ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ενισχυτική Μάθηση για την Αυτοματοποιημένη Σχεδίαση Ψηφιακών Ολοκληρωμένων Κυκλωμάτων

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Απόστολος Στεφανίδης, ΑΕΜ: 70334

Επιβλέπων Καθηγητής: Γεώργιος, Δημητρακόπουλος, Καθηγητής,
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Δ.Π.Θ.

Ξάνθη, 2025

ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ενισχυτική Μάθηση για την Αυτοματοποιημένη Σχεδίαση Ψηφιακών Ολοκληρωμένων Κυκλωμάτων

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Απόστολος Στεφανίδης, ΑΕΜ: 70334

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ

Μέλη της Συμβουλευτικής Επιτροπής:

Επιβλέπων Καθηγητής: Γεώργιος, Δημητρακόπουλος, Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Δημοκρίτειο Πανεπιστήμιο Θράκης

Μέλος 2: Ιωάννης, Καραφυλλίδης, Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Δημοκρίτειο Πανεπιστήμιο Θράκης

Μέλος 3: Χαρίδημος, Βέργος, Καθηγητής, Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής, Πανεπιστήμιο Πατρών

Μέλη της Εξεταστικής Επιτροπής:

Μέλος 4: Γεώργιος, Συρακούλης, Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Δημοκρίτειο Πανεπιστήμιο Θράκης

Μέλος 5: Ιωάννης, Ανδρεάδης, Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Δημοκρίτειο Πανεπιστήμιο Θράκης

Μέλος 6: Βασίλειος, Παλιουράς, Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών, Πανεπιστήμιο Πατρών

Μέλος 7: Λάζαρος, Παπαδόπουλος, Επίκουρος Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Δημοκρίτειο Πανεπιστήμιο Θράκης

Η παρούσα διδακτορική διατριβή υποβλήθηκε στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Δημοκρίτειου Πανεπιστημίου Θράκης για την απόκτηση του διδακτορικού τίτλου σπουδών.

Ξάνθη, 2025

DEMOCRITUS UNIVERSITY OF THRACE
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Integrated Circuits Design Automation with Reinforcement Learning

DOCTORAL THESIS

Apostolos Stefanidis, Registration Number: 70334

COMMITTEE OF EXAMINERS

Members of the Advisory Committee:

Supervisor: Giorgos, Dimitrakopoulos, Professor, Department of Electrical and Computer Engineering, Democritus University of Thrace

Member 2: Ioannis, Karafyllidis, Professor, Department of Electrical and Computer Engineering, Democritus University of Thrace

Member 3: Haridimos, Vergos, Professor, Department of Computer Engineering and Informatics, University of Patras

Members of the Committee of Examiners:

Member 4: Georgios, Sirakoulis, Professor, Department of Electrical and Computer Engineering, Democritus University of Thrace

Member 5: Ioannis, Andreadis, Professor, Department of Electrical and Computer Engineering, Democritus University of Thrace

Member 6: Vasileios, Paliouras, Professor, Department of Electrical and Computer Engineering, University of Patras

Member 7: Lazaros, Papadopoulos, Assistant Professor, Department of Electrical and Computer Engineering, Democritus University of Thrace

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy (Ph.D.), Department of Electrical and Computer Engineering, Democritus University of Thrace

Xanthi, 2025

ΑΝΑΦΟΡΑ ΣΤΗΝ ΤΗΡΗΣΗ ΤΩΝ ΑΚΑΔΗΜΑΪΚΩΝ ΑΡΧΩΝ ΔΕΟΝΤΟΛΟΓΙΑΣ

Η παρούσα εργασία με τίτλο «Ενισχυτική Μάθηση για την Αυτοματοποιημένη Σχεδίαση Ψηφιακών Ολοκληρωμένων Κυκλωμάτων» είναι πρωτότυπη και πραγματοποιήθηκε από τον φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών με Αρ. Μητρώου ΑΕΜ 70334 στο Εργαστήριο Ολοκληρωμένων Κυκλωμάτων του τομέα Ηλεκτρονικής, υπό την επίβλεψη του καθηγητή Γεώργιου Δημητρακόπουλου. Η συγγραφή της εργασίας/διατριβής πραγματοποιήθηκε εξολοκλήρου από τον φοιτητή Απόστολο Στεφανίδη, υπό την καθοδήγηση και τις υποδείξεις του επιβλέποντά του.

Βεβαιώνεται ότι, κατά την εκπόνηση και τη συγγραφή της εργασίας/διατριβής του, ο φοιτητής τήρησε τα προβλεπόμενα από τον νόμο και τον αντίστοιχο εσωτερικό κανονισμό του Τμήματος, σεβάστηκε πλήρως τις Αρχές της Ακαδημαϊκής Ηθικής και του Κώδικα Δεοντολογίας, οι οποίες απαγορεύουν την παραποίηση των ερευνητικών/πειραματικών αποτελεσμάτων, την αναφορά ψευδών στοιχείων, την κατάχρηση της διανοητικής ιδιοκτησίας τρίτων και τη λογοκλοπή και ότι έγινε με σεβασμό στις αρχές.

Περίληψη

Η συνεχώς αυξανόμενη πολυπλοκότητα και το μέγεθος των σύγχρονων ολοκληρωμένων κυκλωμάτων απαιτούν νέες μεθοδολογίες οι οποίες θα βελτιστοποιούν την επίδοση των μεθόδων Αυτοματοποιημένης Σχεδίασης Υλικού (Electronic Design Automation, EDA). Οι παραδοσιακές μέθοδοι φυσικής σύνθεσης βασίζονται συχνά σε ευρετικές προσεγγίσεις και σε σταθερές ροές βελτιστοποίησης, οι οποίες είναι εν δυνάμει μη βέλτιστες, και γενικεύονται δύσκολα σε διαφορετικά κυκλώματα. Η πρόσφατη πρόοδος στον τομέα της Μηχανικής Μάθησης, και συγκεκριμένα της Ενισχυτικής Μάθησης (Reinforcement Learning, RL), προσφέρει μια ισχυρή εναλλακτική επιλογή. Η ενισχυτική μάθηση επιτρέπει τη λήψη προσαρμοσμένων αποφάσεων μέσω συνεχούς ανάδρασης, καθιστώντας την ιδανική για την πλοήγηση στους τεράστιους διακριτούς χώρους βελτιστοποίησης που χαρακτηρίζουν τα προβλήματα στο χώρο του EDA. Αυτή η διατριβή εξερευνεί την ενσωμάτωση της ενισχυτικής μάθησης σε δύο κρίσιμα πεδία της ψηφιακής σχεδίασης: στην ικανοποίηση περιορισμών χρονισμού στη φυσική σχεδίαση, και στη σύνθεση προσεγγιστικών αριθμητικών κυκλωμάτων.

Το πρώτο μέρος της διατριβής εστιάζει στο κλείσιμο χρονισμού (ικανοποίηση περιορισμών χρονισμού), μια κεντρική πρόκληση στη φυσική σχεδίαση. Προτείνουμε μια καινοτόμη μεθοδολογία βελτιστοποίησης η οποία συνδυάζει ένα πλαίσιο μοντελοποίησης κόστους βασισμένο στη μέθοδο Lagrangian Relaxation (LR) με έναν αλγόριθμο Multi-Armed Bandit (MAB), ο οποίος καθοδηγεί τη επιλογή των ευρετικών μεθόδων βελτιστοποίησης χρονισμού που θα εφαρμοστούν. Αυτή η υβριδική προσέγγιση επιτρέπει την αυτόνομη εκτέλεση μιας εκτενούς συλλογής μεθόδων βελτιστοποίησης, οι οποίες περιλαμβάνουν την επιλογή μεγέθους πυλών και καταχωρητών, την προσθήκη buffer, την εναλλαγή συνδέσεων στην είσοδο μιας πύλης, την συγχώνευση ή τον διαχωρισμό πυλών, και την ανάθεση χρόνου άφιξης του σήματος ρολογιού. Η κάθε ευρετική μέθοδος επιλέγεται δυναμικά, λαμβάνοντας υπόψη το πλαίσιο στο οποίο θα εφαρμοστεί, με βάση το ιστορικό των επιβραβεύσεων που έλαβε ως προς την βελτίωση που προκάλεσε στον χρονισμό του κυκλώματος, σε σχέση με το υπολογιστικό κόστος. Το αποτέλεσμα είναι μία πλήρως προσαρμοστική και αυτορυθμιζόμενη ροή βελτιστοποίησης χωρίς ανάγκη για χειροκίνητο συντονισμό.

Το δεύτερο μέρος της διατριβής διερευνά τη χρήση της ενισχυτικής μάθησης για το σχεδιασμό προσεγγιστικών αριθμητικών κυκλωμάτων στοχευμένα σε συγκεκριμένες εφαρμογές, με έμφαση στους αθροιστές παράλληλου προθέματος. Η προσεγγιστική αριθμητική γίνεται ολοένα και πιο χρήσιμη σε τομείς όπου η ενεργειακή απόδοση έχει προτεραιότητα έναντι της πλήρους αριθμητικής ακρίβειας. Προτείνουμε μια μέθοδο ενισχυτικής μάθησης η οποία βελτιστοποιεί ταυτόχρονα την αποδοτικότητα του κυκλώματος και την αριθμητική επίδοση σε επίπεδο εφαρμογής. Το σύστημα ενισχυτικής μάθησης εξερευνά τον χώρο λύσεων των προσεγγιστικών αθροιστών λαμβάνοντας ανατροφοδότηση τόσο από μετρικές σύνθεσης υλικού όσο και από τη συμπεριφορά ως προς την αριθμητική ακρίβεια στην εφαρμογή. Σε αντίθεση με προηγούμενες δημοσιεύσεις που βασίζονται σε προκαθορισμένες αρχιτεκτονικές, η μέθοδός μας χρησιμοποιεί μια καινοτόμο τεχνική που παράγει όλες τις έγκυρες λύσεις που ικανοποιούν συγκεκριμένους περιορισμούς. Αυτό επιτρέπει την εξερεύνηση βέλτιστων αρχιτεκτονικών με διάφορα χαρακτηριστικά καθυστέρησης, εμβαδού,

ισχύος και ακρίβειας σε μια ποικιλία πραγματικών εφαρμογών.

Συνοψίζοντας, η παρούσα διατριβή αναδεικνύει την αποτελεσματικότητα της ενισχυτικής μάθησης και της στατιστικής λήψης αποφάσεων για τον επαναπροσδιορισμό των παραδοσιακών μεθοδολογιών EDA. Με την έξυπνα συντονισμένη λήψη αποφάσεων βελτιστοποίησης τόσο στο φυσικό σχεδιασμό, όσο και στη σύνθεση αριθμητικών κυκλωμάτων, οι προτεινόμενες μέθοδοι επιτυγχάνουν ανταγωνιστικά αποτελέσματα, μειώνοντας ταυτόχρονα σημαντικά τον χρόνο χειροκίνητου σχεδιασμού. Αυτές οι συνεισφορές ανοίγουν το δρόμο για πιο αυτόνομα, αποδοτικά και έξυπνα εργαλεία σχεδιασμού σε μελλοντικά συστήματα VLSI.

Λέξεις Κλειδιά: Εργαλεία αυτοματοποιημένης σχεδίασης υλικού, Φυσική σχεδίαση, Ενισχυτική μάθηση, Προσεγγιστική αριθμητική, Αθροιστές παράλληλου προθέματος, Βελτιστοποιήσεις χρονισμού, Σχεδίαση χαμηλής ισχύος

Abstract

The increasing complexity and scale of modern integrated circuits demand new methodologies that can optimize both performance and design effort in Electronic Design Automation (EDA). Traditional approaches in physical synthesis often rely on hand-crafted heuristics and fixed optimization flows, which can be suboptimal and difficult to generalize across design contexts. Recent advances in machine learning and particularly in Reinforcement Learning (RL), offer a powerful alternative. RL enables adaptive decision-making through continuous learning from feedback, making it an ideal candidate for navigating the vast and discrete optimization spaces characteristic of EDA problems. This thesis explores the integration of RL into two critical areas of digital design: timing closure in physical design and approximate datapath synthesis.

The first part of this thesis focuses on timing closure, a core challenge in the physical design flow. We propose a novel optimization methodology that combines a Lagrangian Relaxation (LR)-based cost modeling framework with a Multi-Armed Bandit (MAB) algorithm to guide the selection of timing optimization heuristics. This hybrid approach enables the autonomous execution of a rich set of optimization techniques, including gate sizing, flip-flop resizing, buffer insertion, pin swapping, gate merging/splitting, and the application of useful clock skew. Each heuristic is dynamically selected in a context-aware manner, based on its historical reward in terms of timing improvement versus computational cost. The result is a fully adaptive and self-tuning optimization loop that eliminates the need for manual flow tuning.

The second part of the thesis investigates the use of RL for the design of application-specific approximate arithmetic units, with a focus on parallel prefix adders. Approximate computing is increasingly relevant for domains where energy efficiency is prioritized over full numerical accuracy. We introduce an RL framework that co-optimizes hardware efficiency and application-level performance. An RL agent learns to explore the design space of approximate adders by receiving feedback from both hardware synthesis metrics and application error behavior. In contrast to previous approaches that rely on predefined architectural templates, our method employs a novel enumerative synthesis technique that generates all structurally valid solutions satisfying given constraints. This allows the discovery of Pareto-optimal designs with diverse trade-offs in delay, area, power, and accuracy across a variety of real-world applications.

In summary, this thesis demonstrates the power of Reinforcement Learning and statistical decision-making to redefine traditional EDA methodologies. By intelligently orchestrating optimization decisions in both physical design and datapath synthesis, the proposed methods achieve state-of-the-art results while significantly reducing manual design effort. These contributions pave the way for more autonomous, efficient, and intelligent design tools in future VLSI systems.

Keywords: Electronic Design Automation, Physical design, Reinforcement Learning, Approximate computing, Parallel prefix adders, Timing optimizations, Low-power design

Acknowledgements

I would like to express my deepest gratitude to everyone who supported me throughout the completion of this thesis. This has been a long and rigorous journey, and it would not have been possible without the support and encouragement of everyone involved.

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Giorgos Dimitrakopoulos, for his invaluable guidance, mentorship, and support throughout the years. Our collaboration has been instrumental in my academic and professional development, and I can clearly recognize how much I have grown since the beginning of this journey. I am especially grateful for his patience and encouragement, which consistently helped me believe that completing this thesis was within reach. Beyond academic guidance, Prof. Dimitrakopoulos also provided me with numerous research and professional opportunities that have been pivotal for my future steps.

Furthermore, I would like to thank the members of my advisory committee, Prof. Ioannis Karafyllidis and Prof. Haridimos Vergos, for their thoughtful evaluation of my work and the valuable suggestions they provided for its improvement. Their technical and academic expertise contributed to significant enhancements in the quality of this work.

My first experience in the professional world came through my internship, and I would like to thank everyone who made it a positive and fulfilling chapter in my journey. I am deeply grateful to David Chinnery, Ankur Sharma, Pavlos Mattheakis, Nikita Nikitin, Laurent Masse-Navete, Javier de San Pedro Martin, Gilles Brenet, Dario Socci, and Hamid Bouzouzou, who is sadly no longer with us, for the engaging collaboration and insightful conversations we shared.

A special thank you goes to all my friends and colleagues from the IC-Lab at our university. I am truly grateful to Dimitris Mangiras, Dionysis Filippas, Dimitris Konstantinou, Haris Takakis, Karyofyllis Patsidis, Tasos Martidis, Christos Gkantidis, Ioannis Seitanidis, and Ioanna Zoumpoulidou for the harmonious collaboration, the exchange of ideas, and all the much-needed shared laughs in the office.

Finally, I would like to express my deepest gratitude to my close family. My parents, Zisis and Paraskevi, have supported me in every possible way throughout this journey. They always listened to my concerns with empathy, and never stopped believing in me, offering the emotional support I needed during the most demanding moments of this thesis. And last but certainly not least, I dedicate this thesis to my wife, Ma-An, who has been by my side from the very beginning, supporting me with unwavering patience, encouragement, and love.

Contents

1 Introduction

The semiconductor industry has been steadily growing in the past years due to the constant reduction in transistor size and consequently the increase of integration. Modern chips can fit and effectively use millions of transistors in a small area, leading to a tremendous increase in the achieved frequencies and computational power. This advancement of technology does not come without a cost however. Building more complex chips requires harder designer choices, advanced methods for ensuring functional correctness, mapping the design to technology nodes in the physical world and converting it to a ready to be fabricated chip. Electronic Design Automation (EDA) tools address this challenge by automatically transforming a Register-Transfer Level (RTL) design into a fabrication-ready chip through a series of essential processes.

Digital Integrated Circuits (IC) design tools, broadly categorized as EDA, have always been the connection between technological improvements and the designer. Simulation and verification tools support early architecture exploration and microarchitecture design phases, where the design is verified that it correctly performs the desired functions and that it achieves the needed performance in terms of computational efficiency. Physical synthesis tools, which are the focus of this project, transform verified microarchitecture-level designs into chips ready for fabrication. The logic synthesizer maps the design to a gate-level netlist using the target chip technology. Placement and routing engines then complete the floorplanning, position the cells, synthesize the clock tree, and connect the cells with appropriate wiring. Incremental timing and power optimizers are integrated within these major placement and routing steps, simplifying timing closure (i.e., meeting the designer's timing constraints) while achieving significant reductions in power consumption and chip area.

As fabrication processes continue to scale down, ICs grow increasingly more complex. As any design engineer knows, this rising complexity leads to longer design turnaround times (TAT) and makes it more difficult to achieve improvements in power, performance, and area (PPA) beyond thresholds that justify a new product in a new technology node. In EDA, results are everything. To stay competitive, trade-offs in PPA or TAT are unacceptable—even when optimizing for multiple operating modes and various process-voltage-temperature (PVT) corners.

A scalable physical synthesis engine must support deeper exploration of the solution space to improve Quality-of-Results (QoR) in terms of PPA, while maintaining reasonable runtime and memory usage, even for increasingly complex design blocks. Below 7nm technology nodes, blocks with more than 10 million cells are becoming common, and ICs can easily integrate close to hundred of such blocks. Keeping up with this complexity demands higher design efficiency, which today is most effectively addressed through machine learning techniques.

The application of Reinforcement Learning (RL) in physical design for EDA presents a transformative opportunity to overcome longstanding challenges in design optimization. Traditional physical design tools rely heavily on heuristic algorithms and expert-tuned flows, which often struggle to generalize across varying design blocks, technology nodes,

and performance constraints. In contrast, RL offers a data-driven, adaptive approach able to learn effective design strategies through trial and feedback. By formulating the design process as a sequential decision-making problem, RL agents can be trained to make context-aware decisions, such as placement, sizing, or routing actions that lead to improved PPA outcomes while respecting timing and manufacturability constraints.

Moreover, RL is inherently well-suited to address the multi-objective and non-convex nature of physical design problems. As modern IC designs scale to billions of transistors and integrate dozens of complex functional blocks, manual tuning and fixed-rule methods become increasingly untenable. RL's capacity to explore vast solution spaces and adapt policies to new design contexts enables the development of scalable, generalizable optimization engines. These capabilities not only promise to enhance QoR and reduce design TAT, but also pave the way for intelligent, self-improving design systems that learn continuously from historical design data, marking a significant shift toward autonomous EDA workflows.

1.1 Major steps in physical synthesis

The VLSI design flow refers to the process required for the construction of a chip. A high level overview of this process can be seen on Fig. 1.1. The steps required for transforming the design into a physically realizable layout are collectively known as Physical Design or Physical Synthesis, and their goal is to deliver a chip with optimized area and power consumption which also satisfies the designer's timing constraints.

RTL design and verification: The RTL design process begins with a well-defined specification that outlines the intended functionality, interface behavior, and performance targets of the design, such as the desired operating frequency. Based on this specification, engineers make high-level architectural decisions and implement the design in a hardware description language (HDL) such as Verilog or VHDL, or alternatively, use High-Level Synthesis (HLS) tools that convert high-level programming languages like C++ or SystemC into RTL. Once the RTL implementation is in place, it enters a rigorous simulation and verification phase to ensure functional correctness. During this stage, EDA tools apply extensive test scenarios to evaluate the design's response to a wide range of input conditions. Any discrepancies between the expected and actual outputs must be diagnosed and corrected. Verification efforts also aim to maximize code coverage to ensure that all parts of the RTL have been adequately exercised, thereby minimizing the likelihood of undetected bugs propagating into later stages of the design flow.

Logic synthesis is a crucial step in the digital design flow that transforms an RTL description of a circuit into a gate-level netlist, which consists of logic gates and flip-flops mapped to a specific technology library. This process begins with elaboration, where the design hierarchy is flattened, and parameter values are resolved. The synthesis engine then performs a series of optimizations to convert behavioral and structural RTL constructs into a Boolean representation. Key steps include technology-independent optimizations such as logic minimization, redundancy removal, and retiming, followed by technology mapping, where the abstract logic is implemented using cells from the target standard cell library. Throughout synthesis, constraints on timing, area, and power are considered to guide optimization decisions. The resulting gate-level netlist is functionally equivalent to the RTL but is now suitable for downstream processes such as placement, routing, and physical verification.

Datapath synthesis plays a critical role within the broader logic synthesis process by optimizing arithmetic and word-level operations, such as adders, multipliers, shifters, and

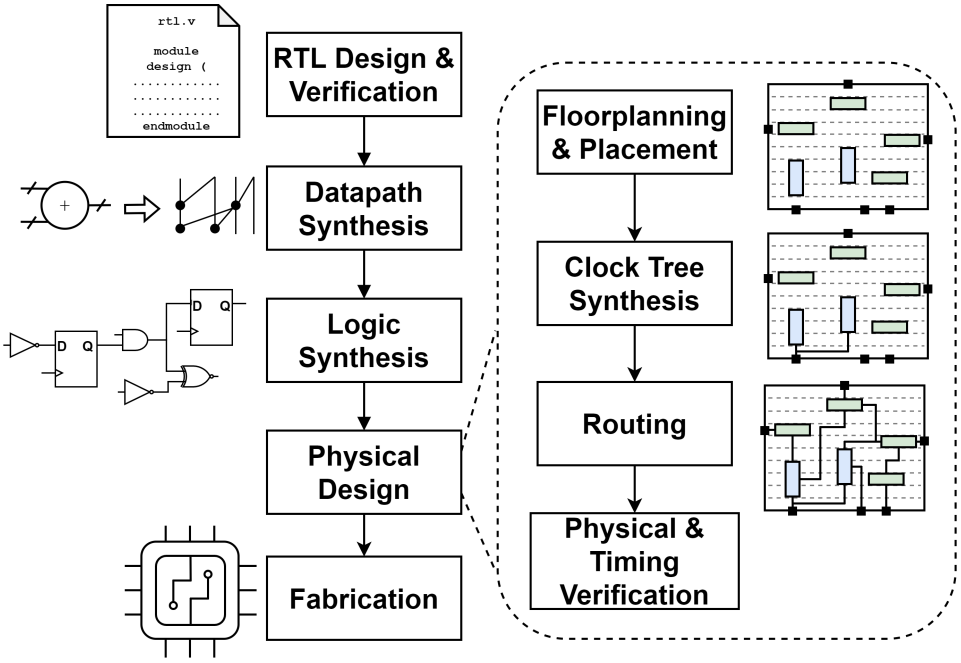


Figure 1.1: RTL to chip fabrication design flow focusing on physical implementation.

comparators, that are common in performance-critical designs like Digital Signal Processors (DSPs), CPUs, and machine learning accelerators. Unlike general logic synthesis, which operates primarily at the gate or Boolean level, datapath synthesis leverages the higher-level structure of arithmetic operations to generate more efficient hardware implementations. It performs technology-independent optimizations (e.g., operator strength reduction, bit-width minimization) as well as technology-mapping optimizations that select the most area- and performance-efficient implementation styles for target libraries. Effective datapath synthesis can significantly improve PPA by exploiting regularity, parallelism, and data dependencies in arithmetic logic, ultimately leading to faster and smaller designs.

Chip floorplanning and placement are foundational stages in the physical design flow that directly influence the overall performance, power, and area of an IC. The process begins with chip planning, where the design is partitioned into sub-circuits or blocks to enable parallel development and easier management of complexity. Each block undergoes floorplanning, which involves defining its physical dimensions, aspect ratio, and position within the chip layout. Floorplanning also establishes the placement of key structures such as I/O ports, power and ground rails, and macro blocks (e.g., memories or IP cores), while considering constraints like routing congestion, timing criticality, and power delivery. A well-conceived floorplan sets the stage for efficient downstream placement and routing by balancing interconnect length and block proximity.

Following floorplanning, the placement phase assigns precise locations to all standard cells and macro blocks defined in the gate-level netlist. This step occurs in two main stages: global placement and detailed placement. Global placement computes approximate cell positions to minimize wirelength, timing violations, and potential congestion, while allowing temporary overlaps to explore optimal configurations. In detailed placement (or legalization), these initial positions are adjusted to satisfy design rules and ensure non-overlapping, legal cell placements while preserving the quality of the global solution. Together, chip

planning and placement form the structural foundation of the physical design, significantly impacting routing complexity, timing closure, and overall QoR.

Clock Tree Synthesis (CTS) is a vital stage in the VLSI physical design flow that constructs the clock distribution network to deliver the clock signal from its sources to all sequential elements in the design with precise timing. The process involves inserting clock buffers and routing clock paths to control clock arrival times, aiming to minimize clock transition time and manage delay variations. A key aspect of CTS is the deliberate introduction of useful clock skew—intentionally adjusting clock arrival times at different registers—to improve timing margins, resolve setup and hold violations, and enhance overall timing closure. CTS also integrates clock gating to selectively disable the clock signal in inactive regions, significantly reducing dynamic power consumption. Through careful manipulation of clock delays and skew, CTS ensures reliable synchronization across the chip, enabling better performance and power efficiency in complex designs.

Routing is a fundamental phase in physical synthesis responsible for physically connecting the pins of each net across the chip to establish the required electrical paths. Similar to placement, routing is generally divided into two complementary steps: global routing and detailed routing. Global routing takes a coarse, high-level approach by planning the approximate routes that wires will follow through predefined routing regions or grids, focusing on balancing resource usage and avoiding congestion hotspots. Detailed routing then refines these plans by specifying the exact geometric paths, including precise metal layers, routing tracks, and vias for each wire, ensuring compliance with strict design rules. Throughout both phases, the routing process aims to minimize total wirelength and reduce resistance-capacitance (RC) parasitic effects that can degrade signal delay and integrity. Additionally, routing algorithms must carefully manage congestion to avoid overcrowding routing resources, while strictly enforcing design rule checks to prevent violations such as spacing, width, and layer constraints. Effective routing is essential for achieving timing closure, signal integrity, and manufacturability in complex integrated circuits.

Physical and Timing Verification is a critical final phase in the physical design flow that ensures the design meets all functional and manufacturability requirements before tape-out. After completing placement, routing, and optimization steps, the designer obtains highly accurate timing estimates that incorporate both cell delays and detailed wire parasitics. At this stage, the design must satisfy all timing constraints across multiple operating modes and PVT corners, guaranteeing the absence of setup and hold violations that could cause functional failures. If any timing violations are identified, targeted timing-driven optimizations are applied to recover timing and achieve a robust, violation-free design. Final timing signoff is performed using advanced Static Timing Analysis (STA) tools that rigorously verify timing closure under worst-case scenarios. In parallel, thorough physical verification is conducted to ensure the design is Design Rule Check (DRC) clean, confirming that all layout geometries comply with foundry-specific manufacturing rules. This verification step is essential to prevent fabrication issues, improve yield, and guarantee that the design is both functionally correct and manufacturable at scale.

1.2 Challenges in EDA

Modern physical synthesis EDA tools face increasing difficulty in coping with the rapid escalation in design complexity driven by aggressive technology scaling. At advanced nodes such as 7nm, 5nm, and beyond, the size and density of design blocks have grown signif-

icantly, often encompassing tens of millions of standard cells. With this scale comes a massive increase in the number of timing paths, placement options, and interconnect permutations, all of which must be optimized simultaneously. The search space becomes exponentially larger, making it harder to converge on globally optimal solutions within practical runtime and memory constraints. Traditional heuristic-based algorithms, while effective in smaller or more predictable designs, struggle to scale or generalize in the face of such high-dimensional solution spaces.

Another challenge is achieving consistent PPA optimization under multiple constraints. Modern designs are rarely optimized for a single corner or mode; instead, they must function reliably across a wide range of PVT conditions and operating scenarios. This multi-corner, multi-mode (MCMM) analysis introduces significant complexity into the synthesis flow, as optimizations that benefit one mode or corner may degrade performance in others. Coordinating timing, power gating, clock tree synthesis, and placement across these varying contexts requires advanced strategies for trade-off management, yet current tools often resort to conservative approaches that leave PPA improvements on the table.

The third challenge lies in the increasing interdependence of design stages, particularly between logical and physical aspects of synthesis. For example, decisions made during logic synthesis—such as gate sizing or technology mapping—can have profound impacts on placement feasibility and routability. Conversely, placement decisions can affect timing paths in ways that necessitate changes to the gate-level structure. Bridging this gap requires a tighter integration of traditionally separate design phases, but such integration is nontrivial due to the computational cost and complexity of maintaining accuracy across abstraction levels. Current tools often rely on iterative optimization loops between loosely coupled engines, which can be time-consuming and prone to suboptimal convergence.

Finally, variability and manufacturability concerns have grown more pronounced at advanced nodes, where physical effects like RC delay, electromigration, and process variation significantly impact circuit behavior. Physical synthesis tools must not only meet nominal timing and power constraints, but also account for these effects through robust modeling and optimization. This requires detailed awareness of parasitics, signal integrity, and layout-dependent effects early in the flow—capabilities that are traditionally deferred to post-synthesis stages. Incorporating these considerations into physical synthesis without overwhelming the runtime or scalability of the tool remains a persistent and pressing challenge.

Reinforcement learning offers a promising solution to the quality and scalability challenges that modern physical synthesis EDA tools face by enabling intelligent, adaptive optimization strategies that go beyond traditional heuristics. Unlike fixed-rule approaches, RL agents can learn from the environment through trial-and-error interactions, continuously refining their decision-making policies to optimize complex objectives such as power, performance, and area across diverse design spaces. This learning-based paradigm is particularly well-suited to navigating the high-dimensional, multi-objective, and constraint-driven nature of physical synthesis, where handcrafted rules often fail to generalize. RL can be trained to make context-aware decisions, such as cell sizing, buffering, or placement adjustments, by modeling the synthesis flow as a sequential decision process, thereby improving QoR while adapting to varying design constraints and technology nodes. Moreover, once trained, RL policies can scale to large and heterogeneous design blocks with minimal human intervention, offering the potential for faster convergence, better generalization across design scenarios, and more automated, efficient design pipelines.

1.3 Reinforcement learning

Reinforcement Learning (RL) refers to a class of problems, as well as the algorithms that address them, which revolve around the interaction between an agent and its environment, shown on Fig. 1.2. The main objective of RL is to teach an untrained agent the optimal strategy for achieving a specific goal. This is facilitated solely through interaction: the agent observes the current state of the environment and selects an action based on that observation. The environment responds to this action by transitioning to a new state and returning a reward reflecting the action's immediate outcome. The agent uses this feedback to evaluate the effectiveness of its selected action and to inform future choices.

This agent-environment interaction lies at the heart of RL algorithms and differentiates them from traditional approaches. In most heuristic algorithms, the designer explicitly defines the strategy used under different conditions. In supervised learning, a human expert provides labeled examples of the correct action for given states, with the goal of teaching the supervised agent to generalize this knowledge to new unseen states. In contrast, RL relies on the agent learning directly from its own experience by interacting with the environment. The designer does not need to instruct the agent on how to act in each case, and in fact, the strategy that the agent will learn to achieve its goal is not known in advance. This unguided interaction of the agent with the environment can lead to new efficient solutions that may have not been conceived by human experts, while also teaching the agent how to efficiently navigate the state-space, focusing on exploring solutions which seem promising.

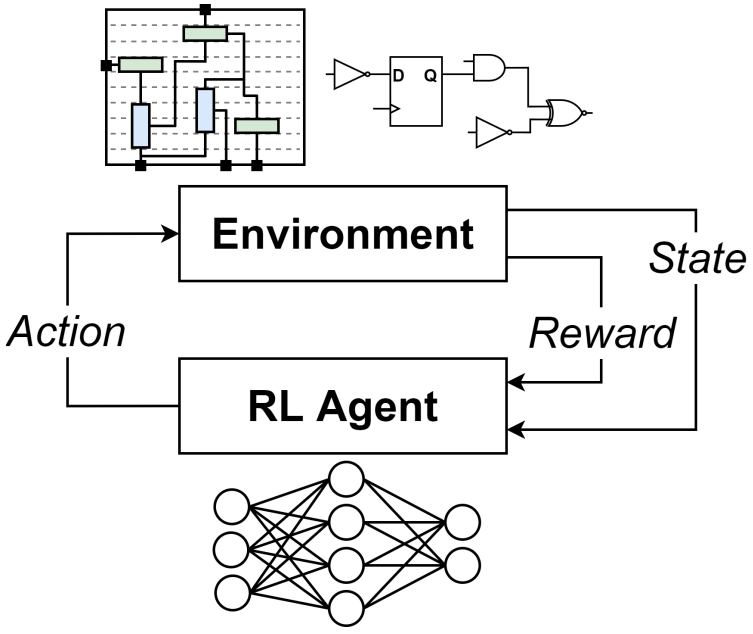


Figure 1.2: The agent-environment interaction at the heart of RL algorithms.

A fundamental challenge that any RL agent needs to address is the tradeoff between *exploration* and *exploitation*. In every state, the agent must decide between actions which have been previously explored and shown to yield positive outcomes, and actions that remain unexplored or less explored, but could potentially lead to even greater rewards. Finding the balance is critical: the agent must avoid spending excessive time exploring parts of the state space which have consistently yielded poor results, but also avoid over-exploiting familiar

actions to prevent getting trapped at local minima and missing better solutions. RL algorithms utilize a variety of strategies to manage this tradeoff, ranging from simple approaches, like occasionally selecting a random action instead of the best known one, to more sophisticated approaches that consider both the average reward an action has yielded and how frequently it has been tried out as an indicator to the actions potential.

Formally, RL problems can be modeled as a Markov Decision Process (MDP). An MDP is defined by a set of states (the state space), a set of actions (the action space), and a reward function specifying the immediate reward received for taking a particular action at a given state. This thesis focuses on deterministic environments, where taking an action at a certain state will always lead to the same next state, which is the most common case for EDA environments. Stochastic environments, where an action may lead to different outcomes based on a probability distribution, are out of the scope of this work. The core interaction of an RL framework is illustrated on Fig. 1.2. The environment presents the current state to the agent, which uses this observation to select an action. The environment applies this action, transitions to a new state and returns the reward of this action to the agent. The agent will use this reward to evaluate its decision and adjust its future behavior accordingly.

What makes RL approaches particularly powerful is their ability to assess the *value* of a state or a state-action pair (taking an action at a specific state) not only based on the immediate reward, but also by considering the sum of potential future rewards. In a sequence of actions taken by the agent, the value of the first action is not limited to its immediate reward, but includes the expected sum of rewards of all subsequent actions taken, discounted by a factor that prioritizes rewards closer to the present action. This enables the agent to favor long-term gains over short-term rewards, avoiding the pitfall of getting stuck in local minima.

When an RL agent is untrained it has no knowledge of which states or actions are valuable. As it explores the environment and receives feedback, it gradually improves its estimates of state and action values. This estimation process can be implemented either by maintaining a table of value estimates and updating it after receiving a reward, or through function approximation methods such as Neural Networks. In the latter case, the network receives the state as an input and can output the value of that state, the values of available actions, or the probabilities of choosing each action, depending on the method. This neural network-based approach has been widely used in recent years, leading to the rise of *Deep Reinforcement Learning* (DRL), a class of RL methods that have shown superhuman performance in multiple domains.

When formulating an RL problem, the designer must make a variety of choices which will greatly affect the effectiveness of the RL approach.

- **State space**. The state space defines all the possible states the environment can exist in at any time. A central challenge in RL problems is designing an efficient state representation. The state must capture all necessary information about the environment in an intuitive way depending on the problem at hand. For instance, on image processing applications the state can be represented by raw pixel values, whereas in a graph problem it can be encoded by the adjacency matrix or a learned graph embedding. In other applications the state can be a vector of hand crafted features. The state representation must also align with the Neural Network architecture, if one is used for value or policy estimation.

- **Action space**. The action space defines the set of all possible actions the agent can

take at any given state. In some problems the action space is straightforward, while in others it requires careful design. For instance, if the RL agent is trying to route a net on a 2D mesh, the action space might consist of connecting the net's head to the position directly north, south, west or east from its current location. In contrast, in a gate sizing task, the designer could select a minimal action space of 2 actions, upsizing or downsizing a gate by one drive strength, or opt for a larger action space where the agent can choose between multiple predetermined increments of drive strength to upsize or downsize by, making the decision process more challenging.

- **Reward**. After the agent takes an action, the environment provides the immediate reward corresponding to that action. The reward function must be defined in a way that effectively guides the agent towards the desired outcome. One common approach is to assign a zero or small negative reward for every step the agent takes, and a large positive reward once the goal is achieved. This encourages exploration and allows the agent to discover successful strategies independently. However, in problems with sparse rewards, the agent may struggle to ever reach the goal, especially in early training. An alternative is shaping the reward function by providing intermediate rewards based on partial progress or achieving subgoals. This can accelerate learning, but risks leading the agent towards easier-to-reach local optima and missing more optimal solutions. The best approach is application-specific and must balance exploration, learning speed and solution quality.

- **Policy**. The policy of an agent defines the probability of selecting each possible action at any given state. For some problems the policy may simply be choosing the action with the highest estimated value, whereas in other cases the policy may intentionally introduce randomness to promote exploration. It is also common practice to differentiate between the policy used for training, where exploration should be promoted, and the one used after the agent is trained, which focuses on exploiting the knowledge gained to achieve the best result.

- **RL training algorithms**. Finally, the designer must choose the training method for the RL agent. There exist a variety of RL algorithms, each suited to different types of problems, and their performance is strongly influenced by the choices made for the state and action space. Some algorithms, such as Q-learning [56] and Deep Q-learning [98], are value-based methods. These algorithms estimate the value of state-action pairs and select actions based on these estimates. Other algorithms, such as Proximal Policy Optimization [122] and Deep Deterministic Policy Gradient (DDPG) [78], are policy-based methods and focus on directly learning the optimal policy for selecting actions. In value-based methods, the policy is typically derived from value estimates, for example by choosing the action with the highest estimated value, whereas in policy-based methods, the policy itself is the function being directly optimized. There are also methods which combine these approaches and attempt to approximate both the value and the policy function simultaneously, called Actor-Critic methods [67, 97]. Different RL algorithms are used for discrete and continuous state and action spaces. Additionally, the neural network architecture, training loop and hyperparameters may vary depending on the chosen RL method.

Multi-Armed Bandits. Multi-Armed Bandits (MABs) are a subclass of RL problems. In MAB problems, the agent is presented with a set of actions (often referred to as "arms") and

must iteratively select one, observe the reward and use this information to identify which action yields the highest reward. Like in other RL problems, the exploit/explore dilemma is integral to MAB algorithms. However, there are some key differences in the way that MAB problems are formulated.

Unlike classic RL problems, MABs do not involve sequences of actions: the agent takes an action and receives an immediate reward, but the outcome of future actions does not influence the current action's value. As such, the agent's decision-making is based solely on the immediate reward from each action, rather than the cumulative effects of a series of actions.

Additionally, most MAB methods do not consider the state of the environment. However, there are variations of contextual or associative bandits where a state is used to influence action selection. The simplicity of MAB methods, with their lack of state encoding, neural network training and other complexities, make them well-suited for problems where quick and efficient agent training is essential, such as online recommendation systems and A/B testing.

1.4 Reinforecement learning in EDA state-of-the-art

In this section, we review the application of RL techniques to physical synthesis EDA tools and flows.

1.4.1 Datapath synthesis

Arithmetic components, such as adders and multipliers, can be implemented using a variety of architectures, each offering different trade-offs in terms of area, delay, and power. These range from well-established academic designs to novel architectures tailored for specific performance or efficiency goals. Additionally, approximate computing techniques can be employed to further reduce area and delay by relaxing accuracy requirements, enabling more efficient arithmetic units in error-tolerant applications.

The use of RL for synthesizing parallel prefix adders is explored in [116]. A Double Deep Q-Network (DQN)-based agent [46] iteratively adds or removes operators in the parallel prefix graph to optimize the adder architecture. The state is represented as a bitmap representing the MSB and LSB involved in each operator's computation. After logic synthesis, the reward is computed based on changes in delay and area. These objectives are treated independently and balanced using the scalarized DQN algorithm [99].

Reinforcement Learning has also been applied to multiplier optimization, shown in [156]. In this work, a DQN-based agent modifies the multiplier architecture by adding, removing, or replacing 2:2 and 3:2 compressors, with the goal of minimizing area and delay. The state is encoded as a tensor representing the compressor configuration, and the reward is computed as a weighted sum of the area and delay differences after synthesis. This work is extended in [157] to target Multiply-Accumulate (MAC) blocks, where parallel A2C-based agents are trained to accelerate exploration, which is further optimized by search space pruning heuristics. Another parallel approach is proposed in [33], where multiple DQN agents operate on different parts of the multiplier. These agents share a global Q-value, enabling coordinated optimization. The state is represented by a matrix encoding the compressor types, their levels, and conjunctive ordering, while the reward reflects changes in area and delay.

1.4.2 Logic synthesis

Optimizations during Logic Synthesis typically consist of two phases. The first phase contains technology independent optimizations, whereas in the second phase the delay and area information from the target technology library are used to accurately guide the optimization. Common optimizations goals during Logic Synthesis are the reduction of node count (or Lookup Tables for FPGA), as well as the minimization of logic levels.

A central motivation for using RL in Logic Synthesis is the observation that applying fixed optimization sequences across designs results in suboptimal behavior. Instead, dynamically adjusting which transformation to apply based on the design state and characteristics can potentially yield better outcomes in terms of area and delay. An early work utilizing RL in Logic Synthesis is presented in [43], where the authors train a Policy Gradient-based RL agent [134] to apply transformations on Majority Inverter Graphs (MIGs). The logic function to be synthesized is first mapped to an MIG using Shannon decomposition, and the agent then performs a sequence of transformations aimed at reducing either the size or the depth of the graph. The state is represented by the adjacency matrix of the MIG, scaled by the identity function and passed as input to a Deep Neural Network. The authors evaluate their method by comparing against the results of the CirKit logic synthesis package [127], demonstrating the methods efficiency.

The application of Deep Reinforcement Learning for Logic Synthesis is extended in [50], where the authors employ an A2C agent to select which transformation to apply within the ABC logic synthesis framework. The state representation consists of a collection of design metrics, such as the number of nodes and logic levels, while the action space consists of seven primitive transformations available in ABC. The authors use a multi-factor reward function, accounting for improvements in both area and delay. A similar approach is adopted in [147], where the authors target power minimization. In [153], the authors use a REINFORCE-trained RL agent to guide the transformation selection. Compared to previous publications, the state representation is greatly enhanced by including the action history, AIG statistics and a graph embedding generated by a Graph Convolutional Network (GCN) [63]. The reward reflects either the reduction of node count or logic depth, depending on the configuration. The authors of [148] further develop the state representation by using an Long Short-Term Memory (LSTM) network to encode the last 10 actions taken. Additional features include scalar metrics of the AIG, as well as structural features extracted by a Graph Isomorphic Network (GIN). The action space is a collection of ABC logic transformations, while the reward is a weighted sum of area and delay improvements. The agent is trained using Proximal Policy Optimization (PPO).

The previous works focus on QoR improvement but do not consider the time required to achieve it. Execution time consideration is first incorporated in [110], which uses a REINFORCE-trained RL agent for the selection of the applied optimization. The state includes scalar design features, the history of previous actions and a GCN-based graph embedding. Runtime is factored in the reward function, evaluating the benefit of each heuristic relatively to the runtime it consumed. In [111], the authors present two RL environments, one for logic optimization and one for FPGA Lookup table (LUT) mapping. Both share the same state representation, consisting of progress ratios relative to the initial state, action history, the proportion of actions taken compared to the allowed maximum, and the ratio of runtime compared to the that of a baseline script. The action space is different in the two RL environments, consisting of transformations for logic optimization and FPGA mapping respectively. The reward includes the ratios of the number of nodes (or number of LUTs for

FPGA mapping) and the number of logic levels of the current state compared to the initial state. The process terminates if the number of actions surpasses the maximum allowed, or if the runtime exceeds that of the baseline script.

In [22], the authors introduce automatic multi-parameter tuning for FPGA mapping. A softmax classifier selects the optimal mapping parameters, while the state consists of scalar features, graph features encoded by a GIN and the past actions history encoded by an LSTM. A PPO-based RL agent then selects the next optimization to apply. The reward is a scaled sum of improvement in area and logic depth count. In [152], the authors experiment with both PPO and A2C agents with the goal of minimizing LUT count in FPGAs. The state is a concatenation of circuit characteristics and the last 8 actions applied. The action space is inherited from [50], and the reward reflects the improvement of LUT count compared to the best solution found. A Random Forest classifier is used for feature importance analysis, discovering the most important features and allowing the removal of the ones which have minimal impact on the decision making of the agent. The trained agents are also evaluated on unseen designs, demonstrating generalization capabilities. In [10], an A2C RL agent is trained using a state which captures both pre-mapping design characteristics, such as the number of nodes and logic levels, as well as post-mapping, namely the maximum and average fan-ins and fan-outs. This is designed to prevent the agent from performing optimizations that appear beneficial in the pre-mapping stage but degrade the QoR post-mapping. The reward is also adapted, giving a higher significance to the delay improvement once the delay target is met, discouraging the re-introduction of negative slack.

Beyond traditional RL approaches, several works explore a variety of other methods to improve solution quality. The work of [88] is based on the observation that the logic synthesis problem can be modeled as a sequence generation problem. The authors collect a large dataset and use it to train a Decision Transformer in a paradigm of offline RL. The state representation is derived by processing the AIG graph through a GCN, and the reward reflects the reduction in node count. The generative pre-trained transformer (GPT) is then used to generate optimization sequences for unseen designs, achieving significant runtime improvements compared to previous works. A Multi-Armed Bandit approach is followed by [150] with various optimization goals, such as area and delay minimization for ASIC and FPGA, pre and post-Static Timing Analysis metrics, and Conjunction Normal Form (CNF) minimization. This method is integrated into multiple logic synthesis frameworks, including ABC, Yosys, VTR [100] and industrial tools. The work of [89] presents a method of accelerating the exploration and training of RL agents, which is orthogonal to the aforementioned publications. Multiple RL agents explore the state space in parallel, broadcasting their performance and weights to each other. The agents periodically copy the weights of the best performer, achieving faster convergence towards optimal solutions.

RL has also been applied in the context of Approximate Logic Synthesis (ALS), as shown in [108]. The primary objective of ALS is to map a given boolean network to library standard cells while ensuring that the primary output error does not surpass a given constraint. The introduced approximation can be leveraged for area and delay improvements. In this work, a tabular Q-learning agent is trained to select the Maximum Hamming Distance (MHD) of each node in the design, which serves as an estimate of the acceptable error. The library cell with the best area and delay satisfying the MHD constraint is then selected for each node. The corresponding area and delay improvements are then used for the reward computation.

1.4.3 Placement

During the placement stage, the EDA tool assigns positions to all standard cells and macro blocks within the chip. In global placement, optimization algorithms primarily focus on minimizing of the estimated wirelength and congestion, while also accounting for delay along timing-critical paths. During detailed placement, the focus shifts to legalizing the position of cells while limiting their displacement from their positions determined during global placement.

An early application of RL in the context of global placement is presented in [94]. A PPO-trained RL agent is responsible for placing the macro cells of the design, while standard cells are handled by a force-directed method. The state representation includes information about the node to be placed, technology and netlist-related characteristics, and graph features embedded via a GCN. The action space corresponds to assigning the macro block to any valid position on the (x, y) canvas, and the reward is the weighted sum of the estimated wirelength and congestion. The authors also employ a supervised learning model to predict the expected reward of each placement configuration, enabling transfer learning by using reward predictions across different designs.

In [41], Google Brain presents a brief overview of how placement problems in ASICs, FPGAs and TensorFlow Graphs can be formulated using RL, offering observations on how reward design, state representation and constraint handling influence the final QoR. Their work is further detailed in [95], where an RL agent is trained to place macro blocks, while standard cells are placed using a traditional force-directed method. The state is represented by the circuit graph, and the reward reflects improvements in design metrics. In [59], the method is extended by replacing the force directed method for standard cell placement with DREAMPlace [81], a state-of-the-art placer. The authors also justify the approximations made in [95], arguing that they improve runtime efficiency without degrading the final QoR. A critical assessment of this approach [95] is presented in [16]. The authors attempt to reproduce the method and benchmark setup of [95], noting challenges due to the limited availability of open-source material and data. Through a careful re-implementation of the method and evaluation scripts, and expansion of the benchmark collection, the authors report several limitations, including sensitivity to the initial placement and inferior performance compared to Simulated Annealing (SA) on academic and modern designs. Furthermore, human-designed placements outperform the RL agent of [95] on certain large, macro-heavy designs. Those observations are further supported by the analysis in [93], which raises concerns about the experimental setup and results, and suggests that correctly implemented SA methods can outperform the RL approach of [95] in many cases.

An similar approach to [59] is adopted in [18], where an RL agent is trained for placing macro blocks, while standard cells are placed using DREAMPlace. A key novelty of this work is the introduction of a joint state representation that captures both global and local perspectives. The global view is achieved by encoding the placement grid as a binary image, where each pixel denotes the presence (”1”) or absence (”0”) of a macro block in that position, while the local view is derived from a GCN applied to the design graph. To encourage routing-aware decisions, the authors also pass the placed netlist through a router and incorporate the routing wirelength and congestion into the reward function. In [11], the authors adopt a Multi-Objective RL framework, enabling the agent to learn placement strategies under varying weights for different optimization targets such as wirelength, congestion and timing. This allows the model to generalize across placement scenarios with different design metrics focus. The work of [69] treats placement as an image-based task by constructing

three pixel-level feature maps as the state representation. The position mask indicates valid placement locations, the wire mask reflects the half perimeter wirelength (HPWL) increase if the macro is placed at each location, and the view mask represents the current design state. The PPO-based agent places macro blocks sequentially, using the difference in HPWL to compute the reward. In [151], the authors extend the Actor-Critic network architecture by adding a third ”reconstruction” head that attempts to rebuild the placed netlist. The reconstruction error serves as an indicator of the training progress and helps guide the balance between exploration and exploitation during learning.

Another common strategy for incorporating RL into the placement process involves using a trained agent to performing the initial placement of the design, which is then used as the starting point of SA-based placement. This approach is used in [138], where an A2C agent is trained to generate an initial placement. The state representation is formed by concatenating the current sequence pair of the chip and the current block to be moved, while the action space consists of choosing another block to swap the current block with. The reward is divided in local and global components. The local reward reflects the wirelength difference after the chosen action has been applied, while the global reward is based on the final wirelength after a fixed number of steps and subsequent SA placement. The method proposed in [154] follows a similar strategy, where an RL agent is used to modify an existing placed netlist before passing it to SA. The RL framework is enhanced by the incorporation of a Random Distillation Network to better guide the exploration and the use of Spatial Pyramid Pooling to enable the generalization to netlists of different sizes. The agent is trained using a Multi-Scale Soft Actor-Critic algorithm.

A different direction is explored in [2], where RL is used to tune the placement parameters of a commercial EDA tool rather than directly determining the positions of cells and macros. A set of 12 placement parameters of *Cadence Innovus* is defined and an A2C-based RL agent is employed to modify these parameters with the goal of minimizing the HPWL. The state is composed of the current placement parameters, hand-crafted topological graph features, and learned graph embeddings generated via a Graph Neural Network (GNN) [44, 120]. After the agent modifies the placement parameters, the tool performs placement and the reward is computed based on the resulting HPWL compared to the one achieved by a human selection of placement parameters. In [64], a similar parameter-tuning strategy is applied to DREAMPlace, using an Asynchronous Advantage Actor Critic (A3C) agent. The state representation is a 3-dimensional matrix containing both global circuit and local cell features. The agent’s actions include adjusting the density parameter in the cost function, increasing or decreasing its significance, as well as modifying each cell’s tendency to move or remain fixed during placement.

Reinforcement Learning has also been explored for Detailed Placement. In [29], the authors train multiple RL agents to decide which placement optimization to apply at any given design state. This work focuses on Detailed Placement for FPGAs and is integrated in the GPlace3.0 [1] placement tool. The design, already processed through global placement, is split into regions. Each region’s state is represented by a concatenation of features, including its estimated local HPWL and LUT density. The agent selects from two actions provided by GPlace3.0, moving registers or LUTs, and from two optimization targets, either minimizing external Configurable Logic Block (CLB) pin count or wirelength. The reward is computed as the weighted sum of the new wirelength and external pin count. The authors evaluate three RL approaches: tabular Q-learning, deep Q-learning and an Advantage Actor-Critic, concluding that all three can match or outperform the baseline flow of the tool while also

reducing runtime. The work of [73] trains an A3C-based agent for determining the order in which cells are legalized. The state consists of a set of 13 features for each movable cell, while the action space is the assignment of legalization priority to each cell. The reward reflects the change in HPWL and displacement. Additional RL-related optimization techniques are applied for enhancing runtime and QoR.

1.4.4 Clock tree synthesis

Optimizations during the clock tree synthesis step mainly rely on adjusting the clock arrival time of registers, a process called clock skew scheduling or clock skewing. The introduced clock skew can improve timing closure, reduce peak current, and lower both the power consumption and area of the clock tree.

The use of RL in combination with Generative Adversarial Networks (GAN) [42] for efficient clock tree synthesis is explored in [86]. The proposed framework is divided into three stages. In the first stage, design features are extracted from placement images which represent register distribution, clock net distribution, and trial routing results. This is accomplished by using the pre-trained layers of a Convolutional Neural Network (CNN), with additional fully connected layers introduced by the authors to enable transfer learning. These extracted features are then passed to a regression model which estimates three key CTS metrics: clock power, clock wirelength and achieved maximum skew. The next stage involves a conditional GAN [96] network, a GAN variant with conditional inputs which allow control over the images generated by the GAN generator. The estimated placement values serve as conditional inputs, while the CNN-derived placement features are used as regular inputs to the generator. The generator, guided by an RL agent trained with the REINFORCE algorithm, outputs CTS modeling parameters aimed at optimizing the clock tree. Finally, in the third stage, the discriminator evaluates whether the generated data is realistic and whether these CTS parameters will lead to a clock tree which outperforms the baseline flow for at least two of the three target metrics (clock power, clock wirelength, and maximum skew).

In [7], a tabular Q-learning agent is trained to assign clock buffer counts to the clock pins of each register in a design, with the objective of minimizing peak current and, consequently, inductive IR drop. The registers are randomly distributed across a grid, and the agent navigates this grid, making decisions on whether to increase or decrease the number of clock buffers at each register. The reward function incorporates the standard deviation of clock arrival times, and applies significant penalties for introducing timing violations, along with high positive rewards for fixing such violations. A follow-up study in [6] proposes a modified Q-table update strategy, further enhancing the performance of the approach.

The work in [85] addresses the decision making process of whether timing violations at a failing endpoint should be resolved by clock skew scheduling or deferred to datapath optimization. To this end, a REINFORCE-based agent is trained to decide which failing endpoints should have their timing violations fixed through clock skewing and which should rely on datapath optimization. The state representation consists of a novel GNN encoding of the endpoints. Once the agent has made decisions for all endpoints, both clock skewing and datapath optimizations are applied, and the reward is computed based on the Total Negative Slack (TNS) recovered. The authors incorporate various methods to further optimize their approach such as an LSTM encoding of the action history and transfer learning for generalization to unseen designs.

1.4.5 Routing

During the routing stage, the EDA tool determines the exact paths that interconnect all placed standard cells and macros. Optimizations at this stage primarily aim to minimize wirelength and avoid congested regions, while also ensuring adherence to design rules to prevent Design Rule Violations (DRVs). Since long or heavily loaded wires contribute significantly to delay, routing decisions have a direct impact on both timing and power. Routing optimization must therefore balance timing closure, congestion mitigation, and DRC compliance to ensure a manufacturable and high-performance layout.

In [77], an early attempt of using RL for global routing is presented, where a DQN agent solves the two-pin routing problem iteratively for each net. The environment is a 2-layer 8×8 grid representing routing GCells, and the agent selects movement directions (north, south, east, west, up, down) to connect pins. The state encompasses the coordinates of the current position of the agent, the distance between the agent and the output pin it is trying to connect to, and the routing capacity of the 6 edges the agent is able to cross in the next step. The agent is rewarded for successfully reaching the target pin. The results of the RL agent are compared to the A^* algorithm, and it is observed that the performance is similar for low-congestion designs, while the RL agent outperforms A^* for more congestion-heavy cases due to its ability to consider global routing impact. The idea of transforming routing in a 3-D path planning problem is extended in [60], where multiple asynchronous RL agents are trained to simultaneously navigate a 3-D routing grid and route the design's nets. Each agent has a localized view of it's neighboring 3-D cube, as well as some information on the other agents provided by the state representation to avoid conflicting routing decisions. In [47], RL guided Monte Carlo Tree Search (MCTS) is leveraged for global routing. The state includes the routing grid, information on the net being routed, and the already routed nets. A new state in the tree can be reached by taking one step in the grid (for instance, for a 2-D grid this means going up, down, left or right) and a leaf state is reached if an illegal action is taken or if the target pin is reached. The reward involves a user defined loss related to design rules and metrics being optmized such as wirelength, but incurs a penalty if the new wire causes other nets to become unroutable, or greatly increases their path length, offering a global optimum view to the routing problem. Instead of performing random rollouts on the MCTS, the exploration is guided by a PPO-trained RL agent.

In [76], the authors utilize RL combined with Pattern Routing to solve the global routing problem while considering congestion. A Graph Attention Network (GAT) [139] is trained to take a number of GCell features as input and output the prediction of each GCell's congestion. This is used for the construction of each net segment's state representation, also including the net density and the capacity ratio value. An A3C-based RL agent receives this state representation and decides the appropriate action, which is to select an appropriate pattern and perform global routing. The reward is equal to the negative value of the failed nets, leading the agent to the minimization of routing congestion problems.

The ordering of elements to be routed can have a significant impact both on global and detailed routing. In [112] the authors train an A3C RL agent to decide in which order a design's nets should undergo detailed routing. The state is the concatenation of the design's net features, and the action space is the assignment of a priority to each net. After the RL agent assigns priorities to each net, a state-of-the-art router [74] executes detailed routing, and its results are used for the reward computation, which reflects the improvement on short violations, spacing violations, number of vias and wirelength. In [14], the authors employ a negotiation-based rip-up and re-route algorithm (NRR), with a PPO-based agent selecting

which nets should be ripped up. This work targets both traditional optimization goals, like minimizing DRC violations and wirelength, as well as custom electrical constraints, like path matching. The state is represented by a heterogeneous graph which includes routing segments, existing routing paths and path matching constraints, processed through a GNN. The reward reflects the improvement on routing metrics, as well as the similarity of each path in the path matching constraints. Using an RL agent to decide whether a net should be rerouted or not in global routing is the focus of [39]. The state is represented by a collection of scalar features. The reward is computed by rerouting the net, calculating the violation count difference, and assigning this number as the reward if the agent decided to reroute the net, or its opposite if the agent decided to not reroute. In this way the agent can be rewarded positively if it decides to not reroute a net that would cause extra violations, or negatively if it overlooks a promising net. The authors extend their approach in [40] by applying various enhancements, such as enriching the state representation and reward function.

An RL approach to Rectilinear Steiner Minimum Tree (RSMT) construction is presented on [84]. An A2C agent selects the placement of Steiner points in the design, targeting the connection of all pins of a net in the shortest way possible. The state representation is the sequence of (x, y) coordinates of all pins of the net, which are then passed through an encoder for enhanced feature extraction. The RL agent recursively selects pairs of pins, drawing a vertical line through the first and a horizontal line through the second and placing a Steiner point at their intersection. Constraints are imposed to ensure that the final placement of Steiner points results in a routable configuration. The reward is the negative length of the constructed RSMT, driving the agent to solutions with smaller trees. The application of RL for RSMT construction is extended in [15] by including obstacles that the tree has to avoid. A PPO-based agent selects the steiner points of the design, while a state-of-the-art algorithm [80] performs the actual routing. The state representation includes information on the routing grid, existing steiner points and obstacles, while the reward reflects the wirelength reduction after a steiner point is added. The state exploration is carried out using MCTS, while techniques such as Curriculum Learning [8] and learning from an opponent are used to further enhance the training convergence. In [102], Particle Swarm Optimization [61] is combined with RL for RSMT construction targeting wirelength minimization.

In [115], a PPO-based agent is used for fixing DRC violations on metal layer M1 of designs which have already undergone detailed routing by a genetic algorithm. The state representation consists of a 3-dimensional array, with the first dimension reflecting the M1 routes, the second showing the routing mask and the last one the DRC violations. The agent assigns the probability of additionally routing any of the M1 tracks, while the reward reflects the change in DRC violation count.

1.4.6 Gate sizing

Gate sizing selects the drive strength and threshold voltage for each standard cell to balance delay, power, and area. Increasing drive strength reduces delay and output transition time but raises input capacitance, potentially slowing upstream paths. Lowering threshold voltage improves speed without increasing input capacitance but increases power and area. Gate sizing addresses timing violations by upsizing cells on critical paths and reduces power/area by downsizing non-critical cells. It is *applied progressively throughout physical design flow*, from post-synthesis, where only intrinsic delays are known, through placement, CTS, and post-routing stages, as increasingly accurate timing and wire delay information becomes available, enabling more precise, effective optimizations to ensure timing closure

and resource efficiency.

In [140], the authors propose a transistor sizing framework that combines GCNs [63] and RL. The design is modeled as a graph, where each node represents a component (NMOS, PMOS, resistors, capacitors), and each edge reflects the connectivity. A GCN encodes global design information into embeddings for each component, which are then used by an DDPG agent to perform sizing, such as modifying transistor width and length, or resistance and capacitance values. These continuous values are refined to match the discreet legal technology options, and the reward is based on a weighted sum of design metrics. The authors also leverage transfer learning to generalize across different topologies and technologies.

Reinforcement Learning for standard cell gate sizing is proposed in [87], where an RL agent is used to minimize Total Negative Slack with minimal power overhead at the post-routing stage. The instances to be resized are selected based on their proximity to timing critical paths, and a GNN is used to compute their feature embeddings. The agent then assigns a new drive strength to each instance, rounded to the closest available value in the library. After all selected instances undergo resizing, timing analysis is performed and the resulting TNS is used for the reward computation. The RL agent is trained using the DDPG [78] method, a variant of the actor-critic algorithm.

Applying RL for ECO gate sizing at late stages of the flow is presented in [57]. In this work, RL is combined with Lagrangian Relaxation (LR) and applied in the post-routing stage, targeting timing violations. The state encompasses technology and graph-related features, extracted by a GCN and giving a global view of the design and encoding directional features (i.e. some features should be propagated to the inputs, outputs or both). The RL agent selects which cell to resize and whether to upsize or downsize it. After applying each action, incremental timing analysis is performed and the reward is computed based on the change of the LR-based cost function, which balances power and timing closure. The agent is trained using the DQN method. This approach is also transferable across designs and timing specifications after fine tuning.

1.5 Thesis contribution

This thesis advances the field of VLSI physical design by proposing an enhanced optimization framework for timing closure. The proposed approach incorporates a suite of optimization techniques, such as gate and flip-flop sizing, buffer insertion for fixing early/late timing violations, pin swapping, gate merging/splitting, and useful clock skew application, into each iteration of LR optimization [130]. These techniques are guided by LR-based cost functions to ensure locally optimal decisions at each step. By embedding a Multi-Armed Bandit (MAB) model into the flow [131], the optimizer adaptively selects the most beneficial heuristic at each iteration based on observed performance-reward trade-offs. This integration of learning-based decision-making [132] with classical optimization enables the autonomous execution of the flow, improving design outcomes in both timing and resource efficiency.

Empirical validation of the proposed LR-MAB framework demonstrates substantial QoR improvements over state-of-the-art methods. On the TAU2019 benchmark suite, the method achieves an average of 17% lower clock period, 15% reduction in power, and 6% area savings compared to the TAU2019 contest winner. Additionally, on the ISPD13 benchmark suite, it outperforms the best published results by achieving 25% better leakage power. These results underscore the significance of combining machine learning models with do-

main specific optimization in EDA workflows, and confirm the scalability and generalizability of the proposed approach across diverse benchmarks.

In the domain of approximate computing, this thesis also introduces a Reinforcement Learning-based framework for the synthesis of application-specific approximate parallel prefix adders [129]. Unlike conventional methods that focus solely on architectural templates, our framework allows an RL agent to learn approximate addition strategies by exploring the complete design space. It co-optimizes hardware efficiency and application-level quality through feedback from both hardware synthesis tools and application performance metrics. The resulting designs demonstrate, on average, 12% area savings and 10% power reduction, all while preserving acceptable application-level error characteristics. Furthermore, the framework can alternatively optimize to improve error behavior without incurring additional hardware cost, highlighting its versatility.

To enable this exploration, we develop a novel enumerative synthesis method that systematically generates all feasible approximate adder designs based on user-defined constraints [133]. Unlike prior works which explore fixed architectural families, our synthesizer uncovers a broader design space encompassing a wide variety of trade-offs in delay, area, and accuracy. This capability allows us to identify solutions that are either close to optimal or previously unattainable using standard parallel prefix architectures. Consequently, the proposed methodology represents a significant advancement in the automated design of approximate arithmetic units, enabling more efficient and tailored hardware for error-resilient applications.

1.6 Thesis organization

The rest of this thesis is organized as follows:

Chapter 2 presents a methodology for concurrently interleaving multiple heuristic optimizations to achieve timing closure in a multi-mode, multi-corner design environment. These optimization techniques, such as gate sizing, buffering, and pin swapping, are integrated within an LR-based optimization framework, which ensures convergence toward a feasible and improved timing solution.

Chapter 3 explores a novel stochastic approach to applying timing optimization heuristics, marking the first effort to decouple these optimizations from a fixed convergence flow. Instead of following a predetermined sequence, the optimization process is guided by a baseline Multi-Armed Bandit model, which dynamically selects heuristics based on their historical performance, offering a more flexible and adaptive optimization strategy.

Chapter 4 builds upon the methods discussed in Chapters 2 and 3, and presents the complete autonomous timing closure flow proposed in this thesis. The global optimization process continues to follow an LR-driven framework, but at each iteration, a fine-grained decision is made using a statistical recommendation system based on the MAB model. This hybrid approach enables automatic selection of the most promising heuristic at each step, allowing for fully autonomous optimization. The integration of LR-based cost modeling with machine learning-based control significantly improves quality-of-results, delivering better timing, power, and area metrics without manual intervention.

Chapter 5 shifts focus to the domain of approximate computing and introduces an RL framework for the synthesis of application-specific approximate parallel prefix adders. This framework co-optimizes hardware complexity and application-level performance by learning effective approximation strategies through exploration of the entire design space. The

resulting adders achieve substantial reductions in power and area while maintaining or even improving the quality of application-level outcomes.

Chapter 6 concludes the thesis by summarizing the key contributions and outlining potential directions for future research. These include extending the autonomous optimization framework to broader stages of physical design and exploring additional applications of machine learning in EDA and approximate computing.

2 Concurrent Logic-Restructuring Optimizations for Timing Closure

2.1 Introduction

Multimode multi-corner timing-driven design optimization aims at satisfying timing constraints in all modes and corners, while improving the area and power performance of the design. Fixing a violation in one timing scenario is likely to cause a new violation in another. This problematic behavior is accentuated when power also needs to be optimized, since the worst-case power corner could be different from the worst-case timing corner. Such a multi-objective problem is inherently complex and computationally challenging, when considering the highly increasing number of mode/corner combinations. Over the years, multiple optimization methods have been introduced to achieve significant Power-Performance-Area (PPA) improvement. Cell sizing, transistor voltage threshold selection (V_T-swap), netlist restructuring, timing-driven cell relocation, useful clock skew, and buffer insertion/deletion are just a few examples of optimization methods supported by modern physical design tools [71].

Though the independent application of such netlist transformations has offered improvements to the performance of designs, the transformations' combined application in the same optimization loop has not been explored. For the first time (to the best of our knowledge), we apply concurrently all relevant netlist transformations inside the same timing-driven optimization loop.

To do this, we extend the well-known Lagrangian Relaxation (LR)-based optimization formulation used for gate sizing [12], by interleaving in each iteration additional netlist transformations, including also register resizing. Each netlist transformation is smoothly integrated, without being disruptive to the overall optimization process. For instance, each resizing decision drives buffering additions in the same or following iterations, and each buffering addition guides the next sizing choices. In this way, buffers are added gradually and cell sizes adapt smoothly to it. The LR-based optimization orchestrates these heuristics with the goal of achieving global convergence.

In all cases, locally optimal decisions are taken using just the LR-based cost functions, without the need for incremental timing updates [38]. This increases the modularity of the proposed approach: any local netlist transformation that can be applied incrementally could be included in the set of available transformations.

To speed up the optimization, each logic-restructuring transformation is applied to a critical subset of the design's cells. In each LR iteration, the cells are selected based on sensitivity criteria that identify the cells that are more promising to improve timing or reduce area and power.

This approach has been successfully applied to the TAU 2019 multi-corner design optimization contest benchmarks [17]. In all cases, our approach successfully optimizes the designs in all given corners and achieves (a) lower clock period, or (b) reduced area and

leakage power at the same clock period as the contest's winner, without violating any slew or maximum capacitance constraints.

The contributions of this work are summarized as follows:

- In each local iteration, we integrate LR-based buffering, useful skew, and logic restructuring, without the need for incremental timing updates per iteration.
- The LR-based formulation is enhanced to handle the sizing of both registers and gates using the same cost function.
- Timing conflicts, where late and early timing violations lead to contradictory sizing decisions using LR-based cost function, are identified and handled appropriately.
- To reduce runtime, sizing is applied only to a subset of cells selected by sensitivity metrics. This is the first time that sensitivity based selection is combined with LR gate sizing.

2.2 Typical timing optimization approaches

Design optimization includes a multitude of different techniques. A standard industry approach is to iteratively select the targets on timing-critical paths and try different optimizations on each cell, picking the best alternative, before optimizing the next target. This can be quite runtime expensive, and such greedy approaches can be sub-optimal from a global perspective. Typical optimization techniques include gate sizing, buffering, useful skew, remapping and swapping connections to logically equivalent pins of a gate.

Early work on gate sizing proposed convex optimization with a continuous range of transistor sizes [36]. Subsequent work suggested use of more accurate polynomial models to provide a convex formulation to ensure optimality [62]. However, convex delay models are too inaccurate for modern technologies and do not adequately model standard-cell libraries with discrete cell sizes. Discrete gate sizing has historically been solved with greedy gate-sizing approaches, such as proposed by Coudert [21], focusing on timing-critical portions of the circuit and the best delay vs. power sensitivity trade-off, or vice versa, when trying to reduce power in portions of the circuit with timing slack. Greedy gate-sizing, which is still commonly used in commercial Electronic Design Automation (EDA) tools, has been shown to be suboptimal in comparison to global optimization [19].

Several global optimization approaches for discrete gate sizing have been proposed, resizing gates to the discrete version that achieves the closest slack-slew values to those optimally calculated. Nguyen *et al.* [103] used inaccurate timing models for standard-cell delays, ignoring slew dependence and wire loads, and used Linear Programming (LP) to assign timing slack to resize gates for maximizing power savings subject to timing constraints. Chinnery *et al.* [19] improved this by accurately accounting for the delay changes on the neighboring gates due to a resize, but the runtime of this timing-accurate LP approach was prohibitive on larger designs, due to roughly quadratic runtime growth with design size. Held *et al.* [48] assigned slew targets, instead of delay targets. Fatemi *et al.* [32] presented sensitivities that can be used for timing, power, area, and slew optimization across multiple corners. The core algorithm in these papers can be sped up by relaxing the timing constraints with LR.

LR has been widely used for design optimization in recent years. It was first used by Chen *et al.* [12] for continuous wire and gate sizing. Hu *et al.* [51] used LR to define optimal continuous gate sizes that are then clustered to discrete sizes based on proximity to the optimal

size. Ozdal *et al.* [106] formulated the LR sub-problem to trade off leakage power and fix timing violations, choosing gate sizes with Dynamic Programming (DP). Flach *et al.* [38] sized each gate to locally minimize the LR cost. This provided great leakage power savings, with faster runtime than other approaches, achieving the best results to date on the ISPD 2012 and 2013 benchmarks. Sharma *et al.* [125] extended this work with multi-threading and a new Lagrangian Multiplier update method to reduce the number of iterations to converge, achieving a 15× speedup at the cost of 2.5% higher leakage power compared to [38]. We use essentially the same LR gate-sizing approach as in [38] and [125], extending it for register resizing and for handling sizing when facing conflicting late/early timing violations.

Roy *et al.* [118] extended the LR formulation to handle multiple modes and corners. Daboul *et al.* [24] used a resource sharing formulation, which is a specialization of LR. Shklover *et al.* [126] added clock skew to the LR formulation and resized both gates and clock buffers. Sharma *et al.* [123] combined LR gate sizing and slack-based clock skew assignment, while, more recently, Mangiras *et al.* [92] extended the LR formulation for timing-driven placement to include flip-flops, gates, and local clock buffers, leading to efficient placements.

Delay buffer insertion is a standard technique to fix early timing violations by increasing the path delay. Huang *et al.* [53] used an LP formulation to minimize the number of added hold buffers. Tu *et al.* [136] tackled hold violations across different power modes in ultra-low voltage designs. Wu *et al.* [145] presented an linear-programming approach to model setup and hold constraints and assign delays that should be inserted on each node to solve hold violations. They then used dynamic programming to perform buffer insertion. Han *et al.* [45] proposed a hold-buffer insertion method that achieves hold timing closure across multiple corners.

Buffers can also be used to increase the drive strength and speed up timing on nets with high fanout and significant load capacitance. Van Ginneken [137] found the optimal buffer positions along the route of a net given a set of arrival timing constraints. Lillis *et al.* [79] extended this to account for multiple buffer types; while Wang *et al.* [141] presented a lower complexity buffering algorithm. Jiang *et al.* [58] formulated simultaneous transistor sizing and buffer insertion used for late critical path isolation. Liu *et al.* [83] used an LR-based cost function for buffer insertion on each net using only one buffer size from the library, while Ho *et al.* [49] allowed for multiple buffer options. Finally, Hu *et al.* [52] proposed an approximation method that reduces runtime with minimum impact on the result.

2.3 Lagrangian-relaxation based design optimization

Formulating a design optimization problem can consider timing, power, and area objectives, or constraints in various combinations. In this work, we aim at minimizing the sum of power, area, and Total Negative Slack (TNS) of the design (TNS is the sum of timing violations at the timing endpoints), such that the timing constraints are met.

$$\begin{aligned}
&\textbf{min}: \sum_{c\in\text{cells}} P(c) + A(c) - \sum_{j\in\text{POs}} slk_j^L - \sum_{j\in\text{POs}} slk_j^E \\
&\textbf{s.t.:}\ slk_j^L \le 0 \text{ and } slk_j^E \le 0, \forall j \in \text{POs} \\
&\quad slk_j^L \le r_j^L - a_j^L \text{ and } slk_j^E \le a_j^E - r_j^E, \forall j \in \text{POs} \\
&\quad a_i^L + d_{i\to j}^L \le a_j^L \text{ and } a_i^E + d_{i\to j}^E \ge a_j^E, \forall \text{arcs } i \to j
\end{aligned} \tag{2.1}$$

$P(c)$, $A(c)$ are the leakage power and area of cell c; slk_j is endpoint j's negative slack; a_j and r_j are pin j's arrival and required times; and $d_{i\to j}$ is the arc delay from pin i to pin j. The arc delays include both cell input pin to output pin delays (cell arc delays), and the wire delay from a cell output pin to the input pin of another cell. The indices E and L represent early and late timing, respectively, and timing endpoint primary outputs (POs) include the input-D pins of all flip flops and the primary outputs of the design.

Lagrangian Relaxation is applied on the formulated problem to incorporate the constraints into the cost function and simplify the cost function [12]. Each constraint is multiplied by λ weights called Lagrangian Multipliers (LMs), as shown in (2.2). The higher the LM value, the more critical the respective constraint is.

$$\begin{aligned}
&\textbf{min}: \sum_{c\in\text{cells}} P(c) + A(c) - \sum_{j\in\text{POs}} slk_j^L - \sum_{j\in\text{POs}} slk_j^E + \\
&\sum_{j\in\text{POs}} \left(\lambda_{j0}^L slk_j^L + \lambda_{j0}^E slk_j^E\right) + \\
&\sum_{j\in\text{POs}} \left(\lambda_{j1}^L (slk_j^L - r_j^L + a_j^L) + \lambda_{j1}^E (slk_j^E - a_j^E + r_j^E)\right) + \\
&\sum_{i\to j\in\text{arcs}} \lambda_{i\to j}^L (a_i^L + d_{i\to j}^L - a_j^L) + \lambda_{i\to j}^E (a_j^E - a_i^E - d_{i\to j}^E)
\end{aligned} \tag{2.2}$$

By differentiating (2.2) with respect to the arrival times, according to the Karush-Kuhn-Tucker optimality conditions, we end up with the following LM flow-conservation rules [12]:

$$\sum_{i\in\text{fanin}(j)} \lambda_{i\to j}^L = \sum_{k\in\text{fanout}(j)} \lambda_{j\to k}^L \text{ and } \sum_{i\in\text{fanin}(j)} \lambda_{i\to j}^E = \sum_{k\in\text{fanout}(j)} \lambda_{j\to k}^E \tag{2.3}$$

The LMs for the output pin of a cell are proportionally distributed to the LMs of the cell's input-output arcs. For example for gate G in Fig. 2.1, net 6's outgoing LM is propagated to the LMs into net 6, preserving the equalities: $\lambda_{4\to6}^L + \lambda_{5\to6}^L = \lambda_{6\to7}^L$, $\lambda_{4\to6}^E + \lambda_{5\to6}^E = \lambda_{6\to7}^E$. For flip-flops (FFs), following [92], the LMs added at their output Q pin are distributed to the clock-to-Q timing arc. For the example shown in Fig. 2.1, this translates to $\lambda_{13\to9}^L = \lambda_{9\to11}^L + \lambda_{9\to12}^L$, $\lambda_{13\to9}^E = \lambda_{9\to11}^E + \lambda_{9\to12}^E$.

Substituting (2.3) into (2.2), we simplify the objective to (2.4).

$$\min \sum_{gates} P(c) + A(c) + \sum_{i\to j} \lambda_{i\to j}^L d_{i\to j}^L - \lambda_{i\to j}^E d_{i\to j}^E \tag{2.4}$$

From (2.4), it can be observed that each LM highlights the criticality of its associated timing arc. A large late LM means that the delay of the arc should be decreased to minimize the cost function, while a large early LM means that the delay of the arc should be increased. On the other hand, a low value for an LM means that the delay of the arc is not critical to the optimization.

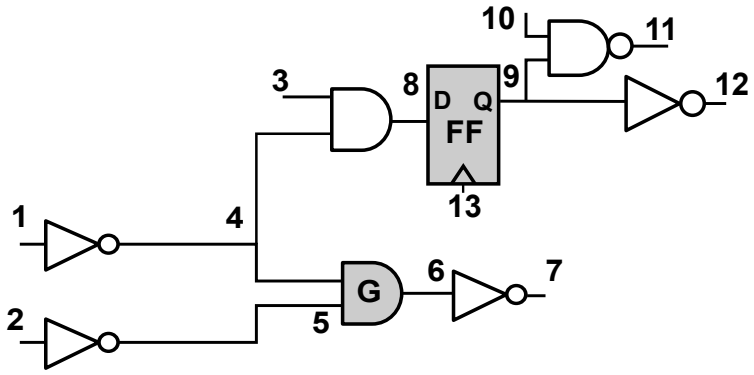


Figure 2.1: Example of a small circuit, used to demonstrate LM propagation and local cost calculation

2.3.1 Overall optimization flow

The proposed optimization flow is presented in Figure 2.2. Initially, all cells are sized to the smallest size that does not violate any maximum load or slew constraints [75]. The LMs for every arc are initialized to 1, and the iterative optimization process begins.

At the start of each iteration, an incremental timing update for all corners is performed. Then, we use timing information for only two corners: the most critical early and late corner. The critical late corner is that with the highest total negative slack (TNS), or the corner with the lowest late total slack, if no corner has late violations. The same applies for the most critical early corner. Where a timing variable is used for a mode (early, late), it is implied that its value is calculated from the respective critical corner. The critical corners are redetermined after every timing update.

Each LR optimization step begins by updating the LMs according to the process detailed in Section 2.3.2. Then, the selected netlist transformations are executed one after the other. Clock-skew assignment inserts or removes a fixed delay from the clock pins of the registers, in order to help LR cost minimization. Register and gate resizing choose an appropriate version for the most critical cells in the circuit. Pin swapping attempts to swap the sink pin of the most timing critical net of the gate with another equivalent pin, in order to help the timing critical net with its violations. Hold-buffer insertion finds the positions and number of buffers that need to be inserted to reduce early violations. Late buffer insertion chooses if a buffer should be inserted on the sink of cells with high fanout-loads-to-input capacitance ratios, and the size of the buffer if insertion proves beneficial. All of the above methods are driven by the LM values and try to optimize the LR cost using slack information only to ensure that they are not degrading the timing violations. So, accurate slack information is not necessary, which allows for the whole series of transformations within an iteration to take place without the need for any new incremental timing update.

Firstly, we apply the transformations that affect the timing startpoints and endpoints of the design, which are clock-skew optimization and register resizing. Then, gate sizing and pin swapping, that traverse the netlist in topological order, are applied. This also allows them to propagate the updated timing information from the startpoints to the endpoints with local timing updates. The transformations that are applied on intermediate topological levels of the circuit are executed last. This order ensures that the local timing updates will propagate timing information rather accurately, thus removing the necessity of an incremental timing

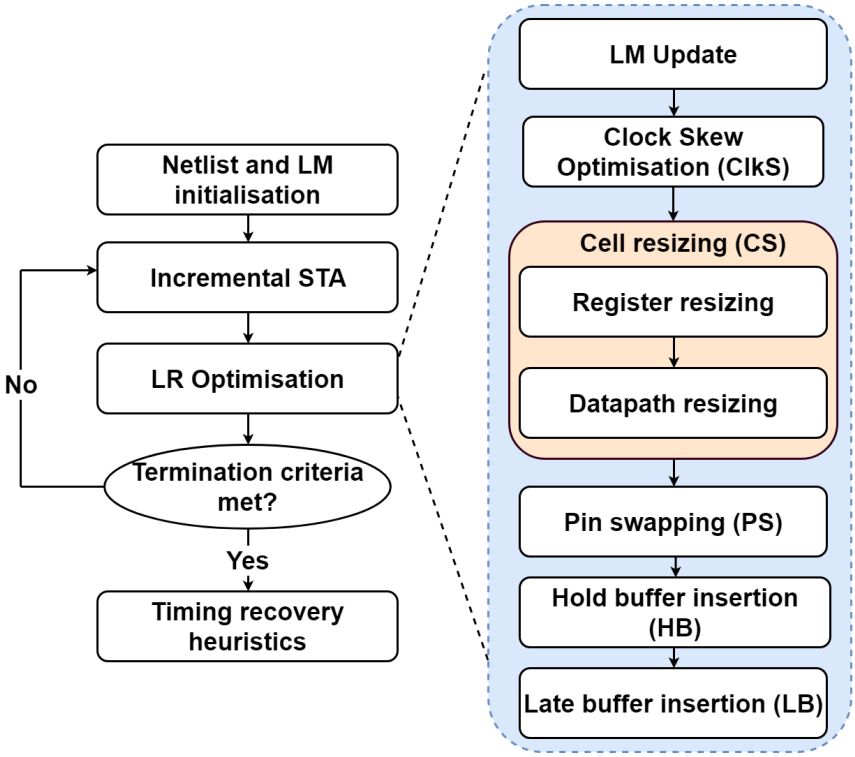


Figure 2.2: The proposed optimization flow.

update after the application of each transformation.

Optimization stops when the maximum number of iterations is reached, or if the solution quality doesn't improve for 2 consecutive iterations. If timing constraints are satisfied, the quality of the solution is equal to the area/power improvement, whereas, if timing violations are present, termination is judged by TNS improvement. In the end, a final brute-force timing recovery step is executed.

2.3.2 LM updates

LMs are updated using the modified sub-gradient method proposed in [135]:

$$\lambda^L_{j0} = \lambda^L_{j0}\left(\frac{a^L_j}{r^L_j}\right), \lambda^E_{j0} = \lambda^E_{j0}\left(\frac{r^E_j}{a^E_j}\right) \forall i \in \text{POs}$$

$$\lambda^L_{i\to j} = \lambda^L_{i\to j}\left(\frac{a^L_i + d^L_{i\to j}}{a^L_j}\right), \lambda^E_{i\to j} = \lambda^E_{i\to j}\left(\frac{a^E_j}{a^E_i + d^E_{i\to j}}\right) \forall \text{arc}_{i\to j}$$

The delays, the arrival times, and the required arrival times for early-late timing modes are calculated from the corresponding critical corner. Gate sizing methods like [38] and [125] utilized exponential LM updates for faster convergence. Even if this is the proper choice for gate-sizing only transformations, it does not fit well in our flow that utilizes many different netlist transformations. In this case, the LMs should adapt smoothly to the changes in the design's timing, allowing for better co-operation across the various optimization heuristics.

The updated values of output LMs should be distributed to all nets satisfying the flow conservation conditions (2.3). The distribution is performed by traversing the circuit in reverse

Algorithm 1: Cell resizing algorithm

```
1  foreach cell c ∈ candidate_cells in topological order do
2      best_cost ← localCost(c) ;
3      best_version ← cellVersion(c) ;
4      neg_slack ← localNegativeSlack(c) ;
5      foreach version v ∈ equivalent_versions do
6          resize cell c to v;
7          if (violates_load_constraints(c)) then
8              skip version;
9          end
10         update timing locally;
11         new_neg_slack ← localNegativeSlack(c) ;
12         if new_neg_slack < γ · neg_slack then
13             skip version;
14         end
15         new_cost ← localCost(c);
16         if new_cost < best_cost then
17             best_cost ← new_cost ;
18             best_version ← new_version ;
19         end
20     end
21     resize cell c to best_version ;
22     update timing locally ;
23 end
```

topological order. At each visited cell, the sum of LMs at the output pins is distributed to the LMs of the input pins. When an LM value needs to be distributed to multiple incoming arcs, this distribution is done based on the ratio of the LMs of the corresponding timing arcs. Such distribution increases the LMs on critical paths, and, therefore, the timing violations are expected to be minimized. Also, since LMs are accumulated at each branching point, the higher the number of violating endpoints affected by an arc, the higher the value of the corresponding LM.

2.4 Netlist transformation optimizations

2.4.1 Cell resizing

The cell resizing algorithm (Algorithm 1) examines different versions for each cell and selects the one that minimizes the local cost function. Every cell is visited in topological order. If a cell is a candidate for resizing (resizing candidate choice is analyzed in subsection 2.4.1), then every size of the cell is explored. After trying all sizes, the cell is resized to the version that minimizes the local cost function without introducing load violations and without degrading the slack of its nets over a threshold γ [38]. Similar to [38] and [125], the cost function (2.5) used for the selection is a localized version of the global cost function (2.4) that includes only local timing arcs:

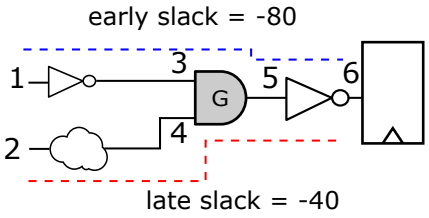
$$LC(v) = P(v) + A(v) + \sum_{i \to j \in local\_arcs} \lambda^L_{i \to j} d^L_{i \to j} - \lambda^E_{i \to j} d^E_{i \to j} \tag{2.5}$$

with $P(v)$ and $A(v)$ being the leakage power and area, respectively, of size v of the examined cell.

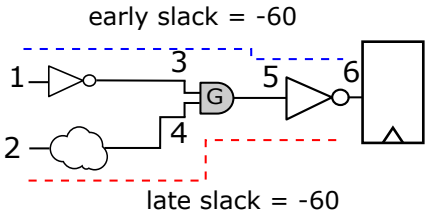
The local timing arcs for each cell include the arcs of the resized cell, its immediate fanin and fanouts, and the arcs that share a common fanin with the resized cell. For gate G in Figure 2.1, the local arcs include: arcs of G $(4 \to 6, 5 \to 6)$, the fanin arcs $(1 \to 4, 2 \to 5)$, the fanout arcs $(6 \to 7)$, and the common fanin arcs $(4 \to 8)$. Flip-flops are handled similarly to gates. For flip-flop FF in Figure 2.1, the local arcs include arcs $(13 \to 9)$, $(3 \to 8, 4 \to 8)$ and $(9 \to 11, 9 \to 12)$.

Handling of conflicting constraints

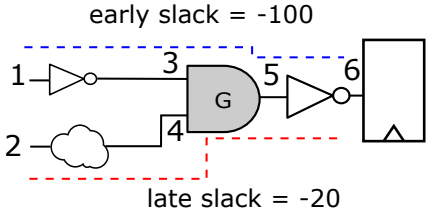
Relying on the local cost function (2.5) may lead to contradicting sizing decisions when a timing arc has early and late violations. Consider the case depicted in Figure 2.3a. In this case, gate G has an early violating path passing through pin 3, and a late violating path



(a) Example of conflicting violations.



(b) Result of normal resizing.



(c) Promoting late timing for conflicting arcs.

Figure 2.3: Example of resizing gate G with conflicting violations. It is seen that late slack is minimized in case (c) and, if early slack is handled by buffer insertion, case (c) will end up with the least amount of negative slack.

through pin 4, ending up with conflicting violations on pin 5. No matter which decision the sizer takes, there is no choice that will reduce both timing violations. More specifically, the LR algorithm would try to reduce the violations on the side with the higher LM. For example, if the early LMs are higher than the late ones, it would try to slow down the arc. This is shown in Figure 2.3b, where the sizer downsized gate G to increase its delay. However, this choice worsens late slack. So, eventually, the algorithm balances the slacks on both (early-late) sides, without truly solving any violations.

In order to tackle this, we decided to focus on one side of the violations only when an arc has conflicting violations. Since early violations can always be fixed by other means, such as hold-buffer insertion, we decided to focus on late violations only. This can be done by omitting the terms $\lambda^E_{i\to j} d^E_{i\to j}$ from the cost function (2.5).

So, in case of conflicts, the LR optimization focuses on speeding up arcs with conflicting violations. Based on this strategy, in our example shown in Fig. 2.3c, the sizer would upsize gate G to reduce late violations, expecting that a hold-buffer insertion algorithm would later insert buffers on the fast path.

Filtering candidate cells

Attempting to resize every cell in the circuit can be very computationally expensive, especially for large circuits. Identifying which cells are critical for improving circuit performance is an important step for reducing the runtime without compromising the final quality of results.

Therefore, before resizing any cell, we make a list of the cells that participate on the top k late and early critical paths, of the cells placed on the fanouts of late critical paths, and cells with significant positive slack and large power and area savings potential. From this list of cells, we need to find the ones that are the most sensitive to resizing. In other words, their resizing greatly affects the cost function. This can be done by calculating a sensitivity for each cell of the list and examine the resizing of only the N cells with the largest sensitivities. Depending on the type of each cell, we employ a different sensitivity factor:

- *Cells on the late critical path*: Cells on the late critical path will usually be upsized to speed up the path, while trading off power and area. So, the cells that will be resized should be the cells that have the best cost-power-area tradeoff. To calculate the sensitivity for those cells, we compare the local cost difference, as well as the power and area difference between the current and the next bigger size.

- *Cells on the early critical path*: Cells on the early critical path will try to slow down the path, usually by downsizing. Since this does not create a trade-off (downsizing on an early critical path will save power, area, and slack), we do not involve power and area in the sensitivity function. For this set of cells, only early slack minimization matters. In this case, we compare the value of the local cost function (2.5) between the current size and the next smaller one, after neglecting the power and area of this choice.

- *Cells on the fanout of late critical paths*: Those cells should be downsized to reduce the output load of the cells on the critical path, and thus area and power are certainly reduced. Therefore, we compare the value of the local cost function (2.5) between the current size and the next smaller one, after neglecting the power and area of each choice.

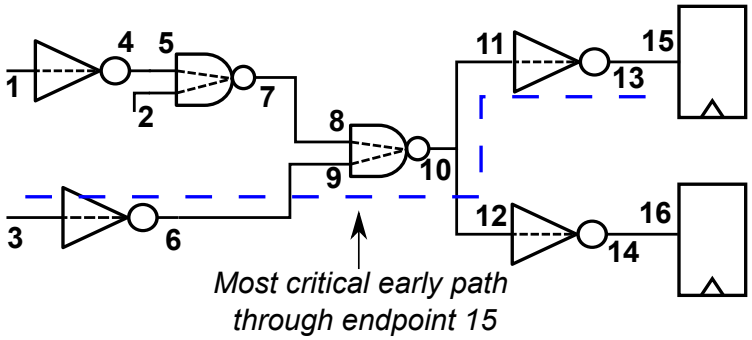


Figure 2.4: Hold buffer insertion example. Pin 10, which is part of the early critical path, has the highest $\lambda^E - \lambda^L$ difference and receives extra buffers for alleviating hold-time violations.

- *Cells with positive slack*: When a cell has both positive early and late slacks, it will be downsized to save leakage power and area, providing this does not create new slack violations. In this step, we want cells that have both a large margin for leakage and area minimization, and enough positive slack to allow a downsize. To achieve this, we calculate the power-area difference of the current size and the next smaller size for cells that have enough positive slack.

Please note that, even though we assume that a cell should be upsized or downsized for its sensitivity calculation, during true resizing all possible sizes for the selected cells are tried with the full local cost function, including area and leakage.

2.4.2 Buffer insertion

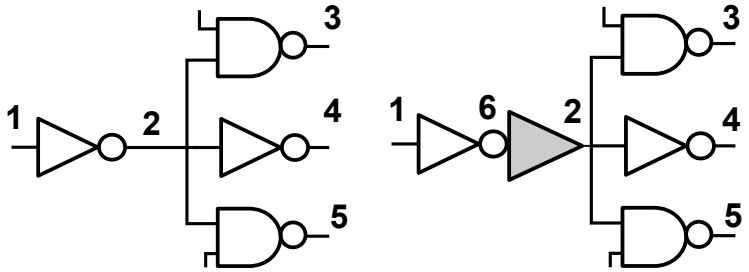
Buffering is a versatile design optimization that can be efficiently applied for various design targets. In this work, we employ two forms of buffering that target early and late timing violations.

Early timing violations

Solving early (hold) timing violations requires slowing down the signal propagation on violating paths. This can be achieved by appropriate delay buffer insertion. Hold buffers should be inserted in positions where they will affect many hold violating paths, thus minimizing the number of buffers that need to be added. We avoid adding buffers on paths with both late (setup) and early violations. Hold buffers are only inserted in pins where there is room to trade off positive late slack with negative early slack.

For every hold violating endpoint, we store the worst early path through it and keep in a list of candidates the path's pin with the highest difference $\lambda^E - \lambda^L$ (i.e., critical in early timing and not critical in late timing) and with positive late slack. Then, on those candidate pins, we add delay by using a chain of identical inverters, gradually consuming the available positive late slack.

Let us assume that, in Fig. 2.4, the candidate pin for the worst early path through endpoint 15 could be pin 10. To add a chain of buffers at pin 10, without the need to resize that driving cell, we want the cell driving pin 10 to see a similar output load. If adding a buffer at the output reduces the load, it would speed up the driver, contrary to our purpose of adding delay



(a) Attempting buffering on pin 2 (b) After buffer insertion

Figure 2.5: Example of late buffer insertion on pin 2. The local cost (2.5) without a buffer is computed over the arcs for pin 2: $1 \rightarrow 2, 2 \rightarrow 3, 2 \rightarrow 4, 2 \rightarrow 5$. The local cost with a buffer inserted is the cost of the arcs around the buffer: $1 \rightarrow 6, 6 \rightarrow 2, 2 \rightarrow 3, 2 \rightarrow 4, 2 \rightarrow 5$, including the buffer's power and area. If the new cost is lower than the original, the buffer is kept provided that no maximum slew/load violation is introduced.

on the path with the early timing violation. In Fig. 2.4, the appropriate buffer for adding delay on pin 10 should have an input capacitance close to $C_{\text{in}}^{11} + C_{\text{in}}^{12}$, plus any effective wire capacitance of the net connecting pins 10, 11, and 12.

Since all the buffers of the chain would see the same load (similar to their input capacitance), we can assume that they will all have the same delay d^E in early mode timing. Therefore, for a negative early slack of slk^E, we need at least $N_{\text{min}} = -slk^E/d^E$ buffers to remove early slack violations.

At the same time, assuming that the available positive late slack is slk^L, we cannot add more than $N_{\text{max}} = slk^L/d^L$ buffers, where d^L is the delay of the same buffer in late mode timing. Since we decided not to consume more than 50% of the available positive late slack, we limit the maximum number of buffers allowed to $N_{max}/2$. Overall, the number of buffers added is equal to $\min(N_{\text{min}}, N_{\text{max}}/2)$ provided that buffer addition does not introduce any maximum slew/load violation.

Late timing violations

Buffering can reduce the output load of cells with a large fanout, thus decreasing their delay and helping reduce late slack violations.

Initially, the gates with negative late slack are sorted by their output capacitance over input capacitance ratio. Then, buffer insertion is attempted on the source of the fanout net of the top 100 gates. All available buffer choices are examined and the one with the lowest cost is kept. If the lowest cost with the buffer added is smaller than the cost without the buffer and buffer insertion does not introduce slew/load violations, the selected buffer is actually inserted. The local cost around the buffer is calculated using equation (2.5) involving all arcs connected to the net where the buffer will be inserted. An example of the involved timing arcs is shown in Fig. 2.5.

2.4.3 Cell merging and splitting

While gate sizing and buffering transformations are very effective in improving timing and reducing the area and power of the design, merge-split transformations can broaden the

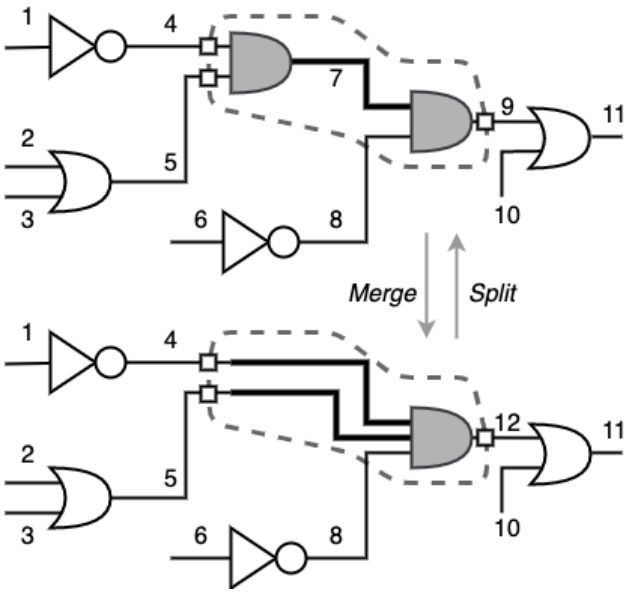


Figure 2.6: Merge/Split example highlighting the placement of the replaced gate and the timing arcs used for local cost calculations.

solution space by trading off the number of logic levels with the simplicity of the gates per level. This transformation explores simple merge/split options across independent AND, OR, NAND and NOR gates, while complex cases of merging/splitting combined AND-OR Boolean functions are not supported. Both transformations are applied on every cell eligible for merging or splitting in topological order. As noticed on the flow of Fig. 2.2, cell merging and splitting are not included in the optimization process, but are used in the future work presented in Chapter 4.

Cell merging

Cell merging replaces two serially connected logic gates with a multi-input gate of the same functionality. For example, in Fig. 2.6, two-input AND gates G1 and G2 are replaced by a three-input AND gate G3. The new gate is always assumed to replace the end gate, e.g., G2 in Fig. 2.6, and spatially placed in its position. The wire resistance and capacitance (RC) parasitics of the net connecting the two gates (net 7 in Fig. 2.6) are added to the input net parasitics of the fanin gate, i.e., nets 4 and 5 in Fig. 2.6. To judge if this replacement is beneficial, we compute the local LR cost (2.5) before and after the merge. If the LR cost after the merge is lower, the local slack is not degraded, and there are no load violations introduced, the merge is accepted.

For computing the local cost before the merge in the example shown in Fig. 2.6, we use the timing arcs of the two gates that will be merged, the arcs of their fanins and their immediate fanouts: $\{1 \rightarrow 4, 2 \rightarrow 5, 3 \rightarrow 5, 4 \rightarrow 7, 5 \rightarrow 7, 6 \rightarrow 8, 7 \rightarrow 9, 8 \rightarrow 9, 9 \rightarrow 11\}$.

After the merge, some timing arcs inevitably vanish. Therefore, for the example shown in Fig. 2.6, the local cost *after the merge* is calculated by using arcs $4 \rightarrow 12, 5 \rightarrow 12$, and $8 \rightarrow 12$, in the place of arcs $4 \rightarrow 7, 5 \rightarrow 7, 7 \rightarrow 9$, and $8 \rightarrow 9$. The LM used for these new timing arcs is derived after redistributing locally the LM values that were already present on the fanout of the cell to be merged. To compute the local cost for the merged gate, we need

to know its exact size. Instead of trying various sizes (this is a task of cell sizing), the new merged cell is assigned the size of the first gate, or the size that exhibits the closest input capacitance to that gate.

Cell splitting

The cell splitting transformation separates a multi-input gate to two gates with fewer input pins. The gate splitting algorithm is applied on every multi-input AND/OR/NAND/NOR cell that can be split in cells with a smaller number of inputs.

To limit the possible splitting options, in this work, a cell is always split to a cell with two inputs, and a second cell for the rest of the inputs. The sizes of the resulting gates will be the same as the size of the original gate, or the one that matches more closely its input capacitance. The two new cells are connected serially (connected with a net of minimum length), with the multi-input one driving the two-input cell. The most critical late net is connected to the two-input cell and the most critical early net is connected to the multi-input cell. If the same net is the most critical in both early and late timing, it will be assigned to the position satisfying the most critical mode, thus improving the timing of the local most critical path. A split actually occurs only when the local cost after the split (calculated similarly to cell merging) is less than the cost before the split, without introducing maximum load/slew violations.

2.4.4 Pin swapping

When logic functionality allows, we examine connecting the most timing-critical input to the fastest input-to-output path of the gate. The pin swapping algorithm visits every combinational gate with negative slack, or on the immediate fanout of a cell with negative slack in topological order. For each gate, the input pin with the most negative early or late slack is swapped with the other equivalent pins. After each swap, the timing is updated locally and the local cost function (2.5) of the gate is recalculated. After every swap is done, the swap with the best local cost which doesn't violate slew/load constraints is kept and the next gate is processed.

2.4.5 Useful clock skew assignment

To maximize the overall optimization efficiency, we combine the previously mentioned datapath-driven optimizations with clock skew optimizations, which allow for wider-range timing tuning. Instead of setting up a new linear program to determine optimal clock skew offsets [35], we decide locally and incrementally on the clock arrival time of each flip-flop, thus allowing useful clock skew to adjust smoothly to the optimizations of the other netlist transformations.

Based on the formulation presented in [92], LMs can be propagated to the clock pins of the flip-flops as follows: $\lambda^L_{CLK} = \lambda^L_Q + \lambda^E_D$, $\lambda^E_{CLK} = \lambda^E_Q + \lambda^L_D$. Thus, the sign of $\lambda^L_{CLK} - \lambda^E_{CLK}$ determines if we should delay or speed up clock arrival. If it is positive, a constant delay d_{clk} is subtracted from clock arrival, favoring slk^L_Q and slk^E_D; instead, if it is negative, the same delay is added to clock arrival in favor of slk^L_D and slk^E_Q. In both cases, the update of the useful clock skew is kept as long as it does not degrade the late/early worst slack on all data pins of the flip-flop. d_{clk} equals the average delay of all available clock buffers driving a minimum sized flip-flop of the library.

Since we operate in a multi-corner environment, the delay added should be scaled across corners. Since the clock arrival offsets would be implemented using the available clock buffers of the library, we compute a scaling factor for each corner by approximating the ratio of the clock buffer delay of that corner compared to the buffer delay of the typical corner, for multiple input slew and output load values. So, when a delay d_{clk} is added on a clock pin for the typical corner, a delay $d_{clk} \cdot ratio_{cr}$ will be added for corner cr, where $ratio_{cr}$ is the average buffer delay ratio for corner cr and the typical corner.

2.4.6 Timing recovery heuristics

After the LR-based optimization is complete, some recovery steps are executed as a way to perform minor enhancements to the Quality-of-Results (QoR) and eradicate any small leftover timing violations.

The timing recovery steps resizes the driver cell affecting the most violating endpoints [38]. The sizes tested are the immediately smaller and bigger size. After either move, we perform local timing update on the critical corner and calculate the new local negative slack on the cell's output. If it is improved compared to the initial slack, we perform an incremental timing update and ensure that the TNS has improved too. If it has, this cell version is kept and the process is repeated. If the timing has not improved we revert the change and move on to the next most critical cell. Timing recovery stops if all timing violations are solved, if the TNS stops improving, or if a certain number of incremental timing updates is reached. This recovery step is performed twice: once for the remaining late timing violations, and once for the early timing violations.

For the last remaining early timing violations, a buffer chain is inserted in front of every hold violating endpoint, until the violations are fixed. This is the last optimization step performed, ensuring that the final design is hold-violation free.

2.5 Experimental results

The proposed method was implemented in C++ inside the open-source RSyn framework [37] after extending it for multi-corner timing analysis. The results of this work were validated using the TAU 2019 contest benchmarks and compared against the winner of the contest [17]. We extracted the presented results by running the executable the TAU 2019 winner sent to us, which was the one submitted to the contest. The 6 benchmarks of the contest include SPEF and SDC files for each netlist, and 5 different standard-cell library files, one for each corner. The designs only have timing constraints on register-to-register paths, leaving primary inputs and outputs unconstrained. All benchmarks also include clock trees that are mostly unrealistic, since they do not have RC parasitics, and the clock arrival times are highly unbalanced. For this reason, the TAU 2019 contest winner removed the clock buffers from the clock tree and rebuilt it without including any RC parasitics of the nets connecting the clock-tree buffers. Note that this feature of assuming ideal wires on the clock tree was, indeed, permitted by the contest.

Therefore, in order to allow for a more realistic comparison (assuming that designs are compared in a pre-CTS stage), we removed all clock buffers from both the initial set of benchmarks fed to our algorithm, and from the final netlist of the TAU 2019 winner. Nevertheless, for the latter, we kept the assigned clock pin arrival times (acting as useful clock skew values), so that the timing performance of the TAU 2019 winner remained unaffected.

Even though we removed the unrealistic – due to the lack of RC parasitics – clock trees, we retained their useful effect on timing.

Table 2.1: The leakage and area under the best period reported by the contest for typical corner.

Design	#Cells	Period (ps)	Leakage (uW)		Area (um^2)	
			Ours	Winner	Ours	Winner
s1196	584	334	12	13	545	565
systemcdes	2825	696	68	82	3190	3565
usb_funct	10535	828	352	402	17058	17831
vga_lcd	87958	684	2826	3125	143168	146802
leon2_iccad	793286	1483	24925	30249	1232760	1311985
leon3mp_iccad	649191	1661	19589	23806	961941	1028117

In Table 2.1, we show the results of running both flows for only the typical corner, and for the clock period specified by the contest as the one where the contest winner achieves timing closure for the typical corner. Our flow also achieves timing closure for these clock periods, saving 17% more leakage power and 6% more area.

Secondly, in Table 2.2, we present the best clock period for which each flow achieves closure for all corners. On average, our flow achieves 14% better clock period, while, simultaneously, saving 15% in leakage and 5% in area. More specifically, our flow results in a better clock period for all designs except one, with the benefits reaching up to 35%. The exception is s1196, where our clock period is 5% worse. In every other design, our flow achieves a better clock period with minimal, or no power and area overhead.

Furthermore, Table 2.3 depicts the results after running both flows on the same clock period for which both achieve timing closure, i.e., the clock period of the slower. Our flow achieves on average 16% lower leakage power and 6% lower area, and results in smaller leakage and area for every benchmark.

These improvements are the result of the smooth cooperation between the transformation optimizations. Sizing identifies the most critical cells and resizes them to the size that locally reduces the LR score. Resizing targets the 2000 highest sensitivity cells for the 1000 most critical paths. Early slack is allowed to degrade when resizing cells with conflicting violations, in favor of late slack optimization, and it is then reduced by cell sizing on early violating paths, hold-buffer insertion, and clock-tree optimization. Thus, hold violations stay under control, while late violations are constantly reduced. This is highlighted in Fig.

Table 2.2: The best clock period achieved for all corners

Design	Period (ps)		Leakage (uW)		Area (um^2)	
	Ours	Winner	Ours	Winner	Ours	Winner
s1196	1040	918	12.1	12.6	550	569
systemcdes	1777	1788	87	96	3665	3975
usb_funct	2166	2306	419	402	18579	17812
vga_lcd	1871	2826	3215	3106	149033	146257
leon2_iccad	4532	4677	25170	30354	1237510	1312960
leon3mp_iccad	3878	5246	20045	23816	971773	1030860

Table 2.3: Comparison of the leakage and area at the clock period where both flows achieve closure for all corners. The number of iterations for the LR flow, as well as the runtime for both flows are also presented.

Design	Prd	Leakage (uW)		Area (um^2)	
	(ps)	Ours	Winner	Ours	Winner
s1196	1040	12	13	550	569
systemcdes	1788	85	96	3603	3975
usb_funct	2306	397	402	18002	17812
vga_lcd	2826	3075	3106	145752	146257
leon2_iccad	4677	24996	30354	1234180	1312960
leon3mp_iccad	5246	19632	23816	962764	1030860

Design	LR	Runtime (s)						
	iters	CS	PS	LB	HB	ClkS	Total	Winner
s1196	4	1	0.02	0.01	0	0.01	2	2
systemcdes	8	13	0.92	0.10	0.12	0.03	17	21
usb_funct	10	43	1.14	0.21	0.88	0.41	50	52
vga_lcd	8	312	9.02	2.92	5.47	21.81	455	24
leon2_iccad	10	2341	23.58	10.48	19.91	1291.11	4471	452
leon3mp_iccad	9	2046	13.12	8.11	22.04	812.95	3862	362

2.7 for vga_lcd. The late TNS starts off as very negative and slowly converges to 0 as iterations progress. Early TNS is always 0 at first for all designs, since removing the clock tree solves all hold violations. Early violations appear during optimization as paths become faster and clock skew assignment trades off early for late slack. Appropriate cell sizing and hold-buffer insertion keep early slack under control, usually reducing it back to 0 at the end of the iterative optimization.

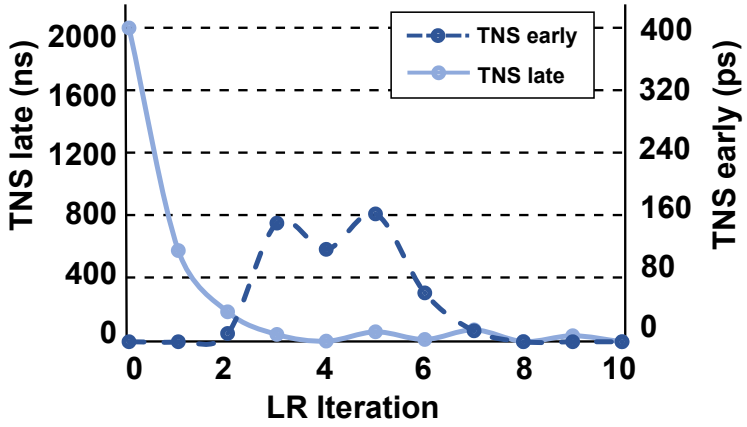


Figure 2.7: Early and late TNS optimization for vga_lcd.

The runtime of the compared methods are also presented in Table 2.3. They were run on a 3.6GHz Intel Core i7-4790 Linux workstation with four cores and 32GB of RAM. The iterations that the proposed LR-based optimization needed per benchmark are shown on the second column of the bottom half of Table 2.3. In the smaller benchmarks, the runtime of

the two methods is almost the same. However for the larger benchmarks, the winner of the contest is significantly faster. We cannot tell if the latter experiences a runtime-vs-quality-of-results tradeoff, i.e., if the results could improve with more runtime. Our future goal is to adopt a multi-threaded approach for each transformation method to reduce runtime. The runtime contribution of each method is detailed in columns 8 to 12 of Table 2.3. The lion's share belongs to cell sizing and clock skew assignment. Early and late buffer insertion heuristics are applied on a small number of pins, so they are fast.

To quantify the effectiveness of each transformation, we reran the flow excluding one of the available transformations in each run. As an example, Table 2.4 presents the results for the indicative design vga_lcd. It shows the TNS, WNS, power, and area for the best clock period achieved by our flow (the clock period reported in Table 2.2) for the initial netlist and the optimized netlist. Since hold timing constraints are satisfied in every case, TNS and WNS refer to late violations only.

It is observed that every method is required in the flow in order to achieve the best clock period reported. Cell sizing is required for leakage power and area minimization, since the leakage and area results for the run without cell sizing is significantly larger. Clock skew assignment has the most significant contribution to timing violation minimization, and the rest of the methods further improve the final results without a significant runtime overhead.

Regarding hold buffer insertion, it is observed that early timing achieves closure even when it is not applied in the optimization loop, due to the early timing recovery that runs in the end of the optimization. Nevertheless, omitting it results in a worse late timing, since the iterative LR based method picks better positions for hold buffer insertion.

Table 2.4: Methods' contributions to the results for vga_lcd.

Netlist		TNS (ns)	WNS (ns)	Leakage (uW)	Area (um^2)
Initial		-19731.00	-12.60	4841	192551
Initial downsizing		-18945.00	-11.80	3078	145692
Full flow		0	0	3215	149033
Opt removed	CS	-50.26	-0.34	3599	161287
	PS	-0.02	-0.01	3205	148836
	LB	-0.11	-0.11	3128	147369
	HB	-2.63	-0.17	2927	145558
	ClkS	-1598.54	-0.44	3174	147832

2.6 Conclusions

This work introduces for the first time, the concurrent application of several timing optimization heuristics within the same LR-based optimization loop. The part that unifies the application of all logic-restructuring transformation is the fact that they operate on similar local cost functions based on the LM weights. The results show that this approach is a scalable alternative for smoothly integrating multiple transformations that, in the past, have been applied independently to the design. Most importantly, this allows for a modular customization of the optimization flow, whereby transformations are added or removed, based on their effectiveness for a design, without altering the global optimization framework.

3 Multi-Armed Bandit Recommendation of Timing Optimization Heuristics

3.1 Introduction

In Chapter 2, we explored the importance of interleaving various optimization algorithms in the same loop, which are applied sequentially in a fixed order. However, even if each algorithm is effective in optimizing the design, the order of application of the various algorithms is equally important to the final result.

In modern industrial tools, the order of applying the available design optimization methods is organized in reference flows based on human experience and on how well the flow has performed so far on other designs. If closure is not achieved by the reference flow, the flow is customized for the specific needs of the current design. Deciding on a new configuration faces the exploitation versus exploration dilemma: exploit the same heuristics that have so far yielded high Quality-of-Results (QoR), or explore new heuristics with more uncertain QoR to gather more information and hope to reach a better overall solution in the end. Even if this dilemma can be answered optimally for a certain design and a fixed set of optimization heuristics, the answer cannot be directly transferred to a new design, nor can it be adapted to a different set of heuristics.

In this work, we leverage the Multi-Arm Bandit (MAB) model – a primitive form of online Reinforcement Learning – in an ASIC design flow to autonomously apply design optimization heuristics and achieve timing closure and power/area reductions. In the past, MAB has been successfully used in online recommendations systems [70, 144] and for tuning generic and FPGA compilation flows [3, 146]. General adaptive optimization flows were also explored in the context of compilers [142], logic synthesis [50], and dynamic clock and voltage scaling for run-time power optimization [119].

The proposed approach tries to answer the following fundamental question regarding design optimization: **given a set of well-defined design optimization heuristics with uncertain QoR, and a set of timing constraints, how should one choose the available heuristics online, and without any previous knowledge of the design, to maximize the final QoR over multiple trials while keeping runtime under control?** According to MAB, for each one of the applied optimization algorithms, a "reward" is recorded. The MAB policy [4] decides (recommends) which optimization method should be selected next, following a balanced exploitation-exploration approach. The desire to choose a method that has paid off well in the past – thereby offering increased Power-Performance-Area (PPA) metrics – is balanced with the desire to try different methods that may produce even better results.

The proposed approach has been successfully applied to the ISPD12 and TAU2019 benchmarks in two recent international design optimization contests [13, 104]. In all cases, the MAB-based optimization successfully optimizes the designs and achieves equal, or even better, results than the most efficient methods available in the open literature for the same benchmarks. **The most important outcome of this work is that MAB-based design optimization not only achieves timing closure and competitive power/area reductions, but it**

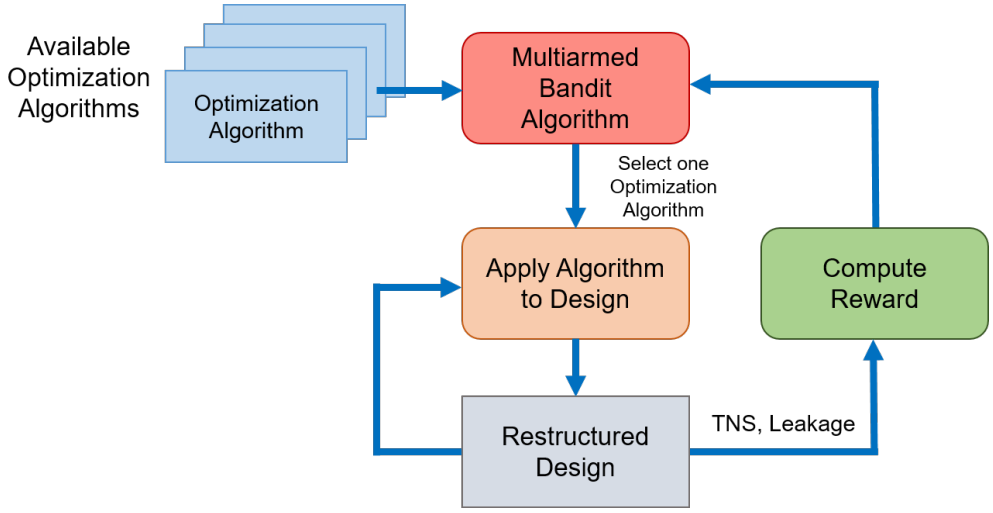


Figure 3.1: Autonomous timing-driven design optimization. The design is repeatedly optimized by the algorithm selected by the MAB methodology. The reward of each algorithm and the MAB policy determine which algorithm will be applied next to the restructured and partially optimized design.

also does that autonomously without relying on any previous knowledge of the circuits or the optimization methods used, and without requiring any human intervention.

3.2 Autonomous online design optimization

The proposed methodology automates the application of optimization algorithms of varying complexities to the design under consideration with the goal to autonomously achieve timing closure and improve PPA. The proposed autonomous optimization framework assumes the role of a gambler sitting in front of slot machines (the set of optimization heuristics), who has to decide which machines to play, how many times to play each machine, and in which order, to maximize their gain (QoR biased by runtime overhead).

Multi-armed bandits (MABs) orchestrate the execution of a set of optimization algorithms in order to maximize the QoR, driven by the reward function defined in Section 3.3. MAB policy exploits the most effective optimization algorithms already tried, while still exploring new options by activating less frequently used algorithms in the hope of achieving a greater payoff [70, 144].

The proposed optimization flow is shown in Fig. 3.1. In each round, one optimization algorithm is selected and applied to the design. When the algorithm finishes, its reward is recorded, in order to guide the selection of the following rounds. The changes performed by the algorithm on the design under consideration are kept for the next rounds, unless the obtained result is worse in all aspects (timing, leakage power, and area). As long as the design is altered in each round, successive plays of the same optimization algorithm would yield different rewards. In fact, the rewards are diminishing as timing closure is approached and power/area are already well optimized. The optimization stops when the reward observed in 10 consecutive iterations does not improve by more than 1%.

An algorithm not selected in a specific round is considered “frozen” and neither receives a reward, nor it changes the average reward observed so far.

The MAB methodology takes the average reward observed so far and tries to pick the next optimization heuristic, according to the chosen exploit-vs-explore policy. If you exploit too much, you might miss out on other, really effective heuristics. On the other hand, if you explore too much, you will waste rounds without taking advantage of effective algorithms. Given a history of rewards and a number of "pulls" for each optimization algorithm, the Upper Confidence Bound (UCB) MAB algorithm [4] (the crux of the MAB methodology) calculates an optimistic guess as to how good the expected payoff of each algorithm is, and picks the sequence with the highest estimate as follows:

1. Apply each one of the optimization algorithms once to initialize the mean payoff $\overline{Q}_0(a)$ of each algorithm a.

2. For each next trial t, select the algorithm a to apply in this trial that maximizes $\overline{Q}_{t-1}(a) + \sqrt{\frac{2 \log t}{N_a}}$, where t is the total number of trials and N_a is the number of times that algorithm a has been played so far.

3. Observe the reward $Q_t(a)$ of this trial using the reward function described in Section 3.3, and update the mean reward for this action $\overline{Q}_t(a)$.

The mean reward $\overline{Q}(a)$ for algorithm a can simply be the running average of the rewards gained in the previous applications of the same optimization algorithm $\overline{Q}_t(a) = \overline{Q}_{t-1}(a)\frac{N_a-1}{N_a} + Q_t(a)\frac{1}{N_a}$. Even if this choice seems reasonable, the newly received $Q_t(a)$ reward is scaled by $1/N_a$, which diminishes its effect as N_a gets large. This is problematic, since, as the optimization evolves, the design is gradually moved to a better state and an algorithm tends to be less effective even if, in the first trials, it received high rewards. For example, a method that aims towards improving the Total Negative Slack (TNS) will receive high rewards as the timing improves, but it will receive zero credit once the timing violations are fixed. Therefore, the MAB algorithm will keep picking this algorithm, since its average reward will still be high, and it will require many trials to shift the average reward to a lower value. In order to avoid this effect, we can over-weigh recent rewards over past rewards in order to better highlight the most recent performance of each algorithm [144]. This process is called drifting in MAB-based policies and updates the average reward for each algorithm as follows:

$$\overline{Q}_t(a) = k\,\overline{Q}_{t-1}(a) + (1-k)Q_t(a)$$

The experimental results presented in Section 3.4 assume k=0.5.
The algorithm stops when a maximum number of iterations is reached (we set it to 200), or when the reward received is less than 0.001 for 10 consecutive iterations.

Regarding the optimization algorithms available to the MAB framework, in this work we focus on the fine-grained interleaving of buffering and gate-sizing transformations. We define a set of seven design heuristics, with each one focusing on a different aspect of the design, seen on Table 3.1 and analyzed in detail in Section 3.4. The MAB policy orchestrates the application of the available heuristics with the goal of achieving global convergence.

In order to allow the smooth integration of the various design optimization heuristics and to facilitate the learning process of the MAB algorithm, each design heuristic operates either for a reduced number of iterations, or on a small set of critical paths. In this way, the netlist restructuring achieved by each heuristic is not disruptive to the overall optimization process. If each algorithm run to completion then switching to a new heuristic would incur significant

Table 3.1: The runtime scaling factor used for tuning the reward according to the runtime spent by each heuristic to achieve it.

Optimization method	r_a
LB–Late buffering	1.00
CPI–Critical Path Isolation	1.00
EB–Early buffering	1.00
CB–Clock buffering	1.00
LR1C–LR 1 Iteration for critical cells	0.85
LR1–LR 1 Iteration for all cells	0.75
LR2–LR 2 Iterations for all cells	0.60

runtime overhead since each new algorithm would need many iterations to re-converge to the new solution after the "disruption" caused by the previously applied heuristic.

3.3 Reward function

The reward function should demonstrate the effect of the chosen optimization method on the performance metrics that characterize the circuit's behavior. The reward given to algorithm a at trial t, i.e., $Q_t(a)$, quantifies the improvement achieved by algorithm a in TNS and leakage power:

$$Q_t(a) = r_a \left(\delta \cdot qTNS + (1 - \delta) \cdot qPower\right)$$

The parameters $qTNS$ and $qPower$ are bound between 0 and 1, and they represent how efficient algorithm a was in reducing TNS and leakage power, respectively. Factor δ defines the importance of TNS optimization compared to leakage-power optimization, and can be changed during the flow depending on the design's current state. In this work, we target timing closure in the first optimization rounds and, thus, $\delta = 0.8$, while, once timing closure is achieved, power reduction is prioritized with $\delta = 0.2$.

Parameter r_a is a scaling factor that depends solely on the runtime complexity of the optimization algorithm a. Each optimization method has a different runtime complexity. We want to include this in our reward function, so that the MAB algorithm is able to effectively distinguish between two methods that have a similar QoR impact, but different runtime needs. In this way, we penalize the reward received by slow methods with low values for $r(a)$. The values of r_a used for each method are statically chosen and are listed in Table 3.1.

3.3.1 TNS reward

The value of $qTNS$ quantifies by how much TNS was improved in this trial relative to how much leakage power was increased. It is calculated separately for early and late timing modes $qTNS = 0.5 \cdot qTNS_{early} + 0.5 \cdot qTNS_{late}$, according to the following rules:
R1–Unproductive optimization: If TNS was degraded or remained unchanged, the algorithm receives $qTNS = 0$.
R2–Always productive: If both TNS and leakage power were reduced $qTNS = 1$, i.e., the algorithm receives the maximum reward,

R3–Tradeoff: If TNS was improved and the leakage power was increased, then

$$qTNS = \min\left(\frac{\Delta TNS}{\gamma\ \Delta P}, 1\right)$$

The $qTNS$ reward quantifies the percentage of TNS improvement over the percentage of power increase. This value can be larger than 1, so we apply a check and set it to 1 if it exceeds it. ΔTNS and ΔP represent the percentage of TNS and power change (between trials t and $t-1$) after the application of the selected optimization algorithm, i.e.,

$$\Delta TNS = \frac{TNS_t - TNS_{t-1}}{TNS_{t-1}} \text{ and } \Delta P = \frac{Power_t - Power_{t-1}}{Power_{t-1}}$$

When TNS_{t-1} is close to 0, it means that timing constraints are almost met. In this case, trying to calculate the relative TNS increase would lead to false/misleading conclusions. Due to the TNS_{t-1} term being the denominator, as its value approaches 0, any increase in TNS_t away from 0 would amount to a near-infinite relative increase, even though the absolute increase is – in reality – very small. Hence, any increase from a small TNS value would probably end up being artificially amplified, thereby derailing the convergence of the optimization algorithm.For example, let us assume that $TNS_{t-1} = -2$, and, after applying the next method, $TNS_t = -6$. The relative degradation would equal the huge value of 200%, while the real degradation is a minor setback compared to the original TNS at the beginning of the optimization process. To combat this artifact in the calculation of ΔTNS, whenever the value of TNS_{t-1} falls below a threshold value (arbitrarily chosen to be 10% of the clock period in our experiments), then the TNS degradation is quantified differently, as $\Delta TNS = TNS_{t-1}/T_{clock}$.

Variable γ defines what $TNS/power$ tradeoff we consider satisfactory. For example, let us assume that the TNS improvement is 40%, for a power increase of 20%. For $\gamma = 1$, the ratio of TNS change to power change will be equal to 2 $(0.4/0.2)$, thus setting $qTNS = 1$. In the case that $\gamma = 4$, TNS reward will be equal to $0.5 = \min\left(\frac{0.4}{0.2 \cdot 4}, 1\right)$. The value of γ should be neither too large, nor too small. A large value for γ will result in most rewards receiving very small values, whereas a small value of γ will result in most rewards being set to 1. In the experimental results presented in this paper, we assumed $\gamma = 4$.

3.3.2 Power reward

The reward with respect to leakage power, $qPower$ is calculated in a similar manner:
R1–Unproductive optimization: If leakage power was increased or remains unchanged: $qPower = 0$.
R2–Always productive: If the leakage power was reduced and the TNS was improved: $qPower = 1$. In this way, the algorithm achieved a truly productive step, since it achieved both a reduction in power and a timing optimization in the same step.
R3–Tradeoff: If the leakage power was reduced and the TNS was degraded, $qPower$ quantifies the relative percentage of power decrease relative to the TNS increase:

$$qPower = \min\left(\frac{\Delta P}{\gamma\ \Delta TNS}, 1\right).$$

3.3.3 Reward expansion

The reward $Q_t(a)$ given to algorithm a at trial t is always in the range [0,1]. Once the MAB-based optimization has played for many trials, we may observe an algorithm achieving only a marginal improvement compared to the improvements achieved during the first optimization rounds. Therefore, the reward given to an algorithm is very low and the learning algorithm cannot effectively distinguish the efficient algorithms. To avoid this, we re-normalize the reward function within the range [0,1], with the goal to enhance small rewards and compress very large ones: $Q_t^{\text{new}}(a) = \log(1-\frac{1}{b})\log(1-\frac{Q_t(a)}{b})$. Assuming a small value for $b = 0.05$, we are able to boost the small rewards observed. This is useful near the end of the simulation runs, where hard-to-verify bins remain uncovered.

3.4 Experimental results

Similarly to Section 2.5, the proposed design optimization flow has been implemented in C++ using the open-source RSyn physical design framework [37]. The new method is evaluated using two benchmarks sets; namely, the ISPD 2012 gate-sizing benchmarks [104] and the TAU 2019 benchmarks for design optimization [13]. The final results are validated with OpenTimer [54], which is the reference timer used for evaluation in both above-mentioned contests.

MAB orchestrator is responsible for selecting which timing-optimization heuristic to select in each phase. In this setup, we experimented with the following optimization heuristics:
LR gate sizing – one iteration – examine all cells (LR1): One iteration of Lagrangian Relaxation (LR)-based gate sizing for all cells of the design, as presented in Section 2.4.1.
LR gate sizing – two iterations – examine all cells (LR2): The same as the previous heuristic with the only difference being that LR optimization evolves for two iterations. It trades off increased runtime with better QoR.
LR gate sizing – one iteration – examine only 1K critical cells (LR1C): This heuristic represents a reduced version of the first one. It applies the same one-loop LR optimization, but it touches only the 1K most timing- critical cells. In this way, optimization finishes earlier, with slightly worse QoR.
Late buffering optimization (LB): Add increasingly larger buffer sizes next to the driver of a large-capacitance net until the ratio of the output load to the input load of each gate added locally is approximately 4 (from logical effort [55]).
Early Buffering at the endpoints (EB) : Add buffer with an input capacitance at least as large as the endpoint capacitance. Ensures that extra delay is always added, since the delay of the driving gate remains either the same or it is slightly increased.
Clock buffering insertion (CB): Insert an additional local-clock buffer on the clock pin of a flip-flop if we need to slow down the clock signal for this register, i.e., the D-pin late slack is more critical than the Q-pin late slack, or the Q-pin early slack is more critical than the D-pin early slack. We do not insert buffers if both sides are not critical.
Buffering for critical path isolation (CPI): Reduce the input capacitance of the non-critical branches of a net by adding buffers at the non-critical sinks, with an input capacitance smaller than the capacitance at each sink. In this way, the driving gate sees a smaller output load, which it decreases.

In the case of the ISPD 2012 benchmark set, we compare against the best results achieved so far in the open literature [38] for this benchmark set. The best results are derived from

Table 3.2: Comparing the quality of the autonomous MAB orchestration to the best published result for the ISPD12 benchmark set

Benchmark	#Cells	Leakage (mW)		Optimization algorithms						
		[38]	MAB	CPI	LB	EB	LR1	LR1C	LR2	#Trials
DMA_slow	25300	132	134	8	6	5	10	9	10	48
DMA_fast		238	248	11	9	8	14	11	10	63
pci_bridge3_slow	33203	96	96	10	7	6	10	9	12	54
pci_bridge3_fast		136	138	6	6	4	11	5	11	43
des_perf_slow	111228	570	601	9	6	4	11	8	5	43
des_perf_fast		1395	1511	8	7	5	11	12	12	55
vga_lcd_slow	164890	328	335	6	5	4	6	8	4	33
vga_lcd_fast		412	430	10	6	6	10	10	9	51
b19_slow	219268	564	660	4	2	2	4	3	3	18
b19_fast		717	769	6	3	3	5	5	4	26
leon3mp_slow	649190	1334	1340	6	6	3	5	5	5	30
leon3mp_fast		1443	1530	6	7	4	9	6	9	41
netcard_slow	958780	1763	1755	4	3	2	4	2	3	18
netcard_fast		1841	1884	4	3	3	5	4	5	24
Average		783.5	816.5	7.0	5.4	4.2	8.2	6.9	7.3	39.1

an LR-based optimization that runs for 120 iterations, together with final timing and power-recovery heuristics. The proposed MAB-based framework successfully approaches the best published results with marginal differences in less iterations, as shown in Table 3.2, by using a fully autonomous optimization and with no human in the loop. Note that Table 3.2 reports only the achieved leakage power, since, in all cases, initial timing violations are closed without introducing any maximum-capacitance or maximum-slew violations. In the case of the ISPD 2012 benchmarks, clock-tree buffering is not applied, since the included library does not include any clock buffers. These results are of paramount significance and unequivocally exemplify the strength of a MAB-based methodology: it paves the way for fully autonomous, self-driving, "no-human-in-the-loop" automation flows *without compromising (trading away) PPA quality* [119].

The MAB methodology chooses online which optimization algorithm to play in each round and records its reward. The reward that corresponds to the algorithms not played so far remains hidden. Therefore, the only way to learn this information is to repeatedly try all algorithms (exploration). In parallel, whenever the MAB process pulls a bad algorithm, it suffers some regret. The MAB methodology should reduce the regret by repeatedly pulling the best algorithm (exploitation)

The number of times that each algorithm has been selected for each benchmark is also shown in Table 3.2. On average, the heuristic with the most plays is LR1, followed by LR2, and CPI. As the optimization progresses, the leakage power reduction becomes smaller after each iteration. This means that the leakage power benefit of LR1 is more than half of the benefit of LR2. Due to the runtime scaling factor (r_a), UCB (the driving force behind MAB) ends up favoring the LR1. LR for critical cells is mostly beneficial in the early optimization stages, but it can also contribute in minimizing the small timing violations that might appear during the leakage-power optimization. CPI is the most efficient buffer insertion algorithm for these designs, as is shown by its high pick rate. The other two algorithms have minimal

Table 3.3: Initial and final timing and power of the TAU 2019 benchmark set under typical corner. The algorithm recommendations made by the MAB policy.

Benchmark	#Cells	Initial					MAB				
		Late (ps)		Early (ps)		Leakage	Late (ps)		Early (ps)		Leakage
		WNS	TNS	WNS	TNS	(uW)	WNS	TNS	WNS	TNS	(uW)
s1196	642	0	0	0	0	30	0	0	0	0	12.8
systemcdes	3441	0	0	346	5809	181	0	0	0	0	79
usb_funct	15743	0	0	984	244911	788	0	0	0	0	511
vga_lcd	139529	1499	14889000	2593	53407000	6002	0	0	0	0	3840

Benchmark	Optimization algorithms						
	LB	EB	CPI	CB	LR1C	LR1	LR2
s1196	2	1	2	1	1	1	2
systemcdes	2	4	2	2	4	2	4
usb_funct	5	8	5	11	8	9	11
vga_lcd	4	8	4	11	8	14	12

effect: (1) LB, because of the absence of very high fanout nets, and (2) EB, because of the absence of early timing violations in this benchmark set. Consequently, these two algorithms have the lowest pick rate.

The same experiments were repeated on the benchmarks of the TAU 2019 contest, using the typical library corner available. The obtained results are depicted in Table 3.3. It is evident that in all cases, timing constraints are met for both early and late timing while leakage power is significantly reduced. These benchmarks have a very high amount of hold violations. Gate sizing has a limited ability to fix them, as opposed to early buffer insertion, which inserts buffers directly at the violating endpoints, being able to virtually solve any hold violations. Thus, it is not surprising that the MAB algorithm picks this method the most. All gate-sizing methods have a similar pick rate, lower than the EB and higher than the LB. Since the designs have easy-to-fix late violations for this corner, LB is picked the least.

3.5 Conclusions

This work leverages Reinforcement Learning to automatically and autonomously apply optimization heuristics to unknown designs and improve their PPA metrics. The proposed MAB-based framework operates on-line, without any previous training, or knowledge of the design, or the optimization methods used. Yet, it achieves PPA metrics that approach the best published results so far. Most importantly, the introduced approach allows for a modular construction of a customized optimization flow, whereby optimization algorithms are added, or removed (even on-line), according to their effectiveness on a per-design basis.

4 LR Global Optimization and Tuned MAB-based Local Recommendations

4.1 Introduction

In Chapter 2 we demonstrated the interleaving of various optimization methods under the same Lagrangian Relaxation (LR) formulation, while in Chapter 3 we utilized the Multi-Armed Bandits (MABs) approach for orchestrating the order of execution of heterogeneous transformations within a loop. In the latter case, the MAB algorithm fails to adjusts to the dynamic nature of the problem, i.e., the design changes as it gets optimized and the same optimization would yield a different reward when applied in a different round. Therefore, the overall optimization followed in [132] is purely statistical.

This work tackles this problem by leveraging a modified version of Multi-Armed Bandits [70] to autonomously optimize unknown designs by choosing in each iteration of a Langrangian Relaxation (LR)-based optimizer which method to apply among a wide range of incremental optimization heuristics. Each heuristic is smoothly integrated without being disruptive to the overall optimization process, thus allowing the solution produced by one heuristic to gradually adapt to the solution produced by a previously applied heuristic. In all cases, the locally optimal decisions needed by each heuristic are taken using LR-based cost functions. This increases the modularity of the proposed approach, since any other local optimization heuristic with similar cost function could be included in the set of available optimizations.

The order of applying the optimization heuristics inside the LR-based optimizer is done autonomously by the proposed recommendation engine. For each one of the applied optimization algorithms, a reward is recorded based on a combination of PPA metrics and runtime. Improvements in PPA increase the reward given to each heuristic, while its increased execution time relative to a runtime target penalizes its future selection. The recommendation engine decides which optimization method to select next, following a balanced exploitation-exploration approach. Thus, **the optimization of each design, even if enclosed in the same LR-based optimization loop, evolves autonomously by adapting both to the design's unique characteristics and to the phase of the overall optimization**.

The proposed approach has been successfully applied to the TAU 2019 multi-corner design optimization contest benchmarks [13] and, for completeness, to the benchmarks of the ISPD13 gate sizing contest [105]. The introduced recommendation engine is demonstrated to be a very effective orchestrator of the overall optimization process. In all cases, the proposed approach successfully optimizes the designs and achieves significantly better results than state-of-the-art. Most importantly, the recommendation engine allows the addition of new heuristics that will be autonomously applied inside the LR-optimization loop, without requiring any manual tuning and without degrading the already achieved Quality-of-Results (QoR). Simultaneously, it addresses the drawbacks of [132] by employing an LR-based global optimizer which operates on a uniform cost function that is minimized locally by each selected optimization method. Also, the proposed rewarding mechanism and recom-

mendation system are more elaborate and calibrated appropriately for the dynamic nature of the optimization process.

4.2 Orchestrating LR optimization with autonomous recommendations

The LR formulation is identical to the one used in [130] and presented in Section 2.3, with the modification of scaling the leakage power, area and delay by η, β and θ, respectively, to tackle the issue of power, area, and delay having different units. The scaling factors are derived according to the method proposed in [114]. The final cost function is equal to:

$$\textbf{min} \sum_{c \in \text{cells}} \eta P(c) + \beta A(c) + \theta \sum_{i \to j} \lambda^L_{i \to j} d^L_{i \to j} - \lambda^E_{i \to j} d^E_{i \to j}) \tag{4.1}$$

In this work, we follow an iterative optimization approach, where each iteration tries to optimize the global cost function (4.1) using one out of a set of eight netlist transformations, i.e., netlist change or restructuring optimization heuristics that try to minimize localized versions of the global cost function.

The overall LR-based design optimization loop is shown in Fig. 4.1. Initially, all cells are sized to the smallest size that does not violate any maximum load or slew constraints [75]. Then, the Lagrangian Multipliers (LMs) for every arc are initialized to 1, and the LR-based optimization begins.

At the beginning of each iteration, and before applying the selected transformation, timing and LMs are updated using the process described in Sections 4.2.1 and 2.3.2, respectively. The netlist transformation that is applied on the design uses the updated LM values for all local optimum decisions. Which netlist transformation is applied to the design is decided using an autonomous recommendation engine that operates according to the rules described in Section 4.2.2.

The optimization loop stops when the solution quality does not improve for a number of consecutive iterations. If timing constraints are satisfied, the quality of the solution is equal to the area/power improvement. On the contrary, if timing violations are still present, termination is judged by TNS improvement. In the end, a set of brute-force recovery steps are executed.

To provide the recommendation engine the opportunity to run many different transformations before deeming that the design can no longer be optimized, the number of iterations that we wait before stopping the optimization is arbitrarily set to be 25% higher than the number of available transformations, i.e., 10 iterations for 8 netlist transformations.

4.2.1 Incremental timing updates – Critical corners

At the start of each iteration, we should be aware of the timing of the design for all corners under consideration. Initially, an incremental timing update for all corners is performed and the two critical corners are selected: the most critical early and late corner. The critical late corner is the one with the worst late TNS, or the corner with the lowest late total slack, if no corner has late violations. The same applies for the most critical early corner.

To save runtime during optimization, incremental timing update on all corners is performed once every five iterations. In the other iterations, timing updates are performed

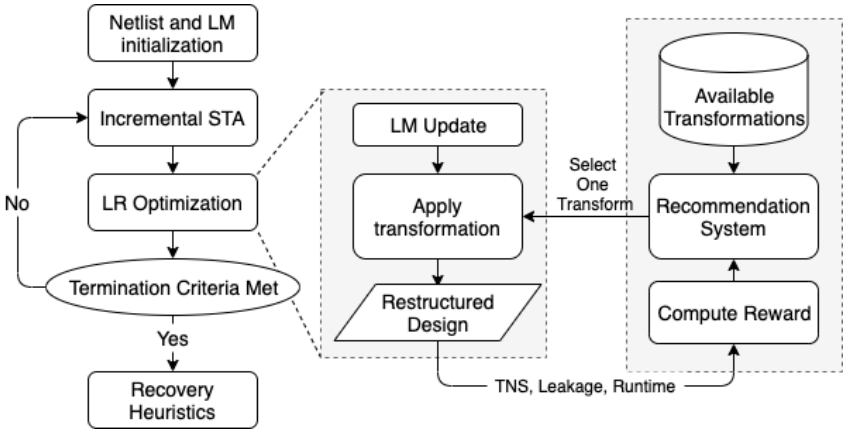


Figure 4.1: The overall design optimization flow. Netlist transformations are applied sequentially on the design, as selected autonomously by the recommendation/rewarding system. Each transformation relies on LM weights, thus contributing to the convergence of the global LR-based optimization loop.

only on the already identified critical corners. The critical corners are not statically determined throughout the optimization, but they are re-evaluated every five iterations of the LR optimization loop. Our experimental results show that updating the most critical corners more often introduces unnecessary runtime overhead without significant improvement of the overall QoR.

4.2.2 Automatic transformation recommendation

The proposed recommendation engine takes the performance observed by each netlist transformation so far and tries to pick the next transformation. The goal is to exploit as much as possible the current best transformation, while maintaining some exploration of the other transformations, in case one of them becomes more efficient at a later stage of the optimization. Well known bandit algorithms, such as the Upper Confidence Bound (UCB) and its variants [4], have been proven to optimally solve the exploration vs. exploitation dilemma, albeit in a *stationary* context [70]. However, the design optimization problem that we are trying to solve is inherently *dynamic*. The design is altered in each iteration, and successive plays of the same optimization algorithm would yield different rewards [132]. To bypass this inherent inefficiency when using bandit algorithms in a dynamic environment, we make decisions based on differential criteria, following the approach used in OpenTuner [3].

Score Computation: When transformation k has been applied on the design in a certain iteration, we record its earned reward, R_k, using the reward function (4.5) described in Section 4.3. Then, we compute the average payoff for each one of the transformations, as follows:

1. Sort the transformations applied in the last N trials according to their received rewards. In the example shown in Fig. 4.2, the last five ($N = 5$) trials are ranked based on the reward earned when applied to the design. Transform A had the highest reward of 0.7 and it is ranked first. Transform C follows with a reward of 0.5.

2. Each transform receives a score according to its position in the rank. Position 1 receives a score of N, position 2 a score of $N - 1$, while the last position receives the

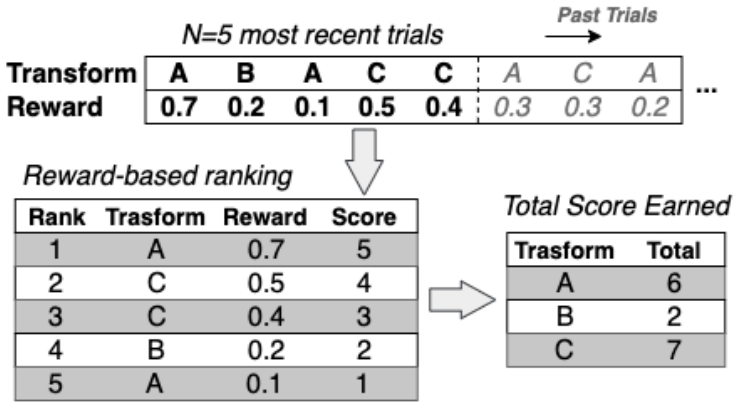


Figure 4.2: Reward-based relative ranking of the transformations applied in the last N most recent trials.

lowest score of 1.

3. Compute the total score of each transform by adding the score of all of its appearances in the rank list. Thus, in the example of Fig. 4.2, C receives a total score of 7, since it occupied the second and the third positions in the rank with corresponding scores of 4 and 3. The total score of each transform actually reflects how good the transform was in the last N trials relative to the others.

The normalized payoff Q_k of each transform k is just the normalized version of its total score:

$$Q_k = \frac{\text{score}(k)}{1 + 2 + ... + N} \tag{4.2}$$

Selection policy: The recommendation engine would select transformation k to apply in this trial, which maximizes

$$(1 - c)Q_k + c\sqrt{\frac{2 \ln N}{N_k}}, \tag{4.3}$$

where N is the window size and N_k is the number of times that k has been played in the last N trials. c is a tuning variable between 0 and 1 that, when increased, favors the exploration of rarely selected transformations (low values of N_k) versus exploiting the ones with the best performance so far (high values of Q_k). For the example shown in Fig. 4.2, $(Q_A, N_A) = (6/15, 2)$, $(Q_B, N_B) = (2/15, 1)$, and $(Q_C, N_C) = (7/15, 2)$. Assuming a balanced exploit vs. explore strategy with $c = 0.5$, transformation B would be applied in the next trial.

The initial payoffs Q of each transform can be obtained either by applying each transformation once (no pretraining), or by using the average scores derived from the previous application of this algorithm on various benchmarks (pretraining).

Productivity Rule: Once a transformation is applied on the design, the changes that it caused *are not necessarily kept*. If the quality of the obtained result is worse by more than 10% in any of the targeted aspects, i.e., timing, leakage power, and area, then the changes are discarded and the design is recovered to its previous state. We allow this small degradation to occur, since it may help the subsequent transformations. For instance, when timing is closed, meaning that timing violations are below a certain threshold [124] (arbitrarily chosen to be

10% of the clock period in our experiments), and we are optimizing for leakage power and area, we allow for small power degradation, since this can lead to positive timing slack that would allow us to optimize leakage power later on.

4.2.3 Netlist transformations

The automatic recommendation engine embedded inside the LR-based optimization loop utilizes a set of eight netlist transformations that collectively optimize the design.

Early and late buffering, pin swapping, clock skew scheduling and cell merging/splitting are applied as presented in Section 2.4. In regards to sizing, we employ two separate sizing transformations in order to allow the recommendation engine to trade off higher sizing quality with lower runtime: *full cell sizing* that examines every sequential and combinational cell; and *fast cell sizing* that operates on a selected subset of cells. Specifically, when timing is not closed, fast cell sizing resizes only cells that either have negative slack on their pins, or are immediate fanouts of cells with negative slack (i.e., to downsize a load to speed up the path). When timing has closed, fast cell sizing picks the cells with the highest power/area potential that also have positive slack. Power potential refers to the difference between their current leakage power and the smallest leakage power they can have without causing maximum slew and load violations, which is the leakage they had after the initial downsizing. Area potential is defined similarly. Each cell is sized in the same manner as described in Section 2.4.1.

All of the above methods are driven by the LM values of each iteration and try to optimize their LR-based cost functions using only slack information to ensure that they are not degrading the timing violations. Any other local optimization heuristics that operate in a similar manner can be added in the library of available transformations. The local cost function for resizing a cell of size v, including the scaling factors is equal to:

$$\eta P(v) + \beta A(v) + \theta \sum_{i \to j \in \text{local\_arcs}} \left(\lambda^L_{i\to j} d^L_{i\to j} - \lambda^E_{i\to j} d^E_{i\to j}\right) \tag{4.4}$$

where $P(v)$ and $A(v)$ are the leakage power and area, respectively, of size v of the examined cell.

The recovery heuristics used in this work include the ones presented in Section 2.4.6, while also adding a power recovery heuristic. In contrast to the eight previous transformations, these recovery steps cannot be selected by the recommendation engine.

The power/area reduction step attempts to downsize every gate in forward topological order [38]. If the downsize degrades the local slack, the move is reverted; else it is kept. After every forward topological pass, an incremental timing update is performed. The process stops when the TNS is degraded, or when no more downsizes are possible.

4.3 Guiding the automatic recommendation: The reward function

To sort out the relative performance of the most recently applied transformations, each transformation k applied to the design in the ith iteration receives a reward $R_k(i)$. The reward function demonstrates the effect of the chosen optimization method on the performance metrics that characterize the circuit's behavior, as well as to the runtime cost paid in achieving

such metrics. According to the productivity rule, a reward is recorded only if the restructured netlist is actually kept, i.e., it does not deteriorate any QoR metrics by more than 10%. Overall, the reward $R_k(i)$ is equal to:

$$R_k(i) = r_k \left(\delta R_{\text{timing}}(i) + (1-\delta) R_{\text{power-area}}(i)\right) \tag{4.5}$$

$R_{\text{timing}}(i)$ and $R_{power-area}(i)$ represent how efficient the selected transformation was in improving TNS, and reducing leakage power or area, respectively. Factor δ defines the importance of timing optimization compared to power-area optimization and it is changed during the flow's execution. For instance, in the first iterations, we prioritize timing optimizations with $\delta = 0.8$. Once timing closure is achieved, power-area reduction is prioritized by changing δ to 0.2.

Parameter r_k is a scaling factor that depends solely on the runtime of transformation k. We include this in the reward function, so that the recommendation algorithm is able to distinguish between two methods that have a similar QoR impact, but different runtime needs. We penalize the reward received by slow methods with low values for r_k. Initially, $r_k = 1$ for all transformations. On the following iterations,

$$r_k = 1 - \frac{\text{avg\_runtime}_k}{\text{runtime\_target}} \tag{4.6}$$

A high average runtime for method k will penalize that method when it is comparable to the runtime target. Setting a strict runtime target can reduce the total runtime of the optimization, since faster methods will be picked more often by receiving higher rewards, possibly at the cost of inferior QoR. When assigning a relaxed runtime target, r_k is used as a tie-breaker between equally efficient methods in terms of timing and power/area, but with different runtimes.

4.3.1 Timing reward

The timing reward $R_{\text{timing}}(i)$ represents the tradeoff between timing improvement $\Delta T(i)$ and power/area overhead $\Delta PA(i)$ achieved in the ith iteration.

$$R_{\text{timing}}(i) = \begin{cases} \frac{\Delta T(i)}{\min(\Delta PA(i), 0.01)}, & \text{if timing is improved} \\ 0, & \text{if timing is not improved} \end{cases}$$

If timing closure has not been achieved, timing improvement ΔT refers to the average percentage of TNS improvement between two consecutive iterations i and $i-1$ for late and early mode:

$$\Delta T(i) = \frac{1}{2}\frac{\text{TNS}_i^L - \text{TNS}_{i-1}^L}{\text{TNS}_{i-1}^L} + \frac{1}{2}\frac{\text{TNS}_i^E - \text{TNS}_{i-1}^E}{\text{TNS}_{i-1}^E} \tag{4.7}$$

After timing closure, ΔT refers to the percentage of total slack increase.

Similarly, $\Delta PA(i)$ is the percentage increase of the total sum of scaled leakage power and area of all cells in one iteration relative to the previous iteration:

$$\Delta PA(i) = \frac{\sum(\eta P + \beta A)_i - \sum(\eta P + \beta A)_{i-1}}{\sum(\eta P + \beta A)_{i-1}} \tag{4.8}$$

The definition of the timing reward reflects our strategic decision *to prefer overall efficiency over exclusive TNS improvement*. For instance, a method that achieves TNS reduction of 10%, while degrading power and area by 5%, is preferred over a method that reduces TNS by 20%, but degrades power and area by 15%.

$\Delta PA(i)$ is lower-bounded to 0.01 in the denominator of R_{timing} to avoid cases where the power/area difference is too small and diminishes the impact of timing in the reward. A very low $\Delta PA(i)$, can cause the timing reward to overshoot regardless of the actual timing improvement. Without this lower bound, a method that marginally improves timing by 0.1% and degrades leakage power and area by 0.001% would receive a timing reward of 100, which is unrealistically high for the small impact the method had. By lower-bounding the power and area improvement, the reward will be equal to 0.1.

4.3.2 Power-Area reward

The power/area reward has a different meaning depending on the phase of the optimization. If timing has not closed, power reward reflects the tradeoff between timing degradation and power/area improvement. If timing has closed, and the LR-optimization focuses on power/area reduction, the reward is simply the percentage of power/area reduction across iterations, irrespective of timing. This is safe to do, since, according to the productivity rule, if the applied transformation worsens any quality metric by more than 10%, its effect is not kept and the design is restored to its previous state. Therefore, once timing has closed in some iteration, any timing degradation that may appear as a result of the applied power-area optimizations will be very small due to the 10% per-iteration limit. Overall,

$$R_{\text{power-area}}(i) = \begin{cases} 0, & \text{if power-area increased} \\ \frac{\Delta PA(i)}{\min(\Delta T(i), 0.01)}, & \text{if timing is not closed} \\ \Delta PA(i), & \text{if timing is closed} \end{cases}$$

This time, we bound $\Delta T(i)$ to 0.01, in order to avoid methods that cause a minimal TNS change getting high rewards regardless of their power and area effect.

4.4 Experimental results

The proposed method was implemented in C++ inside RSyn [37] after extending it for multi-corner timing analysis. The presented results were evaluated using the benchmarks of TAU 2019 [13] and ISPD 2013 contests [105]. All experiments were performed on a Linux workstation using a 3.6 GHz Intel Core i7-4790 with 4 cores and 32 GB of RAM. The results are validated using OpenTimer [54], which is the reference timer in the TAU 2019 contest.

4.4.1 Setup of the recommendation engine

The recommendation engine operates on a sliding window of $N = 32$ rewards (four times the number of available transformations). A very small window is prone to being overflowed by one method, which is then repeatedly ranked highly, not because of its quality but because of its high presence in the window. On the other hand, a very large window can give a method a high ranking because of its performance many iterations ago, even if the timing profile of the design has completely changed.

Table 4.1: The initial average rewards for each transformation. When timing is not closed, the timing reduction rewards are used. When timing is closed, rewards are re-initialized to the average reward of the power reduction phase.

Method	Full sizing	Pin Swap	Late Buff.	Hold Buff.	Clock Skew	Gate Merge	Gate Split	Fast sizing
Timing reduction	0.133	0.043	0.129	0.012	0.247	0.078	0.176	0.180
Power reduction	0.262	0.027	0.020	0.020	0.128	0.182	0.081	0.282

To avoid blindly running every heuristic at the start of the optimization, the recommendation engine was pre-trained using ten average-sized benchmarks from the TAU 2019 and ISPD13 benchmark suites. Each one of these benchmarks was optimized by running every method serially (for example, full gate sizing on iteration 1, pin swapping on iteration 2, late buffering on iteration 3, etc.) and storing the rewards of the 32 first iterations. Thus, the initial window included four times each all eight transformations. The duration of the pre-training runs was 10 minutes.

During pre-training, we recorded for each transformation two sets of average rewards. The one refers to the timing reduction phase (i.e., when timing is not closed) and the other to the power reduction phase (i.e., when timing is closed and power-area reduction is targeted). The initial rewards of each transformation derived in the training phase are shown in Table 4.1. At the start of the optimization, the initial sliding window is populated by the average rewards of the transformations during the pre-timing closure phase. Once timing closure is achieved, the sliding window is re-initialized to the average rewards of each transformation in the power reduction phase.

The behavior of the recommendation engine also depends on the explore coefficient c in (4.3). Setting c high will treat all methods equally, instead of focusing on the most efficient ones. On the contrary, setting c at a low value will limit the optimization to only a few methods, failing to reap the benefits of the combined application of all heuristics. We observed the best results when $c = 0.35$ to 0.5. The results presented were run for c being equal to 0.4.

Finally, the runtime target of the runtime scaling factor (4.6) that determines how much each transformation is penalized due to runtime was set to 40 mins for all designs. This target actually has no effect in small designs and only constrains the runtime of large designs. Larger runtime targets improved only slightly the overall QoR. Note that the runtime target is not a hard constraint, but just a penalty factor that helps increasing the rewards of fast transformations relative to equally-good but slower transformations.

4.4.2 Results on the benchmarks of the TAU 2019 contest

The TAU results were compared against the TAU contest winner's results [17], which were extracted by running the executable that they submitted to the contest (and also provided to us). Each benchmark includes SPEF and SDC files for each netlist. The TAU benchmarks provide five different standard-cell library files, one for each corner. Each library provides a range of cells of varying complexity. It includes positive, negative and non unate cells, registers with Set/Reset pins, and complex combinational cells, such as half and full adders.

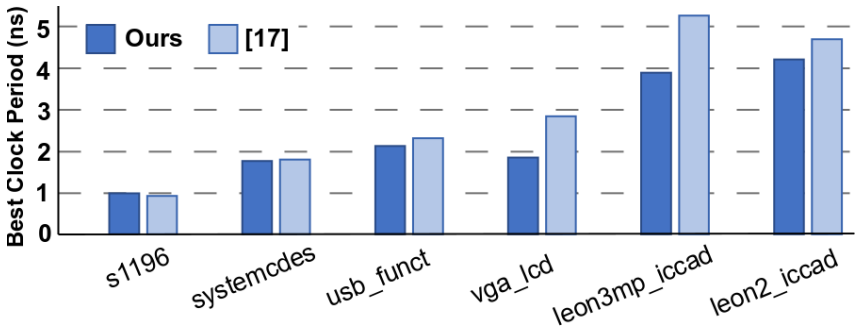


Figure 4.3: The best clock period achieved for each benchmark of the TAU 2019 contest covering all corners, by the proposed method and the winner of the contest [17].

Each library cell has a variety of different versions, ranging from 1 to 6 separate sizes and 1 threshold voltage. The designs only have timing constraints on register-to-register paths, leaving primary inputs and outputs unconstrained. There is also a different transition time constraint for each corner.

As described in Section 2.5, we removed all clock buffers from both the initial set of benchmarks fed to our algorithm, and from the final netlist of the TAU 2019 winner, while preserving the assigned clock pin arrival times for the latter. This ensures a more realistic comparison in a pre-CTS stage.

Fig. 4.3 presents the best clock period for which each flow achieves closure for all corners. Our flow achieves 17% lower period, on average. We achieve a better clock period for every design, with the exception of s1196.

Table 4.2 depicts the leakage and area achieved when both flows are constrained by the worst clock period of Fig 4.3. Our flow achieves a lower leakage and area for every benchmark, resulting in 6% area and 15% power improvements, on average. Our QoR on the larger benchmarks is much superior to the result of the contest winner, in exchange for a longer runtime. The runtime overhead mainly stems from methods that operate on the entire design, such as full gate sizing and cell merging/splitting. Even if gate-sizing is multithreaded using eight threads each time is applied, all the rest transformations are single-threaded.

The proposed recommendation engine can operate equally well (albeit with an increased runtime), even if it is not pre-trained and the average payoff of each transformation is initialized to 0. Table 4.3 depicts the leakage power and area results, when the designs are optimized for the clock period of Fig. 4.3 (the best our flow can achieve), with and without

Table 4.2: Comparison of the leakage and area at the clock period where both flows achieve closure for all corners

Design	#cells	Period	Leakage (μW)			Area (μm^2)			Runtime (min)		
		(ps)	Ours	[17]	Win(%)	Ours	[17]	Win(%)	Ours	#It.	[17]
s1196	584	985	12	13	7.69	552	569	2.99	0.03	24	0.03
systemcdes	2825	1788	74	96	22.92	3368	3975	15.27	0.19	46	0.35
usb_funct	10535	2306	370	402	7.96	17599	17812	1.20	0.77	45	0.87
vga_lcd	87958	2826	2780	3106	10.50	142153	146257	2.81	6.82	20	0.40
leon3mp_iccad	649191	5246	19351	23816	18.75	957947	1030860	7.07	57.25	22	6.03
leon2_iccad	793286	4677	23989	30354	20.97	1203920	1312960	8.30	46.72	19	7.53
average saves	-	-	-	-	14.80	-	-	6.27	-	-	-

Table 4.3: Comparison of the leakage and area at the best clock period achieved by our flow, with (with pr.) and without (w/o pr.) using pretrained reward initialization.

Design	Period (ps)	Leakage (μW)		Area (μm^2)		Runtime (min)	
		with pr.	w/o pr.	with pr.	w/o pr.	with pr.	w/o pr.
s1196	985	12	12	552	551	0.03	0.03
systemcdes	1754	80	81	3510	3533	0.16	0.17
usb_funct	2116	404	412	18412	18654	1.51	2.89
vga_lcd	1840	2796	2781	140813	141477	9.42	10.33
leon3mp_iccad	3872	19686	19613	963614	962976	92.84	93.68
leon2_iccad	4190	24170	24003	1208100	1206040	91.81	110.35

pretraining (with pr. and w/o pr., respectively). In all cases, it is evident that the proposed flow, *even when applied blindly*, achieves equally good, or better QoR. Pre-training just improved runtime by favoring faster convergence.

Therefore, the proposed optimization flow not only achieves timing closure and competitive power/area reductions, but it also does so autonomously, without relying on any previous knowledge of the circuits, and without requiring any manual tuning/intervention.

In order to demonstrate the importance of the proposed recommendation engine to the final QoR, we ran additional experiments, whereby the recommendation engine is disabled and the transformation executed in each iteration is picked randomly. The final WNS and the leakage power for 50 random experiments on the *leon3pm_iccad* design are presented in Figure 4.4, when the designs are optimized for the best clock period of Fig. 4.3. The results are sorted by the achieved WNS.

The 4 trials shown on the right-hand side of Fig. 4.4 managed to close timing, but with worse leakage power than the proposed approach, while the other 46 trials achieve a wide range of worse timing and leakage performance. It should be noted that, even if the applied transformations are selected randomly in each iteration, their result is kept only if it improves the design according to the proposed productivity rule. Therefore, in the end, only

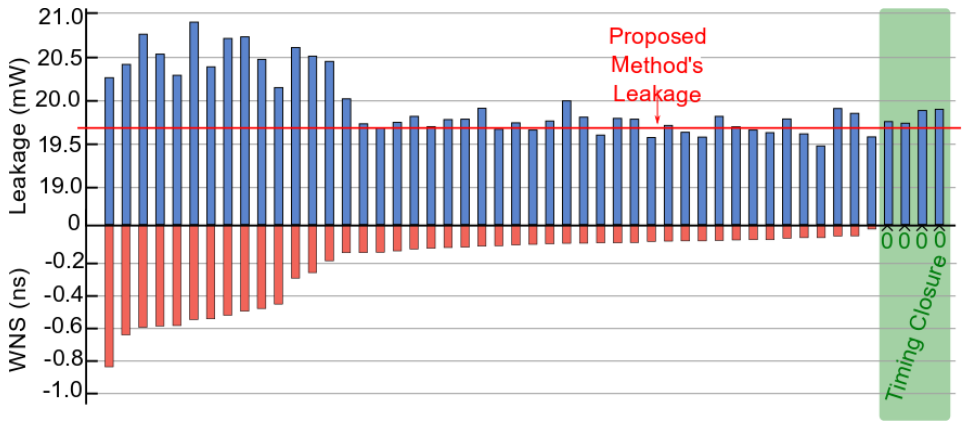


Figure 4.4: WNS and leakage power for 50 random experiments on the leon3mp_iccad, without using the proposed recommendation engine and targeting the best clock period achieved by the proposed flow.

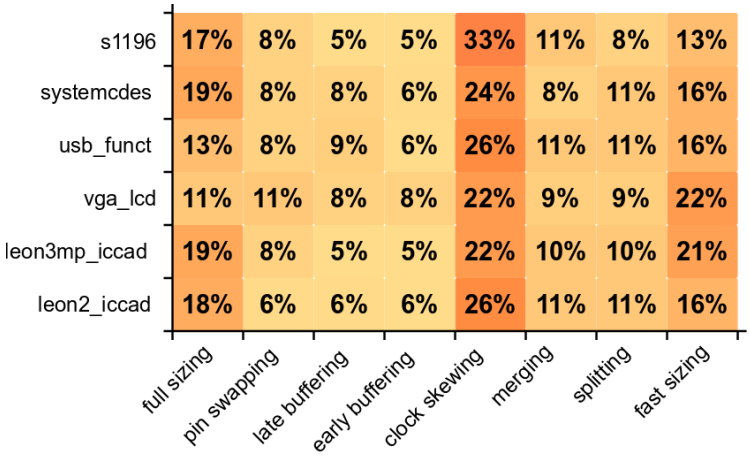


Figure 4.5: How often each transformation is selected per benchmark.

the results of useful transformations are kept. This indicates that, even if the proposed netlist transformations provided to the LR-based optimizer are effective, they cannot reach the QoR achieved when these methods are appropriately rewarded and orchestrated, as done by the proposed recommendation system.

4.4.3 The interaction of netlist transformation with the recommendation engine

In this sub-section, our goal is to demonstrate the behavior of the recommendation engine, and which transformations it decided to prioritize in order to achieve the overall results of Table 4.2. Fig. 4.5 depicts the percentage that each transformation is utilized per benchmark during the LR-optimization loop, as decided independently, by the MAB-based recommendation engine. In all cases, the lion's share belongs to useful clock skew and the two versions of cell sizing, since they are the strongest methods for both TNS and power reduction. Because of the runtime scaling factor, the optimization method picks fast gate sizing as often as full gate sizing.

Hold buffering usually has the lowest pick rate, since the designs do not originally have early violations, so it can only be useful around the end of the flow. Even then, since we do not allow for early TNS to degrade too much, early buffering cannot receive a high enough reward.

Late buffering is picked infrequently for the opposite reason. It is very useful in the first iterations to buffer large fanout nets, but its benefit diminishes later on. This behavior is highlighted in Fig. 4.6, which depicts the normalized payoff, calculated on equation (4.2) of early and late buffering over time for the design usb_funct. The payoff of late buffering is high at the start of the flow and gradually decreases, while the payoff of early buffering increases over time, but still stays low. Recall that the payoff demonstrates how high or low a method is ranked compared to the other methods.

Utilizing more elaborate methods for buffering nets with timing violations than the local buffering approaches used in this work could possibly increase the pick rate of buffering transformations. It is part of our future plans to adapt efficient dynamic buffering techniques to use LM-based cost functions and include them in our set of available transformations.

Gate merging receives higher rewards in the power reduction phase, since it reduces the

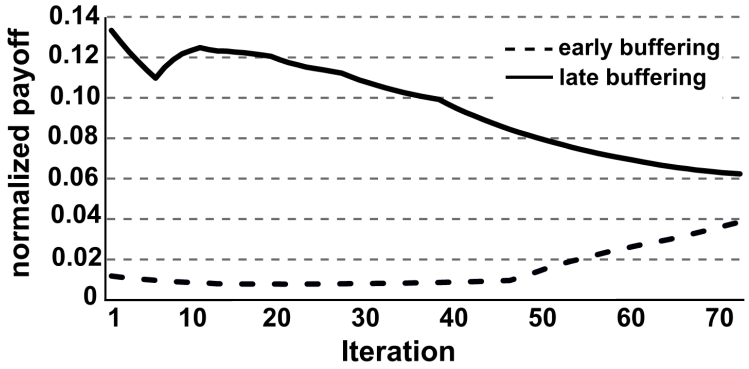


Figure 4.6: The normalized payoff of early and late buffering over time for usb_funct.

total number of cells, while gate splitting is more effective during the timing reduction phase. Pin swapping has a low pick rate, since its immediate effect on the timing is usually small.

Table 4.4: The impact of omitting a transformation on the results for usb_funct. In all cases listed below the "full flow" row in the table, hold timing was closed, as ensured by the post-pass of inserting buffers to fix any remaining hold violations.

Netlist		TNS (ns)	WNS (ns)	Leakage (μW)	Area (μm^2)	Runtime (min)
Initial		-392.697	-1.104	392	17561	-
Initial downsizing		-736.635	-1.678	394	17597	-
Full flow		0	0	404	18412	1.51
Opt removed	Full sizing	-0.010	-0.010	408	18549	1.67
	Pin Swap	-0.052	-0.052	413	18352	1.36
	Late Buff.	-0.036	-0.036	399	18466	2.24
	Hold Buff.	-0.027	-0.027	398	18322	1.54
	Clock Skew	-1.375	-0.075	426	18908	9.76
	Gate merge	0	0	413	18671	1.59
	Gate split	-0.138	-0.072	395	18234	1.38
	Fast sizing	-0.027	-0.027	407	18507	1.28
	Sizing	-0.030	-0.030	455	19490	4.19

In order to better quantify the contribution of each method to the final QoR, we re-ran our flow by excluding an optimization method on each run. Table 4.4 presents the results for the indicative design usb_funct for the best achieved period. It can be observed that clock skew scheduling is the most efficient method for reducing timing violations, and cell sizing for reducing leakage power and area. Nevertheless, removing any method will worsen the QoR or the runtime, as the optimization framework will need to achieve its effect by running other methods for potentially more iterations, which is an overhead that can become prohibitively expensive for larger designs. Therefore, all methods are required to achieve the best QoR in the fastest runtime.

The number of netlist transformations employed does not alter the tuning of the proposed recommendation system. In fact, appropriate netlist transformations can be added, or re-

Table 4.5: Autonomous optimization using in each case a reduced number of heuristics on the usb_funct design.

# Opt. used	Opt. available	TNS (ns)	WNS (ns)	Leakage (μW)	Area (μm^2)	Runtime (min)
2	Clock Skew Fast sizing	-0.601	-0.222	390	18118	1.09
4	Previous + Full sizing Gate merge	-0.554	-0.153	393	18187	1.75
6	Previous + Gate split Pin Swap	-0.042	-0.037	402	18378	1.11
8	Previous + Late Buff. Early Buff.	0	0	404	18412	1.51

moved, in a plug-and-play manner. To demonstrate this, we ran experiments with a reduced number of available netlist transformations. Also, to examine if the recommendation system can still adapt and produce reasonable results, despite the varying number of available optimization methods, *we did not retune any parameter for those runs.*

The results obtained for the same indicative *usb_funct* design are presented in Table 4.5. In the first scenario, we use fast cell sizing and clock skew assignment as the only available transformations. Both represent the two most utilized methods according to Fig. 4.5. The following three scenarios each add two more transformations, following the order of their average utilization. In every case, the obtained timing, power, area, and runtime results are reasonable. The quality improves as more netlist transformations are added, since the proposed recommendation system can autonomously adapt and combine the benefits of a wider range of methods, without requiring any manual calibration.

4.4.4 Results on the benchmarks of the ISPD 2013 contest

The presented flow was also applied on the ISPD 2013 gate sizing contest benchmarks. In this case, each cell in the library has ten sizes available at three threshold voltages, with a total of 30 sizes per cell. There is only one setup timing corner, and registers are actually non-sizeable, since there is only one version of flip-flop cells available. For each benchmark, two clock period targets are given: a fast and a slow target. Each design has a SPEF file describing a set of RC parasitics for each net. The ISPD library file does not include buffers, so they are replaced by inverters. Every buffering heuristic ensures that the number of added inverters is even.

The results obtained after running the proposed optimization on the benchmarks of the ISPD 2013 contest are depicted in Table 4.6. Since timing constraints were met in all cases, only leakage power is reported together with the measured runtime. Table 4.6 also includes the results achieved by two state-of-the-art gate sizing methods [38, 125], which yield the best results in this benchmark set.

The proposed optimization that includes all eight available netlist transformations achieves

Table 4.6: Results for the ISPD 2013 contest benchmarks.

Design	#cells	Leakage (mW)			Runtime (min)		
		Ours	[38]	[125]	Ours	[38]	[125]
usb_phy_slow	623	1	1	1	0.03	0.49	0.22
usb_phy_fast		1	2	2	0.06	0.42	0.23
pci_bridge32_slow	30763	53	57	58	1.93	10.53	0.97
pci_bridge32_fast		60	85	90	1.70	22.62	1.54
fft_slow	33792	84	87	88	1.61	25.71	1.37
fft_fast		131	194	213	3.30	40.43	1.64
cordic_slow	42937	220	267	293	4.24	69.04	2.29
cordic_fast		785	980	1080	4.38	117.08	5.66
des_perf_slow	113346	340	327	332	14.38	132.27	7.27
des_perf_fast		663	644	639	21.19	347.87	26.1
edit_dist_slow	129227	428	416	440	5.74	123.90	4.92
edit_dist_fast		532	535	549	10.44	352.96	6.66
matrix_mult_slow	159642	451	443	448	7.62	226.13	8.80
matrix_mult_fast		721	1541	1633	21.56	395.96	13.9
netcard_slow	984094	3663	5155	5170	58.36	483.55	24.6
netcard_fast		3772	5182	5205	64.52	400.89	30.6

superior results in most cases, achieving 25% better leakage power, on average, than the most efficient previous work [38].

The runtime evaluation results in Table 4.6 indicate that the proposed approach is comparable to the state-of-the-art. Note that the reported runtimes for the gate-sizing-only techniques are taken verbatim from their respective papers. Consequently, those runtimes correspond to other machines with different specifications than the one we used. Therefore, the comparisons can only be broadly and generally indicative. Regardless, we include those runtime numbers here in a big-picture context, to demonstrate that the runtimes of the proposed approach are reasonable, as compared to the others.

Finally, it should be noted that, since our cell sizing algorithm is effectively an enhanced version of the LR-based sizers used in [38] and [125], our optimization can approach the results reported in said papers using gate-sizing-only transformations. The extra savings reported, which are significant for the larger benchmarks, are the outcome of the autonomous orchestration of all available netlist transformations. More specifically, like in the TAU 2019 benchmarks, cell sizing and clock skew scheduling are the most efficient methods for reducing leakage power and timing violations respectively, with the rest of the methods having smaller but considerably positive impact on the final result.

4.5 Conclusions

This work investigates adaptive design optimization techniques, where a modified version of Multi-Armed Bandits is employed to autonomously optimize unknown designs. In each iteration of an LR-based global optimizer, the MAB-based orchestrator chooses a particular

method to apply among several netlist transformations. The introduced recommendation engine can start either blind-ly, or with an initial estimate of rewards from profiling the optimization transforms across a subset of smaller, representative benchmarks. It adapts on-the-fly across iterations, based on analysis of which optimization approaches have been most successful thus far on this design. This dynamic adaptation is performed while balancing the exploration of alternative optimizations versus the reward, and by avoiding the need for any manual tuning. The proposed approach achieves superior results on the TAU 2019 and ISPD 2013 benchmarks, as compared to current state-of-the-art.

We have provided a new generalized paradigm for a digital circuit optimization engine that combines the following: (a) state-of-the-art Lagrangian Relaxation (LR) that identifies how best to weigh (with Lagrange multipliers) the delay versus power tradeoff at each cell to achieve lower power and timing closure; (b) plug-and-play capability for different optimization transforms that are typically used in an industrial Electronic Design Automation (EDA) flow; and (c) a system to recommend which type of optimization may give more benefit at this point during optimization. The heuristics optimize the design based on the local LR cost function, harmoniously operating under the guidance of the proposed recommendation engine to achieve timing convergence and lower power. Any optimization method that can be adapted to use the local LR cost function can be added.

5 Reinforcement Learning-Based Synthesis of Approximate Adders

5.1 Introduction

Approximate computing that sacrifices accuracy in computations for energy savings has emerged as a promising approach for addressing the growing energy demands of modern computing systems [128]. By intentionally introducing errors into hardware components, approximate computing techniques aim to reduce power consumption without significantly impacting overall system performance. A variety of error-resilient applications make use of approximate computing for area and power reduction, such as image processing [113], FFT hardware components [34], clock generators [25] and neural networks [91].

One of the key challenges in designing approximate hardware lies in identifying the optimal trade-offs between accuracy and energy efficiency. In this paper, we explore the design space of approximate adders. Instead of focusing on a specific adder architecture, we design approximate parallel prefix adders that effectively represent a wide range of adder architectures that span from serial ripple-carry adders to fast carry-lookahead adders [155]. Optimizing the structure of parallel prefix adders is a complex task that has been extensively studied in the past for accurate parallel prefix adders [65, 90, 116, 117].

By automating the synthesis of approximate parallel prefix adders tailored to specific applications, we aim to provide a flexible and efficient approach to harnessing the benefits of approximate computing.

Previous research on approximate parallel prefix adders has focused on various techniques to reduce their complexity and energy consumption. These approaches include shortening the carry chains [30, 31], pruning lower levels of the adder [20], and removing logic gates from specific cells [121]. While these methods have yielded promising results, they often rely on predefined architectures, limiting the exploration of the entire design space. Our recent work [133] has sought to address this limitation by proposing a more comprehensive approach to the design of approximate parallel prefix adders.

The work in [133] enumerates all possible approximate parallel prefix adders that satisfy the designer's constraints, such as carry chain length and the number of prefix levels. This exhaustive enumeration led to the discovery of highly efficient approximate adder designs. However, even if we can identify all possible approximate adders for a given set of constraints, there is no clear way to determine which one is the best for a specific application. Not all approximate adder architectures work well for every problem. Therefore, to approach optimal results, approximate adders should be customized for each application.

In this work, approximate parallel prefix adder customization is performed using Reinforcement Learning (RL) [98, 134]. **RL optimizes the hardware's area and, at the same time, minimizes the approximation error at the application level by incorporating appropriate feedback during learning.** The evaluation of solutions during learning does not rely on heuristics or abstract performance prediction metrics but uses a neural network that

learns strategies purely through exploration of the unexplored design space and direct feedback from logic synthesis and application-level performance [98]. In this way, **for each examined application, we can automatically synthesize bespoke approximate parallel prefix adders** without the need to rely on generic architectures that may perform poorly on certain occasions. Overall, the novelty of the proposed approach can be summarized as follows:

- The introduced RL agent learns how to synthesize adders on a per application basis, satisfying certain application-level quality metrics and hardware complexity constraints. Effectively, the RL framework identifies the specific approximation strategy that maps to energy-efficient hardware and offers the quality needed by the application and the associated user constraints.
- The RL agent is able to adapt its approximation strategy to the input data that the application uses, taking into consideration both signedness and data range, resulting in even more efficient adder architectures.
- The proposed approach can generate various approximate adders that outperform existing designs. These adders have been rigorously tested using both synthetic data and real-world applications. In all cases, they are superior or at least equally efficient to state-of-the-art parallel prefix adders by either improving the area and power consumption by 12% and 10% on average, respectively, without degrading the application performance, or by significantly improving the application level error behavior without increasing the area and power consumption.

5.2 Accurate and approximate parallel prefix addition

When adding two n-bit binary numbers $A = A_{n-1}A_{n-2}\dots A_0$ and $B = B_{n-1}B_{n-2}\dots B_0$, the sum bit S_i at the i-th bit position is computed by combining the modulo-2 sum (exclusive OR) of bits A_i and B_i, i.e., $H_i = A_i \oplus B_i$ (XOR), and the carry C_{i-1} computed in the previous bit position:

$$S_i = H_i \oplus C_{i-1}. \tag{5.1}$$

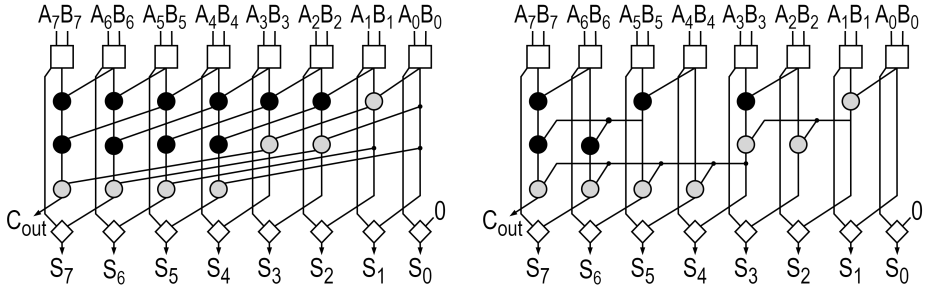
For computing the sum bits of the following bit positions, the incoming carry C_{i-1} needs to propagate to the next position. The carry out of the i-th bit position C_i is computed using the local carry generate and propagate bits $G_i = A_i\, B_i$ (AND) and $P_i = A_i + B_i$ (OR) and the fundamental recursive carry propagation formula

$$C_i = G_i + P_i\, C_{i-1}. \tag{5.2}$$

For short bit widths, ripple-carry adder structures [143] offer compact implementations. For increased bit widths, carry-lookahead adders improve the linear growth of carry chain delay [72]. In this case, carries are computed in parallel and addition is implemented in logarithmic logic depth.

Mapping carry-lookahead adders to parallel-prefix structures maximize their efficiency and increase their placement and wiring regularity [65, 117, 155]. Carry computation is transformed to a prefix problem [9] by using the associative operator $\circ$ which associates pairs of generate and propagate bits as follows:

$$(G, P) \circ (G', P') = (G + P\, G', P\, P').$$



(a) Kogge-Stone (b) Ladner-Fischer

Figure 5.1: The (a) Kogge-Stone [66] and (b) Ladner-Fischer [68] parallel prefix carry-propagate adders.

In a series of consecutive associations of generate and propagate pairs, we denote the group generate and propagate term produces out of bits $k, k-1, \ldots, j$ as $(G_{k:j}, P_{k:j})$,

$$(G_{k:j}, P_{k:j}) = (G_k, P_k) \circ (G_{k-1}, P_{k-1}) \circ \ldots \circ (G_j, P_j), \tag{5.3}$$

and carry C_i corresponds to $G_{i:0}$. This condition for carry C_i makes the adder *accurate*.

Fig. 5.1 depicts two parallel-prefix carry-propagate adders and Fig. 5.2 highlights the logic-level implementation of their basic blocks. Each adder is organized in three stages. The pre-processing stage computes bits G_i, P_i and H_i. Parallel prefix carry computation is done in the second stage using $\log_2 n$ prefix levels. The last node of each bit column requires a simpler implementation of the $\circ$ operator since only a group generate term $G_{i:0}$ needs to be computed. The last stage computes the sum bits according to (5.1).

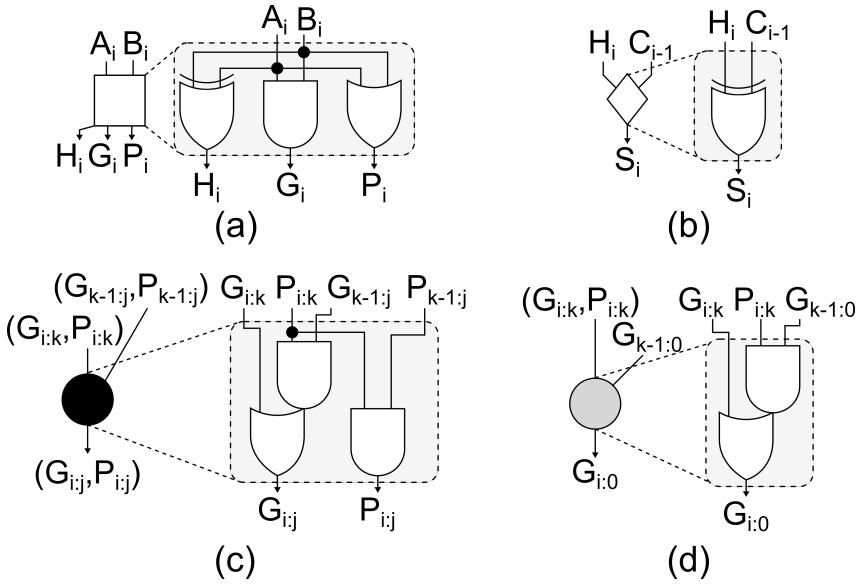


Figure 5.2: The blocks used in the (a) first and (b) in the last stage of a parallel-prefix adder. (c) The logic-level implementation of the prefix operator $\circ$. (d) The simplified operator $\circ$ used in the last node of each column.

Parallel prefix adders can take many forms depending on the number of prefix levels

used and the internal fanout allowed. This affects directly the number of operators needed to complete accurate carry computation. By allowing more prefix levels than the minimum possible, offers more freedom that can be translated to reduced number of operators [65, 117, 155]. For large input wordlengths sparse parallel prefix adders are preferred, since the wiring and area of the design are significantly reduced without sacrificing delay [5]. Parallel prefix formulation of addition has been expanded to other forms of carries, such as Ling carries [27], and even to sum-propagate adders [28].

In parallel prefix adders, approximation can take many different forms, such as replacing some complex logic gates with simpler ones, removing operators from the lower significance bits [23], pruning entire logic levels [31], or even performing transistor-level pruning [101, 149]. For the purpose of this study, approximate carry computation allows parallel prefix adders to use group generate terms of shorter length, such as $G_{i:m}$, with $m > 0$, instead of the full group generate term $G_{i:0}$.

Following [31], approximate parallel prefix adders can be designed by removing prefix levels from equal-bit-width accurate parallel prefix adders and modifying the remaining ones. The choice of prefix level removal is carried out based on the desired minimum and maximum carry chain lengths, as well as the length of each approximate carry propagation segment. For example, an adder resulting from the pruning of levels of a Han Carlson adder is shown in Figure 5.3a, with minimum and maximum carry chains of 8 and 9 bits, respectively. This approximate 16-bit adder requires fewer $\circ$ operators than an accurate adder.

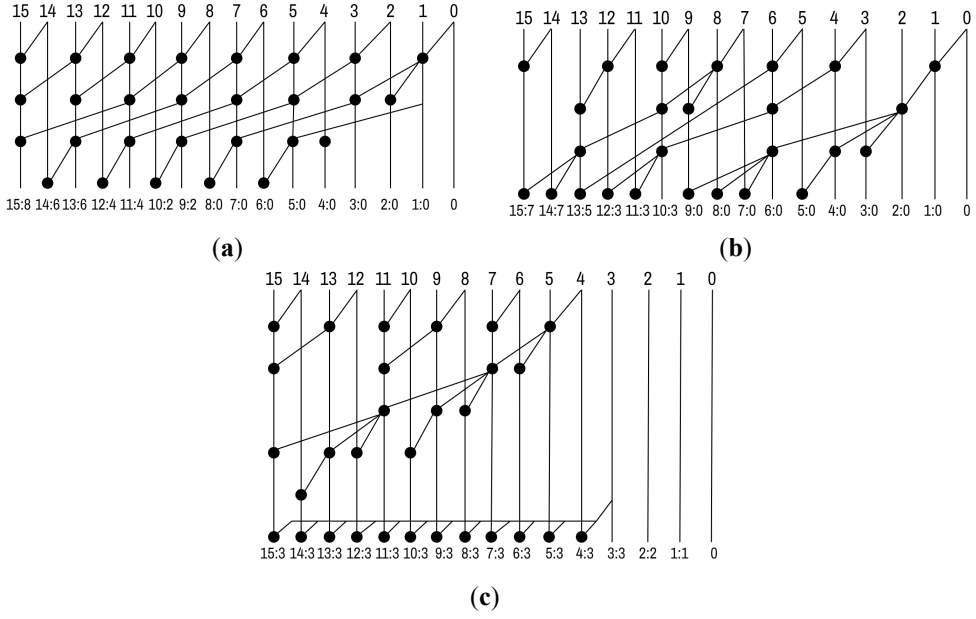


Figure 5.3: Examples of state-of-the-art approximate parallel prefix adders.(**a**) Approximate adder proposed in [31]. (**b**) Approximate adder proposed in [133]. (**c**) Approximate adder proposed in [23].

The work in [133] enumerates all possible approximate parallel prefix adders given a set of input design constraints, such as the desired logic level and the maximum fan-out, as well as the minimum allowed carry chain for all output bits. For instance, the adder in Figure 5.3b can be generated using the following constraints: 4 maximum prefix levels, a maximum

fan-out of 3 and a minimum carry chain of 8 bit positions.

In [23], the designers split the adder into two parts, as can be seen in Figure 5.3c. In the lower-significance part, all operators are removed. The higher-significance part is implemented as a Ladner–Fischer adder, where all carry chains look back to the bit where the splitting happened. There is a final row of operators that extend the carry chain of the higher-significance part by 1. In the example of Figure 5.3c, the adder is split on bit 4.

The method presented in [133] serves as a versatile tool capable of generating any required approximate parallel prefix structure. In this study, we leverage the approximate adder generator from [133] to create custom-designed approximate parallel prefix adders tailored to the specific demands of our applications. This approach departs from the abstract hardware-centric constraints, such as prefix levels and fan-out, employed in [133], which may not be tightly correlated with actual application performance.

5.3 Generic architecture for approximate parallel prefix adders

To be able to automatically synthesize the approximate parallel prefix adders that best fit the application domains, we need to first define a unified representation for the carry computation approximation implemented by the adder. The representation should be versatile enough to cover a wide range of approximations and simple enough so that it can be used in an RL-based optimization framework. Actually, the chosen representation for the approximation used in each adder is a mix of the architecture of generic configurable adders [133] and the adders of [31].

5.3.1 Representation of carry computation approximation

The output bits of a parallel prefix adder are separated into $B+1$ segments. The first segment *always* begins from bit position 0 and involves the L least significant bits of the adder. The rest of the B segments split the $N - L$ most significant output bits of the adder into equal size groups of $b = \frac{N-L}{B}$ bits each. When $N - L$ and B are not divided without a remainder, the most significant segment may involve fewer than b bits.

An example of the correspondence of output segments to a 16-bit parallel prefix adder is shown graphically in Figure 5.4. Each adder of Figure 5.4 is split into four segments, i.e., $B = 3$. In both cases, the $L = 7$ least significant bits form the first segment, and the other 9 most significant bits are separated to $B = 3$ segments of $b = 3$ bits each.

The segment of the L least significant bits is called the *accurate part* of the adder, since, in this part, the output carries and, in effect, the sum bits are computed accurately. The rest of the B segments involve approximate carry computation. For the bits that belong to approximate segments, carry propagation does not reach bit position 0 but stops at a higher bit position.

Carry propagation inside the introduced segments may overlap or not. In Figure 5.4a, the four defined segments are fully isolated and carry propagation of one segment begins after the bit position that the previous segment finishes. On the contrary, in Figure 5.4b, carry propagation across segments overlaps. For instance, the carry computed at position 11 that belongs to the second approximate segment goes back to bits of the first approximate segment as well. This feature is controlled by the third variable of the unified approximate representation used in this work and is called *overlap*. For the example shown in Figure 5.4b, overlap = 2. Please note that overlap allows for the extension of the carry propagation of an

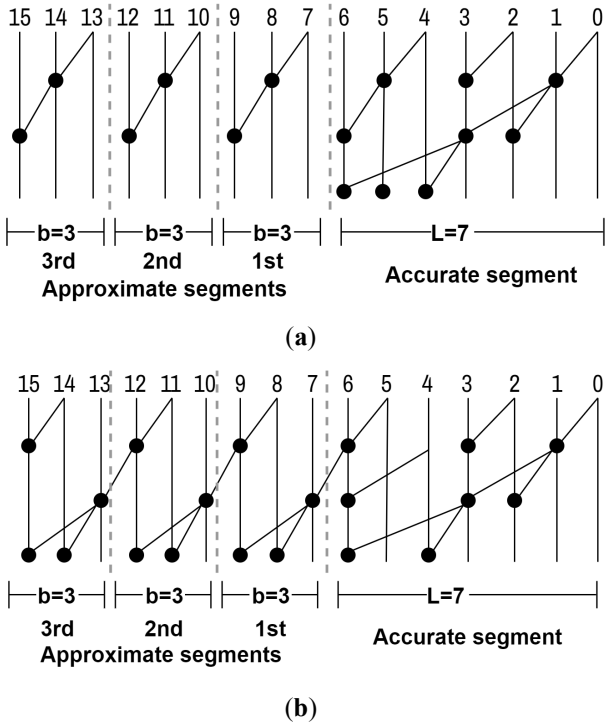


Figure 5.4: Two examples of the proposed generic carry-computation approximation. Both adders have an accurate segment of $L = 7$ bits, and three approximate segments of $B = 3$ bits each. In case (**a**), there is no overlap across approximate segments, whereas in case (**b**), there is an overlap of two bit positions.

approximate segment inside the fully accurate one. The adder of Figure 5.4a exhibits zero overlap in its approximate segments.

Formally, the start s_i and end e_i bit position of the ith approximate segment with $B > i > 0$ and $s_i \geq e_i$ are calculated as follows:

$$s_i = i\,b + L - 1 \tag{5.4}$$

$$e_i = (i-1)b + L \tag{5.5}$$

Carry computation at the ith approximate segment involves the bits from s_i to $e_i - overlap$. This means that each bit in the segment will have a carry chain until the end of its segment, plus the overlap bits.

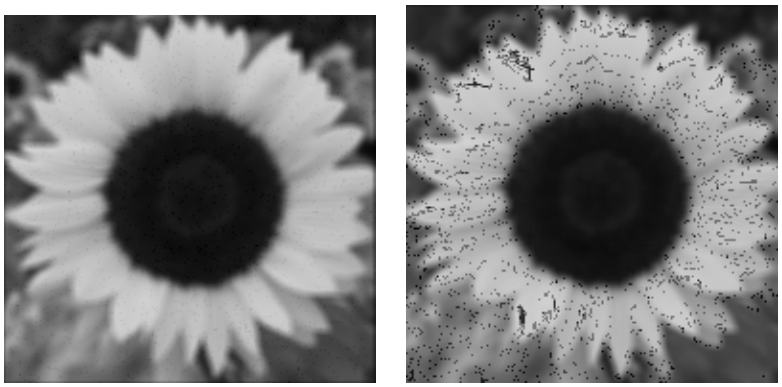
This unified representation covers also already-known approximate adders that are shown in Figure 5.3. The adder of Figure 5.3a is built using the set of constraints $(L = 9, B = 4, overlap = 7)$, while the adder of Figure 5.3b is characterized by the following set of parameters $(L = 10, B = 3, overlap = 7)$. The enumerative approach of [133] can even produce approximate adders with unequal approximate segments or even unequal overlap regions. However, the overall hardware complexity of the derived adders is indistinguishable to the ones produced by the proposed representation that utilizes uniform approximate segments.

5.3.2 Approximation structure and application performance

Deciding which set of parameters $(L, B, overlap)$ satisfies the error quality metrics for each application and still leads to a lower hardware area and power relative to accurate adders is not an easy task.

To give a first example, in Figure 5.5, we compare the application-level performance of two different approximate 16-bit adders for mean filtering. The first adder is built according to the set of parameters $(L = 7, B = 2, overlap = 6)$, while the second uses $(L = 10, B = 1, overlap = 6)$.

As shown in Figure 5.5, the first approximate adder greatly outperforms the second one for mean filtering with respect to peak signal-to-noise ratio (PSNR).



(**a**) PSNR = 41.30 (**b**) PSNR = 22.07

Figure 5.5: The performance of two approximate adder architectures for mean filtering. The adder used in case (**a**) performs much better than the approximate adder used in case (**b**).

However, if we change the application to Laplacian filtering as performed in Figure 5.6, the outcome is reversed. The PSNR of the images after Laplacian filtering shown in Figure 5.6 proves that the second adder is more efficient than the first one.

(**a**) PSNR = 25.73 (**b**) PSNR = 32.97

Figure 5.6: The performance of the same adders used for mean filtering in Laplacian filtering. The second adder that lags in mean filtering performs better in Laplacian filtering as shown by the PSNR of case (**b**) relative to case (**a**).

These two contradictory examples show that customizing approximate addition based on the application characteristics is the only way to yield optimal results. Moreover, it shows the difficulty of predicting this effect: the two adders have the same carry overlap, with the second one having a lengthier accurate segment and one less approximate segment, making it hard to select one of the two as the predominantly best adder.

5.4 Enumeration of approximate parallel prefix adders

The RL agent proposed in this work will decide which set of adder characteristics (L, B, $overlap$) yield the optimal area-error tradeoff for a selected application. To evaluate its decision, a parallel prefix adder with those characteristics will need to be synthesized. For this purpose, the enumeration algorithm we present in [133] is utilized, which extends the approach of [117] to include also approximate solutions that satisfy the given design constraints.

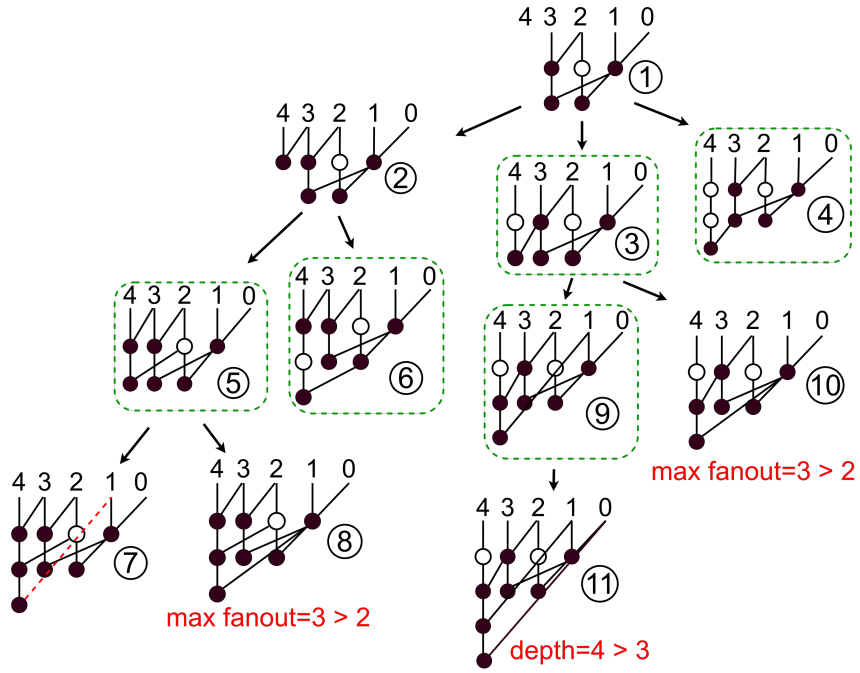


Figure 5.7: Illustrative example of recursively generating 5 bit adders starting from a 4 bit adder. Every solution with an accuracy of $k = 3$ or higher is saved.

Building an approximate parallel-prefix adder of $n + 1$ bits involves adding recursively prefix operators to an adder of n bits only to the most significant column of the $(n + 1)$-bit adder. The n less significant bit positions are kept unmodified. This process is applied by using every possible valid n-bit adder as the starting point for generating $(n+1)$-bit adders.

5.4.1 Designer constraints

The size of the solution space increases rapidly since an n bit adder can lead to multiple $(n + 1)$-bit approximate adders. Therefore, some possible solutions are skipped.

A solution is considered invalid when it violates the maximum allowed number of prefix levels (depth) and maximum fanout set by the designer.

The designer must also specify the approximation target for the adders. This is quantified by a single number k, and means that each carry chain in the adder should be at least k bits long for bit positions larger than k, and fully accurate for every bit position smaller than k. This constraint affects which adders are considered complete and will be passed on to the construction of the next bit width adders. Any adder at least as accurate as the requested accuracy constraint is acceptable, even if it is of higher accuracy than requested.

In split-accuracy configuration the designer sets a minimum carry chain length k separately for each part.

5.4.2 Recursive enumeration of prefix trees

The recursive procedure used for generating approximate parallel prefix adders will be described via the example shown in Fig. 5.7. In this example we arbitrarily start from the 4-bit adder depicted as (1) in Fig. 5.7. The full application of the proposed approach would start from every possible valid 4-bit adder as a root solution. Our goal is to generate all valid 5-bit adders stemming from this initial solution that satisfy our design constraints. Let's assume that the design constraints are: 5-bit adders with a maximum fan-out of 2, maximum depth of 3 and minimum accuracy of $k = 3$. This means that the last operator on column 4 will need to look back at least until bit position 2.

The following steps are executed for each solution:

1. Find the LSB of the latest node inserted in column $n + 1$. The LSB is the least significant bit included in the computation in column $n+1$. If no operators have been added yet, the LSB is equal to $n + 1$.

2. Create the next set of solutions by inserting the new operator in column $n + 1$ and connecting it to the node directly above it and to every possible node in column $LSB-1$.

3. Proceed to the next set of solutions and repeat the process. A solution does not generate new solutions either if it's fully accurate or if it already violates some design constraint.

We execute the above steps for the example of Fig. 5.7, starting with solution (1). Since a 5-bit adder is being constructed, operators will be added only on column 4. Since no operator has been inserted yet, $LSB = 4$ (step 1). For step 2, we need to identify all the valid positions in column $LSB - 1 = 3$ for the new operator to be connected.

The first valid connection is the input of column 3. This connection produces solution (2). The other two solutions connect the new operator to the already existing operators of column 3. The first one is placed in the first prefix level producing the group generate and propagate term 3:2, and the second one in the second prefix level computing the group term 3:0. Connecting the new operator to be added to these positions will generate solutions (3) and (4), respectively. Solution (3) has a minimum accuracy of 3 so it's accepted as a final solution. Solution (4) is fully accurate, so it is also accepted as a final solution. On the other hand solution (2) is a valid solution but doesn't meet the minimum required accuracy, so it is not added to the list of final solutions.

The process is repeated recursively for each new solution. For instance, for solution (2) the LSB is equal to 3 since the last operator on column 4 computes the bits 4:3. Therefore the new operator added should be connected to column 2. The choices are to either connect it to input bit 2, or to the operator that generates the group term 2:0. This expansion of the prefix tree leads to solutions (5) and (6), respectively. Solution (5) has an accuracy of 3 and solution (6) is fully accurate, so they are both accepted as final solutions.

Even though solution (5) is accurate enough to be accepted as a final solution, it is not fully accurate. Therefore any solution stemming from it should still be examined, since they can lead to a better QoR solution later on. In this case, $LSB = 2$, and thus the new operator will be connected either to input bit 1 or to operator that generates group term 1:0 in column 1.

It has been observed that connecting new operators to input bits will potentially grant a worse QoR compared to connecting them to another operator, since the adder topology is used less efficiently. The first new operator added on each column (e.g. the operator added to generate solution 2) is excluded from this, since connecting it to an input bit is the only way to insert it on the first logic level, therefore no inefficiency is observed. On the other hand, connecting to an input bit instead of other operators can help generate solutions with smaller maximum fan-out, in cases where this constraint is tight. Our experiments have shown that connecting an operator to an input bit for two consecutive steps will only grant suboptimal solutions, consuming runtime without generating a high quality adder with small fan-out. Therefore no two consecutive operators should be connected to an input bit. After generating solution (7) by connecting the new operator to input bit 1, it is observed that the previous operator is also connected to an input bit (bit 2). Therefore solution (7) will be rejected for using the topology inefficiently. Solution (8) is also rejected since it violates the maximum fanout constraint on operator 1:0.

By continuing the recursive process, five final solutions are generated: solutions (3), (5) and (9) are approximate, whereas solutions (4) and (6) are fully accurate. Each one of these solutions will be passed on to the algorithm as a root solution (just like solution 1 is for this example) to generate one-bit larger 6-bit adders.

5.4.3 Solution pruning criteria

To avoid excessive runtime, two more pruning techniques are applied. First, we only keep a fixed number of solutions that have the same operator number, maximum level and maximum fanout. This pruning is applied when it is time to accept a solution as final. If there are more than 500 solutions with exactly the same characteristics, the new solution is rejected. This pruning of a valid solution is not expected to ruin the design-space exploration efficiency of the proposed approach since solutions that have the same characteristics most probably already explore a similar part of the design space. For instance, the 7-bit adders (A) and (B) of Fig. 5.8, have exactly the same characteristics in terms of prefix levels, number of operators and maximum fanout. Keeping both of these adders would most probably lead to synthesizing almost-similar adders and also paying an unnecessary runtime overhead. Therefore only a specific number of such adders will be passed on to the next level of the algorithm, e.g., for generating the 8-bit adders using these 7-bit adders as root.

Second, a new valid solution that is much larger than the smallest valid solution already identified for the same bitwidth is automatically rejected. Specifically, if the smallest n-bit solution has s operators, every n-bit solution with more than $s + \Delta$ operators will be

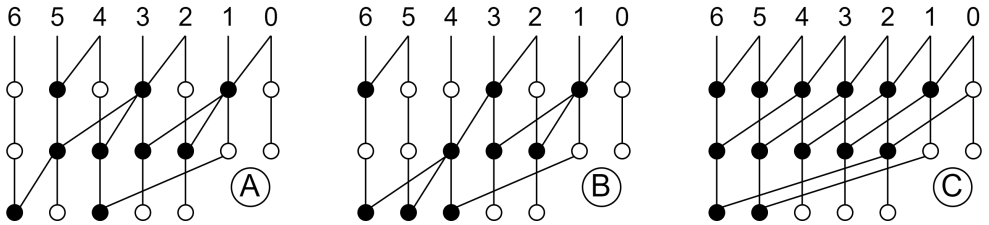


Figure 5.8: Three 7-bit adders synthesized by our method. Adders A and B have similar characteristics and only a certain number of such adders will be kept, whereas adder C is much larger than A and B and can be possibly rejected for tight pruning criteria.

rejected. For example in Fig. 5.8, solutions (A) and (B) have 9 operators, whereas solution (C) has 13 operators. For a value of Δ smaller than 4, solution (C) would have been rejected. High values of Δ can produce a larger set of solutions, e.g., solutions with tight depth and maximum fanout constraints, whereas smaller values of Δ will greatly improve the runtime if the designer selected relaxed design constraints for the approximate adder under design.

We noticed experimentally that Δ should be linearly dependent to the adder's bit width. Therefore, we empirically set $\Delta = 0.7n$ where n is the adder's bit width. For instance, for $n = 16$ bits $\Delta = 11$, and for $n = 32$ bits $\Delta = 22$. This pruning is applied as new solutions get generated: if any new solution (final or intermediate) has more operators than the minimum adder size + Δ for this specific bit width, it is rejected.

5.4.4 Supporting split accuracy

Another broad category of adders that can be synthesized by our method are adders of split accuracy. In this case, the adder is split in two parts that do not share any operator. Also, each part is characterized by its own minimum carry chain length requirement, while both parts share the same constraints with respect to the maximum number of prefix levels and fanout. To support the synthesis of split-accuracy adders, the proposed method does not require any significant modification to its main recursive algorithm.

To be more specific, let's assume that we need to design an n-bit adder that consists of an m-bit low-accuracy part and $(n-m)$-bits high-accuracy part. Each part is characterized by its own minimum carry length requirement. At some recursive step, the algorithm would have generated all valid m-bit adders. Taking such adders as root solutions and extending them by one bit, means that we cross the border between the low-accuracy and the high-accuracy part. When this happens, the process effectively restarts and the new operators added in the high-accuracy part are not allowed to connect to any operator of the lower part. In this way, the solutions derived so far for the lower-accuracy part remain unchanged. Also, for the rest of the enumeration procedure, the minimum carry chain constraint of the high-accuracy part is only taken into account.

5.5 RL framework for the synthesis of approximate parallel prefix adders

Selecting the approximation structure and the way that it will be implemented is not an easy task, since it is application-dependent and small changes in this configuration can lead to a very different quality of results. For this reason, we utilize RL as a way of exploring the state space and finding high-quality solutions.

The designer needs to implement an application in hardware, and for this reason, decides to use approximate parallel prefix adders in their implementation, with the goal of saving area and power without affecting the quality of results of the selected application. To achieve this goal, the designer sets the required constraints such as (a) the bit width of the adder, (b) the application to optimize for and (c) constraints related to the error behavior, as well as (d) the maximum delay, area and maximum fan-out of the desired adder. Then, the RL agent starts training from scratch, with the goal of providing the user with an area-efficient adder that fits those criteria.

The overall flow of the proposed approach is shown in Figure 5.9. The RL agent shown on the left takes actions to select the set of variables $(L, B, overlap)$ that determine the approximation architecture of each adder. As shown in the 'Environment' part of Figure 5.9, for each selected set of parameters $(L, B, overlap)$, the approximate adder generator of [133] is used to generate the corresponding approximate parallel prefix adder. The method of [133] returns multiple adders that satisfy the given parameters. In each step, we select the adder with the smallest number of prefix operators that performs addition to the minimum required prefix levels (i.e., $\log_2 N$). The hardware complexity of the generated approximate parallel prefix adder is quantified after logic synthesis. In parallel, the application-level error performance is also quantified using simulation with a representative set of input data. The resulting area, delay and error are then used to reward the RL agent and drive it toward optimal solutions.

Formally, the RL problem includes a state space S, an action space A and a reward function R, where applying action $a \in A$ on state $s \in S$ will take the environment in the next state s' with a reward of $r = R(s, a)$ [134]. The process is carried out over a set of discrete timesteps $t = 0, 1, 2\ldots$, where, for each timestep, the agent observes s_t and selects a_t based on some action selection policy $\pi(a_t|s_t)$, with each value representing the probability of a_t being chosen for state s_t, until a terminal state is reached. This series of actions from the starting to the ending state is called an episode. The goal of the agent is to learn the policy that maximizes the cumulative future rewards, meaning the sum of the rewards from the starting to the ending state of an episode.

5.5.1 State representation and actions

The state s_t at time step t of each approximate adder is a vector of three integers $s_t = \{L, B, overlap\}$, i.e., the length L of the accurate segment of the adder, the number of approximate segments B and the overlap between the segments $overlap$.

For each state representing an approximate adder, there are six possible actions: increase/decrease L, increase/decrease B and increase/decrease $overlap$. Not all of these actions are valid in each state since some of them can lead to invalid states. For example, if there are zero approximate segments, B cannot be reduced since this would lead to the contradicting state of an approximate adder without approximate segments. In each step,

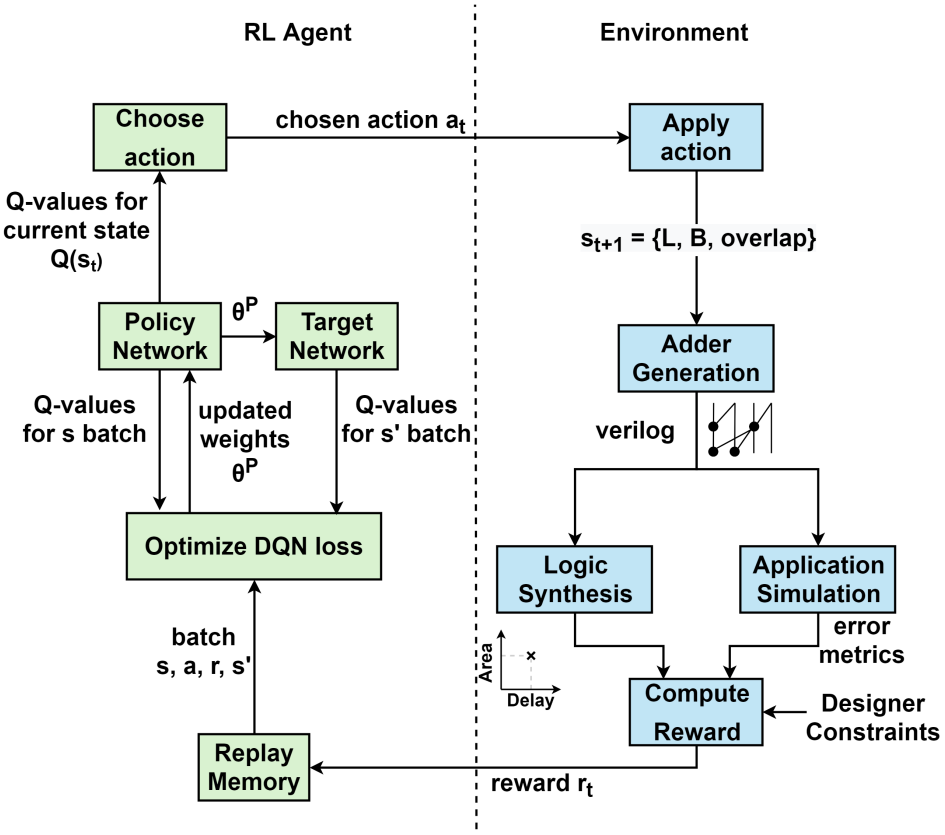


Figure 5.9: RL framework for the customization of approximate prefix adder design. The RL agent chooses an action that modifies the structure of approximation during carry computation. For this structure, an approximate parallel prefix adder is synthesized and its hardware complexity and application performance is quantified. The resulting performance is transformed into a reward, according to the designer's constraints, that drives RL training.

the RL agent chooses a valid action to apply to the current state and move on to the next state.

5.5.2 Reward

After each action is taken and its outcome is evaluated via logic synthesis and simulation as shown in Figure 5.9, the agent must be provided with a reward that should reflect how well the new action performed. This reward should consider how the resulting design compares to the one produced in the previous time step, while also accounting for how closely the design adheres to the original design constraints.

The reward r_t after taking action a_t and leading to a synthesized adder that corresponds to state s_{t+1} is equal to

$$r_t = a\,\Delta_{\text{delay}} + b\,\Delta_{\text{area}} + c\,\Delta_{\text{error}}, \tag{5.6}$$

and its computation is illustrated in Figure 5.10. The differences in hardware delay and area, $\Delta_{\text{delay}} = \text{delay}_{t+1} - \text{delay}_t$ and $\Delta_{\text{area}} = \text{area}_{t+1} - \text{area}_t$, as well as application-level error $\Delta_{\text{error}} = \text{error}_{t+1} - \text{error}_t$, correspond to the differences between the corresponding metric

at the current and the previous timestep. Parameters a, b and c are used to change the focus of the optimization process, as well as to normalize the values to comparable units. In our experiments, we set them to 0.15, 0.05 and 0.8, respectively, prioritizing timing closure over area savings and giving the greatest importance to satisfying the error constraints.

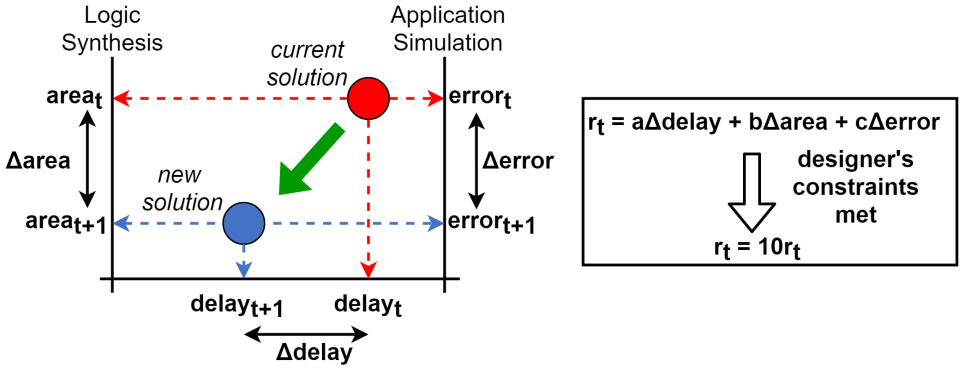


Figure 5.10: Visualization of the reward computation for transitioning from state s_t (current solution) to state s_{t+1} (new solution). After logic synthesis and application simulation is executed, the area, delay and error differences between the two states are used to compute the reward for this time step, which is then multiplied by a constant number if the new state is an adder that satisfies the designer's constraints.

To further ensure that the agent does not try to optimize a metric more than the constraint that the designer has set, the Δ difference of a constraint that is already satisfied is set to 0. For example, once the hardware has reached the delay target set by the designer, there will not be any additional delay reward unless the agent regresses and increases the delay past the designer's target.

Finally, the computed reward r_t is multiplied by a high constant value, in our experiments, set to 10, if an adder that satisfies all the designer's constraints is reached:

$$r_t = 10r_t, \text{ if } s_{t+1} \text{ satisfies all designer's constraints} \tag{5.7}$$

This will assist the agent in avoiding being stuck in a local optima by only taking actions with a high immediate reward.

5.5.3 Deep Q-Network algorithm

In this work, we utilize an efficient and widely used RL algorithm, the deep Q-network (DQN) [98]. The main idea of the DQN algorithm is training a neural network that takes the current state as input and outputs the Q-value of each possible action. By taking the action with the maximum predicted Q-value in each state at the end of training, the agent should be able to reach a high-quality final solution. The basic blocks of a DQN agent can be seen in the left part of Figure 5.9.

The policy network is a neural network that takes a representation of present state s_t as an input and outputs the estimated Q-values for each possible action at this state. The Q-value of a state–action pair quantifies the expected cumulative reward for taking action a_t at state s_t and then taking the action with the highest Q-value until the end of the episode, and it is defined as

$$Q(s_t, a_t) = r_t + \gamma \max_a Q(s_{t+1}, a)), \tag{5.8}$$

where γ is a hyperparameter between 0 and 1 and is used to discount future rewards.

These Q-values are used by the agent when making a decision on the next action. The agent will choose either the action with the maximum Q-value in its current state or a random action, based on a linearly declining variable ϵ, which denotes the probability of choosing a random action.

The structure of the policy network used in this framework is shown in Figure 5.11. It consists of a fully connected neural network layer of size 3×128 that takes as inputs the state variables $\{L, B, overlap\}$, an RELU activation and a second fully connected layer of size 128×6, with the Q-values of the six possible actions.

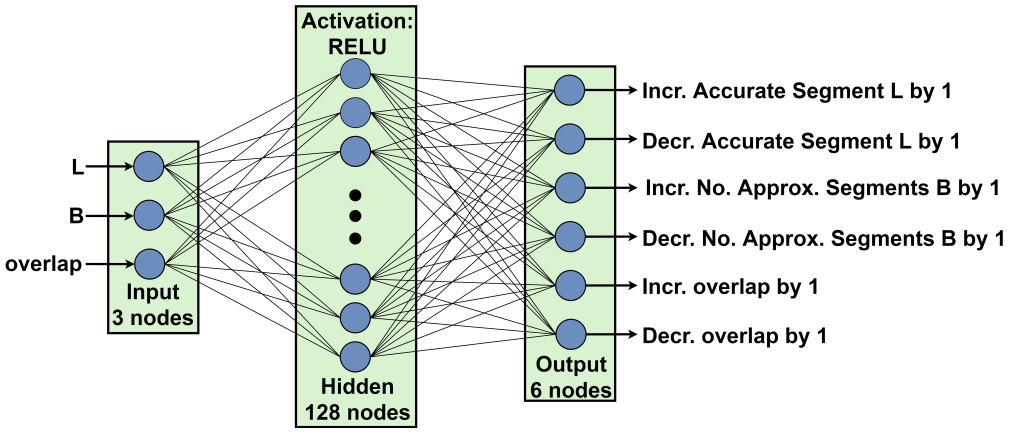


Figure 5.11: Structure of the policy and target network. It consists of an input layer of 3 neurons, a fully connected 3×128 hidden layer using an RELU activation function and a fully connected 128×6 output layer that decides the actions of the next round.

The target network is an identical copy of the policy network but with frozen weights, helping to ensure the stability of the DQN method. As seen in Equation (5.8), both the estimated current Q-value ($Q(s_t, a_t)$) and the estimated target value ($\max_a Q(s_{t+1}, a)$ are computed from the same network. This can lead to instability, since the network would be trying to converge toward a moving target, because updating its weights can potentially change both the current and the target Q-value. To address this, the target Q-value comes from the target network, which provides the policy network with a more fixed target. After a certain number of training steps, the policy network's weights are copied to the target network so that the target is up to date.

The replay memory stores the current state, action taken, next state and reward observed by applying this action to reuse at a later time for learning. This helps to improve the robustness of the algorithm and enhance learning.

5.5.4 Training flow

The approximate adder environment is initialized to a fully accurate N-bit adder $L = N$, $B = 0, overlap = 0$). The exploration ϵ variable is initialized to 1 (meaning that the

agent will choose a random action at the start of the algorithm) and an untrained RL agent is initialized, meaning that the policy and target networks start with random weights. Also, an empty replay memory is assumed at the beginning.

At each step, the chosen action is passed to the environment that applies it, meaning that it will increase or decrease L, B or $overlap$ by 1 based on the chosen action. The new state variables $L, B, overlap$ are then provided to the approximate adder generator that produces the corresponding approximate parallel prefix adder. The adders generated during the calibration of the proposed framework for various applications are stored in a database and reused when necessary to avoid executing the adder generator again for the same examined state. In this way, the adder generator is only applied to new state vectors that have not been explored so far.

The corresponding Verilog netlist is used for measuring the error performance of the application as well as the area(power)–delay characteristics of the adder. The area, delay and error metric values, combined with the designer's constraints, are then used to calculate the reward r_t based on Equation (5.6). If either a maximum number of steps in this episode is reached or if the adder returned satisfies the designer's constraints, the episode ends.

The next step is to optimize the policy network, which is performed by sampling a random batch of fixed size transitions from the replay memory and using it to train the policy network using Equation (5.8). The loss function for a transition of state s, action a, reward r and next state s' sampled by the replay memory can be evaluated as

$$\text{Loss}\left(Q(s,a,\theta^P), r + \gamma \max_{a'} Q(s',a',\theta^T)\right),$$

where θ^P is the weights of the policy network and θ^T is the weights of the target network. After the network is trained, ϵ decreases linearly.

After optimizing, a check is made to see if it is time for the target network to be updated, i.e., if the weights of the main network θ^P should be copied to the target network. Then, the agent receives the next state from the environment and a new iteration can start if the episode has not finished.

At the end of the episode, the average returns (discounted sum of rewards in an episode) of the last 50 episodes are compared against the average returns of the previous 50 episodes. If the average return is not increasing and the agent can synthesize an adder respecting the designer's constraints, the training loop is terminated.

After training finishes, the agent can be used to generate an approximate parallel prefix adder that can satisfy the designer's constraints, or try to meet them as close as possible. When synthesizing a new approximate adder for an already used bit width and application, the weights of the first layer of the already trained neural network can be reused for speeding up the training process. In this work, all of the agents presented are trained from scratch.

5.6 Experimental results

For the evaluation of the proposed approach, we generate approximate parallel prefix adders for four different applications, and compare them to state-of-the-art approximate parallel prefix adders [23, 31, 133]. The adders of [31] include also an error correction mechanism. This is out of the scope of this work, and it is orthogonal to the proposed approach. So in order to ensure a fair comparison we excluded it from the adders we recreated.

The evaluation of the hardware complexity of each adder is carried out by synthesizing the Verilog RTL file describing each adder using Cadence's digital implementation flow and a commercial-grade 40 nm standard-cell library under a mixed V_T configuration of 0.8V. All synthesis runs target a clock period of 1 ns, which is relaxed enough to allow for timing closure for all adders.

To ensure fair comparison, for each application, we choose to compare against the state-of–the-art adder that has the best area–error efficiency. This is decided by carefully reconstructing all relevant state-of-the-art adders and measuring their area and power for a clock period of 1ns. Then, the area ratio of each adder is computed by dividing its hardware area with the area of the largest state-of-the-art adder. Similarly, we measure the error performance of each for all applications under examination. The error ratio of each adder is computed by diving the error of each adder with the largest error observed. In each application, we select the state-of-the-art adder that has the lowest product of area–error ratio. This means that the adders that we compare against may change per application.

For naming conventions, adders from [23] will be named *AxPPA*, followed by the bit on which the adder is split. Adders from [31] are built by approximating an existing adder architecture, so they will be named *Approx* followed by the name of the original adder architecture. Finally, in [133], the adders are built by specifying a minimum carry chain (MCC) for one of the two possible segments. Therefore, we name the adders as *MCC*, followed by the length of the minimum carry chain and the bit in which splitting happens, with 0 indicating that there is no splitting. The proposed adders are named as *RL*, followed by the values of L, B and $overlap$, which describe the adder's approximate carry computation structure.

Table 5.1: Values of the hyperparameters used to train the RL adder generation agent.

Hyperparameter	**Value**
learning rate	4×10^{-5}
max. episode number	5000
max. actions in episode	300
ϵ linear decrease	3×10^{-7}
minimum ϵ value	0.01
γ value for Q-learning equation (see Equation (5.8))	0.95
batch size of experiences sampled from memory for each optimization step	64
number of steps before target network is updated	60

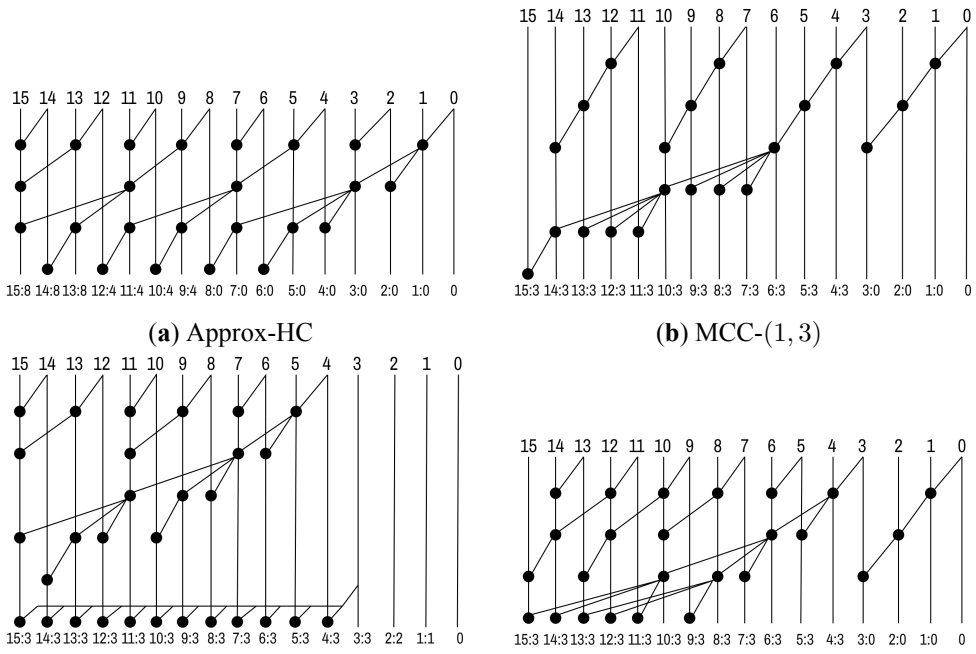
The values of the hyperparameters for training the RL agent are shown in Table 5.1. They were chosen experimentally with the goal of providing a fast convergence without falling in local optima. For each agent trained, we provided a clock period target of 1 ns, an example maximum fan-out of 3 for 16-bit adders and 4 for 32-bit adders and varying area and error constraints based on the application and the adders that we compare against. These maximum fan-out constraints are similar to the ones of the adders presented in [31, 133], and can generally achieve high-quality results.

5.6.1 Image mean filtering

The mean filtering application is used for smoothing 8-bit grayscale images by computing the average value of the pixels in a window around each pixel. In our implementation, an 11×11 window was used, meaning that the mean values of 121 neighbors are computed for each pixel using 16-bit approximate adders. The size of the filter ensures that the whole bit width of a 16-bit adder is used.

To quantify the overall impact of approximate addition to the performance of mean filtering relative to the result taken from a fully-accurate 16-bit adder, we used two well-known metrics: the Peak Signal-to-Noise Ratio (PSNR) and the Mean Structural Similarity (MSSIM) [82]. PSNR quantifies the amount of noise present in an image. PSNR is a logarithmic quantity in the decibel (dB) scale that has low values for noisy images, while its value increases with the reduction of noise. The MSSIM is a metric that quantifies the similarity between two images and its values range from -1 to 1.

The performance of state-of-the-art adders shown in Figure 5.12a–c is depicted in Table 5.2 for five representative images. For each category of state-of-the-art adders, we chose the ones that give the best overall results for PSNR and MMSIM. An example of applying mean filtering to an image using the three state-of-the-art approximate adders shown in Figure 5.12a–c is depicted in Figure 5.13b–d, respectively. It is evident that Approx-HC has many inaccuracies, which can be seen as black pixels in places where they should be absent. The average PSNR and MSSIM of this adder are also much lower than the ones of the other two state-of-the-art adders. MCC-$(1, 3)$ and AxPPA-4 exhibit very similar perfor-



(**a**) Approx-HC (**b**) MCC-$(1, 3)$

(**c**) AxPPA-4 (**d**) RL-$(4, 1, 1)$

Figure 5.12: State-of-the-art approximate parallel prefix adders that give the best performance in each category for mean filtering: (**a**) Approx-HC [31]; (**b**) MCC-$(1, 3)$ [133]; (**c**) AxPPA-4 [23]; and (**d**) the adder RL-$(4, 1, 1)$ synthesized by the proposed RL framework.

mance, both resulting in images that are almost indistinguishable from the reference image. AxPPA-4 has slightly higher average MSSIM but it does not make a noticeable difference in the resulting image.

(**a**) Accurate (**b**) Approx-HC (**c**) MCC-$(1,3)$ (**d**) AxPPA-4 (**e**) RL-$(4,1,1)$

Figure 5.13: Examples of applying mean filtering using different adders. It can be seen that the RL-based approach is able to match the performance of the highly optimized adders from [23, 31].

For the design of the proposed adder, we targeted a PSNR of 38db, essentially aiming to match the performance of MCC-$(1,3)$ and AxPPA-4. For the training of the RL agent, we used one thousand 11×11 image frames randomly extracted from a set of one thousand real images. Training the RL agent on these realistic image frames is enough for handling other images too. The RL agent is trained to optimize for the PSNR. We report also MSSIM as an additional performance metric.

The derived optimized approximated parallel prefix adder that follows the structure RL-$(4,1,1)$ is shown in Figure 5.12d. Its performance in terms of PSNR and MMSIM is shown in the last row of Table 5.2, and an example of its application can be seen in Figure 5.13e. The RL-driven synthesis method is able to match the performance of MCC-$(1,3)$, meaning that it also matches the PSNR of AxPPA-4 while having a slightly inferior MSSIM on average, while greatly outperforming Approx-HC. In the last columns of Table 5.2, we can see that our adder closely matches the area and power consumption of MCC-$(1,3)$ and AxPPA-4. Compared to Approx-HC, our adder improves on the average PSNR by a factor of 1.29 and the average MSSIM by a factor of 3.5, while also saving 8% area and 10% power.

Table 5.2: The performance of the approximate adders under comparison for a mean filtering application relative to a fully accurate result using the peak signal-to-noise ratio (PSNR) and the mean structural similarity (MSSIM) quality metrics.

Adder	**Metric**	**Image**						**Area** (μm^2)	**Power** (nW)
		Apple	**Flower**	**Cherry**	**Horse**	**House**	**Avg**		
Approx-HC	PSNR	17.38	16.92	15.89	15.50	15.29	16.20	93	34
	MSSIM	0.26	0.37	0.10	0.10	0.17	0.20		
MCC-$(1,3)$	PSNR	37.11	37.25	37.11	37.08	36.91	37.09	86	28
	MSSIM	0.95	0.97	0.92	0.91	0.75	0.90		
AxPPA-4	PSNR	37.28	37.44	37.26	37.24	37.06	37.26	86	30
	MSSIM	0.98	0.99	0.95	0.95	0.79	0.93		
RL-$(4,1,1)$	PSNR	37.11	37.25	37.11	37.08	36.91	37.09	86	29
	MSSIM	0.95	0.97	0.92	0.91	0.75	0.90		

For this application, the adder of MCC-$(1,3)$ is highly optimized, since the work of [133] tried different bit positions for splitting their adder and then exhaustively enumerated all

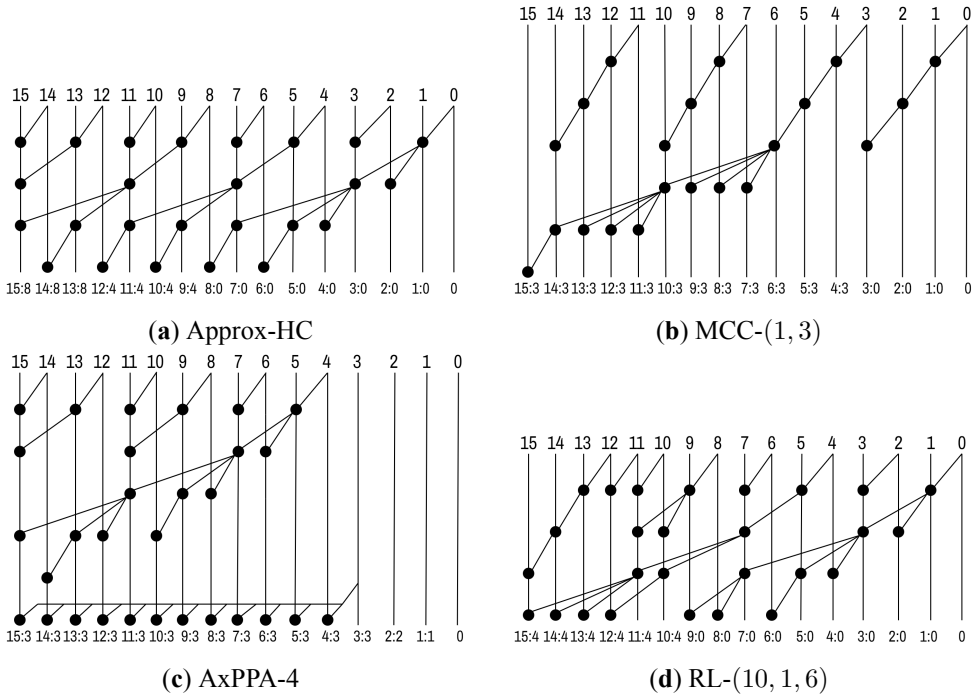
possible resulting split adders on the chosen bit, with the goal of specifically optimizing the performance for mean filtering. AxPPA-4 ends up being equally efficient, since it is similar to MCC-$(1, 3)$, with some structural differences and also the removal of operators in bit positions 1-2, which does not impact the result negatively in this case. Given the highly optimized profile of these two adders, it is expected that the RL agent cannot find a better design. In fact the adder that it returns *automatically without any manual tuning* as performed in [133], RL-$(4, 1, 1)$, has the exact same carry chains as MCC-$(1, 3)$, and consequently very similar area and power behavior. This demonstrates the ability of the RL agent to match highly optimized architectures for applications where such adders exist.

5.6.2 Laplacian filtering

Laplacian filtering, or image edge detection, is applied for detecting edges in 8-bit grayscale images by applying the following 3×3 filter across the image:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Each pixel of the image aligned with the 3×3 window is multiplied by the respective coefficient of the filter, and the derived products are summed together. This final addition is performed by an approximate 16-bit parallel prefix adder.



(**a**) Approx-HC

(**b**) MCC-$(1, 3)$

(**c**) AxPPA-4

(**d**) RL-$(10, 1, 6)$

Figure 5.14: State-of-the-art approximate parallel prefix adders that give the best performance in each category for Laplacian filtering: (**a**) Approx-HC [31]; (**b**) MCC-(1,3) [133]; (**c**) AxPPA-4 [23]; and (**d**) the adder RL-$(10, 1, 6)$ synthesized by the proposed RL framework.

Table 5.3: The performance of the approximate adders under comparison for a Laplacian filtering application relative to a fully accurate result using the peak signal-to-noise ratio (PSNR) and the mean structural similarity (MSSIM) quality metrics.

Adder	Metric	Image					
		Apple	**Flower**	**Cherry**	**Horse**	**House**	**Avg**
Approx-HC	PSNR	31.24	25.78	25.25	23.06	28.96	26.86
	MSSIM	0.37	0.58	0.53	0.66	0.46	0.52
MCC-$(1,3)$	PSNR	34.78	30.86	30.26	28.53	31.32	31.15
	MSSIM	0.79	0.85	0.77	0.86	0.84	0.82
AxPPA-4	PSNR	33.94	31.48	31.74	30.02	32.16	31.87
	MSSIM	0.62	0.80	0.64	0.84	0.71	0.72
RL-$(10,1,6)$	PSNR	38.99	32.97	38.60	30.27	38.12	35.79
	MSSIM	0.85	0.91	0.77	0.91	0.76	0.84

The fundamental difference in this application compared to mean filtering is the presence of a subtraction, which tests if approximate adders work well for signed operands. The error metrics used to quantify the effectiveness of each approximate adder are again PSNR and MSSIM, similarly to the mean filtering. The performance of the state-of-the-art adders is shown in Table 5.3 for the same five representative images and their structure is depicted in Figure 5.14a–c. An example of applying Laplacian filtering to an image using the state-of-the-art adders is shown in Figure 5.15. Approx-LF has many white spots compared to the reference, meaning that it is missing many edges. AxPPA-4 and MCC-$(1,3)$ are closer to the reference, with AxPPA-4 missing edges closer to the top of the image and MCC-$(1,3)$ having an overall better performance for this specific image, but also failing to detect some edges, making the resulting image slightly sparser toward the top and center. Based on the PSNR and MSSIM metrics showed in Table 5.3 for all state-of-the-art adders, it can be seen that, on average, Approx-HC is the most error-prone adder of the three, with AxPPA-4 marginally winning in terms of PSNR and MCC-$(1,3)$ having the highest average MSSIM value.

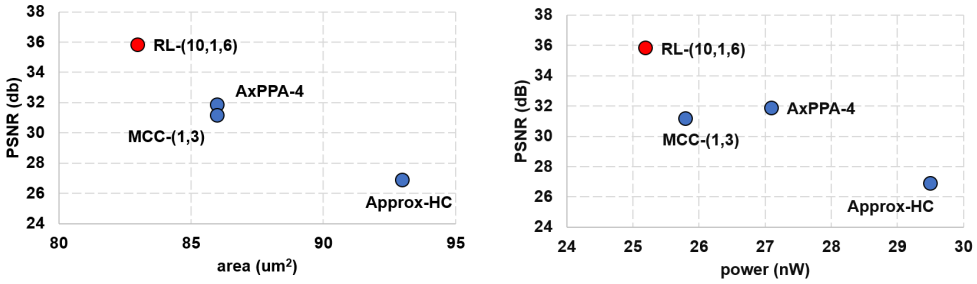
Using the proposed automated approximate adder design framework, we attempted to outperform the most accurate state-of-the-art adder by targeting a PSNR of 35dB. The RL agent is trained on one-thousand random 3×3 frames extracted from real images. The resulting approximate parallel prefix adder after the training of the RL agent is RL-$(10,1,6)$, which is shown in Figure 5.14d, and its performance can be seen in the last row of Table 5.3. The proposed adder outperforms all other state-of-the-art adders for this application, both for the PSNR and the MSSIM error metrics. An example of applying Laplacian filtering to an example image using the RL-generated adder is seen in Figure 5.15e, where the proposed adder produces an image that is nearly identical to the reference image processed by the accurate adder.

The efficiency in terms of PSNR for each adder relative to its hardware complexity is presented in Figure 5.16, where it can be seen that our adder has the highest PSNR compared to all three state-of-the-art adders while having the smallest area and closely matching the power consumption of MCC-$(1,3)$. Our proposed adder RL-$(10,1,6)$ improves the average PSNR by 33% compared to Approx-HC while simultaneously reducing the area and power consumption by 11% and 17%, respectively. Compared to MCC-$(1,3)$ and AxPPA-4, our

(**a**) Accurate (**b**) Approx-HC (**c**) MCC-$(1,3)$ (**d**) AxPPA-4 (**e**) RL-$(10,1,6)$

Figure 5.15: Examples of applying Laplacian filtering using different adders. It can be seen that the RL-based approach is the closest to the reference.

adder improves the average PSNR by 12% while also improving the area by 3% and achieving virtually the same power consumption as MCC-$(1,3)$ and 7% better power compared to AxPPA-4.



(**a**) (**b**)

Figure 5.16: Plot of average PSNR in relation to the (**a**) area and (**b**) power of the approximate adders examined for the application of Laplacian filtering.

To intuitively understand why the proposed adders performed better than the state of the art, we examine the approximation of MCC-$(1,3)$ relative to RL-$(10,1,6)$ in more detail. After applying Laplacian filtering to the examined images, we observe that MCC-$(1,3)$ makes fewer mistakes on average than RL-$(10,1,6)$, erroneously calculating 18% of the output pixels, compared to 22% for RL-$(10,1,6)$. However, the proposed adder better adapts to this specific application, making errors with smaller magnitudes. For example, the average absolute error of RL-$(10,1,6)$ is 4.42, while that of MCC-$(1,3)$ is 9.07. In general, image filtering applications can tolerate multiple small errors more effectively than fewer errors with larger magnitudes, which negatively impact image quality. For a different application where the number of mistakes is more critical than their magnitude, MCC-$(1,3)$ could have been a better choice than RL-$(10,1,6)$, further emphasizing the importance of selecting approximate adders based on the specific requirements of the target application.

5.6.3 Vector inner product

The vector inner product multiplies the corresponding elements of two vectors and then sums the resulting products to a scalar value. For vectors c and x^T, the computation is the following:

$$S = \begin{bmatrix} c_0 & c_1 & \dots & c_n \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix} = c_0x_0 + c_1x_1 + \dots c_nx_n$$

For this example, we assume that vector c contains fixed values and only x^T can assume arbitrary inputs. In our implementation, we chose 16-element vectors, where each element is a 12-bit signed integer. The final result will fit in a 32-bit signed number assuming that multiplication is performed by an accurate multiplier producing full-width products. Approximate addition is again used to perform the addition of products at 32 bits. The goal of this application is to discover adders that can perform well in a general multiply–accumulate scenario, as opposed to the image processing applications where the input is limited to image frames and the filter has specific values that serve this application.

The error metric used for the evaluation of the addition results is the error frequency (EF). It measures how often the adder returns an inaccurate result and is equal to

$$EF = \frac{\text{total inaccurate samples}}{\text{total samples}}$$

An inaccurate sample is a set of inputs for which the adder returns an erroneous sum. The error is computed after the final sum S is calculated, not for every individual summation. It is measured over 50.000 different sets of inputs that are sampled from a uniform random distribution.

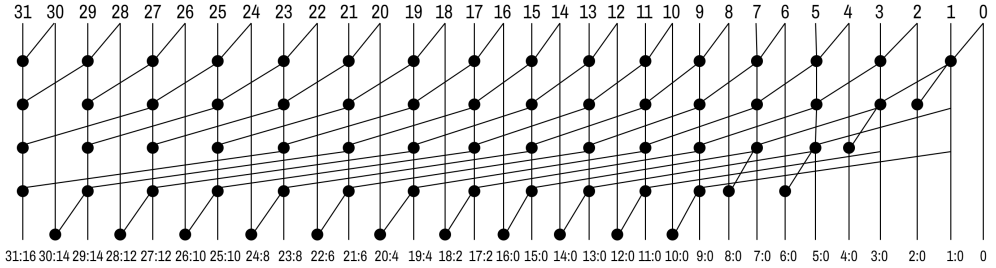
The performance of state-of-the-art adders with the highest EF efficiency for this application is shown in Table 5.4. Their structure is depicted in Figure 5.17. The results show that the AxPPA-2 adder gives the worst performance. The adders from AxPPA [23] make many small mistakes, which can be effective in error-resilient applications such as image processing, but yield low-quality results when an application requires a low frequency of errors, regardless of the magnitude. Approx-HC performs better than AxPPA both in terms of error frequency and hardware complexity. When compared to MCC(21, 0), it offers better hardware and power but at an inferior performance in terms of EF.

Table 5.4: The performance in terms of error frequency and hardware complexity of state-of-the-art adders and two variants generated automatically by the proposed approach for the vector inner product application.

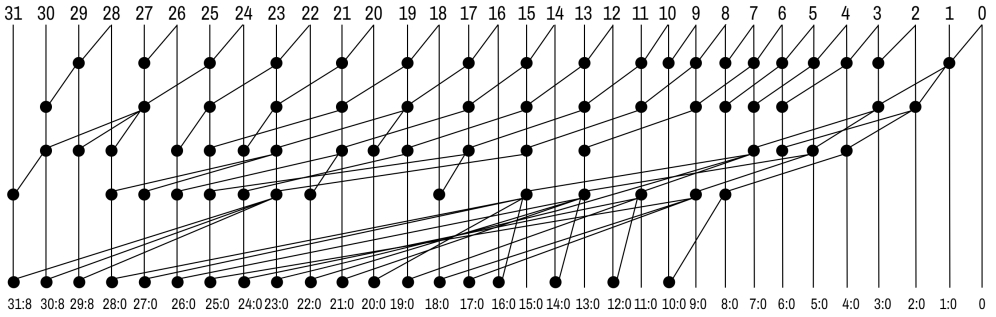
Adder	**#Nodes**	**Area (μm^2)**	**Power (nW)**	**EF**
Approx-HC	72	211	72	0.0109
MCC-$(21, 0)$	85	220	75	0.0002
AxPPA-2	87	242	82	0.6593
RL-$(20, 3, 15)$	70	199	71	0.0055
RL-$(27, 3, 21)$	85	218	73	0.0002

For this application, we designed two RL-based adders. For the first one, the constraint for the RL agent was to achieve an EF of 0.01 and, for the second one, the constraint was tightened to 0.0002, which is 50 times more accurate. During training the error estimation

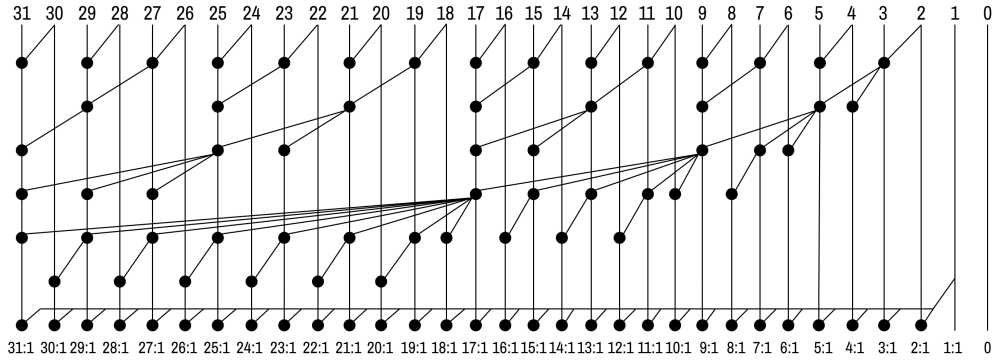
happens over 400 samples, and the values of vector c are kept fixed to ensure that the training does not diverge.



(**a**) Approx-HC



(**b**) MCC-$(21, 0)$



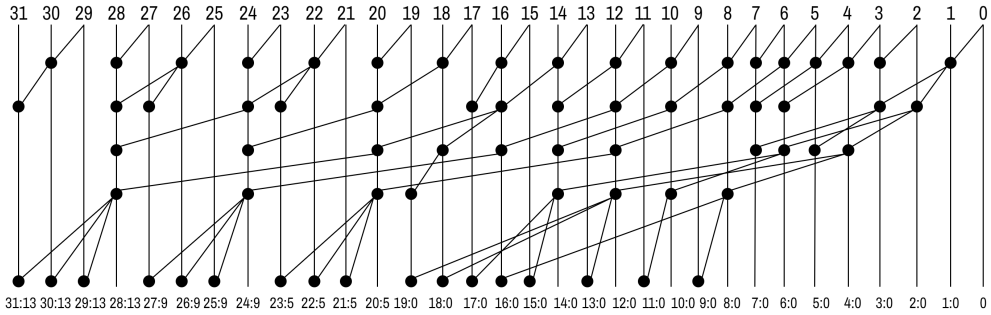
(**c**) AxPPA-2

Figure 5.17: State-of-the-art adders that achieve the lowest EF achievable for their respective family of adders when computing vector inner products.

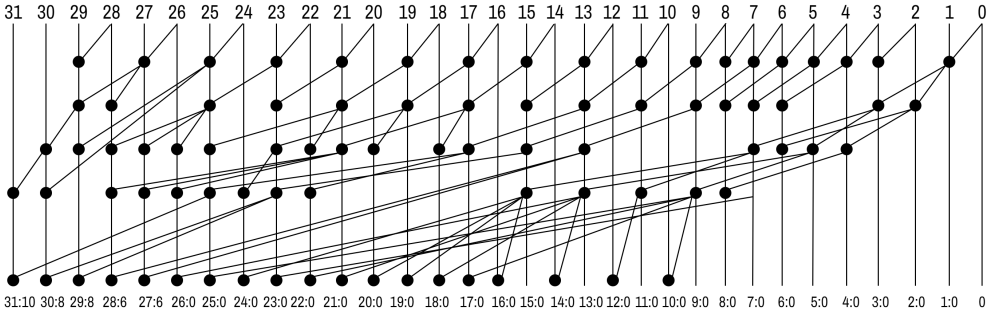
The first RL-generated adder is RL-$(20, 3, 15)$, shown in Figure 5.18a, and the second is RL-$(27, 3, 21)$, shown in Figure 5.18b. Their performance is shown in the last two rows of Table 5.4. RL-$(20, 3, 15)$ should be compared against Approx-HC and AxPPA-2 since it has a higher error frequency in favor of saving area. On the contrary, the second adder RL-$(27, 3, 21)$ targets a much lower EF, so it is compared against MCC-$(21, 0)$.

RL-$(20, 3, 15)$ optimized for a relaxed error target is able to outperform Approx-HC in terms of error behavior, resulting in a 50% lower EF, while also saving 6% area and 1% power. Compared to AxPPA, it saves 18% area and 13% power while significantly decreasing the error frequency. The more accurate adder RL-$(27, 3, 21)$ matches the error frequency of MCC-$(21, 0)$ while improving the area and power by 1% and 3%, respectively. It can be

seen from Figure 5.17b that MCC-$(21, 0)$ is a fully accurate adder up until bit 28, and only introduces approximation in the last 3 bits. This imposes a very tight error constraint on the RL agent, which has very little margin for optimization and for trading off accuracy for improved QoR, making it expected that the resulting adder is not dramatically improving power and area.



(**a**) RL-$(20, 3, 15)$



(**b**) RL-$(27, 3, 21)$

Figure 5.18: RL-generated adders with efficient performance for the vector multiplication application. Adder (**a**) is more area-efficient whereas adder (**b**) achieves a better EF.

5.6.4 Neural network

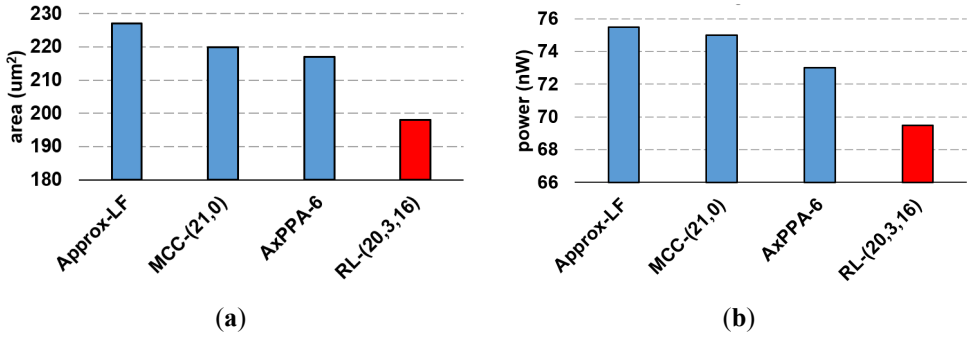
For the final application, we designed a multi-layer perceptron (MLP) consisting of an input layer of 784 neurons, a hidden layer of 500 and an output layer of 10, which has been pre-trained for digit recognition. It was trained with PyTorch 2.0.1 [109] on the MNIST dataset [26] using 50.000 28×28 images, with the goal of minimizing the classification error. The weights of the MLP were quantized in 8-bit values. For the inference stage, we used the remaining 10.000 images, evaluating the classification accuracy using approximate adders. Similarly to the previous application, the multiplication happens using fully accurate multipliers that produce 16-bit products. The approximate adder is a 32-bit wide adder that sums those 16-bit products. When using a fully accurate adder, the reference accuracy is equal to 96.82%.

For this application, we chose the most area-efficient state-of-the-art adders that can achieve a classification error equal to that of the fully accurate adder for comparison. The adders can be seen in Figure 5.20, while their area and power metrics can be compared in Figure 5.19. It can be seen that AxPPA-6 is the most efficient adder for both area and power,

followed by MCC-$(21, 0)$ and Approx-LF.

For the design of the proposed adder, the agent was presented with the constraint of achieving a classification accuracy of 96.82%. During training, inference is executed on 1.000 images instead of the full 10.000 image dataset for a faster accuracy estimation.

The resulting adder is RL-$(20, 3, 16)$, shown in Figure 5.21, which is able to match the classification accuracy of the accurate adder. It can be seen in Figure 5.19 that our adder is able to achieve the accuracy target while lowering both area and power compared to the state-of-the-art adders. Specifically, our approach saves 13% area and 8% power compared to Approx-LF, 10% area and 7% power compared to MCC-$(21, 0)$ and, finally, 9% area and 5% power compared to AxPPA-6.



(**a**) (**b**)

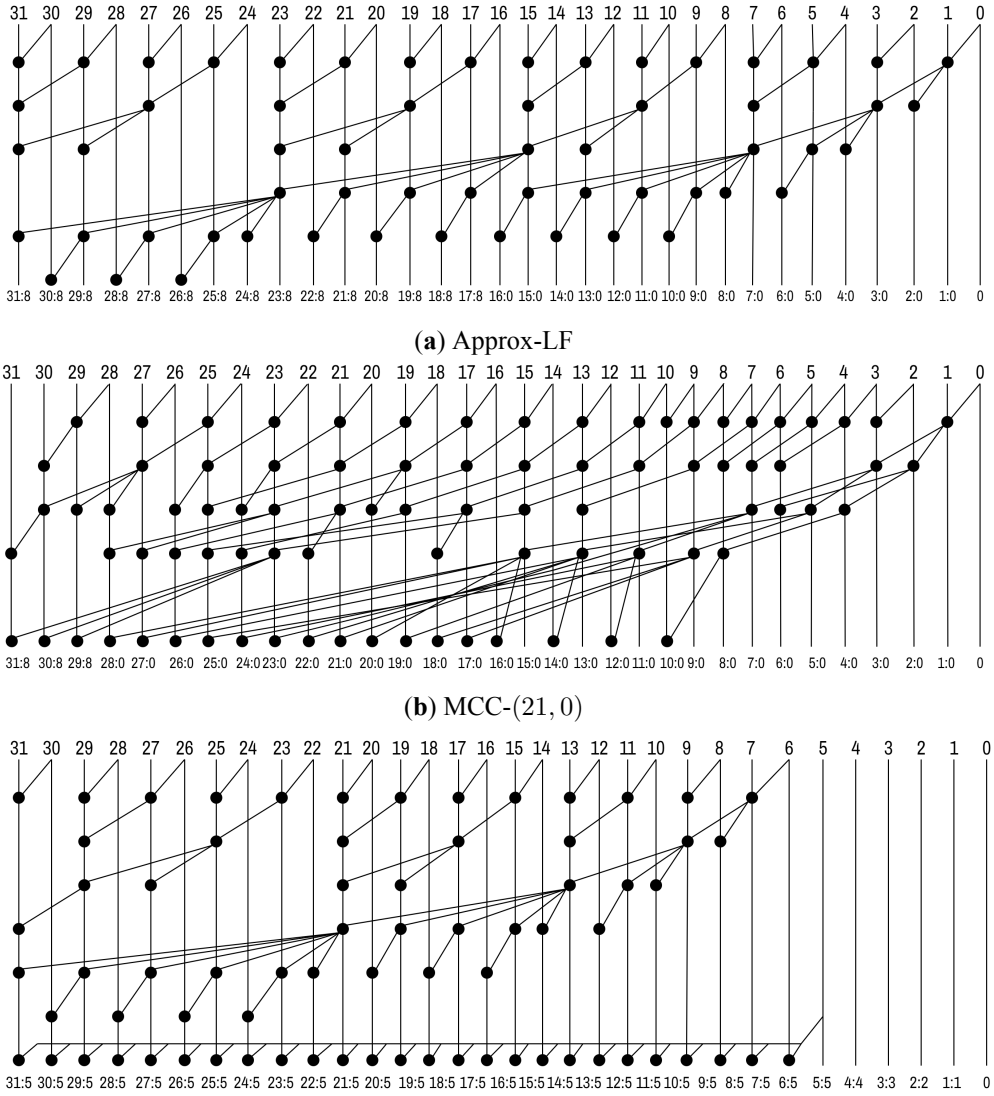
Figure 5.19: (**a**) Area and (**b**) power consumption of all adders under comparison for the case of neural network inference on digit classification.

5.6.5 Runtime analysis

The runtime analysis for the execution of the RL framework for each of the proposed adders is presented in Table 5.5. For each adder, we report the total time taken to train the DQN and the time necessary for adder and logic synthesis, as well as the duration of the application simulation in the training loop. These results are cumulative for running the entire framework, starting with an untrained RL agent. We also report the percentage of state space explored during the execution.

As mentioned in Section 5.5, adders encountered when performing experiments are stored in a database to avoid re-running adder and logic synthesis for them. Roughly 90% of adders encountered by the RL agent during the framework execution for obtaining the proposed adders are retrieved from the database, and the approximate adder generation runtime reported in Table 5.5 results from the generation of the remaining 10% of adders who had not already been visited in previous experiments. Adder simulation is executed when an adder is encountered for the first time in a specific experiment and the error metrics are stored so that, if the adder is visited again in the same experiment, simulation is skipped and its saved error result is retrieved.

The first observation made is that the neural network training time is very short compared to the total runtime of the algorithm due to the simplicity of the structure of the network. This is true for both 16-bit and 32-bit applications since the network structure does not change with a different bit width, making our proposed RL formulation scalable to higher-order adders.



(**a**) Approx-LF

(**b**) MCC-$(21, 0)$

(**c**) AxPPA-6

Figure 5.20: State-of-the-art approximate adders used in the digit classification application. All adders achieve a classification accuracy equal to the one of an accurate adder.

The adder generation time is a big factor of the runtime for the 16-bit applications (mean and Laplacian filtering), and takes up the biggest part of the runtime for the 32-bit applications. More specifically, approximate adder generation lasts for approximately half an hour for 16-bit adders, with the average runtime needed to generate one 16-bit adder being 12 s (the exact value differs based on the L, B and $overlap$ provided to the adder generator). The runtime for approximate adder generation increases to almost a day for the 32-bit applications, with the average time needed to generate a 32-bit adder being close to 8 min.

The time needed for logic synthesis is very short, both for 16- and 32-bit applications. Similarly to adder generation, if an adder is found in the database of previously explored adders, then logic synthesis is not performed, and its area and delay information are re-

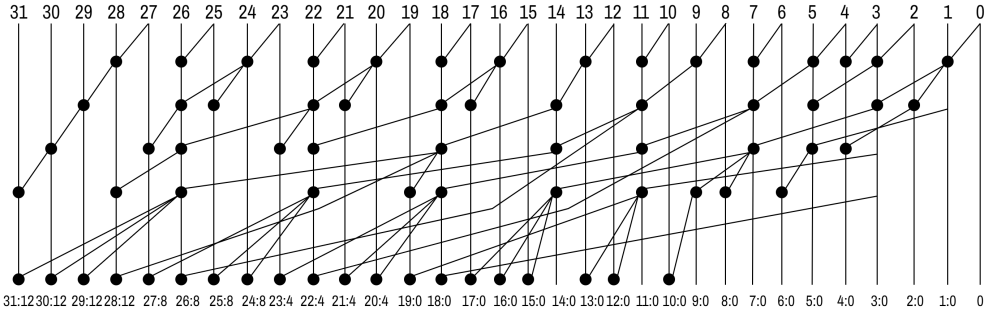


Figure 5.21: The RL-generated adder RL-$(20, 3, 16)$ for the neural network application, achieving a classification accuracy equal to an accurate adder while outperforming the state-of-the-art adders for area and power consumption.

Table 5.5: The total runtime required to execute the reinforcement learning framework and generate each of the proposed adders.

Application	Adder	Explored States (%)	Runtime (mins)				
			Train	Adder Generation	Logic Synthesis	Application Simulation	Total
Mean filtering	RL-$(4, 1, 1)$	84	6.87	32.49	4.92	457.84	502.12
Laplacian filtering	RL-$(10, 1, 6)$	83	8.73	32.04	5.06	33.66	79.49
Inner product	RL-$(20, 3, 15)$	19	6.63	1382.16	20.23	166.21	1575.23
	RL-$(27, 3, 21)$	18	10.37	1312.61	19.96	157.35	1500.29
Neural network	RL-$(20, 3, 16)$	20	8.92	1426.31	21.01	631.28	2087.52

trieved. For 16-bit applications, logic synthesis takes approximately 5 min, and, for 32 bits, it runs for slightly more than 20 min.

The application simulation time depends both on the application and the bit width of the adder. In the case of mean filtering, it takes up the lion's share of the runtime, since, for each adder, we execute mean filtering over 1.000 11×11 frames, resulting in a high number of 16-bit additions. On the other hand, simulating the Laplacian filtering application is much faster since the frame dimensions are lowered to 3×3. The behavior for the 32-bit applications is similar, with the inner product application simulation being relatively fast, while simulating the neural network application takes over 10 h due to the presence of a high number of 32-bit additions.

Overall, it can be observed that the main runtime bottleneck is the approximate adder generation, especially when the bit width increases from 16 to 32 bits. This overhead can be mitigated with a variety of techniques, including extended book-keeping of intermediate solutions and multithreading. Any improvement on the adder generator would be orthogonal to our proposed RL framework, which demonstrates its efficiency by resulting in low training times, showing that it is an efficient method of state space exploration with very small overhead.

By examining the 'Explored states' column, we observe that our framework efficiently

finds high-quality solutions by exploring approximately 85% of the valid state space for 16-bit adders and roughly 20% for 32-bit adders. This efficiency translates to significant runtime savings compared to a greedy approach, which would require exploring the entire state space. While a greedy approach avoids neural network training time, this overhead is negligible compared to the overall runtime. Consequently, our framework achieves a 1.17× speedup for 16-bit adders and a 5× speedup for 32-bit adders.

5.7 Conclusions

Approximate parallel prefix adders provide a versatile solution for designers seeking to balance addition accuracy and quality of results (QoR). However, the multitude of approximation options and varying error tolerances across applications make this a challenging task. This work introduces a Reinforcement Learning (RL) approach that combines a structured state representation with real-world application simulations to train an RL agent to synthesize approximate prefix adders based on designer constraints.

By combining the RL agent with a novel enumeration method for approximate prefix adders, the proposed framework consistently generates architectures that achieve superior or comparable QoR–accuracy tradeoffs to existing methods when evaluated through logic synthesis. The agent learns to exploit application-specific characteristics, enabling the design of adders that reduce area and power with negligible accuracy loss, or that enhance summation quality with minimal overhead.

Future work is planned for extending the RL-based approach to other arithmetic components such as approximate multipliers or combined multiply–add datapaths. Additionally, we plan to investigate the interplay between approximation criteria and how they affect the characteristics of the synthesized parallel prefix adders under Process-Voltage-Temperature (PVT) variations [107].

6 Conclusions

6.1 Summary

This thesis addresses two critical challenges in modern digital design automation: achieving efficient physical design closure and synthesizing customized arithmetic units for application-specific needs. As chip complexity continues to grow, traditional heuristic-based flows in both physical design and datapath synthesis become increasingly inefficient, inflexible, and dependent on manual tuning. This work proposes learning-based solutions to these challenges by introducing Reinforcement Learning (RL) frameworks that intelligently navigate large and complex design spaces. These RL-driven methods are not tied to specific designs, enabling them to adapt dynamically to each unique circuit, improving automation and quality-of-results.

The first core contribution of this thesis lies in the domain of physical design timing optimization. Specifically, we introduce an autonomous flow that integrates Lagrangian Relaxation (LR)-based cost modeling with a Multi-Armed Bandit (MAB) framework to guide the selection of local transformation heuristics. These heuristics include gate sizing, buffer insertion, pin swapping, register resizing, and clock skew scheduling, among others. Over the course of three iterative developments, the optimization flow evolves from a standalone MAB-guided system into a fully cohesive LR-driven methodology, where each transformation step contributes to a unified optimization objective. This synergy allows the MAB model to prioritize the most effective actions at each iteration, leading to better adaptability, improved convergence, and higher overall design quality with reduced runtime effort.

The autonomous timing closure system proposed in this thesis not only automates the heuristic selection process but also generalizes well across a variety of industrial designs. By combining fine-grained control through MAB with a globally coherent LR-based cost function, the framework achieves robust timing improvements without requiring manual intervention or design-specific tuning. This advancement illustrates the benefit of bringing learning-based adaptivity into traditionally deterministic optimization flows, allowing the system to balance multiple competing objectives, such as timing closure, power minimization, and area efficiency, in a context-sensitive manner.

In the second major contribution, the thesis explores the design of application-specific approximate arithmetic units using Deep Reinforcement Learning. In many energy-sensitive applications, approximate computing offers a viable trade-off between computational accuracy and hardware efficiency. This work proposes a novel RL-driven framework for generating customized parallel prefix adders, where the agent learns structural design strategies that are optimized for both hardware metrics (e.g., area and power) and application-level error behavior. Unlike conventional template-based design approaches, our method performs enumerative synthesis based on structural parameters selected by the RL agent, offering a much richer and more flexible design space exploration.

Together, the methods introduced in this thesis demonstrate the power and versatility of RL as a transformative tool in Electronic Design Automation (EDA). By applying RL tech-

niques at different abstraction levels—logic netlist optimization in the physical design flow and approximate datapath synthesis—this work highlights how learning-based models can outperform fixed heuristics and manual strategies in both scope and quality. These contributions pave the way for more intelligent, adaptive, and automated EDA tools that are better equipped to meet the growing complexity and performance demands of next-generation VLSI systems.

6.2 Future work

The main focus of this thesis was to use RL not just to automate, but to intelligently orchestrate timing closure and approximate datapath synthesis, potentially reducing engineering effort and convergence time across design projects. In modern EDA flows, timing closure often involves scheduling thousands of optimizations (e.g., upsizing, cloning, logic restructuring). However, knowing which actions to prioritize and in what order is non-trivial. RL was applied to learn prioritization policies that dynamically schedule optimizations based on current timing reports, path sensitivity, and congestion conditions. The RL agent handled highly dynamic reward structures, where one action can drastically change the landscape for others, and learned to balance local improvements with global design goals like hold margins and power overhead.

In the future we plan to further explore the application of RL in physical synthesis problems. Physical synthesis involves balancing multiple conflicting objectives such as timing, area, power, and routability. Traditional RL struggles with large action spaces and delayed rewards. A future research direction is hierarchical RL, where high-level agents select objectives or constraints (e.g., timing or congestion focus), and low-level agents learn fine-grained actions like buffer insertion or cell sizing. Designing reward functions that reflect tradeoffs between these objectives and training stable policies across hierarchical levels is a significant challenge.

Another promising unexplored direction is the application of RL for incremental timing optimization strategies across design flow stages. Timing violations can emerge and evolve during synthesis, placement, clock tree synthesis, and routing stages. Traditional EDA flows typically address them reactively through local heuristics (e.g., resizing, buffering). A promising direction is using RL to learn incremental timing optimization strategies that span multiple stages. The RL agent could learn when and where to apply specific transformations based on current timing paths, slack distribution, and physical context. The challenge lies in defining intermediate rewards that reflect downstream timing improvements and ensuring that the learned policy remains robust across designs-flow phases.

An additional plan is to perform RL-Guided Technology Mapping for timing and area co-optimization. Technology mapping converts generic logic gates to a target technology library, impacting the downstream physical design heavily. Using RL to guide this step could lead to better outcomes by learning mappings that anticipate their physical synthesis implications. The key challenge is that the mapping's quality is only partially observable until post-layout, requiring RL agents to learn long-term consequences of decisions in early stages, and handle sparse, delayed, and noisy rewards.

Lastly, we should not forget that most current uses of RL in EDA treat the tool as a black box. A future research challenge is co-designing RL algorithms and EDA tools that allow fine-grained feedback, action injection, and real-time observability into internal tool metrics (e.g., congestion heatmaps, timing slack). This requires open APIs and new RL architectures

that can interact with complex non-stationary environments like a full EDA flow, enabling closed-loop learning for optimal design outcomes.

Bibliography

[1] Ziad Abuowaimer, Dani Maarouf, Timothy Martin, Jeremy Foxcroft, Gary Gréwal, Shawki Areibi, and Anthony Vannelli. GPlace3.0: Routability-Driven Analytic Placer for UltraScale FPGA Architectures. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 23(5), October 2018.

[2] Anthony Agnesina, Kyungwook Chang, and Sung Kyu Lim. Parameter Optimization of VLSI Placement Through Deep Reinforcement Learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(4):1295–1308, 2023.

[3] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. OpenTuner: An Extensible Framework for Program Autotuning. In *International Conference on Parallel Architectures and Compilation*, pages 303–316, 2014.

[4] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2-3):235–256, 2002.

[5] Andrew Beaumont-Smith and Cheng-Chew Lim. Parallel Prefix Adder Design. In *Symposium on Computer Arithmetic*, pages 218–225, 2001.

[6] Sayed Aresh Beheshti-Shirazi, Najmeh Nazari, Kevin Immanuel Gubbi, Banafsheh Saber Latibari, Setareh Rafatirad, Houman Homayoun, Avesta Sasan, and P. D. Sai Manoj. Advanced Reinforcement Learning Solution for Clock Skew Engineering: Modified Q-Table Update Technique for Peak Current and IR Drop Minimization. *IEEE Access*, 11:87869–87886, 2023.

[7] Sayed Aresh Beheshti-Shirazi, Ashkan Vakil, Sai Manoj, Ioannis Savidis, Houman Homayoun, and Avesta Sasan. A Reinforced Learning Solution for Clock Skew Engineering to Reduce Peak Current and IR Drop. In *Great Lakes Symposium on VLSI*, page 181–187, 2021.

[8] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum Learning. In *Annual International Conference on Machine Learning*, page 41–48, 2009.

[9] Richard P. Brent and H. T. Kung. A Regular Layout for Parallel Adders. *IEEE Transactions on Computers*, C-31(3):260–264, Mar. 1982.

[10] Yuting Cai, Yue Wu, Xiaoyan Yang, and Zhufei Chu. A Logic Optimization Method Using Reinforcement Learning. In *International Symposium of Electronics Design Automation (ISEDA)*, pages 312–317, 2024.

[11] Fu-Chieh Chang, Yu-Wei Tseng, Ya-Wen Yu, Ssu-Rui Lee, Alexandru Cioba, I-Lun Tseng, Da shan Shiu, Jhih-Wei Hsu, Cheng-Yuan Wang, Chien-Yi Yang,

Ren-Chu Wang, Yao-Wen Chang, Tai-Chen Chen, and Tung-Chieh Chen. Flexible multiple-objective reinforcement learning for chip placement. *arXiv preprint arXiv:2204.06407*, 2022.

[12] Chung-Ping Chen, Chris C. N. Chu, and D. F. Wong. Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation. *IEEE Transactions on CAD*, 18(7):1014–1025, 1999.

[13] George Chen, Tsung-Wei Huang, and Jignesh Shah. Design Optimization Contest. In *International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, 2019.

[14] Hao Chen, Kai-Chieh Hsu, Walker J. Turner, Po-Hsuan Wei, Keren Zhu, David Z. Pan, and Haoxing Ren. Reinforcement Learning Guided Detailed Routing for Custom Circuits. In *International Symposium on Physical Design (ISPD)*, page 26–34, 2023.

[15] Po-Yan Chen, Bing-Ting Ke, Tai-Cheng Lee, I-Ching Tsai, Tai-Wei Kung, Li-Yi Lin, En-Cheng Liu, Yun-Chih Chang, Yih-Lang Li, and Mango C.-T. Chao. A Reinforcement Learning Agent for Obstacle-Avoiding Rectilinear Steiner Tree Construction. In *International Symposium on Physical Design (ISPD)*, page 107–115, 2022.

[16] Chung-Kuan Cheng, Andrew B. Kahng, Sayak Kundu, Yucheng Wang, and Zhiang Wang. Assessment of Reinforcement Learning for Macro Placement. In *International Symposium on Physical Design (ISPD)*, page 158–166, 2023.

[17] H-H Cheng, T-W Lin, Y-C Lin, Iris H-R Jiang, and P-Yu Lee. Team iTimer, TAU workshop 2019, 2019.

[18] Ruoyu Cheng and Junchi Yan. On Joint Learning for Solving Placement and Routing in Chip Design. In *International Conference on Neural Information Processing Systems*, 2021.

[19] David G. Chinnery and Kurt Keutzer. Linear Programming for Sizing, Vth and Vdd Assignment. In *International Symposium on Low Power Electronics and Design*, pages 149–154, 2005.

[20] Alessandro Cilardo. A New Speculative Addition Architecture Suitable for Two's Complement Operations. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 664–669, 2009.

[21] Olivier Coudert. Gate Sizing for Constrained Delay/Power/Area Optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 5(4):465–472, 1997.

[22] Zhenghao Cui and Minghua Shen. Automatic Multi-Parameter Tuning for Logic Synthesis with Reinforcement Learning. In *International Symposium of Electronics Design Automation (ISEDA)*, pages 318–323, 2024.

[23] Morgana Macedo Azevedo da Rosa, Guilherme Paim, Patrícia Ucker Leleu da Costa, Eduardo Antonio Ceśar da Costa, Rafael Soares, and Sergio Bampi. AxPPA: Approximate Parallel Prefix Adders. *IEEE Transactions on Very Large Scale Integration Systems*, 31(1):17–28, 2023.

[24] Siad Daboul, Nicolai Hähnle, Stephan Held, and Ulrike Schorr. Provably Fast and Near-Optimum Gate Sizing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(12):3163–3176, 2018.

[25] Davide De Caro, Gennaro Di Meo, Ettore Napoli, Nicola Petra, and Antonio Giuseppe Maria Strollo. A 1.45 GHz All-Digital Spread Spectrum Clock Generator in 65nm CMOS for Synchronization-Free SoC Applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(11):3839–3852, 2020.

[26] Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[27] Giorgos Dimitrakopoulos and Dimitris Nikolos. High-Speed Parallel-Prefix VLSI Ling Adders. *IEEE Transactions on Computers*, 54(2):225–231, 2005.

[28] Giorgos Dimitrakopoulos, Kleanthis Papachatzopoulos, and Vassilis Paliouras. Sum Propagate Adders. *IEEE Transactions on Emerging Topics in Computing*, 9(3):1479–1488, 2021.

[29] Pooria Esmaeili, Timothy Martin, Shawki Areibi, and Gary Grewal. Guiding FPGA Detailed Placement via Reinforcement Learning. In *International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6, 2022.

[30] Darjn Esposito, Davide De Caro, Ettore Napoli, Nicola Petra, and Antonio Giuseppe Maria Strollo. Variable Latency Speculative Han-Carlson Adder. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(5):1353–1361, 2015.

[31] Darjn Esposito, Davide De Caro, and Antonio Giuseppe Maria Strollo. Variable Latency Speculative Parallel Prefix Adders for Unsigned and Signed Operands. *IEEE Transactions on Circuits and Systems I*, 63(8):1200–1209, 2016.

[32] Hamed Fatemi, Andrew B. Kahng, Hyein Lee, Jiajia Li, and Jose Pineda de Gyvez. Enhancing Sensitivity-based Power Reduction for an Industry IC Design Context. *Integration*, 66(C):96–111, May 2019.

[33] Yi Feng and Chao Wang. GOMARL: Global Optimization of Multiplier using Multi-Agent Reinforcement Learning. In *International Symposium of Electronics Design Automation (ISEDA)*, pages 728–733, 2024.

[34] Guilherme Ferreira, Pedro T. L. Pereira, Guilherme Paim, Eduardo Costa, and Sergio Bampi. A Power-Efficient FFT Hardware Architecture Exploiting Approximate Adders. In *Latin America Symposium on Circuits and System*, pages 1–4, 2021.

[35] John P. Fishburn. Clock Skew Optimization. *IEEE Transactions on Computers*, 39(7):945–951, 1990.

[36] John P. Fishburn and Alfred E. Dunlop. TILOS: A Posynomial Programming Approach to Transistor Sizing. In *International Conference on Computer-Aided Design (ICCAD)*, pages 326–328, 1985.

[37] Guilherme Flach, Mateus Fogaça, Jucemar Monteiro, Marcelo Johann, and Ricardo Reis. Rsyn: An Extensible Physical Synthesis Framework. In *International Symposium on Physical Design (ISPD)*, pages 33–40, 2017.

[38] Guilherme Flach, Tiago Reimann, Gracieli Posser, Marcelo Johann, and Ricardo Reis. Effective Method for Simultaneous Gate Sizing and *V*th Assignment Using Lagrangian Relaxation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(4):546–557, 2014.

[39] Upma Gandhi, Erfan Aghaeekiasaraee, Ismail S. K. Bustany, Payam Mousavi, Matthew E. Taylor, and Laleh Behjat. RL-Ripper: A Framework for Global Routing Using Reinforcement Learning and Smart Net Ripping Techniques. In *Great Lakes Symposium on VLSI*, page 197–201, 2023.

[40] Upma Gandhi, Erfan Aghaeekiasaraee, Sahir, Payam Mousavi, Ismail S. K. Bustany, Mathew E. Taylor, and Laleh Behjat. Applying reinforcement learning to learn best net to rip and re-route in global routing. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 29(4), July 2024.

[41] Anna Goldie and Azalia Mirhoseini. Placement Optimization with Deep Reinforcement Learning. In *International Symposium on Physical Design (ISPD)*, page 3–7, 2020.

[42] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *Communications of the ACM*, 63(11):139–144, October 2020.

[43] Winston Haaswijk, Edo Collins, Benoit Seguin, Mathias Soeken, Frédéric Kaplan, Sabine Süsstrunk, and Giovanni De Micheli. Deep Learning for Logic Optimization Algorithms. In *International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2018.

[44] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *International Conference on Neural Information Processing Systems*, page 1025–1035, 2017.

[45] Inhak Han, Daijoon Hyun, and Youngsoo Shin. Buffer Insertion to Remove Hold Violations at Multiple Process Corners. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 232–237, 2016.

[46] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-Learning. In *Conference on Artificial Intelligence*, page 2094–2100, 2016.

[47] Youbiao He, Hebi Li, Tian Jin, and Forrest Sheng Bao. Circuit Routing Using Monte Carlo Tree Search and Deep Reinforcement Learning. In *International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pages 1–5, 2022.

[48] Stephen Held. Gate Sizing for Large Cell-based Designs. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 827–832, 2009.

[49] Yi-Ju Ho and Wai-Kei Mak. Power and Density-Aware Buffer Insertion. In *International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pages 287–290, 2008.

[50] Abdelrahman Hosny, Soheil Hashemi, Mohamed Shalan, and Sherief Reda. DRiLLS: Deep Reinforcement Learning for Logic Synthesis. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 581–586, 2020.

[51] Shiyan Hu, Mahesh Ketkar, and Jiang Hu. Gate Sizing for Cell Library-Based Designs. In *Design Automation Conference (DAC)*, pages 847–852, 2007.

[52] Shiyan Hu, Zhuo Li, and Charles J. Alpert. A Fully Polynomial Time Approximation Scheme for Timing Driven Minimum Cost Buffer Insertion. In *Design Automation Conference (DAC)*, pages 424–429, 2009.

[53] Shih-Hsu Huang, Guan-Yu Jhuo, and Wei-Lun Huang. Minimum Buffer Insertions for Clock Period Minimization. In *International Symposium on Computer, Communication, Control and Automation (3CA)*, pages 426–429, 2010.

[54] Tsung-Wei Huang and Martin D. F. Wong. OpenTimer: A High-performance Timing Analysis Tool. In *International Conference on Computer-Aided Design (ICCAD)*, pages 895–902, 2015.

[55] Bob Sproull Ivan Sutherland and David Harris. *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann, 1999.

[56] Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, and Jong Wook Kim. Q-Learning Algorithms: A Comprehensive Classification and Applications. *IEEE Access*, 7:133653–133667, 2019.

[57] Wenjing Jiang, Vidya A. Chhabria, and Sachin S. Sapatnekar. IR-Aware ECO Timing Optimization Using Reinforcement Learning. In *Symposium on Machine Learning for CAD (MLCAD)*, pages 1–7, 2024.

[58] Yanbin Jiang, Sachin S. Sapatnekar, Cyrus Bamji, and Juho Kim. Interleaving Buffer Insertion and Transistor Sizing into a Single Optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(4):625–633, 1998.

[59] Zixuan Jiang, Ebrahim Songhori, Shen Wang, Anna Goldie, Azalia Mirhoseini, Joe Jiang, Young-Joon Lee, and David Z. Pan. Delving into Macro Placement with Reinforcement Learning. In *Workshop on Machine Learning for CAD (MLCAD)*, pages 1–3, 2021.

[60] Xuhua Ju, Konglin Zhu, Yibo Lin, and Lin Zhang. Asynchronous Multi-Nets Detailed Routing in VLSI using Multi-Agent Reinforcement Learning. In *International Conference on Network Intelligence and Digital Content (IC-NIDC)*, pages 250–254, 2021.

[61] James. Kennedy and Russell. Eberhart. Particle swarm optimization. In *International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[62] Mahesh Ketkar, Kishore Kasamsetty, and Sachin Sapatnekar. Convex Delay Models for Transistor Sizing. In *Design Automation Conference (DAC)*, pages 655–660, 2000.

[63] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, 2017.

[64] Robert Kirby, Kolby Nottingham, Rajarshi Roy, Saad Godil, and Bryan Catanzaro. Guiding Global Placement With Reinforcement Learning. *arXiv preprint arXiv:2109.02631*, 2021.

[65] Simon Knowles. A Family of Adders. In *Symposium on Computer Arithmetic*, pages 277–281, 2001.

[66] Peter M. Kogge and Harold S. Stone. A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations. *IEEE Transactions on Computers*, C-22(8):786–792, Aug. 1973.

[67] Vijay R. Konda and John N. Tsitsiklis. Actor-Critic Agorithms. In *International Conference on Neural Information Processing Systems*, page 1008–1014, 1999.

[68] Richard E. Ladner and Michael J. Fischer. Parallel Prefix Computation. *Journal of the ACM*, 27(4):831–838, October 1980.

[69] Yao Lai, Yao Mu, and Ping Luo. MaskPlace: Fast Chip Placement via Reinforced Visual Representation Learning. In *International Conference on Neural Information Processing Systems*, 2022.

[70] Tor Lattimore and Csaba Szepesvari. Bandit Algorithms, 2018.

[71] Luciano. Lavagno, Igor Markov, Grant Martin, and Louis Scheffer. *Electronic Design Automation for IC Implementation, Circuit Design, and Process Technology*. Taylor and Francis group, 2016.

[72] Brian D. Lee and Vojin G. Oklobdzija. Improved CLA scheme with optimized delay. *Journal of VLSI Signal Processing Systems*, 3(4):265–274, October 1991.

[73] Sung-Yun Lee, Seonghyeon Park, Daeyeon Kim, Minjae Kim, Tuyen P. Le, and Seokhyeong Kang. RL-Legalizer: Reinforcement Learning-based Cell Priority Optimization in Mixed-Height Standard Cell Legalization. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2023.

[74] Haocheng Li, Gengjie Chen, Bentian Jiang, Jingsong Chen, and Evangeline F. Y. Young. Dr. CU 2.0: A Scalable Detailed Routing Framework with Correct-by-Construction Design Rule Satisfaction. In *International Conference on Computer-Aided Design (ICCAD)*, pages 1–7, 2019.

[75] Li Li, Peng Kang, Yinghai Lu, and Hai Zhou. An Efficient Algorithm for Library-based Cell-type Selection in High-performance. In *International Conference on Computer-Aided Design (ICCAD)*, pages 226–232, 2012.

[76] Lin Li, Yici Cai, and Qiang Zhou. Global Routing Under a Congestion-Aware Reinforcement Learning Model. In *International Symposium of Electronics Design Automation (ISEDA)*, pages 213–218, 2023.

[77] Haiguang Liao, Wentai Zhang, Xuliang Dong, Barnabas Poczos, Kenji Shimada, and Levent Burak Kara. A Deep Reinforcement Learning Approach for Global Routing. *arXiv preprint arXiv:1906.08809*, 2019.

[78] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous Control with Deep Reinforcement Learning. *arXiv preprint arXiv:1509.02971*, 2019.

[79] John Lillis, Chung-Kuan Cheng, and Ting-Ting Y. Lin. Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model. *IEEE Journal of Solid-State Circuits*, 31(3):437–447, 1996.

[80] Kuen-Wey Lin, Yeh-Sheng Lin, Yih-Lang Li, and Rung-Bin Lin. A Maze Routing-Based Methodology With Bounded Exploration and Path-Assessed Retracing for Constrained Multilayer Obstacle-Avoiding Rectilinear Steiner Tree Construction. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 23(4), May 2018.

[81] Yibo Lin, Zixuan Jiang, Jiaqi Gu, Wuxi Li, Shounak Dhar, Haoxing Ren, Brucek Khailany, and David Z. Pan. DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40:748–761, 2020.

[82] Cong Liu, Jie Han, and Fabrizio Lombardi. An Analytical Framework for Evaluating the Error Characteristics of Approximate Adders. *IEEE Transactions on Computers*, 64(5):1268–1281, 2014.

[83] I-Min Liu, A. Aziz, D.F. Wong, and Hai Zhou. An Efficient Buffer Insertion Algorithm for Large Networks Based on Lagrangian Relaxation. In *International Conference on Computer Design*, pages 210–215, 1999.

[84] Jinwei Liu, Gengjie Chen, and Evangeline F.Y. Young. REST: Constructing Rectilinear Steiner Minimum Tree via Reinforcement Learning. In *Design Automation Conference (DAC)*, pages 1135–1140, 2021.

[85] Yi-Chen Lu, Wei-Ting Chan, Deyuan Guo, Sudipto Kundu, Vishal Khandelwal, and Sung Kyu Lim. RL-CCD: Concurrent Clock and Data Optimization using Attention-Based Self-Supervised Reinforcement Learning. In *Design Automation Conference (DAC)*, pages 1–6, 2023.

[86] Yi-Chen Lu, Jeehyun Lee, Anthony Agnesina, Kambiz Samadi, and Sung Kyu Lim. GAN-CTS: A Generative Adversarial Framework for Clock Tree Prediction and Optimization. In *International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019.

[87] Yi-Chen Lu, Siddhartha Nath, Vishal Khandelwal, and Sung Kyu Lim. RL-Sizer: VLSI Gate Sizing for Timing Optimization using Deep Reinforcement Learning. In *Design Automation Conference (DAC)*, pages 733–738, 2021.

[88] Chenyang Lv, Ziling Wei, Weikang Qian, Junjie Ye, Chang Feng, and Zhezhi He. GPT-LS: Generative Pre-Trained Transformer with Offline Reinforcement Learning

for Logic Synthesis. In *International Conference on Computer Design (ICCD)*, pages 320–326, 2023.

[89] Chenyang Lv, Boning Zhang, Weikang Qian, and Zhezhi He. ERL-LS: Accelerating Primitive Sequence Generation of Logic Synthesis with Evolutionary Reinforcement Learning. In *International Symposium of Electronics Design Automation (ISEDA)*, pages 672–677, 2024.

[90] Yuzhe Ma, Subhendu Roy, Jin Miao, Jiamin Chen, and Bei Yu. Cross-Layer Optimization for High Speed Adders: A Pareto Driven Machine Learning Approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(12):2298–2311, 2019.

[91] Poonam Magadum and Soma Ghosh. Network Compression for end-to-end trainable neural networks using Approximate Computing. In *International Conference on Computing Communication and Networking Technologies*, pages 1–5, 2021.

[92] Dimitrios Mangiras, Apostolos Stefanidis, Ioannis Seitanidis, Chrysostomos Nicopoulos, and Giorgos Dimitrakopoulos. Timing-driven placement optimization facilitated by timing-compatibility flip-flop clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2835–2848, 2020.

[93] Igor L. Markov. Reevaluating Google's Reinforcement Learning for IC Macro Placement. *Communications of the ACM*, 67(11):60–71, October 2024.

[94] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Anand Babu, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. Chip Placement with Deep Reinforcement Learning. *arXiv preprint arXiv:2004.10746*, 2020.

[95] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim M. Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, Will Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. A Graph Placement Methodology for Fast Chip Design. *Nature*, 594:207 – 212, 2021.

[96] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. *arXiv preprint arXiv:1411.1784*, 2014.

[97] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on International Conference on Machine Learning*, page 1928–1937, 2016.

[98] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602*, 2013.

[99] Hossam Mossalam, Yannis M. Assael, Diederik M. Roijers, and Shimon Whiteson. Multi-Objective Deep Reinforcement Learning. *arXiv preprint arXiv:1610.02707*, 2016.

[100] Kevin E. Murray, Oleg Petelin, Sheng Zhong, Jia Min Wang, Mohamed Eldafrawy, Jean-Philippe Legault, Eugene Sha, Aaron G. Graham, Jean Wu, Matthew J. P. Walker, Hanqing Zeng, Panagiotis Patros, Jason Luu, Kenneth B. Kent, and Vaughn Betz. VTR 8: High-performance CAD and Customizable FPGA Architecture Modelling. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 13(2), June 2020.

[101] Dhani Nanu, P. Roshini, D. Sowkarthiga, and K. S. A. Ameen. Approximate Adder Design Using CPL Logic for Image Compression. *International Journal of Innovative Research and Development*, 3(4), May 2014.

[102] Pradyut Nath, Sumagna Dey, Subhrapratim Nath, Aditya Shankar, Jamuna Kanta Sing, and Subir Kumar Sarkar. VLSI Routing Optimization Using Hybrid PSO Based on Reinforcement Learning. In *VLSI Device Circuit and System (VLSI DCS)*, pages 238–243, 2022.

[103] David Nguyen, Abhijit Davare, Michael Orshansky, David Chinnery, Brandon Thompson, and Kurt Keutzer. Minimization of Dynamic and Static Power Through Joint Assignment of Threshold Voltages and Sizing Optimization. In *International Symposium on Low Power Electronics and Design*, pages 158–163, 2003.

[104] Muhammet Mustafa Ozdal, Chirayu Amin, Andrey Ayupov, Steven Burns, Gustavo Wilke, and Cheng Zhuo. The ISPD-2012 Discrete Cell Sizing Contest and Benchmark Suite. In *International Symposium on Physical Design (ISPD)*, page 161–164, 2012.

[105] Muhammet Mustafa Ozdal, Chirayu Amin, Andrey Ayupov, Steven M. Burns, Gustavo R. Wilke, and Cheng Zhuo. An Improved Benchmark Suite for the ISPD-2013 Discrete Cell Sizing Contest. In *International Symposium on Physical Design (ISPD)*, page 168–170, 2013.

[106] Muhammet Mustafa Ozdal, Steven Burns, and Jiang Hu. Algorithms for Gate Sizing and Device Parameter Selection for High-Performance Designs. *IEEE Transactions on CAD*, 31(10):1558–1571, 2012.

[107] Kleanthis Papachatzopoulos and Vassilis Paliouras. Path-Based Delay Variation Models for Parallel-Prefix Adders. *IEEE Transactions on Emerging Topics in Computing*, 11(3):689–705, 2023.

[108] Ghasem Pasandi, Shahin Nazarian, and Massoud Pedram. Approximate Logic Synthesis: A Reinforcement Learning-Based Technology Mapping Approach. In *International Symposium on Quality Electronic Design (ISQED)*, pages 26–32, 2019.

[109] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, 32, 2019.

[110] Yasasvi V. Peruvemba, Shubham Rai, Kapil Ahuja, and Akash Kumar. RL-Guided Runtime-Constrained Heuristic Exploration for Logic Synthesis. In *International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2021.

[111] Yu Qian, Xuegong Zhou, Hao Zhou, and Lingli Wang. Efficient Reinforcement Learning Framework for Automated Logic Synthesis Exploration. In *International Conference on Field-Programmable Technology (ICFPT)*, pages 1–6, 2022.

[112] Tong Qu, Yibo Lin, Zongqing Lu, Yajuan Su, and Yayi Wei. Asynchronous Reinforcement Learning Framework for Net Order Exploration in Detailed Routing. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1815–1820, 2021.

[113] Sabeeh Abdul Rehman, Zoe Jeffrey, Yichuang Sun, and Oluyomi Simpson. SASHA: A Shift-Add Segmented Hybrid Approximated Multiplier for Image Processing. In *International Conference on Intelligent Systems and Computer Vision*, pages 1–5, 2022.

[114] Tiago J. Reimann, Cliff C.N. Sze, and Ricardo Reis. Cell Selection for High-Performance Designs in an Industrial Design Flow. In *International Symposium on Physical Design (ISPD)*, pages 65–72, 2016.

[115] Haoxing Ren and Matthew Fojtik. Standard Cell Routing with Reinforcement Learning and Genetic Algorithm in Advanced Technology Nodes. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 684–689, 2021.

[116] Rajarshi Roy, Jonathan Raiman, Neel Kant, Ilyas Elkin, Robert Kirby, Michael Siu, Stuart Oberman, Saad Godil, and Bryan Catanzaro. PrefixRL: Optimization of Parallel Prefix Circuits using Deep Reinforcement Learning. In *Design Automation Conference (DAC)*, pages 853–858, 2021.

[117] Subhendu Roy, Mihir Choudhury, Ruchir Puri, and David Z. Pan. Towards Optimal Performance-Area Trade-Off in Adders by Synthesis of Parallel Prefix Structures. In *Design Automation Conference (DAC)*, pages 1–8, 2013.

[118] Subhendu Roy, Derong Liu, Jagmohan Singh, Junhyung Um, and David Z. Pan. OSFA: A New Paradigm of Aging Aware Gate-Sizing for Power/Performance Optimizations Under Multiple Operating Conditions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35:1618–1629, 2016.

[119] Shankar Sadasivam, Zhuo Chen, Jinwon Lee, and Rajeev Jain. Efficient Reinforcement Learning for Automating Human Decision-making in SoC Design. In *Design Automation Conference (DAC)*, pages 1–6, 2018.

[120] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[121] Jeremy Schlachter, Vincent Camus, Krishna V. Palem, and Christian Enz. Design and Applications of Approximate Circuits by Gate-Level Pruning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(5):1694–1702, 2017.

[122] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[123] Ankur Sharma, David Chinnery, and Chris Chu. Lagrangian Relaxation Based Gate Sizing With Clock Skew Scheduling - A Fast and Effective Approach. In *International Symposium on Physical Design (ISPD)*, pages 129–137, 2019.

[124] Ankur Sharma, David Chinnery, Shrirang Dhamdhere, and Chris Chu. Rapid Gate Sizing with Fewer Iterations of Lagrangian Relaxation. In *International Conference on Computer-Aided Design (ICCAD)*, pages 337–343, 2017.

[125] Ankur Sharma, David Chinnery, Tiago Reimann, Sarvesh Bhardwaj, and Chris Chu. Fast Lagrangian Relaxation Based Multi-Threaded Gate Sizing Using Simple Timing Calibrations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(7):1456–1469, 2019.

[126] Gregory Shklover and Ben Emanuel. Simultaneous Clock and Data Gate Sizing Algorithm with Common Global Objective. In *International Symposium on Physical Design (ISPD)*, pages 145–152, 2012.

[127] 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.

[128] Phillip Stanley-Marbell, Armin Alaghi, Michael Carbin, Eva Darulova, Lara Dolecek, Andreas Gerstlauer, Ghayoor Gillani, Djordje Jevdjic, Thierry Moreau, Mattia Cacciotti, Alexandros Daglis, Natalie Enright Jerger, Babak Falsafi, Sasa Misailovic, Adrian Sampson, and Damien Zufferey. Exploiting Errors for Efficiency: A Survey from Circuits to Applications. *ACM Computing Surveys*, 53(3), 2020.

[129] Apostolos Stefanidis and Giorgos Dimitrakopoulos. Reinforcement-Learning-Based Synthesis of Custom Approximate Parallel Prefix Adders. *Journal of Low Power Electronics and Applications*, 14(4), 2024.

[130] Apostolos Stefanidis, Dimitrios Mangiras, Chrysostomos Nicopoulos, David Chinnery, and Giorgos Dimitrakopoulos. Design Optimization by Fine-Grained Interleaving of Local Netlist Transformations in Lagrangian Relaxation. In *International Symposium on Physical Design (ISPD)*, pages 87–94, 2020.

[131] Apostolos Stefanidis, Dimitrios Mangiras, Chrysostomos Nicopoulos, David Chinnery, and Giorgos Dimitrakopoulos. Autonomous Application of Netlist Transformations Inside Lagrangian Relaxation-Based Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(8):1672–1686, 2021.

[132] Apostolos Stefanidis, Dimitrios Mangiras, Chrysostomos Nicopoulos, and Giorgos Dimitrakopoulos. Multi-Armed Bandits for Autonomous Timing-driven Design Optimization. In *International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 17–22, 2019.

[133] Apostolos Stefanidis, Ioanna Zoumpoulidou, Dionysios Filippas, Giorgos Dimitrakopoulos, and Georgios Ch. Sirakoulis. Synthesis of Approximate Parallel-Prefix Adders. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 31(11):1686–1699, 2023.

[134] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[135] Hiran Tennakoon and Carl Sechen. Nonconvex Gate Delay Modeling and Delay Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27:1583–1594, 2008.

[136] Wen-Pin Tu, Chung-Han Chou, Shih-Hsu Huang, Shih-Chieh Chang, Yow-Tyng Nieh, and Chien-Yung Chou. Low-power Timing Closure Methodology for Ultra-low Voltage Designs. In *International Conference on Computer-Aided Design (ICCAD)*, pages 697–704, 2013.

[137] L.P.P.P. van Ginneken. Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay. In *International Symposium on Circuits and Systems (ISCAS)*, volume 2, pages 865–868, 1990.

[138] Dhruv Vashisht, Harshit Rampal, Haiguang Liao, Yang Lu, Devika Shanbhag, Elias Fallon, and Levent Burak Kara. Placement in Integrated Circuits using Cyclic Reinforcement Learning and Simulated Annealing. *arXiv preprint arXiv:2011.07577*, 2020.

[139] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks, 2018.

[140] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. GCN-RL Circuit Designer: Transferable Transistor Sizing with Graph Neural Networks and Reinforcement Learning. In *Design Automation Conference (DAC)*, pages 1–6, 2020.

[141] Xinsheng Wang, Wenpan Liu, and Mingyan Yu. A Distinctive o(mn) Time Algorithm for Optimal Buffer Insertions. In *International Symposium on Quality Electronic Design*, pages 293–297, 2015.

[142] Zheng Wang and Michael O'Boyle. Machine Learning in Compiler Optimization. *Proceedings of the IEEE*, 106(11):1879–1901, 2018.

[143] Neil Weste and David Harris. *CMOS VLSI Design a Circuits and Systems Perspective*. Addison Wesley (3rd Edition), 2010.

[144] John White. *Bandit Algorithms for Website Optimization*. O'Reilly Media, 2012.

[145] Pei-Ci Wu, Martin D. F. Wong, Ivailo Nedelchev, Sarvesh Bhardwaj, and Vidyamani Parkhe. On Timing Closure: Buffer Insertion for Hold-violation Removal. In *Design Automation Conference (DAC)*, pages 1–6, 2014.

[146] Chang Xu, Gai Liu, Ritchie Zhao, Stephen Yang, Guojie Luo, and Zhiru Zhang. A Parallel Bandit-Based Approach for Autotuning FPGA Compilation. In *International Symposium on Field-Programmable Gate Arrays*, pages 157—166, 2017.

[147] Chenghao Yang and Yinshui Xia. Power optimization with Reinforcement Learning in Logic Synthesis. In *International Conference on ASIC (ASICON)*, pages 1–3, 2021.

[148] Chenghao Yang, Yinshui Xia, Zhufei Chu, and Xiaojing Zha. Logic Synthesis Optimization Sequence Tuning Using RL-Based LSTM and Graph Isomorphism Network. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(8):3600–3604, 2022.

[149] Zhixi Yang, Ajaypat Jain, Jinghang Liang, Jie Han, and Fabrizio Lombardi. Approximate XOR/XNOR-based Adders for Inexact Computing. In *International Conference on Nanotechnology (IEEE-NANO)*, pages 690–693, 2013.

[150] Cunxi Yu. FlowTune: Practical Multi-armed Bandits in Boolean Optimization. In *International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2020.

[151] Dengwei Zhao, Shuai Yuan, Yanan Sun, Shikui Tu, and Lei Xu. DeepTH: Chip Placement with Deep Reinforcement Learning Using a Three-Head Policy Network. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–2, 2023.

[152] Guanglei Zhou and Jason H. Anderson. Area-Driven FPGA Logic Synthesis Using Reinforcement Learning. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 159–165, 2023.

[153] Keren Zhu, Mingjie Liu, Hao Chen, Zheng Zhao, and David Z. Pan. Exploring Logic Optimizations with Reinforcement Learning and Graph Convolutional Network. In *Workshop on Machine Learning for CAD (MLCAD)*, pages 145–150, 2020.

[154] Ming Zhu, Han Chen, Jinyao Li, Chunyu Peng, Jun Tang, Jian Wang, and Xinxin Zhang. Multi-Scale Placement via Reinforcement Learning and Simulated Annealing. In *International Conference on Artificial Intelligence, Automation and High Performance Computing*, page 334–339, 2024.

[155] Reto Zimmermann. *Binary Adder Architectures for Cell-Based VLSI and their Synthesis*. PhD thesis, ETHZ, 1998.

[156] Dongsheng Zuo, Yikang Ouyang, and Yuzhe Ma. RL-MUL: Multiplier Design Optimization with Deep Reinforcement Learning. In *Design Automation Conference (DAC)*, pages 1–6, 2023.

[157] Dongsheng Zuo, Jiadong Zhu, Yikang Ouyang, and Yuzhe Ma. RL-MUL 2.0: Multiplier Design Optimization with Parallel Deep Reinforcement Learning and Space Reduction. *ACM Transactions on Design Automation of Electronic Systems*, January 2025.

Ενισχυτική Μάθηση για την Αυτοματοποιημένη Σχεδίαση Ψηφιακών Ολοκληρωμένων Κυκλωμάτων

Απόστολος Στεφανίδης

Εκτεταμένη Περίληψη της Διδακτορικής Διατριβής

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Δημοκρίτειο Πανεπιστήμιο Θράκης

Ξάνθη, 2025

Περίληψη

Η συνεχώς αυξανόμενη πολυπλοκότητα και το μέγεθος των σύγχρονων ολοκληρωμένων κυκλωμάτων απαιτούν νέες μεθοδολογίες οι οποίες θα βελτιστοποιούν την επίδοση των μεθόδων Αυτοματοποιημένης Σχεδίασης Υλικού (Electronic Design Automation, EDA). Οι παραδοσιακές μέθοδοι φυσικής σύνθεσης βασίζονται συχνά σε ευρετικές προσεγγίσεις και σε σταθερές ροές βελτιστοποίησης, οι οποίες είναι εν δυνάμει μη βέλτιστες, και γενικεύονται δύσκολα σε διαφορετικά κυκλώματα. Η πρόσφατη πρόοδος στον τομέα της Μηχανικής Μάθησης, και συγκεκριμένα της Ενισχυτικής Μάθησης (Reinforcement Learning, RL), προσφέρει μια ισχυρή εναλλακτική επιλογή. Η ενισχυτική μάθηση επιτρέπει τη λήψη προσαρμοσμένων αποφάσεων μέσω συνεχούς ανάδρασης, καθιστώντας την ιδανική για την πλοήγηση στους τεράστιους διακριτούς χώρους βελτιστοποίησης που χαρακτηρίζουν τα προβλήματα στο χώρο του EDA. Αυτή η διατριβή εξερευνεί την ενσωμάτωση της ενισχυτικής μάθησης σε δύο κρίσιμα πεδία της ψηφιακής σχεδίασης: στην ικανοποίηση περιορισμών χρονισμού στη φυσική σχεδίαση, και στη σύνθεση προσεγγιστικών αριθμητικών κυκλωμάτων.

Το πρώτο μέρος της διατριβής εστιάζει στο κλείσιμο χρονισμού (ικανοποίηση περιορισμών χρονισμού), μια κεντρική πρόκληση στη φυσική σχεδίαση. Προτείνουμε μια καινοτόμη μεθοδολογία βελτιστοποίησης η οποία συνδυάζει ένα πλαίσιο μοντελοποίησης κόστους βασισμένο στη μέθοδο Lagrangian Relaxation (LR) με έναν αλγόριθμο Multi-Armed Bandit (MAB), ο οποίος καθοδηγεί τη επιλογή των ευρετικών μεθόδων βελτιστοποίησης χρονισμού που θα εφαρμοστούν. Αυτή η υβριδική προσέγγιση επιτρέπει την αυτόνομη εκτέλεση μιας εκτενούς συλλογής μεθόδων βελτιστοποίησης, οι οποίες περιλαμβάνουν την επιλογή μεγέθους πυλών και καταχωρητών, την προσθήκη buffer, την εναλλαγή συνδέσεων στην είσοδο μιας πύλης, την συγχώνευση ή τον διαχωρισμό πυλών, και την ανάθεση χρόνου άφιξης του σήματος ρολογιού. Η κάθε ευρετική μέθοδος επιλέγεται δυναμικά, λαμβάνοντας υπόψη το πλαίσιο στο οποίο θα εφαρμοστεί, με βάση το ιστορικό των επιβραβεύσεων που έλαβε ως προς την βελτίωση που προκάλεσε στον χρονισμό του κυκλώματος, σε σχέση με το υπολογιστικό κόστος. Το αποτέλεσμα είναι μία πλήρως προσαρμοστική και αυτορυθμιζόμενη ροή βελτιστοποίησης χωρίς ανάγκη για χειροκίνητο συντονισμό.

Το δεύτερο μέρος της διατριβής διερευνά τη χρήση της ενισχυτικής μάθησης για το σχεδιασμό προσεγγιστικών αριθμητικών κυκλωμάτων στοχευμένα σε συγκεκριμένες εφαρμογές, με έμφαση στους αθροιστές παράλληλου προθέματος. Η προσεγγιστική αριθμητική γίνεται ολοένα και πιο χρήσιμη σε τομείς όπου η ενεργειακή απόδοση έχει προτεραιότητα έναντι της πλήρους αριθμητικής ακρίβειας. Προτείνουμε μια μέθοδο ενισχυτικής μάθησης η οποία βελτιστοποιεί ταυτόχρονα την αποδοτικότητα του κυκλώματος και την αριθμητική επίδοση σε επίπεδο εφαρμογής. Το σύστημα ενισχυτικής μάθησης εξερευνά τον χώρο λύσεων των προσεγγιστικών αθροιστών λαμβάνοντας ανατροφοδότηση τόσο από μετρικές σύνθεσης υλικού όσο και από τη συμπεριφορά ως προς την αριθμητική ακρίβεια στην εφαρμογή. Σε αντίθεση με προηγούμενες δημοσιεύσεις που βασίζονται σε προκαθορισμένες αρχιτεκτονικές, η μέθοδός μας χρησιμοποιεί μια καινοτόμο τεχνική που παράγει όλες τις έγκυρες λύσεις που ικανοποιούν συγκεκριμένους περιορισμούς. Αυτό επιτρέπει την εξερεύνηση βέλτιστων αρχιτεκτονικών με διάφορα χαρακτηριστικά καθυστέρησης, εμβαδού,

ισχύος και ακρίβειας σε μια ποικιλία πραγματικών εφαρμογών.

Συνοψίζοντας, η παρούσα διατριβή αναδεικνύει την αποτελεσματικότητα της ενισχυτικής μάθησης και της στατιστικής λήψης αποφάσεων για τον επαναπροσδιορισμό των παραδοσιακών μεθοδολογιών EDA. Με την έξυπνα συντονισμένη λήψη αποφάσεων βελτιστοποίησης τόσο στο φυσικό σχεδιασμό, όσο και στη σύνθεση αριθμητικών κυκλωμάτων, οι προτεινόμενες μέθοδοι επιτυγχάνουν ανταγωνιστικά αποτελέσματα, μειώνοντας ταυτόχρονα σημαντικά τον χρόνο χειροκίνητου σχεδιασμού. Αυτές οι συνεισφορές ανοίγουν το δρόμο για πιο αυτόνομα, αποδοτικά και έξυπνα εργαλεία σχεδιασμού σε μελλοντικά συστήματα VLSI.

Λέξεις Κλειδιά: Εργαλεία αυτοματοποιημένης σχεδίασης υλικού, Φυσική σχεδίαση, Ενισχυτική μάθηση, Προσεγγιστική αριθμητική, Αθροιστές παράλληλου προθέματος, Βελτιστοποιήσεις χρονισμού, Σχεδίαση χαμηλής ισχύος

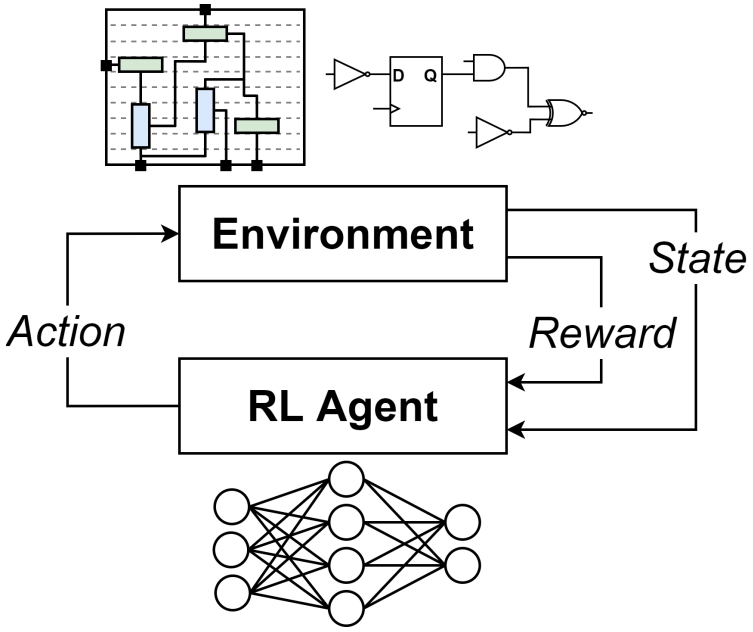
1 Εισαγωγή

Η ροή σχεδίασης ολοκληρωμένων κυκλωμάτων αναφέρεται στη διαδικασία που απαιτείται για την κατασκευή ενός φυσικού κυκλώματος. Τα βήματα που χρειάζονται για τη μετατροπή ενός κυκλώματος από την περιγραφή σε κάποια γλώσσα υλικού σε ένα φυσικά συνθέσιμο κύκλωμα ονομάζονται συνολικά Φυσική Σχεδίαση ή Φυσική Σύνθεση, και στόχος τους είναι η παράδοση ενός κυκλώματος με βελτιστοποιημένο εμβαδόν και κατανάλωση ενέργειας, το οποίο παράλληλα ικανοποιεί τους χρονικούς περιορισμούς του σχεδιαστή. Τα βήματα της σχεδίασης ολοκληρωμένων κυκλωμάτων περιλαμβάνουν την περιγραφή του κυκλώματος σε κάποια γλώσσα υλικού και την επαλήθευση της ορθής λειτουργίας, την σύνθεσης υποκυκλωμάτων όπως αριθμητικές μονάδες, την λογική σύνθεση όπου οι πύλες αντιστοιχίζονται σε κελιά της βιβλιοθήκης που θα χρησιμοποιηθεί, την φυσική σχεδίαση και τέλος, την κατασκευή του κυκλώματος στο εργοστάσιο. Τα κύρια βήματα της φυσικής σχεδίασης είναι η τοποθέτηση των κελιών στο χώρο, η σύνθεση του δένδρου διαμοιρασμού σήματος ρολογιού, η σύνδεση των κελιών μέσω καλωδίων και η χρονική και λειτουργική επαλήθευση του κυκλώματος.

Η περίπλοκη αυτή διαδικασία εισάγει σημαντικές προκλήσεις στη σχεδίαση κυκλωμάτων. Τα σύγχρονα ολοκληρωμένα κυκλώματα διαθέτουν δεκάδες εκατομμύρια κελιά, μια κλίμακα η οποία αυξάνει σε τεράστιο βαθμό τον αριθμό των χρονικών μονοπατιών, επιλογών θέσεων των κελιών, συνδυασμών διασυνδέσεων κ.α., κάνοντας το πρόβλημα βελτιστοποίησης εξαιρετικά περίπλοκο και χρονοβόρο. Επιπλέον, το κύκλωμα πρέπει να ικανοποιεί τους περιορισμούς του σχεδιαστή για πολλαπλές συνθήκες λειτουργίας, οι οποίες πρέπει να λαμβάνονται υπόψη ταυτόχρονα. Η ανεξαρτησία των βημάτων σχεδίασης δυσχεραίνει την λήψη αποφάσεων στα πρώτα βήματα, όπως η λογική σύνθεση, με γνώμονα την επίδρασή τους στα επόμενα βήματα. Τέλος, η ενίσχυση της σημαντικότητας των φυσικών φαινομένων, όπως η RC καθυστέρηση καλωδίων, εισάγει ακόμη περισσότερους παράγοντες που πρέπει να ληφθούν υπόψη, αυξάνοντας περαιτέρω την πολυπλοκότητα.

Η ενισχυτική μάθηση προσφέρει μια υποσχόμενη λύση στις παραπάνω προκλήσεις εξερευνώντας έξυπνες, προσαρμοστικές στρατηγικές βελτιστοποίησης. Ο όρος ενισχυτική μάθηση αναφέρεται σε μια κατηγορία προβλημάτων, και των αλγορίθμων που τα αντιμετωπίζουν, τα οποία έχουν ως κύρια ιδέα την αλληλεπίδραση μεταξύ ενός "πράκτορα" ενισχυτικής μάθησης και του περιβάλλοντος, όπως φαίνεται στο σχήμα 1.1. Ο βασικός στόχος είναι η εκπαίδευση για την επίτευξη ενός στόχου. Αυτό επιτυγχάνεται αποκλειστικά μέσω αλληλεπίδρασης με το περιβάλλον: η παρούσα κατάσταση του περιβάλλοντος παρατηρείται, και μια κίνηση επιλέγεται. Το περιβάλλον εφαρμόζει την κίνηση και επιστρέφει μια επιβράβευση, καθώς και την νέα κατάσταση στην οποία εισέρχεται. Με βάση την επιβράβευση, ο αλγόριθμος ενισχυτικής μάθησης εκπαιδεύεται με σκοπό την μεγιστοποίηση των επιβραβεύσεων που θα λάβει.

Η ιδιότητα των αλγορίθμων ενισχυτικής μάθησης στην εκπαίδευση αποκλειστικά μέσω αλληλεπίδρασης επιτρέπει την συνεχή βελτίωση της πολιτικής λήψης αποφάσεών τους για την βελτιστοποίηση περίπλοκων στόχων όπως η ελαχιστοποίηση καθυστέρησης, εμβαδού και κατανάλωσης ενέργειας σε πολλαπλά είδη κυκλωμάτων. Αυτό το μοντέλο ενισχυτικής μάθησης είναι κατάλληλο για την πλοήγηση του πολυδιάστατου, με πολλαπλούς στόχους



Σχήμα 1.1: Η αλληλεπίδραση με το περιβάλλον, η ποία βρίσκεται στο κέντρο των αλγορίθμων ενισχυτικής μάθησης.

και περιορισμούς χώρου λύσεων της φυσικής σχεδίασης, όπου οι μέθοδοι σχεδιασμένοι από ανθρώπους συχνά αποτυγχάνουν στην γενίκευση. Η ενισχυτική μάθηση μπορεί να εκπαιδευτεί στο να λαμβάνει αποφάσεις σε συγκεκριμένα πλαίσια, όπως επιλογή μεγέθους κελιών, εισαγωγή buffer ή προσαρμογές στις θέσεις κελιών, μοντελοποιώντας τη ροή σύνθεσης ως μια ακολουθιακή διαδικασία λήψης αποφάσεων, βελτιώνοντας τα αποτελέσματα ενώ προσαρμόζεται σε διάφορους περιορισμούς και τεχνολογίες κυκλωμάτων.

Αυτή η διατριβή προωθεί τον τομέα της φυσικής σχεδίασης ολοκληρωμένων κυκλωμάτων προτείνοντας ένα βελτιωμένο πλαίσιο βελτιστοποίησης και κλεισίματος χρονισμού. Η προτεινόμενη προσέγγιση ενσωματώνει ένα σύνολο μεθόδων βελτιστοποίησης, όπως επιλογή μεγέθους πυλών και καταχωρητών, εισαγωγή buffer για επίλυση παραβιάσεων χρονισμού, συγχώνευση/διαχωρισμό πυλών, και επιλογή χρόνου άφιξης σήματος ρολογιού, σε κάθε επανάληψη βελτιστοποίησης βασισμένη σε Lagrangian Relaxation (LR). Ενσωματώνοντας ένα μοντέλο Multi-Armed Bandit (MAB) στη ροή, η μέθοδος επιλέγει σε κάθε επανάληψη τον πιο ωφέλιμο αλγόριθμο βελτιστοποίησης. Ο συνδυασμός λήψης αποφάσεων βασισμένο σε μάθηση με κλασικές μεθόδους βελτιστοποίησης επιτρέπει την αυτόνομη εκτέλεση της ροής, βελτιώνοντας το αποτέλεσμα τόσο στο χρονισμό, όσο και στο εμβαδόν και την κατανάλωση ενέργειας.

Στον τομέα της προσεγγιστικής αριθμητικής, η διατριβή εισάγει ένα πλαίσιο ενισχυτικής μάθησης για τη σύνθεση προσεγγιστικών αθροιστών παράλληλου προθέματος με βάση την επιθυμητή εφαρμογή. Σε αντίθεση με τις παραδοσιακές μεθόδους οι οποίες στοχεύουν αποκλειστικά σε αρχιτεκτονικά πρότυπα, η μέθοδός μας επιτρέπει σε έναν αλγόριθμο ενισχυτικής μάθησης να μάθει στρατηγικές προσεγγιστικής πρόσθεσης εξερευνώντας τον χώρο λύσεων. Βελτιστοποιεί τόσο το εμβαδόν και την κατανάλωση ενέργειας, όσο και το σφάλμα στην επιθυμητή εφαρμογή μέσω ανατροφοδότησης από εργαλεία σύνθεσης υλικού και μετρικές απόδοσης σε επίπεδο εφαρμογής.

2 Ταυτόχρονες βελτιστοποιήσεις αναδιάρθρωσης λογικής για κλείσιμο χρονισμού

Οι βελτιστοποιήσεις χρονισμού κυκλωμάτων σε πολλαπλές συνθήκες λειτουργίας στοχεύει στην ικανοποίηση χρονικών περιορισμών σε όλες τις συνθήκες, και παράλληλα στη μείωση του εμβαδού και της κατανάλωσης ενέργειας του κυκλώματος. Η δυσκολία επίτευξης αυτού του στόχου είναι υψηλή, καθώς μια αλλαγή στο κύκλωμα που μπορεί να λύσει ένα πρόβλημα χρονισμού σε μία συνθήκη λειτουργίας μπορεί να δημιουργήσει πρόβλημα σε μία άλλη. Αυτό το πολύπλευρο πρόβλημα βελτιστοποίησης είναι υπολογιστικά απαιτητικό, ειδικά όταν λαμβάνονται υπόψη πολλαπλοί συνδυασμοί συνθηκών λειτουργίας. Τα προηγούμενα χρόνια, παρουσιάστηκαν διάφορες μέθοδοι βελτίωσης χρονισμού και εμβαδού/ισχύος, όπως η αλλαγή μεγέθους κελιών, η εισαγωγή buffer, η επιλογή χρόνου άφιξης ρολογιού κ.α. Ενώ η κάθε μέθοδος πετυχαίνει ικανοποιητική βελτίωση χρονισμού και ισχύος, ο συνδυασμός πολλαπλών μεθόδων δεν έχει εξερευνηθεί. Σε αυτή την εργασία σχεδιάσαμε την ταυτόχρονη εφαρμογή μεθόδων βελτιστοποίησης κυκλωμάτων σε μια ενιαία ροή.

Για να το πετύχουμε, επεκτείναμε τη γνωστή μέθοδο μοντελοποίησης προβλημάτων βελτιστοποίησης Lagrangian Relaxation (LR) η οποία έχει χρησιμοποιηθεί για επιλογή μεγέθους πυλών, ενσωματώνοντας σε κάθε επανάληψη επιπρόσθετους μετασχηματισμούς κυκλώματος. Ο κάθε μετασχηματισμός βελτιώνει το αποτέλεσμα χωρίς να διαταράσσει τη συνολική διαδικασία βελτιστοποίησης. Αυτό επιτυγχάνεται με τη λήψη τοπικά βέλτιστων αποφάσεων, με βάση τη συνάρτηση κόστους LR. Αυτή η προσέγγιση διευκολύνει και την επέκταση της μεθόδου, καθώς οποιοσδήποτε τοπικός μετασχηματισμός ο οποίος μπορεί να εφαρμοστεί σταδιακά και λαμβάνει αποφάσεις με βάση το κόστος LR μπορεί να προστεθεί στη ροή βελτιστοποίησης.

Η προσέγγιση αυτή εφαρμόστηκε με επιτυχία στα κυκλώματα του διαγωνισμού βελτιστοποίησης κυκλωμάτων TAU2019. Σε κάθε περίπτωση η προσέγγισή μας βελτιστοποιεί τα δεδομένα κυκλώματα με επιτυχία, πετυχαίνοντας μικρότερη περίοδο όπου επιτυγχάνεται το κλείσιμο χρονισμού, ή μειωμένο εμβαδό και κατανάλωση ενέργειας για την ίδια περίοδο που επιτυγχάνει ο νικητής του διαγωνισμού, χωρίς την παρουσία παραβιάσεων μέγιστου χρόνου μετάβασης ή χωρητικότητας.

2.1 Βελτιστοποίηση κυκλωμάτων βασισμένη σε Lagrangian Relaxation

Ο στόχος της μεθόδου αυτής της διατριβής είναι η ελαχιστοποίηση του αθροίσματος κατανάλωσης ισχύος, εμβαδού, και συνολικού αρνητικού slack (TNS), με τέτοιο τρόπο ώστε να πληρούνται οι χρονικοί περιορισμοί. Το TNS συμβολίζει το άθροισμα των χρονικών παραβιάσεων στα τελικά σημεία των χρονικών μονοπατιών.

$$\begin{aligned} \textbf{min}: & \sum_{c\in\text{cells}} P(c) + A(c) - \sum_{j\in\text{POs}} slk_j^L - \sum_{j\in\text{POs}} slk_j^E \\ \textbf{s.t.:}\ & slk_j^L \le 0 \text{ and } slk_j^E \le 0, \forall j \in \text{POs} \\ & slk_j^L \le r_j^L - a_j^L \text{ and } slk_j^E \le a_j^E - r_j^E, \forall j \in \text{POs} \\ & a_i^L + d_{i\to j}^L \le a_j^L \text{ and } a_i^E + d_{i\to j}^E \ge a_j^E, \forall \text{arcs } i \to j \end{aligned} \tag{2.1}$$

τα $P(c)$, $A(c)$ συμβολίζουν την κατανάλωση ενέργειας διαρροής και το εμβαδό του κελιού c, το slk_j συμβολίζει το αρνητικό slack του τελικού σημείου j, τα a_j και r_j είναι οι χρόνοι άφιξης και επιθυμητής άφιξης του pin j; και το $d_{i\to j}$ είναι η καθυστέρηση από το pin i στο j. Η καθυστέρηση αυτή περιλαμβάνει τόσο την καθυστέρηση του κελιού με είσοδο το pin i, όσο και την καθυστέρηση του καλωδίου μέχρι το pin j. Οι δείκτες E και L αναπαριστούν αντίστοιχα τη χρονική πληροφορία για early και late χρονισμό, και τα τελικά σημεία χρονισμού (PO) συμπεριλαμβάνουν τις εισόδους D pin όλων των καταχωρητών και τις εξόδους του κυκλώματος.

Η μέθοδος Lagrangian Relaxation εφαρμόζεται στο παραπάνω πρόβλημα εισάγοντας τους περιορισμούς στη συνάρτηση κόστους πολλαπλασιασμένους με έναν πολλαπλασιαστή Lagrange, με τέτοιο τρόπο ώστε η παραβίαση ενός περιορισμού να οδηγεί σε αύξηση του κόστους. Μετά την εισαγωγή των περιορισμών, την διαφοροποίηση της συνάρτησης κόστους, και την εφαρμογή των συνθηκών βέλτιστης απόδοσης, καταλήγουμε στην τελική συνάρτηση κόστους LR.

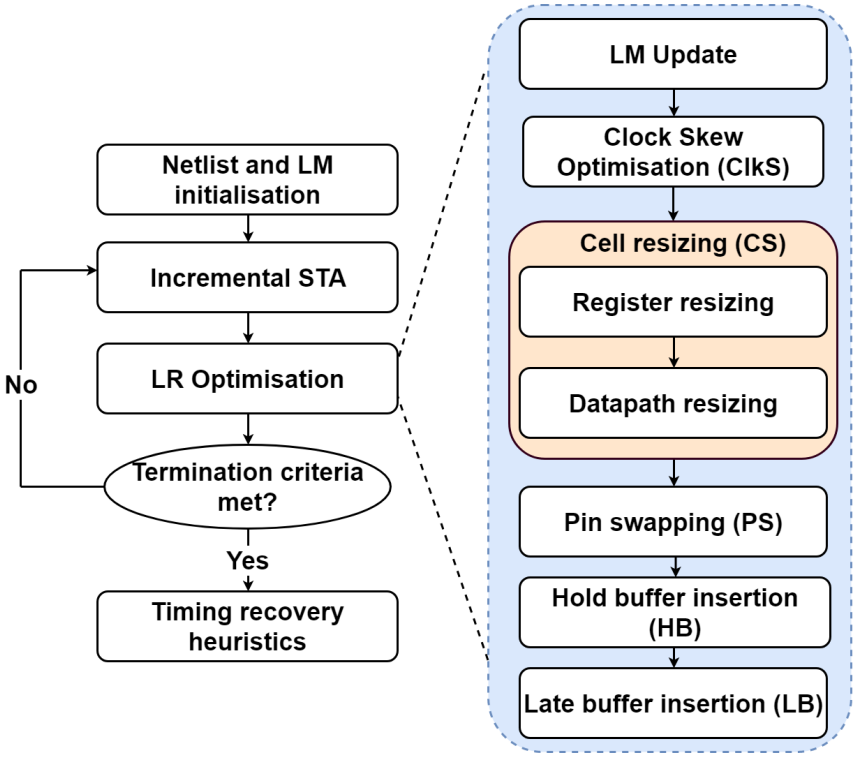
$$\min \sum_{gates} P(c) + A(c) + \sum_{i\to j} \lambda_{i\to j}^L d_{i\to j}^L - \lambda_{i\to j}^E d_{i\to j}^E \tag{2.2}$$

Παρατηρούμε ότι ο κάθε πολλαπλασιαστής Lagrange υπογραμμίζει την κρισιμότητα της εκάστοτε καθυστέρησης. Ένας υψηλός late πολλαπλασιαστής υποδηλώνει ότι η αντίστοιχη καθυστέρηση θα πρέπει να μειωθεί για να ελαχιστοποιηθεί το συνολικό κόστος, ενώ ένας υψηλός early πολλαπλασιαστής θα πρέπει να οδηγήσει σε αύξηση της καθυστέρησης. Μια μικρή τιμή πολλαπλασιαστή δείχνει ότι η συγκεκριμένη καθυστέρηση δεν είναι κρίσιμη για τη βελτιστοποίηση του κυκλώματος. Οι πολλαπλασιαστές Lagrange ανανεώνουν τακτικά τις τιμές τους ώστε να αντικατοπτρίζουν τη χρονική κρισιμότητα του κυκλώματος.

Η προτεινόμενη ροή βελτιστοποίησης παρουσιάζεται στο σχήμα 2.1. Αρχικά, όλα τα κελιά ανατίθενται στο μικρότερο μέγεθος το οποίο δεν οδηγεί σε παραβιάσεις μέγιστου χρόνου μετάβασης ή χωρητικότητας. Οι πολλαπλασιαστές Lagrange αρχικοποιούνται στην τιμή 1, και η επαναληπτική διαδικασία βελτιστοποίησης ξεκινά. Στην αρχή κάθε επανάληψης, ο χρονισμός υπολογίζεται για όλες τις συνθήκες λειτουργίας, ενώ στη διάρκεια της επανάληψης χρησιμοποιούνται μόνο 2 συνθήκες: η πιο κρίσιμη για early και για late χρονισμό.

Η κάθε επανάληψη ξεκινά με την ενημέρωση των πολλαπλασιαστών Lagrange. Έπειτα οι μετασχηματισμοί κυκλώματος εκτελούνται σειριακά, όπως φαίνεται στο σχήμα 2.1. Οι διαθέσιμοι μετασχηματισμοί είναι οι εξής:

- **Ανάθεση χρόνων άφιξης σήματος ρολογιού**: Εισάγει ή αφαιρεί μια σταθερή καθυστέρηση από τις εισόδους ρολογιού των καταχωρητών, με βάση τις τιμές των early και late πολλαπλασιαστών Lagrange στην είσοδο ρολογιού του κάθε καταχωρητή.
- **Επιλογή μεγέθους κελιών**: Επιλέγει την κατάλληλη εκδοχή για τα πιο κρίσιμα κελιά του κυκλώματος. Αφότου δοκιμαστούν όλες οι διαθέσιμες εκδοχές, επιλέγεται αυτή



Σχήμα 2.1: Η προτεινόμενη ροή βελτιστοποίησης.

που επιτυγχάνει το μικρότερο τοπικό κόστος LR, το οποίο περιλαμβάνει το εμβαδόν και την κατανάλωση ενέργειας του κελιού, καθώς και το άθροισμα των γινομένων πολλαπλασιαστών Lagrange και καθυστερήσεων των γειτονικών πυλών.

- **Εναλλαγή καλωδίων**: Δοκιμάζει την εναλλαγή συνδέσεων του καλωδίου με τον πιο κρίσιμο χρονισμό με ένα άλλο, ισοδύναμο καλώδιο στην ίδια πύλη, διατηρώντας την σύνδεση με το μικρότερο τοπικό κόστος LR.
- **Εισαγωγή buffer για ελάττωση των early παραβιάσεων χρονισμού**: Βρίσκει τη θέση και των αριθμό των buffer που πρέπει να τοποθετηθούν στο κύκλωμα, κάνοντας αυτή την επιλογή με βάση την διαφορά $\lambda^E - \lambda^L$.
- **Εισαγωγή buffer για ελάττωση των late παραβιάσεων χρονισμού**: Επιλέγει εάν θα πρέπει να τοποθετηθεί buffer στην έξοδο κελιών με υψηλό λόγο χωρητικότητας εξόδου/εισόδου. Εξετάζει όλες τις πιθανές εκδοχές buffer και διατηρεί την επιλογή με το μικρότερο τοπικό κόστος LR, συμπεριλαμβανομένης και της περίπτωσης μη τοποθέτησης buffer.
- **Συγχώνευση - Διαχωρισμός πυλών**: Η συγχώνευση πυλών δοκιμάζει την αντικατάσταση δύο όμοιων κελιών συνδεδεμένων σε σειρά με ένα κελί τριών εισόδων, ενώ ο διαχωρισμός πυλών δοκιμάζει την αντίθετη διαδικασία. Οι μετασχηματισμοί αυτοί δεν χρησιμοποιούνται σε αυτή την εφαρμογή, αλλά στη μέθοδο που παρουσιάζεται στο κεφάλαιο 4.

Όλες οι παραπάνω μέθοδοι οδηγούνται από τις τιμές των πολλαπλασιαστών Lagrange και αποσκοπούν στη μείωση της συνάρτησης κόστους LR, χρησιμοποιώντας πληροφορίες

χρονισμού μόνο για να επιβεβαιώσουν ότι οι τοπικές αποφάσεις δεν εισάγουν παραβιάσεις χρονισμού. Εφόσον η πληροφορία χρονισμού δεν είναι απαραίτητο να είναι ακριβής, όλοι οι μετασχηματισμοί μέσα σε μία επανάληψη μπορούν να εκτελεστούν χωρίς την ανάγκη πλήρους ενημέρωσης χρονισμού.

2.2 Πειραματικά αποτελέσματα

Η προτεινόμενη μέθοδος υλοποιήθηκε σε C++ και ενσωματώθηκε στο πρόγραμμα ανοιχτού κώδικα RSyn. Τα πειραματικά αποτελέσματα επαληθεύτηκαν στα κυκλώματα του διαγωνισμού TAU 2019 και συγκρίθηκαν με αυτά του νικητή του διαγωνισμού. Για την εξασφάλιση αρτιότερης σύγκρισης, αφαιρέσαμε τα μη ρεαλιστικά δίκτυα ρολογιού από τα κυκλώματα του διαγωνισμού, διατηρώντας τους χρόνους άφιξης ρολογιού που επιλέχθηκαν στα κυκλώματα του νικητή.

Πίνακας 2.1: Η χαμηλότερη επιτεύξιμη περίοδος ρολογιού για όλες τις συνθήκες λειτουργίας

Κύκλωμα	Περίοδος (ps)		Ισχύς (uW)		Εμβαδόν (um^2)	
	Ours	Winner	Ours	Winner	Ours	Winner
s1196	1040	918	12.1	12.6	550	569
systemcdes	1777	1788	87	96	3665	3975
usb_funct	2166	2306	419	402	18579	17812
vga_lcd	1871	2826	3215	3106	149033	146257
leon2_iccad	4532	4677	25170	30354	1237510	1312960
leon3mp_iccad	3878	5246	20045	23816	971773	1030860

Στον πίνακα 2.1 παρουσιάζουμε την καλύτερη περίοδο ρολογιού για την οποία η κάθε μέθοδος μπορεί να πετύχει κλείσιμο χρονισμού. Κατά μέσο όρο, η προσέγγισή μας πετυχαίνει 14% χαμηλότερη περίοδο ρολογιού, ενώ ταυτόχρονα εξοικονομεί 15% κατανάλωση ενέργειας και 5% εμβαδόν. Συγκεκριμένα, η μέθοδός μας πετυχαίνει χαμηλότερη περίοδο ρολογιού για όλα τα κυκλώματα εκτός από ένα, με τη βελτίωση να φτάνει μέχρι και το 35%. Με εξαίρεση το κύκλωμα s1196 όπου η προσέγγισή μας πετυχαίνει 5% υψηλότερη περίοδο, σε κάθε άλλο κύκλωμα πετυχαίνουμε καλύτερη περίοδο ρολογιού με ελάχιστη ή μηδαμινή επιβάρυνση στην κατανάλωση ενέργειας και στο εμβαδόν.

Πίνακας 2.2: Σύγκριση κατανάλωσης ενέργειας και εμβαδού στην περίοδο ρολογιού όπου και οι δύο προσεγγίσεις επιτυγχάνουν κλείσιμο χρονισμού για όλες τις συνθήκες λειτουργίας.

Κύκλωμα	Περίοδος (ps)	Ισχύς (uW)		Εμβαδόν (um^2)	
		Ours	Winner	Ours	Winner
s1196	1040	12	13	550	569
systemcdes	1788	85	96	3603	3975
usb_funct	2306	397	402	18002	17812
vga_lcd	2826	3075	3106	145752	146257
leon2_iccad	4677	24996	30354	1234180	1312960
leon3mp_iccad	5246	19632	23816	962764	1030860

Επιπλέον, ο πίνακα 2.2 παρουσιάζει τα αποτελέσματα της εκτέλεσης των δύο μεθόδων για την ίδια περίοδο ρολογιού, στην οποία και οι δύο προσεγγίσεις επιτυγχάνουν κλείσιμο χρονισμού. Η μέθοδός μας πετυχαίνει 16% μείωση της ισχύος και 6% μείωση του εμβαδού κατά μέσο όρο. Οι βελτιώσεις αυτές αναδεικνύουν την αποτελεσματικότητα της προσέγγισής μας, με την αρμονική συνεργασία των μετασχηματισμών κάτω από το κοινό μαθηματικό πλαίσιο του Largangian Relaxation.

3 Οργάνωση ευριστικών μεθόδων βελτιστοποίησης χρονισμού με τη χρήση Multi-Armed Bandit

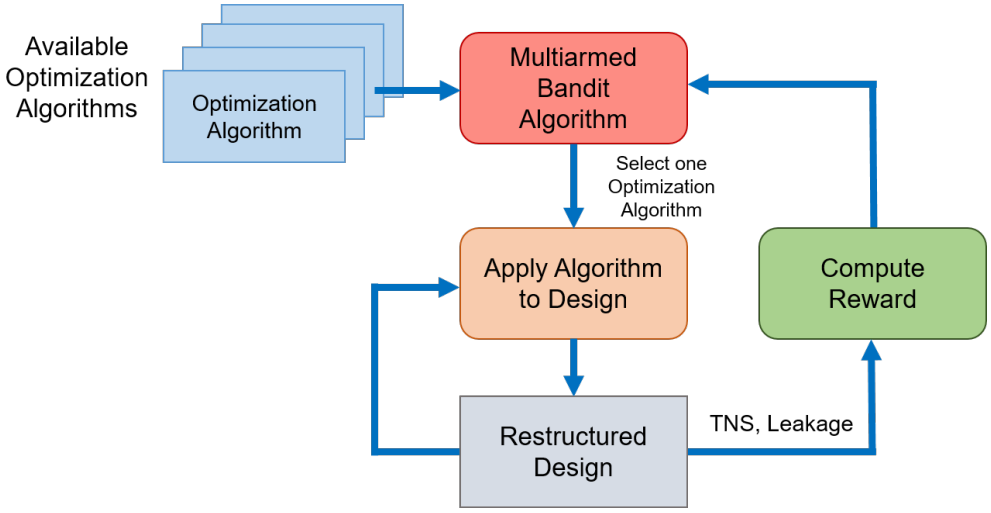
Στο προηγούμενο κεφάλαιο εξετάσαμε τη σημασία της ενσωμάτωσης διαφόρων αλγορίθμων βελτιστοποίησης στον ίδιο βρόχο, οι οποίοι εφαρμόζονται διαδοχικά με σταθερή σειρά. Ωστόσο, ακόμη και αν κάθε αλγόριθμος είναι αποτελεσματικός στη βελτιστοποίηση του κυκλώματος, η σειρά εφαρμογής των διαφόρων αλγορίθμων είναι εξίσου σημαντική για το τελικό αποτέλεσμα. Το πρόβλημα αυτό υπάρχει και σε σύγχρονα εργαλεία αυτόματης σύνθεσης υλικού, όπου η σειρά εκτέλεσης διαφόρων μεθόδων βελτιστοποίησης είναι σταθερή και συνήθως επιλέγεται από έμπειρους μηχανικούς. Σε περίπτωση που αυτή η προαποφασισμένη σειρά δεν επιτύχει κλείσιμο χρονισμού, τότε χρειάζεται κοπιαστικός και χρονοβόρος πειραματισμός για να βρεθεί η κατάλληλη αλληλουχία βελτιστοποιήσεων για το εκάστοτε κύκλωμα.

Σε αυτή τη διατριβή χρησιμοποιούμε το μοντέλο Multi-Armed Bandit (MAB), μια μορφή ενισχυτικής μάθησης, σε μια ροή σχεδίασης κυκλωμάτων για την αυτόνομη εφαρμογή μεθόδων βελτιστοποιήσεων, με στόχο την επίτευξη κλεισίματος χρονισμού με το ελάχιστο δυνατό εμβαδόν και κατανάλωσης ενέργειας. Η προτεινόμενη μέθοδος στοχεύει στην απάντηση της παρακάτω κρίσιμης ερώτησης: με δεδομένα ένα σύνολο από σαφώς ορισμένες μεθόδους βελτιστοποίησης κυκλωμάτων με αβέβαιη απόδοση, και ένα σύνολο από χρονικούς περιορισμούς, πως θα μπορούσε κάποιος να διαλέξει ποια μέθοδο να εφαρμόσει σε πραγματικό χρόνο, και χωρίς προηγούμενη γνώση του κυκλώματος, με σκοπό τη βελτιστοποίηση του κυκλώματος μέσα από πολλαπλά βήματα, διατηρώντας παράλληλα το χρόνο εκτέλεσης υπό έλεγχο. Σύμφωνα με τη μέθοδο MAB, μια "επιβράβευση" καταγράφεται για κάθε μέθοδο βελτιστοποίησης που εφαρμόζεται. Το υποσύστημα MAB επιλέγει (προτείνει) ποια μέθοδος θα πρέπει να εφαρμοστεί στη συνέχεια, ακολουθώντας μια προσέγγιση η οποία ισορροπεί την εκμετάλλευση μεθόδων με υψηλή απόδοση στις προηγούμενες δοκιμές, και την εξερεύνηση μεθόδων που δεν έχουν εφαρμοστεί αρκετά ώστε να έχουμε ξεκάθαρη εικόνα για την απόδοσή τους.

Η μέθοδος εφαρμόστηκε με επιτυχία στα κυκλώματα των διαγωνισμών βελτιστοποίησης χρονισμού ISPD12 και TAU2019. Η σημαντικότερη συνεισφορά αυτής της εργασίας είναι το ότι η βελτιστοποίηση κυκλωμάτων βασισμένη σε MAB όχι μόνο πετυχαίνει κλείσιμο χρονισμού και ανταγωνιστικά αποτελέσματα εμβαδού και ισχύος, αλλά και το ότι είναι αυτόνομη, χωρίς να βασίζεται σε προηγούμενη γνώση των κυκλωμάτων ή των μεθόδων βελτιστοποίησης, και χωρίς την ανάγκη ανθρώπινης παρέμβασης.

3.1 Αυτόνομη βελτιστοποίηση κυκλωμάτων

Η προτεινόμενη μεθοδολογία αυτοματοποιεί την εφαρμογή αλγορίθμων βελτιστοποίησης διαφόρων επιπέδων πολυπλοκότητας στο εκάστοτε κύκλωμα, με στόχο την αυτόνομη επί-



Σχήμα 3.1: Η ροή της αυτόνομης βελτιστοποίησης κυκλωμάτων. Το κύκλωμα βελτιστοποείται επαναληπτικά από τον αλγόριθμο ο οποίος επιλέγεται σε κάθε βήμα από τη μεθοδολογία MAB, με βάση το ιστορικό επιβραβεύσεων που έχει λάβει και τη συχνότητα με την οποία έχει εφαρμοστεί.

τευξη κλεισίματος χρονισμού και βελτίωσης του εμβαδού και της κατανάλωσης ενέργειας. Ουσιαστικά το προτεινόμενο πλαίσιο βελτιστοποίησης παρομοιάζεται με έναν πάικτη τυχερών παιχνιδιών μπροστά από διάφορες μηχανές "ληστών" (οι αλγόριθμοι βελτιστοποίησης), και πρέπει να διαλέξει σε ποια μηχανή να παίξει, πόσες φορές να παίξει σε κάθε μηχανή και με ποια σειρά ώστε να μεγιστοποιήσει το κέρδος (ελαχιστοποίηση παραβιάσεων χρονισμού, εμβαδού και κατανάλωσης ενέργειας). Η ποιότητα του κάθε αλγορίθμου αξιολογείται με βάση την επιβράβευση που λαμβάνει μετά τηςν εκτέλεσή του, η οποία βασίζεται στην αλλαγή των μετρικών του κυκλώματος.

Η προτεινόμενη ροή βελτιστοποίησης παρουσιάζεται στο σχήμα 3.1. Σε κάθε επανάληψη, ένας αλγόριθμος βελτιστοποίησης επιλέγεται και εφαρμόζεται στο κύκλωμα. Μετά την εκτέλεσή του, υπολογίζεται η επιβράβευση που θα λάβει με βάση τις αλλαγές του στο κύκλωμα, με σκοπό να καθοδηγήσει την επιλογή αλγορίθμου στις επόμενες επαναλήψεις. Οι αλλαγές που προκάλεσε ο αλγόριθμος μπορεί να απορριφθούν εάν οδηγήσουν σε σημαντική χειροτέρευση του κυκλώματος. Η διαδικασία σταματά όταν η λαμβανόμενη επιβράβευση δεν μπορεί να βελτιωθεί περισσότερο από 1% για 10 συνεχόμενες επαναλήψεις.

Η μεθοδολογία MAB χρησιμοποιεί τον μέσο όρο επιβραβεύσεων που έχει λάβει ο κάθε αλγόριθμος βελτιστοποίησης για να διαλέξει το επόμενο βήμα, με βάση την επιλεγμένη στρατηγική εκμετάλλευσης-εξερεύνησης. Η συνεχής εκμετάλλευση μεθόδων με υψηλή επιβράβευση μπορεί να οδηγήσει στην αμέλεια άλλων, ακόμη πιο αποτελεσματικών μεθόδων, ενώ η υπερβολική εξερεύνηση μπορεί να σπαταλήσει πολλές δοκιμές χωρίς τη χρησιμοποίηση της γνώσης που έχουμε αποκομίσει από το μέσο όρο επιβραβεύσεων. Αφότου όλες οι κινήσεις έχουν εφαρμοστεί από μία φορά, για το επόμενο βήμα επιλέγεται η κίνηση με τη μέγιστη τιμή $\overline{Q}_{t-1}(a) + \sqrt{\frac{2 \log t}{N_a}}$, όπου το $\overline{Q}_{t-1}$ συμβολίζει τη μέση επιβράβευση του αλγορίθμου a στα προηγούμενα $t-1$ βήματα, και το N_a αναπαριστά το πόσες φορές έχει επιλεχθεί ο συγκεκριμένος αλγόριθμος. Λόγω των δυναμικών χαρακτηριστικών του προβλήματος, η μέση επιβράβευση υπολογίζεται με τέτοιο τρόπο ώστε να δίνει μεγαλύτερη έμφαση στις πιο πρόσφατες εκτελέσεις του αλγορίθμου.

3.2 Συνάρτηση επιβράβευσης

Η συνάρτηση επιβράβευσης αναπαριστά την επίδραση του επιλεγμένου αλγορίθμου βελτιστοποίησης στις μετρικές απόδοσης του κυκλώματος (χρονισμός, κατανάλωση ενέργειας). Η επιβράβευση που δίνεται στον αλγόριθμο a στο βήμα t είναι ίση με:

$$Q_t(a) = r_a \left(\delta \cdot qTNS + (1 - \delta) \cdot qPower\right)$$

Οι παράμετροι $qTNS$ και $qPower$ έχουν τιμές μεταξύ 0 και 1, και αναπαριστούν την απόδοση του αλγορίθμου στη μείωση των παραβιάσεων χρονισμού και κατανάλωσης ενέργειας αντίστοιχα. Ο παράγοντας δ ελέγχει την προτεραιότητα μεταξύ βελτίωσης του χρονισμού ή της κατανάλωσης ενέργειας, παίρνοντας την τιμή 0.8 αρχικά, και την τιμή 0.2 αφότου επιτευχθεί το κλείσιμο χρονισμού. Η παράμετρος r_a μειώνει την επιβράβευση πολύπλοκων μεθόδων με υψηλό χρόνο εκτέλεσης.

Η τιμή του $qTNS$ ποσοτικοποιεί τη μείωση των παραβιάσεων χρονισμού σε σχέση με την αύξηση της κατανάλωσης ενέργειας. Υπολογίζεται χωριστά για early και late συνθήκες, και η τελική τιμή είναι ο μέσος όρος μεταξύ των δύο. Σε περίπτωση που τόσο οι παραβιάσεις όσο και η κατανάλωση ενέργειας αυξηθούν, το $qTNS$ ισούται με 0. Αντίθετα, σε περίπτωση που ελαττωθούν, το $qTNS$ ισούται με 1. Σε κάθε άλλη περίπτωση:

$$qTNS = \min\left(\frac{\Delta TNS}{\gamma\ \Delta P}, 1\right)$$

Τα ΔTNS και ΔP αναπαριστούν την σχετική μείωση των παραβιάσεων χρονισμού και κατανάλωσης ενέργειας, ενώ ο όρος γ καθορίζει τον λόγο μείωσης παραβιάσεων χρονισμού και αύξηση κατανάλωσης ενέργειας που θεωρούμε ικανοποιητικό.

Με αντίστοιχη λογική, το $qPower$ ισούται με

$$qPower = \min\left(\frac{\Delta P}{\gamma\ \Delta TNS}, 1\right).$$

3.3 Πειραματικά αποτελέσματα

Όπως και στο προηγούμενο κεφάλαιο, η ροή βελτιστοποίησης υλοποιήθηκε σε C++ και ενσωματώθηκε στο εργαλείο ανοιχτού κώδικα RSyn. Η μέθοδος ενορχήστρωσης MAB είναι υπεύθυνη για την επιλογή του αλγόριθμου βελτιστοποίησης που πρέπει να εφαρμοστεί σε κάθε περίπτωση. Σε αυτή την εργασία οι διαθέσιμοι αλγόριθμοι είναι οι εξής:

- Επιλογή μεγέθους κελιών βασισμένη σε LR (LR1), για μία επανάληψη, εξετάζοντας όλα τα κελιά του κυκλώματος.
- Επιλογή μεγέθους κελιών βασισμένη σε LR (LR2), για δύο επαναλήψεις, εξετάζοντας όλα τα κελιά του κυκλώματος.
- Επιλογή μεγέθους κελιών βασισμένη σε LR (LR1C), για μία επανάληψη, εξετάζοντας τα 1000 πιο κρίσιμα κελιά του κυκλώματος.
- Εισαγωγή buffer για βελτίωση του late χρονισμού (LB), με βάση τον λόγο χωρητικότητας εισόδου/εξόδου.
- Εισαγωγή buffer για βελτίωση του early χρονισμού (EB) σε σημεία του κυκλώματος με early χρονικές παραβιάσεις.

- Εισαγωγή buffer στην είσοδο ρολογιού καταχωρητών (CB) με βάση την κρισιμότητα χρονισμού στα D και Q pin.
- Εισαγωγή buffer για απομόνωση του κρίσιμου μονοπατιού (CPI), με στόχο τη μείωση της χωρητικότητας εισόδου μονοπατιών με late χρονικές παραβιάσεις.

Πίνακας 3.1: Σύγκριση αποτελεσμάτων της προσέγγισης βασισμένης σε MAB με τα καλύτερα δημοσιευμένα αποτελέσματα για τα κυκλώματα του διαγωνισμού ISPD12

Κύκλωμα	#Κελιών	Ισχύς (mW)		Αλγόριθμοι βελτιστοποίησης						
		Flach-2012	MAB	CPI	LB	EB	LR1	LR1C	LR2	#Δοκιμών
DMA_slow	25300	132	134	8	6	5	10	9	10	48
DMA_fast		238	248	11	9	8	14	11	10	63
pci_bridge3_slow	33203	96	96	10	7	6	10	9	12	54
pci_bridge3_fast		136	138	6	6	4	11	5	11	43
des_perf_slow	111228	570	601	9	6	4	11	8	5	43
des_perf_fast		1395	1511	8	7	5	11	12	12	55
vga_lcd_slow	164890	328	335	6	5	4	6	8	4	33
vga_lcd_fast		412	430	10	6	6	10	10	9	51
b19_slow	219268	564	660	4	2	2	4	3	3	18
b19_fast		717	769	6	3	3	5	5	4	26
leon3mp_slow	649190	1334	1340	6	6	3	5	5	5	30
leon3mp_fast		1443	1530	6	7	4	9	6	9	41
netcard_slow	958780	1763	1755	4	3	2	4	2	3	18
netcard_fast		1841	1884	4	3	3	5	4	5	24
Μέσος όρος		783.5	816.5	7.0	5.4	4.2	8.2	6.9	7.3	39.1

Στον πίνακα 3.1 παρουσιάζουμε τη σύγκριση των αποτελεσμάτων μας με τα καλύτερα δημοσιευμένα αποτελέσματα στη βιβλιογραφία για τα κυκλώματα ISPD12. Η μεθοδολογία MAB προσεγγίζει με επιτυχία τη βέλτιστη κατανάλωση ισχύος στην πλειοψηφία των κυκλωμάτων με μικρότερο αριθμό επαναλήψεων, χρησιμοποιώντας πλήρως αυτόνομη ροή βελτιστοποίησης χωρίς ανθρώπινη παρέμβαση. Στον πίνακα παρουσιάζεται επίσης ο αριθμός εκτέλεσης κάθε αλγορίθμου βελτιστοποίησης, από τον οποίο φαίνεται ότι η μέθοδος ενορχήστρωσης MAB καταναλώνει τον περισσότερο χρόνο σε μεθόδους που αποδυκνείονται πιο αποδοτικές για το εκάστοτε κύκλωμα.

4 Γενική βελτιστοποίηση LR και συντονισμένες τοπικές προτάσεις βασισμένες σε MAB

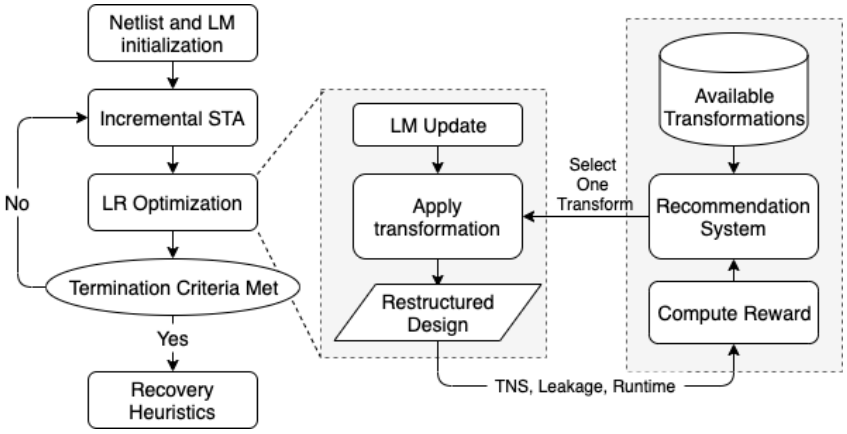
Στο κεφάλαιο 2 παρουσιάσαμε την ενσωμάτωση διάφορων μεθόδων βελτιστοποίησης στο ίδιο μαθηματικό πλαίσιο βασισμένο σε LR, ενώ στο κεφάλαιο 3 χρησιμοποιήσαμε την προσέγγιση MAB για την ενορχήστρωση της σειράς εκτέλεσης ετερογενών μετασχηματισμών. Στην τελευταία περίπτωση, ο αλγόριθμος MAB δεν καταφέρνει να προσαρμοστεί στην δυναμική φύση του προβλήματος, κάνοντας την συνολική βελτιστοποίηση καθαρά στατιστική.

Αυτή η εργασία αντιπετωπίζει το πρόβλημα χρησιμοποιώντας μια τροποποημένη εκδοχή του MAB με σκοπό την αυτόνομη βελτιστοποίηση άγνωστων κυκλωμάτων, επιλέγοντας ποια μέθοδο βελτιστοποίησης βασισμένη σε LR να εφαρμόσει σε κάθε επανάληψη από ένα ευρύ σύνολο μετασχηματισμών. Κάθε μετασχηματισμός ενσωματώνεται ομαλά χωρίς να διαταράσσει τη συνολική διαδικασία βελτιστοποίησης, επιτρέποντας έτσι στη λύση που παράγεται από μία μέθοδο να προσαρμοστεί σταδιακά στη λύση που παράχθηκε από έναν προηγούμενο μετασχηματισμό. Σε κάθε περίπτωση, οι απαιτούμενες τοπικά βέλτιστες αποφάσεις για κάθε μετασχηματισμό λαμβάνονται χρησιμοποιώντας συναρτήσεις κόστους βασισμένες σε LR. Αυτό επιτρέπει την έυκολη ενσωμάτωση επιπλέον τοπικών μετασχηματισμών με την προϋπόθεση ότι λαμβάνουν αποφάσεις βασισμένες στο κόστος LR.

Η προτεινόμενη μέθοδος εφαρμόστηκε με επιτυχία στα κυκλώματα των διαγωνισμών TAU2019 και ISPD13, στα οποία αποδεικνύεται η αποτελεσματικότητά της. Σε κάθε περίπτωση, η προσέγγισή μας βελτιστοποιεί με επιτυχία τα κυκλώματα, και πετυχαίνει σημαντικά καλύτερα αποτελέσματα από τα δημοσιευμένα. Το πιο σημαντικό είναι το ότι ο μηχανισμός αυτόματης επιλογής μετασχηματισμών επιτρέπει την ενσωμάτωση νέων μεθόδων στη ροή βελτιστοποίησης LR χωρίς την ανάγκη χειροκίνητης ρύθμισης και χωρίς την υποβάθμιση των αποτελεσμάτων. Ταυτόχρονα, αντιμετωπίζει τα μειονεκτήματα της προσέγγισης του κεφαλαίου 3 με τη χρήση της μεθόδου LR για τη δημιουργία ενός ολικού κόστους βελτιστοποίησης το οποίο ελαχιστοποιείται τοπικά από τον εκάστοτε επιλεγμένο μετασχηματισμό. Επιπλέον, ο προτεινόμενος μηχανισμός επιβράβευσης και το σύστημα MAB είναι περισσότερο πολύπλοκα και έχουν ρυθμιστεί κατάλληλα για τη δυναμική φύση της διαδικασίας βελτιστοποίησης.

4.1 Αυτόνομη ενορχήστρωση βελτιστοποιήσεων LR

Η μοντελοποίηση LR του προβλήματος είναι όμοια με αυτή του κεφαλαίου 2, με την προσθήκη συντελεστών κλιμάκωσης, καθώς η κατανάλωση ενέργειας, η καθυστέρηση και το εμβαδόν μετρώνται σε διαφορετικές μονάδες. Ακολουθούμε μια προσέγγιση επαναληπτικής βελτιστοποίησης, όπου σε κάθε επανάληψη το σύστημα MAB επιλέγει την εφαρμογή ενός από τους 8 διαθέσιμους μετασχηματισμούς, οι οποίοι προσπαθούν να ελαχιστοποιή-



Σχήμα 4.1: Η συνολική ροή βελτιστοποίησης. Οι μετασχηματισμοί εφαρμόζονται σειριακά στο κύκλωμα και επιλέγονται αυτόματα από το σύστημα πρότασης/επιβράβευσης. Ο κάθε μετασχηματισμός βασίζεται στα βάρη πολλαπλασιαστών Lagrange, συνεισφέροντας έτσι στη σύγκλιση της συνολικής ροής βελτιστοποίησης βασισμένη σε LR.

σουν τοπικές εκδοχές της συνολικής συνάρτησης κόστους.

Η συνολική ροή βελτιστοποίησης παρουσιάζεται στο σχήμα 4.1. Αρχικά, όλα τα κελιά αντιστοιχίζονται στο μικρότερο μέγεθος το οποίο δεν εισάγει παραβιάσεις μέγιστης χωρητικότητας ή χρόνου μετάβασης. Στη συνέχεια, οι πολλαπλασιαστές Lagrange αρχικοποιούνται στην τιμή 1, και η βελτιστοποίηση ξεκινά.

Στην αρχή κάθε επανάληψης ο χρονισμός και οι τιμές των πολλαπλασιαστών Lagrange ενημερώνονται. Ο μετασχηματισμός που θα εφαρμοστεί χρησιμοποιεί τις ενημερωμένες τιμές των πολλαπλασιαστών για να λάβει τοπικά βέλτιστες αποφάσεις. Η επιλογή μετασχηματισμού πραγματοποιείται μέσω του συστήματος MAB. Η χρονική πληροφορία προέρχεται από τις 2 πιο κρίσιμες συνθήκες λειτουργίας, την πιο κρίσιμη early και late. Ο χρονισμός ανανεώνεται για όλες τις συνθήκες λειτουργίας κάθε 5 επαναλήψεις, όπου και προσδιορίζονται οι νέες κρισιμότερες συνθήκες λειτουργίας. Οι διαθέσιμοι μετασχηματισμοί είναι αυτοί που περιγράφονται στο κεφάλαιο 2.

Η επιλογή του μετασχηματισμού που θα εφαρμοστεί περιλαμβάνει βελτιώσεις σε σχέση με αυτή του κεφαλαίου 3, έτσι ώστε να προσαρμόζεται καλύτερα στο δυναμικό πρόβλημα της βελτιστοποίησης κυκλωμάτων. Η διαδικασία που ακολουθείται είναι η εξής:

Όταν ο μετασχηματισμός k εφαρμόζεται στο κύκλωμα, αποθηκεύουμε την επιβράβευση που λαμβάνει, R_k, χρησιμοποιώντας τη συνάρτηση επιβράβευσης η οποία περιγράφεται στο επόμενο υποκεφάλαιο. Στη συνέχεια οι επιβραβεύσεις των μετασχηματισμών που εφαρμόστηκαν στις τελευταίες N επαναλήψεις ταξινομούνται σε φθίνουσα σειρά. Ο κάθε μετασχηματισμός λαμβάνει ένα σκορ με βάση τη θέση του στην ταξινομημένη σειρά, με τον πρώτο μετασχηματισμό να λαμβάνει το σκορ $N-1$, τον δεύτερο $N-2$ κτλ. Το συνολικό σκορ του κάθε μετασχηματισμού ισούται με το άθροισμα των σκορ της κάθε παρουσίας του μετασχηματισμού στην ταξινομημένη σειρά. Στη συνέχεια, η απόδοση Q_k ισούται με το κανονικοποιημένο σκορ του μετασχηματισμού k:

$$Q_k = \frac{\text{score}(k)}{1 + 2 + ... + N} \tag{4.1}$$

Στη συνέχεια, ο μηχανισμός MAB επιλέγει την εφαρμογή του μετασχηματισμού σε αυτή

την επανάληψη ο οποίος ελαχιστοποιεί την συνάρτηση

$$(1-c)Q_k + c\sqrt{\frac{2\ln N}{N_k}}, \tag{4.2}$$

όπου το N_k αναπαριστά το πόσες φορές έχει εφαρμοστεί ο μετασχηματισμός k. Το c είναι μια μεταβλητή συντονισμού μεταξύ 0 και 1 η οποία, όταν αυξάνεται, ευνοεί την εξερεύνηση μετασχηματισμών οι οποίοι δεν έχουν επιλεγεί αρκετά (χαμηλή τιμή N_k) σε σχέση με την εκμετάλλευση αυτών οι οποίοι έχουν την καλύτερη απόδοση μέχρι στιγμής (υψηλή τιμή Q_k).

4.2 Καθοδηγώντας την αυτόματη βελτιστοποίηση: Η συνάρτηση επιβράβευσης

Η συνάρτηση επιβράβευσης αντικατοπτρίζει το αποτέλεσμα της επιλεγμένης μεθόδου βελτιστοποίησης στις μετρικές απόδοσης οι οποίες χαρακτηρίζουν τη συμπεριφορά του κυκλώματος, όπως επίσης και το χρόνο εκτέλεσης που χρειάστηκε για να επιτευχθούν αυτές οι μετρικές. Συνολικά, η επιβράβευση για την επανάληψη i ισούται με:

$$R_k(i) = r_k\left(\delta R_{\text{timing}}(i) + (1-\delta)R_{\text{power-area}}(i)\right) \tag{4.3}$$

Τα $R_{\text{timing}}(i)$ και $R_{\text{power-area}}(i)$ αναπαριστούν την αποτελεσματικότητα της επιλεγμένης μεθόδου στη μείωση των χρονικών παραβιάσεων, και της κατανάλωσης ενέργειας και εμβαδού αντίστοιχα. Ο παράγοντας δ καθορίζει την έμφαση στη βελτίωση του χρονισμού ή της κατανάλωσης ενέργειας-εμβαδού, παίρνοντας την τιμή 0.8 κατά τη διάρκεια κλεισίματος χρονισμού, και 0.2 μετέπειτα. Ο παράγοντας r_k μειώνει την επιβράβευση μεθόδων με υψηλό χρόνο εκτέλεσης, με σκοπό το σύστημα MAB να μπορεί να δώσει προτεραιότητα σε μια πιο γρήγορη μέθοδο με ανάλογη επίδραση στο κύκλωμα. Εξελίσσεται δυναμικά με βάση το μέσο χρόνο εκτέλεσης της κάθε μεθόδου.

Η επιβράβευση χρονισμού $R_{\text{timing}}(i)$ αναπαριστά τη βελτίωση του χρονισμού σε σχέση με την αύξηση της κατανάλωσης ενέργειας-εμβαδού, και ισούται με:

$$R_{\text{timing}}(i) = \begin{cases} \frac{\Delta T(i)}{\min(\Delta PA(i), 0.01)}, & \text{εάν ο χρονισμός βελτιώθηκε} \\ 0, & \text{εάν ο χρονισμός δεν βελτιώθηκε} \end{cases}$$

Ο όρος $\Delta T(i)$ αναφέρεται στην ποσοστιαία βελτίωση των χρονικών παραβιάσεων (TNS) σε σχέση με την προηγούμενη επανάληψη για Late και Early χρονισμό, ενώ ο $\Delta PA(i)$ αναπαριστά την ποσοστιαία αύξηση της κατανάλωσης ενέργειας-εμβαδού.

Χρησιμοποιώντας την ίδια φιλοσοφία, η επιβράβευση κατανάλωσης ενέργειας-εμβαδού ισούται με:

$$R_{\text{power-area}}(i) = \begin{cases} 0, & \text{εάν η κατανάλωση ενέργειας-εμβαδό έχουν αυξηθεί} \\ \frac{\Delta PA(i)}{\min(\Delta T(i), 0.01)}, & \text{εάν ο χρονισμός δεν έχει κλείσει} \\ \Delta PA(i), & \text{εάν ο χρονισμός έχει κλείσει} \end{cases}$$

4.3 Πειραματικά αποτελέσματα

Όπως και στα προηγούμενα κεφάλαια, η μέθοδος ενσωματώθηκε στο εργαλείο ανοιχτού κώδικα RSyn, αφότου το επεκτείναμε ώστε να μπορεί να διαχειριστεί χρονική ανάλυση σε πολλαπλές συνθήκες λειτουργίας. Τα αποτελέσματα αξιολογήθηκαν στα κυκλώματα TAU2019 και ISPD13. Το παράθυρο το οποίο χρησιμοποιείται για την ταξινόμηση των εφαρμοσμένων μετασχηματισμών είναι 4 φορές μεγαλύτερο από το σύνολο των διαθέσιμων μετασχηματισμών και ισούται με 32.

Πίνακας 4.1: Σύγκριση της κατανάλωσης ενέργειας και εμβαδού στην περίοδο ρολογιού όπου και οι 2 προσεγγίσεις πετυχαίνουν κλείσιμο χρονισμού

Κύκλωμα	#κελιών	Περίοδος (ps)	Ισχύς (μW)			Εμβαδόν (μm^2)			Χρόνος εκτ. (min)		
			Ours	Winner	Win(%)	Ours	Winner	Win(%)	Ours	#Επαν.	Winner
s1196	584	985	12	13	7.69	552	569	2.99	0.03	24	0.03
systemcdes	2825	1788	74	96	22.92	3368	3975	15.27	0.19	46	0.35
usb_funct	10535	2306	370	402	7.96	17599	17812	1.20	0.77	45	0.87
vga_lcd	87958	2826	2780	3106	10.50	142153	146257	2.81	6.82	20	0.40
leon3mp_iccad	649191	5246	19351	23816	18.75	957947	1030860	7.07	57.25	22	6.03
leon2_iccad	793286	4677	23989	30354	20.97	1203920	1312960	8.30	46.72	19	7.53
average saves	-	-	-	-	14.80	-	-	6.27	-	-	-

Στον πίνακα 4.1 παρουσιάζουμε τα αποτελέσματα της προσέγγισής μας σε σύγκριση με τον νικητή του διαγωνισμού TAU2019, για την περίοδο ρολογιού όπου και οι 2 προσεγγίσεις πετυχαίνουν κλείσιμο χρονισμού. Η ροή βελτιστοποίησής μας πετυχαίνει χαμηλότερη κατανάλωση ενέργειας και χαμηλότερο εμβαδόν σε κάθε κύκλωμα, έχοντας κατά μέσο όρο ως αποτέλεσμα 6% βελτιωμένο εμβαδόν και 15% βελτιωμένη κατανάλωση ενέργειας. Τα αποτελέσματά μας στα μεγαλύτερα κυκλώματα είναι πολύ καλύτερα από του νικητή του διαγωνισμού, με αυξημένο χρόνο εκτέλεσης. Ο χρόνος εκτέλεσης προέρχεται κυρίως από μεθόδους οι οποίες δρουν σε ολόκληρο το κύκλωμα, όπως η επιλογή μεγέθους κελιών.

Τέλος, στον πίνακα 4.2 παρουσιάζονται τα αποτελέσματα στα κυκλώματα του διαγωνισμού ISPD13. Εφόσον επιτυγχάνεται κλείσιμο χρονισμού σε όλα τα κυκλώματα, στον πίνανα αναγράφονται μόνο η ισχύς (μετρική η οποία έπρεπε να ελαχιστοποιηθεί στο διαγωνισμό) και ο χρόνος εκτέλεσης. Η προσέγγισή μας συγκρίνεται με δύο προσεγγίσεις με αποτελέσματα υψηλής ποιότητας: η Flach-2012 επιστρέφει τα καλύτερα αποτελέσματα ισχύος, ενώ η Sharma-2019 καταλήγει σε ανταγωνιστικά αποτελέσματα με το μικρότερο χρόνο εκτέλεσης. Η προσέγγισή μας καταλήγει σε 25% χαμηλότερη κατανάλωση ενέργειας κατά μέσο όρο, ενώ ταυρόχρονα έχει συγκρίσιμο χρόνο εκτέλεσης.

Πίνακας 4.2: Αποτελέσματα για τα κυκλώματα του διαγωνισμού ISPD13.

Κύκλωμα	#κελιών	Ισχύς (mW)			Χρόνος εκτέλεσης (min)		
		Ours	Flach-12	Sharma-19	Ours	Flach-12	Sharma-19
usb_phy_slow	623	1	1	1	0.03	0.49	0.22
usb_phy_fast		1	2	2	0.06	0.42	0.23
pci_bridge32_slow	30763	53	57	58	1.93	10.53	0.97
pci_bridge32_fast		60	85	90	1.70	22.62	1.54
fft_slow	33792	84	87	88	1.61	25.71	1.37
fft_fast		131	194	213	3.30	40.43	1.64
cordic_slow	42937	220	267	293	4.24	69.04	2.29
cordic_fast		785	980	1080	4.38	117.08	5.66
des_perf_slow	113346	340	327	332	14.38	132.27	7.27
des_perf_fast		663	644	639	21.19	347.87	26.1
edit_dist_slow	129227	428	416	440	5.74	123.90	4.92
edit_dist_fast		532	535	549	10.44	352.96	6.66
matrix_mult_slow	159642	451	443	448	7.62	226.13	8.80
matrix_mult_fast		721	1541	1633	21.56	395.96	13.9
netcard_slow	984094	3663	5155	5170	58.36	483.55	24.6
netcard_fast		3772	5182	5205	64.52	400.89	30.6

5 Σύνθεση προσεγγιστικών αθροιστών βασισμένη σε ενισχυτική μάθηση

Η προσεγγιστική αριθμητική αναφέρεται σε κυκλώματα τα οποία θυσιάζουν την αριθμητική ακρίβεια του αποτελέσματος για βελτιωμένη κατανάλωση ενέργειας, και χρησιμοποιείται κυρίως σε εφαρμογές με ανοχή στο σφάλμα. Μία από τις βασικές προκλήσεις στη σχεδίαση προσεγγιστικών κυκλωμάτων έγκειται στην ταυτοποίηση της βέλτιστης ισορροπίας μεταξύ αριθμητικής ακρίβειας και κατανάλωσης ενέργειας. Σε αυτή τη διατριβή εξερευνούμε το χώρο λύσεων των προσεγγιστικών αθροιστών. Δεν εστιάζουμε σε συγκεκριμένη αρχιτεκτονική αθροιστών, αλλά σχεδιάζουμε προσεγγιστικούς αθροιστές παράλληλου προθέματος οι οποίοι αναπαριστούν ένα ευρύ φάσμα αρχιτεκτονικών αθροιστών. Με την αυτοματοποίηση της σύνθεσης προσεγγιστικών αθροιστών προσαρμοσμένους σε συγκεκριμένες εφαρμογές, στοχεύουμε στη δημιουργία μιας ευέλικτης και αποτελεσματικής προσέγγισης για την αξιοποίηση των πλεονεκτημάτων της προσεγγιστικής αριθμητικής.

Σε αυτή τη διατριβή, η σχεδίαση προσεγγιστικών αθροιστών παράλληλου προθέματος πραγματοποιείται με τη χρήση ενισχυτικής μάθησης. Η ενισχυτική μάθηση βελτιστοποιεί το εμβαδό του κυκλώματος, και ταυτόχρονα ελαχιστοποιεί το σφάλμα προσέγγισης σε επίπεδο εφαρμογής με την ενσωμάτωση κατάλληλης ανατροφοδότησης κατά τη διάρκεια της μάθησης. Η αξιολόγηση της κάθε λύσης κατά τη διάρκεια της μάθησης δεν βασίζεται σε ευρετικές μεθόδους ή σε γενικές μετρικές πρόβλεψης της απόδοσης, αλλά χρησιμοποιεί ένα νευρωνικό δίκτυο το οποίο μαθαίνει στρατηγικές αποκλειστικά μέσω της εξερεύνησης του χώρου λύσεων και από άμεση ανατροφοδότηση από λογική σύνθεση και από την απόδοση σε επίπεδο εφαρμογής. Με αυτό τον τρόπο, για κάθε εφαρμογή που εξετάζεται σε αυτή τη διατριβή, μπορούμε να συνθέσουμε αυτόματα νέους προσεγγιστικούς αθροιστές παράλληλου προθέματος χωρίς να βασιζόμαστε σε γενικές αρχιτεκτονικές οι οποίες μπορεί να έχουν κακή απόδοση σε συγκεκριμένες περιπτώσεις.

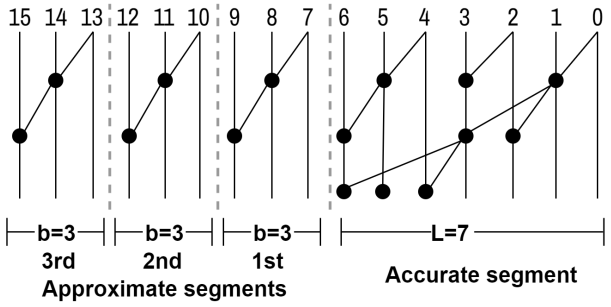
5.1 Γενική αρχιτεκτονική για προσεγγιστικούς αθροιστές παράλληλου προθέματος

Πριν συνθέσουμε αυτόματα προσεγγιστικούς αθροιστές παράλληλου προθέματος, πρέπει να ορίσουμε μια ενοποιημένη αναπαράσταση για την προσέγγιση κρατουμένου η οποία θα υλοποιηθεί από τον αθροιστή. Η αναπαράσταση πρέπει να είναι αρκετά ευέλικτη ώστε να καλύπτει ένα ευρύ φάσμα προσεγγίσεων, και αρκετά απλή ώστε να μπορεί να χρησιμοποιηθεί σε ένα πλαίσιο βελτιστοποίησης ενισχυτικής μάθησης.

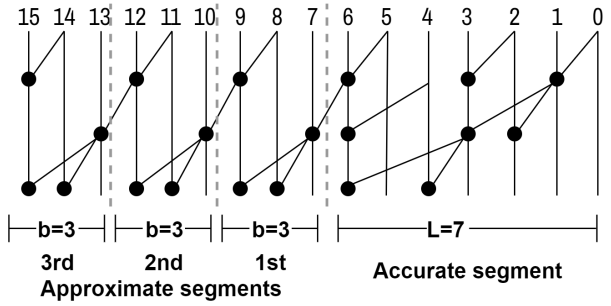
Τα bit εξόδου ενός αθροιστή παράλληλου προθέματος χωρίζονται σε $B + 1$ τμήματα. Το πρώτο τμήμα ξεκινά πάντα από τη θέση bit 0, και περιλαμβάνει τα L λιγότερο σημαντικά bit του αθροιστή. Τα υπόλοιπα B τμήματα χωρίζουν τα $N - L$ περισσότερο σημαντικά bit εξόδου του αθροιστή σε ίσα τμήματα μεγέθους $b = \frac{N-L}{B}$ το κάθε ένα. Εάν το $N - L$ και το B δεν διαιρούνται χωρίς υπόλοιπο, το περισσότερο σημαντικό τμήμα μπορεί να συμπεριλαμβάνει λιγότερα από b bit.

Το τμήμα των L λιγότερο σημαντικών bit ονομάζεται το *ακριβές τμήμα* του αθροιστή, καθώς τα κρατούμενα εξόδου, και συνεπώς και τα bit αθροίσματος υπολογίζονται με ακρίβεια. Τα υπόλοιπα B τμήματα συμπεριλαμβάνουν προσεγγιστικό υπολογισμό κρατουμένου. Για τα bit τα οποία ανήκουν σε προσεγγιστικά τμήματα, η διάδοση κρατουμένου δεν φτάνει στη θέση bit 0 αλλά σταματά σε υψηλότερη θέση bit.

Η διάδοση κρατουμένου σε κάθε τμήμα μπορεί να έχει επικάλυψη με άλλα τμήματα. Αυτό ελέγχεται από την τρίτη μεταβλητή της αναπαράστασης και ονομάζεται $overlap$.



(**a**)



(**b**)

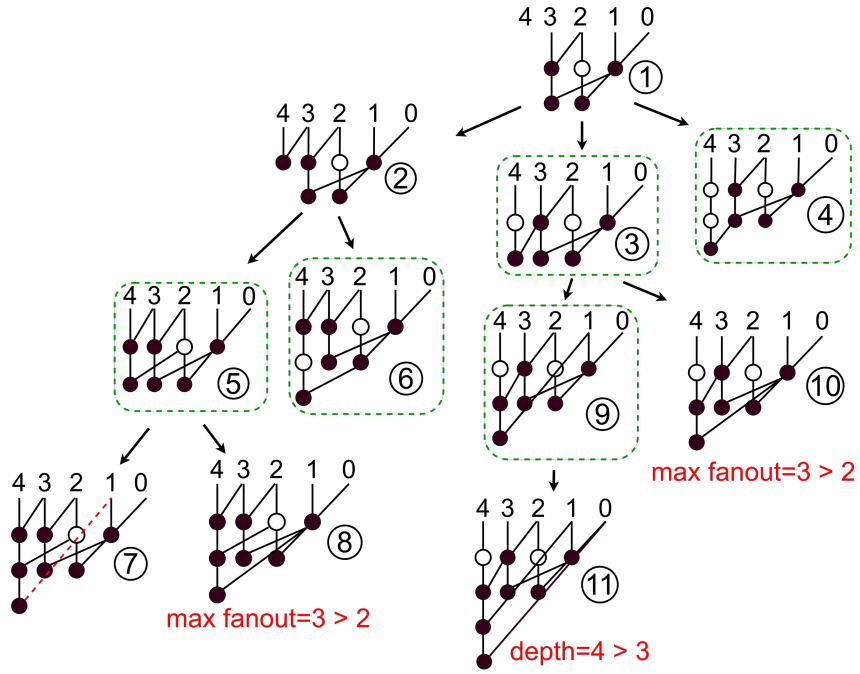
Σχήμα 5.1: Δύο παραδείγματα της προτεινόμενης γενικής προσέγγισης υπολογισμού κρατουμένου. Και οι δύο αθροιστές έχουν ένα ακριβές τμήμα μεγέθους $L = 7$ bit, και τρία προσεγγιστικά τμήματα μεγέθους $B = 3$ bit το κάθε ένα. Στην περίπτωση **a**), δεν υπάρχει επικάλυψη μεταξύ τμημάτων, ενώ στην περίπτωση **b**) υπάρχει επικάλυψη μεγέθους 2 θέσεων bit.

Ένα παράδειγμα της αναπαράστασης παρουσιάζεται στο σχήμα 5.1. Ο κάθε αθροιστής χωρίζεται σε 4 τμήματα, άρα $B = 3$. Και στις δύο περιπτώσεις, τα $L = 7$ λιγότερο σημαντικά bit σχηματίζουν το πρώτο ακριβές τμήμα, εφόσον η αλυσίδα κρατουμένου τους φτάνει μέχρι το 0, και τα υπόλοιπα 9 περισσότερο σημαντικά bit χωρίζονται σε $B = 3$ προσεγγιστικά τμήματα μεγέθους $b = 3$ bit το κάθε ένα, και η αλυσίδα κρατουμένου τους σταματάει πριν το bit 0. Στο σχήμα 5.1a, τα τέσσερα τμήματα είναι απομονωμένα και η διάδοση κρατουμένου ενός τμήματος ξεκινά μετά τη θέση bit όπου σταματάει το προηγούμενο τμήμα ($overlap = 0$). Αντιθέτως, στο σχήμα 5.1b, η διάδοση κρατουμένου μεταξύ τμημάτων έχει επικάλυψη. Για παράδειγμα , το κρατούμενο το οποίο υπολογίζεται στη θέση 11, το οποίο ανήκει στο 2ο προσεγγιστικό τμήμα, "κοιτάει" 2 θέσεις bit του 1ου προσεγγιστικού τμήματος ($overlap = 2$).

5.2 Απαρίθμηση προσεγγιστικών αθροιστών παράλληλου προθέματος

Η μέθοδος ενισχυτικής μάθησης η οποία προτείνεται σε αυτή τη διατριβή θα αποφασίζει ποιες τιμές χαρακτηριστικών αθροιστή (L, B, $overlap$) οδηγούν στην καλύτερη ισορροπία εμβαδού-σφάλματος για μία επιλεγμένη εφαρμογή. Για να αξιολογηθεί η απόφαση, ένας αθροιστής παράλληλου προθέματος με αυτά τα χαρακτηριστικά πρέπει να συντεθεί. Γι αυτό το σκοπό, προτείνουμε έναν αλγόριθμο απαρίθμησης ο οποίος παράγει όλους τους αθροιστές παράλληλου προθέματος που ικανοποιούν συγκεκριμένους περιορισμούς.

Η κατασκευή ενός προσεγγιστικού αθροιστή $n+1$ bit συμπεριλαμβάνει την αναδρομική προσθήκη τελεστών προθέματος στην στήλη του πιο σημαντικού bit ενός αθροιστή n bit. Οι n λιγότερο σημαντικές θέσεις bit παραμένουν ως έχουν. Η διαδικασία εφαρμόζεται χρησιμοποιώντας κάθε έγκυρο n bit αθροιστή ως αφετηρία για τη δημιουργία $n+1$ bit αθροιστών. Ως έγκυροι λογίζονται οι αθροιστές οι οποίοι ικανοποιούν τις συνθήκες εισόδου, οι οποίες περιλαμβάνουν το μέγιστο αριθμό λογικών επιπέδων, το μέγιστο αριθμό εξόδων ενός τελεστή, και την ελάχιστη προσεγγιστική αλυσίδα κρατουμένου.



Σχήμα 5.2: Παράδειγμα αναδρομικής δημιουργίας αθροιστών 5 bit ξεκινώντας από έναν αθροιστή 4 bit. Κάθε λύση με ελάχιστη προσεγγιστική αλυσίδα κρατουμένου μεγέθους $k = 3$ που να ικανοποιεί τους περιορισμούς αποθηκεύεται.

Η αναδρομική διαδικασία δημιουργίας αθροιστών θα περιγραφεί χρησιμοποιώντας το παράδειγμα του σχήματος 5.2. Σε αυτό το παράδειγμα θα ξικνήσουμε αυθαίρετα από τον αθροιστή 4 bit που απεικονίζεται ως (1). Στόχος μας είναι η δημιουργία κάθε έγκυρου αθροιστή 5 bit ξεκινώντας από αυτή την αρχική λύση. Για το παράδειγμα, ως έγκυρες λογίζονται οι λύσεις με μέγιστο αριθμό εξόδων μικρότερο η ίσο του 2, μέγιστο λογικό βάθος ίσο με 3 και ελάχιστη προσεγγιστική αλύσίδα κρατουμένου $k = 3$. Αυτό σημαίνει ότι ο τελευταίος τελεστής της στήλης 4 θα πρέπει να κοιτάει μέχρι τη θέση bit 2.

Τα παρακάτω βήματα εκτελούνται για κάθε λύση:

1. Βρίσκουμε το LSB του τελευταίου τελεστή της στήλης $n+1$. Το LSB είναι το λιγότερο σημαντικό bit που περιλαμβάνεται στον υπολογισμό κρατουμένου στη στήλη $n+1$. Εάν δεν έχουν προστεθεί ακόμη τελεστές, το LSB ισούται με $n+1$.

2. Δημιουργούμε το επόμενο σύνολο λύσεων προσθέτοντας τον νέο τελεστή στη στήλη $n+1$ και συνδέοντάς τον με τον τελεστή που βρίσκεται ακριβώς από πάνω του, και με κάθε πιθανό τελεστή της στήλης $LSB-1$.

3. Προχωράμε στο επόμενο σύνολο λύσεων και επαναλαμβάνουμε τη διαδικασία. Μία λύση δεν παράγει νέες λύσεις εάν είναι ακριβής, ή εάν ήδη παραβιάζει κάποιους περιορισμούς.

Η εκτέλεση των παρακάτω βημάτων παρουσιάζεται στο σχήμα 5.2, ξεκινώντας με τη λύση (1). Οι νέοι τελεστές θα προστεθούν στη στήλη 4, καθώς κατασκευάζουμε αθροιστές 5 bit. Καθώς δεν υπάρχουν ακόμη τελεστές στη στήλη 4, $LSB=4$ (βήμα 1). Για το βήμα 2, πρέπει να προσδιορίσουμε όλες τις έγκυρες θέσεις στη στήλη $LSB-1=3$ για τη σύνδεση του νέου τελεστή. Αυτές είναι στην είσοδο της στήλης 3 (λύση (2)), στην έξοδο του 1ου τελεστή της στήλης 3 (λύση (3)), και στην έξοδο του 2ου τελεστή της στήλης 3 (λύση (4)). Η λύση 4 είναι ακριβής (το κρατούμενο διαδίδεται μέχρι το 0 για όλες τις θέσεις bit), επομένως δεν δημιουργούνται νέες λύσεις 5 bit από αυτήν. Η διαδικασία επαναλαμβάνεται για όλες τις λύσεις.

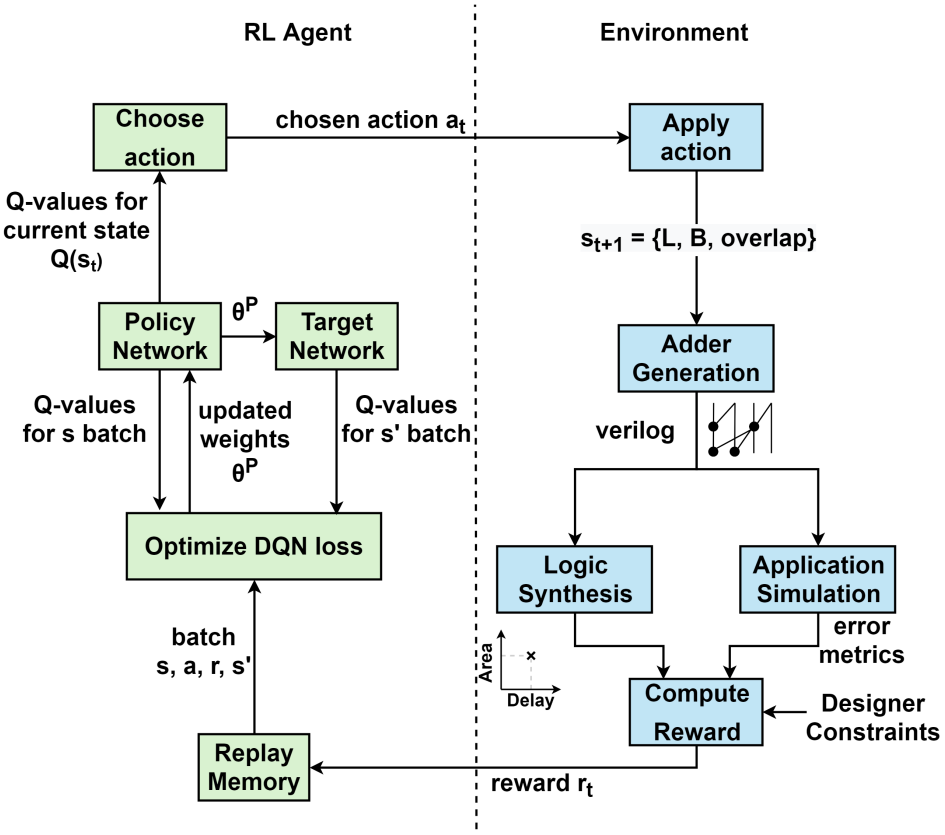
Παρατηρούμε ότι κάποιες λύσεις απορρίπτονται. Οι λύσεις (8) και (10) ξεπερνούν τον περιορισμό μέγιστου αριθμού εξόδων, η λύση (11) έχει 4 λογικά επίπεδα, ενώ η λύση (7) διαθέτει πολλαπλές μη βέλτιστες συνδέσεις. Για τη μείωση του χώρου λύσεων χρησιμοποιούνται δύο επιπλέον στρατηγικές απόρριψης αθροιστών. Διατηρούμε ένα συγκεκριμένο αριθμό λύσεων με τον ίδιο αριθμό bit, τελεστών, μέγιστου αριθμού εξόδων και λογικών επιπέδων, καθώς οι περισσότερες λύσεις με κοινά χαρακτηριστικά θα οδηγήσουν σε παρόμοιους αθροιστές. Επιπλέον, εάν ένας αθροιστής έχει πολύ περισσότερους τελεστές από τον μικρότερο έγκυρο αθροιστή του ίδιου μεγέθους bit, απορρίπτεται.

Στο τέλος της διαδικασίας, οι λύσεις οι οποίες ικανοποιούν τους περιορισμούς ορίζονται ως έγκυρες, και θα χρησιμοποιηθούν ως αφετηρία για τη δημιουργία αθροιστών 6 bit. Στο σχήμα 5.2 είναι οι λύσεις (3), (4), (5), (6) και (9). Σημειώνεται ότι ο αλγόριθμός μας παρέχει και την επιλογή διαχωρισμού του αθροιστή στα 2, με διαφορετικούς περιορισμούς ελάχιστης προσεγγιστικής αλυσίδας κρατουμένου σε κάθε τμήμα, χωρίς επικάλυψη.

5.3 Πλαίσιο ενισχυτικής μάθησης για τη σύνθεση προσεγγιστικών αθροιστών παράλληλου προθέματος

Ο στόχος του αλγορίθμου ενισχυτικής μάθησης είναι η εύρεση προσεγγιστικών αθροιστών οι οποίοι να ικανοποιούν τους περιορισμούς του χρήστη, οι οποίοι είναι ο αριθμός bit του αθροιστή, η εφαρμογή στην οποία θα χρησιμοποιηθεί, το σφάλμα του αθροιστή σε αυτή την εφαρμογή και η μέγιστη καθυστέρηση, εμβαδόν και αριθμός εξόδων του αθροιστή. Ο αλγόριθμος ενισχυτικής μάθησης ξεκινά τη μάθηση από την αρχή, με στόχο το σχεδιασμό ενός αποτελεσματικού αθροιστή που θα πληρεί τις παραπάνω προϋποθέσεις.

Η συνολική ροή της προτεινόμενης προσέγγισης παρουσιάζεται στο σχήμα 5.3. Ο αλγόριθμος ενισχυτικής μάθησης αποφασίζει ποια κίνηση να εφαρμόσει για την τροποποίηση



Σχήμα 5.3: Το πλαίσιο ενισχυτικής μάθησης για την προσαρμογή προσεγγιστικών αθροιστών παράλληλου προθέματος. Ο αλγόριθμος ενισχυτικής μάθησης επιλέγει μια κίνηση η οποία τροποποιεί τη δομή της προσέγγισης στον υπολογισμό κρατουμένου. Για αυτή τη δομή, συντίθεται ένας προσεγγιστικός αθροιστής παράλληλου προθέματος και υπολογίζεται η πολυπλοκότητα υλικού και η απόδοσή του στην επιλεγμένη εφαρμογή. Το αποτέλεσμα μετατρέπεται σε μια επιβράβευση με βάση τους περιορισμούς του χρήστη, η οποία οδηγεί την εκπαίδευση του αλγορίθμου.

του συνόλου των μεταβλητών $(L, B, overlap)$ που καθορίζουν την αρχιτεκτονική του κάθε αθροιστή. Για κάθε επιλεγμένο σύνολο μεταβλητών, συνθέτουμε έναν αθροιστή χρησιμοποιώντας τη μέθοδο της ενότητας 5.1, διατηρώντας τον αθροιστή με το μικρότερο αριθμό τελεστών με τα ελάχιστα δυνατά λογικά επίπεδα. Η πολυπλοκότητα υλικού του αθροιστή υπολογίζεται μετά από λογική σύνθεση. Παράλληλα, υπολογίζεται το σφάλμα στην επιθυμητή εφαρμογή. Τα αποτελέσματα εμβαδού, καθυστέρησης και σφάλματος χρησιμοποιούνται για τον υπολογισμό της επιβράβευσης, η οποία οδηγεί τον αλγόριθμο ενισχυτικής μάθησης σε βέλτιστες λύσεις.

Όπως για κάθε τυπικό πρόβλημα ενισχυτικής μάθησης, πρέπει να ορίσουμε την αναπαράσταση κατάστασης, τις επιτρεπτές κινήσεις και τη συνάρτηση επιβράβευσης. Η αναπαράσταση κατάστασης αποτελείται από 3 ακέραιους αριθμούς, το σύνολο $L, B, overlap$. Οι πιθανές κινήσεις είναι 6, και είναι η αύξηση/μείωση του $L/B/overlap$ κατά 1. Η συνάρτηση κόστους ισούται με το σταθμισμένο άθροισμα της διαφοράς καθυστέρησης, εμβαδού και σφάλματος του νέου αθροιστή σε σχέση με τον αθροιστή του προηγούμενου βήματος.

Η επιβράβευση r_t μετά την εφαρμογή της κίνησης a_t η οποία οδηγεί σε έναν αθροιστή ο οποίος ανιτστοιχεί στην κατάσταση s_{t+1} είναι ίση με:

$$r_t = a\,\Delta_{\text{delay}} + b\,\Delta_{\text{area}} + c\,\Delta_{\text{error}}, \tag{5.1}$$

Οι διαφορές σε καθυστέρηση και εμβαδόν, $\Delta_{\text{delay}} = \text{delay}_{t+1} - \text{delay}_t$ και $\Delta_{\text{area}} = \text{area}_{t+1} - \text{area}_t$, καθώς και το σφάλμα στην επιθυμητή εφαρμογή $\Delta_{\text{error}} = \text{error}_{t+1} - \text{error}_t$, αντιστοιχούν στις διαφορές μεταξύ των λύσεων του παρόντος και του προηγούμενου βήματος. Οι παράμετροι a, b και c χρησιμοποιούνται για να αλλάξουν την εστίαση της διαδικασίας βελτιστοποίησης, όπως επίσης και για την κανονικοποίηση των τιμών σε συγκρίσιμες μονάδες. Εάν ένας στόχος έχει ήδη επιτευχθεί τότε θέτουμε την Δ διαφορά του στο 0, ώστε ο αλγόριθμος να μην επιβραβευτεί για βελτιστοποίηση πέραν του επιθυμητού. Εάν ο αθροιστής που παράγεται ικανοποιεί όλους τους περιορισμούς, πολλαπλασιάζουμε την επιβράβευση με το 10, ώστε να οδηγήσουμε τον αλγόριθμο ενισχυτικής μάθησης προς αυτή τη λύση.

Ο αλγόριθμος ενισχυτικής μάθησης που χρησιμοποιείται είναι ο deep Q-network (DQN). Σύμφωνα με αυτό τον αλγόριθμο, εκπαιδεύουμε ένα νευρωνικό δίκτυο το οποίο λαμβάνει ως είσοδο την αναπαράσταση της τρέχουσας κατάστασης, και επιστρέφει ως έξοδο την αξία (Q-value) της κάθε πιθανής επόμενης κίνησης. Στην περίπτωσή μας, το νευρωνικό δίκτυο έχει 3 εισόδους (L, B, $overlap$), ένα κρυφό επίπεδο με 128 κόμβους, και το επίπεδο εξόδου με 6 κόμβους (αυξομείωση $L/B/overlap$ κατά 1). Τεχνικές όπως η χρήση μνήμης επανάληψης και δεύτερου νευρωνικού δικτύου που χρησιμοποιείται ως σταθερός στόχος ενσωματώθηκαν για τη βελτίωση της σύγκλισης.

5.4 Πειραματικά αποτελέσματα

Για την αξιολόγηση της προτεινόμενης προσέγγισης, συνθέσαμε προσεγγιστικούς αθροιστές παράλληλου προθέματος για τέσσερις διαφορετικές εφαρμογές, και τους συγκρίναμε με σύγχρονες δημοσιευμένες προσεγγίσεις. Για κάθε αθροιστή που θέλουμε να παραγάγουμε, δίνουμε διαφορετικούς περιορισμούς καθυστέρησης, εμβαδού και σφάλματος με βάση την απαιτούμενη εφαρμογή.

Όσον αφορά την ονοματολογία, οι 3 διαφορετικές προσεγγίσεις συμβολίζονται με *MCC*, *AxPPA* και *Approx*, ενώ οι αθροιστές της προσέγγισής μας συμβολίζονται με *RL*. Σε κάθε εφαρμογή επιλέγουμε να συγκριθούμε με τον πιο αποδοτικό αθροιστή της κάθε προσέγγισης.

5.4.1 Φίλτρο μέσου όρου εικόνας

Η εφαρμογή φίλτρου μέσου όρου εικόνας χρησιμοποιείται για την εξομάλυνση εικόνων 8 bit, υπολογίζοντας το μέσο όρο των pixel σε ένα παράθυρο γύρω από κάθε pixel. Στην υλοποίησή μας χρησιμοποιήθηκε ένα παράθυρο μεγέθους 11×11, άρα ο μέσος όρος 121 γειτονικών pixel υπολογίζεται για κάθε pixel εξόδου χρησιμοποιώντας αθροιστές 16 bit. Η αξιολόγηση του σφάλματος πραγματοποιήται υπολογίζοντας το Peak Signal-to-Noise Ratio (PSNR) και το Mean Structural Similarity (MSSIM), δύο δημοφιλής μετρικές για την ποσοτικοποίησης της διαφοράς της προσεγγιστικής εικόνας σε σχέση με το αποτέλεσμα όπου χρησιμοποιούνται ακριβείς αθροιστές.

Τα αποτελέσματα παρουσιάζονται στον πίνακα 5.1. Για αυτή την εφαρμογή, ο αθροιστής MCC-$(1, 3)$ είναι βέλτιστος, καθώς προήλθε από πλήρη απαρίθμηση προσεγγιστικών αθροιστών και επιλέχθηκε για την εφαρμογή μέσου όρου εικόνας. Παρομοίως, ο AxPPA-4

Πίνακας 5.1: Η απόδοση των προσεγγιστικών αθροιστών υπό σύγκριση για την εφαρμογή φίλτρου μέσου όρου εικόνας σε σχέση με το ακριβές αποτέλεσμα χρησιμοποιώντας τις μετρικές PSNR και MSSIM.

Αθροιστής	**Μετρική**	**Εικόνα**						**Εμβαδόν (μm^2)**	**Ισχύς (nW)**
		Apple	**Flower**	**Cherry**	**Horse**	**House**	**Avg**		
Approx-HC	PSNR	17.38	16.92	15.89	15.50	15.29	16.20	93	34
	MSSIM	0.26	0.37	0.10	0.10	0.17	0.20		
MCC-$(1,3)$	PSNR	37.11	37.25	37.11	37.08	36.91	37.09	86	28
	MSSIM	0.95	0.97	0.92	0.91	0.75	0.90		
AxPPA-4	PSNR	37.28	37.44	37.26	37.24	37.06	37.26	86	30
	MSSIM	0.98	0.99	0.95	0.95	0.79	0.93		
RL-$(4,1,1)$	PSNR	37.11	37.25	37.11	37.08	36.91	37.09	86	29
	MSSIM	0.95	0.97	0.92	0.91	0.75	0.90		

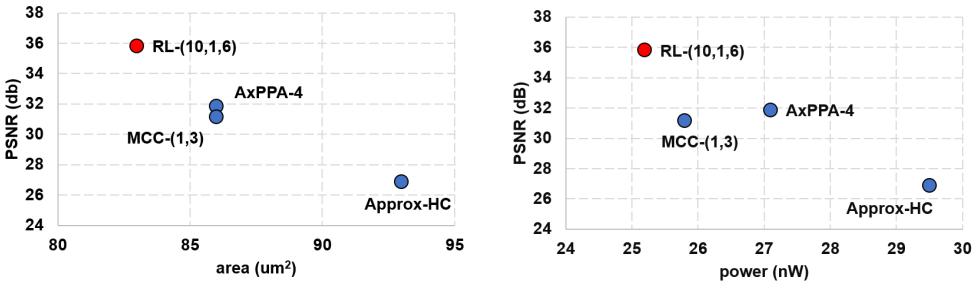
είναι εξίσου αποτελεσματικός, εφόσον έχει παρόμοια δομή με τον MCC-$(1,3)$. Με αυτό το δεδομένο, είναι αναμενόμενο το ότι η μέθοδος ενισχυτικής μάθησης δεν θα μπορέσει να βρει έναν αποδοτικότερο αθροιστή. Παρόλα αυτά, ο RL-$(4,1,1)$ έχει τις ίδιες ακριβώς αλυσίδες κρατουμένου με τον MCC-$(1,3)$, και συνεπώς παρόμοια απόδοση. Αυτό αποδεικνύει την ικανότητα της μεθόδου ενισχυτικής μάθησης στο να βρίσκει εξίσου αποδοτικούς αθροιστές με βέλτιστες ντετερμινιστικές μεθόδους.

5.4.2 Φίλτρο Laplace

Το φίλτρο Laplace εφαρμόζει το παρακάτω 3×3 φίλτρο σε μια εικόνα με στόχο την ανίχνευση ακμών:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Το κάθε pixel της εικόνας πολλαπλασιάζεται με τον αντίστοιχο συντελεστή του φίλτρου, και τα γινόμενα αθροίζονται με τη χρήση των προσεγγιστικών αθροιστών.



(**a**) (**b**)

Σχήμα 5.4: Γράφημα του μέσου PSNR σε σχέση με **(a)** το εμβαδόν και **(b)** την ισχύ των προσεγγιστικών αθροιστών υπό εξέταση για την εφαρμογή του φίλτρου Laplace.

Τα αποτελέσματα του μέσου όρου PSNR σε σχέση με την πολυπλοκότητα υλικού παρουσιάζονται στην εικόνα 5.4, όπου φαίνεται ότι ο αθροιστής μας έχει το υψηλότερο PSNR,

ενώ ταυτόχρονα έχει και το χαμηλότερο εμβαδόν και κατανάλωση ενέργειας. Ο προτεινόμενος αθροιστής RL-$(10, 1, 6)$ πετυχαίνει 33% καλύτερο PSNR σε σχέση με τον Approx-HC, ενώ ταυτόχρονα μειώνει το εμβαδό και την ισχύ κατά 11% και 17% αντίστοιχα. Σε σχέση με τους MCC-$(1, 3)$ και AxPPA-4, ο αθροιστής μας βελτιώνει το μέσο PSNR κατά 12%, ενώ βελτιώνει το εμβαδόν κατά 3% και πετυχαίνει παρόμοια κατανάλωση ενέργειας με τον MCC-$(1, 3)$ και 7% λιγότερη κατανάλωση ενέργειας από τον AxPPA-4.

5.4.3 Εσωτερικό γινόμενο διανυσμάτων

Το διανυσματικό εσωτερικό γινόμενο πολλαπλασιάζει τα στοιχεία δύο διανυσμάτων και τα προσθέτει χρησιμοποιώντας προσεγγιστικούς αθροιστές. Για τα διανύσματα c και x^T, ο υπολογισμός είναι ο εξής:

$$S = \begin{bmatrix} c_0 & c_1 & \dots & c_n \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix} = c_0x_0 + c_1x_1 + \dots c_nx_n$$

Στην υλοποίησή μας χρησιμοποιούμε τυχαίες σταθερές τιμές για το διάνυσμα c, και τυχαίες μεταβλητές προσημασμένες τιμές 12 bit για το διάνυσμα x^T. Η μετρική σφάλματος που χρησιμοποιούμε για την αξιολόγηση της απόδοσης των αθροιστών είναι η συχνότητα σφάλματος (EF), η οποία ποσοτικοποιεί το πόσο συχνά το τελικό αποτέλεσμα του εσωτερικού γινομένου είναι λανθασμένο.

Πίνακας 5.2: Η απόδοση όσον αφορά τη συχνότητα σφάλματος και την πολυπλοκότητα υλικού των δημοσιευμένων αθροιστών και δύο επιλογών αυτόματα δημιουργημένων από την προσέγγισή μας για την εφαρμογή του εσωτερικού γινομένου.

Αθροιστής	**#Κόμβων**	**Εμβαδόν (μm^2)**	**Ισχύς (nW)**	**EF**
Approx-HC	72	211	72	0.0109
MCC-$(21, 0)$	85	220	75	0.0002
AxPPA-2	87	242	82	0.6593
RL-$(20, 3, 15)$	70	199	71	0.0055
RL-$(27, 3, 21)$	85	218	73	0.0002

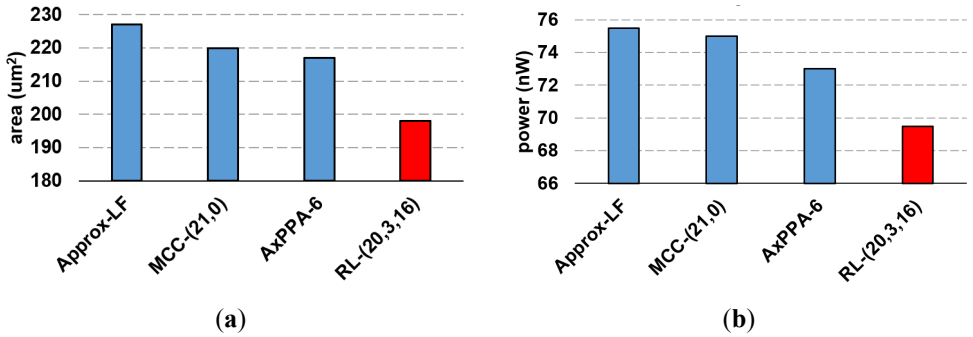
Για αυτή την εφαρμογή παραγάγαμε δύο αθροιστές, και τα αποτελέσματα παρουσιάζονται στον πίνακα 5.2. Ο πρώτος αθροιστής, ο RL-$(20, 3, 15)$, έχει υψηλότερο σφάλμα και χαμηλότερο εμβαδόν και κατανάλωση ενέργειας και συγκρίνεται με τους Approx-HC και AxPPA-2. Πετυχαίνει 50% χαμηλότερη συχνότητα σφάλματος, 6% χαμηλότερο εμβαδόν και 1% χαμηλότερη κατανάλωση ενέργειας σε σχέση με τον Approx-HC, ενώ πετυχαίνει 18% βελτίωση στο εμβαδόν και 13% στην κατανάλωση ενέργειας σε σχέση με τον AxPPA-2, ενώ μειώνει σημαντικά τη συχνότητα σφάλματος.

Ο δεύτερος αθροιστής, ο RL-$(27, 3, 21)$ στοχεύει σε πολύ μικρή συχνότητα σφάλματος και συγκρίνεται με τον MCC-$(21, 0)$. Ο αθροιστής ενισχυτικής μάθησης πετυχαίνει την ίδια συχνότητα σφάλματος, ενώ βελτιώνει το εμβαδό και την ισχύ κατά 1% και 3% αντίστοιχα. Η

μικρή βελτίωση είναι αναμενόμενη καθώς ο αθροιστής MCC-$(21, 0)$ είναι σχεδόν ακριβής, δίνοντας μικρό περιθώριο για βελτιστοποίηση.

5.5 Νευρωνικό Δίκτυο

Για την τελευταία εφαρμογή σχεδιάσαμε ένα νευρωνικό δίκτυο το οποίο αποτελείται από το επίπεδο εισόδου με 784 νευρώνες, ένα κρυφό επίπεδο με 500 νευρώνες, και το επίπεδο εξόδου με 10 νευρώνες. Το δίκτυο έχει εκπαιδευτεί για την αναγνώριση ψηφίων του συνόλου δεδομένων MNIST χρησιμοποιώντας 50.000 28×28 εικόνες, με στόχο την ελαχιστοποίηση σφάλματος ταξινόμησης. Οι προσεγγιστικοί αθροιστές 32 bit χρησιμοποιούνται στον υπολογισμό των αθροισμάτων γινομένων στο επίπεδο εξόδου. Το σφάλμα ταξινόμησης με την χρήση ενός απόλυτα ακριβούς αθροιστή είναι 96.82%.



(**a**) (**b**)

Σχήμα 5.5: (**a**) Το εμβαδόν και (**b**) η κατανάλωση ενέργειας όλων των αθροιστών υπό σύγκριση για την περίπτωση του νευρωνικού δικτύου για αναγνώριση ψηφίων.

Για αυτή την εφαρμογή, όλοι οι αθροιστές οι οποίοι συγκρίνονται πετυχαίνουν σφάλμα ταξινόμησης ίδιο με αυτό της ακριβούς άθροισης. Τα αποτελέσματα εμβαδού και ισχύος παρουσιάζονται στην εικόνα 5.5. Παρατηρούμε ότι ο αθροιστής ενισχυτικής μάθησης RL-$(20, 3, 16)$ βελτιώνει το εμβαδόν κατά 13% και την ισχύ κατά 8% σε σχέση με τον Approx-LF, το εμβαδόν κατά 10% και την ισχύ κατά 7% συγκριτικά με τον MCC-$(21, 0)$, και τέλος, το εμβαδόν κατά 9% και την ισχύ κατά 5% σε σχέση με τον AxPPA-6.

6 Συμπεράσματα

Αυτή η διατριβή αντιμετωπίζει δύο κρίσιμες προκλήσεις στη σύγχρονη αυτοματοποιημένη ψηφιακή σχεδίαση υλικού: την επίτευξη κλεισίματος χρονισμού με αποτελεσματικό τρόπο και τη σύνθεση προσαρμοσμένων αριθμητικών μονάδων για την ικανοποίηση των αναγκών συγκεκριμένων εφαρμογών.

Η πρώτη βασική συνεισφορά αυτής της διατριβής βρίσκεται στο πεδίο της χρονικής βελτιστοποίησης στη φυσική σύνθεση. Προτείνουμε μια αυτόνομη ροή η οποία ενσωματώνει μοντελοποίηση κόστους βασισμένη σε Lagrangian Relaxation (LR) με ένα πλαίσιο Multi-Armed Bandit (MAB) για την καθοδήγηση της επιλογής τοπικών μετασχηματισμών κυκλωμάτων. Στην πορεία τριών εργασιών, η ροή βελτιστοποίησης εξελίχθηκε από ένα απλό σύστημα καθοδηγούμενο από MAB σε μια πλήρως συνεκτική μεθοδολογία LR, όπου ο κάθε μετασχηματισμός συνεισφέρει σε έναν ενοποιημένο στόχο βελτιστοποίησης. Αυτή η συνεργασία επιτρέπει στο μοντέλο MAB να δώσει προτεραιότητα στις πιο αποτελεσματικές κινήσεις σε κάθε επανάληψη, οδηγώντας σε καλύτερη προσαρμοστικότητα, βελτιωμένη σύγκλιση, και υψηλότερη ποιότητα αποτελεσμάτων σε μικρότερο χρόνο εκτέλεσης. Συνδυάζοντας την επιλογή μετασχηματισμών μέσω MAB με μια γενική συνάρτηση κόστους βασισμένη σε LR, η ροή βελτιστοποίησης πετυχαίνει σημαντικές χρονικές βελτιώσεις χωρίς την ανάγκη για χειροκίνητες παρεμβάσεις ή προσαρμογές με βάσει το κύκλωμα. Αυτή η πρόοδος δείχνει το όφελος της ενσωμάτωσης της προσαρμοστικότητας που προκύπτει από τις μεθόδους μάθησης στις παραδοσιακές ντετερμινιστικές ροές βελτιστοποίησης, επιτρέποντας στο σύστημα να ισορροπήσει πολλαπλούς στόχους, όπως κλείσιμο χρονισμού, ελαχιστοποίηση ισχύος και εμβαδού.

Στην δεύτερη βασική συνεισφορά, αυτή η διατριβή εξερευνά τη σχεδίαση προσεγγιστικών αριθμητικών μονάδων για συγκεκριμένες εφαρμογές χρησιμοποιώντας ενισχυτική μάθηση. Προτείνουμε τη χρήση ενός πλαισίου οδηγούμενο από ενισχυτική μάθηση για τη δημιουργία προσεγγιστικών αθροιστών παράλληλου προθέματος, όπου ο αλγόριθμος μαθαίνει δομικές στρατηγικές σχεδιασμού βελτιστοποιημένες τόσο για μετρικές υλικού (ισχύς, εμβαδόν) όσο και για την απόδοση σφάλματος σε επίπεδο εφαρμογών. Σε αντίθεση με παραδοσιακές προσεγγίσεις όπου χρησιμοποιούνται συγκεκριμένες αρχιτεκτονικές, η μέθοδός μας πραγματοποιεί αναδρομική σύνθεση βασισμένη σε δομικές παραμέτρους, επιλεγμένες από το πλαίσιο ενισχυτικής μάθησης, προσφέροντας πλουσιότερη και πιο ευέλικτη εξερεύνηση του χώρου λύσεων.

Συνολικά, οι μέθοδοι που προτείνουμε σε αυτή τη διατριβή επιδεικνύουν τη δύναμη και την ευελιξία της ενισχυτικής μάθησης ως εργαλείο στην αυτοματοποιημένη σχεδίαση υλικού. Εφαρμόζοντας τεχνικές ενισχυτικής μάθησης σε διαφορετικά επίπεδα της ροής σχεδιασμού, όπως η βελτιστοποίηση κυκλωμάτων στη ροή φυσικής σχεδίασης και η σύνθεση προσεγγιστικών κυκλωμάτων, αυτή η εργασία υπογραμμίζει το πως μοντέλα βασισμένα στη μάθηση μπορούν να ξεπεράσουν σε ποιότητα σταθερές και χειροκίνητα κατασκευασμένες μεθόδους και στρατηγικές. Αυτές οι συνεισφορές ανοίγουν το δρόμο για πιο έξυπνα, προσαρμοστικά και αυτόματα εργαλεία φυσικής σχεδίασης, τα οποία θα είναι καλύτερα εξοπλισμένα για να ανταποκριθούν στην αυξανόμενη πολυπλοκότητα και τις απαιτήσεις απόδοσης της επόμενης γενιάς ψηφιακών ολοκληρωμένων κυκλωμάτων.