations). Both AIGs and MIGs are universal logic representations, capable of representing any arbitrary Boolean function.

2.2.6 And-Inverter Graphs (AIGs)

And-Inverter Graphs (AIGs) [69, 92] are logic networks where internal (i.e., non-terminal) nodes represent AND gates. Typically, 2-input AND gates are used as the basic building block in AIGs, ensuring that each internal node has an in-degree of two. Inverters are usually represented through edge annotations.

Given that the basic building block of CMOS technology is the 2-input NAND gate—which can be mapped to a 2-input AND gate followed by an output inverter—AIGs are a natural choice for representing logic circuits in the context of CMOS technology. AIGs have been widely used in logic synthesis tools such as ABC [34] and libraries such as `mockturtle` [154], owing to their simplicity and compatibility with many logic synthesis algorithms.

AIGs support efficient *structural hashing*, a technique that uniquely identifies gates by their fanin configuration, thereby allowing the identification and collapsing of logically equivalent nodes.

2.2.7 Majority-Inverter Graphs (MIGs)

The k-input Boolean majority gate outputs 1 if and only if more than $k/2$ of its inputs are 1. While majority-based logic synthesis has been studied since the 1960s [6, 7, 119], it has recently been re-introduced as a new paradigm for logic synthesis [8, 9].

Majority-Inverter Graphs (MIGs) are defined similarly to AIGs, with the key difference being that the internal nodes represent 3-input majority gates. Thus, each internal node has an in-degree of three.

Note that when one input of a majority gate is tied to a constant 0 or 1, the gate behaves as a 2-input AND or OR gate, respectively. Thus, MIGs are a generalization of AIGs. Similar to AIGs, MIGs also support efficient structural hashing.

MIGs are supported by a sound and complete set of algebraic rules, known as the majority algebra, which enables efficient manipulation and optimization of MIGs [13]. Using the notation $\langle x_1, x_2, x_3 \rangle$ to denote a majority gate with inputs x_1, x_2, x_3, the majority algebra rules are as follows:

$$\begin{aligned}
&\text{Commutativity:} && \langle x_1, x_2, x_3 \rangle = \langle x_2, x_3, x_1 \rangle = \langle x_3, x_1, x_2 \rangle,\\
&\text{Associativity:} && \langle x_1, \langle x_2, x_3, x_4 \rangle, x_5 \rangle = \langle x_5, \langle x_1, x_2, x_3 \rangle, x_4 \rangle,\\
&\text{Distributivity:} && \langle x_1, x_2, \langle x_3, x_4, x_5 \rangle \rangle = \langle \langle x_1, x_2, x_3 \rangle, x_4, \langle x_1, x_2, x_5 \rangle \rangle,\\
&\text{Majority:} && \langle x, x, y \rangle = x, \quad \langle x, \bar{x}, y \rangle = y,\\
&\text{Inverter Propagation:} && \langle \bar{x}, \bar{y}, \bar{z} \rangle = \overline{\langle x, y, z \rangle}.
\end{aligned}$$

MIGs are particularly useful in superconducting technologies (see Section 2.4 for more details), where majority gates are the natural gate type. They are one of the fundamental logic network types supported in the logic synthesis library `mockturtle` [154].

Other homogeneous networks using different types of 3-input gates have also been studied for their representative power [109]. In addition to AIGs and MIGs, their heterogeneous counterparts with additional XOR (XAGs and XMGs [59, 66]) or other operators such as multiplexers have also been utilized in various contexts [15].

2.3 Logic Optimization

Logic optimization is the process of transforming a given Boolean function into an equivalent function that satisfies certain criteria, such as minimizing the number of gates (e.g., in a network representation) or reducing the number of literals [37] (e.g., in an algebraic expression).

Optimizations consist of heuristic-based methods and exact algorithms. Heuristic-based methods often involve algebraic and Boolean manipulations to simplify the function, while exact algorithms rely on formal methods such as Boolean satisfiability (SAT) [26, 88] solvers.

In algebraic optimizations of logic networks, the outputs of internal nodes are expressed as polynomials of the primary inputs (or with respect to some intermediate nodes). Algebraic optimization [35, 37, 118] methods include:

Extraction:	Identifying and factoring out common subexpressions.
Substitution:	Replacing parts of an expression with equivalent expressions derived from another node in the network.
Decomposition:	Breaking down a node's polynomial into smaller components.
Rewriting:	Transforming a node's expression into a different form.

For Majority-Inverter Graphs (MIGs), rewriting can be performed based on the majority algebra rules.

In Boolean optimizations, similar to algebraic optimizations, substitutions and rewriting are performed while also considering the *Don't-Care* conditions [30, 45, 56, 118, 120, 133]. A don't-care condition for an internal wire (or node) is an input combination under which the output of the wire (or node) is not significant. This could occur either because the input combination never arises in practice or because the output is never observed at any primary output under that input combination.

Another type of Boolean optimization is redundancy removal under don't-care conditions. Nodes that are stuck at a constant value for all non-don't-care inputs can be removed after propagating the stuck-at constant to the outputs.

Exact methods aim to find the optimal solution to a given optimization problem. While there are a few cases where efficient (polynomial-time) exact algorithms exist (e.g., minimum delay LUT mapping [52]), most optimization problems are NP-hard [53]. Consequently, these problems are often solved using SAT or Integer Linear Programming (ILP) solvers after encoding the problem appropriately [151]. Alternatively, explicit exhaustive enumeration can be used to systematically search through all possible solutions to find the optimal one [65].

An important exact synthesis problem in logic synthesis is to synthesize the minimum area circuit for a given Boolean function, while meeting a given set of constraints on the input arrival times. Prior work on exact synthesis includes [14, 57, 67, 87, 89, 141] and employs various approaches such as decomposition-based, SAT-based, and explicit enumeration-based methods.

More details on logic optimization can be found in [68, 118].

2.4 Post-CMOS / Emerging Technologies

The continuous scaling of CMOS technology is approaching its fundamental limits, driving research into alternative technologies to sustain the advancement of digital circuits. These post-CMOS emerging technologies offer potential advantages in terms of performance, power consumption, and density, while introducing unique challenges for logic synthesis. Several promising families of post-CMOS technologies are being actively researched and developed, including superconducting electronics [73], field-coupled nanotechnologies [16], and photonic crystals [82, 179].

As these alternative technologies are gaining momentum, logic synthesis techniques are being adapted to address the challenges posed by these technologies [12, 20, 71, 138]. Our work primarily focuses on superconducting electronics and field-coupled nanotechnologies.

2.4.1 Superconducting Electronics (SCE)

Superconducting Electronics (SCE) utilize superconducting materials to achieve ultra-low power dissipation and high operating speeds. SCE circuits operate at cryogenic temperatures, where the resistance of superconducting interconnects becomes zero. SCE logic gates are typically based on Josephson junctions, which exploit the Josephson effect [72]. This effect exhibits non-linear current-voltage characteristics and is used to implement various logic functions. Candidate technologies in the SCE domain include Rapid Single Flux Quantum (RSFQ) [104], Reciprocal Quantum Logic (RQL), and Adiabatic Quantum-Flux-Parametron (AQFP) [157]. Some of these technologies have gained sufficient maturity so that medium-sized circuits have been realized [10, 17, 20, 127]. In many of these technologies, the fundamental logic gate is the 3-input majority gate, while some technologies, such as AQFP, also support higher-fanin majority gates.

The AQFP technology has recently gained attention due to its ultra-low power consumption and adiabatic operation. Namely, it achieves superior energy efficiency (uses two orders of magnitude less energy compared to semiconductor technologies even after cooling energy is taken into consideration [51]) using AC-biased Josephson junctions, whereas most other SCE technologies use DC-biased junctions, which cause static power dissipation. Since AQFP is one of the main post-CMOS technologies we focus on in this work, we provide a brief overview of its key characteristics.

Adiabatic Quantum-Flux-Parametron (AQFP)

In AQFP technology, logic gates are constructed using superconductive inductors and *Josephson Junctions* (JJs). The number of JJs in an AQFP circuit is commonly used as a proxy for area cost.

Takeuchi et al. [158] proposed a simple cell library for AQFP, consisting of four primitive cells: buffer, inverter, constant, and branch. Gates are created using arrays of primitive cells combined with branches, while splitters are constructed using buffers and branches. The majority-3 gate comprises three buffer cells and a branch. Input-inverted versions of majority-3 gates are constructed by substituting buffer cells with inverter cells [158]. As such, explicit inverter cells are typically not needed except at the primary outputs if an inverted gate output is required. Similarly, 2-input AND and OR gates are derived by substituting a buffer cell with a constant 0 or 1 cell. Each primitive cell—buffer, inverter, and constant—uses two JJs, meaning a splitter also uses 2 JJs. Thus, all gates, including majority-3, AND-2, and OR-2, as well as their input-inverted versions, use 6 JJs each.

In AQFP logic, the majority-3 gate is the elementary gate, as AND and OR gates are derived from it, all having the same area. Consequently, Cai et al. [41] proposed majority-gate-based logic synthesis as a suitable optimization strategy for AQFP technology.

AQFP gates have weak output signals that cannot directly drive multiple fanouts. Instead, splitters (or trees of splitters) must be used to boost the output signal for multiple fanouts. Depending on the implementation, splitters can drive three or four fanouts [42, 158]. Our synthesis experiments assume a branching capacity of four.

As with many superconducting technologies, AQFP gates are clocked, and logic values propagate between consecutive gates when their active periods overlap. To ensure overlap, all fanins of a gate n are clocked in the same phase, and n itself is clocked in the next available phase (e.g., for a 4-phase clocking scheme, if fanins are activated by phase ϕ, n is activated by $\phi + \pi/4$). To achieve this:

1. Ensure all fanins of a gate are in the same logic level.
2. Map consecutive logic levels to consecutive rows of gates/buffers in the physical circuit.
3. Activate consecutive rows of gates with clock signals in consecutive phases.

We note that, in general, it is not mandatory for all fanins to be exactly in the same logic level; it is sufficient for them to be in the same logic level modulo the number of clock phases. Even this requirement can be eliminated by adopting a more elaborate clocking scheme, where non-consecutive clock phases can also overlap [144]. However, such approaches are not yet widely adopted in practice, and most existing works on

AQFP focus on the setting where all fanins of a gate must be in the same logic level. This thesis adheres to this convention as well.

The design of registers and the clocking mechanism can introduce varying requirements, such as the need for splitters at primary inputs, path-balancing for primary inputs, and path-balancing for primary outputs [143].

2.4.2 Field-Coupled Nanotechnologies (FCN)

Field-Coupled Nanotechnologies (FCN) encompass a range of nanotechnologies, including Quantum-Dot Cellular Automata (QCA) [101, 102], Nanomagnet Logic (NML) [24, 54], and Silicon Dangling Bonds (SiDB) [79, 135, 175]. In these technologies, information is encoded using the polarization or magnetization of nanoscale building blocks called cells. When placed in close proximity, these cells influence each other's polarization or magnetization through Coulomb interactions, enabling information transmission via electric or magnetic coupling without the actual flow of charge. This allows for the design of ultra-dense, low-power circuits.

As with superconducting technologies, FCN technologies exhibit tight fanout constraints and path-balancing requirements. In addition, FCN technologies often require *planarization*, where crossing wires are prohibited. Instead, special crossing cells must be used to handle signal path intersections. Making a circuit compatible with this constraint by properly placing crossing cells is referred to as planarization. This process often lengthens signal paths, increasing buffer overhead due to path-balancing constraints. Consequently, in these technologies, interconnect costs (crossings and buffers) far exceed gate costs [148].

For more details, we delve deeper into FCN technologies in Section 6.2.1 of Chapter 6, focusing primarily on logic synthesis targeting these technologies.

3 Scalable Sequential Logic Synthesis

As the scaling of CMOS technology approaches fundamental physical limits, optimizing the power, performance, and area (PPA) of digital circuits has become increasingly critical. While significant progress has been made in combinational logic synthesis, the potential of sequential logic optimizations remains underexplored. This chapter delves into sequential optimization, proposing novel optimization methods to bridge gaps in conventional synthesis methodologies and unlock new PPA opportunities.

This chapter is based on the work [113] published in the Design, Automation, and Test in Europe Conference (DATE) 2024. An extended version of the work has been submitted to the journal IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems [112].

3.1 Introduction

Logic synthesis optimizes logic networks under various metrics, such as area, power, and delay. It plays a crucial role in modern *Electronic Design Automation* (EDA) flows, and can be broadly categorized into two types: combinational and sequential logic synthesis.

Combinational logic synthesis focuses on optimizing logic networks while maintaining combinational equivalence. Even if a logic network has sequential elements, combinational logic synthesis can still be applied by treating the register inputs/outputs as primary outputs/inputs, ignoring any constraints on reachable states.

Sequential logic synthesis, in contrast, specifically targets the optimization of logic networks with sequential elements. Since it can account for the fact that not all register value combinations are reachable, it offers a more powerful form of logic synthesis. It is well known that sequential logic synthesis explores a broader solution space and generally achieves better *power, performance, and area* (PPA)[125]. These PPA benefits become increasingly critical as the cost of chip design continues to rise [3].

Sequential logic synthesis has been studied in the past considering various approaches [32, 47, 48, 58, 90, 99, 125, 146, 149]. One such approach is to integrate combinational optimizations together with retiming [81, 99], which can exploit optimization opportunities arising due to structural properties across register boundaries [32]. The key idea behind this line of work is to move registers across combinational logic while optimizing the resulting combinational logic segments with existing combinational optimization techniques. A powerful yet scalable state-of-the-art approach is given by the *sequential SAT-sweeping* (SSW) algorithm from [125]. The basic idea of SSW is to merge sequentially equivalent nodes, where it uses *Bounded Model Checking* (BMC), Boolean Satisfiability (SAT), and sequential induction [27, 128, 168] to prove the validity of such merge candidates.

In this chapter, we present a novel scalable algorithm for don't cares-based sequential logic synthesis. Our approach, by design, can work with *dependencies* among *observability don't cares* (ODCs), which is a challenging problem that has not been addressed in prior approaches. Our method is based on sequential induction and is orthogonal to the approach presented in [125]. In fact, our new method integrates both redundancy removal and resubstitution, leveraging *sequential observability don't cares* (SODCs). Moreover, our method can be enhanced to work with multistep induction and the use of assumptions to detect additional optimization opportunities. The latter is a version of sequential induction where the validity of candidate logic transformations in the inductive case are verified *assuming* they are already present in prior frames.

In combinational logic synthesis, ODCs—i.e., input patterns where the value of a wire is not observed at the outputs—can be utilized to uncover better optimization opportunities [55, 98, 120, 126, 162, 180]. However, employing ODCs for optimization poses inherent challenges, as the remaining ODCs can change after applying an ODC-based optimization. These challenges are even more pronounced in sequential optimizations, where it is particularly difficult to account for the sequential nature of circuits while managing ODC dependencies. (It is worth noting that applying an ODC-based optimization can also alter the set of reachable states.) Although the dependency issues can be mitigated by restricting the analysis to *compatible ODCs* (CODCs)[145], i.e., ODCs that can be used independently at each node, this approach can result in missed optimization opportunities.

Nevertheless, the method we propose inherently handles ODC dependencies and hence utilizes the full power of ODCs in sequential optimizations. This is achieved using an inductive approach that takes the reachable states into account and performs simultaneous, in-place optimization of two networks (base case and inductive case) using ODC-based combinational optimization methods that are built on *Boolean Satisfiability* (SAT) [115]. To make it scalable, our approach uses windowing so that the SAT-problem sizes remain manageable.

The simultaneous optimization of the derived networks enables our method to naturally identify valid ODC-based sequential optimizations that are compatible with one another, without restricting the search space to CODC-based optimizations. As a result, it achieves superior optimizations, particularly in circuits with sequential feedback, which can pose challenges for traditional retiming-based techniques. In essence, our method uncovers sequential optimizations that were previously unexplored by other approaches, while maintaining scalability. Additionally, since our approach avoids moving registers over combinational logic, the verification of the optimized networks is more likely to succeed, making it more suitable for use in industrial tools. In contrast, retiming-based methods often present challenges for verification tools due to the potential loss of anchor points essential for verification.

Through optimization of technology independent logic, we have demonstrated that each of the extensions improves the quality of results compared to the base version. Moreover, our approach is shown to achieve a 6.9% average reduction in area after technology mapping on top of state-of-the-art sequential optimization methods (e.g., SSW), with a 2.89% reduction of combinational area and a 1.43% of sequential area post place & route on industrial designs. All designs were verified using state-of-the-art sequential verification tools [164].

The organization of the chapter is as follows: Section 3.2 discusses some background and relevant prior work. Section 3.3 presents a motivating example for sequential synthesis with ODCs followed by our novel scalable sequential logic synthesis approach, together with the proofs of correctness for different versions of the algorithm. Section 3.4 presents our experimental results, and finally, Section 3.5 concludes with a brief discussion of the results and future work.

3.2 Preliminaries

In this section, we provide some background that will be useful to better understand the rest of the chapter and briefly describe some state-of-the-art prior work in sequential synthesis.

3.2.1 Sequential Logic Optimizations

In this section, we briefly introduce two important concepts that we use in our proposed optimization approach: *sequential redundancy* and *sequential resubstitution.*

Sequential Redundancy

In logic synthesis, a redundancy is a node or a wire whose value is stuck at a constant in all observable input (PI/RO) patterns. A redundant wire can be optimized away by removing the origin node of the wire from the fanin set of the destination node and modifying the destination node's function accordingly. A node is redundant if all its outgoing wires are redundant. Sequential redundancy is a generalization of a redundancy where the stuck-at-constant property holds considering all *observable input patterns* and *reachable states*.

Sequential Resubstitution

In logic synthesis, resubstitution refers to replacing a node n with a different node $m \neq n$. In general, m can be any existing node that is not in the TFO cone of n, or it can be a new node constructed by combining several other non-TFO nodes (called divisors). Boolean resubstitution refers to equivalence-preserving resubstitutions that are computed considering Boolean properties such as *don't cares* (DCs). Additionally, when the implicit restrictions on reachable states of a sequential logic network are considered during resubstitution, we refer to it as sequential resubstitution.

3.2.2 Don't Cares in Logic Networks

In logic synthesis, a *don't care* (DC) is an input pattern (i.e., similar to a minterm) for which the output value of a node is not important [30]. Such patterns can be specified either with respect to the primary inputs or any cut of the node. Note that the latter is a generalization of the former since the set of PIs is a valid cut for any node. Don't cares have been extensively used in logic optimizations [22, 45, 140].

There are different types of don't cares such as *Controllability Don't Care* (CDC) and *Observability Don't Care* (ODC), which are described below. Note that, in addition to CDC and ODCs, there is also the notion of *Satisfiability Don't Care* (SDC)[118] which are Boolean value combinations that never occur considering an internal wire. SDCs are primarily used in the exhaustive computation of ODCs and CDCs.

Controllability Don't Cares

The CDCs for a node n with respect to a cut I are the Boolean value combinations for I which are impossible to occur. When the considered cut contains some internal nodes, CDCs can occur due to the structure of the network. For example, if a node n as two inputs x and y where $x = a \wedge b$ and $y = a \vee b$, the value combination $x = 1$ and $y = 0$ can never occur. Thus, considering the cut $\{x, y\}$, the value combination $x \wedge \neg y$ is a

CDC for n. (Consequently, if n computes $x \wedge \neg y$, a CDC-based optimization algorithm might optimized it away and replace it with the constant 0.) A logic network might also have some impossible PI patterns due to external constraints,, and such patterns are called *external CDCs*.

Observability Don't Cares

The ODCs for a node n with respect to a cut I are the Boolean value combinations for I for which the output of n is not observed at any PO. For example, in Fig. 3.1 (a), consider the node w_1 with respect to the cut $\{a, b, d\}$. For any pattern where $a = 1, b = 0$, w_1 is not observed at the output o_1 because the output gate's second fanin, g_2, will be 0 under such a pattern. Thus $a \wedge \neg b$ is an ODC for w_1.

The ODCs of nodes in a logic network can have dependencies among themselves. Namely, if an optimization with respect to an ODC is performed for a particular node, it can change the ODCs of other downstream nodes, and hence, ODCs will have to be recomputed for those nodes. This added complexity can be avoid by using a less powerful version of ODCs, called *Compatible ODCs* (CODCs) [33, 145], that do not have dependencies among themselves.

3.2.3 Prior Work on Sequential Synthesis

A common optimization approach in early works on sequential synthesis is to use retiming together with logic transformations [32, 58]. In this approach, first, the registers are moved around, then the resulting circuit is optimized using combinational methods, and finally retiming is performed again to minimize the register count. During retiming, if some reconvergent paths have varying numbers of registers, the usual practice is to remove such paths by duplicating the shared nodes, considering small blocks of logic. In contrast, our SODC-based approach does not move registers, thus it avoids the duplication requirement of shared logic. Moreover, our approach scales well to much larger logic blocks.

Another prominent sequential optimization method is *sequential SAT-sweeping (SSW)*, which is a generalization of combinational SAT-sweeping [92, 121] to the sequential setting, where the idea is to merge sequentially-equivalent nodes. If two nodes m and n are equivalent under all observable input patterns of n in all reachable states, n can be merged with m by transferring the fanouts of n to m, without changing the overall output function of the network. An efficient SSW algorithm is proposed in [125] where the sequential equivalences among nodes are proven using *bounded model checking* (BMC) [25] and SAT together with induction [27, 128, 168]. Once the equivalence classes are identified, all nodes in a class are merged into a chosen representative node and the dangling nodes are removed. Despite its practical success, SSW misses many optimizations made possible due to SODCs. Notably, SSW cannot optimize the simple sequential logic network in Fig. 3.2(a) into the one in Fig. 3.2(b).

Additionally, Case et al. [48] considered a simulation-based approach to find merge candidates considering ODCs. Namely, the network is simulated with random bit patterns to identify node pairs a, b such that for each simulated pattern, either a and b are equal or all paths from b to combinational outputs are non-controlling. Then a new network is created with all candidates merged, the equivalence of the new and original network is proven/disproven using SAT, and if disproved, the merge candidates are refined. However, this approach does not scale well to large networks due to large miters used in equivalence checking and hence misses many optimization opportunities. In contrast, we use a window-based approach and check the validity of each optimization in isolation; hence the SAT-based validity checks are scalable.

The method we propose in the next section is able to find optimizations that were never found by the prior approaches.

3.3 Scalable Sequential Optimization

In this section, we first give a brief motivation for our proposed method and introduce sequential induction. Then, we discuss our novel sequential optimization approach in detail with a formal proof of correctness. Lastly, we analyze some of the limitations of the proposed method and possible workarounds.

3.3.1 Motivation

Consider the purely combinational logic network shown in Fig. 3.1(a) and observe that the wires g_1 and g_2 can never be 1 at the same time. This implies that whenever $g_2 = 1$, g_1 must be 0. Since w_2 is observed at output o_1 only when $g_2 = 1$, one can simplify the circuit by assuming that g_1 is stuck at 0, which, in turn, implies that w_1 is also stuck at 0. This leads to the optimized circuit in Fig. 3.1(b).

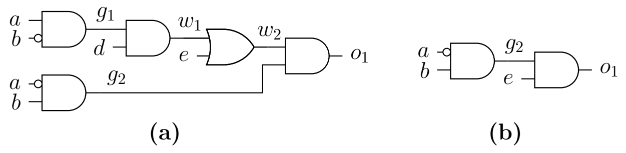


Figure 3.1: A combinational logic network (a) and its optimized version (b).

Now consider the sequential circuit in Fig. 3.2(a) which is similar to the one in Fig. 3.1(a) except for the two registers at g_1 and g_2. If we consider this as a combinational network (i.e., disregard the registers, consider g_1, g_2 to be POs, and consider lo_1, lo_2 to be PIs), the previous reasoning no longer applies; lo_1, lo_2 can take arbitrary values, and hence lo_1

is observed even when it is 1. However, if we additionally know that the initial values of lo_1, lo_2 are $(0,0)$ (or any combination of values different from $(1,1)$), the optimization is still possible. This is because, by design, lo_1 and lo_2 can never be 1 at the same time in the subsequent clock cycles. This observation yields the optimized circuit shown in Fig. 3.2(b). The state-of-the-art sequential optimization routines such as *scorr*, *lcorr*, *scl*, and *retime* of the logic synthesis tool ABC [34] are unable to find this optimization.

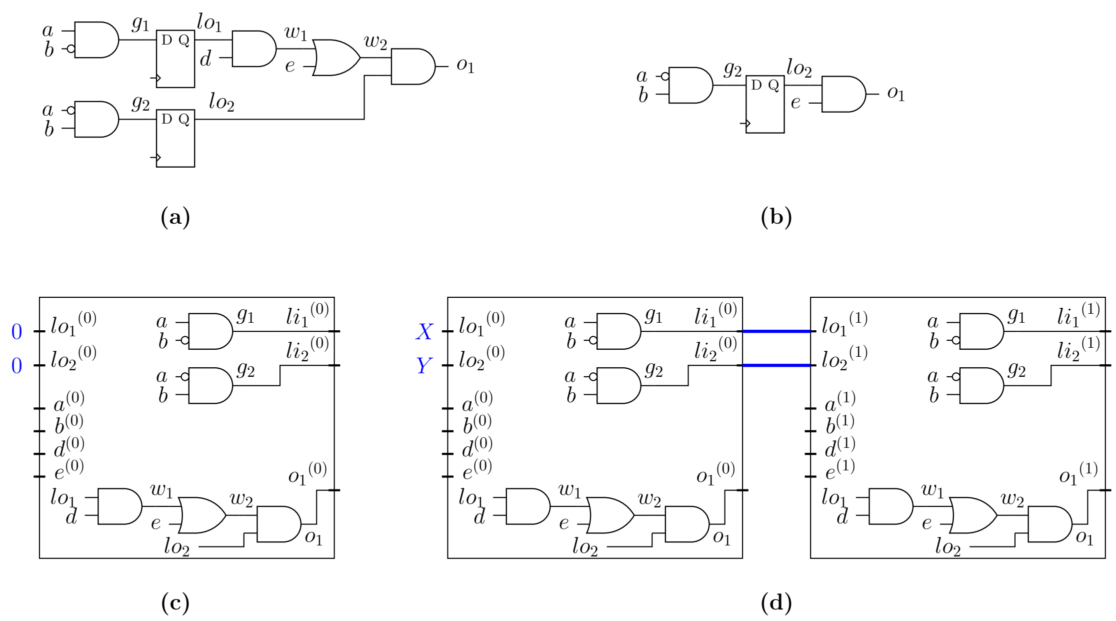


Figure 3.2: A sequential logic network (a), its optimized version (b), and its base case network (c) and the inductive case network (d) for 1-step sequential induction.

The goal of the proposed method is to identify this kind of optimization opportunities in sequential logic networks in a scalable way. We remark that, while a retiming-based optimization method might be able to optimize the example above, such methods perform poorly especially when there is sequential feedback (e.g., finite state machines) or varying numbers of registers along different reconvergent logic paths.

3.3.2 Sequential ODCs

The optimization in the example holds due to two facts:

1. (*Reachability*) Not all states (value combinations for the sequential elements) are reachable.
2. (*Observability*) In all reachable states, the optimization is valid due to the ODCs.

These two facts together form a notion of *sequential ODCs* (SODCs), a generalization of ODCs in combinational logic networks into the sequential setting.

Recall that a sequential network can be optimized by considering it as a combinational network, using combinational synthesis algorithms, where the register inputs are considered as primary outputs and the register outputs are considered as primary inputs. In this setting, suppose that the complete set of unreachable states are somehow known beforehand. Then, we can input such unreachable states as external CDCs, and use an external-CDC-based synthesis algorithm to optimize the network, and this would effectively perform sequential synthesis. The ODCs that exists when such unreachable states are considered as external CDCs are called SODCs.

In practice, the set of unreachable states is not known beforehand; but are implicitly defined by the structure of the network and the initial states. Thus, to consider SODCs in optimizations, an algorithm has to somehow reason about the reachable/unreachable states, and completely characterizing the set of SODCs is a computationally hard problem. In what follows, we present a framework that can be used to approximate SODCs of sequential networks.

3.3.3 Framework Definition

To use SODCs in optimization, we first take the *reachability* of states into account. To this end, a widely used technique is to use the so-called sequential induction [27, 168] which leverages two combinational networks called the base case network and the inductive case network that are obtained using k-step unrolling as defined below in Definition 1.

Definition 1 (k-Step Unrolling)**.** *For a sequential logic network N, the k-step base case network N^b is the combinational network obtained by*

1. *taking k copies of N (referred to as frames),*
2. *connecting the RIs of each frame to the corresponding ROs of the subsequent frame,*
3. *replacing the ROs of the first frame with the respective register initial values, and*
4. *designating RIs of each frame as POs.*

The k-step inductive case network N^i is similarly defined except with the following changes:

1. *it has $k+1$ frames, and*
2. *the ROs of the first frame are designated as PIs.*

For the example network of Fig. 3.2(a), the base case and the inductive case networks for 1-step (i.e., for $k = 1$) sequential induction are shown in Fig. 3.2(c) and Fig. 3.2(d) respectively. Note that in all figures, wires between frame inputs and gates are implicit, i.e., a frame input $x^{(i)}$ is connected to gate pins denoted by x. The behavior of N^b is the same as that of the original network N for the initial k clock cycles and the behavior of N^i is the same as $k+1$ consecutive clock cycles of N for any initial state. As formally stated in Theorem 1, if an optimization is valid in all frames of the base case and the last frame of the inductive case, then it is a valid sequential optimization for the original network.

On top of the reachability criterion, we consider the *observability* to identify sequential optimization opportunities. It seems straightforward to consider the sets of ODC-based optimizations in the base case and inductive networks and then take the intersection of the two sets as the final set of optimizations. However, as discussed in Section 3.1, this approach only works with CODCs which do not have dependencies among them. Unfortunately, using CODCs in place of ODCs leads to many missed optimization opportunities. The regular ODCs can have dependencies in them, and cause this simple algorithm to fail. In the remainder of this section, we present an algorithm that, by design, avoids dependency issues of regular ODCs without falling back to CODCs.

3.3.4 Proposed Method

Our proposed algorithm is based on sequential induction and it can fully utilize ODCs by simultaneously optimizing base case and inductive case networks.

Namely, we start by constructing the 1-step unrolled base case and inductive case networks for sequential induction. Then, considering one node at a time, we check if there is a valid optimization for that node in both the base case and the inductive case networks. If so, we immediately update both the derived networks as well as the original network by applying the optimization. This approach allows the algorithm to find subsequent optimizations for the remaining nodes that may depend on the already applied optimizations. Thus it avoids any dependency issues that would arise if we were to use the simple approach we stated at the end of Section 3.3.3 with regular ODCs. Hence, the algorithm computes compatible sequential optimizations without limiting to CODCs.

To find optimizations, the algorithm considers fanin redundancies for each gate n in the network. Namely, for each fanin f of n we check whether f is effectively stuck at 0 or 1 in

1. the the base case network, and
2. the last frame of the inductive case network.

Since both the base case and inductive case networks are purely combinational, it is possible to use any combinational redundancy check for this purpose. To this end, let n' be the node obtained by fixing fanin f of n at the target constant value. In our implementation, we check if replacing n with n' is valid using a SAT problem. If the problem is unsatisfiable (UNSAT), then the optimization is valid. To make the overall algorithm scalable, we optimize the SAT formulation not to consider all POs and RIs, but instead consider the leaf nodes of a small TFO cone rooted at n. If the optimization is shown to be valid for both the base and inductive case networks, then we apply it in both the networks as well as in the original network (see Section 3.3.5 for details).

As an illustrative example, consider Fig. 3.2(a). We can prove, that w_1 is stuck at 0, in both the base case and the inductive case networks, which will result in the optimized network in Fig. 3.2(b) (assuming all registers are initially 0).

We consider three enhancements on our proposed method which enables it to find more optimization opportunities.

Enhancement 1

We extend our algorithm to use k-step sequential induction where the base case network has $k \geq 1$ frames and the inductive case network has $k + 1$ frames. In this case, we check if the target Δ redundancy is valid in

1. all k frames of the base case network, and
2. the last frame of the inductive case network.

If the considered redundancy is valid in both cases, then we apply it in all frames of the two derived networks as well as in the original network.

Fig. 3.3(a) shows an example sequential network which can be optimized to the one in Fig. 3.3(b) with 2-step sequential induction (assuming all registers are initially 0). In the last frame of the inductive network (Fig. 3.3(c)), lo_3, lo_4 are fed by the gates g_1, g_2 of the first frame, so lo_3, lo_4 of the last frame are never 1 at the same time. Thus the algorithm is able to prove that w_1 of the last frame is stuck at zero. In contrast, if 1-step induction were to be used, we only get the first two frames of Fig. 3.3(c), and the second frame's lo_3, lo_4 are driven by two arbitrary inputs from the first frame. Hence, all value combinations are possible, so w_1 of the second frame is not stuck at zero.

Enhancement 2

Our approach is not limited to fanin redundancies but also extends to resubstitutions under ODCs. Namely, for a considered node n, we consider a subset D (called divisors)

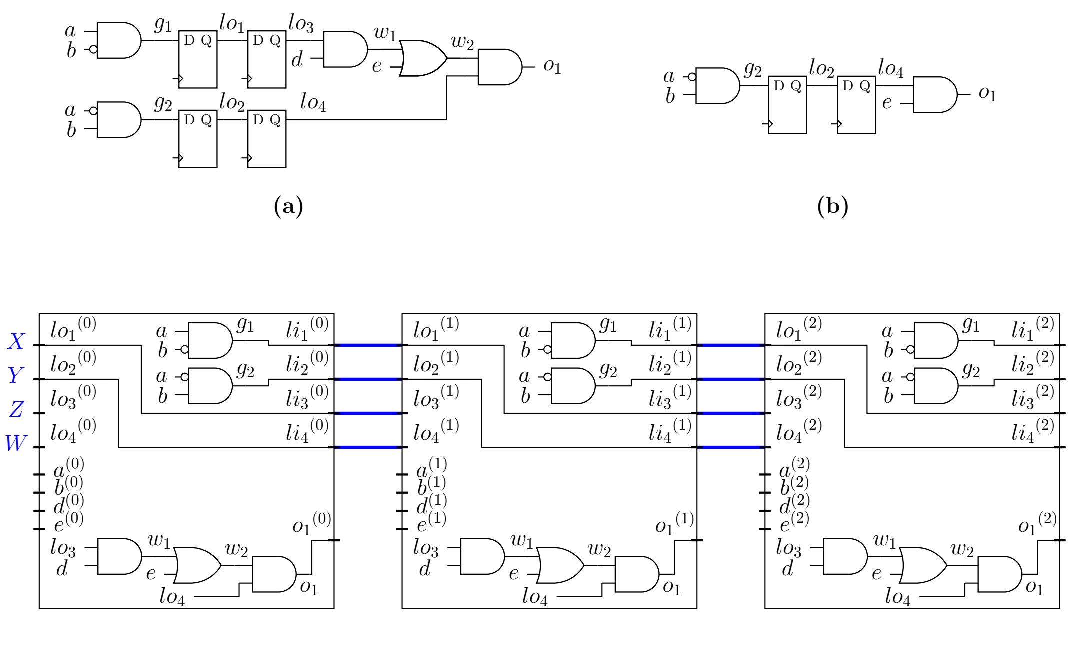


Figure 3.3: A sequential logic network (a), its optimized version (b), and its inductive case network (c) for 2-step sequential induction.

of nodes that are not in the TFO cone of n. Then, we consider all versions of n obtained by replacing one of its fanins with one of the nodes in D as resubstitution candidates for n. As with the redundancies, for each resubstitution candidate n', we use SAT to check if some window output would differ when n is replaced with n'.

Enhancement 3

We further improve our method by considering redundancy assumptions in the base case and the inductive case networks. To elaborate, suppose that we found a valid optimization Δ for the first frame of the base case. Then, we check whether Δ is also valid for the subsequent frames, *assuming* that the all preceding frames are already updated with Δ. Namely, for $i > 1$, we check if Δ is valid for frame i of the base case, assuming all frames $1, \ldots, i-1$ are transformed with Δ. Once the validity of Δ is confirmed in all frames of the base case network, update the first k frames of the inductive case network with Δ and check for its validity in the last frame. At any point, if we find Δ is not valid for the considered frame, we undo it in all previous frames.

Fig. 3.4(a) shows a simple sequential network with feedback whose output is always zero provided that the initial state of the register is zero. With assumptions, our proposed method is able to prove this. For the base case, it is clear that $g1$ is stuck at zero. For the inductive case (shown in Fig. 3.4(b)), if we assume g_1 of the first frame is stuck at

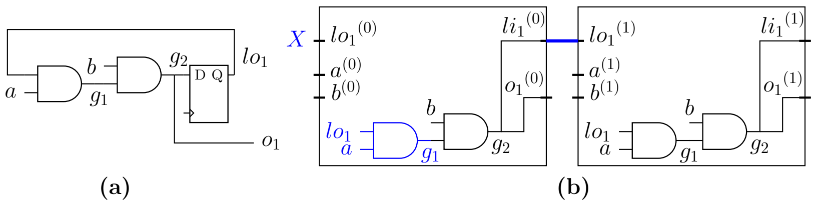


Figure 3.4: A sequential logic network with feedback (a) and its inductive case network (b) for 1-step sequential induction before applying assumptions in the first frame.

zero, then so is g_1 in the second frame. *This is also an example of a sequential network that is not optimized by retiming-based methods.*

3.3.5 Complete Algorithm

The high-level pseudocode of our method without assumptions (i.e., with the first and second enhancements above) is presented in Algorithm 3.1 whereas the Algorithm 3.2 shows the pseudocode considering all three enhancements.

In both our algorithms, we use the following definition of dangling registers:

Definition 2 (Dangling Registers)**.** *We say that a register r in a logic network N is non-dangling if there is a combinational logic path from the RO of r to either*

1. *any PO, or*
2. *the RI of any other non-dangling register.*

All remaining registers are called dangling registers.

In Algorithm 3.1, after constructing the two derived networks N^b and N^i, the gates of the input network are processed one at a time. For each gate, the algorithm iterates of candidate optimizations Δ and checks if Δ is valid for all frames of the base case network N^b. If so, it checks if Δ is also valid for the last frame of the inductive network N^i. If both checks succeed, the algorithm applies Δ in all frames of the two derived networks as well as in the original network N.

Algorithm 3.2 is an enhanced version of Algorithm 3.1 which additionally support temporary application, and if necessary, undoing of unconfirmed candidate optimizations. In Line 4 of Algorithm 3.2, the algorithm iterates over all gates in the original network, and in Line 5, it considers different optimization candidates Δ. We use the letter Δ to denote a simple logic transformation such as a fanin redundancy or a resubstitution. Then, for each frame of the base case network, the algorithm checks if Δ is valid in that frame; if it is valid, then the algorithm applies Δ in that frame (Line 6-Line 10). If it is

Algorithm 3.1: High-level pseudocode of sequential optimization with k-step induction without assumptions.

```
Input  : Input network $N$. Number of frames $k$.
Output : Updated network $N$.
1  $N^b \leftarrow$ N unrolled into $k$ frames and first frame ROs replaced with initial states.
2  $N^i \leftarrow$ N unrolled into $k+1$ frames.
3  Let $N^{b,j}, N^{i,j}$ denote the $j$-th frame of $N^b, N^i$ respectively.
4  for each gate $g \in N$ do
5      for each candidate optimization $\Delta$ for $g$ do
6          if $\Delta$ is not valid in all frames of $N^b$ then
7              Continue loop.
8          if $\Delta$ is invalid for $g$ in $N^{i,k+1}$ then
9              Continue loop.
10         Apply $\Delta$ in $N^{b,1}, \ldots, N^{b,k}$.
11         Apply $\Delta$ in $N^{i,1}, \ldots, N^{i,k+1}$.
12         Apply $\Delta$ in $N$.
13 Recursively remove all dangling registers and their MFFCs from $N$.
14 return $N$
```

valid in all frames of the base case network, then it applies Δ in the first k frames of the inductive case network (Line 11) and checks for the validity in the last frame (Line 12) of the inductive case network. If it is valid, the algorithm applies Δ in the last frame (Line 15) as well as in the original network (Line 16). At any point, if Δ is invalid for the considered frame, it undoes all preceding applications of it (Line 8 and Line 13).

In Lines 7 and 12, to check for the validity of a target optimization Δ, the algorithm first constructs a window around the target node in the respective network. (Note that the window is not restricted to the considered frame; to take the reachable states into consideration, the window should span to all previous frames in general.) Then it encodes the following as a SAT problem:

Is there an input pattern (for the window) that would make at least one output differ for the window with and without the candidate optimization?

In other words, the algorithm constructs a miter to compare the outputs of the window with and without the optimization and checks if the miter can be satisfied. If it is UNSAT, then the target optimization is valid. The size of the window and the conflict limit for the SAT solver are configurable parameters.

Algorithm 3.2: High-level pseudocode of sequential optimization with k-step induction with assumptions.

```
Input  : Input network $N$. Number of frames $k$.
Output : Updated network $N$.
1  $N^b \leftarrow$ N unrolled into $k$ frames and first frame ROs replaced with initial states.
2  $N^i \leftarrow$ N unrolled into $k+1$ frames.
3  Let $N^{b,j}, N^{i,j}$ denote the $j$-th frame of $N^b, N^i$ respectively.
4  for each gate $g \in N$ do
5      for each candidate optimization $\Delta$ for $g$ do
6          for $j = 1, \ldots, k$ do
7              if $\Delta$ is invalid for $g$ in $N^{b,j}$ then
8                  Undo $\Delta$ in all frames $N^{b,1}, \ldots, N^{b,j-1}$.
9                  Continue outer loop.
10             Apply $\Delta$ in $N^{b,j}$.
11         Apply $\Delta$ in $N^{i,1}, \ldots, N^{i,k}$.
12         if $\Delta$ is invalid for $g$ in $N^{i,k+1}$ then
13             Undo $\Delta$ in $N^{b,1}, \ldots, N^{b,k}$ and $N^{i,1}, \ldots, N^{i,k}$.
14             Continue loop.
15         Apply $\Delta$ in $N^{i,k+1}$.
16         Apply $\Delta$ in $N$.
17 Recursively remove all dangling registers and their MFFCs from $N$.
18 return $N$
```

3.3.6 Correctness of the Proposed Approach

In this section, we show the correctness of our algorithms, both with and without the use of assumptions. To this end, we consider sequential networks in the following setting:

1. All sequential elements in the network are positive-edge-triggered D flip-flops,
2. PIs are set on the negative edge of the clock, and
3. The clock edge that immediately following the reset is a negative edge.

We label the positive clock edges that follows reset by non-negative integers $0, 1, 2, \ldots$. Unless explicitly mentioned otherwise, we use the following notations in our theorems and proofs: we denote the input sequential logic network by N, and assume that it has m PIs, n POs, and ℓ registers. Let For $x \in \mathbb{B}^m$ and $y \in \mathbb{B}^\ell$, let $\mathrm{PO}(N, x, y) \in \mathbb{B}^n$ and

$\mathrm{RI}(N, x, y) \in \mathbb{B}^{\ell}$, respectively, denote the PO and RI values of N when PIs are set to x and register states (ROs) are set to y. We use $y_0 \in \mathbb{B}^{\ell}$ to denote the initial state of the registers. Let $\{x\}_0^T = x_0, x_1, \ldots, x_T$ where $x_i \in \mathbb{B}^m$ for all $i = 0, 1, \ldots, T$ be a sequence of Boolean vectors. Given a sequence $\{x\}_0^T$ and initial state $y \in \mathbb{B}^{\ell}$, suppose that registers are set to y at reset and that x_t is set as PI values at t-th negative clock edge. We observe the PO and RI values just before the T-th positive clock edge. Let $\mathrm{PO}(N, \{x\}_0^T, y) \in \mathbb{B}^n$ and $\mathrm{RI}(N, \{x\}_0^T, y) \in \mathbb{B}^{\ell}$, respectively, denote the resulting PO values and RI values.

For two networks N_1, N_2, we define $\mathrm{EQ}(N_1, N_2, T, y^1, y^2)$ to be the proposition that for any sequence of T PI vectors, the PO values of the two networks are the same, when started, respectively, from y^1 and y^2 as the initial state. Formally,

$$\begin{aligned}\mathrm{EQ}(N_1, N_2, T, y^1, y^2) := {} & \text{For all sequences } \{x\}_0^T \in \mathbb{B}^{m \times T}, \\ & \mathrm{PO}(N_1, \{x\}_0^T, y^1) = \mathrm{PO}(N_2, \{x\}_0^T, y^2).\end{aligned}$$

Similarly, we define $\mathrm{EQ}^{\star}(N_1, N_2, T, y)$ to be the proposition that for any sequence of T PI vectors, the PO and RI values of the two networks are the same, when started from y as the initial state. Formally,

$$\begin{aligned}\mathrm{EQ}^{\star}(N_1, N_2, T, y) := {} & \text{For all sequences } \{x\}_0^T \in \mathbb{B}^{m \times T}, \\ & \mathrm{PO}(N_1, \{x\}_0^T, y) = \mathrm{PO}(N_2, \{x\}_0^T, y) \text{ and} \\ & \mathrm{RI}(N_1, \{x\}_0^T, y) = \mathrm{RI}(N_2, \{x\}_0^T, y).\end{aligned}$$

Note that, here, both networks are initialized with the same initial state y.

We now define the notion of sequential equivalence and strong sequential equivalence.

Definition 3 (Sequential Equivalence)**.** *Suppose that N_1 and N_2 are two sequential networks with m PIs and n POs, and suppose that N_1 has ℓ_1 registers and N_2 has ℓ_2 registers. Let $y_0^1 \in \mathbb{B}^{\ell_1}$ and $y_0^2 \in \mathbb{B}^{\ell_2}$ be the initial states of the registers of N_1 and N_2, respectively. We say that two sequential networks N_1 and N_2 are sequentially equivalent if $\mathrm{EQ}(N_1, N_2, T, y_0^1, y_0^2)$ is true for all $T \in \mathbb{N}_0$.*

When $\ell_1 = \ell_2$ and $y_0^1 = y_0^2$, we say that N_1 and N_2 are strongly sequentially equivalent if $\mathrm{EQ}^{\star}(N_1, N_2, T, y_0^1)$ is true for all $T \in \mathbb{N}_0$.

Our goal is to show that the output of Algorithm 3.2 is sequentially equivalent to the input network. Outline of our proof is as follows:

1. Show that the intermediate network we get before removing dangling registers, i.e., the network N just before Line 17, is sequential equivalent to the input network.
2. Show that removing the dangling registers in Line 17 does not affect the sequential

equivalence.

For step 1 above, we in fact show the stronger result that the intermediate network is strongly sequentially equivalent to the input network. To this end, we first start with the following lemma, which essentially yields an alternative definition of sequential equivalence.

Lemma 1. *Let N_1, N_2 be two logic networks with the same initial state y_0, and let $S \subseteq \mathbb{B}^\ell$ be the set of all reachable states of N_1. If $\mathrm{PO}(N_1, x, y) = \mathrm{PO}(N_2, x, y)$ and $\mathrm{RI}(N_1, x, y) = \mathrm{RI}(N_2, x, y)$ for all $x \in B^m$ and $y \in S$, then N_1 and N_2 are strongly sequentially equivalent.*

Proof. Suppose that for all $x \in B^m$ and $y \in S$, it holds that

$$\mathrm{PO}(N_1, x, y) = \mathrm{PO}(N_2, x, y) \tag{3.1}$$

and

$$\mathrm{RI}(N_1, x, y) = \mathrm{RI}(N_2, x, y). \tag{3.2}$$

We show that, for any sequence $\{x\}_0^T$,

$$\mathrm{PO}(N_1, \{x\}_0^T, y_0) = \mathrm{PO}(N_2, \{x\}_0^T, y_0) \tag{3.3}$$

and

$$\mathrm{RI}(N_1, \{x\}_0^T, y_0) = \mathrm{RI}(N_2, \{x\}_0^T, y_0) \tag{3.4}$$

using induction.

To this end, fix any sequence $\{x\}_0^T$. We use the notation $\{x\}_0^t$ to denote the subsequence $x_0, x_1, \ldots, x_t$. Considering the initial clock cycle, we have

$$\begin{aligned} \mathrm{PO}(N_1, \{x\}_0^0, y_0) &= \mathrm{PO}(N_1, x_0, y_0) \\ &= \mathrm{PO}(N_2, x_0, y_0) \\ &= \mathrm{PO}(N_2, \{x\}_0^0, y_0) \end{aligned}$$

and

$$\begin{aligned} \mathrm{RI}(N_1, \{x\}_0^0, y_0) &= \mathrm{RI}(N_1, x_0, y_0) \\ &= \mathrm{RI}(N_2, x_0, y_0) \\ &= \mathrm{RI}(N_2, \{x\}_0^0, y_0). \end{aligned}$$

In derivations above, the second equality follows due to Eqs. (3.1) and (3.2).

As the inductive hypothesis, assume that for a positive integer t such that $T \geq t > 0$,

$$\mathrm{RI}(N_1, \{x\}_0^{t-1}, y_0) = \mathrm{RI}(N_2, \{x\}_0^{t-1}, y_0).$$

Note that $\mathrm{RI}(N_1, \{x\}_0^{t-1}, y_0)$ is also in S. Then we have

$$\begin{aligned}\mathrm{PO}(N_1, \{x\}_0^{t}, y_0) &= \mathrm{PO}\left(N_1, x_t, \mathrm{RI}(N_1, \{x\}_0^{t-1})\right)\\ &= \mathrm{PO}\left(N_2, x_t, \mathrm{RI}(N_2, \{x\}_0^{t-1})\right)\\ &= \mathrm{PO}(N_2, \{x\}_0^{t}, y_0)\end{aligned}$$

and

$$\begin{aligned}\mathrm{RI}(N_1, \{x\}_0^{t}, y_0) &= \mathrm{RI}\left(N_1, x_t, \mathrm{RI}(N_1, \{x\}_0^{t-1})\right)\\ &= \mathrm{RI}\left(N_2, x_t, \mathrm{RI}(N_2, \{x\}_0^{t-1})\right)\\ &= \mathrm{RI}(N_2, \{x\}_0^{t}, y_0).\end{aligned}$$

In derivations above, again, the second equality follows due to Eqs. (3.1) and (3.2).

Thus, by induction, it follows that Eqs. (3.3) and (3.4) hold for $\{x\}_0^T$. □

With Lemma 1, we now show that the strong sequential equivalence holds with respect to a single valid transformation Δ. Namely, we show the following lemma:

Lemma 2. *Consider a logic network N and its k-step base case and inductive versions N^b and N^i. Let Δ be a logic transformation and let $N_\Delta, N_\Delta^b, N_\Delta^i$, respectively, be the networks obtained by applying Δ to N, to all frames of N^b, and to the last frame of N^i. If N^b and N^i, respectively, are combinationally equivalent to N_Δ^b and N_Δ^i, then N and N_Δ are sequentially equivalent.*

Proof. Let S be the set of reachable states, and note that S can be decomposed as the countable union $S = S_0 \cup S_1 \cup S_2 \ldots$, where S_0 is the set with only the initial state, S_1 is the set of reachable states after the first clock cycle, S_2 is the set of reachable states after the second clock cycle, and so on.

We first claim the following: For $i = 0, 1, 2, \ldots$, for any $x \in \mathbb{B}^m$ and $y \in S_0 \cup S_1 \cup S_2 \ldots$, we have that $\mathrm{PO}(N, x, y) = \mathrm{PO}(N_\Delta, x, y)$ and $\mathrm{RI}(N, x, y) = \mathrm{RI}(N_\Delta, x, y)$.

To see this, first fix any $i \in \{0, \ldots, k-1\}$, and let $y \in S_i$. Let $x_0, \ldots, x_{i-1}$ be the sequence of PI vectors that resulted in state y after i clock cycles. Now, for networks N^b and N_Δ^b, set the j-th frame PIs to x_j for $j = 0, \ldots, i-1$, set i-th frame PIs to x, and set the remaining PIs arbitrarily (but same for both networks). By design of N^b, we must have its $i-1$-th frame RIs set to y, and by combinational equivalence, the same holds for N^b_Δ. Thus, considering the combinational equivalence for i-th frame POs and RIs, the claim above holds for i-th frame.

Now, fix any $i \geq k$, let $y \in S_i$ and let $x_0, x_1 \ldots x_{i-1}$ be the sequence of PI vectors that resulted in state y after i clock cycles as before. This time, we use the equivalence of N^i and N^i_Δ, and for this, we set ROs of the 0-th frame to state we get on sequence $x_0, \ldots, x_{i-k}$, and we set j-th frame PIs to x_{i-k+j} for $j = 0, \ldots, k-1$, and k-th frame PIs to x. Then, by the combinational equivalence of N^i and N^i_Δ, considering the last frame POs and RIs, we have that the claim holds for i-th frame. □

With Lemma 2, we now proceed to show that the input network of Algorithm 3.2 is sequentially equivalent to the output network.

Theorem 1. *Let N be the input network of Algorithm 3.1 and let $N^\star$ be the output network. Then N and $N^\star$ are sequentially equivalent.*

Proof. Let N^{int} denote the intermediate network produced by Algorithm 3.1 just before removing dangling registers, i.e., just before Line 17. Note that N^{int} is obtained from N by applying a sequence of valid transformations, and each transformation preserves the strong sequential equivalence due to Lemma 2. Thus N^{int} and N are strongly sequentially equivalent, which is a special case of being sequentially equivalent.

Observe that removing dangling registers and their MFFCs does not affect the value of any PO or any remaining non-dangling register. Thus, N^{int} and $N^\star$ are sequentially equivalent. Finally, since sequential equivalence is transitive by definition, we have that N and $N^\star$ are sequentially equivalent. □

3.3.7 Characterizing SODC-Optimizable Transformations

As proven above, our approach only finds correct SODC-based optimizations, so it has the soundness property. However, due to the limitations of sequential induction, the proposed algorithm is not complete, i.e., it is unable to prove all valid SODC-based fanin redundancies/resubstitutions.

In the remainder of this section, we examine in what situations the proposed method can find the SODC-based optimizations and propose workarounds for situations where it may struggle to do so.

Networks without sequential feedback

Consider a sequential network with no sequential feedback. This means that for any register, its future (not necessarily the immediate next state) state does not depend its

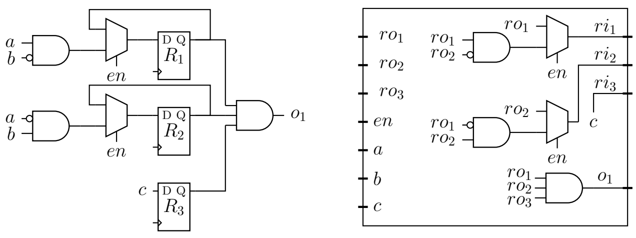


Figure 3.5: An example sequential circuit and its frame representation.

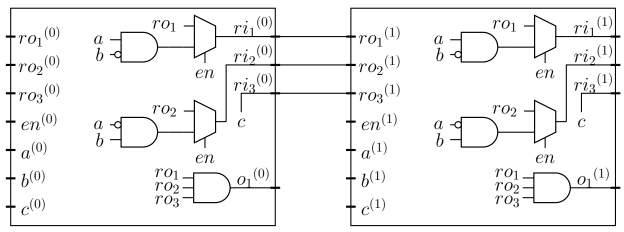


Figure 3.6: The inductive network for the sequential circuit in Figure 3.5.

current state. For such a network N, let d be the sequential depth, which is the maximum number of registers in any path from a PI to a PO. If our proposed methods is used to optimize N using d-step sequential induction, then our method is capable of finding any valid SODC-based optimization. This is because, the reachable states approximated by the inductive case network will be the same as the actual reachable states after d clock cycles.

Networks with sequential feedback

In networks with sequential feedback, the future state of a register can depend on its current state. In such networks, the number of sequential induction steps required to characterize all reachable states can be exponential, and hence it is impractical. Moreover, unless assumptions are used, it may not be possible to find all valid SODC-based optimizations with sequential induction.

Consider the following example sequential circuit (Fig. 3.5) and its inductive case network for sequential induction (Fig. 3.6). Suppose that all registers are initially set to 0, and observe that R_1 and R_2 can never be 1 at the same time. Thus, the output o_1 is stuck at 0.

We analyze under what conditions our proposed method is able to find the above optimization.

Note that in the 1-step inductive case network shown in Fig. 3.6, the ro_1 and ro_2 of the initial frame are considered as PIs, thus they can take arbitrary values. When en is 0, these values can propagate to the next frame, and if ro_1, ro_2, and c of initial frame are

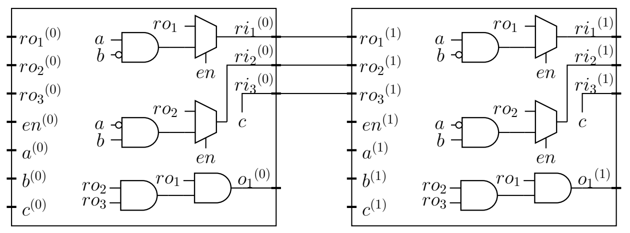


Figure 3.7: The inductive network for the sequential network in Figure 3.5, where the decomposition of the 3-input AND is unfavorable to the proposed algorithm.

all 1 and $en = 0$, then the output o_1 of the last frame is 1. Thus, this setup is unable to prove the the stuck-at-0 property of o_1, unless we use assumptions. This remains the case even if use k-step induction with any $k > 1$ since the values of ro_1 and ro_2 of the initial frame can propagate to the last frame as long as en remain 0.

However, from Fig. 3.5, we clearly see that the values of both R_1 and R_2 can never be 1 at the same time if the initial states are 0. This is because, if R_1 and R_2 are not 1 in the current clock cycle, then they are not 1 in the next clock cycle as well. Thus, one would hope to find the optimization considering the assumptions. The challenge for our algorithm is to find the right assumptions to make.

In the proposed method, we consider a simplified set of assumptions. Namely, as assumptions, we always use the candidate redundancy (or resubstitution) property that we are trying to prove. Consequently, there arise cases where these simplified assumptions are not sufficient as-is to prove the property.

To illustrate, consider again the sequential network in Fig. 3.5 or a different version of the same network as shown in Fig. 3.7 where the 3-input AND gate is decomposed into two 2-input AND gates. Let us assume that o_1 is 0 in the initial frame.

With the goal of proving that o_1 is stuck at 0 in the last frame, suppose that o_1 is 0 in the initial frame. Unfortunately, the condition of $o_1 = 0$ in the initial frame does not prevent ro_1 and ro_2 from being 1 at the same time in the first frame, because $o_1 = 0$ is possible with $ro_1 = 1$, $ro_2 = 1$, and $ro_3 = 0$. Thus, with this assumption, the algorithm is unable to prove the stuck-at-0 property of o_1.

Alternatively, consider the case where the 3-input AND gate is decomposed into two 2-input AND gates in a different manner, as shown in Fig. 3.8. In contrast to the previous case, in this network, the algorithm is able to prove the stuck-at-0 property of o_1, by first proving that gate g_1 of the decomposition is stuck at 0. Namely, the algorithm first assumes that g_1 is stuck at 0 in the first frame. This implies that ro_1 and ro_2 are not 1 at the same time in the first frame. For the second frame, the value of g_1 is either AND of ro_1 and ro_2 values from the first frame, or its the and of $a \wedge \neg b$ and $\neg a \wedge b$, which can never be 1 at the same time.

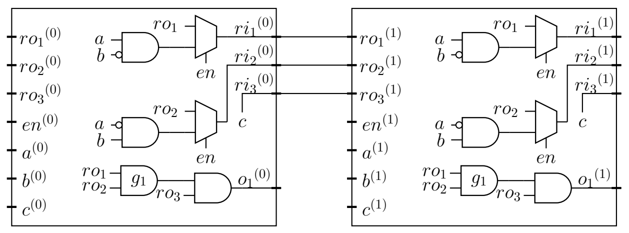


Figure 3.8: The inductive network for the sequential network in Figure 3.5, where the decomposition of the 3-input AND is conducive to the proposed algorithm.

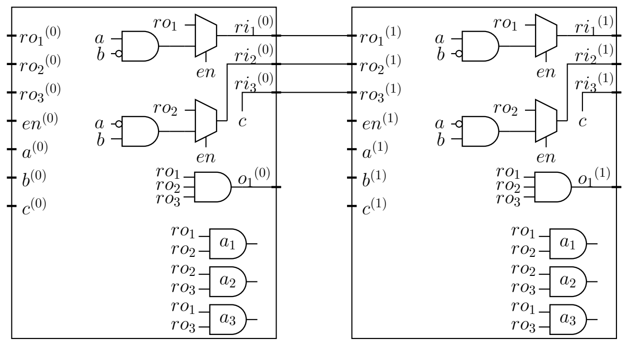


Figure 3.9: The inductive network for the sequential network in Figure 3.5, where shadow nodes are added to reflect the assumptions on the reachable states of registers.

Considering the two scenarios described above, it is clear that the proposed method's ability to find the SODC-based optimizations depends on the network structure. To mitigate this issue, we propose two approaches.

Equivalent Structural Transformations One straightforward way to address the issue is to consider different equivalent structures for different portions of the network and different decompositions of complex gates. This will allow the algorithm to consider different anchor nodes and explore assumptions on them.

External Assumptions The other approach is to expand the possible types of assumptions used by the algorithm. Recall that the current method uses the candidate redundancy property as the only assumption. However, this is strictly not necessary, and the algorithm may consider any arbitrary assumption on the reachable states of registers. Such general assumptions can be proven by the current algorithm, simply by introducing additional shadow nodes to serve as anchor nodes for the assumptions.

To elaborate on external assumptions, suppose that the algorithm is faced with the network in Fig. 3.7. From the initial conditions, we know that o_1 is stuck at zero. If no decomposition is preformed, the algorithm will again construct the inductive network as shown in Fig. 3.6. However, instead of assuming that the 3-input AND gate of the first frame gate is stuck at 0 and then trying to prove the same for the second frame, the algorithm may perform the following steps:

1. Find the registers on which the value of o_1 depends. This can be done by traversing the transitive fanin cone of o_1. (In general, such registers can be computed for all gates in a more efficient manner using a topological traversal from inputs to outputs.) In this case, the registers are ro_1, ro_2, and ro_3.

2. Explore different subsets of value combinations for these registers which would make o_1 stuck at 0. The number of such subsets grows doubly-exponentially with the number of registers (e.g., for r registers, there are 2^r possible states, so there are 2^{2^r} many subsets of states). Since exploring all possible subsets is impractical, the algorithm may use heuristics to select a subset of states to explore. For example, the algorithm may choose states where a pair of registers are not both 1 at the same time.

3. For each considered subset S of states, construct a new temporary node n_S in the network which is 0 if and only if the register value combination is in S. E.g., for o_1 of the network in Fig. 3.5, the algorithm may create new temporary AND gates $ro_1 \wedge ro_2$, $ro_2 \wedge ro_3$, and $ro_2 \wedge ro_3$ as shown in Fig. 3.9. The gate a_1 denotes the assumption that the state $(ro_1 = 1, ro_2 = 1, ro_3 = *)$ is not reachable, gate a_2 denotes the assumption that the state $(ro_1 = 1, ro_2 = *, ro_3 = 1)$ is not reachable, and gate a_3 denotes the assumption that the state $(ro_1 = *, ro_2 = 1, ro_3 = 1)$ is not reachable.

4. Assuming that n_S is stuck at 0 in the first frame, check whether the n_S is stuck at 0 in the second frame. If so, assuming that n_S is stuck at 0 in the second frame, check whether the n_S is stuck at 0 in the third frame, and so on. If n_S is stuck at 0 in all frames, then conclude that the assumption is valid, and hence o_1 is stuck at 0 in all reachable states.

Once the algorithm has proven the stuck at 0 property of o_1, it can proceed to remove the redundant gates as well as the shadow nodes.

To summarize, while the proposed algorithm always finds correct SODC-based optimizations, the limited horizon of sequential induction and the implementation choices for accommodating assumptions can lead to missed optimization opportunities. These drawbacks can be mitigated by using symmetry-breaking logic transformations and adding support for more elaborate assumptions.

3.4 Experimental Results

In this section, we present and discuss the experimental results obtained using the proposed approach which is implemented as part of a commercial EDA tool. For the evaluations, we consider a subset of OpenCores [2] and some industrial designs as the sequential benchmark design suite.

We proceed in three steps. First, we compare the effects of different configurations of our algorithm by optimizing technology-independent logic, using And-Inverter Graphs (AIGs) as the logic representation. Next, we evaluate the performance of the model on technology-mapped designs, and finally we present the results obtained on industrial designs after place and route. Note that in experimental results, we report the area improvements as negative percentages, indicating by how much the area is reduced.

3.4.1 Comparison of Different Configurations

Recall that, on top the basic version of our algorithm, we also proposed several extensions, namely, multi-step sequential induction, assumptions, and support for resubstitutions. To compare the effectiveness of these extensions, we optimize a subset of OpenCores designs using different configurations of our algorithm.

As the baseline for comparisons, we consider a state-of-the-art sequential optimization flow [125] together with combinational rewriting (commands *scorr* and *rewrite* in ABC [34]). The two optimizations are interleaved and run until saturation, i.e., no further reduction is observed. In the experimental flow, we additionally run our algorithm (without and with respective extensions) on top of the baseline.

Redundancy removal vs. redundancy removal + resubstitution

In Table 3.1, we present the results obtained by running our method with only redundancy removals and with both redundancy removals and resubstitutions. The columns 'NAND2', 'Lev', and 'FF' show the number of two-input NAND-gates, combinational logic levels, and flip-flops, respectively. The last two columns for each experimental setting show the runtime of the experimental flow in seconds and the percentage NAND2 reduction over the baseline.

Here we used 1-step sequential induction with windowing, where the window size is limited to 500 nodes with at most 16 levels in the transitive fanout cone of a target node. For the resubstitution, we limit the divisor count to 100 nodes. We also set a tight control on the level count to prevent increasing it during resubstitution.

As seen in Table 3.1, the redundancy removals alone lead to an average reduction of

Table 3.1: Comparison of the proposed method against the baseline (redundancy removal)

	Baseline			**Our Method (redundancy removal)**				
Name	NAND2	Lev	FF	NAND2	Lev	FF	Time (s)	NAND2%
aes_core	22026	32	530	21576	32	530	22.7	-2.04
des_area	4611	37	64	4611	37	64	0.7	0.00
des_perf	77288	23	8808	76691	23	8808	302.7	-0.77
ethernet	168	13	47	166	13	47	0	-1.19
i2c	931	24	126	891	24	126	0.1	-4.30
mem_ctrl	7097	31	1050	7003	31	1050	1.9	-1.32
pci_bridge32	17656	32	3198	17403	32	3198	10.2	-1.43
pci_spoci_ctrl	704	20	60	678	20	60	0.2	-3.69
sasc	597	10	117	568	10	117	0.1	-4.86
simple_spi	779	12	131	772	12	131	0.1	-0.90
spi	3621	31	229	3590	31	229	0.5	-0.86
ss_pcm	464	9	87	399	9	87	0	-14.01
steppermotor	138	17	25	125	17	25	0	-9.42
systemcaes	11106	42	670	11105	42	670	4.3	-0.01
systemcdes	2696	36	190	2692	34	190	2	-0.15
tv80	7740	58	359	7553	58	359	2.5	-2.42
usb_funct	13910	27	1722	13560	26	1721	8	-2.52
usb_phy	457	12	98	407	11	98	0	-10.94
vga_lcd	89555	27	17032	89392	27	17032	286.8	-0.18
wb_conmax	47026	32	770	42491	32	770	43.4	-9.64
wb_dma	3283	19	521	3258	19	521	0.6	-0.76
Average								-3.40

3.40% in the number of NAND2 gates. When resubstitutions are allowed on top of redundancies, the average reduction increases to 3.65%, as shown in Table 3.2.

As a final remark, note that all testcases have been verified using sequential verification (*dsec*) in ABC [34] where the verification time is below 13 seconds for all the benchmarks.

Single-step vs. multi-step induction

In Table 3.3, we present the results obtained by running our method with 2-step sequential induction. In this case, we use the same settings as in the previous experiment with both redundancy removals and resubstitutions enabled, but with a window size of 5000. Note that we use an increased window size to support larger number of inductive steps. To effectively find potential optimizations in the unrolled inductive-case network, the window need to contain logic from all unrolled frames. When 2-step induction is used, the average reduction in the number of NAND2 gates increases to 3.97% from 3.65%. As before, all testcases have been verified with ABC's *dsec*, where the maximum observed

Table 3.2: Comparison of the proposed method against the baseline (redundancy removal + resubstitution)

	Baseline			Redundancy removal & resubstitution				
Name	NAND2	Lev	FF	NAND2	Lev	FF	Time (s)	NAND2%
aes_core	22026	32	530	21535	32	530	49.2	-2.23
des_area	4611	37	64	4611	37	64	1.7	0.00
des_perf	77288	23	8808	76534	23	8808	592.6	-0.98
ethernet	168	13	47	166	13	47	0	-1.19
i2c	931	24	126	891	24	126	0.3	-4.30
mem_ctrl	7097	31	1050	6998	31	1050	3.7	-1.39
pci_bridge32	17656	32	3198	17403	32	3198	20.9	-1.43
pci_spoci_ctrl	704	20	60	674	20	60	0.5	-4.26
sasc	597	10	117	568	10	117	0.1	-4.86
simple_spi	779	12	131	772	12	131	0.2	-0.90
spi	3621	31	229	3592	31	229	0.9	-0.80
ss_pcm	464	9	87	399	9	87	0	-14.01
steppermotor	138	17	25	125	17	25	0	-9.42
systemcaes	11106	42	670	11105	42	670	8.2	-0.01
systemcdes	2696	36	190	2687	34	190	5.7	-0.33
tv80	7740	58	359	7527	58	359	6.2	-2.75
usb_funct	13910	27	1722	13557	26	1721	16.8	-2.54
usb_phy	457	12	98	407	11	98	0.1	-10.94
vga_lcd	89555	27	17032	89385	27	17032	477.3	-0.19
wb_conmax	47026	32	770	40734	32	770	84.9	-13.38
wb_dma	3283	19	521	3257	19	521	1.2	-0.79
Average								-3.65

verification time is 74 seconds.

Without assumptions vs. with assumptions

When assumptions are allowed, we observed improvements in the NAND2 reduction for specific benchmarks. Namely, when our algorithm is used with 1-step sequential induction allowing both redundancy removals and resubstitutions, reductions of -4.94%, -2.98%, and -11.16% were observed, respectively, for benchmarks `ethernet`, `i2c`, and `usb_phy`.

Redundancy removal + resubstitution with different window sizes

When using both redundancy removals and resubstitutions with 1-step sequential induction, we observed that increasing the window size from 500 to 50 000 nodes produced

Table 3.3: Comparison of the proposed method against the baseline (*k*-step induction)

	Baseline			**Our Method with 2-step induction**				
Name	NAND2	Lev	FF	NAND2	Lev	FF	Time (s)	NAND2%
aes_core	22026	32	530	21132	31	530	288.9	-4.06
des_area	4611	37	64	4604	37	64	37.7	-0.15
des_perf	77288	23	8808	76507	23	8808	1509	-1.01
ethernet	168	13	47	166	13	47	0.1	-1.19
i2c	931	24	126	891	24	126	0.9	-4.30
mem_ctrl	7097	31	1050	6976	31	1048	28.6	-1.70
pci_bridge32	17656	32	3198	17376	32	3197	153.1	-1.59
pci_spoci_ctrl	704	20	60	670	20	60	1.5	-4.83
sasc	597	10	117	568	10	117	0.2	-4.86
simple_spi	779	12	131	772	12	131	0.5	-0.90
spi	3621	31	229	3586	31	229	31.9	-0.97
ss_pcm	464	9	87	399	9	87	0.1	-14.01
steppermotor	138	17	25	125	17	25	0.1	-9.42
systemcaes	11106	42	670	11087	42	670	41.7	-0.17
systemcdes	2696	36	190	2685	34	190	12.5	-0.41
tv80	7740	58	359	7419	58	359	37.6	-4.15
usb_funct	13910	27	1722	13541	26	1721	49.7	-2.65
usb_phy	457	12	98	403	11	98	0.2	-11.82
vga_lcd	89555	27	17032	89352	27	17032	1633.5	-0.23
wb_conmax	47026	32	770	40375	32	770	344.3	-14.14
wb_dma	3283	19	521	3257	19	521	3.2	-0.79
Average								-3.97

even better results as shown in Table 3.4. The reduction in the number of NAND2 gates increased from 3.65% to 4.10% on average. In this case, the maximum observed sequential verification time is 27 seconds.

3.4.2 Technology Mapped Results

In Table 3.5, we present the results obtained after area-oriented technology mapping for the same OpenCores designs we used in the previous experiments. In this experiment, the baseline is an industrial synthesis flow that does not use any sequential logic optimizations, and Flow1 uses two iterations of sequential SAT-sweeping for mapped networks on top of the baseline. Flow2 runs Flow1 followed by our proposed method, with 1-step sequential induction, with both redundancies and resubstitutions enabled, and with a window size of 50 000.

The table shows the average improvements over the baseline. Our flow achieves an 18.3% area reduction, compared to the baseline, and a 6.9% reduction, compared to

Table 3.4: Comparison of the proposed method against the baseline (Increased window size)

	Baseline			Our Method				
Name	NAND2	Lev	FF	NAND2	Lev	FF	Time (s)	NAND2 %
aes_core	22026	32	530	21061	31	530	1404.9	-4.4
des_area	4611	37	64	4594	37	64	71.6	-0.4
des_perf	77288	23	8808	76053	23	8808	811.2	-1.6
ethernet	168	13	47	166	13	47	0	-1.2
i2c	931	24	126	889	24	126	1	-4.5
mem_ctrl	7097	31	1050	6961	31	1048	27	-1.9
pci_bridge32	17656	32	3198	17292	32	3198	151.8	-2.1
pci_spoci_ctrl	704	20	60	671	20	60	2.4	-4.7
sasc	597	10	117	568	10	117	0.2	-4.9
simple_spi	779	12	131	772	12	131	0.6	-0.9
spi	3621	31	229	3583	31	229	86.4	-1.1
ss_pcm	464	9	87	399	9	87	0.1	-14.0
steppermotor	138	17	25	125	17	25	0.1	-9.4
systemcaes	11106	42	670	11070	42	670	137.7	-0.3
systemcdes	2696	36	190	2685	34	190	22.7	-0.4
tv80	7740	58	359	7396	57	359	208.9	-4.4
usb_funct	13910	27	1722	13506	26	1721	38.2	-2.9
usb_phy	457	12	98	403	11	98	0.2	-11.8
vga_lcd	89555	27	17032	89294	27	17032	1993.1	-0.3
wb_conmax	47026	32	770	40184	32	770	391.8	-14.5
wb_dma	3283	19	521	3257	19	521	4	-0.8
Average								-4.1

Flow1 with sequential SAT-sweeping, at a cost of 20% increase in runtime. The results confirm that the two methods, the sequential SAT-sweeping and our proposed method, are orthogonal; our proposed method finds new optimization opportunities on top of state-of-the-art sequential optimizations. All benchmarks were equivalence-checked using existing sequential verification tools.

3.4.3 Post Place & Route Results on Industrial Designs

In Table 3.6, we present the results obtained after place and route for 4 industrial benchmarks, where the baseline does not use any sequential logic optimizations and the experimental flow combines our proposed method and sequential SAT-sweeping.

Our flow achieves a 2.89% reduction in combinational area, a 1.43% reduction in sequential area, a 0.18% improvement in worst negative slack (WNS), a 1.12% improvement in total negative slack (TNS), and a 1.96% reduction in total power. These results confirm

Table 3.5: Results after technology mapping for OpenCores designs

Flow	Comb. Area	Seq. Area	# Cells	Runtime
Baseline	1	1	1	-
Flow1 (SSW)	-12.2%	-4.8%	-9.6%	1
Flow2 (SSW + new method)	-18.3%	-4.8%	-14.9%	+20%

Table 3.6: Results after place and route for industrial designs

	Comb. Area	Seq. Area	WNS	TNS	Tot. Power
Average	-2.89%	-1.43%	-0.18%	-1.12%	-1.96%

that our proposed method provides significant improvements in both area and timing metrics, even after place and route. This further demonstrates that our method scales well such that it can be effectively used on large industrial benchmarks. All 4 designs have been verified with an industrial sequential verification tool which uses an state-of-the-art sequential verification flow [164].

3.5 Summary

In this chapter, we introduced a scalable sequential optimization method based on multi-step induction, extending the concept of Observability Don't Cares (ODCs) to the sequential domain through Sequential Observability Don't Cares (SODCs), which explicitly account for reachability constraints. By simultaneously optimizing two derived combinational networks—representing the base and inductive cases—our approach effectively addresses the dependency issues inherent in traditional ODC-based optimizations. Candidate optimizations are verified using a SAT-based approach that encodes ODC constraints, with a windowing technique employed to maintain scalability. Leveraging SODCs, our method uncovers optimization opportunities that prior approaches were unable to detect.

Experimental results demonstrate the scalability of our method across large industrial designs, yielding significant area improvements in both technology-mapped circuits and post place-and-route designs.

While the method incurs non-negligible runtime, it proves valuable as a high-effort sequential optimization, particularly for area-critical applications. Our analysis shows that most of the runtime is spent on SAT solving, and we anticipate substantial speedups by reducing the number of equivalence-checking SAT calls. Techniques such as randomized or counter-example-guided simulation could efficiently filter out invalid optimizations early. Additionally, exploring heuristics for optimization candidate ordering may expose further opportunities for improvement. As outlined in Section 3.3, applying symmetry-breaking logic transformations and more advanced reachability assumptions could also reveal additional optimization potential. These enhancements are left for future work.

Moreover, our method's integration into a state-of-the-art industrial sequential optimization flow yielded promising results, maintaining significant reductions in both combinational and sequential areas, even after place-and-route. We hope that the enhanced sequential optimization offered by this method will inspire further research in the field, driving efforts to better approximate reachable states and uncover more optimization opportunities.

4 Fanout-Bounded Logic Synthesis

In digital electronics, the ability to have multiple fanouts per gate allows for compact implementations of complex logic functions. However, increasing the number of fanouts of a gate can negatively impact delay performance, and the maximum number of fanouts a gate can support is typically limited. This limitation is particularly stringent in emerging technologies such as superconducting electronics, but it also has implications for CMOS technology. In this chapter, we take a rigorous approach to the generic *fanout-bounded synthesis* (FBS) problem and propose both exact and heuristic algorithms that are readily adaptable to different technologies.

This chapter is based on the work [107] presented at Design, Automation, and Test in Europe (DATE) 2023, which was also presented at the International Workshop on Logic Synthesis (IWLS) 2022 [106]. The extended version of the work, which includes adaptations to emerging technologies, was published in Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) [108].

4.1 Introduction

In conventional CMOS technology, fanout optimization has been well-studied, both as a means of improving the critical path delay [4, 21, 103, 131, 156], and as a method of optimizing special high-fanout nets such as clock and reset signals [167]. However, the techniques developed for CMOS technology are not generally transferable to emerging technologies such as superconducting electronics (e.g., AQFP [157], RQL [31], RSFQ [104]), field-coupled nano-computing technologies (e.g., QCA [101]), and spintronics [43], which generally have tight, explicit fanout bounds and/or significantly different timing models (e.g., clocked gates). Thus the allowed circuit transformations in such technologies can be fundamentally different.

For instance, in CMOS technology, the delay increase caused by a high number of fanouts can be mitigated by techniques such as transistor sizing. However, this option is not available for post-CMOS technologies. Instead, when designing for emerging technologies that have globally imposed, hard fanout limits, fanout-bounding is achieved through a combination of gate duplications and buffer insertions. This procedure tends to consume a significant portion of resources as compared to CMOS, so it is typically considered relatively early in the synthesis process, e.g., in the logic synthesis stage.

Motivated by the aforementioned differences from CMOS, we first consider the following general *fanout-bounded synthesis* (FBS) problem in the unit-delay model: Given an input logic network and the fanout bounds and area costs of different gate types/buffers, re-synthesize the logic network by means of gate duplications and buffer insertions such that each gate meets its respective fanout bound while the total area is minimized. Note that the unit-delay model encompasses many emerging technologies that have clocked gates (e.g., AQFP, QCA). Zhang and Jiang [178] recently studied this general FBS problem (in the same unit-delay model) and presented an algorithm composed of several heuristics, where the main idea was to duplicate gates if doing so locally reduces the number of buffers (see Section 4.3 for more details).

In this work, we revisit the FBS problem by taking a rigorous approach: namely, we present the first known *integer linear programming* (ILP) formulation of this problem for a fixed target delay and use it to obtain optimum area FBS solutions for a number of EPFL [11] benchmarks and benchmarks of [76]. Our ILP uses the number of copies and buffers associated with different gates and levels as variables, and has constraints to ensure that there are sufficiently many gate copies and buffers to support all fanouts subject to fanout bounds. As we see in Section 4.4, this formulation is versatile and can be extended, for example, to facilitate different types of gates and buffers as well as different fanout constraints for primary inputs.

We then present a scalable top-down synthesis algorithm for the general FBS problem based on a heuristic different than that of [178], where we give preference to adding buffers over duplicating gates. Specifically, the main idea of the new approach is to duplicate gates *only if* the critical path delay would be increased otherwise. As we explain in Section 4.3, our heuristic exploits several improvement opportunities we identified in the algorithm of [178]. We also present an additional optimization step on top of the proposed top-down approach which can be used as a high-effort optimization step to obtain even better results. Our basic top-down heuristic achieves a 10.9% better area as compared to the state of the art [178] while the top-down approach with the additional optimization step allows an 11.82% improvement on average. Notably, the critical path delays of the resulting output networks of our top-down approaches are less than or equal to those obtained by the state of the art because they retain the same logic depth as the original fanout-unbounded network[I].

[I]Note that, to have a fair comparison with [178], we assume the primary inputs have unbounded fanout capacity.

Next, we consider the FBS with the additional requirement of *path-balancing*, which is a crucial constraint of several emerging technologies such as AQFP and QCA. In these technologies, gates can only drive at most one fanout, and special branching cells called splitters are required to support multiple fanouts. By considering the splitters as buffers with a fanout capacity of at least two, synthesizing for such technologies can be considered a special case of FBS. However, what makes synthesizing for these technologies more challenging is the constraints on the arrival times of fanins of a gate. For example, in AQFP technology, the signal propagation between gates is facilitated by a multi-phase clocking scheme, which requires all fanins of a gate to be clocked in the same phase (see Section 4.2 for details). One way to ensure this same-phase-fanins constraint is to require that all fanins of a gate be at the same logic level by adding extra buffers as necessary, which is referred to as path-balancing in the literature.

For FBS in the path-balanced case, we first adapt the aforementioned general-case-ILP to account for the path-balancing constraints considering both scenarios where gate duplications are enabled and disabled. (The latter setting has been studied as the AQFP splitter/buffer insertion problem in a series of research work [44, 61, 76, 97] as we describe in Section 4.3.)

Then, as with the general FBS problem, we present a scalable heuristic algorithm for path-balanced FBS focusing on AQFP technology. This algorithm starts with a top-down approach resembling our heuristic for the general setting (with some differences to avoid excessively duplicating gates) to determine initial gate/buffer counts, and then follows on with additional optimizations to mitigate the overhead of path-balancing buffers. Remarkably, as compared to the optimum delays in the setting *without* gate duplications, our heuristic *with* gate duplications achieves 8.76% better delays on average together with a 0.5% average area improvement.

In the rest of this chapter we first summarize some concepts useful to better understand our work including the logic network structures we use, timing and node equivalence concepts, and a brief introduction to AQFP technology (Section 4.2). Then we discuss some prior work on general FBS as well as splitter-buffer insertion for AQFP technology (Section 4.3). Next, in Section 4.4, we describe our ILP formulation for the general FBS, and in Section 4.5, we present our scalable top-down algorithm and related further optimizations. Following that, in Section 4.6, we extend our approaches from Section 4.4 and Section 4.5 to facilitate the path-balancing constraints. Finally, in Section 4.7, we present our experimental results, and in Section 4.8, we conclude with a brief discussion on the results and possible future directions.

4.2 Preliminaries

In this section, we briefly discuss the different logic representations used in our algorithms, the concept of static timing analysis for the unit delay model, and the notion of node equivalence. Additionally, we recall some key aspects of AQFP technology, which serve as the basis for demonstrating our FBS approaches in the path-balanced setting.

4.2.1 And-Inverter Graphs / Majority-Inverter Graphs

As discussed in Section 2.2, an *and-inverter graph* (AIG) is a *directed acyclic graph* (DAG) representation of logic where nodes represent either primary inputs or 2-input AND gates. Primary input nodes have an in-degree of zero, while AND gate nodes have an in-degree of two. Due to their simplicity and support for structural hashing, AIGs are the preferred logic representation in Section 4.4 and Section 4.5.

In Section 4.6, we focus on synthesis for AQFP technology, which is inherently based on majority gates. Consequently, the *majority-inverter graph* (MIG) is used as the preferred logic representation in that section. For more details on MIGs and their relevance to superconducting technologies, we refer the reader to Section 2.2.

4.2.2 Static Timing Analysis

In this work, we use the unit-delay model which assumes that a signal incurs a unit delay when it passes through a gate. The arrival time of a node n, denoted by t_n^{arr} is defined as follows: If n is a primary input, $t_n^{\text{arr}} = 0$. Otherwise $t_n^{\text{arr}} = 1 + \max_{m \in \text{FI}(n)} t_m^{\text{arr}}$, where $\text{FI}(n)$ denotes the set of fanin nodes of n. Note that the arrival time of a node is equal to the maximum length of a path from the node to any primary input. Hence, we sometimes use the term *level* to refer to the arrival time. The overall circuit delay (depth of the circuit) is defined as the maximum arrival time of any primary output.

For a given target delay D, the required time t_n^{req} of a node n is defined as follows: If n has no fanout nodes which are internal to the logic network (i.e., all fanouts are primary outputs), $t_n^{\text{req}} = D$. Otherwise, $t_n^{\text{req}} = \min_{m \in \text{FO}(n)} t_m^{\text{req}} - 1$, where $\text{FO}(n)$ denote the set of fanout nodes of node n.

A critical path in a network is an input-to-output path of nodes where each node n on the path satisfies $t_n^{\text{req}} = t_n^{\text{arr}}$. We say a node is critical if it lies on at least one critical path.

4.2.3 Node Equivalence

In general, we say two nodes m and n in a logic network are equivalent if their outputs are equal under all possible value combinations of primary inputs. If the input graph contains two or more equivalent nodes, their fanouts can be re-distributed among themselves at the discretion of a synthesis algorithm without altering the overall output of the circuit. However, for a network with many primary inputs, the computation needed to identify all sets of equivalent nodes can be prohibitively expensive. Thus, a more practical approach is to find equivalent nodes by considering a node's function with respect to a small cut, i.e., a set of nodes that separates the considered node from primary inputs. An example of this type of weaker equivalence checking is *structural hashing* which was originally used in IBM CAD tools [150]; For AIGs, a widely used structural hashing technique is to identify each gate with a signature consisting of the gate's fanins and flags denoting which fanins are inverted.

In this work, we do not explicitly check for equivalent nodes; instead, we allow the AIG data structure to internally use structural hashing to collapse any equivalent nodes. For the output logic network, our algorithms may explicitly duplicate some gates, hence we disable structural hashing for the output.

4.2.4 AQFP Logic Circuits

Adiabatic quantum-flux-parametron (AQFP) is a superconducting electronics technology with very low power consumption due to adiabatic operations. In AQFP, logic gates are constructed using superconductive inductors and *Josephson Junctions* (JJs) which are based on the *Josephson effect* [72]. The number of JJs in an AQFP circuit is commonly used as a proxy for the area cost.

As described in Section 2.4, the basic logic gates in AQFP are majority-3 gates, and the technology consists of clocked gates that are activated by a multi-phase clocking scheme. The fanouts of a gate are limited to one, and splitters are used to support multiple fanouts. To facilitate the proper signal propagation, all fanins of a gate must be at the same logic level. Thus all input-to-output paths must be balanced in terms of logic levels, which is referred to as path-balancing.

Depending on register implementations and clocking mechanisms, there can be different requirements on whether splitters are needed for primary inputs, whether path balancing is needed for primary inputs, and if path balancing is needed for primary outputs [143]. In our proposed FBS approaches for the path-balanced setting, we assume that splitters are needed for primary inputs (which is a notable difference from the general FBS setting where we assume primary inputs have unbounded fanout capacity to be consistent with [178]) and that path-balancing is needed for primary inputs and primary outputs (i.e., all primary outputs are at the same level).

To illustrate synthesis for AQFP under fanout and path-balancing constraints, consider the example logic network on the left of Fig. 4.1 and two of its fanout-bounded, path-balanced versions in the middle and on the right. The one in the middle does not have any duplicated gates while the one on the right has one gate duplication. In this example, duplicating gates benefits both the area and the delay; the delay is reduced by one logic level and the area is reduced by two JJs.

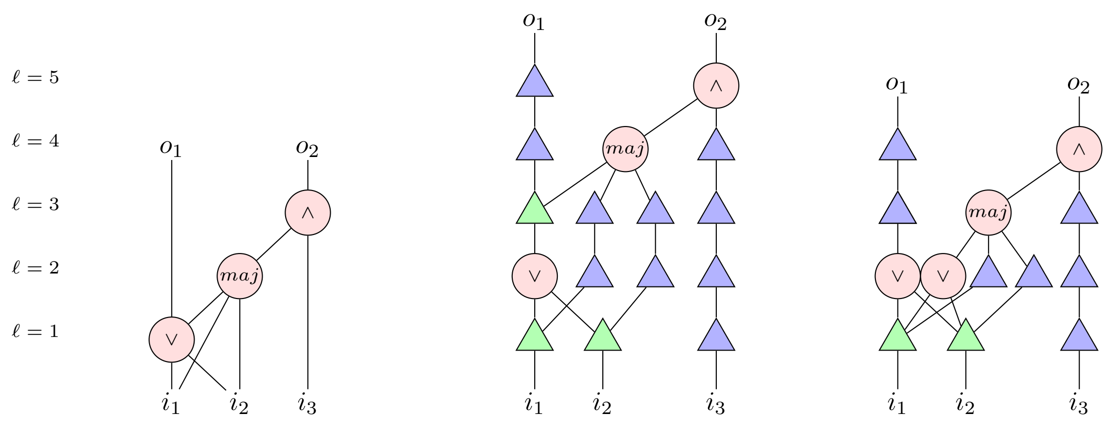


Figure 4.1: Example logic network (left) and two of its possible fanout-bounded, path-balanced versions targeting AQFP technology assuming a fanout capacity of 1 for gates and 3 for splitters. (Buffers and splitters are shown by triangles.) The version in the middle does not use any gate duplication whereas the version on the right allows gate duplication resulting in a reduction in both the overall number of logic levels as well as the total area.

4.3 Related Work

In this section, we first discuss some notable work related to FBS and briefly explain how our approach differs from the existing methods. Then, we also discuss some work related to the AQFP splitter/buffer insertion problem which can be viewed as a special case of FBS with path-balancing constraints.

4.3.1 General Fanout-Bounded Synthesis

An early theoretical work on general FBS using gate duplications and buffers by Hoover et al. [74] presented an algorithm that limits the number of fanouts of each gate by any given constant $c \geq 2$ at the expense of a constant factor increase in both the total number of gates and the depth[II]. Their algorithm assumes the natural setting that the input consists of bounded-*fanin* gates.

[II]The depth increase allows for an additive $O(\log_c(\#\text{ primary outputs}))$-term, which is unavoidable under constant-factor size increase considering a network with a single gate that feeds to a large number of primary outputs.

A vital ingredient of their work that is pertinent to FBS in general is the minimum-size minimum-height buffer tree construction. Namely, given the levels of fanouts of a gate, construct a tree consisting of the gate and a set of buffers such that 1) the gate is at the root, 2) the total number of buffers is minimized, and 3), the height of the tree is minimized. In the case where the gates and buffers have the same fanout bound $t \geq 2$, Golumbic [62] showed how to construct such a tree using a slightly modified Huffman-coding-like algorithm [80].

Recently, Zhang and Jiang [178] studied the problem of general FBS in the unit delay model and proposed an algorithm consisting of several heuristic optimizations. The main idea of their work is to duplicate gates if that results in a buffer reduction in the local neighborhood without significantly affecting the critical path delay. To this end, they proposed a recursive evaluation procedure to determine the number of duplicates for each gate. After the duplicate count for each gate has been determined, for each node in the reverse topological order, their algorithm constructs “skewed” buffer trees using an algorithm similar to [80]. Finally, for each set of equivalent nodes, their buffer trees are considered together and the load is re-distributed. This step does not alter the levels of the nodes but may remove some redundant equivalent nodes.

After further analyzing the algorithm of Zhang and Jiang, we identify the following optimization opportunities:

1. The computed numbers of gate duplicates in the recursive evaluation step do not guarantee that the fanout-bounded version achieves the same minimum possible logic depth as the original, fanout-unbounded network. (Note that the original depth is always achievable using gate duplicates under the assumption that the number of fanouts for a primary input is unbounded.)
2. The priority-queue-based method used in [178] for skewed buffer tree construction, although achieves the best possible size for the buffer tree, is *not* guaranteed to achieve the best possible level for the root node *unless* the fanout bound is two. However, for fanout bounds ≥ 3, it is always possible to obtain the best size for the buffer tree as well as the optimal level for the root node using the method proposed by Golumbic [62].
3. In [178], it is not stated how the fanouts are initially assigned to the duplicated copies prior to the skewed buffer tree construction or how their initial levels are determined. For instance, if all copies of a gate are naively placed at the same level when it is possible to place some copies at higher levels, the critical path delay can be adversely affected. However, it is difficult for an algorithm to make such decisions *unless* it already knows the levels of the fanouts.
4. The buffer forest re-balancing step does not guarantee that we get the minimum possible duplicate count (even locally for a considered set of equivalent nodes). This

is because the re-balancing step is run only *after* fixing the levels of the duplicated nodes.

In our scalable algorithm for general FBS, we capitalize on all these optimization opportunities. Specifically, by reconstructing the network in the reverse topological order, our algorithm has the full knowledge of the levels of fanouts of a gate, before the gate itself is synthesized. In Section 4.5, we describe in detail how our top-down approach enables exploiting each aforementioned opportunity.

4.3.2 Path-Balanced Fanout-Bounded Synthesis

As for the path-balanced setting, there is a line of work on satisfying fanout and path-balancing constraints for the AQFP technology (e.g., [44, 61, 97]), but these works mainly consider doing so without gate duplications. In literature, this problem is often referred to as the AQFP splitter/buffer insertion problem, and it is a special case of the path-balanced FBS.

In early work on AQFP splitter/buffer insertion, the main idea was to optimize individual fanout nets using different approaches such as dynamic programming and local retiming-like methods for pushing buffers from fanins to fanouts. The work of Lee et al. [97] took a rigorous approach where they presented an exact formulation of the problem as a *satisfiability modulo theory* (SMT) problem using the theory of integer linear arithmetic. Namely, they use the logic depth of each gate as an SMT variable and, for each fanout net, they consider constraints that must be satisfied by any valid splitter/buffer insertion. In contrast, our proposed method uses an ILP to encode the problem and uses the number of gate copies/buffers of each fanout net in each level as variables, which supports gate duplications.

The work in [97] also presented a more elaborate retiming algorithm where an initial splitter/buffer inserted network is further optimized by identifying collections of tightly-connected gates (chunks) where buffers can be pushed forward (from inputs to outputs) or vice-versa to reduce the buffer count. This retiming technique was later used in [44] for area recovery in delay optimal AQFP synthesis. More recently, Fu et al. [61] presented a dynamic programming approach to globally optimize splitters and buffers in AQFP synthesis and an ILP-based solution to approximate the optimum solution.

As a final remark, we emphasize the lack of gate duplications in the existing work on splitter/buffer insertion. However, duplicating gates is an important option that warrants increased attention because it can reduce both the area and the delay as we see in the example of Fig. 4.1.

4.4 Globally Optimum General Fanout-Bounded Synthesis

In this section, we present our ILP formulation of FBS in the unit-delay model. Given an input logic network, a predefined target logic depth D, the gate and buffer costs (e.g., area), and their respective fanout bounds, the proposed ILP finds the minimum cost logic network that meets all fanout bounds, has logic depth at most D, and is functionally equivalent to the input logic network.

We remark that we do not aim to make any logic restructuring; instead, our ILP determines how to duplicate gates and add buffers to the input logic network. For instance, consider the logic network shown on the left of Fig. 4.2 where the primary inputs $(i_1, \ldots, i_4)$ are shown on the bottom and the primary outputs $(o_1, \ldots, o_5)$ are at the top. If we assume gates and buffers both have fanout capacity 2, then one possible solution to the FBS problem is the network shown on the right, where we have two gates duplications (n_1 and n_3) and added two buffers (shown in blue triangles.)

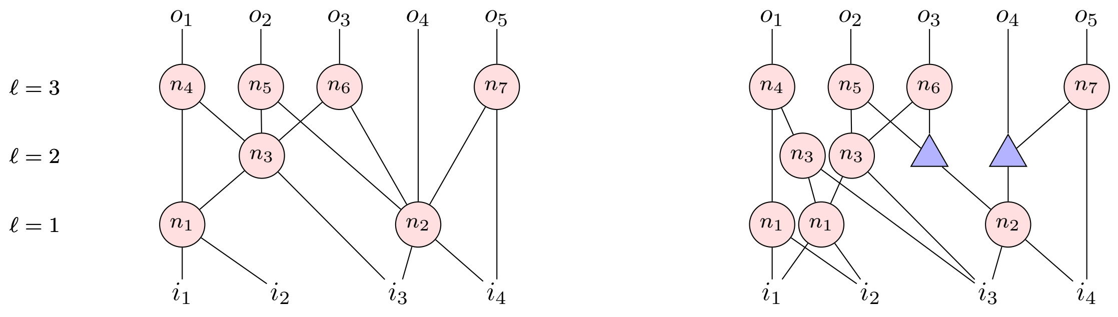


Figure 4.2: Example logic network (left) and a possible fanout-bounded version assuming a fanout limit of 2 (right).

To derive the ILP, we start with the following notation: Let I be the set of all primary inputs of the input network, let G be the set of all gates, and let $N = I \cup G$ be the set of all nodes. For example, in the example network shown in Fig. 4.2, $I = \{i_1, \ldots, i_4\}, G = \{n_1, n_2, \ldots, n_7\}$ and $N = \{i_1, \ldots, i_4, n_1, \ldots, n_7\}$.

For a node $n \in N$, let $\mathrm{FO}(n)$ be the collection of fanout nodes of n. Let k_n be the number of primary outputs directly connected to node n. Thus, for example, for the network in Fig. 4.2, we have $\mathrm{FO}(n_1) = \{n_3, n_4\}$ and $\mathrm{FO}(n_3) = \{n_4, n_5, n_6\}$, and $k_{n_2} = k_{n_4} = k_{n_5} = k_{n_6} = k_{n_7} = 1$.

Let c_{gate} be the cost (area) of a gate (we assume the network is homogeneous, but our ILP can easily be generalized to support different types of gates), let c_{buff} be the cost of a buffer, let f_{gate} be the fanout capacity of a gate, and let f_{buff} be the fanout capacity of a buffer.

For example, the setting studied in [178] for FBS assumed gates and buffers each have fanout capacity 2 and considered the optimization of the total node count. For this case, we thus have $f_{\text{gate}} = f_{\text{buff}} = 2$ and $c_{\text{gate}} = c_{\text{buff}} = 1$.

Let $n \in N$ be a node in the original graph. We say a node m in a fanout-bounded circuit is n-equivalent if one of the following holds:

1. n is a primary input and m is the corresponding primary input in the fanout-bounded version.
2. n is a gate with fanins n_1, n_2 and m is a gate with fanins m_1, m_2 such that m_1 is n_1-equivalent and m_2 is n_2-equivalent.
3. m is a buffer such that its fanin m_1 is n-equivalent.

Note that by the third criterion, any buffer in a buffer tree rooted at an n-equivalent gate is also n-equivalent. According to this definition, in the example fanout-bounded network (assuming $f_{\text{gate}} = f_{\text{buff}} = 2$) shown on the right of Fig. 4.2, there are two n_1-equivalent gates and two n_2-equivalent gates. Moreover, the two buffers represented as blue triangles in level 2 are n_2-equivalent.

Variables

We use two kinds of integer variables. For each node $n \in N$ and for each level $\ell \in \{1, \dots, D\}$, we introduce variables $g_{n,\ell}$ to denote the number of gate copies in level ℓ in the fanout-bounded circuit that are n-equivalent.

Similarly, we introduce variables $b_{n,\ell}$ to denote the number of buffers in level ℓ in the fanout-bounded circuit that are n-equivalent.

For example, for the logic network shown in Fig. 4.2, the introduced variables take the following values: $g_{n_1,1} = 2, g_{n_2,1} = 1, g_{n_3,2} = 2, g_{n_4,3} = 1, g_{n_5,3} = 1, g_{n_6,3} = 1, g_{n_7,3} = 1, b_{n_2,2} = 2$ and $g_{n,\ell} = 0$ for all unspecified variables $g_{n_q,\ell}$ with $q \leq 7$ and $\ell \leq 3$.

Constraints

Next, we introduce constraints to ensure that the values of variables indeed correspond to a valid fanout-bounded logic network that is equivalent to the input network. To this end, we first have that $g_{n,0} = 0$ and $b_{n,0} = 0$ for all $n \in N$ since there cannot be any gates or buffers in the same level as the primary inputs. (In fact, these variables are redundant and we can write the ILP without them, but having these variables with the above constraint makes it easier to specify the remaining constraints in a concise manner.) Next, consider a fixed level $\ell \in \{1, \dots, D\}$ and a fixed gate $n \in G$. We denote by $\text{avl}(n, \ell)$, which stands for "availability of n-equivalent signals by level ℓ," the total fanout capacity of all n-equivalent gates/buffers that are placed in levels strictly less than ℓ. Note that

$$\text{avl}(n, \ell) = \sum_{\ell'=0}^{\ell-1} (f_{\text{buff}} \cdot b_{n,\ell'} + f_{\text{gate}} \cdot g_{n,\ell'}),$$

which is a linear function of the ILP variables. We denote by $\text{req}(n, \ell)$, which stands for the "requirement of n-equivalent signals by level ℓ," the total fanout requirement of n-equivalent gates/buffers by all gates and buffers in level ℓ or below. Note that each copy of a fanout of an n-equivalent gate increases the fanout requirement by one, and each n-equivalent buffer also increases the fanout requirement by one. Namely, we can write

$$\text{req}(n, \ell) = \sum_{\ell'=1}^{\ell} \left(b_{n,\ell'} + \sum_{m \in \text{FO}(n)} g_{m,\ell'} \right),$$

which is again a linear function of the ILP variables.

Now, observe that, in any variable assignment that corresponds to a valid fanout-bounded network with depth D, it must hold that

$$\text{avl}(n, \ell) \geq \text{req}(n, \ell) \text{ for all } n \in G \text{ and } \ell \in 1, \ldots, D.$$

To see this, consider any valid depth-D fanout-bounded version of the input network, and let $g_{n,\ell}, b_{n,\ell}$ be the corresponding ILP variable values. Fix any gate $n \in G$ and let $\ell = 1$. Note that for any gate $m \in \text{FO}(n)$, $g_{m,1}$ must be 0. Otherwise, there must be a copy of n at level 0, which is a contradiction as n is not a primary input.

Similarly, there cannot be any n-equivalent buffer at level 1 either. Thus it must hold that

$$\text{avl}(n, 1) = 0 \geq 0 = \text{req}(n, 1).$$

Now, suppose that $\text{avl}(n, \ell) \geq \text{req}(n, \ell)$ must hold for any valid depth-D fanout-bounded version. We inductively show that $\text{avl}(n, \ell + 1) \geq \text{req}(n, \ell + 1)$ must also hold. Observe that the total number of connections between n-equivalent gates/buffers and their fanouts that must cross the boundary between level ℓ and $\ell + 1$ is at least

$$\sum_{m \in \text{FO}(n)} g_{m,\ell+1} + b_{n,\ell+1}.$$

The total remaining capacity of n-equivalent gates/buffers that are at levels below ℓ is $\text{avl}(n, \ell) - \text{req}(n, \ell)$. Thus the additional capacity needed to support all crossing connections must be provided by n-equivalent gates/buffer that are at level ℓ. Namely, we must have

$$\begin{aligned} f_{\text{gate}} \cdot g_{n,\ell} + f_{\text{buff}} \cdot b_{n,\ell} \geq \sum_{m \in \text{FO}(n)} & g_{m,\ell+1} + b_{n,\ell+1} \\ & - (\text{avl}(n,\ell) - \text{req}(n,\ell)), \end{aligned}$$

which yields

$$\begin{aligned} & \text{avl}(n,\ell) + f_{\text{gate}} \cdot g_{n,\ell} + f_{\text{buff}} \cdot b_{n,\ell} \\ & \qquad \geq \text{req}(n,\ell) + \sum_{m \in \text{FO}(n)} g_{m,\ell+1} + b_{n,\ell+1}, \end{aligned}$$

or equivalently, $\text{avl}(n,\ell+1) \geq \text{req}(n,\ell+1)$ after re-arranging.

Finally, we ensure that we have enough capacity remaining in n-equivalent gates/buffers to support the respective primary outputs (if any). Namely, for all n, it must hold that

$$\text{avl}(n, D+1) - \text{req}(n, D) \geq k_n.$$

The same can be achieved by *viewing* all fanouts connected to a gate n as n-equivalent buffers placed at level $D+1$, and simply adding the constraint

$$\text{avl}(n, D+1) \geq \text{req}(n, D+1).$$

We thus get the following ILP formulation for FBS under a predetermined depth bound D, where the objective function is to minimize the total area.

Minimize $\sum_{n \in G} \sum_{\ell=1}^{D} (c_{\text{gate}} \cdot g_{n,\ell} + c_{\text{buff}} \cdot b_{n,\ell})$,

Subject to

$$\begin{aligned} \text{avl}(n,\ell) - \text{req}(n,\ell) &\geq 0 & \forall\, n \in N, 1 \leq \ell \leq D, \\ \text{avl}(n, D+1) - \text{req}(n, D) &\geq k_n & \forall\, n \in N, \\ g_{n,0} &= 0 & n \in G, \\ b_{n,0} &= 0 & n \in N, \\ g_{n,\ell}, b_{n,\ell} &\in \mathbb{Z} & \forall\, n \in N, 1 \leq \ell \leq D. \end{aligned}$$

Let OPT be the optimum area of a fanout-bounded version of the input network with maximum depth D. Since any such valid network corresponds to a feasible solution for the ILP, it is clear that the value of ILP is at most OPT. We now give an algorithm (Algorithm 4.1) to transform any feasible ILP solution to a fanout-bounded network of maximum depth D, which is equivalent to the original network, thus showing that our ILP in fact finds the optimum area.

The algorithm first sorts all variables $g_{n,\ell}, b_{n,\ell}$ in the increasing order of ℓ. Then, considering the variable values in that order, construct the $g_{n,\ell}$ gate copies or $b_{n,\ell}$ buffers in a new network. To facilitate this construction, for each $n \in N$, the algorithm maintains a queue of currently constructed n-equivalent gates/buffers together with their remaining fanout capacities. Each time it uses such a gate/buffer, it decrements the count; once the count reaches zero, the corresponding gate/buffer instance is removed from the queue. Since the algorithms construct gates/buffers in a level-by-level fashion using a feasible variable assignment, we can see that the algorithm always has sufficient equivalent signals in the corresponding queues when executing Line 11 and Line 15.

4.5 Top-Down Heuristic Approach for General Fanout-Bounded Synthesis Problem

In this section, we first present our scalable top-down heuristic algorithm that greedily finds a feasible solution to the derived ILP. We then propose an additional optimization step that we can integrate with the top-down approach that allows further area reductions in certain cases.

Although solving the ILP introduced in Section 4.4 gives the optimum solution, solving it optimally for large networks which we often encounter in practice is a prohibitively expensive computation, and hence not a viable approach in many practical settings. On the other hand, the top-down approach we propose in this section is scalable to very large networks as it runs in $O(S \log S)$ time where S is the size of the input network (i.e., the number of wires in the network). Although this approach is not optimum in general, we note that it achieves optimum or near-optimum areas for several considered benchmarks in our experiments.

In the proposed approach, we consider the gates $n \in G$ in the reverse topological order, and for each n in this order, determine values for variables $g_{n,\ell}$ and $b_{n,\ell}$ such that the constraints

$$\mathrm{avl}(n, \ell) - \mathrm{req}(n, \ell) \geq 0$$

and

$$\mathrm{avl}(n, D+1) - \mathrm{req}(n, D) \geq k_n$$

are satisfied.

Algorithm 4.1: Algorithm for constructing a fanout-bounded network using a feasible solution to the ILP.

```
Input  : Input network ntk, parameters $f_{\text{gate}}, f_{\text{buff}}$, and a feasible ILP solution
         $g_{n,\ell}, b_{n,\ell}$ for $n \in N$ and $0 \leq \ell \leq D$.
Output : A fanout-bounded version of ntk.
1  Let newsig be a map from nodes in ntk to a queue of pairs (new node, remaining
     capacity)
2  for all $p \in$ primary inputs of ntk do
3      newsig[$p$].push((newntk.create_pi(), $\infty$))
4  Let data be an empty list.
5  for all nonzero $g_{n,\ell}$ do  Add ($\ell, n$, "gate") to data
6  for all nonzero $b_{n,\ell}$ do  Add ($\ell, n$, "buff") to data
7  Sort data in the ascending order of levels.
8  for all $(\ell, m, t) \in$ data in the ascending order of levels do
9      if $t$ = "gate" then
10         Look up fanins of $m$ in newsig.
11         newgate ← Create a new gate by choosing the first available equivalent fanins
             in newsig.
12         Decrement remaining capacity for used fanin nodes and remove them from
             the queue if remaining capacity reach zero.
13         newsig[$m$].push((newgate, $f_{\text{gate}}$))
14     else
15         newbuff ← Create a new buffer by choosing the first available equivalent node
             in newsig[$m$].
16         Decrement remaining capacity for the used fanin.
17         Pop from newsig[$m$] if remaining capacity is zero.
18         newsig[$m$].push((newbuff, $f_{\text{buff}}$))
19 return the constructed network.
```

Since we consider the nodes in the reverse topological order, when we consider a node n, we already know the levels of all fanouts of n-equivalent gates/buffers except for those fanouts that arise due to fanins of n-equivalent buffers. We call those fanouts *external fanouts* of n-equivalent gates/buffers.

When determining the values for $g_{n,\ell}$ and $b_{n,\ell}$, we prefer minimizing the number of gate duplicates by utilizing buffers as much as possible to support the fanout requirement. This decision is motivated by the following facts: First, duplicating a gate will increase the fanout requirement of other nodes: For example, suppose that n's fanins are m_1 and m_2. Then, duplicating a n-equivalent gate increases the fanout load of m_1 and m_2-equivalent gates/buffers. This is in contrast to adding a buffer which only increases the fanout load by one. Secondly, it is natural to assume that the area of a buffer is not more than that of a gate, and the fanout capacity of a buffer is usually more than that of a gate. Thus, in terms of area, replacing a gate copy with a buffer is always beneficial.

However, we cannot completely eliminate gate duplication because the addition of buffers can increase the number of logic levels (i.e., the critical path length). Recall that t_n^{arr} is the minimum level node n can be at even if we assume unbounded fanout capacities. Thus, for any $\ell < t_n^{\text{arr}}$, setting $g_{n,\ell}$ to a non-zero value makes the solution infeasible. Similarly, for any $\ell \leq t_n^{\text{arr}}$ (note the inclusion of equality), setting $b_{n,\ell}$ to a non-zero value also makes the solution infeasible.

For given levels of external fanouts of n-equivalent gates/buffers and the minimum possible level (i.e., t_n^{arr}) for an n-equivalent gate, we use Algorithm 4.2 to determine the values of $g_{n,\ell}$ and $b_{n,\ell}$ variables by considering each node in the reverse topological order. We then use Algorithm 4.1 to construct the corresponding fanout-bounded logic network.

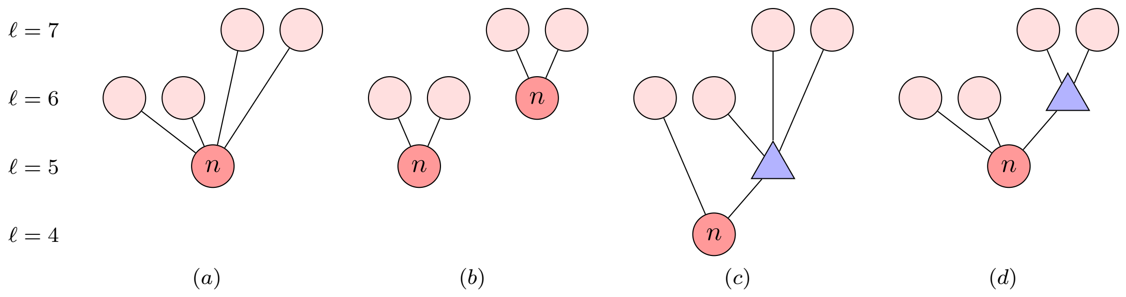


Figure 4.3: A fanout net for a node n with levels of fanouts already decided (a), two possible outcomes for the fanout net of n if the algorithm of [178] is used (b and c), and the optimum buffer tree for n (d) when $f_{\text{buff}} = f_{\text{gate}} = 3$ and $c_{\text{gate}} > c_{\text{buff}}$.

We remark that our top-down approach is fundamentally different from the work of Zhang and Jiang [178]. In [178], a set of n-equivalent gates and their corresponding levels are already determined when the buffer-forest re-balancing algorithm is run in order to reduce the number of gate duplicates. This can lead to some redundant gate copies that remain in the network even after re-balancing is performed.

In contrast, our algorithm uses Algorithm 4.2 to decide the set of n-equivalent gates that we absolutely need *along with their levels*, thus redundant gate copies are never created. Moreover, in the "skewed buffer tree construction" and "buffer-forest re-balancing" algorithms of [178], there can be situations where it does not construct the best buffer tree/forest when $f_{\text{gate}}, f_{\text{buff}} > 2$ and $c_{\text{gate}} > c_{\text{buff}}$. To see this, suppose that $f_{\text{gate}} = f_{\text{buff}} = 3$ and $c_{\text{gate}} > c_{\text{buff}}$ and consider the fanout net shown in Fig. 4.3 (a). The algorithm of [178] may either decide to duplicate node n and produce the forest shown

Algorithm 4.2: Algorithm for determining $g_{n,\ell}$ and $b_{n,\ell}$ values for a node $n \in N$, given ℓ_n^{min} and the levels of all external fanouts of n-equivalent gates/buffers.

Input : Input network ntk, parameters $f_{\text{gate}}, f_{\text{buff}}$, a node n, t_n^{arr}, and a list $folev_n$ of levels of n's fanouts.

Output : Values of $g_{n,\ell}, b_{n,\ell}$ variables for $\ell = 1, \ldots, D$.

```
1  Set $g_{n,\ell}, b_{n,\ell} = 0$ for all $\ell$
2  for $t$ = 1 to length($folev_n$) do
3      Let $rem \leftarrow length(folev_n) - t \cdot f_{\text{gate}}$
4      if $rem \leq 0$ then
5          for $i$ = 1 to length($folev_n$) in steps of $f_{\text{gate}}$ do
6              Increment $g_{n,folev_n[i]-1}$.
7              return variable values
8      $s \leftarrow rem \mod (f_{\text{buff}} - 1)$
9      if $s > 0$ then
10         Add $f_{\text{buff}} - s$ many copies of $\infty$ to $folev_n$ (i.e., dummy fanouts with
           unbounded required time).
11     Use the skewed buffer tree construction from [178] until we have $t$ buffer trees.
12     if the root levels of all buffer trees are at least $t_n^{\text{arr}}$ then
13         Set $g_{n,\ell}$ and $b_{n,\ell}$ according to the construction.
14         return variable values
```

in Fig. 4.3 (b) which has a cost of $2 \cdot c_{\text{gate}}$ or it may construct the skewed buffer tree shown in Fig. 4.3 (c) where the node n is placed at level 4. However, the buffer tree shown in Fig. 4.3 (d) is better than both the options; it has a lower area than the one in Fig. 4.3 (b) and gives a better placement for node n than the one in Fig. 4.3 (c). In contrast to [178], our algorithm always constructs the optimum buffer forest for given levels of external fanouts and t_n^{arr}. Namely, for $r = 1, 2, \ldots$, we consider r copies for the root gate, employ a modified version of the algorithm of Golumbic [62] to derive r buffer trees, and find the minimum value of r such that roots of all trees meet the arrival time requirement.

4.5.1 Improved Top-Down Approach with Over-Duplication

Recall that in our vanilla top-down approach, for each fanout net, we find the smallest buffer forest that does not increase the overall critical path length. The intuition behind settling for the smallest buffer forest is to minimize duplication of gates, and hence avoid unnecessarily increasing the load on the fanins of those gates.

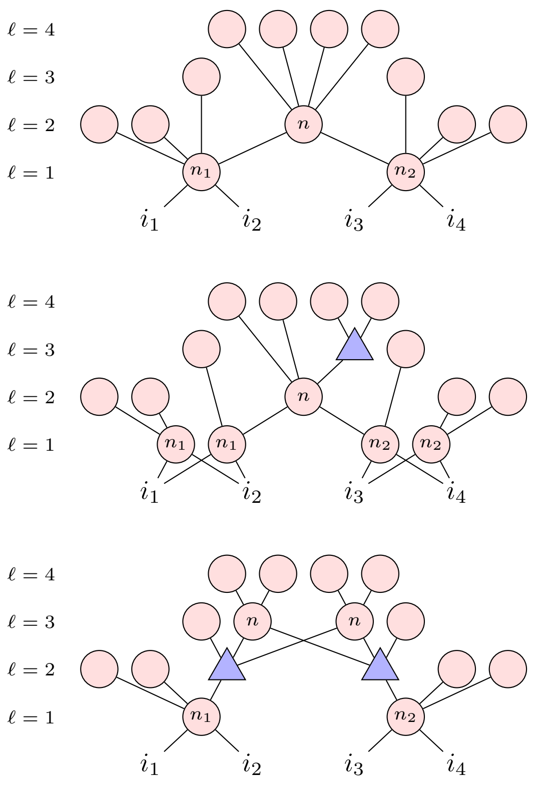


Figure 4.4: An intermediate step of fanout-bounded synthesis with levels decided for all nodes except n, n_1, n_2 (top), the synthesized fanout nets by the algorithm described in naive top-down approach (middle), and the synthesized fanout nets if over-duplication allowed (bottom) when $f_{\text{buff}} = f_{\text{gate}} = 3$ and $c_{\text{gate}} > c_{\text{buff}}$.

One potential drawback of this frugal approach is the following: Consider a scenario where we may have the option of placing two copies of a node n at level $t_n^{\text{arr}} + 1$. However, we may end up placing a single copy of n at level t_n^{arr} instead, thus forcing more duplications for n's fanin nodes as *their* fanout nets do not have enough slack to add buffers. To illustrate this point, assuming that $f_{\text{gate}} = f_{\text{buff}} = 3$ and $c_{\text{gate}} > c_{\text{buff}}$, consider the time our algorithm processes the fanout net of node n in the setting shown on top in Fig. 4.4 where the levels are already decided for all nodes except n, n_1 and n_2. Since the naive top-down approach greedily tries to minimize the number of duplicates for n, it will be placed at level 2 (no duplication) with one buffer at level 3 as shown in the middle of Fig. 4.4. This forces both n_1 and n_2 to be duplicated (unless the critical path length is to be increased) which results in an overall cost of $5 \cdot c_{\text{gate}} + c_{\text{buff}}$ for the fanout nets of n, n_1, and n_2. However, if we allow the locally suboptimal choice of duplicating n, it is possible to place two copies of n in level 3. This allows more room for fanout nets of n_1 and n_2 to have buffers, resulting in the outcome shown at the bottom of Fig. 4.4 with an overall cost of $4 \cdot c_{\text{gate}} + 2 \cdot c_{\text{buff}}$ (which is strictly a better cost when $c_{\text{gate}} > c_{\text{buff}}$). As such, allowing more duplicates than absolutely necessary (i.e., *over-duplication*) can be good if that provides more room for the fanins to have buffers and prevents them from being duplicated.

In an improved version of our top-down approach, we incorporate this idea of over-duplication as follows: For the fanout net of a considered node n, instead of stopping the algorithm at minimum possible number of trees t, we continue increasing t and construct the corresponding buffer forests. For each such buffer forest, we consider the overall area incurred by the fanout net of the considered node *and* the fanout nets of its fanin nodes, *assuming that we do not use over-duplication for those fanin nodes.* Then for node n, we choose the buffer forest that gives the minimum overall area computed in the above step.

There are two issues with this approach: First, due to the top-down implementation, when considering node n, all levels of its fanouts (including their potential copies) are known. However, for a fanin m of n, there can be some fanouts that are yet to be considered by the algorithm, and hence their final levels are not known. Secondly, suppose that a node m has k fanouts. For each of those fanouts, the cost of the fanout net of m will be re-evaluated multiple times. I.e., the fanout net of m is evaluated at least k-times. Since each evaluation also takes time at least linear in k, the total work involved in evaluating a node's fanout net can be very expensive for high-fanout nodes.

To circumvent the first issue, we propose to use a proxy level for the so-far unconsidered nodes; namely, we use their maximum possible level (i.e., the required time) as the proxy level. To mitigate the effects of the second issue, we set a constant bound F_{max} (e.g., 10) and ignore nodes with more than F_{max} fanouts when computing the overall area impact.

4.6 Path-Balanced Fanout-bounded Synthesis

In this section, we focus on FBS with the additional requirement of path-balancing.

4.6.1 ILP Formulation for the Global Optimum

Recall that the path-balancing constraint states that all input-to-output paths are of the same length. Equivalently, for a gate in level ℓ, all of its fanins must be in level $\ell - 1$. Thus for a gate or primary input n in the input network and for a level ℓ in the output network, it must hold the following: The total available fanout capacity of all n-equivalent nodes in level $\ell - 1$ must be at least the total required number of n-equivalent signals by nodes in level ℓ. We can easily incorporate this constraint into the ILP of Section 4.4 by simply redefining $\mathrm{avl}(n, \ell)$ and $\mathrm{req}(n, \ell)$ as

$$\mathrm{avl}(n, \ell) = f_{\mathrm{buff}} \cdot b_{n,\ell} + f_{\mathrm{gate}} \cdot g_{n,\ell},$$

and

$$\mathrm{req}(n,\ell) = b_{n,\ell} + \sum_{m\in \mathrm{FO}(n)} g_{m,\ell}.$$

As discussed in Section 4.2, the FBS with path-balancing constraints is a generalization of splitter/buffer insertion for AQFP technology, and AQFP technology can have different assumptions on the need for buffers/splitters on primary inputs and primary outputs. In particular, the requirement that *all* input-to-output paths must be of the same length falls under the assumption that both primary inputs and primary outputs need path-balancing.

However, our ILP is versatile as it can be adapted to different AQFP-technology-specific assumptions. For example, we can remove the path-balancing requirement on primary inputs by retaining the definitions of $\mathrm{avl}(n,\ell)$ and $\mathrm{req}(n,\ell)$ from Section 4.4 for nodes $n \in I$, i.e., the primary inputs. Similarly, we can remove the path-balancing requirement on primary outputs by retaining those definitions only in the constraint $\mathrm{avl}(n, D+1) - \mathrm{req}(n, D) \geq k_n$. Moreover, if we need to also enforce that primary inputs need splitters to support multiple fanouts, we can add constraints dictating $g_{n,0} = 1$ and $g_{n,\ell} = 0$ for all $n \in I$ and $\ell > 0$. (In the ILP for the general fanout-bounded setting with no fanout limit on primary inputs, we simply omitted these constraints. This allows the ILP solver to place as many copies of primary inputs anywhere in the network, which is effectively equal to assuming unbounded fanout capacity.)

In addition to supporting the different AQFP-specific assumptions, we can also change the ILP to match the original splitter/buffer insertion problem where duplicating gates is not an option. To this end, we simply have to introduce a new constraint that $\sum_{1\leq\ell\leq D} g_{n,\ell} = 1$ for all gates $g \in G$.

4.6.2 Scalable Heuristic Approach

In the path-balanced setting, we need buffers not only to support multiple fanouts, but also to ensure that all input-to-output paths are of the same length. If we naively use the same top-down approach from the general FBS for the path-balanced setting, it can unnecessarily increase the area due to path-balancing buffers. To see this, suppose that we have a gate n whose arrival time is 1, but its only fanout is determined to be in level 3 by our top-down algorithm. In this case, the algorithm prefers to keep n in level 2 (as opposed to 1) because the main idea of the algorithm from Section 4.5 was to keep gates in the highest level possible to give sufficient room for its fanins to have buffers. Now suppose that n's fanins have no other fanouts, in which case, we will need two path-balancing buffers at n's fanins. However, if we placed n in level 1 instead, we could only use one path-balancing buffer at n's output. In general, the situation can be much worse: for example, we could have a block of logic that has k_1 outputs and k_2 inputs in place of n. If $k_1 < k_2$, moving the whole logic block down by 1 level can save $k_2 - k_1$ buffers. On the other hand, if $k_2 < k_1$, then the algorithm's choice to keep the logic block in the highest possible level is meaningful.

Taking such scenarios into account, for the path-balanced setting, we start with a top-down approach similar to Section 4.5 to determine initial gate/buffer counts in different levels (i.e., values for variables $g_{n,\ell}$ and $b_{n,\ell}$), but we then perform an additional optimization to modify these gate/buffer counts to further reduce the area. To this end, we first identify (gate, level)-pairs that may correspond to potential path-balancing buffers. If all fanins of a gate are path-balancing buffers, we can push the buffers towards the output of the gate. In general, this can be done on blocks of logic whose inputs all correspond to path-balancing buffers.

This kind of retiming techniques have already been considered in the past [44, 97], but they work on existing AQFP netlists. Our proposed method is more general and works on gate/buffer counts in each level, *before* constructing the netlist, and hence it is able to capture more retiming opportunities. To illustrate, consider the part of a netlist shown in Fig. 4.5 (b), where we assume that splitter fanout capacity is 2 for the sake of simplicity. The existing retiming techniques can optimize this by moving node a one level down to obtain the configuration in Fig. 4.5 (c), saving one buffer in the process. However, these algorithms fail to identify an optimization opportunity in for the configuration in Fig. 4.5 (a) because one of the fanins of node a is not a path-balancing buffer but a splitter. Our approach, instead works on gate/buffer counts in each level and hence is able to identify the optimization opportunity in both scenarios. Namely, for each fanin x of node a, we check if we have a potential path-balancing buffer by checking

1. if we have x-equivalent buffers in the lower level,
2. and if we can isolate one path-balancing buffer (a buffer with fanout one) from those.

To check the first condition for a fanin x of node a in level ℓ, we check if $b_{x,\ell-1} > 0$. For the second condition, we check whether the remaining x-equivalent nodes in level $\ell - 1$ can still satisfy the requirement of remaining fanouts of x in level ℓ *after dedicating a single x-equivalent buffer to supply node* a*'s fanin*; namely, we check if $\mathrm{avl}(x, \ell - 1) - f_{\mathrm{buff}} \geq \mathrm{req}(x, \ell) - 1$. After optimizing the gate/buffer counts with this improved retiming step, we construct an AQFP netlist using Algorithm 4.1. Then we also run the state-of-the-art retiming from [44] on the constructed circuit to further optimize buffer counts.

In addition to the retiming, our initial top-down heuristic has some minor differences with respect to Section 4.5. Namely, when computing the arrival times for signals, if the originating gate of a signal has more than one fanout, we assume a delay of 2 (instead of 1)

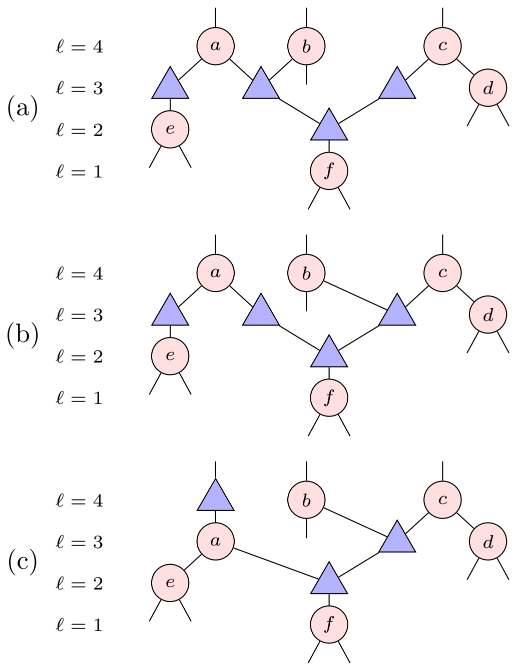


Figure 4.5: Two possibilities for a part of an AQFP netlist and their retimed version.

accounting for an additional splitter at the output of that gate. This is an AQFP-specific setting: in AQFP, the gates can only support one fanout, and if we always assume a delay of 1 for a gate, the top-down algorithm can end up excessively duplicating gates to meet this delay bound. However, if a gate is in the critical path, has only two fanouts, and if its fanins will have splitters added at their outputs (i.e., they have multiple fanouts), then it is likely that we may be able to duplicate the gate with only a small additional cost. So for such gates, we take the delay to be 1 when computing the arrival time.

4.7 Experimental Results

In this section, we present the experimental results obtained from our ILP formulations and heuristic FBS algorithms for both the general and path-balanced settings. All our experiments were run on a MacBook Pro M1 with 10 cores of CPU, 16 cores of GPU, and 32 GB of RAM.

Note that in all our experiments for the general FBS setting, the benchmarks are preprocessed with a single round of *resyn2* command in ABC [34], to do a fair comparison with prior work [178]. No such preprocessing was done in experiments for the path-balanced FBS setting.

4.7.1 Global Optimum for General Fanout-Bounded Synthesis

First, for a set of small benchmarks, we use the ILP to find the global optimum solutions; Namely, using the minimum possible circuit delay as the delay bound, we write the ILP introduced in Section 4.4, and then solve it using the Gurobi optimizer [64]. In the ILP formulation, we use the same setting as [178] where we have fanout capacity 2 and unit-area for both AND gates and buffers.

Table 4.1: The global optimums for general fanout-bounded synthesis.

	Input network		**Output network**				
Benchmark	And gates	Levels	And gates	Buffers	Total gates (Area)	Levels	Time(s)
adder	1019	255	1021	126	1147	255	8538.26
bar	3141	12	3901	0	3901	12	242.97
cavlc	662	16	733	13	746	16	12.14
ctrl	108	8	123	3	126	8	0.24
dec	304	3	768	0	768	3	0.21
i2c	1162	15	1255	113	1368	15	59.27
int2float	214	15	224	7	231	15	2.76
router	177	19	180	5	185	19	1.52
adder1	7	4	7	0	7	4	0.01
adder8	77	17	78	7	85	17	0.37
mult8	439	35	447	13	460	35	1129.73
counter16	49	13	55	4	59	13	0.10
counter32	125	19	139	11	150	19	2.55
counter64	285	25	311	28	339	25	11.71
counter128	613	31	650	76	726	31	67.21
c17	6	3	6	0	6	3	0.03
c432	121	26	136	6	142	26	1.92
c499	387	18	410	42	452	18	8.15
c880	306	27	322	28	350	27	16.84
c1355	388	17	412	44	456	17	3.92
c1908	286	21	318	32	350	21	6.31
c2670	169	9	178	9	187	9	0.33
c3540	789	32	905	127	1032	32	3521.49
c5315	1294	26	1403	118	1521	26	553.95
c7552	1385	33	1562	192	1754	33	1335.10
sorter32	480	15	512	0	512	15	4.31
sorter48	984	25	984	64	1048	25	68.47

The results are shown in Table 4.1 where the first 8 benchmarks are from the EPFL logic synthesis benchmarks suite [11] and the rest of the benchmarks are a subset of those used in [76].

4.7.2 Heuristics for General Fanout-Bounded Synthesis

Next, we evaluate our top-down FBS approaches on the benchmarks of [76] and on EPFL benchmarks [11].

Table 4.2: Results of the top-down fanout-bounded synthesis on benchmarks of [76].

	Top-down approach			**Top-down approach with over-duplication**		
Benchmark	And gates	Buffers	Total gates (Area)	And gates	Buffers	Total gates (Area)
adder1	7	0	7	7	0	7
adder8	77	8	85	77	8	85
mult8	441	19	460	441	19	460
counter16	52	7	59	52	7	59
counter32	130	20	150	130	20	150
counter64	298	41	339	298	41	339
counter128	638	88	726	638	88	726
c17	6	0	6	6	0	6
c432	132	13	145	134	9	143
c499	409	44	453	410	42	452
c880	306	47	353	306	47	353
c1355	412	44	456	414	42	456
c1908	314	44	358	314	44	358
c2670	172	18	190	172	18	190
c3540	819	256	1075	825	237	1062
c5315	1311	288	1599	1378	153	1531
c6288	1903	7	1910	1903	7	1910
c7552	1393	420	1813	1422	364	1786
sorter32	512	0	512	512	0	512
sorter48	984	64	1048	984	64	1048
alu32	1512	434	1946	1513	432	1945

For benchmarks of [76], we present the results in Table 4.2. As we see, our initial *top-down approach* already achieves the optimum on several benchmarks. Our *top-down approach with over-duplication* performs even better and achieves results that are optimum or closer to optimum on some additional benchmarks. We recall that both our approaches do not increase the number of logic levels of the input network (computed with no restrictions on the fanout capacity of gates).

For EPFL benchmarks, we present the results in Table 4.3 together with the results of [178] for a comparison. We remark that the measure of quality of results (QoR) used in [178] is slightly different, and if we were to use their QoR measure on our results, our

Table 4.3: Results of the top-down fanout-bounded synthesis algorithm on EPFL benchmarks.

	Input network		**Output of [178]**		**Output (top-down)**					**Output (top-down with over-duplication)**				
Benchmark	And gates	Levels	Total gates (Area)	Levels	And gates	Buffers	Total gates (Area)	Area Impr.%	Time (s)	And gates	Buffers	Total gates (Area)	Area Impr.%	Time (s)
adder	1019	255	1273	255	1020	128	1148	9.82	0.00	1020	128	1148	9.82	0.08
arbiter	11839	87	22911	87	11839	10176	22015	3.91	0.01	11839	10176	22015	3.91	0.04
bar	3141	12	3901	12	3425	952	4377	-12.20	0.00	3901	0	3901	0.00	0.01
cavlc	662	16	840	16	663	128	791	5.83	0.00	677	100	777	7.50	0.00
ctrl	108	8	147	8	108	26	134	8.84	0.00	114	14	128	12.93	0.00
dec	304	3	768	3	768	0	768	0.00	0.00	768	0	768	0.00	0.00
div	40772	4361	79413	4365	41087	12126	53213	32.99	0.04	41131	12038	53169	33.05	1.72
hyp	211330	24794	332744	24817	211458	45199	256657	22.87	0.20	212237	43641	255878	23.10	41.01
i2c	1162	15	1530	15	1162	264	1426	6.80	0.00	1171	247	1418	7.32	0.01
int2float	214	15	251	15	214	23	237	5.58	0.00	216	19	235	6.37	0.00
log2	29370	376	56617	376	29893	15018	44911	20.68	0.03	29857	15045	44902	20.69	1.08
max	2834	204	4157	206	3094	997	4091	1.59	0.00	3096	993	4089	1.64	0.09
mem_ctrl	45614	110	63788	110	45662	15326	60988	4.39	0.04	46140	14642	60782	4.71	2.18
multiplier	24556	262	31930	262	24567	7011	31578	1.10	0.02	24618	6909	31527	1.26	0.90
priority	676	203	795	203	676	59	735	7.55	0.00	676	59	735	7.55	0.05
router	177	19	222	19	177	8	185	16.67	0.00	177	8	185	16.67	0.00
sin	5039	177	10329	178	5415	2747	8162	20.98	0.01	5431	2677	8108	21.50	0.13
sqrt	19437	4968	32141	5449	20152	9432	29584	7.96	0.02	20152	9432	29584	7.96	0.65
square	16623	248	27556	248	16625	1533	18158	34.11	0.01	16720	1343	18063	34.45	1.44
voter	9756	57	13158	58	9810	1185	10995	16.44	0.01	9810	1185	10995	16.44	0.06
sixteen	11976864	99	24461292	99	11976864	9510308	21487172	12.16	23.61	12084231	9443891	21528122	11.99	527.31
twenty	15317374	86	31481612	86	15317374	12493285	27810659	11.66	29.84	15460597	12411371	27871968	11.47	520.78
twentythree	17168790	94	35358029	94	17168790	14056097	31224887	11.69	32.97	17316727	13968865	31285592	11.52	655.45
Average								10.93					11.82	

approach would score even higher. Namely, the QoR measure used in [178] is

$$\frac{size(G)}{size(G')} + \frac{depth(G)}{depth(G')}$$

where G is the original input network and G' is the fanout-bounded version produced by the algorithm. In our approach, the depths of G and G' are always equal, whereas in [178], $depth(G) \leq depth(G')$ with strict inequality for some benchmarks (e.g., see the results for benchmark “sqrt”).

In our top-down approach without over-duplication, the average improvement over all standard EPFL benchmarks is 10.93%. However, for the benchmark “bar”, our algorithm’s result is 12.2% worse. Remarkably, combining the top-down algorithm with the over-duplication step from Section 4.5.1 achieves the same results as [178] for that benchmark, while increasing the average improvement over all EPFL benchmarks to 11.82%. Notably, our method results in fanout-bounded circuits that are much closer to the optimum results on several benchmarks (e.g., adder, cavlc, int2float, and router).

4.7.3 Global Optimum Splitter/Buffer Insertion for AQFP

In this section, we present the results of our ILP-based global optimization algorithm for the FBS in the path-balanced setting targeting the AQFP technology. To this end, we set $f_{\text{buff}} = 4$ and $f_{\text{gate}} = 1$ to capture the fanout constraints commonly used in prior work on the AQFP technology. We use the number of JJs as the area cost, and hence we set $c_{\text{gate}} = 6$ and $c_{\text{buff}} = 2$. Recall that, according to the ILP formulation, the *global optimum* means the minimum area for a fixed depth.

In our experiments, we consider two scenarios: one without gate duplications and one with gate duplications. To the best of our knowledge, *no* prior work on AQFP splitter/buffer insertion considers gate duplications.

In Table 4.4, we present our optimum results on the same benchmarks used by [76] for the case with no gate duplicates and compare them with the results of four prior work [44, 61, 76, 97] in the same setting. In this experiment, we use the minimum achievable delay without duplicating gates as the target depth bound. In the table, the optimum area for the target depth is shown in blue. The term “opt" in the last columns means that the ILP solver was able to find the optimum solution. On the other hand, the term “tle" (time-limit-exceeded) means that the solver failed to find the optimum solution within a given time limit of 300 seconds, so the presented results for "tle" rows are based on a tentative feasible solution found by the solver. Note that having the global optimum results in this setting allows for an objective evaluation of other heuristic algorithms.

In Table 4.5 and its continuation Table 4.6, we present the optimum results obtained

Table 4.4: Results of AQFP splitter/buffer insertion without gate duplication.

	Input network		**ICCAD'21 [76]**			**DAC'22 [97]**			**ASP-DAC'23 [44]**			**ASP-DAC'23 [61]**			**Global optimum**			
Benchmark	Area	Levels	#B/S	#JJ	Levels	#B/S	#JJ	Levels	#B/S	#JJ	Levels	#B/S	#JJ	Levels	#B/S	#JJ	Levels	opt/tle
adder1	7	4	16	74	8	16	74	8	16	74	8	-	-	-	16	74	8	opt
adder8	77	17	371	1204	33	371	1204	33	372	1206	33	-	-	-	371	1204	33	opt
mult8	439	35	1833	6300	70	1869	6372	71	1688	6010	70	1681	5996	70	1724	6082	70	tle
counter16	29	9	82	338	17	65	304	17	65	304	17	66	306	17	65	304	17	opt
counter32	82	13	189	912	23	155	802	23	154	800	23	156	804	23	154	800	23	opt
counter64	195	17	419	2134	30	352	1874	30	347	1864	30	351	1872	30	347	1864	30	opt
counter128	428	22	895	4652	38	760	4088	38	747	4062	38	755	4078	38	747	4062	38	opt
c17	6	3	12	60	5	12	60	5	12	60	5	-	-	-	12	60	5	opt
c432	121	26	837	2406	37	874	2474	38	839	2404	37	829	2384	37	829	2384	37	opt
c499	387	18	1251	4858	30	1275	4872	31	1173	4668	29	1173	4668	29	1173	4668	29	opt
c880	306	27	1723	5296	40	1703	5242	41	1511	4858	40	1536	4908	40	-	-	-	tle
c1355	389	18	1216	4784	29	1290	4914	31	1184	4702	29	1186	4706	29	1178	4690	29	opt
c1908	289	21	1505	4810	35	1298	4330	35	1236	4206	34	1253	4240	34	1232	4198	34	opt
c2670	368	21	2055	7392	27	2132	6472	30	1932	6072	28	1869	5954	28	1794	5796	28	opt
c3540	794	32	2395	9610	53	2266	9296	55	1972	8708	52	1963	8690	52	1926	8516	52	tle
c5315	1302	26	6447	20854	41	6026	19864	42	5646	19104	40	5505	18942	40	6260	20332	42	tle
c6288	1870	89	9297	29814	179	9893	31006	180	9009	29238	179	8832	28884	179	-	-	-	tle
c7552	1394	33	8342	25140	59	8759	25882	66	7505	23374	56	6768	21908	58	-	-	-	tle
sorter32	480	15	480	3840	30	480	3840	30	480	3840	30	-	-	-	480	3840	30	opt
sorter48	880	20	880	7040	35	880	7040	35	880	7040	35	-	-	-	880	7040	35	opt
alu32	1513	100	17178	43574	170	14655	38388	171	13837	36752	169	13976	37030	169	-	-	-	tle

Table 4.5: Results of AQFP splitter/buffer insertion with gate duplication (part i).

Benchmark	Gates	Buffers	#JJ	Levels	Time (s)	opt/tle
adder1	7	16	74	8	0.04	opt
adder1	8	13	74	7	0.03	opt
adder1	9	10	74	6	0.03	opt
adder8	82	352	1196	33	305.20	tle
adder8	81	347	1180	32	304.82	tle
adder8	81	337	1160	31	304.61	tle
adder8	81	328	1142	30	304.28	tle
adder8	81	320	1126	29	304.01	tle
counter16	29	65	304	17	21.47	opt
counter16	31	57	300	16	17.46	opt
counter16	32	53	298	15	6.48	opt
counter16	34	47	298	14	1.28	opt
counter16	36	45	306	13	0.95	opt
counter32	82	154	800	23	303.32	tle
counter32	84	149	802	22	303.12	tle
counter32	88	133	794	21	302.83	tle
counter32	92	121	794	20	302.58	tle
counter32	99	116	826	19	302.34	tle
counter64	195	347	1864	30	328.99	tle
counter64	198	335	1858	29	327.17	tle
counter64	203	323	1864	28	325.26	tle
counter64	209	299	1852	27	323.59	tle
counter64	225	284	1918	26	321.81	tle
counter128	431	742	4070	38	515.67	tle
counter128	439	730	4094	37	505.04	tle
counter128	440	706	4052	36	494.14	tle
counter128	455	662	4054	35	483.08	tle
counter128	458	651	4050	34	474.59	tle
c17	6	12	60	5	0.04	opt
c17	7	8	58	4	0.02	opt
c432	122	822	2376	37	316.69	tle
c432	122	793	2318	36	315.87	tle
c432	123	784	2306	35	314.97	tle
c432	130	775	2330	34	314.12	tle
c432	135	757	2324	33	313.41	tle
c499	396	1123	4622	29	386.41	tle
c499	398	1086	4560	28	380.61	tle
c499	400	1051	4502	27	375.08	tle
c499	403	1007	4432	26	369.49	tle
c499	444	962	4588	25	364.10	tle

Table 4.6: Results of AQFP splitter/buffer insertion with gate duplication (part ii).

Benchmark	Gates	Buffers	#JJ	Levels	Time (s)	opt/tle
c2670	370	1825	5870	28	398.12	tle
c2670	372	1793	5818	27	391.00	tle
c2670	378	1749	5766	26	384.23	tle
c2670	382	1699	5690	25	378.11	tle
sorter32	566	481	4358	30	435.07	tle
sorter32	534	450	4104	29	426.82	tle
sorter32	553	417	4152	28	420.74	tle
sorter32	576	384	4224	27	410.68	tle
sorter32	608	352	4352	26	402.67	tle
sorter48	896	896	7168	35	913.68	tle
sorter48	896	848	7072	34	875.92	tle
sorter48	944	816	7296	33	841.85	tle
sorter48	960	756	7272	32	807.86	tle
sorter48	1008	704	7456	31	778.18	tle

Table 4.7: Results of scalable heuristic approach for AQFP.

Benchmark	Gates	Buffers	#JJ	Levels	Time (s)	Area Impr. %	Delay Impr.%
adder1	9	10	74	6	0	0.00	25.00
adder8	84	285	1074	26	0.01	10.95	21.21
mult8	469	1543	5900	58	0.06	1.83	17.14
counter16	29	65	304	17	0	0.00	0.00
counter32	82	154	800	23	0.01	0.00	0.00
counter64	195	347	1864	30	0.01	0.00	0.00
counter128	428	747	4062	38	0.03	0.00	0.00
c17	6	12	60	5	0	0.00	0.00
c432	139	821	2476	36	0.02	-3.00	2.70
c499	391	1131	4608	28	0.04	1.29	3.45
c880	306	1545	4926	40	0.08	-1.40	0.00
c1355	392	1151	4654	28	0.03	1.02	3.45
c1908	293	1176	4110	32	0.05	2.28	5.88
c2670	371	2172	6570	27	0.09	-8.20	3.57
c3540	797	1930	8642	50	0.15	0.76	3.85
c5315	1302	5710	19232	40	0.23	-0.67	0.00
c6288	1908	8468	28384	163	0.31	2.92	8.94
c7552	1400	9163	26726	54	0.61	-14.34	3.57
sorter32	704	256	4736	23	0.01	-23.33	23.33
sorter48	912	816	7104	33	0.04	-0.91	5.71
alu32	1543	11459	32176	139	0.65	12.45	17.75
Weighted average compared to [44]						0.51	8.76

considering *different target logic depths* on the same benchmarks for the setting with gate duplicates, which can be used to evaluate future algorithms in this setting. To obtain these results, we start with the minimum delay achievable without gate duplications as the target delay, and proceed with gradually decreasing the target delay. In the table, for each benchmark, the minimum observed is shown in blue where the ties are broken using the overall delay. Note that these results serve as a proof-of-concept that allowing gates duplications can help improve both the area and delay in AQFP synthesis.

4.7.4 Heuristic Splitter/Buffer Insertion for AQFP

Finally, we run our scalable heuristic algorithm for path-balanced FBS on the same benchmarks used by [76] and compare our results with the latest scalable algorithm for AQFP splitter/buffer insertion [44] in Table 4.7. For all benchmarks, our approach achieves the same or significantly better delays as compared to the optimum delay achieved by the method in [44]. For some benchmarks with significant delay improvements, there is a considerable area overhead which is likely caused by duplicated gates. However, some other benchmarks with higher delay improvements show considerable area improvements as well, which can be attributed to the decrease in path-balancing buffers. The average delay improvement of our approach is 8.76% while the average area improvement is 0.5%. Notably, our heuristic algorithm achieves more than 17% delay improvements on several benchmarks.

4.8 Summary

In this work, we took a rigorous approach for the FBS of circuits in the unit-delay model. To this end, we formulated the problem of FBS for fixed target delay as an ILP and obtained the global optimum solutions for a number of benchmarks. We then showed how to find a feasible solution to the ILP using a scalable top-down approach while mitigating some shortcomings of earlier work. As compared to the known best results for this problem, our algorithm produced an 11.82% improved area while achieving matching or better delays.

As we see in Section 4.7, the over-duplication heuristic with a local cost function improved the area reduction. It will be interesting to find a more elaborate but efficiently computable cost function for evaluating heuristic choices such as the one we introduced in Section 4.5.1. We also believe that a deeper analysis of the benchmark “bar” might hint at what kind of real-world circuit patterns benefit more from such heuristics.

We extended both our optimum and heuristic approaches to the setting with path-balancing constraints and demonstrated their effectiveness considering the splitter/buffer insertion problem in the AQFP technology. Our globally optimum results considering different target depths for the setting with gate duplications show that there exists a large gap in existing AQFP splitter/buffer insertion techniques. Remarkably, our scalable heuristic algorithm for this setting was able to exploit many optimization opportunities by considering the duplication of gates on critical paths. In particular, several benchmarks showed over 17% delay improvements under our method including two benchmarks (adder8, alu32) that also showed over 10% area improvements. However, comparing the results of our heuristic with the globally optimum solutions, it is clear that there are many opportunities for further improvements.

Considering these promising findings, we envision that FBS will have a bigger role to play in logic synthesis for emerging technologies with unconventional design constraints, and we hope that our work will motivate more research in this direction that would ultimately lead to better heuristics. The globally optimum solutions presented in this work can serve as the ground truth for evaluating such heuristics.

5 Logic Synthesis for AQFP Technology

Superconducting electronic (SCE) circuits are getting increasingly popular in the electronics industry due to their low energy consumption and high-speed operation. The growing interest in SCE is further fueled by the escalating challenges and higher costs of transistor downscaling in traditional CMOS technologies. The potential of SCE to revolutionize the electronics industry is widely recognized, as evidenced by the growing involvement of the EDA industry in developing tools and synthesis flows for SCE, supported by government-funded programs such as IARPA's SuperTools program [60, 173].

However, the EDA support for SCE is far from reaching the maturity level of CMOS EDA tools, and there are still many challenges to overcome before SCE can be widely adopted in the industry. Due to their unique interconnect requirements, the existing made-for-CMOS synthesis tools often result in suboptimal circuits with a high interconnect overhead. In this chapter, we propose the simultaneous optimization of logic and interconnect resources in SCE technologies, focusing on Adiabatic Quantum-Flux-Parametron (AQFP) circuits.

The work presented in this chapter is based on a paper [110] published in the International Symposium on Nanoscale Architectures 2021, which was also presented in the International Workshop on Logic Synthesis (IWLS).

5.1 Introduction

SCE circuits are based on superconductive inductors and Josephson Junctions (JJs) [73], and there are several families of SCE circuits. The examples include Rapid Single Flux Quantum (RSFQ) [104], Energy-efficient SFQ (eSFQ) [130], Reciprocal Quantum Logic (RQL) [70], Low-Voltage RSFQ (LV-RSFQ) [159], Dynamic Single Flux Quantum (DSFQ) [91] etc., and Adiabatic Quantum-Flux-Parametron (AQFP) [157]. While most of such families use DC-biased junctions which cause static power dissipation, the technologies such as AQFP achieve superior energy-efficiency using AC-biased junctions,

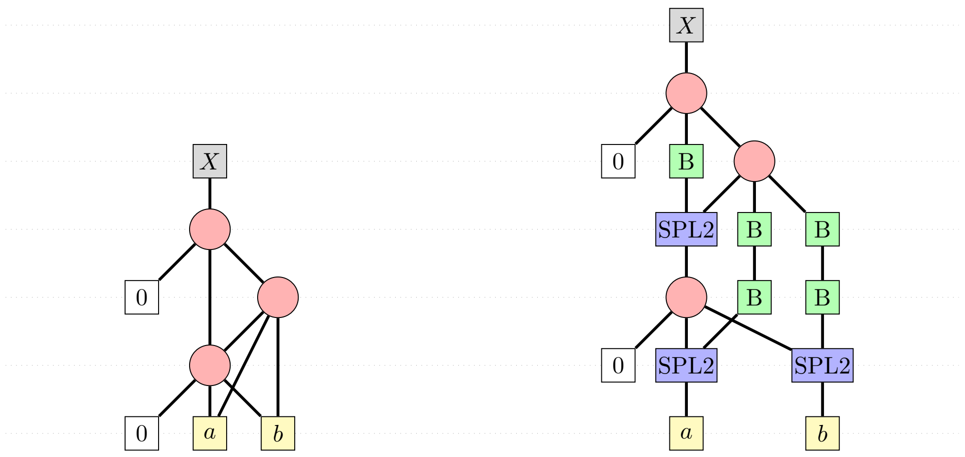


Figure 5.1: An example logic network with unit-delay gates (left) and its splitter-inserted, path-balanced version (right).

and this work focuses on the AQFP technology.

The AQFP technology provides efficient implementations of majority-3 and majority-5 gates that offer more complex logic functionalities at a comparably low resource usage (see Section 5.3), leading to more compact circuitry. On the other hand, AQFP logic gates cannot directly drive more than one fanout, necessitating clocked splitters to support multiple fanouts.

Moreover, the gates are also clocked, requiring each gate's fanins to arrive at the same time via logic paths that are balanced with (clocked) buffers. Figure 5.1 shows a simple logic network with unit-delay gates (left) and its splitter-inserted, path-balanced version (right). In all figures, green squares labeled "B" denote buffers while blue rectangles labeled "SPLk" denote splitters of branching factor k.

The path balancing and splitter requirements of AQFP technology pose additional challenges in logic synthesis because, in addition to the gates that implement logic functions, buffers and splitters also significantly affect the area and delay of a circuit. As such, there have been several attempts to optimize the splitter insertion and path balancing of AQFP circuits [19, 42, 163]. However, existing work suffers from one or more of the following weaknesses:

1. Lack of consideration for interdependent logic paths [42],
2. Bias towards using balanced splitter trees [163], and
3. Lack of support for more complex logic gates such as majority-5 [19, 42, 163].

In this work, we show how to mitigate these shortcomings using exact synthesis on small blocks of logic in a given logic network. Our main idea is to generate a database of minimum area (the number of JJs for example) AQFP circuits for all single-output, 4-input functions and for a set of different input arrival-time patterns, and then use the database to rewrite logic blocks of a larger network in the topological order.

Our approach differs from the similar-looking method proposed by Amaru et al. [14] for exact delay synthesis in two key aspects.

1. First, as their goal was to minimize the delay, their database generation ignores possible area improvements that result from logic sharing and enumerates only the tree structures. In contrast, our goal is to minimize the overall area, and logic sharing can both support (by reducing the gate count) and hinder (by increasing the splitter count, the number of levels, and consequently the number of buffers) this goal. Therefore we enumerate all directed acyclic graph (DAG) structures within a predetermined size bound.

2. Second, the outputs of logic blocks chosen by the algorithm can have multiple fanouts, and our algorithm takes the splitter requirements of such logic blocks into account before synthesizing the other logic blocks that use the outputs of already synthesized logic blocks as inputs. To synthesize such fanout nets, we use balanced splitter trees. But unlike the work of Testa et al. [163] which uses balanced splitter trees on all multi-fanout nodes, we use this strategy on only a small fraction of such nodes (i.e., only on the outputs of the blocks of logic chosen by the algorithm).

We restrict the set of enumerated DAG structures by pruning redundant structures using several symmetry-breaking heuristics. The database size is further reduced by exploiting the NPN equivalence of Boolean functions.

We evaluate our synthesis algorithm on the same subset of MCNC benchmarks considered by Testa et al. [163] using two different strategies, area-oriented and delay-oriented. The former strategy improves the area by over 21% while reducing the critical path delay by over 35% on average compared to the optimized results of Testa et al. [163]. Compared to the same results, the latter strategy reduces the area by over 19% while decreasing the critical path delay by more than 40% on average. Note that we use the total number of Josephson Junctions as the measure of area and the number of gate levels (including buffers and splitters) as the delay measure.

The remainder of this chapter is organized as follows: In Section 5.2, we discuss the shortcomings of existing approaches which motivated this work, and in Section 5.3, we provide some relevant background. In Section 5.4, we describe our database generation method and introduce the algorithm for synthesizing AQFP circuits. In Section 5.5, we present our experimental results, and in Section 5.6, we conclude with a brief discussion on our results and some potential improvements to the proposed method.

5.2 Motivation

In this section, we discuss existing work on optimizing AQFP circuits and their main drawbacks.

The work of Cai et al. [42] presents a new framework for inserting the optimum number of buffers and splitters for a given fanout net with fixed levels. Both Ayala et al. [19] and Testa et al. [163] proposed new synthesis flows for AQFP circuits. The synthesis flows first apply logic optimizations, then using the required number of splitters on multi-fanout nets, determine levels of the gates, and finally insert buffers as required together with optimizations that move splitters around to decrease the buffer count. In the logic optimization phase, the work of Testa et al. [163] uses depth optimizations based on majority-inverter graphs (MIGs) [8] (see Section 5.3), where the depth optimization algorithms use estimated splitter requirement as the cost heuristic to achieve better results.

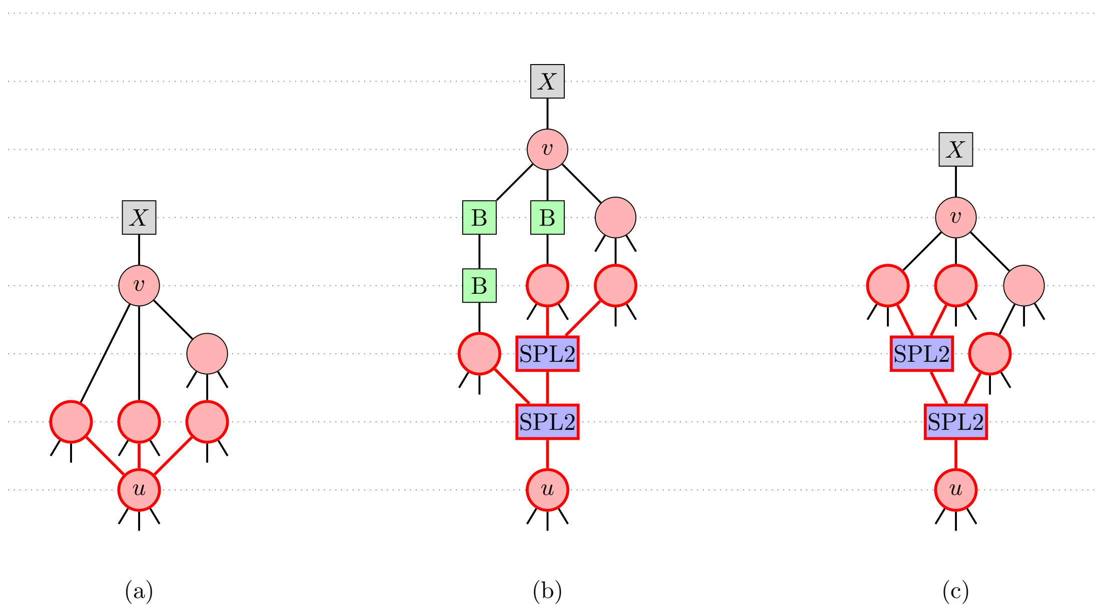


Figure 5.2: A part of a logic network with unit-delay gates (left), and its path-balanced versions (middle and right) using two choices of locally optimum splitter-trees with 1-to-2 splitters.

We identify three shortcomings in the existing work:

Lack of consideration for interdependent logic paths Consider Fig. 5.2 which shows a simple logic network on the left, and two possible splitter tree choices for the fanout net of node u assuming we have 1-to-2 splitters. (For simplicity, we disregard the splitter/buffer requirement of all unspecified fanins.) The *logic path interdependencies* make the choice on the right a much better option than the choice in the middle.

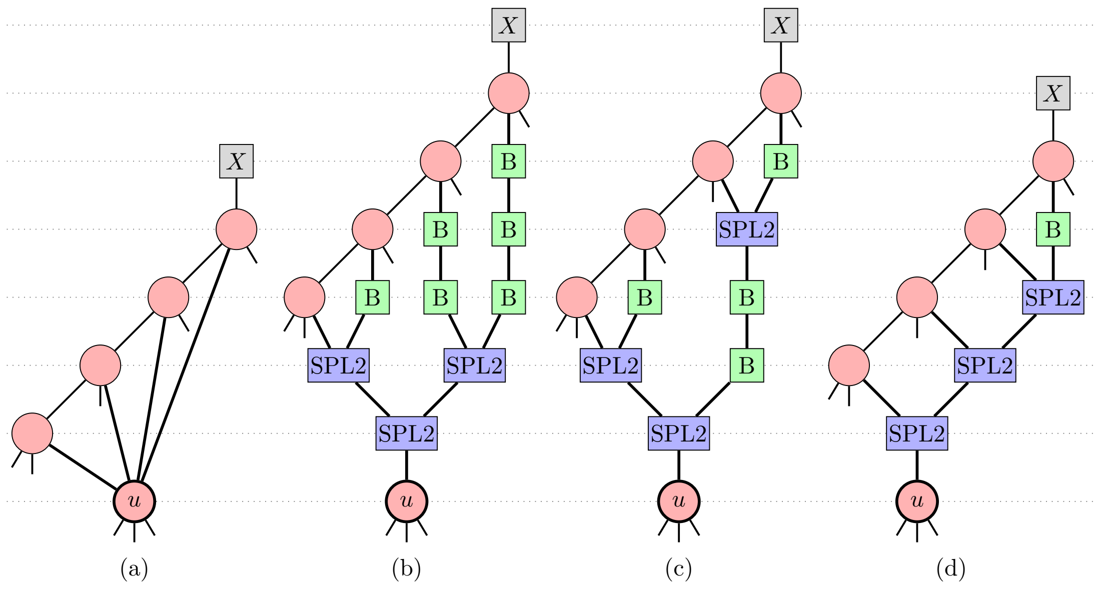


Figure 5.3: (a) A logic network with unit-delay gates, (b) its path balanced version using a balanced splitter tree, (c) an optimized version of (a) by pushing one splitter up in the hierarchy, and (d) optimum path balancing with an unbalanced splitter tree.

To elaborate, consider the three logic paths from node u to node v. The first arrangement of splitters in Fig. 5.2 (middle) increases the length of the path from u to v that goes through the right most fanin of v. Consequently, it increases the lengths of the remaining paths as well due to the balanced path requirement. The existing algorithms are susceptible to making suboptimal choices in such situations as they reason based solely on the local view in the vicinity of u and hence consider both splitter trees as equally good options.

Bias for balanced-splitter trees The approaches in prior work naively use balanced splitter trees for multiple-fanout nets when deciding the levels of those fanouts [163]. However, always using balanced splitter trees incurs additional buffer costs if the fanouts of a node have to be placed at uneven depths due to constraints dictated by other shared logic paths. In such cases, an unbalanced splitter tree can be a better match as shown in Fig. 5.3. (We again ignore the splitter/buffer requirement of all unspecified fanins for simplicity.)

For the given logic network (left), the unbalanced splitter tree (right) costs fewer buffers compared to a balanced splitter tree (middle two trees) assuming 1-to-2 splitters. The second network naively uses a balanced splitter-tree and adds buffers on top of it whereas the third network optimizes the buffer count by pushing one of the splitters up in the hierarchy. Nevertheless, the network on the right with the unbalanced splitter tree has a better resource usage.

Lack of support for more complex gates None of the prior works support the generation of netlists with majority-k gates for $k > 3$ although the AQFP technology can support efficient implementations of such gates [18]. For example, consider the logic function of majority-5 itself. Using only AND-2, OR-2, and majority-3 gates, computing this function needs at least four gates which costs at least 24 JJs whereas using a single majority-5 gate uses only 10 JJs. (See Section 5.3.)

In AQFP technology, the area overhead of buffers and splitters is quite significant. For example, in the work of Testa et al. [163], the buffers and splitters in optimized circuits amount to 39% of the total number of JJs. Improved path-balancing algorithms can result in significant area and delay reductions, but finding the optimal arrangement of buffers and splitters is a non-trivial task. However, such optimal arrangements can be computed reasonably fast for small logic networks using an exhaustive search algorithm, and this eliminates the aforementioned drawbacks when synthesizing such networks. We exploit this fact to mitigate those shortcomings when optimizing larger networks. To this end, we precompute a database of minimum area AQFP circuits (i.e., considering the buffers and splitters as well) for all logic functions with a small number of variables under different input arrival time patterns. We then use the database to rewrite logic blocks of larger networks.

5.3 Preliminaries

In this section, we give background on majority-inverter graphs (MIGs), AQFP logic circuits, and exact synthesis. Many of the concepts here have already been introduced in Chapter 2, but we restate some key points here for easier reference.

5.3.1 Majority-Inverter Graphs (MIGs)

A majority-inverter graph (MIG) is a directed acyclic graph (DAG) representation of a Boolean function. The internal nodes of an MIG represent majority gates, and the edges represent the connections between the gates with optional binary flags indicating the presence of inverters.

MIGs are supported by a sound and complete set of algebraic rules [13], which includes the following rules:

$$\begin{aligned}
\textbf{Commutativity}: &\quad \langle x\,y\,z\rangle = \langle y\,z\,x\rangle = \langle z\,x\,y\rangle,\\
\textbf{Associativity}: &\quad \langle x\,u\,\langle y\,u\,z\rangle\rangle = \langle z\,u\,\langle x\,u\,y\rangle\rangle\\
\textbf{Distributivity}: &\quad \langle x\,u\,\langle y\,z\,v\rangle\rangle = \langle\langle x\,u\,y\,\rangle\,z\,\langle x\,u\,v\rangle\rangle,\\
\textbf{Majority}: &\quad \langle x\,x\,y\rangle = x \text{ and } \langle x\,\bar{x}\,y\rangle = y,\\
\textbf{Inverter Propagation}: &\quad \langle \bar{x}\,\bar{y}\,\bar{z}\rangle = \overline{\langle x\,y\,z\rangle}.
\end{aligned}$$

The rules on *commutativity* and *inverter propagation* are particularly useful as they can be used to simplify the database generation process. These rules help identify MIG structures that are both *functionally* and *structurally* equivalent upto a reordering and possible negation of gate-fanins. We use such equivalences to simplify the enumeration of MIG structures in the database generation process.

5.3.2 AQFP logic circuits

As introduced in Section 2.4, AQFP technology offers logic gates which are constructed using superconductive inductors and Josephson Junctions (JJs) based on the Josephson effect [72]. Below, we recall some important details relevant to the AQFP technology including the composition of a target cell library and the resource usage of different gates.

Takeuchi et al. [158] proposed a simple cell library for AQFP technology based on four primitive cells—buffer, inverter, constant, and branch—where a gate is created using an array of primitive cells together with a branch while a splitter is constructed using a buffer and a branch. The majority-3 gate consists of three buffer cells together with a branch, and different fanin inverted versions of a majority-3 gate are constructed by substituting a subset of buffer cells with inverter cells [158]. The 2-input AND and OR gates are constructed by substituting a buffer cell with a constant cell.

Each of the three primitive cells, buffer, inverter, and constant, consists of two JJs, and hence a splitter also uses 2 JJs while all gates—majority-3, AND-2, and OR-2—as well as all their input-inverted versions use 6 JJs each. Additionally, the majority-5 gate and its input-inverted versions can be implemented with 10 JJs each [18]. As a majority-3 gate uses the same resources as an AND-2 or an OR-2 gate, Cai et al. [41] proposed that majority logic synthesis is more suitable for optimizing AQFP circuits.

Logic gates in AQFP technology cannot drive multiple fanouts and therefore a tree of splitters must be used in case a gate has to feed several fanouts. The gates and the splitters are clocked, and consequently, buffers must be inserted to make the circuit pipelined (see Section 2.4 for more details on clocking in AQFP circuits). Depending on the design of registers and the used clocking schemes, there can be different requirements on whether splitters are needed for primary inputs, whether path balancing is needed for primary inputs, and how the path balancing is done for primary outputs [143].

5.3.3 Exact synthesis

Exact synthesis is the process of synthesizing circuits to meet exact specifications. The different types and usecase of exact synthesis are discussed in Section 2.3.

Since exact methods are often computation-heavy, it is impractical to use them for large networks. Instead, a typical usecase for exact methods is to construct databases for logic functions with a small number of variables. Separate rewriting algorithms [123, 139] are then used to optimize a larger networks in a divide-and-conquer fashion, where small logic blocks are rewritten using precomputed database entries.

In the context of this chapter, we focus on exact synthesis of minimum-cost AQFP circuit structures for Boolean functions with small support.

5.4 AQFP Resynthesis Approach

In the first half of this section, we describe in detail the generation of the exact synthesis database, and in the second half, we present the overall algorithm for synthesizing path-balanced AQFP circuits using the precomputed database.

5.4.1 Generation of the database

The exact synthesis database contains MIGs that give the minimum area (after splitter-insertion and path-balancing) for each 4-input NPN class[I] under different input arrival-time patterns. We identify the input arrival-time pattern of a single-output MIG structure by the depths of its inputs with respect to the output. For simplicity, we explain the method focusing only on 3-input gates, but it is straightforward to extend it to include k-input majority gates for $k > 3$. The database is generated in three steps which are summarized below. Note that we use the term *DAG* to mean the underlying directed acyclic graph structure of an MIG *without* considering inverters.

First, we enumerate all single-output DAGs with two, three, four, and five leaf nodes (up to four inputs and a constant) and seven[II] gates. Next, we compute the area of realizing those DAGs as splitter-inserted, path-balanced circuits under a set of different input arrival patterns. We use the number of JJs (proportional to the number of primitive cells) to measure the area, but our method can support more general measures such as the physical cell area. Finally, we enumerate the 4-input NPN classes computable by each DAG structure (considering fanin inverters) and store the best-area MIG (i.e., the DAG together with a fanin inverter configuration) for each 4-input NPN class and for a set of input arrival-time patterns. We now describe each of the three steps in detail.

[I]We store MIGs that compute the function with the lexicographically smallest truth-table in each NPN class.

[II]It is known that all 4-input functions can be synthesized with MIGs of at most seven majority-3 gates [153].

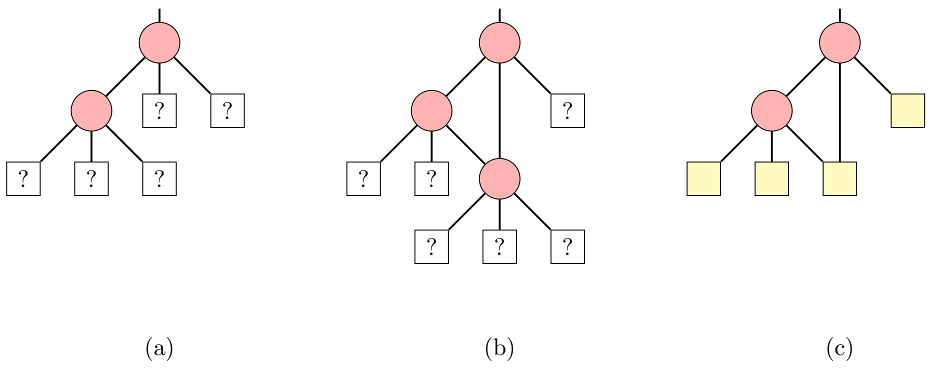


Figure 5.4: Two partial DAGs (left and middle) and a DAG (right). The partial DAG in the middle is derived from the one on the left by tying a new gate and connecting it to one uncommitted leaf in each of the existing gates. The DAG on the right is derived from the partial DAG on the left by attaching four inputs (leaf nodes) to the five uncommitted leaves.

Generating DAG structures

To systematically generate DAGs, we first generate *partial DAGs*, which are DAGs where some gates have leaves that are not committed to inputs. This notion of partial DAGs is similar to the one used by Haaswijk [65]. Note that, a partial DAG can be extended to a larger partial DAG by binding new gates to a subset of uncommitted leaves, or they can be converted to a DAG by binding inputs to uncommitted leaf nodes. Figure 5.4 shows a partial DAG (left), and another partial DAG (middle) and a DAG (right) obtained from the first partial DAG. Note that the uncommitted leaves are designated with question marks and inputs are shown as filled squares.

Starting with a single-gate partial DAG with three uncommitted leaves, we generate other partial DAGs by attaching gates to the uncommitted leaves of existing partial DAGs in a level-by-level fashion. Having generated all partial DAGs with k gate levels, we generate all partial DAGs that have $k + 1$ gate levels by extending those with k gate levels in such a way that each newly added gate has at least one fanout that is in level k. To generate DAGs from partial DAGs, we consider all different ways of attaching at most five inputs to available uncommitted leaves of a partial DAG. When enumerating DAGs, we use the *commutativity* and *majority* rules (see Section 5.3.1) to avoid having redundant gates.

Computing Area of DAGs

Recall from Section 5.3.2 that all input-inverted versions of a logic gate use the same amount of resources in AQFP technology. Thus, to determine the minimum amount of resources needed to realize an MIG as an AQFP circuit, we only need to know its underlying DAG structure ignoring the inverters. We now explain how to find the optimum buffer-splitter insertion for such DAGs. Note that we do not need splitter insertion or path balancing for constant nodes as we have different versions of majority gates with implicit constant fanins [158] (see Section 5.3). Thus the optimum buffer-splitter insertion for a DAG may depend on which leaf (if any) is treated as a constant. However, at the time of database generation, we do not know which leaf of an MIG in the database gets connected to a constant during synthesis. Therefore, for each DAG, we consider all versions of it obtained by assigning a constant to at most one leaf and compute their optimum splitter-inserted path balanced versions separately. Note that treating one leaf as a constant affects all shared logic paths. Hence it can also decrease the splitter-buffer resources needed in other fanout nets.

If we fix the depths of a gate and all of its fanouts, we can find the best buffer-splitter tree for that fanout net using dynamic programming. Once the gate depths are fixed, the best splitter-buffer tree for a given fanout net depends only on the *relative levels of the fanout nodes*. The relative levels of fanout nodes in the fanout net of a node u are the level differences between those fanout nodes and u. For example, Fig. 5.5 shows two fanout nets (of nodes u and v respectively) in red and blue that have the same relative fanout levels $\{2, 3, 3\}$.

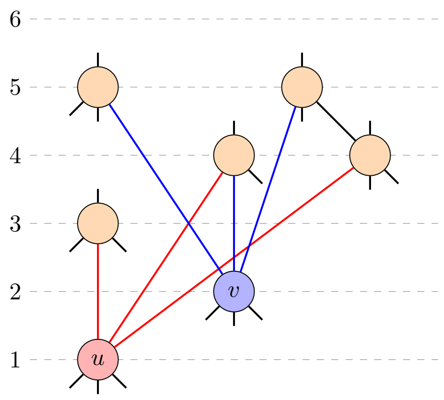


Figure 5.5: Two fanout-nets with the same relative fanout levels.

For a given multiset $S_{\text{lev}} = \{\ell_1, \ldots, \ell_k\}$ of relative fanout levels, the buffer cost c_b, the splitter cost c_s, and the splitter branching factor f_s, the recursive function in Algorithm 5.1 computes the optimum buffer-splitter cost or returns ∞ if there is no valid splitter-buffer configuration.

In any valid fanout net, all relative fanout levels must be positive (Line 2), and if a node has only one fanout, we only need to add sufficiently many buffers (Line 3). If there are multiple fanouts, the algorithm iterates over fanout choices for the top-most splitter in the splitter tree (Line 5), computes the cost of the chosen splitter and that of the buffers needed on the outputs of the chosen splitter (Lines 6-7), and recursively compute the best cost for the remaining tree (Line 8), while keeping track of the best cost found so far

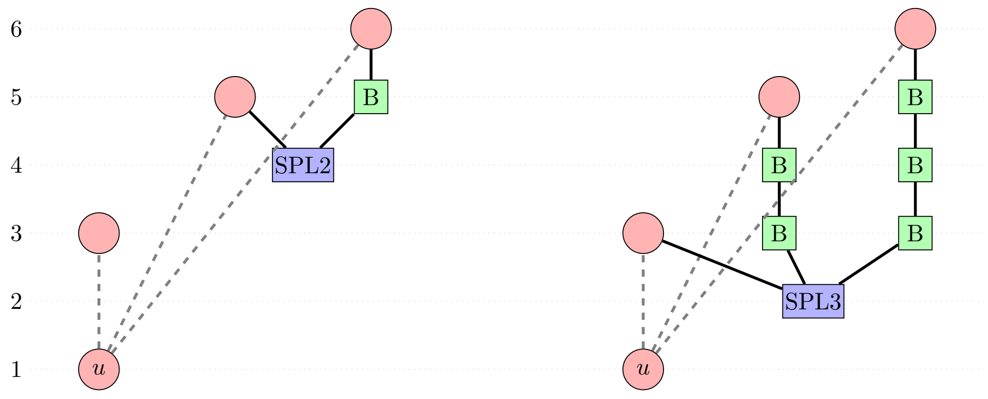


Figure 5.6: Computing best area for a relative level configuration using dynamic programming.

Algorithm 5.1: Computing minimum cost for a given multiset of relative fanout levels S_{lev}, buffer cost c_b, splitter cost c_s, and splitter branching factor f_s.

```
function cost($S_{\text{lev}} = \{\ell_1, \ldots, \ell_k\}$, $c_b$, $c_s$, $f_s$):
    if $0 \geq \min_{\ell \in S_{\text{lev}}} \ell$ then return $\infty$
    if $|S_{\text{lev}}| = 1$ then return $c_b(\ell_1 - 1)$
    $c_{\text{best}} \leftarrow \infty$
    for all $T \subseteq S_{\text{lev}}$ such that $2 \leq |T| \leq f_s$ do
        $c_T \leftarrow c_s$, $\ell_{\min} \leftarrow \min T$
        for $t \in T$ do $c_T \leftarrow c_T + c_b(t - \ell_{\min})$
        $c_T \leftarrow c_T$ + cost ( $\{\ell_{\min} - 1\} \cup S_{\text{lev}} \setminus T$, $c_b$, $c_s$, $f_s$)
        $c_{\text{best}} \leftarrow \min(c_{\text{best}}, c_T)$
    return $c_{\text{best}}$
```

(Line 9). For example, Fig. 5.6 shows two possible choices for the top-most splitter in the splitter tree (Line 5 of Algorithm 5.1) assuming that the splitter branching factor is at least three. We can speed-up Algorithm 5.1 by caching the already computed optimum buffer-splitter costs for different relative level configurations.

To find the minimum area for a DAG structure, we employ a depth bounded search algorithm. We consider all possible ways of assigning depths (relative to the output node) to the gates within a gradually increasing bound on the maximum depth. For each assignment of depths to gates, we compute the best splitter-buffer tree for each fanout net using Algorithm 5.1. If no valid splitter-buffer configuration is found for the considered bound on the maximum depth, we increase the bound and try again. Once we reach a depth bound that gives at least one depth assignment with a valid splitter-buffer configuration, we stop increasing the maximum depth. However, we still consider all

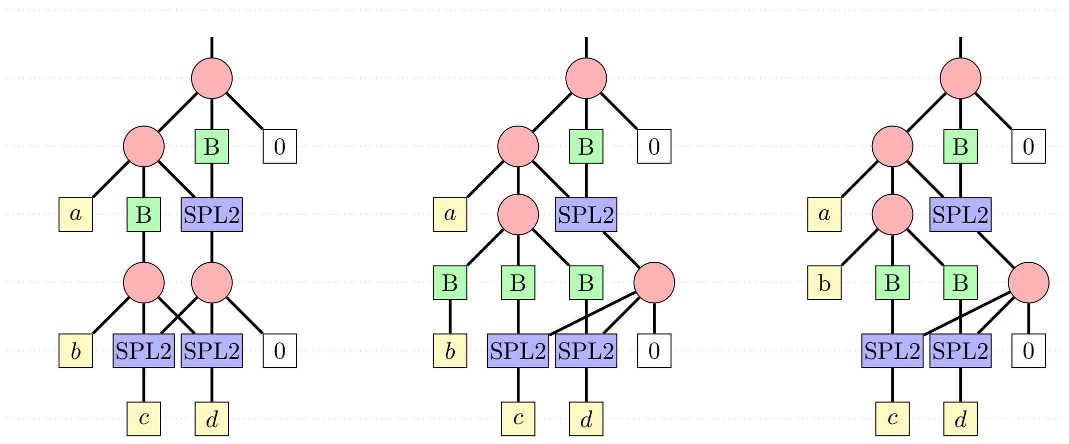


Figure 5.7: Three different buffer-splitter configurations to realize the same DAG structure.

possible depth combinations for the inputs (leaf nodes) and find the best cost for each such combination over all depth assignments to other gates.

For example, Fig. 5.7 shows three different depth assignments to gates and inputs for the same underlying DAG structure. One can verify there is no valid splitter-buffer configuration that achieves a maximum depth of 4, but there are several valid assignments for a maximum depth of 5 including the three shown in the figure. In the first two cases, the inputs (a, b, c, d) have the depths $(2, 4, 5, 5)$. (I.e., the input a is at depth 2, the input b is at depth 4, etc., assuming the top node has depth 0. This correspond to an arrival-time pattern where inputs a and b, respectively, arrive 3 and 2 unit-delays later than inputs c and d.) However, the first case uses only $(6 \cdot 4 + 2 \cdot 3 + 2 \cdot 2) = 34$ JJs whereas the second case uses $(6 \cdot 4 + 2 \cdot 3 + 2 \cdot 4) = 38$ JJs. Hence, the minimum cost for the considered DAG under the input depths $(2, 4, 5, 5)$ is 34. In the third case depicted in Fig. 5.7, the inputs (a, b, c, d) have depths $(2, 3, 5, 5)$ and the total cost is 36 JJs. Since this is the only valid buffer-splitter configuration for this input depth pattern, the minimum cost for the DAG under this input depth pattern is 36.

Enumerating NPN classes and constructing the database

To generate the database, for each DAG, we compute the different 4-input NPN-classes computable by it. We do this by considering all possible inverter configurations (i.e., different ways of inverting fanins of gates), evaluating the functions computed by the DAG under these inverter configurations, and determining the corresponding NPN classes together with the respective NPN transformation (i.e., how to permute and negate inputs and outputs). For a given majority-3 gate, although there are eight different inverter configurations for its fanins, we need to explore only half of them due to the inverter propagation property (see Section 5.3.1). Thus, in total, we explore 4^n different inverter configurations for a DAG with n gates. We use a pre-computed look-up table to efficiently find the NPN classes of 4-input functions and their associated NPN transformations.

Recall that, in the cost computation step, we computed costs of DAGs for different input arrival-time patterns identified by the pattern of the input depths. To construct the database, we go over different NPN classes computable by the DAG (under a suitable inverter configuration), and for each NPN class, we go over the different patterns of input depths and associated costs. For each NPN class f computable by a DAG g and for each input depths d and associated cost c, we then compute the new input depth pattern d' we would get if we use DAG g to compute f (this is computed by permuting d according to the input permutation of the associated NPN transformation). Finally, we update the database entry for (NPN class, input depth pattern) pair (f, d') by c and the respective input permuted DAG if there is no existing entry for the pair (f, d') or if c is smaller than the cost of the existing entry for the pair (f, d').

5.4.2 Synthesis Algorithm

We now describe our algorithm for synthesizing a large logic network as an AQFP circuit using the generated database. Given a logic network, it outputs a new MIG (which we refer to as the AQFP circuit) together with an assignment of levels to each gate. The best splitter-buffer configuration can then be recovered from the level assignment by adapting Algorithm 5.1 to output the minimum area splitter-buffer configuration instead of only returning the minimum area.

The algorithm, outlined in Algorithm 5.2, first maps the input network to a 4-LUT circuit using ABC's [34] LUT mapping [124]. Then, it maintains two mappings, one from the 4-LUT nodes to the corresponding signals (a signal denotes the output of nodes or its complement) in the new AQFP circuit, and another from the majority nodes in the new AQFP circuit to their assigned levels. Initially, it replicates all primary inputs in the target AQFP network, assigns level 0 to each primary input, and initializes the two mappings accordingly. Then it traverses the 4-LUT nodes in the topological order and resynthesizes each 4-LUT node according to a DAG structure chosen from the exact synthesis database. To choose the best DAG structure for a given 4-LUT node n with fanins $n_1, \ldots, n_4$, the algorithm does the following: First, it computes the node function h (i.e., h such that the output of n is $h(n_1, \ldots, n_4)$), its NPN class f, and the input permutation σ that describes how to permute the inputs $n_1, \ldots, n_4$ in order to compute h from f. Then it computes the current levels of the signals in the AQFP circuit that correspond to nodes $n_1, \ldots, n_4$. In case any fanin n_i has multiple fanouts, its level is computed assuming a nearly balanced splitter tree at its output. For example, suppose that n_1 has 10 fanouts and we have 1-to-4 splitters. Let $n_1^{(\mathrm{aqfp})}$ be the corresponding node in the new AQFP network. Then, the algorithm assumes a splitter tree with three splitters and two levels at the output of $n_1^{(\mathrm{aqfp})}$. Consequently, if $n_1^{(\mathrm{aqfp})}$ is at some level

Algorithm 5.2: Algorithm to synthesize a given logic network as an AQFP circuit.

```
1  $\text{ntk}_{\text{aqfp}} \leftarrow$ Empty AQFP circuit.
2  $\text{ntk}_{\text{lut}} \leftarrow$ ABC_LUT_MAP($\text{ntk}_{mig}$).
3  $m_{\text{lev}} \leftarrow$ Empty map from $\text{ntk}_{\text{aqfp}}$ nodes to integers.
4  $m_{\text{sig}} \leftarrow$ Empty map from $\text{ntk}_{\text{lut}}$ nodes to $\text{ntk}_{\text{aqfp}}$ signals.
5  foreach primary input $p$ of $\text{ntk}_{\text{lut}}$ do
6      Create new primary input $p'$ in $\text{ntk}_{\text{aqfp}}$.
7      $m_{\text{lev}}[p'] \leftarrow 0$, $m_{\text{sig}}[p] \leftarrow p'$.
8  foreach node $n \in \text{ntk}_{\text{lut}}$ in topological order do
9      $n_i \leftarrow$ The $i$-th fanin of $n$ for $i = 1, ..., 4$.
10     $\ell_i \leftarrow$ Adjusted level of $m_{\text{sig}}[n_i]$ for $i = 1, ..., 4$.
11     $h \leftarrow$ Node function of $n$.
12     $f \leftarrow$ NPN class of $h$.
13     $\sigma \leftarrow$ Input permutation to get $h$ from $f$.
14     $(\ell'_1, \ldots, \ell'_4) \leftarrow \sigma(\ell_1, \ldots, \ell_4)$.
15     $g \leftarrow$ Best DAG from DB for $f$ and $(\ell'_1, \ldots, \ell'_4)$.
16     $g \leftarrow$ Input permuted $g$ according to inverse of $\sigma$.
17     $g \leftarrow$ Fanin inverted $g$ such that it computes $h$.
18     Create $g$ in $\text{ntk}_{\text{aqfp}}$ with inputs $(m_{\text{sig}}[n_1], \ldots, m_{\text{sig}}[n_4])$ and let $n'$ be the root
         node.
19     Update $m_{\text{lev}}[n']$ and $m_{\text{sig}}[n]$.
```

ℓ, for two of n_1's fanouts, we assume n_1 is available at level $\ell + 1$ (the first splitter in the tree has two slots remaining), and for the other eight of n_1's fanouts, we assume n_1 is available at level $\ell + 2$ (after two successive splitters). After finding the input levels of the fanin nodes, we permute those levels according to σ.

Next, in the database, we go over the entries with different input depth patterns for NPN class f. Suppose that for NPN class f and input depth pattern d, we have some DAG g with cost c. If we were to use DAG g to resynthesize node n, the cost would be c plus the cost of buffers needed to fill in the gaps between the permuted fanin levels and the input depths d of DAG g. Furthermore, we can also compute the level of the new node in the AQFP circuit that would represent the 4-LUT node n, using the permuted input levels and the input depth pattern d. At this point, to choose the best DAG, we propose two strategies: Either we use an *area-oriented* strategy where we choose the DAG that minimizes the area and break ties using the level we would get for the output node, or we use a *delay-oriented* strategy where we choose the DAG that minimizes the level of the output node and break ties using the area cost that would be incurred.

5.5 Experimental Results

In this section, we present the experimental results obtained from our AQFP synthesis algorithm and compare them with the results obtained by the AQFP synthesis flow presented in [163][III]. We consider the same subset of 18 MCNC benchmarks [176] used in that work.

Similarly, we also use the same AQFP cell library discussed in Section 5.3 and use the number of JJs in the synthesized network as the area measure and the number of levels on the critical path as the delay measure. We construct two exact synthesis databases assuming we have 1-to-4 splitters:

DB1 A database that uses DAGs with up to seven 3-input gates without any 5-input gates.

DB2 DB1 extended with a) DAGs with up to three 5-input gates and three 3-input gates with a limit of four on the total number of gates and b) DAGs with up to three 5-input gates, four 3-input gates, and three levels with a limit of five on the total number of gates.

To generate the database, we used a cluster with 48 cores of Intel Xenon E5-2680 v3 CPUs running at 2.5GHz, and 256GB main memory, and each of the three steps was executed using 48 parallel threads. The generation of all DAGs for DB1 using the method described in Section 5.4.1 consumed ∼20 minutes, and the output consists of 440 million DAGs (including different versions obtained by designating one leaf node as the constant node). The cost computation step Section 5.4.1 took ∼1.5 hours whereas enumerating computable NPN classes and constructing the final database took ∼30 hours. Extending DB1 to DB2 using the DAGs with the given constraints took ∼25 hours in total. After removing redundant input depths patterns, DB1 and DB2 consist of only 5744 and 4317 DAGs respectively over all 222 4-input NPN classes.

We perform two experiments with the two databases: In the first experiment, we first synthesize the initial MIG as an AQFP circuit using our proposed algorithm with DB1. Then using the underlying MIG in the synthesized AQFP circuit as the input, the same algorithm was repeatedly applied for a total of 10 iterations, and considered the best result obtained among all iterations. The second experiment is the same as the first one except that we make sure the last iteration of the algorithm is run with DB2. To elaborate, for each $i \in 1, \ldots, 10$, we run $i - 1$ iterations of the synthesis algorithm using DB1 and one iteration with DB2, and we pick the best result out of the 10 versions.

The reason not using DB2 in the intermediate iterations is that, if it creates majority-5

[III] In fact, we reproduced their results using the scripts provided by the authors that are available online.

Table 5.1: Results for the experiment where the proposed AQFP synthesis algorithm is applied for 10 iterations under the assumption that no splitter-buffer insertion is needed for primary inputs but primary outputs need path-balancing. The *reference* column shows the optimized results from [163].

	Reference		**All iterations use DB1**								**Last iteration uses DB2**							
			Area-Oriented				**Delay-Oriented**				**Area-Oriented**				**Delay-Oriented**			
Bench-mark	Delay (Levels)	Area (#JJs)	Delay (Levels)	Area (#JJs)	Delay Impr. %	Area Impr. %	Delay (Levels)	Area (#JJs)	Delay Impr. %	Area Impr. %	Delay (Levels)	Area (#JJs)	Delay Impr. %	Area Impr. %	Delay (Levels)	Area (#JJs)	Delay Impr. %	Area Impr. %
5xp1	8	824	8	716	0.00	13.11	8	742	0.00	9.95	8	674	0.00	18.20	8	730	0.00	11.41
c1908	53	5242	39	4512	26.42	13.93	36	5204	32.08	0.72	36	4082	32.08	22.13	32	4498	39.62	14.19
c432	50	2198	35	2178	30.00	0.91	36	2944	28.00	-33.94	32	1994	36.00	9.28	35	2696	30.00	-22.66
c5315	49	18932	33	14976	32.65	20.90	30	16312	38.78	13.84	31	14410	36.73	23.89	29	14850	40.82	21.56
c880	36	4520	24	3406	33.33	24.65	21	3678	41.67	18.63	22	3200	38.89	29.20	20	3402	44.44	24.73
chkn	28	4022	18	3312	35.71	17.65	14	3398	50.00	15.51	15	2900	46.43	27.90	13	2988	53.57	25.71
count	18	1426	13	1184	27.78	16.97	11	1346	38.89	5.61	13	1126	27.78	21.04	11	1326	38.89	7.01
dist	17	4208	13	3802	23.53	9.65	11	3990	35.29	5.18	12	3502	29.41	16.78	10	3480	41.18	17.30
in5	20	4312	15	3522	25.00	18.32	13	3754	35.00	12.94	12	3042	40.00	29.45	12	3116	40.00	27.74
in6	17	3472	12	2978	29.41	14.23	10	2952	41.18	14.98	10	2572	41.18	25.92	8	2552	52.94	26.50
k2	29	18294	19	16380	34.48	10.46	18	16306	37.93	10.87	16	14326	44.83	21.69	16	14372	44.83	21.44
m3	13	3118	12	2964	7.69	4.94	10	3016	23.08	3.27	10	2654	23.08	14.88	9	2680	30.77	14.05
max512	19	5536	14	5018	26.32	9.36	13	5334	31.58	3.65	13	4610	31.58	16.73	12	4636	36.84	16.26
misex3	29	14996	18	11922	37.93	20.50	15	12598	48.28	15.99	17	10580	41.38	29.45	14	10584	51.72	29.42
mlp4	19	3622	13	3222	31.58	11.04	11	3326	42.11	8.17	11	2938	42.11	18.88	10	2998	47.37	17.23
prom2	22	28774	16	26300	27.27	8.60	14	27302	36.36	5.12	14	24374	36.36	15.29	13	24586	40.91	14.55
sqr6	11	1102	9	962	18.18	12.70	8	978	27.27	11.25	8	896	27.27	18.69	7	902	36.36	18.15
x1dn	15	1296	11	1126	26.67	13.12	10	1148	33.33	11.42	10	988	33.33	23.77	10	1010	33.33	22.07
Total	453	125894	322	108480	**28.92**	**13.83**	289	114328	**36.20**	**9.19**	290	98868	**35.98**	**21.47**	269	101406	**40.62**	**19.45**

gates, the subsequent LUT mapping operation can potentially increase the overall size since a single 4-LUT cannot compute majority-5.

The two experiments were done using both the area-oriented and delay-oriented strategies for selecting an appropriate DAG from the database. When using the area-oriented strategy, we select the circuit with the minimum area over the 10 iterations as the best result. Similarly, when using the delay-oriented strategy, we select the circuit with the minimum critical-path length as the best result.

Table 5.2: JJ utilization of majority-3 and majority-5 gates in the output of the proposed AQFP synthesis algorithm under the assumption that no splitter-buffer insertion is needed for primary inputs but primary outputs need path-balancing.

	All iterations use DB1		**Last iteration uses DB2**			
	Area-Oriented	**Delay-Oriented**	**Area-Oriented**		**Delay-Oriented**	
Benchmark	Maj-3	Maj-3	Maj-3	Maj-5	Maj-3	Maj-5
5xp1	624	654	408	180	384	240
c1908	2334	2616	1716	570	1632	910
c432	1296	1692	864	380	882	650
c5315	7470	8202	6396	920	5886	1800
c880	1890	2094	1236	510	1248	700
chkn	2622	2778	1278	1120	1278	1230
count	774	852	624	130	576	230
dist	3252	3576	1992	1100	1812	1370
in5	2652	2850	1206	1210	1230	1250
in6	2232	2340	996	1030	996	1120
k2	12042	11964	4476	6310	4356	6420
m3	2514	2664	1452	890	1386	1060
max512	4296	4638	2598	1430	2496	1680
misex3	9276	10164	4878	3650	4872	3950
mlp4	2802	2982	1800	840	1596	1180
prom2	22458	23766	14316	7050	12930	9130
sqr6	816	864	516	260	492	330
x1dn	930	948	360	470	390	460
Total	80280	85644	47112	28050	44442	33710
% w.r.t. area (#JJs)	74.00%	74.91%	47.65%	28.37%	43.82%	33.24%

We first perform all experiments under the same assumptions used by Testa et al. [163] that no splitters or buffers are needed on primary inputs but all primary outputs have to be path-balanced using buffers. The results are shown in Table 5.1 together with the improvements as compared to the results of Testa et al. [163]. As seen from Table 5.1, the repeated application of our proposed algorithm reduces the delay by 40.62% and decreases the area by 19.45% when the delay-oriented strategy was used, while achieving a

35.98% reduction in delay and a 21.47% decrease in area when the area-oriented strategy was used. It is evident that having majority-5 gates in the database allows the algorithm to achieve up to 9% delay improvements with further area reductions as compared to the case where only majority-3 gates were allowed in the database. Note that, in the output circuits of our algorithm, the percentage of path balancing resources, i.e., the percentage of JJs in splitters and buffers compared to the total number of JJs, is ∼25% on average whereas that quantity is over ∼39% in [163].

In Table 5.2, we show the resource usage by different types of majority gates in the output circuits. When majority-5 gates are allowed, 28% (33%) of the logic resources are used by the majority-5 gates in the area-oriented (delay-oriented) strategy, implying that majority-5-like functions often occur as parts of larger logic networks. Such functions include 5-input functions that are in the same NPN-class as majority-5, as well as their versions where one input is repeated. For example, $w(xy+yz+xz)+xyz = \langle w, w, x, y, z\rangle$ is a four input function synthesizable with a single majority-5 gate.

Table 5.3: Results for the experiment where the proposed AQFP synthesis algorithm is applied for 10 iterations under the assumption that primary inputs need splitters to support multiple fanouts and primary outputs need path-balancing.

	All iterations use DB1				**Last iteration uses DB2**			
	Area-Oriented		**Delay-Oriented**		**Area-Oriented**		**Delay-Oriented**	
Benchmark	Delay (Levels)	Area (#JJs)	Delay (Levels)	Area (#JJs)	Delay (Levels)	Area (#JJs)	Delay (Levels)	Area (#JJs)
5xp1	10	870	10	888	10	830	10	884
c1908	37	5972	37	6328	37	5580	35	5810
c432	41	4002	39	4200	37	3714	35	3944
c5315	31	17600	32	19518	30	17166	31	17914
c880	25	4984	24	5012	23	4070	23	4086
chkn	16	3812	16	3912	13	3496	14	3556
count	12	1632	12	1660	12	1574	12	1592
dist	14	4332	15	4456	13	4060	13	4036
in5	15	3942	14	4070	13	3544	13	3520
in6	12	3360	12	3292	11	3010	10	2930
k2	22	17634	22	17632	20	15696	19	15444
m3	13	3334	13	3320	13	3108	12	3168
max512	17	5798	15	5796	15	5324	14	5306
misex3	21	13160	20	13512	20	11928	19	11996
mlp4	15	3714	14	3820	13	3480	13	3538
prom2	19	29362	19	30238	18	27608	18	27946
sqr6	11	1154	10	1134	10	1084	9	1070
x1dn	12	1348	11	1356	10	1214	10	1214
Total	343	126010	335	130144	318	116486	310	117954

In Table 5.3, we present the results for the same experiments under the assumption that splitters are needed for primary inputs to support multiple fanouts and primary outputs have to be balanced. We still assume that path balancing is not needed for primary inputs. It is noteworthy that, even when using splitters for primary inputs, our algorithm achieves better area and delay as compared to the reference work that did not use splitters on primary inputs.

5.6 Summary

As seen from the results in Section 5.5, our proposed algorithm for the exact synthesis of AQFP circuits achieves much improved circuits in terms of both area and delay. These improvements are enabled by holistic and simultaneous optimizations of logic and path balancing resources that capture more optimization opportunities compared to prior work in the field. The exact synthesis method employed in this work is able to use unbalanced splitter trees effectively in most parts of the circuit, and the balanced splitter tree assumption is used only on outputs of blocks of logic. To the best of our knowledge, this is the first AQFP synthesis approach that can produce AQFP circuits with majority-5 gates. Our results demonstrate that having majority-5 gates can help significantly improve the resource usage and reduce delay.

Moreover, our database generation method is not restricted to using the number of JJs as the area cost. Instead, it can also work with more general cost functions such as the cell area and consider multiple types of splitters with varying branching factors and area with minor modifications to Algorithm 5.1.

We remark that our database can be further improved by considering more DAG structures at the expense of the one-time computational cost of generating the database. Also, the current algorithm depends on an external LUT mapping algorithm and hence a better LUT mapping algorithm can yield better overall results. Alternatively, it is interesting to see if we can directly integrate the database with a technology mapper to achieve even better results.

In conclusion, optimizing path balancing resources in AQFP circuits is a non-trivial problem, and effective path-balancing can heavily reduce the resource usage in AQFP circuits. Considering logic optimization together with path balancing as a whole while taking interdependent logic paths into account and effectively using unbalanced splitter trees when possible leads to faster logic circuits with a better resource utilization in AQFP technology. It is possible to achieve further area and delay reductions by also allowing higher fanin majority gates in the synthesis process. The superior performance of our approach makes it attractive as a state-of-the-art synthesis flow for the AQFP technology, and we believe that expanding on the insights of this work will yield even better results.

6 Logic Synthesis for FCN Technologies

The dominance of CMOS technology in the semiconductor industry is being challenged by emerging technologies that promise ultra-low power dissipation and high-speed computation. One such family of technologies is Field-Coupled Nanotechnologies (FCN), which have unconventional design constraints that significantly impact the cost of the final circuit layout. The existing logic synthesis algorithms, which are optimized for CMOS technologies, are not adept at handling these constraints. In this chapter, we explore how logic synthesis can be augmented to incorporate these constraints.

This chapter is based on a paper [114] published in the IEEE International Conference on Nanotechnology (IEEE-NANO), which was also presented in the International Workshop on Logic Synthesis (IWLS) 2023 [111].

6.1 Introduction

As Moore's Law has lost momentum, alternative circuit technologies that transcend past conventional transistor-based logic are arising from studies into material science and physics. These beyond-CMOS devices promise enhancements over CMOS circuits in various aspects. While *Photonic Crystals* perform logic operations through wave interference of photons at the speed of light, for instance [82, 179], *Silicon Dangling Bonds* (SiDBs) conduct logic-in-memory computations via the repulsion of electric fields with ultra-low power dissipation [79, 135, 175]. Similar *Field-coupled Nanotechnologies* (FCN) [16] are, e. g., *Quantum-dot Cellular Automata* (QCA) [101, 102] and *Nanomagnet Logic* (NML) [24, 54].[I] While these and future emerging technologies have their own design constraints, some similarities that are not captured by conventional optimization criteria continue to reappear.

To this end, an abundance of design flows for emerging circuit technologies rely on

[I]New technologies are continually being proposed, but their physical details are not crucial for the motivation and comprehension of this work.

conventional logic synthesis that aggressively optimizes the number of nodes in logic networks, e. g., represented as an *And-Inverter Graphs* (AIGs), before incorporating technology-specific constraints on the physical design level. When dedicated placement and routing tools attempt to legalize such sub-par logic networks, it usually results in increased layout costs due to the prior negligence of, e. g., inverter and/or interconnect costs, which add to the total area, delay, and power metrics [148, 165] (see Section 6.2.1).

In beyond-CMOS technologies, the interconnect costs come from a variety of constraints such as path-balancing, branching, and planarization (see Section 2.4 and Section 6.2.1 for details). In some such technologies, interconnect costs commonly dominate gate costs by several orders of magnitude [148]. Therefore, *optimizing logic networks primarily for their number of nodes can be counterproductive* as more important cost factors are completely ignored. For superconducting electronics technologies, there have been attempts to mitigate interconnect overheads due to path-balancing and branching constraints [97, 134]. However, in technologies such as FCN, the main source of interconnect cost is planarization; to the best of our knowledge, no prior work has considered such costs in logic synthesis.

In this work, we propose to avoid the substantial overhead incurred when generating beyond-CMOS circuit layouts from conventionally optimized logic networks by incorporating *recurring generic physical design constraints of emerging technologies* into the *logic synthesis step* through optimizing for *unconventional but realistic cost functions*. To this end, we present a technology mapping algorithm that utilizes databases of exact subcircuits implemented in specific technologies. Our proposed algorithm provides average improvements of 84.5 %, 74.5 %, and 65.2 % for the number of buffers, the number of crossings, and the length of the critical path, respectively, compared to a state-of-the-art physical design algorithm for FCN. The proposed algorithm is not limited to a particular technology but applies to a wide range of non-conventional circuit implementations that may exhibit one or more of the generic design constraints discussed in Section 6.2.1.

The remainder of this chapter is structured as follows: Section 6.5 discusses preliminaries and related work necessary for the comprehension of this manuscript. Section 6.3 proposes the novel technology mapping algorithm that constitutes the main contribution of this work. In Section 6.4, a comparative experimental evaluation is conducted against the state of the art and its results are discussed in detail. Finally, Section 6.5 concludes the chapter.

6.2 Preliminaries

This section briefly discusses technology constraints of beyond-CMOS technologies, a general circuit model, and some related work on technology mapping.

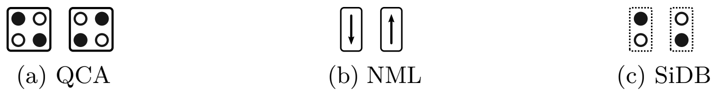


Figure 6.1: Elementary FCN devices.

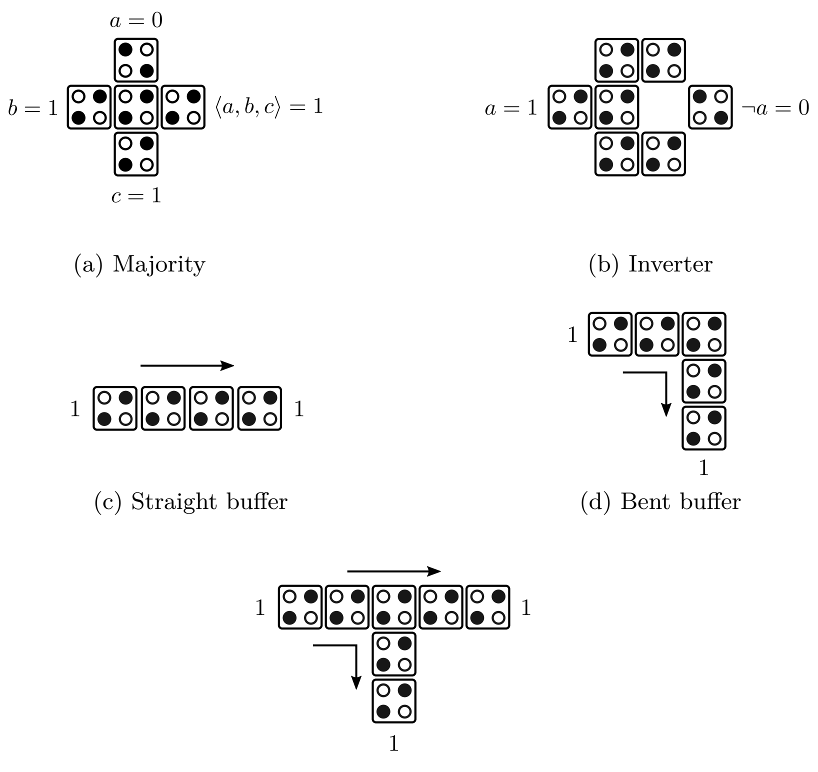
6.2.1 Beyond-CMOS Technologies with Unconventional Costs

In conventional technology-independent logic synthesis, AIGs are often used as circuit representations. For NAND-based CMOS technologies, the AIG size (or depth) measured in the number of AND gates in the network (or on the critical path) serves as a good estimation of the post-mapping area (or delay). However, in many emerging technologies, additional design constraints are imposed requiring special cells to be inserted to fulfill them; which increases the discrepancy between technology-independent and post-mapping layout cost metrics. There exists a plethora of beyond-CMOS technologies with such additional design constraint, and here we describe two such families. Due to this chapter's brevity, the technologies cannot be discussed in-depth. Instead, a general overview with a focus on cost functions is provided. We then identify a set of generic design constraints shared by many of these emerging technologies.

Field-coupled Nanocomputing (FCN)

FCN is an umbrella term for a variety of nanotechnologies that have similar high-level models [16], and it contains *Quantum-dot Cellular Automata* (QCA) [101, 102], *Nanomagnet Logic* (NML) [24, 54], and *Silicon Dangling Bonds* (SiDBs) [79, 175]. Although their physical implementations differ, their concepts are nearly identical. In all cases, the information is represented by the polarization/magnetization of elementary nanometer-scale building blocks called *cells*. When placed in close proximity, cells influence each other's polarization/magnetization through *Coulomb interaction*. Thus, they transmit information through electric/magnetic field coupling without a current flow [16], greatly reducing power dissipation and requiring less cooling than MOSFETs [28, 165]. Reversible FCN can operate below the *Landauer limit* [86, 95], a theoretical energy dissipation bound of non-reversible computation.

Figure 6.1 shows elementary cells of aforementioned FCN technologies in the two binary states `0` (left) and `1` (right). The topological arrangements of cells form wire segments and gates that conduct Boolean operations, as shown in Figure 6.2 for the QCA implementation [100]. During physical design, gates and wire segments are arranged in uniform *standard tiles* [77, 137], which can be viewed as building blocks that abstract physical effects to the logic design layer. Placement and routing of standard tiles attempt to create a layout from these building blocks that is functionally equivalent to a given logic network. Each tile has the same unit cost in both the area and the delay regardless of



(a) Majority (b) Inverter (c) Straight buffer (d) Bent buffer (e) Splitter

Figure 6.2: QCA gates and wire segments.

whether it is a logic gate, a buffer, an inverter, or a splitter. FCN circuits are functionally sensitive to delays; signal paths of different lengths (in terms of the number of tiles) desynchronize, causing incorrect calculations [166].

Photonic Crystals

To realize complete optical logic circuits with information transmission at the speed of light, research has focused on photonic crystals, that is, optical nanostructures with periodically changing refractive indices [82]. This property allows or prohibits electromagnetic radiation to propagate through a photonic crystal based on its wavelength. Within the crystal lattice, *waveguides* [83] can be fabricated that restrict incoming light to propagating along certain channels; effectively creating wires for photons. At intersections, light originating from two different waveguides interferes to cancel out or amplify, based on its phase shift. This property has been used to envision optical Boolean gates [179]. However, interacting waveguides of different lengths can cause signals to desynchronize, thereby distorting or breaking gate functionality.

Unconventional design constraints and their costs

In logic synthesis, the area cost of an optimized network is typically measured as the sum of approximate gate area over all gates of the network. In CMOS, this often provides a good approximation of the final layout area. However, in emerging technologies, the final area is often dominated by interconnect costs, which arise due to unconventional design constraints.

We consider the following costs that are shared among many aforementioned emerging technologies (and more) but not considered in conventional CMOS logic synthesis flows.

1. Path-balancing buffers: Due to the sensitivity to delay differences in signal paths, the path-balancing constraint is imposed, requiring that all paths from primary inputs to the fanins of the same gate have the same length. When shortening longer paths is not possible, buffer cells must be inserted into shorter paths to equalize the delay.

2. Fanout-branching splitters: Because the design of logic gates in these technologies does not naturally support driving multiple fanout signals, additional splitter cells need to be inserted at the output of multi-fanout gates to fulfill the fanout-branching constraint. Moreover, splitters are also counted in the path lengths of path balancing, thus, the fanout-branching and path-balancing constraints are strongly interwoven.

3. Planarizing crossings: The physical design of these technologies often requires special crossing cells to realize wire crossings in a 2-dimensional layout. The placement of logic gates may be altered to minimize such cases, but it is usually not possible to completely planarize the input network. Similarly to splitters, crossings also contribute to the path lengths and have to be considered together with path balancing. In some technologies, crossing cells are hard to fabricate or lead to less robust circuits due to their weaker signal strengths. In these cases, it is important to minimize the number of crossings, and unavoidable crossing cells lead to higher costs.

4. Non-cost-free inverters: Unlike traditional logic synthesis, where inverters (complemented edges) in AIGs do not contribute towards their size, inversion is not free in these technologies. It is not always possible to embed an inversion as a free negated input to a gate. Mitigating the effects of such inverters has been studied in [160]. However, these dedicated inverters not only increase the circuit size but also need to be considered together with the path-balancing constraint.

Through this consideration, it becomes apparent that cost metrics of conventional logic synthesis algorithms are not suited for the beyond-CMOS due to their negligence of the important aspects enumerated above.

6.2.2 Circuit Model

As we propose a new technology mapping algorithm supporting beyond-CMOS cost metrics, we locate this work at the intersection of logic synthesis (concerned with logic networks) and physical design (concerned with circuit layouts). The proposed algorithm's output is a mixed logic network consisting of logic gates that are supported by a given technology library, which we represent as *k-input look-up tables* (k-LUTs), and special cells including path-balancing buffers, fanout-branching splitters, and planarizing crossings. To insert crossings in a meaningful way ensuring the network's planarity, our mapping algorithm entails a coarse-grained placement with relative node positions. I.e., all cells in the mapped network are sorted into path-balanced *ranks* and are ordered within each rank.

Thereby, our algorithm outputs a partially placed, mapped network that 1. is functionally equivalent to the input network; 2. consists only of cells supported by the given technology library; and 3. satisfies all four constraints described in the previous section (path balancing, fanout branching, planarization, and dedicated inverters). Additionally, our algorithm aims to minimize the size and depth of this network model, which considers both logic gates and special cells. The area cost of each cell type can be parameterized to reflect the target technology as precisely as possible.

By considering such a circuit model, our size and depth evaluation is closer to the actual area and delay of the resulting layout after physical design. As design constraints are already satisfied and cells are ranked and ordered, the remaining placement and routing tasks become trivial for some technologies, but others may still need the network to be placed and routed according to the target layout topology, e. g., QCA layouts with non-linear clocking schemes [46, 63].

6.2.3 Conventional Technology Mapping

In typical logic synthesis flows, technology-aware optimizations are performed in the technology mapping stage, which happens after heavily optimizing a technology-independent representation with methods such as rewriting [123]. Technology mapping transforms a technology-independent logic representation into a technology-dependent one, where mapped circuits are obtained by substituting small sections with standard cells that represent the elements of the target technology. Numerous mapping algorithms have been proposed over the years [52, 94], but most such approaches are specific to CMOS-based synthesis flows and produce sub-par results when used for emerging technologies as such methods were not targetting aforementioned unconventional costs.

6.3 Proposed Methodology

In this section, we describe the proposed novel technology mapping approach for beyond-CMOS technologies. While we use FCN as the exemplar technology because it has all four unconventional design constraints of Section 6.5 and it is a promising competitor in the beyond-CMOS domain due to recent fabrication breakthroughs [79], the proposed idea is generally applicable to other emerging technologies that require path-balancing, branching, and (optionally) planarization. To the best of our knowledge, this is the first algorithm that considers planarization during the logic synthesis stage.

We propose generating

1. a design database of optimal subcircuits up to a certain number of inputs and,
2. using it during technology mapping to rewrite small logic blocks of larger networks.

Although this idea has been considered before [123], it was only ever able to capture abstract technology-independent costs such as the size or depth of subcircuits. Instead, we generate the database with an optimal physical design algorithm tuned to the desired target technology, thus incorporating all elements of potential final circuit costs. Consequently, our technology mapper's outputs inherently respect (configurable) inverter, buffer, splitter, and crossing costs,[II] thus they represent the final circuit layout much more closely and prevent overhead at the physical design stage.

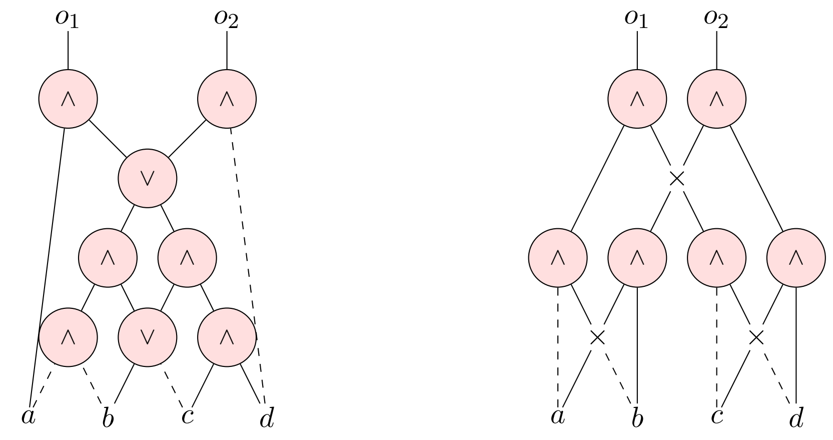


Figure 6.3: Two realizations of a network computing $o_1 = a \wedge b \wedge c \wedge d$ and $o_2 = \neg a \wedge \neg b \wedge \neg c \wedge \neg d$. Inverters are denoted by dashed edges and crossing cells are denoted by a $\times$ symbol.

To illustrate the need for technology-dependent optimizations in the logic synthesis stage, consider a circuit with input variables a, b, c, d that computes $o_1 = a \wedge b \wedge c \wedge d$ and

[II]Although we focus on these four cost functions because they represent important roadblocks to overcome in contemporary emerging technologies, our general approach applies to arbitrary cost functions as long as there exists a physical design algorithm for generating the optimal design database.

$o_2 = \neg a \wedge \neg b \wedge \neg c \wedge \neg d$. Suppose that we are optimizing for a simplified technology with AND2 and OR2 gates, which has no inversion cost, but needs path-balancing and planarization with crossings. While there is a mapped configuration with 8 cells (Figure 6.3 (left)), a naive technology-independent optimization might give a more compact representation with only 6 gates which need 9 cells in total including 3 additional crossing cells to meet the planarization constraint (Figure 6.3 (right)).

In the following, we outline our proposed approach to consider technology-specific constraints during logic synthesis. First, we describe the generation of the database of optimal subcircuits, which is a one-time computation. Then we present the algorithm for mapping an input logic network into a path-balanced, fanout-branched, and planarized circuit using the generated optimal substructures, while also supporting explicit inverters.

6.3.1 Generation of Optimal Subcircuits

Conventional Boolean rewriting has successfully provided the groundwork for utilizing databases of optimal subcircuits. Usually, NPN canonization is utilized to significantly reduce the database sizes [123]. Two single-output Boolean functions belong to the same NPN class if one can be translated into the other by (optionally) negating (N) the primary inputs, permuting (P) the primary inputs, and negating (N) the primary output. The representative of each class is its lexicographically smallest member. Since inverters are considered to be cost-free in AIGs and input permutations can be neglected because no sense of fixed topology is employed, NPN canonization is a strong tool for complexity reduction and optimization.

However, in beyond-CMOS technologies, inverters matter, and input permutations can only be altered using costly crossings. Therefore, only considering NPN representatives is insufficient as the costs of members belonging to the same NPN class might substantially differ in the final layouts.

Therefore, we propose exploring a middle ground between the exhaustive enumeration of all 2^{2^n} Boolean functions in n variables and their NPN representatives. Namely, we rely on a class that we call NN that respects input/output permutations but not primary inversions. The number of NN classes is greater by a factor of $n!$ compared to NPN. For example, while the number of 4-input NPN classes is 222, our 4-input database has $222 \cdot 4! = 5328$ entries, which is still much smaller than the number of 4-input functions ($2^{2^4} = 65536$).

For each canonized (lexicographically smallest) NN representative, we generate an optimal (with respect to the imposed cost functions) subcircuit layout in the target technology. To this end, we modified an open-source physical design algorithm [169, 171] to compute the optimal FCN circuit layouts under different primary input permutations. Since that algorithm is based on SMT solving, we enforce the input permutation $\pi : x_1 \succ x_2 \succ$

$\cdots \succ x_n$ by adding an additional constraint. Let p_{xt} be the Boolean variable that, when set to `1`, represents that node x is placed on layout tile t. To enforce that if a primary input is placed on some tile, any other primary input that follows in the permutation order must not be placed on a prior tile in the layout, the constraint is defined as:

$$\bigwedge_{t \in T, x \in \pi} (p_{xt} \implies \bigwedge_{t' \succ t, x \succ x'} \neg p_{x't'}).$$

The inclined reader is referred to [169] for an in-depth explanation of existing constraints for valid node placement, wire routing, crossing insertion, path balancing, etc. Finally, incremental SMT solver calls that iteratively increase the available layout area for the physical design process ensure optimality of the eventually found result. Symmetry breaking allows effective search space pruning, and highly specialized cardinality constraint engines in the utilized *Z3* solver [129] enable critical runtime reductions that keep the approach scalable up to ≈ 100 layout tiles, which is sufficient to realize all 4-input NN representatives.

6.3.2 Rewriting Using the Exact Database

This step decomposes the input network into small logic blocks and substitutes them using the appropriate optimum structures described in the previous section, while also synthesizing interconnections between logic blocks. The high-level pseudocode of the algorithm is given in Algorithm 6.1.

Decomposing into small logic blocks

Typical technology-mapping algorithms consider small cuts rooted at different nodes in the network and replace them with optimized versions, and when doing so, conventional algorithms give only minor importance to other fanouts of the cut leaves. However, to consider branching and planarization constraints of emerging technologies, when replacing a cut, it is important to know the relative positions of the other fanouts of the cut leaves concerning the part that is being replaced to preserve already instantiated (partial-)planarization. To this end, we 1. fix the decomposition into small logic blocks by mapping the network to 4-LUTs using `if -K 4` command of the logic synthesis tool ABC [34] (Line 1), and 2. fix the relative positions of those logic blocks by assigning ranks to LUTs and imposing an ordering of the LUTs in each rank (Line 2).

Initial path-balancing and crossing optimization

Before rewriting LUTs, the algorithm decides the locations of wires between non-consecutive LUT logic levels. For example, if there is a LUT a in level 1 which is a fanin of a LUT d in level 3 and if level 2 has two LUTs b and c in that order, for proper

Algorithm 6.1: Proposed technology mapping algorithm.

```
Input: Input network $N$ and database DB.
Output: A technology-mapped version of $N$.
1  $N_{\text{lut}} \leftarrow N$ mapped to a 4-LUT network with ABC.
2  Assign levels to nodes in $N_{\text{lut}}$ and fix the ordering of nodes.
3  $N_{\text{buf}} \leftarrow$ buffer inserted version of $N_{\text{lut}}$.
4  $N_{\text{xing}} \leftarrow$ crossing minimized version of $N_{\text{buf}}$.
5  $L \leftarrow$ number of logic levels in $N_{\text{xing}}$.
6  foreach level $\ell \in \{1, \ldots, L\}$ do
7      $DesiredOrder \leftarrow [\,]$.
8      foreach node $n$ of level $\ell$ do
9          Reorder fanins of $n$ to avoid self-crossings.
10         Update node function of $n$.
11         Append reordered fanins to $DesiredOrder$.
12     Construct buffer/splitter/crossing layers in $N_{\text{xing}}$ to achieve the signal order of
         $DesiredOrder$ at the outputs of level $\ell - 1$.
13     foreach node $n$ of level $\ell$ do
14         $(S, InvConfig) \leftarrow$ find best network structure and I/O inversion configuration
             for $n$ from DB.
15         Replace $n$ in $N_{\text{xing}}$ with $S$ after applying $InvConfig$.
16     Add buffers to $N_{\text{xing}}$ to balance the outputs at level $\ell + 1$.
17 return $N_{\text{xing}}$
```

planarization, the algorithm decides whether the wire from a to d goes through the space left of b, between b and c, or right of c. The initial path-balancing thus inserts buffers to denote such path propagation locations for each wire that connects non-consecutive LUT layers (Line 3). During LUT rewriting, these buffers are extended to buffer chains to meet the path-balancing constraint.

To minimize crossings, the initial buffers are inserted in a locally optimal way, keeping a fixed ordering of LUTs. I.e, for a LUT a in level ℓ, if a buffer needs to be inserted in level $\ell + 1$, it is placed at the location that minimizes the number of level-ℓ-to-level-$\ell + 1$ connections that cross the path from the LUT to the buffer. After buffer insertion, crossing optimization is performed for each level (Line 4) by swapping adjacent node pairs in each level as long as it leads to fewer crossings.

Substituting LUTs from the database entries

In the final step, the path-balanced network is reconstructed in a level-by-level fashion (Lines 6-16). Reconstructing level $\ell + 1$ consists of three main steps: 1. for each LUT in $\ell + 1$, their fanins are reordered to avoid crossings between pairs of their fanins, and the node functions are altered accordingly (Lines 9-10); 2. zero or more layers, each consisting of buffers/splitters/crossings, are inserted between level ℓ and $\ell + 1$ to obtain the fanins of level $\ell + 1$ in the correct order (Line 12); and 3. the LUTs in level $\ell + 1$ are replaced with the respective optimal structures from the database (Lines 14-15). The database includes entries for all 4-input NN classes but does not include all input/output inverted versions to avoid pre-computing the entire domain of 4-input functions. Hence, the algorithm considers all possible input/output inversions for LUT node functions and finds a match in the database. Then, the LUT is replaced with the found entry, after applying appropriate input/output inversions.

Run-time and space complexity The run-time and space complexity of our algorithm is dominated by the crossing insertion between two consecutive levels in Line 12. If the number of gates in layer ℓ is n_ℓ and $m_\ell = \max(n_\ell, n_{\ell-1})$, then the worst-case for this step would need $\mathcal{O}(m_\ell)$-many new layers each consisting of $\mathcal{O}(m_\ell)$ crossings/buffers (consider the case where outputs of level $\ell - 1$ are connected to the inputs of level ℓ in the opposite order). Thus the total run-time and space needed for this step is $\mathcal{O}(\sum_{\ell=1}^{L} m_\ell^2)$, which is $\mathcal{O}(n^2)$ in the worst-case where n is the size of the network. As the database of optimum substructures is computed for constant-sized functions, each database entry has constant size, and with proper indexing, the lookup is also constant time; hence replacing the LUTs with optimum structures increases the run-time by only a constant factor.

6.4 Experimental Evaluation

This section constitutes a quantitative evaluation of the proposed technology mapping algorithm. We demonstrate its applicability and compare it against a state-of-the-art technique. First, we describe the experimental setup in Section 6.4.1 before presenting and discussing the results in Section 6.4.2.

6.4.1 Experimental Setup

The proposed algorithm was implemented in C++ on top of the open-source tools *mockturtle* [154] and *fiction* [171] and evaluated using the *ISCAS85* benchmarks [38] and *EPFL Benchmark Suite* [11]. We generated a database of optimal FCN layouts—relying on state-of-the-art technology constraints [172]—implementing all 5328 canonical NN representatives as Verilog modules. (Total uncompressed size is 12MB.) We then

Table 6.1: Results of the proposed technology mapping approach on the ISCAS [38] and EPFL [11] benchmark suites.

	Benchmark Circuit					**State of the Art** [170]				**Proposed Approach**							
	Name	PI	PO	Gates	Depth	Total Nodes	Buffers	Crossings	CP	Total Nodes	Buffers	Crossings	CP	Runtime in sec.	Buffers impr. %	Crossings impr. %	CP impr. %
ISCAS85 [38]	c17	5	2	6	3	99	69	16	26	63	31	6	13	0.04	55.1	62.5	50.0
	c432	36	7	208	26	35 776	31 131	4201	701	14 910	12 170	1973	299	0.06	60.9	53.0	57.3
	c499	41	32	398	19	94 621	88 261	5456	1402	11 842	8754	1948	227	0.06	90.1	64.3	83.8
	c880	60	26	325	25	70 108	61 040	8438	1062	28 920	23 055	4593	457	0.07	62.2	45.6	57.0
	c1355	41	32	502	25	117 056	109 738	6198	1722	11 565	8521	1936	219	0.06	92.2	68.8	87.3
	c1908	33	25	341	27	77 950	71 177	5995	1201	17 201	14 026	2187	364	0.06	80.3	63.5	69.7
	c2670	157	64	716	20	323 824	281 067	41 267	2464	88 295	72 777	13 613	774	0.11	74.1	67.0	68.6
	c3540	50	22	1024	41	587 468	531 807	53 498	3280	97 627	65 925	28 312	977	0.11	87.6	47.1	70.2
	c5315	178	123	1776	37	1 864 282	1 710 369	150 222	5869	301 880	246 660	50 522	1504	0.27	85.6	66.4	74.4
	c6288	32	32	2337	120	988 542	939 626	42 447	8901	97 313	73 052	15 542	1280	0.15	92.2	63.4	85.6
	c7552	207	108	1469	26	1 481 318	1 351 266	126 753	5169	411 383	333 298	73 019	1797	0.36	75.3	42.4	65.2
EPFL [11]	adder	256	129	1020	255	794 313	708 447	83 316	4083	2 181 853	2 146 359	31 625	9350	1.65	−203.0	62.0	−129.0
	arbiter	256	129	11 839	87	61 392 432	56 342 578	5 025 982	36 160	7 432 960	6 661 959	731 089	11 646	6.10	88.2	85.5	67.8
	bar	135	128	3336	12	4 050 823	3 680 537	363 230	10 782	1 023 446	448 804	562 234	2369	0.63	87.8	−54.8	78.0
	cavlc	10	11	693	16	286 509	259 800	25 100	2333	86 999	54 839	29 403	760	0.09	78.9	−17.1	67.4
	ctrl	7	26	174	10	28 188	25 097	2667	654	4480	2689	1273	128	0.05	89.3	52.3	80.4
	dec	8	256	304	3	161 857	154 666	6871	1143	25 321	3859	19 894	215	0.06	97.5	−189.5	81.2
	i2c	147	142	1342	20	1 129 553	1 030 768	95 968	4568	294 762	238 114	52 198	1163	0.30	76.9	45.6	74.5
	int2float	11	7	260	16	48 219	42 793	4875	833	12 161	8175	3124	293	0.06	80.9	35.9	64.8
	max	512	130	2865	287	5 378 865	4 720 729	651 642	10 130	6 182 860	5 879 043	294 321	11 589	8.53	−24.5	54.8	−14.4
	priority	128	8	978	250	668 097	607 825	57 916	3477	290 810	273 901	13 407	2487	0.25	54.9	76.9	28.5
	router	60	30	257	54	54 074	45 627	7955	814	20 033	18 439	648	348	0.06	59.6	91.9	57.2
	sin	24	25	5416	225	8 237 614	7 711 879	514 234	16 990	1 145 537	934 129	192 740	6340	0.64	87.9	62.5	62.7
	voter	1001	1	13 758	70	53 955 839	50 500 625	3 421 110	48 864	3 375 273	2 734 141	601 139	5442	7.60	94.6	82.4	88.9
	div	128	128	57 247	4372		*out of memory*			104 918 980	101 310 752	3 409 196	249 354	56.49	—	—	—
	hyp	256	128	214 335	24 801		*out of memory*			1 128 917 685	1 029 593 313	98 519 782	1 226 868	1172.27	—	—	—
	log2	32	32	32 060	444		*out of memory*			24 816 049	20 493 947	4 215 965	47 005	10.22	—	—	—
	mem_ctrl	1204	1231	46 836	114		*out of memory*			371 071 747	341 555 500	29 369 279	108 019	577.26	—	—	—
	multiplier	128	128	27 062	274		*out of memory*			33 370 845	28 610 811	4 668 200	28 842	16.83	—	—	—
	sqrt	128	64	24 618	5058		*out of memory*			41 768 215	41 226 185	470 491	169 800	26.22	—	—	—
	square	64	128	18 484	250		*out of memory*			17 588 918	14 252 309	3 266 047	14 747	7.36	—	—	—
Weighted average															**84.5**	**74.5**	**65.2**

applied the proposed technology mapping algorithm to all circuits in the aforementioned benchmark suites. We measured the resulting gate-level costs concerning the number of buffers (including splitters), number of crossings, and critical path (CP) length, and compared them with the results of the best available (to best of our knowledge) large-scale FCN physical design algorithm that can handle layouts with more than 100 million tiles [170]. The implementation of [170] is publicly available [171], which enabled us to run all experiments with the same set of configurations. All evaluations were run on a MacBook Pro M1 with 10 CPU cores, 16 GPU cores, and 32 GB of RAM.

6.4.2 Results

To enable a fair comparison, we applied both our proposed algorithm and the state-of-the-art FCN algorithm [170] to all benchmarks, without performing any prior logic optimization.

The obtained results are shown in Table 6.1. It lists the initial properties of the benchmarks under *Benchmark Circuit*; the columns under *State of the Art* indicate the statistics of the FCN layouts generated by [170]; and those under *Proposed Approach* show results obtained from our technology mapping algorithm when applied to the FCN domain. For both algorithms, it lists the number of total nodes, the number of buffers (including splitters) and crossings, and the critical path (CP) length. The last three columns show relative improvements in buffer, crossing, and CP costs. The final row states a weighted average for the reductions in costs across all benchmarks (excluding those for which the state of the art could not generate a solution).

The proposed method consistently achieves over 50 % improvement in the CP length for all benchmarks except for three. A similar level of improvement is also evident in the numbers of crossings and buffers for most of the benchmarks within a similar run-time. The average reductions for the number of buffers, number of crossings, and CP length are 84.5 %, 74.5 %, and 65.2 %, respectively, which is a major improvement over the state of the art. Moreover, our method is more scalable as it yields results for the seven EPFL benchmarks on which the state of the art ran out of memory.

On the downside, a degradation of the CP length for benchmark ‘adder’, and a similar order of magnitude degradation of the number of crossings for benchmark ‘dec’ can be noticed, which appear to be outliers. However, it is to be noted that the benchmark ‘adder’ exhibits an improvement in the number of crossings, and the benchmark ‘dec’ shows an improvement in the CP length. Generally, an increase in the number of crossings results in an increase in the CP length, but, as the results for these outliers suggest, this is not always the case. The CP length is more related to the maximum number of crossings a single wire has than to the total number of crossings. That is, if there is a single wire that crosses m other wires and no other pairs of wires cross, the planarization needs at least m crossing layers and thus increases the CP delay by m levels. On the other hand, even if there are m crossings between two layers, if each wire only crosses a handful of other wires, that structure can be planarized with much fewer crossing layers, so the CP length will be small. Thus, to minimize the CP length, a better objective is to minimize the maximum number of crossings for any wire, rather than minimizing the total number of crossings.

Additionally, when our crossing optimization and planarization steps are applied directly to the 'adder' AIG without any LUT-mapping, it yields a much better CP length. This implies that LUT mapping results in an increased amount of crossings among the resulting LUT nodes, which, in turn, increases the number of logic levels due to crossings that occur in series. I.e., LUT mapping on 'adder' seems to over-optimize for LUT depth, inadvertently making it harder to planarize, because the LUT mapping stage is unaware of the technology constraints. Thus, it seems promising to conduct further research on technology-aware decomposition techniques, which will help mitigate such outlier situations.

6.5 Summary

Many technological implementations in the beyond-CMOS domain come with unconventional cost functions that are not respected by classical logic synthesis and, hence, cause significant overhead in the physical design stage.

In this chapter, we proposed an algorithm for technology mapping of beyond-CMOS circuitry that respects these unconventional cost functions via the application of a physical design database of optimal circuit layouts that is employed for logic rewriting, thus capturing cost factors that would otherwise remain transparent to the logic synthesis. Via an experimental evaluation, we showed that the proposed algorithm delivers average improvements of 84.5 %, 74.5 %, and 65.2 % for the number of buffers, the number of crossings, and the critical path length, respectively, as compared to a state-of-the-art physical design algorithm for FCN circuits. Furthermore, results could be obtained for the seven largest EPFL benchmark circuits on which the state of the art ran out of memory, proving our approach to be more scalable. Thereby, this work constitutes a major improvement for the design automation of several emerging beyond-CMOS technology classes, which enables the cost-effective realization and integration of large-scale circuits in this domain.

7 Conclusion

As technology continues to push the boundaries of digital systems, the demand for efficient, scalable, and adaptable design methodologies becomes increasingly critical. The field of logic synthesis and optimization plays a pivotal role in Electronic Design Automation (EDA), facilitating the continued advancement of digital circuits. This thesis addressed key challenges in logic synthesis and optimization, particularly as the field of CMOS is rapidly approaching its physical limits. To this end, this thesis considers two primary directions: scalable sequential synthesis for CMOS technologies and specialized synthesis techniques for emerging post-CMOS technologies. In this chapter, we summarize the major contributions of this thesis, reflect on their implications, and outline future directions for research to extend and deepen this work.

7.1 Scalable Sequential Logic Synthesis

We introduced a scalable algorithm for sequential logic synthesis that utilizes *sequential observability don't cares* (SODCs). SODCs extend the notion of *observability don't cares* (ODCs) by incorporating the concept of reachable states in sequential circuits. The proposed approach addresses the challenges associated with SODCs, particularly the dependencies between base and inductive cases in k-step induction. By leveraging a combination of redundancy removal and resubstitution under SODCs, the algorithm explores optimization opportunities that have been overlooked by prior methods.

The method was implemented in an industrial EDA tool and achieved significant improvements. Specifically, it demonstrated an average area reduction of 6.9% after technology mapping, with reductions of 2.89% and 1.43% in combinational and sequential areas, respectively, after post place-and-route. The rigorous proofs of correctness and empirical evaluations confirm the effectiveness of the approach in enhancing power-performance-area (PPA) metrics for advanced designs.

The experimental results demonstrate that the method is scalable and yields significant area improvements in both technology-mapped circuits and post place-and-route designs. While the runtime is non-negligible, the method proves valuable for area-critical applications, particularly as a high-effort optimization step.

Future Directions

While the proposed algorithm sets a robust foundation for SODC-based sequential logic synthesis, several avenues for future exploration exist:

Runtime Enhancements: The method's runtime can be further optimized by leveraging randomized or counter-example-guided simulation techniques to filter out invalid optimizations early.

Heuristics for Further PPA Gains: Exploring heuristics for optimization candidate ordering may expose further opportunities for improvement. Additionally, applying symmetry-breaking logic transformations and more advanced reachability assumptions could reveal additional optimization potential. **Integration of Advanced Assumption Models:** Assumptions in the current method are derived directly from candidate redundancies. Exploring richer assumption frameworks, possibly informed by formal verification tools or simulation, could uncover deeper optimization opportunities in circuits with complex feedback loops.

Extending Resubstitution Capabilities: The integration of resubstitution into the framework proved effective but was limited by the scope of divisors and the imposed constraints on network levels. Future improvements could involve dynamic divisor selection strategies or the adoption of SAT-based techniques to handle larger fanout cones efficiently.

7.2 Fanout-Bounded Logic Synthesis

This work introduced a comprehensive approach to Fanout-Bounded Synthesis (FBS) targeted at emerging technologies with strict fanout and path-balancing constraints, such as superconducting electronics. The problem is formulated as an ILP for fixed target delays, achieving global optimization for fanout-bounded networks while minimizing area. To complement the ILP, a scalable top-down heuristic is proposed for practical applications, yielding significant area and delay improvements.

The proposed methodology was particularly relevant for technologies like AQFP, where gates exhibit limited fanout capacity and require strict path balancing for correct operation. For the general FBS problem, the heuristic achieved an 11.82% area reduction compared to state-of-the-art methods without increasing delay. For the path-balanced setting, it achieved an average 8.76% delay improvement with a modest 0.5% area reduction, demonstrating its efficacy in optimizing both general and path-balanced designs.

Although primarily targeted at emerging technologies, the techniques developed in this work also provide valuable insights for CMOS technology. As CMOS nodes scale further into advanced geometries, fanout limitations and interconnect delays are becoming increasingly critical bottlenecks. The proposed FBS framework and heuristics can be adapted to optimize high-fanout nets and critical paths in CMOS designs, such as clock trees and reset networks. By drawing parallels between AQFP constraints and emerging challenges in CMOS, this work offers a foundation for next-generation CMOS logic synthesis techniques that address fanout and timing constraints more effectively.

Future Directions

We identify several promising directions for future research in fanout-bounded synthesis:

Enhancing ILP Scalability: While the ILP formulation achieves global optimization for small benchmarks, scaling it to industrial designs is non-trivial. From the solver side, advances in constraint pruning and parallel solving strategies could provide modest speedups. On the overall formulation of the ILP, a more pragmatic approach could involve partitioning the design into smaller subproblems and leveraging divide-and-conquer strategies to manage complexity.

Path-Balancing Heuristics: Extending the heuristic to incorporate more aggressive path-balancing optimizations could further reduce the number of splitters and buffers required, improving both area and delay metrics.

Generalization to Other Technologies: While the focus is on AQFP, the techniques could be adapted for other post-CMOS technologies with similar constraints, such as spintronics and quantum-dot cellular automata. However, the framework may require some non-trivial modifications to accommodate the unique characteristics of these technologies, such as the preference for fewer wire crossings in FCN circuits.

7.3 Logic Synthesis for AQFP Technology

This work presents a two-stage synthesis framework for optimizing AQFP circuits. The framework addresses unique challenges in AQFP technology, such as splitter and buffer requirements for fanout handling and path balancing. The first stage involves generating a database of minimum-area AQFP circuit structures for all single-output, 4-input logic functions under varying input arrival-time patterns. This database is then used in the second stage to synthesize larger networks by locally rewriting logic blocks with optimal structures from the database in topological order.

The proposed method achieves simultaneous optimization of logic gates, splitters, and buffers, capturing opportunities missed by conventional approaches that separate logic optimization from buffer-splitter handling. Experimental evaluations demonstrate over a 40% improvement in delay (critical path) and up to a 21% reduction in area (measured in Josephson Junctions, JJs) compared to prior work. The inclusion of majority-5 gates in the synthesis framework further enhances resource efficiency and delay reduction, providing a new level of optimization granularity for AQFP circuits.

This work establishes a new paradigm for AQFP circuit synthesis, emphasizing the importance of holistic optimization. By unifying logic and path balancing optimizations, it achieves substantial improvements in delay and area. This idea can be extended to other post-CMOS technologies where interconnect overhead is significant compared to logic resources due to technology-specific constraints. The results also underscore the utility of incorporating larger and more complex gates, such as majority-5, into AQFP circuit designs. These gates effectively reduce the need for additional logic layers, buffers, and splitters, streamlining the synthesis process and reducing resource consumption.

Future Directions

There are two main directions for future research based on the proposed AQFP synthesis framework:

Database Refinement and Expansion: Increasing the scope of the database by including a broader range of gate types, splitter configurations, and cost metrics could significantly enhance the flexibility and applicability of the proposed method. Moreover, while the current database is limited to 4-input functions, expanding it to support functions with larger support could further improve overall optimization quality. To keep the database size manageable, the target functions can be carefully selected based on their frequency of occurrence in real-world designs.

Improved Splitter-Tree Synthesis: The proposed method uses balanced splitter trees at the output of each logic block, which may not be the best choice in all cases. It is worth exploring dynamic splitter-tree synthesis strategies that adapt to the specific requirements of each logic block. For example, if certain fanouts of a logic block have more relaxed required times relative to others, an unbalanced splitter tree may be a better choice. Since this decision has to be made before the downstream logic is synthesized, heuristics or machine learning models can be used to predict the best splitter-tree configuration for each logic block.

7.4 Logic Synthesis for FCN Technologies

This work introduces a novel technology mapping algorithm tailored for beyond-CMOS technologies, emphasizing unconventional cost functions such as path-balancing buffers, fanout-branching splitters, planarizing crossings, and non-cost-free inverters. The approach leverages a database of optimal subcircuits generated using exact physical design techniques, enabling technology-aware mapping at the logic synthesis stage. By incorporating these constraints early, the method mitigates substantial overhead typically encountered during physical design.

Experimental evaluations demonstrate significant improvements across key metrics compared to state-of-the-art physical design approaches for FCN. The proposed method achieves average reductions of 84.5%, 74.5%, and 65.2% in the number of buffers, crossings, and critical path length, respectively. Moreover, it successfully maps circuits that exceed the memory capacity of existing algorithms, proving its scalability and robustness.

A key takeaway from this work is the importance of integrating technology-specific constraints into the logic synthesis stage, enabling a seamless transition to physical design for emerging technologies. By addressing critical challenges like path balancing, branching, and planarization upfront, the proposed method minimizes inefficiencies that arise when conventional synthesis flows are adapted to beyond-CMOS technologies. The scalability of the approach and its applicability to large and complex benchmarks make it a significant advancement in the design automation domain. The consistent performance improvements highlight its potential as a foundational tool for future design automation flows.

Future Directions

One immediate future research direction is to explore better interconnect optimization strategies for planarized and path-balanced technologies such as FCN. The proposed method first decomposes the network into logic blocks and processes them level by level. Between two levels of logic blocks, the interconnects are optimized using existing heuristic methods, which primarily target the minimization of the number of crossings. However, interconnect cost is also influenced by the number of buffers used for path balancing. In certain cases, the buffer cost can be significantly higher than the crossing cost, especially if a single wire passes through multiple crossings, as each such crossing adds a new layer of interconnects. As such, it is worth exploring methods that optimize interconnects while considering buffer cost as well. One possible heuristic is to minimize the maximum number of crossings a wire has to pass through, which indirectly minimizes the number of buffers required for path balancing.

7.5 Final Remarks

This thesis makes significant progress in addressing critical challenges in logic synthesis and optimization, particularly as digital design nears its physical and performance boundaries. By leveraging scalable sequential synthesis, fanout-bounded logic synthesis, and innovative approaches to post-CMOS technologies, this research enables substantial improvements in power, performance, and area (PPA) metrics. In summary, this thesis not only advances the state of the art in logic synthesis but also establishes a robust foundation for future research into scalable and technology-specific optimization methodologies.

Bibliography

[1] Apple introduces M2 Ultra.[Online June 2023] https://www.apple.com/newsroom/2023/06/apple-introduces-m2-ultra/.

[2] Opencores: https://opencores.org.

[3] TSMC's New 3nm Chip Wafers Priced at $20,000.[Online July 2023] https://www.siliconexpert.com/blog/tsmc-3nm-wafer/.

[4] David A. Papa, Igor L. Markov, David A Papa, and Igor L Markov. Physically-driven logic restructuring. *Multi-Objective Optimization in Physical Synthesis of Integrated Circuits*, pages 83–103, 2013.

[5] Akers. Binary decision diagrams. *IEEE Transactions on computers*, 100(6):509–516, 1978.

[6] Sheldon B Akers. Synthesis of combinational logic using three-input majority gates. In *3rd Annual Symposium on Switching Circuit Theory and Logical Design*, pages 149–158, 1962.

[7] S. Amarel, G. Cooke, and R. O. Winder. Majority Gate Networks. *IEEE Trans. on Electronic Computers*, EC-13(1):4–13, 1964.

[8] L. Amarù, P. Gaillardon, and G. De Micheli. Majority-Inverter Graph: A novel data-structure and algorithms for efficient logic optimization. In *Design Automation Conference*, pages 1–6, 2014.

[9] L. Amarú, P. Gaillardon, and G. 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.

[10] L Amarú, A Ajami, S Chen, Y Zhang, TL Tung, T Arifin, T Liu, M Pan, G Naveen, JC Vujkovic, et al. First demonstration of a superconducting electronics microcontroller rtl-to-gdsii flow. In *Government Microcircuit Appl. Crit. Technol. Conf.(GOMACTech)*, pages 1–4, 2021.

[11] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. The EPFL combinational benchmark suite. In *IWLS*, 2015.

[12] 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.

[13] Luca Amarú, Pierre-Emmanuel Gaillardon, Anupam Chattopadhyay, and Giovanni De Micheli. A sound and complete axiomatization of majority-n logic. *IEEE Transactions on Computers*, 65(9):7. 2889–2895, 2016.

[14] Luca Amarú, Mathias Soeken, Patrick Vuillod, Jiong Luo, Alan Mishchenko, Pierre-Emmanuel Gaillardon, Janet Olson, Robert Brayton, and Giovanni De Micheli. Enabling exact delay synthesis. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 352–359. IEEE, 2017.

[15] Luca Amarú, Mathias Soeken, Patrick Vuillod, Jiong Luo, Alan Mishchenko, Janet Olson, Robert Brayton, and Giovanni De Micheli. Improvements to boolean resynthesis. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 755–760. IEEE, 2018.

[16] Neal G. Anderson and Sanjukta Bhanja, editors. *Field-Coupled Nanocomputing - Paradigms, Progress, and Perspectives*, volume 8280 of *Lecture Notes in Computer Science*. Springer, 2014.

[17] Yuki Ando, Ryo Sato, Masamitsu Tanaka, Kazuyoshi Takagi, Naofumi Takagi, and Akira Fujimaki. Design and demonstration of an 8-bit bit-serial rsfq microprocessor: Core e4. *IEEE Transactions on Applied Superconductivity*, 26(5):1–5, 2016.

[18] Christopher Ayala and Nobuyuki Yoshikawa. personal communication.

[19] Christopher L Ayala, Ro Saito, Tomoyuki Tanaka, Olivia Chen, Naoki Takeuchi, Yuxing He, and Nobuyuki Yoshikawa. A semi-custom design methodology and environment for implementing superconductor adiabatic quantum-flux-parametron microprocessors. *Superconductor Science and Technology*, 33(5):054006, 2020.

[20] Christopher L Ayala, Tomoyuki Tanaka, Ro Saito, Mai Nozoe, Naoki Takeuchi, and Nobuyuki Yoshikawa. Mana: A monolithic adiabatic integration architecture microprocessor using 1.4-zj/op unshunted superconductor josephson junction devices. *IEEE Journal of Solid-State Circuits*, 56(4):1152–1165, 2020.

[21] D. Baneres, J. Cortadella, and M. Kishinevsky. Layout-Aware Gate Duplication and Buffer Insertion. In *Design, Automation and Test in Europe*, pages 1–6, 2007.

[22] Karen A Bartlett, Robert K Brayton, Gary D Hachtel, Reily M Jacoby, Christopher R Morrison, Richard L Rudell, Alberto Sangiovanni-Vincentelli, and A Wang. Multi-level logic minimization using implicit don't cares. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(6):723–740, 1988.

[23] Luca Benini and Giovanni De Micheli. A survey of boolean matching techniques for library binding. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2(3):193–226, 1997.

[24] G. H. Bernstein, A. Imre, V. Metlushko, A. Orlov, L. Zhou, L. Ji, G. Csaba, and W. Porod. Magnetic QCA systems. *Microelectronics Journal*, 36(7):619–624, 2005.

[25] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *TACAS*, pages 193–207. Springer, 1999.

[26] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

[27] Per Bjesse and Koen Claessen. SAT-Based Verification without State Space Traversal. In *FMCAD*, page 372–389, 2000.

[28] E. Blair and C. Lent. Clock Topologies for Molecular Quantum-Dot Cellular Automata. *Journal of Low Power Electronics and Applications*, 8(3), 2018.

[29] George Boole. *The mathematical analysis of logic.* CreateSpace Independent Publishing Platform, 1847.

[30] Daniel Brand. Redundancy and don't cares in logic synthesis. *IEEE Transactions on Computers*, 100(10):947–952, 1983.

[31] Alexander L Braun and David Christopher Harms. RQL majority gates, and gates, and or gates, September 25 2018. US Patent 10,084,454.

[32] R. K. Brayton and Alan Mishchenko. Sequential Rewriting and Synthesis. In *IWLS*, 2007.

[33] R.K. Brayton. Compatible observability don't cares revisited. In *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)*, pages 618–623, 2001. doi: 10.1109/ICCAD.2001.968725.

[34] Robert Brayton and Alan Mishchenko. ABC: An academic industrial-strength verification tool. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 24–40, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[35] Robert K Brayton. The decomposition and factorization of boolean expressions. *ISCA-82*, pages 49–54, 1982.

[36] Robert K Brayton, Gary D Hachtel, Lane A Hemachandra, A Richard Newton, and Alberto Luigi M Sangiovanni-Vincentelli. A comparison of logic minimization strategies using espresso: An apl program package for partitioned logic minimization. In *Proceedings of the International Symposium on Circuits and Systems*, pages 42–48, 1982.

[37] 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.

[38] F. Brgles and Hideo Fujiwara. A neutral netlist of 10 combinational circuits and a target translator in fortran. In *ISCAS*, 1985.

[39] Frank Markham Brown. *Boolean reasoning: the logic of Boolean equations.* Courier Corporation, 2003.

[40] Randal E Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.

[41] Ruizhe Cai, Olivia Chen, Ao Ren, Ning Liu, Caiwen Ding, Nobuyuki Yoshikawa, and Yanzhi Wang. A majority logic synthesis framework for adiabatic quantum-flux-parametron superconducting circuits. In *ACM Great Lakes Symposium on VLSI*, pages 189–194, 2019.

[42] Ruizhe Cai, Olivia Chen, Ao Ren, Ning Liu, Nobuyuki Yoshikawa, and Yanzhi Wang. A buffer and splitter insertion framework for adiabatic quantum-flux-parametron superconducting circuits. In *Int'l Conf. on Computer Design*, pages 429–436, 2019.

[43] Vehbi Calayir, Dmitri E. Nikonov, Sasikanth Manipatruni, and Ian A. Young. Static and Clocked Spintronic Circuit Design and Simulation With Performance Analysis Relative to CMOS. *IEEE Trans. on Circuits and Systems I: Regular Papers*, 61(2):393–406, 2014.

[44] Alessandro Tempia Calvino and Giovanni De Micheli. Depth-Optimal Buffer and Splitter Insertion and Optimization in AQFP Circuits. In *Asia and South Pacific Design Automation Conference*, page 152–158, 2023.

[45] Alessandro Tempia Calvino and Giovanni De Micheli. Scalable logic rewriting using don't cares. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2024.

[46] Caio Araujo T Campos, Abner L. Marciano, Omar P. Vilela Neto, and Frank Sill Torres. USE: A Universal, Scalable, and Efficient Clocking Scheme for QCA. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 35(3), 2015.

[47] Michael Case, Jason Baumgartner, Hari Mony, and Robert Kanzelman. Optimal redundancy removal without fixedpoint computation. In *FMCAD*, pages 101–108, 2011.

[48] Michael L. Case, Victor N. Kravets, Alan Mishchenko, and Robert K. Brayton. Merging nodes under sequential observability. In *DAC*, pages 540–545, 2008.

[49] Satrajit Chatterjee. *On algorithms for technology mapping.* University of California, Berkeley, 2007.

[50] Satrajit Chatterjee, Alan Mishchenko, Robert K Brayton, Xinning Wang, and Timothy Kam. Reducing structural bias in technology mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(12):2894–2903, 2006.

[51] Olivia Chen, Ruizhe Cai, Yanzhi Wang, Fei Ke, Taiki Yamae, Ro Saito, Naoki Takeuchi, and Nobuyuki Yoshikawa. Adiabatic quantum-flux-parametron: Towards building extremely energy-efficient circuits and systems. *Scientific reports*, 9(1): 1–10, 2019.

[52] Jason Cong and Yuzheng Ding. Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 13, 1994.

[53] Jason Cong and Yean-Yow Hwang. Structural gate decomposition for depth-optimal technology mapping in lut-based fpga designs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 5(2):193–225, 2000.

[54] R. P. Cowburn and M. E. Welland. Room Temperature Magnetic Quantum Cellular Automata. *Science*, 287(5457), 2000.

[55] Maurizio Damiani and Giovanni De Micheli. Observability don't care sets and boolean relations. In *ICCAD*, volume 90, pages 502–505, 1990.

[56] Maurizio Damiani and Giovanni De Micheli. Don't care set specifications in combinational and synchronous logic circuits. *IEEE transactions on computer-aided design of integrated circuits and systems*, 12(3):365–388, 1993.

[57] E.S. Davidson. An algorithm for NAND decomposition under network constraints. *IEEE Transactions on Computers*, C-18(12):1098–1109, 1969. doi: 10.1109/T-C.1969.222593.

[58] Giovanni De Micheli. Synchronous logic synthesis: Algorithms for cycle-time minimization. *IEEE TCAD*, 10(1):63–73, 1991.

[59] P Fiser, I Halecek, and J Schmidt. Are xors in logic synthesis really necessary. In *IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 138–143, 2017.

[60] Coenrad Johann Fourie, Kyle Jackman, Matthys M Botha, Sasan Razmkhah, Pascal Febvre, Christopher Lawrence Ayala, Qiuyun Xu, Nobuyuki Yoshikawa, Erin Patrick, Mark Law, et al. Coldflux superconducting EDA and TCAD tools project: Overview and progress. *IEEE Transactions on Applied Superconductivity*, 29(5):1–7, 2019.

[61] Rongliang Fu, Mengmeng Wang, Yirong Kan, Nobuyuki Yoshikawa, Tsung-Yi Ho, and Olivia Chen. A Global Optimization Algorithm for Buffer and Splitter

Insertion in Adiabatic Quantum-Flux-Parametron Circuits. In *Asia and South Pacific Design Automation Conference*, page 769–774, 2023.

[62] M. Golumbic. Combinatorial Merging. *IEEE Trans. on Computers*, 25(11):1164–1167, 1976.

[63] Mrinal Goswami, Anindan Mondal, Mahabub Hasan Mahalat, Bibhash Sen, and Biplab K. Sikdar. An Efficient Clocking Scheme for Quantum-dot Cellular Automata. *International Journal of Electronics Letters*, 8(1), 2020.

[64] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL https://www.gurobi.com.

[65] W. Haaswijk, M. Soeken, A. Mishchenko, and G. De Micheli. SAT-based exact synthesis: Encodings, topology families, and parallelism. *IEEE Trans. on CAD of Integrated Circuits and Systems*, pages 1–1, 2019.

[66] Winston Haaswijk, Mathias Soeken, Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. A novel basis for logic rewriting. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 151–156. Ieee, 2017.

[67] Winston Jason Haaswijk. *SAT-Based Exact Synthesis for Multi-Level Logic Networks.* PhD thesis, EPFL, Lausanne, 2019.

[68] Gary D Hachtel and Fabio Somenzi. *Logic synthesis and verification algorithms.* Springer Science & Business Media, 2005.

[69] Leo Hellerman. A catalog of three-variable or-invert and and-invert logical circuits. *IEEE Transactions on Electronic Computers*, (3):198–223, 1963.

[70] Quentin P Herr, Anna Y Herr, Oliver T Oberg, and Alexander G Ioannidis. Ultra-low-power superconductor logic. *Journal of applied physics*, 109(10):103903, 2011.

[71] Gage Hills, Christian Lau, Andrew Wright, Samuel Fuller, Mindy D Bishop, Tathagata Srimani, Pritpal Kanhaiya, Rebecca Ho, Aya Amer, Yosi Stein, et al. Modern microprocessor built from complementary carbon nanotube transistors. *Nature*, 572(7771):595–602, 2019.

[72] D Scott Holmes, Andrew L Ripple, and Marc A Manheimer. Energy-efficient superconducting computing—Power budgets and requirements. *IEEE Trans. on Applied Superconductivity*, 23(3), 2013.

[73] D Scott Holmes, Alan M Kadin, and Mark W Johnson. Superconducting computing in large-scale hybrid systems. *Computer*, 48(12):34–42, 2015.

[74] H James Hoover, Maria M Klawe, and Nicholas J Pippenger. Bounding fan-out in logical networks. *Journal of the ACM (JACM)*, 31(1):13–18, 1984.

[75] Bo Hu, Yosinori Watanabe, Alex Kondratyev, and Malgorzata Marek-Sadowska. Gain-based technology mapping for discrete-size cell libraries. In *Proceedings of the 40th annual Design Automation Conference*, pages 574–579, 2003.

[76] Chao-Yuan Huang, Yi-Chen Chang, Ming-Jer Tsai, and Tsung-Yi Ho. An Optimal Algorithm for Splitter and Buffer Insertion in Adiabatic Quantum-Flux-Parametron Circuits. In *Int'l Conf. on Computer-Aided Design*, page 1–8, 2021.

[77] J. Huang, M. Momenzadeh, L. Schiano, M. Ottavi, and F. Lombardi. Tile-based QCA Design Using Majority-like Logic Primitives. *JETC*, 1(3):163–185, 2005.

[78] Zheng Huang, Lingli Wang, Yakov Nasikovskiy, and Alan Mishchenko. Fast Boolean matching based on NPN classification. In *2013 International Conference on Field-Programmable Technology (FPT)*, pages 310–313, 2013. doi: 10.1109/FPT.2013.6718374.

[79] T. Huff, H. Labidi, M. Rashidi, L. Livadaru, T. Dienel, R. Achal, W. Vine, J. Pitters, and R. A. Wolkow. Binary Atomic Silicon Logic. *Nature Electronics*, 1:636–643, 2018.

[80] David A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proc. IRE*, 40(9):1098–1101, 1952.

[81] Aaron P Hurst, Alan Mishchenko, and Robert K Brayton. Fast minimum-register retiming via binary maximum-flow. In *FMCAD*, pages 181–187, 2007.

[82] John D. Joannopoulos, Pierre R. Villeneuve, and Shanhui Fan. Photonic crystals. *Solid State Communications*, 102(2-3):165–173, 1997.

[83] Steven G. Johnson, Pierre R. Villeneuve, Shanhui Fan, and John D. Joannopoulos. Linear waveguides in photonic-crystal slabs. *Physical Review B*, 62, 2000.

[84] Shrirang K Karandikar and Sachin S Sapatnekar. Logical effort based technology mapping. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, pages 419–422. IEEE, 2004.

[85] Maurice Karnaugh. The map method for synthesis of combinational logic circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 72(5):593–599, 1953.

[86] R. W. Keyes and R. Landauer. Minimal Energy Dissipation in Logic. *IBM Journal of Research and Development*, 14(2), 1970.

[87] Donald E Knuth. *The art of computer programming, volume 4A: combinatorial algorithms, part 1.* 2011.

[88] Donald Ervin Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability.* Addison-Wesley, 2015.

[89] 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.

[90] Victor N. Kravets and Alan Mishchenko. Sequential logic synthesis using symbolic bi-decomposition. In *DATE*, pages 1458–1463, 2009.

[91] Gleb Krylov and Eby G Friedman. Asynchronous dynamic single-flux quantum majority gates. *IEEE Transactions on Applied Superconductivity*, 30(5):1–7, 2020.

[92] A. Kuehlmann, V. Paruthi, F. Krohm, and M.K. Ganai. Robust Boolean reasoning for equivalence checking and functional property verification. *IEEE TCAD*, 21(12): 1377–1394, 2002.

[93] Andreas Kuehlmann. Dynamic transition relation simplification for bounded property checking. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, pages 50–57. IEEE, 2004.

[94] Yuji Kukimoto, Robert K Brayton, and Prashant Sawkar. Delay-optimal technology mapping by DAG covering. In *DAC*, pages 348–351, 1998.

[95] R. Landauer. Irreversibility and Heat Generation in the Computing Process. *IBM Journal of Research and Development*, 5, 1961.

[96] Chang-Yeong Lee. Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal*, 38(4):985–999, 1959.

[97] Siang-Yun Lee, Heinz Riener, and Giovanni De Micheli. Beyond local optimality of buffer and splitter insertion for AQFP circuits. In Rob Oshana, editor, *DAC '22: 59th ACM/IEEE Design Automation Conference, San Francisco, California, USA, July 10 - 14, 2022*, pages 445–450. ACM, 2022. doi: 10.1145/3489517.3530661. URL https://doi.org/10.1145/3489517.3530661.

[98] Siang-Yun Lee, Heinz Riener, Alan Mishchenko, Robert K. Brayton, and Giovanni De Micheli. A Simulation-Guided Paradigm for Logic Synthesis and Verification. *IEEE TCAD*, 41(8):2573–2586, 2022.

[99] Charles E Leiserson and James B Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1-6):5–35, 1991.

[100] C. S. Lent and P. D. Tougaw. A device architecture for computing with quantum dots. *Proceedings of the IEEE*, 85(4), April 1997. ISSN 0018-9219.

[101] C. S. Lent, P. D. Tougaw, W. Porod, and G. H. Bernstein. Quantum Cellular Automata. *Nanotechnology*, 4(1):49, 1993.

[102] C. S. Lent, B. Isaksen, and M. Lieberman. Molecular quantum-dot cellular automata. *Journal of the American Chemical Society*, 125(4):1056–1063, 2003.

[103] Zhuo Li, David A. Papa, Charles J. Alpert, Shiyan Hu, Weiping Shi, Cliff Sze, and Ying Zhou. Ultra-Fast Interconnect Driven Cell Cloning for Minimizing Critical Path Delay. In *Proc. 19th Int'l Symposium on Physical Design*, page 75–82, New York, NY, USA, 2010.

[104] K. K. Likharev and V. K. Semenov. RSFQ logic/memory family: a new Josephson-junction technology for sub-terahertz-clock-frequency digital systems. *IEEE Trans. on Applied Superconductivity*, 1(1):3–28, 1991.

[105] Mordor Intelligence LLP. Electronic design automation tools (EDA) market - growth, trends, forecasts (2020 - 2025). https://www.researchandmarkets.com/reports/4534513/electronic-design-automation-tools-eda-market, May 2020.

[106] Dewmini Sudara Marakkalage and Giovanni De Micheli. Fanout-Bounded Logic Synthesis for Emerging Technologies - A Top-Down Approach. In *International Logic Synthesis Workshop (IWLS)*, pages 1–6, 2022.

[107] Dewmini Sudara Marakkalage and Giovanni De Micheli. Fanout-Bounded Logic Synthesis for Emerging Technologies - A Top-Down Approach. In *Design, Automation and Test in Europe*, pages 1–6, 2023.

[108] Dewmini Sudara Marakkalage and Giovanni De Micheli. Fanout-bounded logic synthesis for emerging technologies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[109] Dewmini Sudara Marakkalage, Eleonora Testa, Heinz Riener, Alan Mishchenko, Mathias Soeken, and Giovanni De Micheli. Three-input gates for logic synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(10):2184–2188, 2020.

[110] Dewmini Sudara Marakkalage, Heinz Riener, and Giovanni De Micheli. Optimizing adiabatic quantum-flux-parametron (AQFP) circuits using an exact database. In *NANOARCH*, pages 1–6, 2021.

[111] Dewmini Sudara Marakkalage, Siang-Yun Lee, Wille Robert, and Giovanni De Micheli. Technology mapping for beyond-cmos circuitry with unconventional cost functions. In *International Logic Synthesis Workshop (IWLS)*, 2023.

[112] Dewmini Sudara Marakkalage, Eleonora Testa, Giulia Meuli, Walter Lau Neto, Alan Mishchenko, Giovanni De Micheli, and Luca Amarù. Scalable sequential logic synthesis using observability don't care conditions. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, page under review. IEEE, 2024.

[113] Dewmini Sudara Marakkalage, Eleonora Testa, Walter Lau Neto, Alan Mishchenko, Giovanni De Micheli, and Luca Amarù. Scalable sequential optimization under observability don't cares. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2024.

[114] Dewmini Sudara Marakkalage, Marcel Walter, Siang-Yun Lee, Robert Wille, and Giovanni De Micheli. Technology mapping for beyond-cmos circuitry with unconventional cost functions. In *2024 IEEE 24th International Conference on Nanotechnology (NANO)*, pages 51–56. IEEE, 2024.

[115] João P Marques-Silva and Karem A Sakallah. Boolean satisfiability in electronic design automation. In *DAC*, pages 675–680, 2000.

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

[117] Giulia Meuli, Mathias Soeken, and Giovanni De Micheli. Xor-and-inverter graphs for quantum compilation. *npj Quantum Information*, 8(1):7, 2022.

[118] Giovanni De Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education, 1994.

[119] H. S. Miller and R. O. Winder. Majority-Logic Synthesis by Geometric Methods. *IRE Trans. on Electronic Computers*, EC-11(1):89–90, 1962.

[120] Alan Mishchenko and Robert K. Brayton. Scalable logic synthesis using a simple circuit structure. In *IWLS*, pages 15–22, 2006.

[121] Alan Mishchenko, Satrajit Chatterjee, Roland Jiang, and Robert K. Brayton. FRAIGs: A unifying representation for logic synthesis and verification. Technical report, UC Berkeley, 2005.

[122] Alan Mishchenko, Satrajit Chatterjee, Robert Brayton, and Niklas Een. Improvements to combinational equivalence checking. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 836–843, 2006.

[123] Alan Mishchenko, Satrajit Chatterjee, and Robert K. Brayton. DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In Ellen Sentovich, editor, *DAC*, pages 532–535, 2006.

[124] Alan Mishchenko, Sungmin Cho, Satrajit Chatterjee, and Robert Brayton. Combinational and sequential mapping with priority cuts. In *2007 IEEE/ACM International Conference on Computer-Aided Design*, pages 354–361, 2007. doi: 10.1109/ICCAD.2007.4397290.

[125] Alan Mishchenko, Michael Case, Robert Brayton, and Stephen Jang. Scalable and scalably-verifiable sequential synthesis. In *ICCAD*, 2008.

[126] Alan Mishchenko, Robert K. Brayton, Jie-Hong R. Jiang, and Stephen Jang. Scalable don't-care-based logic optimization and resynthesis. *ACM TRETS*, 4(4): 34:1–34:23, 2011.

[127] Eric Mlinar, Stephen Whiteley, Anton Belov, Song Chen, Luca Amaru, Tong Liu, Yalan Zhang, Taufik Arifin, Min Pan, Troy Barbee, et al. An rtl-to-gdsii flow for single flux quantum circuits based on an industrial eda toolchain. *IEEE Transactions on Applied Superconductivity*, 33(5):1–7, 2023.

[128] H. Mony, J. Baumgartner, V. Paruthi, and R. Kanzelman. Exploiting suspected redundancy without proving it. In *DAC*, pages 463–466, 2005.

[129] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008.

[130] Oleg A Mukhanov. Energy-efficient single flux quantum technology. *IEEE Transactions on Applied Superconductivity*, 21(3):760–769, 2011.

[131] Rajeev Murgai. On the Global Fanout Optimization Problem. In *Int'l Conf. on Computer-Aided Design*, page 511–515, 1999.

[132] Saburo Muroga. *Threshold Logic and its Applications.* New York: Wiley-Interscience, 1971.

[133] 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*, 38(10):1404–1424, 1989.

[134] G. Pasandi and M. Pedram. PBMap: A path balancing technology mapping algorithm for single flux quantum logic circuits. *IEEE Transactions on Applied Superconductivity*, 29(4):1–14, 2019. doi: 10.1109/TASC.2018.2880343.

[135] J. Pitters et al. Atomically Precise Manufacturing of Silicon Electronics. *ACS Nano*, 2024.

[136] Willard V Quine. The problem of simplifying truth functions. *The American mathematical monthly*, 59(8):521–531, 1952.

[137] Dayane Alfenas Reis, Caio Araújo T. Campos, Thiago Rodrigues B. S. Soares, Omar Paranaiba V. Neto, and Frank Sill Torres. A methodology for standard cell design for QCA. In *ISCAS*, pages 2114–2117, 2016.

[138] Giovanni V Resta, Alessandra Leonhardt, Yashwanth Balaji, Stefan De Gendt, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. Devices and circuits using novel 2-d materials: a perspective for future vlsi systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(7):1486–1503, 2019.

[139] Heinz Riener, Winston Haaswijk, Alan Mishchenko, Giovanni De Micheli, and Mathias Soeken. On-the-fly and DAG-aware: Rewriting Boolean networks with exact synthesis. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1649–1654, 2019. doi: 10.23919/DATE.2019.8715185.

[140] Heinz Riener, Siang-Yun Lee, Alan Mishchenko, and Giovanni De Micheli. Boolean rewriting strikes back: Reconvergence-driven windowing meets resynthesis. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 395–402. IEEE, 2022.

[141] J. Paul Roth and R. M. Karp. Minimization over Boolean graphs. *IBM Journal of Research and Development*, 6(2):227–238, 1962. doi: 10.1147/rd.62.0227.

[142] 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.

[143] Ro Saito, Christopher L. Ayala, Olivia Chen, Tomoyuki Tanaka, Tomohiro Tamura, and Nobuyuki Yoshikawa. Logic Synthesis of Sequential Logic Circuits for Adiabatic Quantum-Flux-Parametron Logic. *IEEE Trans. on Applied Superconductivity*, 31 (5):1–5, 2021.

[144] Ro Saito, Christopher L. Ayala, and Nobuyuki Yoshikawa. Buffer Reduction Via N-Phase Clocking in Adiabatic Quantum-Flux-Parametron Benchmark Circuits. *IEEE Trans. on Applied Superconductivity*, 31(6):1–8, 2021.

[145] Nikhil Saluja and Sunil P Khatri. A robust algorithm for approximate compatible observability don't care (CODC) computation. In *DAC*, pages 422–427, 2004.

[146] Hamid Savoj, Alan Mishchenko, and Robert Brayton. m-Inductive Property of Sequential Circuits. *IEEE TCAD*, 35(6):919–930, 2015.

[147] Claude E Shannon. A symbolic analysis of relay and switching circuits. *Electrical Engineering*, 57(12):713–723, 1938.

[148] F. Sill Torres, P. A. Silva, G. Fontes, M. Walter, J. A. M. Nacif, R. Santos Ferreira, O. P. Vilela Neto, J. F. Chaves, R. Wille, P. Niemann, D. Große, and R. Drechsler. On the Impact of the Synchronization Constraint and Interconnections in Quantum-dot Cellular Automata. *Microprocessors and Microsystems*, 76:103–109, 2020.

[149] Ellen M Sentovich Kanwar Jit Singh, Luciano Lavagno Cho Moon Rajeev Murgai, and Robert K Brayton Alberto Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. *University of California, Berkeley*, 94720:4, 1992.

[150] Gordon L Smith, Ralph J Bahnsen, and Harry Halliwell. Boolean comparison of hardware and flowcharts. *IBM Journal of Research and Development*, 26(1): 106–116, 1982.

[151] M. Soeken, W. Haaswijk, E. Testa, A. Mishchenko, L. Amarú, R. K. Brayton, and G. De Micheli. Practical exact synthesis. In *Design, Automation and Test in Europe*, pages 309–314, 2018.

[152] Mathias Soeken, Alan Mishchenko, Ana Petkovska, Baruch Sterin, Paolo Ienne, Robert K Brayton, and Giovanni De Micheli. Heuristic npn classification for large functions using aigs and lexsat. In *Theory and Applications of Satisfiability Testing–SAT 2016: 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings 19*, pages 212–227. Springer, 2016.

[153] Mathias Soeken, Luca Gaetano Amarù, 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. doi: 10.1109/TCAD.2017.2664059.

[154] Mathias Soeken, Heinz Riener, Winston Haaswijk, Eleonora Testa, Bruno Schmitt, Giulia Meuli, Fereshte Mozafari, Siang-Yun Lee, Alessandro Tempia Calvino, Dewmini Sudara Marakkalage, and Giovanni De Micheli. The EPFL logic synthesis libraries, 2022.

[155] Fabio Somenzi. Cudd: Cu decision diagram package release 2.3. 0. 1998.

[156] A. Srivastava, R. Kastner, and M. Sarrafzadeh. Timing driven gate duplication: complexity issues and algorithms. In *Int'l Conf. on Computer-Aided Design*, pages 447–450, 2000.

[157] Naoki Takeuchi, Dan Ozawa, Yuki Yamanashi, and Nobuyuki Yoshikawa. An adiabatic quantum flux parametron as an ultra-low-power logic device. *Superconductor Science and Technology*, 26(3):035010, 2013.

[158] Naoki Takeuchi, Yuki Yamanashi, and Nobuyuki Yoshikawa. Adiabatic quantum-flux-parametron cell library adopting minimalist design. *Journal of Applied Physics*, 117(17), 2015.

[159] Masamitsu Tanaka, Atsushi Kitayama, Tomohito Koketsu, Masato Ito, and Akira Fujimaki. Low-energy consumption RSFQ circuits driven by low voltages. *IEEE transactions on applied superconductivity*, 23(3):1701104–1701104, 2013.

[160] 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 *NANOARCH*, pages 15–20. Ieee, 2016.

[161] Eleonora Testa, Mathias Soeken, Luca Amarú, and Giovanni De Micheli. Reducing the multiplicative complexity in logic networks for cryptography and security applications. In *Design Automation Conference*, pages 1–6, 2019.

[162] Eleonora Testa, Luca Amaru, Mathias Soeken, Alan Mishchenko, Patrick Vuillod, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. Extending Boolean Methods for Scalable Logic Synthesis. *IEEE Access*, 8, 2020.

[163] Eleonora Testa, Siang-Yun Lee, Heinz Riener, and Giovanni De Micheli. Algebraic and Boolean optimization methods for AQFP superconducting circuits. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ASPDAC '21, page 779–785, New York, NY, USA, 2021. ISBN 9781450379991. doi: 10.1145/3394885.3431606.

[164] Eleonora Testa, Dewmini Sudara Marakkalage, Michael Quayle, Sudipta Kundu, Abhishek Kumar, Diptanshu Ghosh, Giulia Meuli, Giovanni De Micheli, and Luca Amaru. Enabling scalable sequential synthesis and formal verification in an industrial flow. In *International Logic Synthesis Workshop (IWLS)*, 2024.

[165] J. Timler and C. S. Lent. Power Gain and Dissipation in Quantum-dot Cellular Automata. *Journal of Applied Physics*, 91(2), 2002.

[166] Frank Sill Torres, M. Walter, R. Wille, D. Große, and R. Drechsler. Synchronization of Clocked Field-Coupled Circuits. In *IEEE-NANO*, 2018.

[167] Jeng-Liang Tsai, Lizheng Zhang, and Charlie Chung-Ping Chen. Statistical timing analysis driven post-silicon-tunable clock-tree synthesis. In *Int'l Conf. on Computer-Aided Design*, pages 575–581, 2005.

[168] C.A.J. van Eijk. Sequential equivalence checking based on structural similarities. *IEEE TCAD*, 19(7):814–819, 2000.

[169] Marcel Walter, Robert Wille, Daniel Große, Frank Sill Torres, and Rolf Drechsler. An exact method for design exploration of quantum-dot cellular automata. In *DATE*, pages 503–508, 2018.

[170] Marcel Walter, Robert Wille, Frank Sill Torres, Daniel Große, and Rolf Drechsler. Scalable design for field-coupled nanocomputing circuits. In *ASP-DAC*, 2019.

[171] Marcel Walter, Robert Wille, Frank Sill Torres, Daniel Große, and Rolf Drechsler. fiction: An open source framework for the design of field-coupled nanocomputing circuits, 2019.

[172] Marcel Walter, Samuel Sze Hang Ng, Konrad Walus, and Robert Wille. Hexagons are the bestagons: design automation for silicon dangling bond logic. In *DAC*, pages 739–744, 2022.

[173] S. R. Whiteley and J. Kawa. Progress toward VLSI-capable EDA tools for superconductive digital electronics. In *2019 IEEE International Superconductive Electronics Conference (ISEC)*, pages 1–3, 2019. doi: 10.1109/ISEC46533.2019.8990931.

[174] Clifford Wolf, Johann Glaser, and Johannes Kepler. Yosys-a free verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, volume 97, 2013.

[175] R. A. Wolkow, L. Livadaru, J. Pitters, M. Taucer, P. Piva, M. Salomons, M. Cloutier, and B. V. C. Martins. *Silicon Atomic Quantum Dots Enable Beyond-CMOS Electronics*, pages 33–58. Springer, 2014.

[176] Saeyang Yang. *Logic synthesis and optimization benchmarks user guide: version 3.0.* Microelectronics Center of North Carolina (MCNC), 1991.

[177] Mingfei Yu, Dewmini Sudara Marakkalage, and Giovanni De Micheli. Garbled circuits reimagined: Logic synthesis unleashes efficient secure computation. *Cryptography*, 7(4):61, 2023.

[178] He-Teng Zhang and Jie-Hong R. Jiang. SFO: A Scalable Approach to Fanout-Bounded Logic Synthesis for Emerging Technologies. In *Design Automation Conference*, pages 1–6, 2020.

[179] Yuanliang Zhang, Yao Zhang, and Baojun Li. Optical switches and logic gates based on self-collimated beams in two-dimensional photonic crystals. *Optics Express*, 15 (15):9287–9292, 2007.

[180] Qi Zhu, Nathan Kitchen, Andreas Kuehlmann, and Alberto Sangiovanni-Vincentelli. Sat sweeping with local observability don't-cares. In *Proceedings of the 43rd Annual Design Automation Conference*, pages 229–234, 2006.

Marakkalage Dewmini Sudara

Lausanne, Switzerland

dewmini.marakkalage@epfl.ch   mdsudara

Research Interests

Logic synthesis, electronic design automation, synthesis and optimizations for emerging technologies

Education

Sept 2020 - Present	**École polytechnique fédérale de Lausanne (EPFL)** *Ph.D. in Computer Science* ◦ Advisor: Prof. Giovanni De Micheli.	Lausanne, Switzerland
Sept 2018 - Aug 2020	**École polytechnique fédérale de Lausanne (EPFL)** *Master of Science in Computer Science* ◦ Specialized in *Computer Engineering*. ◦ Cumulative GPA - 5.62 out of 6.	Lausanne, Switzerland
Oct 2011 - Mar 2016	**University of Moratuwa** *Honours Degree of Bachelor of the Science of Engineering* ◦ Specialized in *Electronic and Telecommunication Engineering*. ◦ Cumulative GPA - 3.8 out of 4.2 (First class).	Moratuwa, Sri Lanka

Publications

2024 **Marakkalage, D. S.**, Testa, E.,Meuli G., Neto, W. L., Mishchenko, A., De Micheli, G., Amarú, L., "Scalable Sequential Logic Synthesis using Observability Don't Care Conditions", In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) 2024, (under review).*

2024 **Marakkalage, D. S.**, Walter, M., Lee, S-Y., Wille, R., De Micheli, G., "Technology Mapping for Beyond-CMOS Circuitry with Unconventional Cost Functions", In *EEE 24th International Conference on Nanotechnology (NANO) 2024, Available at: https://ieeexplore.ieee.org/document/10628909.*

2024 E. Testa, **D. S. Marakkalage**, M. Quayle, S. Kundu, A. Kumar, D. Ghosh, G. Meuli, G. De Micheli, L. Amaru, "Enabling Scalable Sequential Synthesis and Formal Verification in an Industrial Flow", In *International Workshop on Logic and Synthesis (IWLS) 2024.*

2024 **Marakkalage, D. S.**, Testa, E., Neto, W. L., Mishchenko, A., De Micheli, G., Amarú, L., "Scalable Sequential Optimization Under Observability Don't Cares", In *Design, Automation & Test in Europe Conference & Exhibition (DATE) 2024, Available at: https://ieeexplore.ieee.org/document/10546595.* ***Best Paper Award Nominee.***

2024 Bairamkulov, R., Lee, S. Y., Calvino, A. T., **Marakkalage, D. S.**, Yu, M., De Micheli, G., "Technology-Aware Logic Synthesis for Superconducting Electronics", In *Design, Automation & Test in Europe Conference & Exhibition (DATE) 2024, Available at: https://ieeexplore.ieee.org/document/10546721.*

2023 Yu, M., **Marakkalage, D. S.**, De Micheli G., "Garbled Circuits Reimagined: Logic Synthesis Unleashes Efficient Secure Computation", In *Cryptography 2023, Available at: https://www.mdpi.com/2410-387X/7/4/61.*

2023 **Marakkalage, D. S.**, De Micheli, G., "Fanout-Bounded Logic Synthesis for Emerging Technologies", In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) 2023, Available at: https://ieeexplore.ieee.org/document/10342810.*

2023 **Marakkalage, D. S.**, Walter, M., Lee, S-Y., Wille, R., De Micheli, G., "Technology Mapping for Beyond-CMOS Circuitry with Unconventional Cost Functions", In *International Workshop on Logic and Synthesis (IWLS) 2023.* ***Best Student Paper Award Nominee.***

2023 **Marakkalage, D. S.**, De Micheli, G., "Fanout-Bounded Logic Synthesis for Emerging Technologies - A Top-Down Approach", In *Design, Automation & Test in Europe Conference & Exhibition (DATE) 2023, Available at: https://ieeexplore.ieee.org/document/10137314.* ***Best Paper Award Nominee.***

2022 **Marakkalage, D. S.**, De Micheli, G., "Fanout-Bounded Logic Synthesis for Emerging Technologies - A Top-Down Approach", In *International Workshop on Logic and Synthesis (IWLS) 2022.*

2022 Meuli, G., Possani, V., Singh, R., Lee, S. Y., Calvino, A. T., **Marakkalage, D. S.**, Vuillod, P., Amaru, L., Chase, S., Kawa, J., De Micheli, G., "Majority-based Design Flow for AQFP Superconducting Family", In *Design, Automation & Test in Europe Conference & Exhibition (DATE) 2021, Available at: https://ieeexplore.ieee.org/document/9774558.*

2021 **Marakkalage, D. S.**, Riener, H., De Micheli, G., "Optimizing Adiabatic Quantum-Flux-Parametron (AQFP) Circuits using an Exact Database", In *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH) 2021, Available at: https://ieeexplore.ieee.org/document/9642241.*

2021 **Marakkalage, D. S.**, Riener, H., De Micheli, G., "Optimizing Adiabatic Quantum-Flux-Parametron (AQFP) Circuits using Exact Methods", In *International Workshop on Logic and Synthesis (IWLS) 2021.*

2021 **Marakkalage, D. S.**, Testa, E., Riener, H., Mishchenko, A., Soeken, M., De Micheli, G., "Three-input gates for logic synthesis", In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) 2021, Available at: https://ieeexplore.ieee.org/document/9233431.*

2020 **Marakkalage, D. S.**, Testa, E., Riener, H., Mishchenko, A., Soeken, M., De Micheli, G., "Three-input gates for logic synthesis", In *International Workshop on Logic and Synthesis (IWLS) 2020.*

Work Experience

Mar 2023 - Aug 2023 **Research Intern** Synopsys Switzerland LLC, Zurich, Switzerland

- Developed sequential logic optimization flows.

Mar 2020 - Aug 2020 **Research Intern** Synopsys SARL, Grenoble, France

- Supervisors: Dr. Patrick Vuillod (Synopsys) and Prof. Giovanni De Micheli (EPFL).
- Worked on Master's thesis project based on logic synthesis and optimizations.
- Developed majority-based logic synthesis algorithms for Fusion Compiler.

Sept 2018 - Aug 2019 **Master's Research Scholar** Processor Architecture Laboratory, EPFL

- Supervisors: Prof. Paolo Ienne and Lana Josipović.
- Developed a verification tool in C++ for high-level synthesis from C to VHDL, which can automatically generate VHDL testbenches, simulate them in Modelsim, and verify outputs.
- Generalized an existing VHDL implementation of a hardware load-store queue, ported it to Chisel3, and added AXI protocol support (including the support for out-of-order transaction completions) to the Chisel3 implementation.

Feb 2017 - Jun 2018 **Lecturer** University of Moratuwa, Sri Lanka

- Conducted lectures and tutorials for the course Analog Electronics.
- Conducted tutorials and laboratory experiments, and supervised mini-projects for undergraduate courses Basic Electronics, Introduction to Telecommunication, Digital Electronics, Electronic Product Design and Manufacture, Laboratory Practice I, Laboratory Practice II, Electronics I, and Electronics III.

May 2016 - Jan 2017 **Research Assistant** Singapore University of Technology & Design, Singapore

- Supervisor: Prof. David Braun.
- Performed mathematical analysis of optimal excitation of nonlinear parametric oscillators and proved that the amplification is bounded even under arbitrarily small weak-dissipation.

Oct 2014 - Mar 2015 **Research Intern** Singapore University of Technology & Design, Singapore

- Supervisor: Prof. David Braun.
- Performed numerical simulation of a state-dependent controller for a nonlinear parametric oscillator governed by the Mathieu equation and developed an electronic platform to experimentally validate the simulations.
- Got acknowledged on the following research article: Braun, David J. "Optimal Parametric Feedback Excitation of Nonlinear Oscillators." Physical Review Letters 116.4 (2016)

Skills

Programming	C++, C, Python, Java, Scala, Mathematica, MATLAB, Simulink.
HDL and Simulation	ModelSim, VHDL, Chisel3, Verilog.
EDA Tools	Synopsys Design Compiler, Cadence Virtuoso, Cadence Innovus, Intel Quartus Prime, Vivado Design Suite, Synopsys Galaxy Custom Designer, HSPICE.
Other Technical Tools	Programming tools for PIC/Atmel microcontrollers and ARM microprocessors, OrCAD, Proteus ISIS/ARES, Altium Designer, ANSYS Simplorer, Eagle CAD, NI LabVIEW.
Miscellaneous	Excellent analytic, algorithmic, and problem-solving skills. Good communication, teaching, and interpersonal skills. Working knowledge in Windows, Linux, and OSX platfroms.

Achievements

- EPFL EDIC Doctoral Fellowship 2020.
- EPFL Master's Research Scholarship 2018.
- Distinction (2010) and Higher Distinction (2009) in *Sri Lankan Mathematics Competition* (Organized by Sri Lanka Olympiad Mathematics Foundation).

Teaching Assistance

- Design Technologies for Integrated Systems, M.Sc. course, Fall 2022, Fall 2023, EPFL.
- Digital System Design, B.Sc. course, Spring 2022, Spring 2021, EPFL.
- Linear Algebra, B.Sc. course, Fall 2021, EPFL.

Professional Service

- Reviewer for the Design Automation Conference (DAC) in 2022 and 2021.
- Reviewer for the International Conference On Computer Aided Design (ICCAD) in 2020.
- Reviewer for IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) in 2022, 2023, and 2024.
- Reviewer for the International Workshop on Logic and Synthesis (IWLS) in 2023.
- Reviewer for the Design, Automation & Test in Europe Conference & Exhibition (DATE) in 2024.
- Graduate Student Member at IEEE.