ABSTRACT

Title of Dissertation: SYSTEM LEVEL DESIGN FOR EMERGING SECURITY AND 3D IC TECHNOLOGIES

Daniel H Xing
Doctor of Philosophy, 2025

Dissertation Directed by: Professor Ankur Srivastava
Department of Electrical and
Computer Engineering

Electronic design at the system level enables greater flexibility and quality impact over design algorithms that work with more detailed design abstractions. In this dissertation, new system level methods are applied to two areas in integrated circuit security and design that are typically addressed at the circuit or physical design stages: logic locking and three-dimensional integrated circuits (3D ICs).

Stripped functionality logic locking (SFLL) is one of the leading logic locking methods in the current literature. While SFLL and its variations have good mathematical guarantees against resilience to various attacks, at its core SFLL is a circuit level technique and hence a naive implementation does not depend on system level details. Additionally, implementations of SFLL can come with high overhead, in particular the restore units needed to fully implement SFLL. We show that extending SFLL beyond circuit level boundaries and into the system design

level through resource sharing and reuse significantly reduces SFLL implementation overhead. In particular, we examine two system design approaches that enable sharing at the system level to lower overall locking implementation overhead, as well as a clock gating method that relies on strategic application of SFLL at the datapath architecture level in order to reduce total system power.

3D ICs present a new dimension in the design and packaging of ICs and promise increased density, shorter wirelength, and improved performance. However, the complex interplay between placement and PPA increases substantially, particularly in regards to inter-tier TSVs needed for inter-tier connectivity, complicating the design process. In this dissertation, we describe three co-design approaches to 3D global placement and datapath architecture synthesis via HLS for timing, dynamic power, and TSV area optimization. We show that leveraging the greater flexibility afforded by the system-level perspective of HLS in the loop with feedback from global placement along with providing global placement with design details from HLS significantly enhances overall design quality. Our results show that our co-design approaches significantly reduces total negative slack (TNS), dynamic power consumption, and total TSV usage over conventional methods.

SYSTEM LEVEL DESIGN FOR EMERGING SECURITY AND 3D IC TECHNOLOGIES

by

Daniel H Xing

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2025

Advisory Committee:
Professor Ankur Srivastava, Chair/Advisor
Professor Manoj Franklin
Professor Donald Yeung
Professor Cunxi Yu
Professor Bongtae Han

Dedication

To my family, for encouraging and standing by me throughout this journey.

Acknowledgments

Firstly, I would like to thank my advisor, Professor Ankur Srivastava, for his unwavering guidance and support throughout my Ph.D. study. I'm extremely grateful for his patience, motivation and immense knowledge in all the time we spent on solving challenging research problems. He has always guided me to the correct path and supported my choices, which made my Ph.D. experience incredibly rewarding and productive. The incredible enthusiasm he has for his research will always be a great motivation for me. It is truly a pleasure to work with and learn from him.

I would like to express my gratitude to Professor Manoj Franklin, Professor Donald Yeung, Professor Cunxi Yu, and Professor Bongtae Han for their time to serve on my dissertation committee and for providing valuable feedback to improve this dissertation.

I would also like to extend my thanks to all of my colleagues in Professor Srivastava's research group. My thanks first go to four senior colleagues Ankit Mondal, Abhishek Chakraborty, Yuntao Liu, and Michael Zuzak for showing me how to conduct research, set up experiments and write papers in the early stage of my Ph.D. study. Without them, I would not have been able to adapt to the Ph.D. lifestyle so quickly. Thanks also go to my current colleagues Issac McDaniel, Olsan Ozbay, Abir Akib, and Mumtahina Sukanya for the inspiring research discussions, for the late nights we were working together before deadlines, and for all the fun we have had in our lab.

I would also like to thank the many friends and talented musicians of the UMD Gamer Symphony Orchestra, whose energy and spirit helped push my research forward and for being my second home at UMD for the past 10 years. I'd also like to thank Ting-Chen Shi for providing the freshly roasted coffee that fueled much of this dissertation, Kane Hsu for the many long walks and shared meals that helped ground me when I needed it most, and Christine Zhou for the constant support and motivation.

Finally, I owe the deepest gratitude to my family. I want to express my appreciation to my parents for their endless love, devotion, and support.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

3D IC	Three Dimensional Integrated Circuit
ALAP	As Late As Possible
ASAP	As Soon As Possible
DEF	Design Exchange Format
DFG	Dataflow Graph
FSC	Functionality Stripped Circuit
GDSII	Graphics Design System 2 (File Format)
HI	Heterogeneous Integration
HLS	High Level Synthesis
HPWL	Half-Perimieter Wirelength
IC	Integrated Circuit
ILP	Integer Linear Program
KCL	Kirchhoff's Current Law
LEF	Library Exchange Format
LP	Linear Programming
LSE	Log-Sum Exponential (Approximation)
LUT	Lookup Table
MILP	Mixed-Integer Linear Program
MLIR	Multi-Level Intermediate Representation
MSAG	Mutual Switching Activity Graph
MUX	Multiplexer
OP	Operation
PCB	Printed Circuit Board
PIP	Protected Input Pattern
PPA	Power, Performance, Area
RTL	Register-Transfer Level
RU	(SFLL) Restore Unit
SA	Simulated Annealing
SAT	Boolean Satisfiability

SDFG	Scheduled Dataflow Graph
SFLL	Stripped Functionality Logic Locking
STA	Static Timing Analysis
SiP	System in Package
TNS	Total Negative Slack
TSV	Through-Silicon Via
WA	Weighted Average (HPWL Approximation)
WNS	Worst Negative Slack
XOR	Exclusive-OR

List of Publications

Journals:

1. Md Moshiur Rahman, Jim Geist, **Daniel Xing**, Yuntao Liu, Ankur Srivastava, Travis Meade, Yier Jin, and Swarup Bhunia. "Security Evaluation of State Space Obfuscation of Hardware IP through a Red Team-Blue Team Practice." *ACM Transactions on Design Automation of Electronic Systems*, Volume 29, Issue 3, Article 50, pp 1-18, 2024.

2. Yuntao Liu, **Daniel Xing**, Isaac McDaniel, Olsan Ozbay, Abir Akib, Mumtahina Islam Sukanya, Sanjay Rekhi, and Ankur Srivastava "Security Advantages and Challenges of 3D Heterogeneous Integration" in *Computer*, vol. 57, no. 03, pp. 107-112, 2024.

Book Chapters:

1. Yuntao Liu, Ankit Mondal, Abhishek Chakraborty, Michael Zuzak, Nina Jacobson, **Daniel Xing**, and Ankur Srivastava, "Neural Trojans," in Encyclopedia of Cryptography, Security and Privacy, 2021

Conferences:

1. **Daniel Xing**, Abir Akib, Yuntao Liu, and Ankur Srivastava. "Co-design for Heterogeneous Integration: High Level Decisions to the Rescue." *2024 IEEE 67th International Midwest Symposium on Circuits and Systems (MWSCAS).* IEEE, 2024.

2. **Daniel Xing**, and Ankur Srivastava. "3D-Aware Low Power High-level Resource Binding and Co-Design." (Accepted) *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED).* ACM, 2024.

3. **Daniel Xing**, and Ankur Srivastava. "A High Level Approach to Co-Designing 3D ICs." *61st ACM/IEEE Design Automation Conference.* 2024.

4. **Daniel Xing**, Yuntao Liu, and Ankur Srivastava. "Low Power Logic Obfuscation Through System Level Clock Gating," *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED).* IEEE, 2023

5. **Daniel Xing**, Michael Zuzak, and Ankur Srivastava. "Low Overhead System-Level Obfuscation through Hardware Resource Sharing." *2023 24th International Symposium on Quality Electronic Design (ISQED).* IEEE, 2023.

6. Yuntao Liu, Michael Zuzak, **Daniel Xing**, Isaac McDaniel, Priya Mittu, Olsan Ozbay, Abir Akib, and Ankur Srivastava. "A Survey on Side-Channel-based Reverse Engineering Attacks on Deep Neural Networks." *2022 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS).* IEEE, 2022.

7. Yuntao Liu, Ankit Mondal, Abhishek Chakraborty, Michael Zuzak, Nina Jacobsen, **Daniel Xing**, and Ankur Srivastava "A Survey on Neural Trojans." *2020 21st International Symposium on Quality Electronic Design (ISQED).* IEEE, 2020.

Posters:

1. **Daniel Xing** and Ankur Srivastava. "System Level Design for Hardware Security and 3D ICs" *Design Automation Conference.* 2024.

Under Review:

1. **Daniel Xing**, and Ankur Srivstava. "Architectural-Physical Co-Design for TSV Area, Latency, and Timing Closure in Stacked 3D ICs."

2. **Daniel Xing**, Michael Zuzak, and Ankur Srivstava. "Contention-Free Low Overhead System-Level Logic Locking with Bubble Insertion." *ACM Transactions on Design Automation of Electronic Systems.*

Chapter 1: Introduction

Modern design and verification of integrated circuits (ICs) is a complex process, especially at the latest technology nodes. To cope with this complexity, the design process has been divided into steps, starting from high level specifications and going all the way down to individual transistor layout. As the design flow progresses, the design aspects under consideration become more finer grained. Consequently, decisions made at an earlier, higher, and more coarse-grained design step can affect large changes in quality of the overall IC while decisions at lower level steps tend to have lesser impact on the overall design.

Many techniques and methods for IC design that solve specific problems are specific to just one stage of the overall design flow. For example, logic locking is typically applied during RTL or circuit design, and 3D integration is typically considered during physical design. This is partly due to where each of the respective methods originated: logic locking was originally envisioned as a circuit level technique that randomly inserted XOR gates into a design [56] while 3D IC considerations start with physical placement and partitioning [71, 44]. However, confining these methods to later lower-level design stages can reduce their effectiveness. We propose examining traditionally lower level design methods from the system perspective of high level synthesis (HLS) for the following problems:

- secure and low overhead implementations of stripped functionality logic locking (SFLL)
- timing-valid design of three dimensional integrated circuits (3D ICs)

1.1 Preliminaries

1.1.1 High Level Synthesis

The goal of HLS is to transform a high level language description, such as C, MATLAB, or a design specific language (PyTorch, Chisel, etc), to RTL. The high level description is parsed into an intermediate representation so that dataflow optimizations (e.g. loop unrolling) and code transformations (e.g. constant propagation, operation strength reduction) can be applied, much like what occurs in a software compiler[1]. After compiler optimizations, the final intermediate representation can be decomposed into a set of dataflow graphs (DFGs) $G(V, E)$, where nodes $o \in V$ are operations and edges are data dependencies between operations.

Traditionally, HLS consists of three main stages: operation scheduling, resource allocation, and resource binding, not necessarily in that order. Scheduling determines when operations in the DFG execute along clock boundaries and typically try to optimize for latency and/or satisfy timing and resource constraints. Resource allocation determines the amount and quality of functional resources that these operations will actually run on, and can depend on the technology and total resource

[1]Many modern HLS front ends are built on top of open source compiler infrastructures like LLVM [32] and MLIR [33, 82, 2, 83]

limits of the design. Binding maps operations from the DFG onto allocated resources and can take into account resource sharing and power constraints, as well as any pipelining. Any of these steps can be performed in any order or even simultaneously with any other step.

1.1.2 Hardware Security and Stripped Functionality Logic Locking

As a result of the globalization of the IC supply chain, many entities are involved in the design, manufacture, and testing of chips. While necessary to cope with the ever-increasing complexity of modern devices, this opens up ICs to supply chain attacks. These attacks, which can be mounted by third party adversaries anywhere along the supply chain, render chips vulnerable to overproduction, Trojan insertion, and piracy [55].

Many hardware defenses against these attacks have been proposed, including IC watermarking [29, 23], Trojan detection methods [7], logic locking [8], chip metering [4, 30], and split manufacturing [22, 21]. This dissertation we focus specifically on logic locking approaches.

Logic locking seeks to obfuscate some or all of a circuit behind a locking mechanism such that the design is rendered nonfunctional unless a correct key value is provided [56, 53, 54, 77, 73, 80, 60, 61, 76]. Surveys on logic locking can be found in [8, 16].

Attacks on logic locking include sensitization attacks [53], structural attacks [63, 78], and machine-learning based attacks [69, 9]. One of the most prominent attacks is the SAT attack [67], which was able to circumvent all defense techniques when it was first introduced, and still remains a foundational benchmark against which new logic locking methods are evaluated against today. SAT attacks formulate logic locking key retrieval as an iterative chain of boolean satisfiability (SAT) problems. Each iteration of boolean SAT tries to find an input value for which two possible key values produce differing output values. Since only one of the output values can be correct, at least one of the possible keys must be incorrect. In practice multiple keys can be eliminated per iteration of the SAT attack.

One of the most prominent logic locking methods are those based on stripped-functionality logic locking (SFLL) [79, 59, 58]. SFLL modifies a circuit to be protected by selectively stripping functionality for a small number of protected input patterns (PIPs) such that the output is incorrect for those PIPs. To correct corrupted outputs, a restore unit (RU) compares the input of a locked module against a key value and applies a correction signal if a match is found. The general structure of SFLL is shown in figure 2.1. The correct key value is simply the protected input patterns that were stripped from the circuit. SFLL was designed to be secure against the powerful SAT attack [5, 67], and has since been extended to defend against removal attacks [45, 78], and structural [12] attacks. SFLL is also highly configurable [81] and has been implemented for many real-world designs [68].

1.1.3 Three Dimensional Integrated Circuits

3D ICs promises to elevate chips to higher packaging densities and better interconnects by stacking multiple layers of ICs together. 3D ICs offer numerous advantages including higher packaged transistor densities (even in the face of the end of Moore's Law) [13], reduced routing congestion due to the vertical nature of 3D vias, and improved interconnect delays and power due to shorter wirelengths [6].

3D IC packaging can take many forms, and can be broadly broken down into three categories: face-to-back, face-to-face, and monolithic [34]. Face-to-back 3D integration etches TSVs through the substrate and into the active device layer of the bottom die in order to facilitate die-to-die connections, but suffer from high parasitics since TSVs need to pass through the entire depth of the silicon substrate. Face-to-face stacking avoids the need for TSVs by stacking metal layers directly against each other such that smaller and shorter vias can be used at the cost of limiting 3D stacks to just two dies. Monolithic 3D ICs directly pattern and fabricate transistors without the need for carrier substrates and have the lowest parasitic capacitances at the cost of increased manufacturing complexity [15]. We note that while heterogeneous integration of dies from different process and manufacturing methods means a single package could use any or all of these 3D integration methods [35], we limit our work to 3D IC stacks integrated using the same method.

1.2 Existing System-Level Approaches for SFLL and 3D ICs

There has been some existing work on logic locking at the system level. [52] proposes locking control paths in C dataflows before high level synthesis (HLS) is applied. SFLL-HLS [81] proposes analyzing the intermediate representation emitted by a high-level language compiler for suitable combinational logic cones for locking with SFLL. [87] proposes a security-aware resource binding technique for maximizing the effective error rate of each SFLL-locked module in a system design. However while all of these approaches incorporate system level information in their design decisions, locking is still implemented at the circuit level.

HLS for 3D ICs is much more limited. One work in the literature is 3DHLS [10], which integrates HLS into simulated annealing (SA)-based floorplanning by treating HLS as a potential action to modify the area/power/timing characteristics of each floorplanned block, but does not otherwise make HLS or the SA floorplanner aware of the other. [36] and [70] propose system level techniques for minimizing TSVs between layers of a 3D IC, but do so without considering physical placement of actual hardware modules.

1.3 Our Work: System Level Perspectives, Designs, and Optimization on Hardware Security Primitives and 3D ICs

The objective of this dissertation is to show how a HLS-based system perspective and design optimization strategy can greatly improve on existing logic locking and 3D IC technologies.

1.3.1 System Level Design and Perspectives on SFLL Resource Sharing

The main focus of SFLL research has been to improve and harden the functionality-stripped circuit (FSC) against attack. However, the restore unit (RU) has not had as much attention as it is only responsible for correcting functionality. Indeed, structural attacks assume the restore unit is easily identifiable [12]. However, prior works do not go beyond circuit level implementations of SFLL and likewise assume each RU is tied to a specific FSC. In our work we propose multiple methods of decoupling SFLL RUs from FSCs and treat them as a high level resource. Since each RU's purpose is to simply compare the FSC's input and emit a correction signal if a match is found, the comparison need not be limited to a single unique FSC and can thus be shared across multiple locked modules. We propose two approaches methods to sharing RUs with different design goals: one that maximizes system corruption when incorrect keys are given to a locked design, and one that has less stringent

requirements on the information needed to implement sharing. Additionally, we propose utilizing RUs as precomputed lookup tables (LUTs) for clock gating entire modules at the system level. All approaches come with mathematically rigorous approaches to finding optimal system implementations alongside efficient heuristics.

1.3.2 Co-design of System and Physical Design for 3D ICs

3D IC design automation methods have commonly focused on lower level physical design, without as much consideration on higher level system or architectural design since TSVs are usually placed during physical design. However, since TSVs generally have high parasitic capacitance, their impact on overall design quality is higher than regular vias, especially with regards to timing and power. Additionally, TSVs needed for face-to-back 3D integration technologies take up logic area on the die, and thus can have significant area overheads if not properly accounted for. Our work proposes considering these physical placement details during HLS by co-optimizing physical placement with architecture synthesis. Our co-design approaches has two main components: a HLS-level approach to datapath architecture synthesis that incorporates module distances and shadows when given a 3D placement of datapath modules, and dynamic net weighting for 3D module placement that guides a 3D placer towards timing-valid and power optimal module placements. We show how co-design done in this way can be used to optimize the power, performance, and area (PPA) of 3D ICs using mathematically rigorous optimization programs.

1.4 Organization of the Dissertation

The rest of this dissertation is organized as follows. Chapter 2 discusses a system level perspective to sharing SFLL restore units for maximal system level disruption under wrong keys. Chapter 3 presents an alternative restore unit sharing approach that relaxes some of the stringent requirements presented in Chapter 2 in exchange for some timing overhead. Chapter 4 describes a method to reuse SFLL restore units as LUTs for clock gating in order to reduce total dynamic power. Chapter 5 presents initial work on co-designing a datapath architecture in the loop with 3D module placement in order to reduce timing violations. Chapter 6 builds on the theoretical framework of the previous chapter to co-optimize architecture and physical design to lower dynamic power consumed by modules and interconnects, with timing as a constraint. Chapter 7 further expands the work of chapter 5 to consider an additional architectural degree of freedom (scheduling) in order to enable higher TSV utilization and therefore lower TSV area usage while ensuring placed TSVs do not overlap placed modules. Chapter 8 concludes this dissertation and discusses potential avenues of future work on addressing challenges in both security and 3D ICs with an HLS perspective.

Chapter 2: Low Overhead System-Level Resource Sharing for SFLL

In this chapter, we explore our first approach to augmenting SFLL by taking an architectural approach to locking a design. Instead of considering each locked functional module in isolation when making locking decisions, we show that restore units can be shared across multiple locked modules, enabling a locking scope that expands *beyond gate level boundaries*. While a high level approach to logic locking is not unique [87, 51, 47, 50, 40], this work is the first to explore utilizing restore units as a *shareable architectural resource* that can be bound to multiple locked modules in a design. This high level view affords the designer significant flexibility in reducing area and power overheads as well as allowing for a mathematically rigorous and tunable optimization scheme that gives granular control over the error-overhead trade-off to the engineer.

We present two mathematically-rigorous methods to explore sharing. Both methods come with tunable gate-level SAT attack resilience and guarantee that all stripped functionality will be properly restored given the correct key. Our contributions are summarized as follows:

1. We define a formal mathematical model for restore unit sharing in the form of a 0-1 integer linear program (ILP). Our formulation finds solutions that meet resource contention, attack resilience, and overhead constraints while also optimally maximizing system-level locking-injected errors.

2. We present and mathematically formalize an equivalent graph-theoretic model that we build into a hierarchical clustering heuristic algorithm that satisfies the same constraints as the ILP and finds solutions in polynomial time in exchange for sub-optimal error injection profiles.

3. We empirically evaluate our methods on MediaBench benchmark circuits locked with SFLL-flex [79], and find that our ILP method on average yields a 55% power and 6% area reduction while maintaining the same locking efficacy as a locking architecture without sharing restore units. Similarly, our P-time heuristic achieves a 31% and 4% reduction in power and area overhead respectively.

2.1 Preliminaries: Logic Locking Attacker Model and SFLL

The logic locking attacker model we consider is a common model used by recent works in logic locking [79, 59, 58, 41, 64, 87, 5, 67]. We assume an adversary with access to

1. a locked module's netlists, obtainable via reverse engineering the locked chip's layout

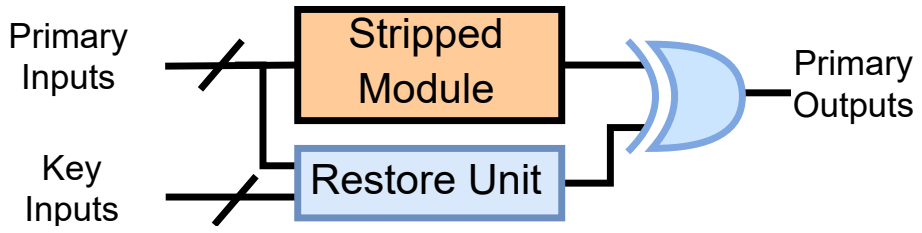


Figure 2.1: The general structure of SFLL locking techniques.

2. an black-box oracle chip with scan-chain access that the attacker can query with inputs and read out correct outputs

The goal of this attacker is to find a key value that produces a fully functional chip.

Logic locking methods are subject to powerful Boolean satisfiability-based attacks (SAT attacks) [67]. The SAT attack finds a key value for the locked module that results in an input-output mapping identical to the oracle. SAT attacks convert the locked module into a Boolean expression that returns TRUE if the module's input and key value yields a value that matches the oracle's output. The attack then iteratively prunes away wrong key values by finding inputs that give different outputs from the oracle until no such inputs can be found. The resulting final key will be functionally equivalent to the correct key.

SFLL [79, 59, 58] is one of the leading logic locking techniques resistant to SAT attacks. At its core is the idea of *functionality stripping*; altering the output of a locked module only when certain inputs are applied. The general structure of a module locked with SFLL is shown in Fig. 2.1. The original module is replaced with a *stripped module* which removes the functionality of selected protected input patterns (PIPs) from the original design. Stripped functionality is restored by

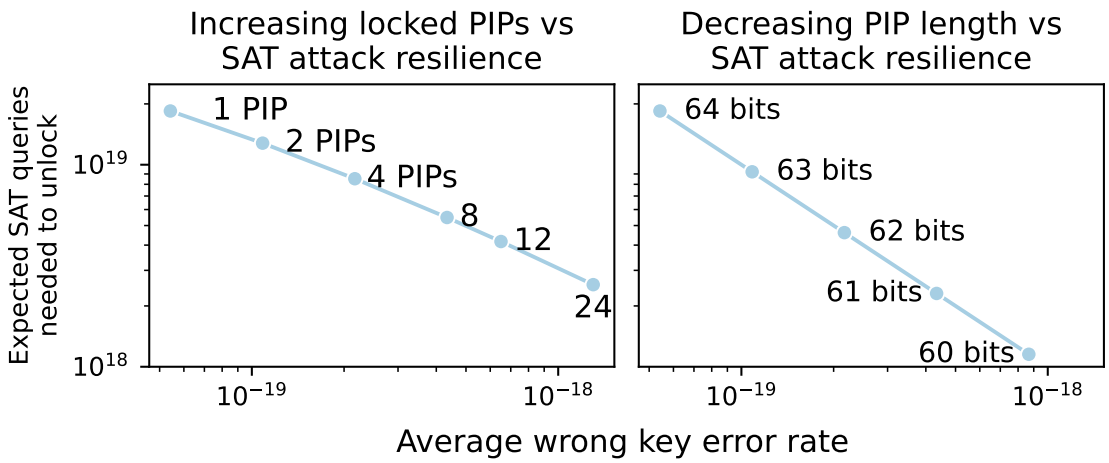


Figure 2.2: Relationships between number of PIPs, PIP width, average error rate, and SAT iterations derived in [86] for a locked module with 64 bit inputs. Increasing the average error rate by either of these two methods always leads to a decrease in SAT attack resilience.

adding a *restore unit* alongside the stripped module. The restore unit corrects incorrect outputs produced by the stripped module provided the PIPs used in the corrupting unit are supplied as key inputs to the restore unit.

By changing the number of PIPs, the rate of wrong key errors can be adjusted, allowing the designer to tune SFLL's construction to the design's target wrong-key error rate. However, inherent to this tunability is a direct inverse relationship between the number of PIPs and the expected time needed to reverse engineer the key by SAT attack [86, 85], thus forcing the designer to choose between a high level of wrong-key error and good resilience against SAT. Selecting PIPs that occur more

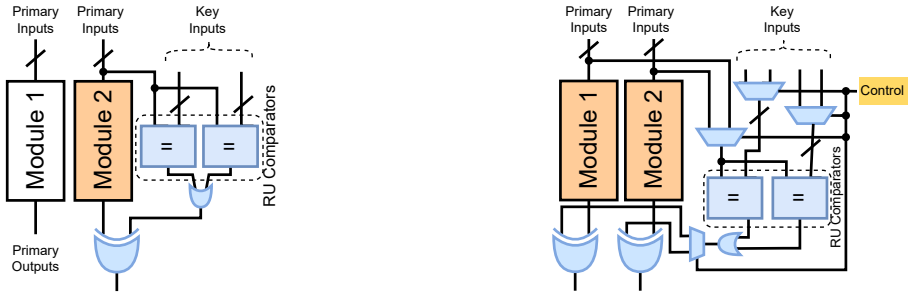
frequently in system traces can somewhat lessen the tradeoff [87]. Fig. 2.2 shows the expected number of SAT queries required for a locked module with 64 bit inputs for both varying numbers of 64-bit PIPs and varying PIP widths.

2.2 Why Share?

Conventional locking approaches decide what functionality to strip on a module-by-module basis after the overall RTL (or even gate level) architecture has been fixed. Subsequently, every locked module has its own dedicated restore unit. Since restore units are generally implemented as comparators, each locked module will contribute a significant amount of additional power and area. Sharing the comparator across multiple functional modules reduces the design overhead incurred by restore units. Additionally, restore units function independently from the specific stripping method used, so sharing them allows the designer to consider overhead and error relevant locking decisions at an architectural level, allowing greater flexibility on where to balance a design's overhead against its security. To illustrate how our method compares to the conventional module-level approach to restore units, we demonstrate how our method can be used to lock a small two module example design using a trace. For this example we assume a constrained overhead budget that only allows for the one restore unit, and only one PIP is used to strip a locked module. Additionally, the designer has access to traces fully describing all candidate PIPs that can be used for locking.

clock	1	2	3	4	5	6
Module 1	⟨15⟩	⟨22⟩	⟨22⟩		⟨35⟩	⟨41⟩
Module 2	⟨22⟩	⟨22⟩	⟨35⟩	⟨35⟩	⟨15⟩	⟨35⟩

Figure 2.3: Two functional modules and a system input trace of the two modules for a single operation.



(a) Non-shared restore unit.

(b) Shared restore unit.

Figure 2.4: Two 2-module locked datapaths. 2.4a is locked with a conventional approach, while 2.4b is locked with our sharing approach. Locked modules are shown in orange and each strip two PIPs.

2.2.1 Conventional Module-level Locking

An example of a conventional restore unit restoring just one stripped module is shown in Fig. 2.4a. Note that only module 2 is locked; the restore unit needed for locking module 1 would require additional overhead. The simplest locking policy would be to randomly choose one input pattern from all possible PIPs to be the PIP for each locked module. However, to differentiate between different PIP selections, we can use the trace to choose a PIP that occurs more often in order to inject more output error. Increasing an individual functional module's error rate in this way results in more errors propagated to the rest of a system, therefore decreasing the functional utility of a locked chip for a malicious user. For the example trace shown in Fig. 2.3, locking module 2 with $\langle 35 \rangle$ as the PIP will create the highest amount of error (three total, at clocks 3, 4, and 6) with the overhead constraint of one locked module.

2.2.2 Locking with a Shared Restore Unit

By sharing a restore unit, we can amortize the area and power cost of the comparators in each restore unit over multiple stripped modules. An example of a shared restore unit restoring two locked modules is shown in Fig. 2.4b. The shared restore unit has the same function as before: compare an input with a set of key inputs, and send a restoration signal if any match. A significant difference is that only one of the two stripped modules can be restored in any clock cycle, which will impose restrictions for the PIPs selected to lock each module.

Which module to examine and which set of key values to use at each clock is controlled by a set of MUXes. On clocks in the trace where PIPs may potentially occur, the controller will direct the restore unit to the locked module where it can occur. On clocks where no module recieves a PIP, the controller can direct the restore unit to the corresponding locked module.

To illustrate the restrictions shared restore units impose, let us examine the system trace in Fig. 2.3 and choose the most common input for each module to be its respective PIP: $\langle 22 \rangle$ for module 1 and $\langle 35 \rangle$ for module 2. In the first clock, neither module receives a PIP as an input, so the controller can have the comparator examine either module without affecting system functionality for correct keys. In the second clock, module 1 receives its PIP $\langle 22 \rangle$ as an input and therefore the output will need to be restored, while module 2 can operate as-is since $\langle 22 \rangle$ is *not* a PIP for module 2, only module 1. Therefore, the controller should direct the restore unit to examine module 1's input and restore if needed.

In the third clock, both modules receive PIPs as inputs, so both module outputs will need to be restored. However, the multiplexer can only choose one module per clock, leading to resource contention. Both locked modules require exclusive use of a resource—the restore unit—at the same clock, but the controller only has one restore unit available to allocate, so the corrupted output of one of the modules will go unrestored, *independent of whether or not the key inputs are correct.*

To avoid resource contention in an overhead constrained design, we can instead pick PIPs that never cause resource contention given a set of comprehensive traces. For this example, if we instead lock module 1 with $\langle 15 \rangle$ and module 2 with $\langle 35 \rangle$, then only one module needs to be restored at any clock in the trace, and therefore no resource contention occurs. This contention-free PIP selection causes error in clock cycles 1, 3, 4, and 6, for a total of four injected errors, an improvement over the three errors injected with an unshared restore unit.

This approach—choosing PIPs that never cause resource contention in a trace—is the approach we take for sharing restore units. Our methods find locking configurations that meet contention avoidance, resource budget and attack resilience constraints while optimizing for high levels of injected error.

2.3 Error-Optimal Sharing via ILP

The problem of finding error optimal restore unit bindings and PIPs can be described as a 0-1 ILP. To make our formulation easy to follow, Table 2.1 lists all constants and variables used, along with their definitions.

We begin with the constraints. First, each stripped module should lock just enough minterms to be sufficiently robust against expected SAT attacks [86, 85]. This can be expressed as the following constraint:

$$\sum_{i=1}^{r} x_{i,j} \leq c_j, \quad \forall j \in [1, u] \tag{2.1}$$

Table 2.1: Variables and constants used in the ILP formulation

Notation	Definition
s	Total number of allocated shared restore units
r	Total number of unique input patterns
u	Number of functional modules
T	Number of clock cycles in the trace
c_j	Maximum number of PIPs allowed for functional module j
$d_{i,j,k}$	Encodes trace information. $d_{i,j,k}$ is 1 if the ith candidate minterm is an input to functional unit j at clock cycle k in the trace, and is 0 otherwise
$x_{i,j}$	Variable that encodes locked minterm assignments to functional units. $x_{i,j}$ is 1 if the ith candidate minterm is a locked minterm for functional unit j, and is 0 otherwise
$a_{j,l}$	Variable that encodes functional unit assignments to restore units. $a_{j,l}$ is 1 if functional unit j is restored by restore unit l, and is 0 otherwise
$h_{i,j,k,l}$	Variable associated with locked input minterms in the trace. $h_{i,j,k,l}$ is 1 if the following hold: 1) minterm i is used to lock module j 2) module j is restored by restore unit l and 3) minterm i is an input minterm for module j at clock cycle k in the trace, 0 otherwise. Highly sparse.

To keep MUX overhead low, stripped modules can only by restored by one shared restore unit. Therefore, a stripped functional unit with locked minterms can only be restored by exactly one restore unit. We use the following inequalities to describe these two requirements:

$$x_{i,j} \leq \sum_{l=1}^{s} a_{j,l} \leq 1, \quad \forall i \in [1, r], j \in [1, u], \tag{2.2}$$

To ensure PIP assignments decided by $x_{i,j}$ do not cause contention over restore units configured by $a_{j,l}$, i.e. the sets of locked minterms belonging to different functional modules unlocked by the same restore unit should never occur in the same clock, we use inequality (2.3) to calculate which input patterns i in the trace are PIPs for a given restore unit l and locked module u at clock cycle k in the trace. Inequality (2.4) constrains the number of restored modules per restore unit per clock

to be at most one, otherwise contention occurs.

$$a_{j,l}x_{i,j}d_{i,j,k} \leq h_{i,j,k,l} \quad \forall i,j,k,l \tag{2.3}$$

$$\sum_{i=1}^{r}\sum_{j=1}^{u} h_{i,j,k,l} \leq 1 \quad \forall k \in [1,T], l \in [1,s] \tag{2.4}$$

Note that inequality (2.3) is not linear. However, the decision variables $x_{i,j}$ and $a_{j,l}$ are constrained to be binary, and d is a constant value representing the trace. Therefore, the quadratic constraint inequality (2.3) can be rewritten as an equivalent set of linear inequalities by adding an additional variable. Additionally, note that if $d_{i,j,k}$ is 0 for some value of (i,j,k) (which is most values of (i,j,k)), then the corresponding constraint is redundant and can be removed, greatly reducing the number of constraints and additional h variables needed. The resulting linear expressions are

$$\left.\begin{aligned} a_{j,l} + x_{i,j} - 1 \leq h_{i,j,k,l} \\ h_{i,j,k,l} \leq a_{j,l} \\ h_{i,j,k,l} \leq x_{i,j} \end{aligned}\right\} \begin{aligned} &\forall (i,j,k) \in \{i,j,k | d_{i,j,k} = 1\} \\ &\qquad \forall l \in [1,s] \end{aligned} \tag{2.5}$$

The optimization objective is to maximize the number of corrupted outputs in the trace caused by PIPs:

$$\max_{a,x,h,g} \sum_{j=1}^{u}\sum_{i=1}^{r} b_{i,j}x_{i,j} \tag{2.6}$$

subject to inequalities (2.1) to (2.3) and (2.5).

Feasible solutions to this ILP represent the set of valid locking configurations which do not cause resource contention and respect overhead and SAT attack resilience requirements. Optimal solutions are locking configurations that maximize injected errors in the trace used to configure locking.

2.3.1 ILP Usage

A designer seeking to secure a design within locking-induced overhead and security constraints while maximizing locking's impact on the functionality of a locked design using shared restore units would need to determine what functional modules should be locked, the number of PIPs each locked module should have, and the number of restore units that can be allocated from the design's power budget. Additionally, a set of traces that fully describe all conflicts between each module's prospective PIPs will be needed.

These values can then be encoded into our ILP formulation as described previously and solutions found using one of the many available ILP solvers. Optimal values of x, a, and h can then be used to set PIP assignments, configure restore unit MUXes, and program controller behavior respectively.

2.4 Efficient Sharing via Graphs

While in practice ILP solvers can efficiently solve some large instances of integer LPs, the worst-case runtime is still NP-complete. To provide an alternative to avoid such worst-case time complexity, we present a graph theoretic model for sharing in this section that will be used as a foundation for our heuristic algorithm in section 2.5.

2.4.1 Locking and Sharing Configurations with the PIP Compatibility Graph

We define entries in the trace as three dimensional vectors of the form $(i, j, k), i \in [1, r], j \in [1, u], k \in [1, T]$. Variables are defined the same as our ILP (e.g. i represents a specific input pattern, u represents the number of lockable modules, etc). We represent the entire trace and its entries as a set of these three dimensional vectors. We define a PIP assignment as a two dimensional vector of the form $(i, j), i \in [1, r], j \in [1, u]$ and represents the decision of stripping minterm i from module j.

We use the term *PIP compatibility* to refer to whether or not a pair of PIP assignments to two different locked modules will cause restore unit contention if a restore unit is shared between the two modules. It is defined as follows:

Definition 2.1 (Incompatible and Compatible PIPs)**.** Two PIP assignments $v_1 = (i_1, j_1), v_2 = (i_2, j_2)$ where $i_1, i_2 \in [1, r]$ and $j_1, j_2 \in [1, u], j_1 \neq j_2$ are *incompatible* if (i_1, j_1, k) and (i_2, j_2, k) are both in the trace for some $k \in [1, T]$. A pair of PIP assignments v_1, v_2 that do not satisfy this property are *compatible*.

We can use this definition to build the multipartite PIP compatibility graph $G_P(V_P, E_p, w)$. G_P represents all pairs of compatible stripping decisions between modules in the design. The set of nodes $V_P = V_{P1} \cup V_{P2} \cup ... \cup V_{Pu}$ consist of two dimensional PIP assignment vectors $(i, j) \in V_{Pj}, i \in R, j \in U$ where each partition V_{P_j} contains all potential PIP assignments that strip module j. An edge exists between two nodes $(v_1, v_2) \in E_P$ if v_1, v_2 are compatible as defined previously. The

node weight function $w : V_P \to \mathbb{N}$ weights each PIP assignment $(i, j) \in V_P$ by how often input pattern i occurs as an input to functional module j in the trace. The set of PIPs chosen to strip locked modules can then be defined as a subset $S \subseteq V_P$ of all possible PIP assignments.

Similarly to PIP assignments, we define a restore unit resource assignment to be a two dimensional vector $(j, l) \in W, j \in [1, u], l \in [1, s]$ that represents the decision of having functional module j be restored by restore unit l.

We denote W as the set containing our chosen module-to-restore unit mapping vectors (j, l) for an entire design. To keep area overhead caused by MUXes low, each locked module can only be restored by exactly one restore unit. Formally, we can define this constraint as follows: we first define W_l to be the set of locked modules restored by restore unit l

$$W_l := \{j \in U | (j, l) \in W\} \tag{2.7}$$

then for two different restore units l_x and l_y,

$$W_{l_x} \cap W_{l_y} = \emptyset \tag{2.8}$$

must hold.

We can now define a *locking configuration* (S, W) as a structure containing both a set of PIP stripping decisions S and a sharing configuration W. Now we can define a contention-free locking configuration. First, we define S_l as the set of PIP assignments restored by restore unit l:

$$S_l := \{(i, j) \in S | (j, l) \in W\} \tag{2.9}$$

We can formally define a *contention-free* locking configuration by using the PIP compatibility graph G_P defined earlier:

$$\begin{aligned} \text{if} \quad & (s_1, s_2) \in E_P \\ & \forall s_1, s_2 \in S_l, \;\; s_1 \in V_{Pi}, s_2 \in V_{Pj}, \;\; i \neq j, \;\; i, j \in W_l \\ & \forall l \in [1, s], \end{aligned}$$

then (S, W) will not cause resource contention.

Note that this definition of a contention-free locking configuration is equivalent to the definition of a set of partition disjoint (from equation 2.8) multipartite cliquess on G_P. Therefore it can be shown that a set of multipartite cliques, each containing members from disjoint partitions of G_P, represents a locking configuration that never causes contention in the trace. From here, we can define a graph problem equivalent to the ILP:

$$\arg\max_{S} \sum_{s \in S} w(s) \tag{2.10}$$

subject to

$$|\{(i, x) \in S | x = j\}| \leq c_j \tag{2.11}$$

and (S, W) being contention-free as defined before.

Finding a locking configuration (S, W) that meets these constraints while maximizing corrupted outputs can be seen as finding a maximum weight set of multipartite cliques on G_P with constraints on the number of cliques and cardinality of each clique. Like the ILP described previously, no polynomial time algorithm for

finding such cliques is known. The rest of this section will be describe our heuristic algorithm which trades output corruption optimality for a polynomial algorithm runtime.

2.5 Towards a Heuristic Algorithm

We propose a method based on hierarchical agglomerative clustering to find locking solutions that meet our other stated objectives of no contention, high corruptability, low overhead, and attack resilience in polynomial time at the expense of a suboptimal number of corrupted outputs. We start with introducing a special case of the optimization problem defined in section 2.4.1—the two-module sharing problem—and show that optimal solutions to this special case can be found in polynomial time. We then use these two-module solutions to initialize our hierarchical clustering heuristic algorithm.

2.5.1 The Two Module Problem

The two module problem considers the specific case of two functional modules sharing a single restore unit, and is needed in the initialization step of our heuristic. We relax inequality (2.1) for the two-module case; guaranteeing attack resilience by limiting the number of PIPs per module will be performed later.

The PIP compatibility graph for two modules is a node-weighted bipartite graph $G_P(V_{P1}, V_{P2}, E_P, w)$. Finding a contention-free locking configuration S with maximum output corruption reduces to finding a maximum node-weighted biclique

in G_P:

$$\begin{aligned} \arg\max_{S} \quad & \sum_{s \in S} w(s) \\ \text{subject to} \quad & S \subseteq V_{P1} \cup V_{P2} \\ & \{v_1, v_2\} \in E_P \qquad \forall v_1 \in V_{P1} \cap S, v_2 \in V_{P2} \cap S \end{aligned} \tag{2.12}$$

Fortunately, this problem has been shown to be optimally solvable in polynomial time by converting (2.12) into an equivalent 0-1 integer linear program and solving its LP relaxation [14].

2.5.2 Hierarchical Clustering Heuristic

Our algorithm uses a graph-based data structure $G_R(V_R, E_R, a, A)$ to store its state. G_R is a complete graph of restore units, where each node represents an allocated restore unit and each edge represents a decision to merge two restore units into one. Each node in V_R and edge in E_R has an associated set of PIP assignments stored as node and edge attributes in a and A respectively. The set of node attributes a keeps track of the current locking configuration solution state, while each edge attribute in A represents a potential decision to merge two restore units together the outcome of clustering the two restore units at each edge's endpoints.

The full algorithm is shown in Alg. 2.1. The algorithm first initializes G_R's nodes to represent a conventional locking configuration where every module has its own dedicated restore unit (line 1). Each node attribute contains PIP assignments that locks every possible PIP for the node's module (line 2). Every edge attribute is initialized to the two module solution introduced in the previous section (line 3).

Algorithm 2.1: Hierarchical clustering algorithm

Input: $G_P(V_P, E_P, w), s_{limit}, c$ in

Output: $G_R(V_R, E_R, a, A)$ out

G_R Initialisation :

1: $V_R \leftarrow \{1, 2, ..., u\}$, E_R is complete

2: $a_i \leftarrow V_{Pi} \quad \forall i = 1...u$

3: $A_{i,j} \leftarrow$ 2 module sol. for modules $i, j \quad \forall \{i, j\} \in E_R$

Iterative Clustering:

4: **while** $|V_R| > s_{limit}$ **do**

5: $i', j' \leftarrow \underset{\{i,j\} \in E_R}{\arg\max} \; (\sum_{s \in A_{i,j}} w(s) - \sum_{s \in a_i} w(s) - \sum_{s \in a_j} w(s))$

6: $a_i \leftarrow A_{i',j'}$

7: **for** $k \in V_R, k \neq i', j'$ **do**

8: $A_{i',k} \leftarrow (A_{i',j} \cap A_{i',k}) \cup (A_{i',j} \cap A_{j',k}) \cup (A_{i',k} \cap A_{j',k})$

9: **end for**

10: $V_R \leftarrow V_R \setminus j'$ and assoc. entries in G_R

11: **end while**

12: **for** each module j restored by each RU in G_R **do**

13: select top-c_j most frequently occurring minterms as PIPs for module j

14: **end for**

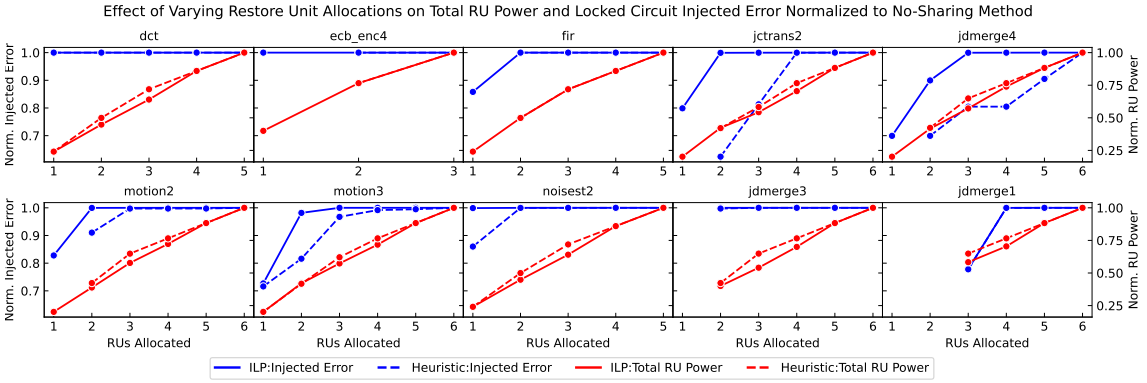


Figure 2.5: Breakdown of how restore unit power and total injected error is affected by changes in the number of restore units allocated for sharing. All axes are normalized to a design that locks every module with a dedicated (i.e. non-shared) restore unit. Note that low restore unit allocations resulted in no feasible solutions for some benchmarks.

The clustering step first chooses the pair of nodes $i', j' \in V_R$ whose merger yields the greatest increase in overall corrupted outputs (line 5). We perform the merger by updating the graph in-place: stripping decisions stored in the edge between i' and j' are copied to node i' (line 6). After that, edge attributes containing future possible merges between the just-merged node and all other nodes are updated. It can be shown that the set operation on line 8 preserves PIP compatibility within each restore unit. An additional check can be added here to remove edges with configurations that do not strip functionality from every module covered by restore units at each edge endpoint. Finally, as a post-processing step, PIP assignments with low weight in G_P are removed until each locked module j only strips c_j input patterns (lines 12 to 14).

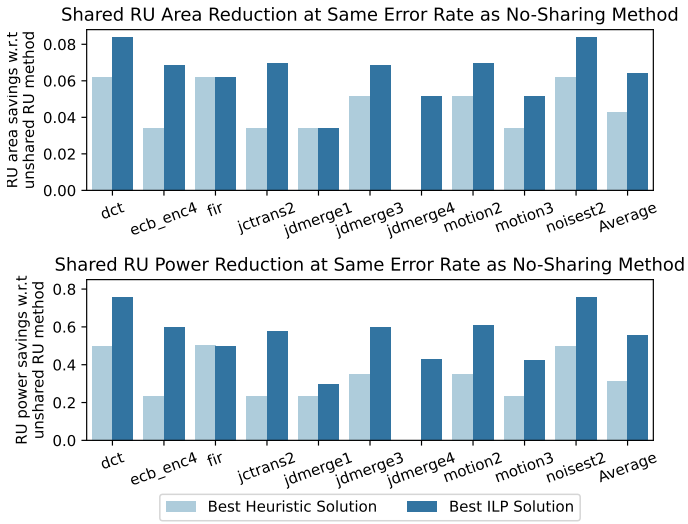


Figure 2.6: Locked datapath area and power for benchmarks tested, normalized to single-PIP unshared restore units. The lowest power and area achieved with the same total system error as the conventional no-sharing approach are shown here.

The G_R initialization step runs an instance of the LP relaxation of eq. (2.12) for every pair of modules in the design, for a complexity of $O(LP(r)u^2)$, where $LP(r)$ is the runtime complexity of solving a LP with r variables. The iterative clustering step evaluates a linear expression for every edge in E_R and performs $O(s)$ set intersection and union operations. This step runs up to $s-1$ times, so the clustering portion runs in $O(s^4)$. Finally, selecting the c_j most common input patterns in each locked module runs in $O(r\log r)$. Therefore, our heuristic runs in $O(LP(r)u^2 + s^4 + r\log r)$, a P-time solution.

2.6 Results

To evaluate our resource sharing approach to logic locking, we used datapaths synthesized from C functions extracted from the MediaBench suite [37]. The DFG of each function was extracted using SUIF and was scheduled with a path-based scheduler [48]. An allocation of up to three adders and three multipliers was used, and each functional unit was bound using a security-aware binding algorithm [87]. We used test inputs supplied by Mediabench to generate input traces that represent typical use scenarios. Traces were generated by simulating the DFG with test inputs.

Stripped modules were protected with SFLL-flex, although any stripping method that relies on restore units could have been used. To maximize security against SAT attack, every locked module is allocated only one PIP (i.e. $c_j = 1 \forall j$). We compared our optimal and heuristic formulations to a error-maximum locking solution that does not share restore units.

We used Gurobi [18] as the ILP solver and NetworkX [19] to implement the graph heuristic algorithm. We used the constraint set from the ILP to verify that the heuristic algorithm returns a feasible and valid solution. All experiments were performed with a standard desktop computer equipped with an AMD Ryzen 3.4 GHz processor and 32GB of system memory. Scripts to implement and manage the solvers were written with Python and run under Linux. Power and area overhead numbers were calculated using designs built using FreePDK45 [66] and synthesized with Cadence Genus.

In order to compare our techniques against the conventional no-sharing approach, Fig. 2.6 shows how power and area overhead used by restore units is reduced by our heuristic algorithm and ILP formulation while maintaining the same number of injected errors. Data shown are normalized by the area and power consumed by the same datapath locked with a conventional error-maximizing approach. Our heuristic found locking solutions that reduced locking-induced power and area overhead by 31% and 4% relative to the conventional approach in most benchmarks, while our optimal ILP solution was able to save 55% and 6% power and area overhead in all benchmarks. Area savings were less significant due to the additional MUXes needed to switch between restore units. Since the restore units do not lie on a critical timing path, no design experienced any change in their timing under any locking scenario.

While both of our methods do not explicitly consider area and power, the clustering heuristic consistently shows reduced area and power savings compared to ILP-derived sharing configurations. Close examination of the sharing configuration solutions derived from each method shows that ILP optimal sharing configurations tend to favor an unbalanced distribution of restore units to functional modules, where one restore unit restores a large set of functional modules and the rest of the restore units each only restore a single module. On the other hand, the heuristic tends to favor a more balanced distribution, where the restoration of modules is evenly divided among the available restore units. This results in more MUXes over the ILP, and therefore a greater area and power overhead.

We can further break down the results by examining how power and total injected errors change with different restore unit resource allocations in Fig. 2.5. Resource allocations range from allocating one restore unit for each functional module to allocating just one restore unit for all functional modules. Note that allocating a restore unit to each functional module will yield a solution that is identical to the conventional approach, and is reflected in our graphs. From Fig. 2.5 we can observe that as allocations decrease, the number of injected errors does not decrease until allocations start to approach 1. The heuristic tends to drop in error sooner than the optimal solution for some benchmarks. Meanwhile, overhead trends roughly linearly with resource allocations. We can see that there is significant power overhead margin that can be reclaimed by efficiently sharing restore units at the architectural level while maintaining the same error severity. This tunability can enable the designer greater control on the overall system, allocating more resources towards security features and meeting area and power constraints.

2.7 Summary

In this chapter we present a resource sharing methodology for restore units used by SFLL-based locking techniques. Our method decreases design overhead when compared to a standard non-shared locking configuration at the same level of injected error while meeting SAT resilience requirements. We model resource-contention free resource sharing as a 0-1 integer linear program, and present a polynomial time heuristic algorithm based on hierarchical clustering to find good

feasible solutions. We apply the technique on sample traces for synthesized functions from the MediaBench [37] suite of benchmarks and compare overhead reductions for the optimal ILP solution, the heuristic solution, and error-maximal non-shared minterm selection algorithm. Our optimal and heuristic formulation reduces total power consumed by a locked design 55% and 31% respectively when compared against an equivalent locking construction without shared restoration hardware.

Chapter 3: Delay Minimal Design and Implementation of Restore Units as System Resources

The previous chapter proposed system level design perspectives, namely treating restore units as shared circuit level resources. In this chapter we implement RUs as a true system level resource in the same way as any other kind of HLS resource. Instead of implementing RUs as fixed circuit-level shared structures as in the previous chapter, we implement RUs as a true system level shareable resource, since at its core a RU's function is to simply compare two values and emit a restore/no restore signal. We eschew the idea that a restore unit must always examine the input of a locked module in the context of system level design and point out that module inputs are dictated by the operations scheduled for and bound to that module, and therefore a RU comparison only needs to occur once per bound operation. This allows us to achieve the same end goal from the previous chapter—reducing locking overhead—more effectively and without any onerous trace assumptions by relying on information generated by HLS tools.

(a) Example DFG 1.

(b) Example DFG 2.

Figure 3.1: Example DFGs used to illustrate our method, with nodes representing operations and edges representing data dependencies.

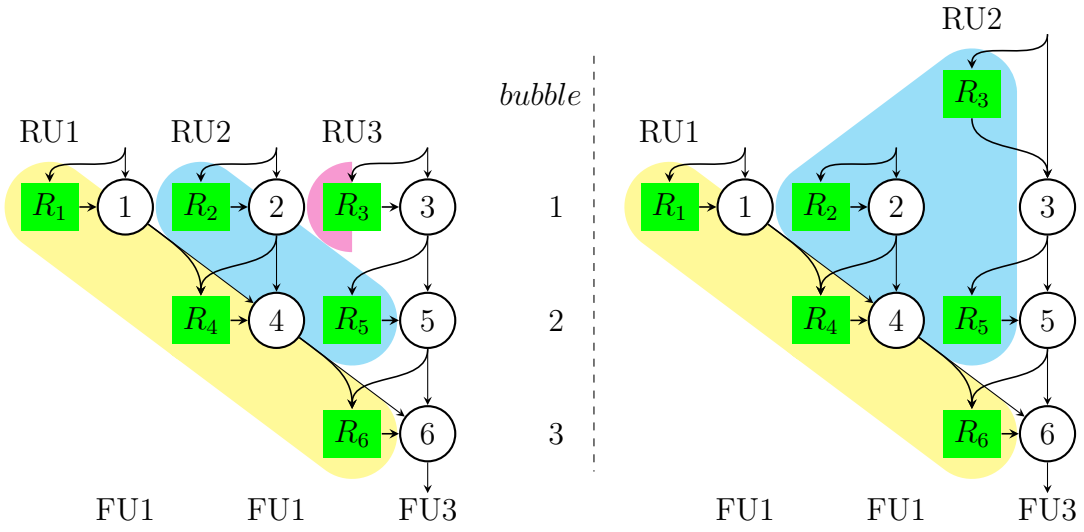
The rest of this chapter is organized as follows. Section 3.1 provides a motivating example illustrating some of the concepts introduced in Section 3.2, which provides a formal description of the ILPs used to implement delay-minimal system level locking. Section 3.3 presents our results and Section 3.4 summarizes the chapter.

3.1 Operation-Oriented Restoration: An Illustrative Example

To illustrate how we can use this operation-level perspective to reduce locking overhead, we will use the scheduled DFGs (SDFGs) shown in Fig. 3.1 to illustrate how we propose lowering the RU allocation without causing resource contention. We will illustrate two concepts: inserting bubbles (or delays in the schedule) into an existing schedule, and *restore mobility*.

First we consider the SDFG shown in Fig. 3.1a. We assume that all HLS steps have taken place and operations have already been bound to hardware modules (binding is not shown). The conventional circuit-level approach of binding a RU to each locked hardware module can be represented in the DFG shown in 3.2a. We explicitly show restore unit operations in each green box. Each restore operation R_n takes in the inputs of operation n and outputs a restore/no restore signal to the correcting XOR gates bound to operation n's hardware module. Note that only in the first clock cycle are all three RUs utilized. When only two RUs can be allocated, there is resource contention in the first clock as three operations are scheduled to be restored but only two restore units are allocated. Leaving one of the operations' input uncompared by a RU raises the risk of letting a corrupted output pass through unrestored, resulting in incorrect outputs and potential system corruption. Hence, minimizing locking resources and avoiding contention over restoration resources are opposing design goals.

To resolve contention without changing the RU allocation, a bubble (or a one-cycle delay to the schedule) can be inserted into the SDFG before the first clock as shown in Fig. 3.2b. The input comparison for operation 3 can then be performed in the inserted bubble without resource contention because the input values for operation 3 are available in the first clock. Since the restore operation depends only on the input values they can be compared by the RU *before* the actual execution of operation 3 takes place. This adds a delay of one clock cycle to the overall SDFG, in exchange for the lower power usage due to the reduced RU allocation.



(a) Conventional restore dataflow for three allocated restore units on example DFG 1.

(b) Contention-free restore dataflow for two allocated restore units on example DFG 1 via bubble insertion.

Figure 3.2: Delay-optimal restoration schemes for an allocation of three and two restore units. A restore unit compare operation for an operation n is denoted by R_n.

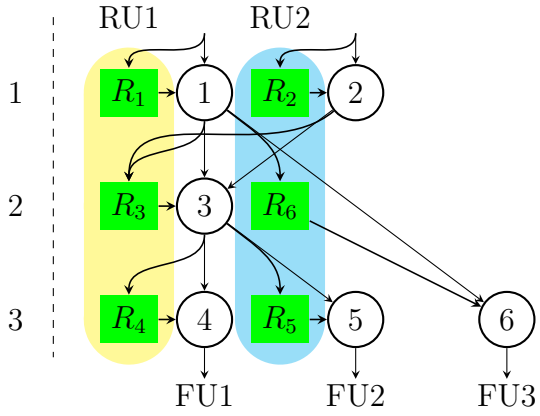


Figure 3.3: Contention-free restore dataflow for two allocated restore units on example DFG 2 via restore mobility.

Now consider the scheduled and bound DFG shown in Fig. 3.1b for an allocation of two RUs. Again note that there are three operations scheduled in the same clock, this time in clock 3, so if all three operations are restored in that clock cycle then there will be resource contention. However, since the input of operation 6 is the output of operation 1, which is scheduled two clocks prior to op. 6, operation 6's inputs are available in clock 2. Since only one other operation is scheduled for clock 2, we can move the input comparison of operation 6 to clock 2 without causing contention over RUs (Fig. 3.3). Any correction to the output of operation 6 will still occur in clock cycle 3, dependent on the results of the comparison now performed in clock 2.

Moving the restore a clock earlier relieves clock cycle 3 of any locking resource contention, ensuring the correct functionality of the system design while without any delay penalty while reducing the RU allocation to 2. Note that for this example SDFG, only operation 6's restore "mobility" extends beyond a single clock cycle: all other operations have inputs that are only available at the end of the previous clock and therefore have a restore mobility limited to the clock cycle they are scheduled for. Modifying operation schedules to adjust each operation's restore mobility is out of scope of this chapter and we leave it to future work.

Table 3.1: ILP Constants and Variables

Notation	Definition
$G(V, E, s)$	Acyclic digraph representing the SDFG, with nodes V representing operations, edges E representing data dependencies, and a schedule of operations s
$RUlim$	Restore unit resource allocation, in terms of comparators
n_cycles	Original schedule length in clock cycles
t_o^a	Clock cycle in the original schedule for which all inputs of operation o are available
t_o^c	Clock cycle in the original schedule that operation o is scheduled for
t_b	Set of potential interleaved bubble clock cycle indices used in the ILP
$A_{t,o}$	Binary variable indicating restoration scheduling. 1 if operation o is scheduled to be restored at clock t, 0 otherwise. Only defined for t in the valid range for each operation o.
y_t	Nonnegative integer variable indicating the number of actual bubbles inserted into bubble site $t \in t_b$

3.2 ILP-Based Delay Minimization Techniques

With the intuition in hand for bubble insertion and restore mobility, we can proceed to our methods for scheduling restore operations within a RU resource constraint and with minimal inserted delays. To simplify our descriptions, Table 3.1 lists all of the constants and variables used for the rest of this chapter.

We will first define how we represent bubbles and restore operation mobility in our formulation before going over constraints common to both our optimal and p-time implementations. We then cover the specific constraints and objective functions of each method separately. We conclude the section with a brief usage overview.

3.2.1 Bubble Indexing and Restore Mobility

Our method considers DFGs that have already gone through the operation scheduling and resource allocation steps, so a scheduled DFG $G(V, E, s)$ is given, where $s(v)$ is the clock cycle that operation v is scheduled to execute in and n_cycles is the length of the schedule in clock cycles.

First, we note that bubbles can be inserted before any clock in the schedule s. So, given there are n_cycles clocks in the schedule, there are therefore n_cycle bubble/delay insertion points, one before each SDFG clock, for a total of $2n\_cycle$ cycles where operation input comparisons can occur.

To represent these n_cycle potential bubble insertion points, we double the length of the schedule by interleaving with psuedo-clocks, one for each bubble insertion point. The original n_cycle schedule clocks $0, ..., n\_cycles - 1$ are indexed by $t_s = 1, 3, ..., 2n\_cycles - 1$ and interleaved potential bubble insertion points are indexed by $t_b = 0, 2, ..., 2n\_cycles - 2$. Note that we do not modify the schedule beyond inserting bubbles for restoring operations: the operation schedule ordering remains the same in t_s, and t_b psuedo-clocks are only reserved for restoration operations. Additionally, each pseudo-clock bubble insertion point is only used to decide if an operation is to be inserted into a bubble before a particular SDFG clock and not how many bubbles are inserted. The *number* of bubbles inserted directly before a regular SDFG clock is determined by the number of operations scheduled for the psuedo-clock divided by the number of restore units $RUlim$: the order in which bubbled restores happen before a t_s clock cycle does not matter.

The restore mobility for an operation is defined similarly to that of operation mobility in HLS, except we use an already scheduled DFG instead of the difference between the ASAP and ALAP schedules. The restore mobility $[t_o^a, t_o^c]$ for an operation o is defined as the range of clocks in the original SDFG where o's inputs are available. Specifically, t_o^a is the first clock where all the inputs of o are ready, expressed as

$$t_o^a = \max\{s(p)|p \in pred(o)\} + 1 \tag{3.1}$$

where $pred(o)$ is the set of predecessors/data dependencies of operation o. For operations that only depend on inputs to the DFG, w.l.o.g. we assume that all DFG inputs are available in the first clock. t_o^c is defined as $s(o)$, the clock that operation o is scheduled to execute in.

3.2.2 General Constraints

First, we constrain the number of operations that can be restored in each original SDFG clock cycle to not exceed the restore unit resource allocation:

$$\sum_{o \in V} A_{t,o} \leq RUlim \quad \forall t \in t_s \tag{3.2}$$

Note that the clock cycles of A are indexed with pseudo-bubble clocks interleaved. Additionally, we only enforce this constraint during non-bubble clock cycles since scheduling more than $RUlim$ restores in a pseudo-cycle in t_b merely increases the number of bubbles actually inserted before a SDFG clock cycle t_s.

The other general constraint ensures every operation has a restore unit scheduled to restore it:

$$\sum_{t \in 0...2n\_cycles-1} A_{t,o} \geq 1 \quad \forall o \in V \tag{3.3}$$

We additionally set values of $A_{t,o}$ that lie outside an operation's restore mobility range to 0:

$$A_{t,o} = 0 \quad \forall o \in V, t \notin [t_o^a, t_o^c] \tag{3.4}$$

as an operation can only be restored when its inputs are available.

3.2.3 Optimal Delay Minimization

We would like to minimize the number of bubble cycles (and hence the delay overhead) needed to restore all operations for a given $RUlim$ allocation. This can be expressed using the following objective function

$$\min_A \sum_{t \in t_b} \left\lceil \frac{\sum_{o \in O} A_{t,o}}{RUlim} \right\rceil \tag{3.5}$$

where the term inside the ceiling function calculates the number of actual bubble cycles that will need to be inserted before a regular clock cycle with DFG operations. This objective cannot be used as-is as the ceiling function is not linear. However, since A is constrained to take integer values it can still be expressed as an ILP by using an additional intermediate integer variables y_t for each clock $t \in t_b$:

$$y_t \geq \frac{\sum_o A_{t,o}}{RUlim} \quad \forall t \in t_b \tag{3.6}$$

$$y_t \leq \frac{\sum_o A_{t,o}}{RUlim} + 0.999 \quad \forall t \in t_b \tag{3.7}$$

The objective function represents the same, the number of actual bubble cycles inserted into the original SDFG:

$$\min_{A,y} \sum_{t \in t_b} y_t \tag{3.8}$$

3.2.4 Near-Optimal Poly-time Bubble Minimization

While the above optimization program optimally minimizes the actual delay required to implement contention-free locking, ILPs are NP-hard and therefore runtime is not guaranteed to be good in practice. We can relax the constraints used to implement the ceiling function in the delay-optimal program to arrive at a related simpler problem that can be reduced to LP, and hence is solvable in p-time. The only constraints are the general constraints introduced in section 3.2.2. The objective function is to minimize the total number of operations that are placed inside interleaved bubbles:

$$\min_{A} \sum_{o \in O, t \in t_b} A_{t,o} \tag{3.9}$$

The coefficient matrix for the constraints in 3.2.2 is totally unimodular, therefore the solution to the LP relaxation of this program is integral, and hence optimal for the ILP. Note that the same technique cannot be applied to the optimal delay insertion method discussed previously as the the coefficients in constraints (3.6), (3.7) have non-integral values.

The realized delay of the relaxed objective function (3.9) may be higher than the delay-optimal objective function (3.5). However, it is easy to see that if optimal value of the delay-minimizing objective function (3.5) is 0, then the optimal values

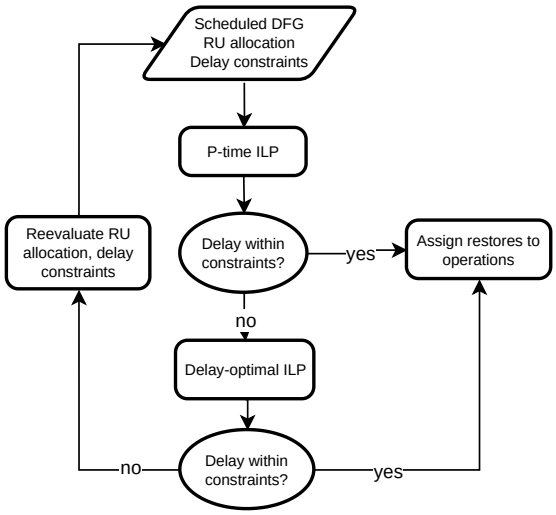


Figure 3.4: Overall design flow for our proposed system-level implementation.

of $A_{t,o}, t \in t_b$ must all be 0, so the optimal solution for the relaxed objective function (3.9) must also 0. In other words, if 0-delay restore unit assignment is in the feasible set, then the optimal value for both optimization programs will be 0. The intuition is that an optimal delay of 0 by definition means that all operations can be restored in regular schedule clocks without the need of any inserted bubbles, and therefore the total number of operations placed in bubbles must be 0 for an optimal value of A.

3.2.5 ILP Usage

A system engineer wishing to reduce the overhead caused by SFLL locking will need to first perform HLS on a target DFG to be locked, performing scheduling, binding, and resource allocation so that a scheduled DFG is available. A target RU allocation and maximum tolerable delay penalty will need to be determined based on system design requirements and overhead limits.

The overall flow is illustrated in Fig. 3.4. First the restore mobility $[t_o^a, t_o^c]$ of each operation will need to be calculated from the SDFG. Then, the p-time ILP can be used first to determine if a zero-overhead restore assignment is feasible. If so, then the optimal values of A determine when each operation is restored and can be used to program the system controller. If the p-time method finds that a nonzero number of operations need to be restored in bubble clocks, then the engineer can choose to accept the resulting delay penalty (calculated by finding the value of the optimal objective function (3.5) using p-time objective solution A), or instead try to find a better solution to the delay-optimal ILP. If the delay overhead found by the optimal method still does not satisfy system delay constraints, then the engineer will need to reevaluate the system delay and/or RU resource constraints.

3.3 Results

We evaluate our methods on synthesized functions extracted from the MediaBench suite of multimedia encoding/deconding functions [37]. The DFG of each tested function was extracted using SUIF and scheduled with a path-based scheduler [48]. Binding was done with a security-aware binding algorithm [87] and an allocation of up to three adders and three multipliers was used.

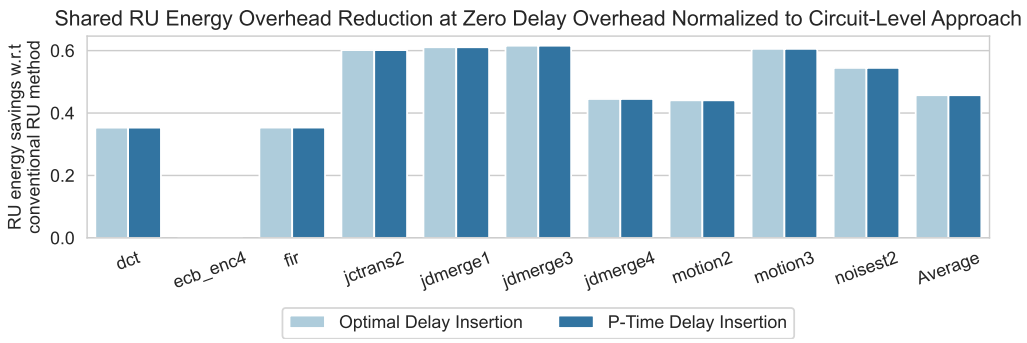


Figure 3.5: Reductions in locking-caused system energy overhead at the lowest zero-delay restore unit allocation, normalized to a conventional locking approach that protects every locked module with a dedicated restore unit.

Our overhead numbers are calculated based on an assumption that each locked hardware resource only protects one PIP in order to maximize resilience against the SAT attack. We compare our delay-optimal and p-time methods against a conventional locking scenario where every module is restored by a dedicated restore unit.

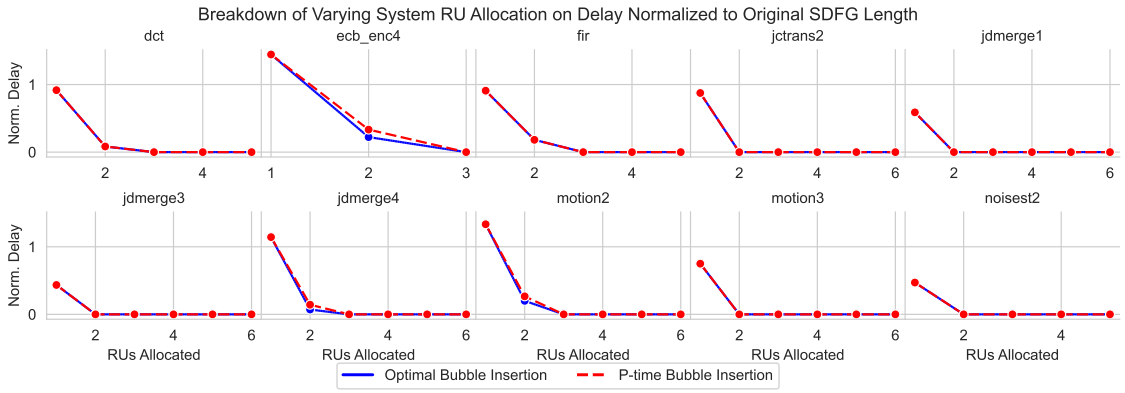
We used Gurobi [18] as the ILP solver. All results were obtained using a standard desktop running Linux and a AMD 3.4 GHz processor. Overhead numbers were calculated based on simulated power data for hardware modules designed with FreePDK45 [66] extracted from Cadence Genus.

Fig 3.5 shows the energy savings normalized to the traditional circuit-level SFLL implementation for both the optimal and poly-time methods at the lowest restore unit allocation that yields zero delay overhead. The energy savings are identical for both methods at 0 delay overhead. As explained in section 3.2.4, both optimal and p-time formulations will find 0-overhead restore scheduling solutions if

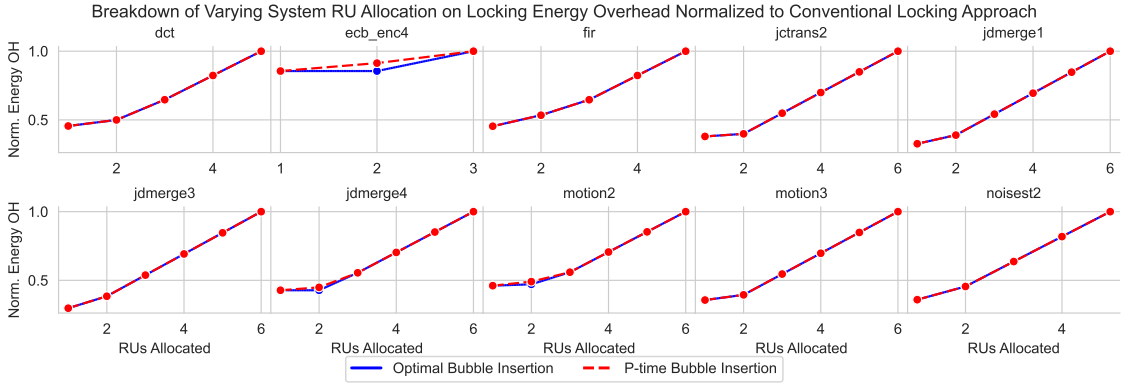
one exists, so these results are in line with expectations. Note that `ecb_enc4` showed 0 energy savings at 0 delay overhead, indicating that the overall restore mobility in that benchmark is limited.

Fig. 3.6 further breaks down how delay and energy overhead change with varying restore unit allocations for both methods. At extremely low restore unit allocations, there are very few restoration resources to go around and hence resource contention over RUs is severe, so delay overheads explode at the lowest allocations. However, for more moderate reductions in restoration resources, both methods provide good power savings at modest or zero delay overhead. Indeed, if the system requirements allow for any kind of additional delay overhead on top of the HLS-generated schedule, then significant power savings can be had. Energy savings tail off at extremely low RU allocations due to the excessive delay overheads contributing significant additional leakage energy in idling functional modules during bubbles.

The performance of the p-time relaxation is almost identical to the optimal method since both methods share the same feasible set and sport very similar objective functions. Only `jdmerge4`, `ecb_enc4`, and `motion2` at $RUlim = 2$ showed any difference between the optimal and p-time objective values.



(a) Delay overhead breakdown for varying RU allocations.



(b) Locking-induced energy overhead breakdown for varying RU allocations.

Figure 3.6: Breakdown on how varying restore unit allocations affects overall system delay overhead and total energy consumption. Delay numbers are normalized to the original length of the DFG (so an overhead of 1 indicates an 100% increase in delay), and energy numbers are normalized to the energy consumed by restore units in a conventional restore unit allocation.

3.4 Summary

In this chapter we present a true system-level restore unit resource sharing scheme for SFLL-based logic locking methods. Our method reduces locking overhead by treating RUs as a system-wide shareable resource at the implementation level as well as the design level without excessive assumptions on the capabilities of the system engineer as in previous works. We propose inserting bubbles into scheduled DFGs to mitigate resource contention for low restore unit allocations and describe two methods, one optimal and one p-time, based on ILP for minimizing the resulting delay overheads due to bubbles. Both methods perform identically at finding zero delay overhead solutions, reducing energy consumed by locking structures by 45.7% for locked MediaBench datapaths compared to the conventional circuit-level locking method.

Chapter 4: Repurposing SFLL Restore Units for Clock Gating

In this chapter we show that giving SFLL restore units (RUs) the ability to clock gate locked modules when restoring functionality lowers total dynamic power overhead. While this method can be applied to modules in isolation, we show that a system-level approach can enable even greater power savings for the same level of SAT attack resilience. Treating the look-up tables (LUTs) in SFLL as a high-level power-saving resource grants system designers greater design flexibility when considering power overheads while still maintaining granular control over SAT attack resilience.

The previous chapters described system level methods for sharing RUs across locked modules in a design so that fewer RUs were needed to restore functionality. In this chapter, we propose utilizing SFLL RUs in a new way: treating them as LUTs that enable clock gating of entire modules by performing RU/LUT lookups in an earlier clock cycle. While this technique can be readily applied at the module level, we demonstrate that a system-level design view enabled by high-level synthesis (HLS) can enable even greater power savings, all while keeping SAT attack resilience and LUT sizes in check. Our contributions can be summarized as follows:

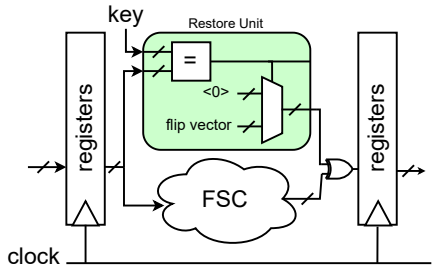
1. We formally define finding the power-optimal operation binding and locking configuration as a 0-1 integer linear program (ILP). This formulation jointly determines which operations to protect (and therefore clock gate) with SFLL and operation-to-hardware binding so that power savings are maximized without compromising SAT resilience.

2. We also present a post-binding greedy heuristic for selecting protected operations and PIPs using system-level information, but implemented at a module level. This simplifies the control circuitry needed to implement clock gating at the expense of sub-optimal power reduction.

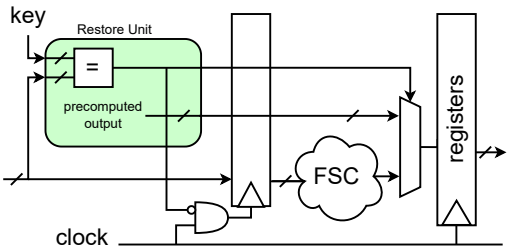
3. We evaluate our clock gating techniques on MediaBench benchmarks. Our heuristic and optimal methods on average reduce dynamic power by 24.5% and 32.9%, respectively, while maintaining maximal SAT attack resilience. Further power reductions are possible under relaxed attack resilience requirements.

4.1 Gating, Graphs, and You: Saving Dynamic Power, Securely

Conventional logic locking methods always require some amount of additional circuitry to implement. SFLL-based methods in particular require a RU (typically implemented as a LUT) to correct stripped functionality when the correct key is applied. However, only a small number of protected inputs are chosen in order to strengthen SFLL against SAT attacks and to keep LUT sizes small. By moving LUT



(a) Conventional restore unit architecture



(b) A clock-gating enabled restore unit

Figure 4.1: Two single-module datapaths locked with SFLL. 4.1a places the RU in the conventional way, while 4.1b configures the RU to clock gate the FSC when protected inputs are applied.

lookup into an earlier clock cycle (i.e. after module inputs are known but before inputs are applied), we can store precomputed output values and save them in the LUT, allowing the system to opportunistically clock gate entire locked modules.

4.1.1 Module-Constrained Gating

To illustrate this idea, consider the locked single-module datapath shown in Fig. 4.1a. A subset of the locked module's input is compared against a stored key value that characterizes the PIP. If they match, then the output of the FSC will be inverted according to the stored flip vector. Since the functionality of SAT-secure stripped modules only differs from the original design for a few inputs, it is feasible

to implement correction functionality using a small LUT, and indeed this practice is adopted by PIP-based locking schemes such as SFLL and RSAS. Note that during this operation, when the PIP are being processed by the LUT, the module itself is doing useless work and can therefore be gated.

Our proposed modification to the RU is illustrated in Fig. 4.1b. Instead of storing a flip vector, we store the module's output value corresponding to the protected input. Note that the RU is moved one clock step earlier and connected to the output of the module feeding the input FFs of the stripped module. Whenever the input to the RU matches the key, instead of correcting the corrupted output of the FSC, we clock gate the entire FSC and supply a precomputed output value to any modules or registers dependent on the FSC's output. Since the FSC's correct output value is completely known at lookup time, the overall functionality of the system is not affected. We therefore perform the LUT lookup operation before the locked circuit is scheduled to operate, but after its input value is available. Since a SAT-secure locked circuit will protect only a few inputs, the LUT only needs to contain a few entries, minimally impacting timing. Provided the correct key values are loaded into the LUT, the locked module will operate correctly. Also, if the PIP are chosen such that they have high frequency of occurrence, the clock gating will be active often leading to large power savings.

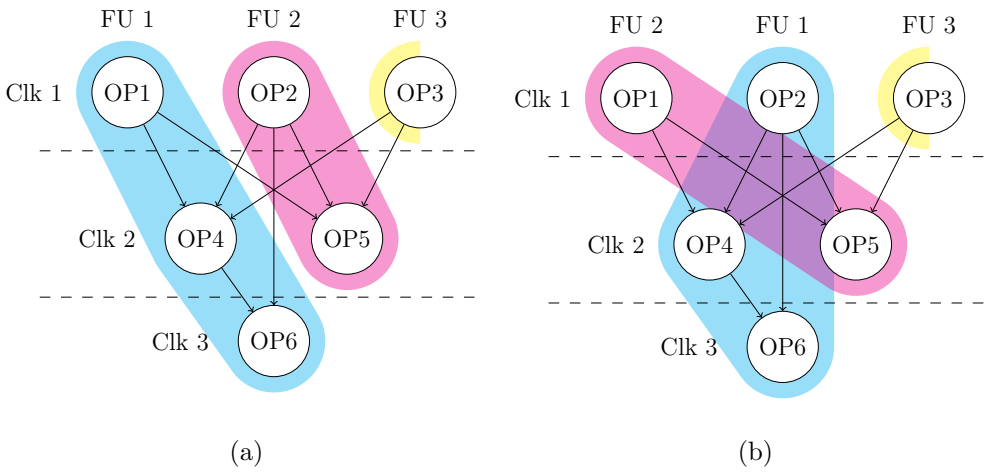


Figure 4.2: Two different resource bindings for the same scheduled DFG.

4.1.2 System-Level Gating with DFGs

While this LUT look-ahead method can be readily applied within the confines of a single module on any synthesized design, we show that system-level modifications can enable more gating opportunities and provide the same guarantees against the SAT attack by considering data dependencies in the DFG and resource binding respectively.

Consider the scheduled DFG in Fig. 4.2a. We assume that a particular DFG input has been chosen for protection, so each operation in the DFG will already have a candidate protected input. Suppose the designer is limited to placing just two LUTs (e.g. due to area limitations). If PIPs for operations 2 and 4 are chosen to be placed in LUTs, then the system can clock gate up to *three* operations: operations 2, 4, and 6. The protected inputs of operations 2 and 4 are both stored by LUTs, and can be directly gated. Operation 6 can be gated if both operations 2 and 4 are also gated because the input of operation 6 is fully determined by

the outputs of operations 2 and 4, which are also stored in our modified LUTs. Therefore, the output value for operation 6 when both operation 2 and 4 receive protected inputs is fully known at design time, and can be saved on-chip. On the other hand, if inputs of operations 1 and 2 are chosen for protection, then the number of gated operations drops from three to two when the protected DFG input is applied, since no operations in the DFG depend on only operations 1 and 2. Hence judicious allocation of limited restore LUT resources to operations impacts how many operations can be gated.

Binding of operations to hardware modules also affects the SAT-secureness of each locked module because each locked module's PIPs are chosen based on which operations are bound to each module. For example, consider the binding shown in Fig. 4.2a. Again assuming operations 2 and 4 are protected, modules FU1 and FU2 will only need to strip one input each, and therefore are maximally secure against SAT attack. If the binding is instead what's shown in Fig. 4.2b, then FU1 will need to strip the protected inputs of both operation 2 and 4, worsening its resilience against SAT attack (since we will need to protect two PIPs instead of one), while FU2 will not be locked at all. Therefore, binding should be carefully done to ensure that each module does not have too many protected inputs and remains sufficiently secure against SAT attacks.

This HLS-driven system-level approach is the one we take when clock gating modules. We select operations to protect and map protected operations to locked functional modules such we maximize the number of gated operations under protected inputs and ensure locked modules remain sufficiently secure against SAT attack.

4.2 Finding Power-Optimal LUT and Binding Configurations with ILP

Finding locking configurations that maximize power saved in this way can be expressed as a 0-1 integer linear program (ILP). To help make our formulation easier to follow, Table 4.1 lists constants, variables, and some notation used in our formulation.

First, the constraints. Besides operations with LUT-protected inputs, we also want to gate operations that depend only on other gated operations. To start, we first describe operations that only depend on other operations and not DFG inputs ($NI(V)$). The two constraints calculates whether a node's predecessors are all gated and are not all gated respectively:

$$\prod_{i \in pre(k)} g_i \leq p_v \quad \forall v \in NI(V) \tag{4.1}$$

$$1 - p_v \leq n_v \quad \forall v \in NI(V) \tag{4.2}$$

Note that while constraint 4.1 is not linear, since all variables are constrainted to be binary, it can be rewritten as a set of linear constraints.

Table 4.1: ILP variables, constants, and definitions

Notation	Definition
$G(V, E)$	a DAG representing the scheduled DFG, where nodes represent operations and edges represent data dependencies
$pre(v)$	predecessors/data dependencies of a node/operation $v \in V$
$NI(V)$	Operations in G that depend only on other operations (i.e. don't depend on DFG inputs)
$start(v)$	The first clock cycle that operation v is active for
$end(v)$	The last clock cycle that operation v is active for
$w(v)$	Expected dynamic power used by operation v
M	set of functional modules
K	set of clock cycles in the schedule
l	User-defined limit on the total number of instantiated LUTs
C_m	User-defined limit on the number of LUTs allowed for each module m, this represents the number of PIPs stripped from m which is decided by the level of SAT attack resilience desired
g_v	Variable that encodes if an operation v is gated. g_v is 1 if operation v is gated, 0 otherwise.
p_v	Variable that encodes whether or not predecessors of v are gated. p_v is 1 if all predecessors are gated, 0 otherwise.
n_v	Logical inverse of p_v
L_v	Variable for keeping track of what operation a LUT is protecting. L_v is 1 if operation v's input is protected by a LUT, 0 otherwise.
$b_{v,m}$	Variable associated operation binding. $b_{v,m}$ is 1 if operation v is bound to functional module m, 0 otherwise.
$busy_{v,m,k}$	Variable for keeping track of the schedule during binding. $busy_{v,m,k}$ is 1 if operation v is scheduled during clock k and is bound to module m.

If every operation that v depends on is gated, then the inputs of v can be fully determined from stored LUT entries, and therefore v itself can also be gated. This can be expressed as the following constraint:

$$p_v \leq g_v \quad \forall v \in NI(V) \tag{4.3}$$

If not every parent operation of v is gated, then the output of v cannot be determined from LUTs assigned to parent operations alone. However, it can still be gated if a LUT is assigned to protect it. We can express this as the following constraint, again noting that it can be rewritten as a set of linear constraints:

$$g_v n_v \leq L_v \quad \forall v \in NI(V) \tag{4.4}$$

For operations v that do depend on DFG inputs (i.e. $V \setminus NI(V)$) then the inputs of v cannot be fully known by examining inputs and outputs of other operations alone. Therefore, v can only be gated if a LUT is assigned to it:

$$g_v \leq L_v \quad \forall v \in V \setminus NI(V) \tag{4.5}$$

For area-limited designs, we can constrain the total number of LUTs to some user-defined limit l with the following inequality:

$$\sum_{v \in V} L_v \leq l \tag{4.6}$$

We use the following constraints to ensure that operations are properly bound to functional modules. Constraint 4.7 determines which which operations at what clock cycles each module is bound to:

$$b_{v,m} \leq busy_{v,m,k} \quad \forall v \in V, m \in M, k \in [start(v), end(v)] \tag{4.7}$$

Constraint 4.8 ensures that there can only be at most one operation bound to a module in any clock cycle:

$$\sum_{v \in V} busy_{v,m,k} \leq 1 \quad \forall m \in M, k \in K \tag{4.8}$$

Constraint 4.9 ensures that every operation is bound to exactly one module:

$$\sum_{m \in M} b_{v,m} = 1 \quad \forall v \in V \tag{4.9}$$

To ensure each locked functional module is still secure against SAT attack, we limit the number of LUTs (and therefore the number of PIPs per module) in each locked module. Since each LUT resource only protects a single input value, limiting the number of LUTs per module will also limit the number of PIPs, and therefore ensures locked modules will be sufficiently resilient against expected SAT attacks:

$$\sum_{v \in V} b_{v,m} L_v \leq C_m \quad \forall m \in M \tag{4.10}$$

The overall optimization objective is to maximize expected dynamic power savings due to gated operations

$$\max_{g,p,n,L,b,busy} \sum_{i \in V} w_i g_i \tag{4.11}$$

subject to the constraints given.

4.2.1 ILP Usage

A system designer seeking to secure a design using lookahead LUT-based RUs will first need to perform the scheduling and resource allocation steps of HLS so that the scheduled DFG $G(V, E)$ and available functional modules M are known.

One or more DFG inputs will need to be selected for protection, and should be propagated through the DFG so that each operation has one or more candidate protected inputs. The designer will also need to determine the maximum number of protected inputs that each module can protect (C_m) before SAT attack resilience degrades excessively. If the design is overhead constrained, then the maximum number of LUTs that can be instantiated l without exceeding area or static power limits will need to be determined.

Once these system parameters have been found, they can be encoded into the ILP described previously, and solved using one of the many available academic or commercial ILP solvers. Optimal solutions of g_v, L_v, and $b_{v,m}$ indicates which operations should be gated, which operations should have its input protected by an LUT, and which operations should be bound to which modules respectively.

4.3 Post-Binding Module-level Heuristic

While in practice ILP solvers can efficiently solve some kinds of large ILP instances, the worst-case runtime is still NP-hard. To avoid this worst-case time complexity and to simplify the control circuitry needed to implement optimal gating of downstream operations, we present a greedy PIP selection heuristic that can be implemented after operation binding has occurred.

While gating downstream operations increases the number of clock gating events without adding additional LUTs, proper implementation requires additional control circuitry to coordinate LUTs across different modules. As an example,

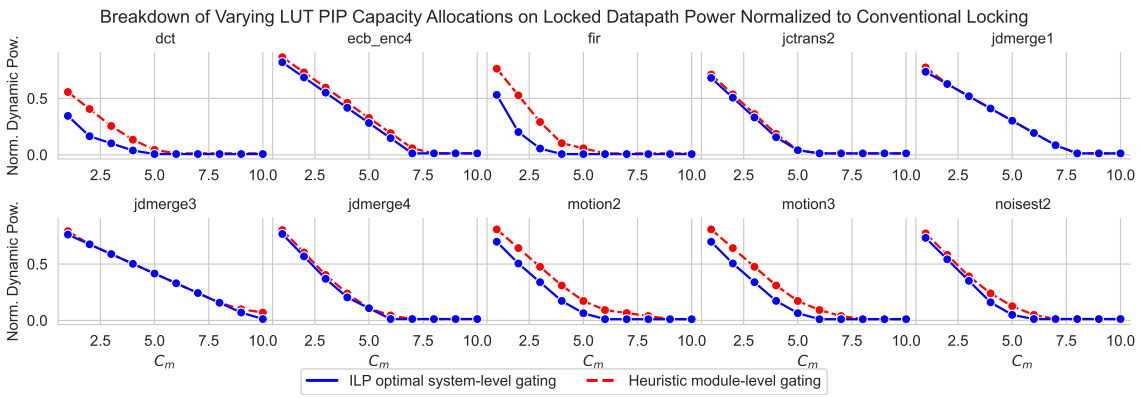


Figure 4.3: Breakdown of how system dynamic power decreases as limits on the number of PIPs per locked module (C_m) are raised for both module-level heuristic and system-level ILP methods. The same C_m value is set for all allocated functional modules. Power is normalized to a system locked with a conventional no-gating appraoch.

again consider the scheduled and bound DFG shown in Fig. 4.2a. Suppose a particular DFG input is selected for obfuscation. While it may be the designer's intention to protect particular inputs of operation 2 and 4 (denoted as PIP_2 and PIP_4 respectively) calculated from the protected DFG input, in practice other DFG inputs may result in PIPs occuring at other clock cycles besides the ones operations 2 and 4 are scheduled for. For example, there may be a DFG input where PIP_2 occurs at FU2 in clock 1 but PIP_4 does *not* occur at FU1 in clock 2, or PIP_4 occurs in clock 1 of FU1 instead of clock 2. We cannot assume PIPs will occur at only in their intended scheduled operation, so some additional control logic is needed to make sure downstream operations with unprotected inputs are only gated when all upstream operations are gated. Not gating downstream operations removes the need for control circuitry beyond what is already required by a single module at the expense of decreased power savings.

Algorithm 4.1: Greedy heuristic algorithm

Input: $G(V, E), w(\cdot), M, l, b, C$

Output: $L(v, m)$

Initialization:

1: $candidates = V$

2: $L(v, m)$ is initialized to 0 for all $v \in V$ and $m \in M$

3: $lcount = 0$

Loop:

4: **while** $lcount < l$ **do**

5: v_{sel} = highest weighted op in *candidates*

6: **if** $\sum_{w \in V} L(w, b(v_{sel})) < C_{b(v_{sel})}$ **then**

7: $L(v_{sel}, b(v_{sel})) = 1$

8: $lcount = lcount + 1$

9: **end if**

10: **end while**

11: **return** L

If operation binding is fixed and clock gating is confined to just the protected operation, then only protected inputs for each module need to be selected. This can be done efficiently with the greedy algorithm shown in Alg. 4.1. The algorithm first sets the pool of candidate gatable operations to V, (line 1) initializes the binding matrix $L(v, m)$ to 0 (line 2), and sets the total number of allocated LUTs to 0 (line 3). In each loop iteration, the operation v_{sel} with the highest dynamic power consumption is chosen (line 5). If the module that v_{sel} is bound to $b(v_{sel}) \in M$ has

not exceeded its SAT-attack resiliency requirement $C_{b(v_{sel})}$ (line 6), then v_{sel} can be added to the set of operations protected by $b(v_{sel})$ (line 7). The loop terminates when all LUTs have been assigned (line 4).

The initialization requires sorting all operations in V by their dynamic power consumption, which takes $O(|V| \log |V|)$ time. The algorithm loop runs at most $|V|$ times (the highest value l can be set to) and the summation in line 6 can be implemented in $O(1)$ time if the length of each row of L is stored and updated separately from L. Therefore the heuristic's runtime is $O(|V| \log |V|)$.

4.4 Results

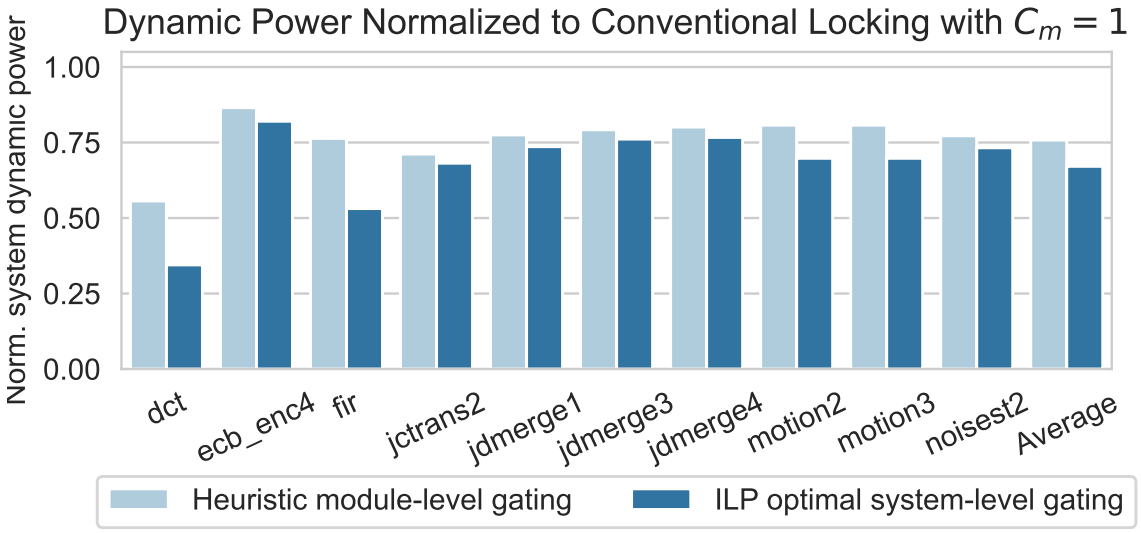


Figure 4.4: Locked datapath dynamic power for benchmarks tested, normalized to a conventionally locked datapath. Data is shown for locking solutions found with $C_m = 1$ for all modules.

To evaluate our optimal and heuristic clock gating approaches, we locked designs synthesized from C functions used in the MediaBench suite [38]. Each function's DFG was extracted using SUIF and scheduled using a path-based scheduler [48]. Up to three adders and three multipliers were allocated for each benchmark.

Locking was performed with SFLL-flex, although our method can be applied to any locking method that uses restore units. While SFLL PIPs can be of any bit width, we set the width to be the same as the input vector size of each functional module to maximize SAT attack resilience and simplify LUT design. For similar security reasons, we limited each locked module to strip at most one input (i.e. $C_m = 1 \forall m$). To evaluate the heuristic method, we used a security-aware binding method [87] to bind each operation to functional modules.

We implemented the ILP using Gurobi [18]. All experiments were performed on a desktop machine equipped with a 2.5 GHz Intel processor and 16 GB of system memory. Dynamic power was calculated for designs built using FreePDK45 [66] and synthesized with Cadence Genus tools.

To compare our techniques with the conventional no-gating approach, Fig. 4.4 shows dynamic power consumed by a locked datapath when a protected DFG input is applied to the system, normalized to the conventional no-gating method. Our optimal system-level approach reduces dynamic power consumed by the locked datapath under protected DFG inputs on average by 32.9% while ensuring that each locked module only strips one input, maximizing SAT attack resilience. Performing clock gating using our module-constrained heuristic at the same security level yields 24.5% dynamic power savings, although the advantage our heuristic holds over the

ILP binding approach strongly depends on the structure of the DFG. For example, all operations in `jctrans2`'s DFG depend directly on a DFG input, so operations can only be gated via a LUT lookup, and cannot depend exclusively on preceding operations alone. However, both `fir` and `dct` have tree-like DFGs with multiple operations that do not directly depend on DFG inputs, so a system-level DFG-aware approach is able to save more power than a module-level one.

We can also explore the security-power tradeoff by varying C_m of each allocated module, shown in Fig. 4.3. Here, we assume that the system design allows enough area overhead to add any additional LUTs needed to implement restore units. When a small number of PIPs protects each module, the opportunities for clock gating decrease and so the dynamic power savings are less. Conversely, increasing the number of protected inputs per locked module increases the number of protected operations, and therefore the number of gated operations. At very high LUT allocation limits, every operation that depends on DFG inputs can be protected, and therefore gated, which in conjunction with gating downstream operations, brings dynamic power down to near-zero for protected inputs. Naturally, this is an extreme case since for such a scenario, much of the power will be dissipated in the LUTs themselves. For the heuristic module-limited gating method, this will happen when every operation in the DFG has a LUT protecting it. However, since there is a direct inverse relationship between the number of locked inputs and expected SAT iterations required, these additional power savings comes at the cost of decreased SAT attack resilience and increased LUT power dissipation.

4.5 Summary

In this chapter we reused SFLL restore unit lookup tables to store precomputed locked module outputs and therefore clock gate entire locked modules to reduce dynamic power consumption. We show that a system-level design approach guided by operation data dependencies can yield power savings beyond what a module-level approach can save. We demonstrate both a module-constrained and a system-level clock gating approach on synthesized designs from the MediaBench [38] suite. Our proposed module-constrained and system-level approach reduces dynamic power for protected inputs by 24.5% and 32.9% respectively compared to the non-gated locking method.

Chapter 5: High Level 3D IC Co-Design for Timing Optimization

3D circuit integration presents a new dimension in the design and packaging of ICs by stacking conventionally fabricated 2D monolithic chips in the vertical dimension, with connections facilitated by through-silicon vias (TSVs). 3D ICs promise improved density, shortened wirelength, and improved performance [74]. However, 3D chips present additional design complexity as each chip still needs to be separately fabricated, and TSV routing overheads need to be considered.

Many other works have focused on improving the 3D IC RTL-to-GDS flow. One method is to "trick" a conventional 2D tool-flow into generating a valid 3D design [49, 31]. Others propose extending 2D design methods to 3D (i.e. so-called "true-3D" methods) [26, 43, 20]. However, these works focus their attention on the physical design of a 3D IC after the architecture has been synthesized.

In this chapter we describe a method to optimize the datapath for timing in a 3D IC design by designing the datapath in the loop with physical placement. We integrate HLS operation binding with 3D module-level placement in order to reduce TNS violations by directly optimizing binding for timing at each iteration

of the binding-placement loop. We treat resources as fixed-outline 2D modules to be placed on a 3D stacked chip design, and integrate module-module interconnect information as constraints on our binding methods. Our contributions are

1. A novel physically-aware binding method based on mixed-integer linear programming (MILP) for optimizing datapath synthesis for minimizing TNS.

2. A dynamic weighting approach compatible with both force-directed 3D analytic placers and simulated annealing that biases placement into timing-valid module placements.

3. We show that our integrated binding-placement co-design flow on average reduces total negative slack (TNS) by 62% and 92% when compared to a conventional placement-unaware binding and unweighted module placement method for 2D and 3D floorplans respectively.

5.1 Analytical Global Placement

The global placement step during physical design concerns itself with the general placement of macros and standard cells, although in this work we will focus on the placement of fixed-size macros. Given a collection U of n nodes representing placement objects and a set of hyperedges E representing nets connecting them, the problem can be represented as a constrained optimization problem. The objective is to find a placement of nodes $\mathbf{x} = (x_1, x_2, ..., x_n, y_1, y_2, ..., y_n)^T$ such that the total

wirelength $W(\mathbf{x})$ and placement density $D(\mathbf{x})$ is minimized:

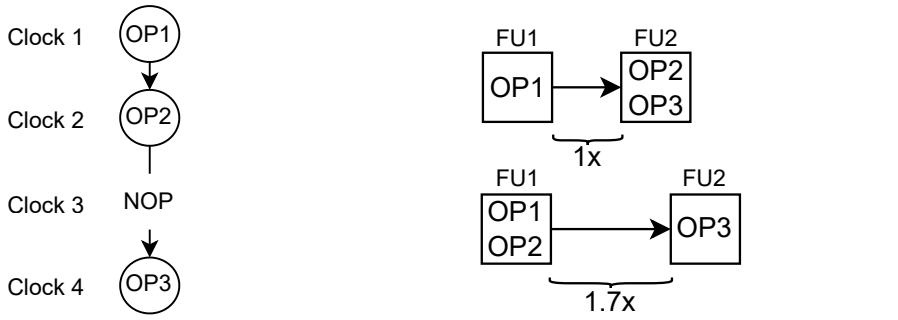
$$\min_{\mathbf{x}} W(\mathbf{x}) + \lambda D(\mathbf{x}) \tag{5.1}$$

Where λ adjusts the tradeoff point between wirelength and density. Multiple density functions have been proposed [27, 28, 17, 42, 11]. The current state of the art uses density functions based on the electrostatic model used in the ePlace[42]/RePlAce[11] family of placers, which models each placement object as a electrostatically charged object. The density function $D(\mathbf{x})$ can then be thought of as the total electric potential given a particular placement $\mathbf{x}$, which the optimizer seeks to minimize by finding its gradient.

The wirelength of each net is traditionally modeled by the half-perimeter wirelength (HPWL) model at the global placement stage [25]. However, since analytic global placers use gradient-based optimization and the HPWL is not differentiable everywhere, various approximation methods have been proposed, including log-sum-exp and weighted-average (WA). We use the WA model in this chapter as it has been shown to yield lower estimation errors compared to LSE [20].

5.2 Module Placement at the System Design Level

Our key insight is that module binding and module placement can be tightly coupled together into a single optimization loop to improve timing. Timing delays between operations derived from the scheduled DFG can be used to inform binding about timing constraints between physically placed modules, and therefore search for resource bindings that minimize timing violations. Timing information can



(a) Scheduled DFG

(b) Possible bindings and timing-valid module placements

Figure 5.1: 3-operation linear scheduled DFG with two possible binding solutions

then be passed along to a force-directed macro placer, which we utilize using our dynamic weighting approach. We will start by illustrating the above in the following example, followed by a description of our binding method and our modification to the electrostatics-based 3D placer ePlace-3D.

5.2.1 An Illustrative Example

We will use Fig. 5.1 to illustrate how binding can affect timing closure between modules. Fig. 5.1a shows a scheduled DFG with three operations sequenced in a chain. OP2 is scheduled in the next clock after OP1, and OP3 is scheduled two clocks after OP2. In the first binding shown in Fig. 5.1b, OP1 is bound to FU1 and OP2, OP3 are bound to FU2. Since OP2 occurs directly after OP1, data needs to be transferred from FU1 to FU2 within one clock cycle. In the second binding, both OP1 and OP2 are bound to FU1, and OP3 is bound to FU2, therefore the data transfer can take up to two clock cycles.

We can then represent the two modules as floorplan blocks, as shown in Fig. 5.1b. Suppose T is the clock period, and each module has a propagation delay estimate of $T/2$. In the first binding, the available timing budget between FU1 and FU2 is only one clock period T, $T/2$ of which is taken up by the delay of FU1, leaving $T/2$ left for the interconnect delay between FU1 and FU2. However, the second binding leaves $3T/2$ for the interconnect delay, a 3x increase in available timing budget. Translated to placement, that means a 1.73x allowable increase in wirelength (due to the quadratic relationship between wire length and wire delay). If the two modules were already placed close to each other, then either binding may work. However, if placed further apart, then only the second binding may be able to meet timing closure. Conversely, if the binding is fixed to the first one shown, then the module placer has less freedom to move FU1 and FU2 if timing is to be met. This motivates the need for a binding algorithm that is aware of these physical realities.

5.2.2 Physically-Aware Binding

Finding a timing-valid binding based on a particular resource placement can be expressed as mixed-integer linear program (MILP). To make our formulation easy to follow, Table 5.1 lists the constants and variables that we use.

Table 5.1: MILP variables, constants, and definitions

Notation	Definition
U	The set of functional modules to be placed
O	The set of operations in the DFG
T	The number of clock cycles in the schedule
$D_{i,j}$	Delay matrix representing the estimated minimum achievable delay between modules i and j extracted from a module-level placement, with the intrinsic delay of module i subtracted.
$P_{x,y}$	Maximum allowed delay between operations x and y, derived from the schedule and target clock frequency. Only specified for operations with direct data dependencies on each other.
$S_{o,t}$	1 if operation o is scheduled for clock t, 0 otherwise
$b_{o,u}$	Binary variable indicating operation binding. 1 if operation o is bound to resource u, 0 otherwise. Only defined for valid o, u mappings.
$c_{x,i,y,j}$	Binary variable capturing how operation binding affects module-to-module data movement. 1 if operation x is bound to module i and operation y is bound to module j.
$s_{i,j}$	Non-negative variable capturing the negative slack between modules i and j. A value of 0 indicates positive slack, any positive values of s indicates the amount of negative slack on the path between i and j
ws	Non-negative variable representing the worst negative slack

The first few constraints ensure that any bindings in the feasible solution space are valid bindings. The first constraint limits the number of operations that can be bound to a module in any clock to at most one:

$$\sum\nolimits_o b_{o,u} S_{o,t} \leq 1 \quad \forall u \in U, t \in T \tag{5.2}$$

Additionally, every operation must be bound to a module:

$$\sum\nolimits_u b_{o,u} = 1 \quad \forall o \in O \tag{5.3}$$

From there we can calculate c, which captures every pair of operation bindings:

$$b_{x,i} b_{y,j} = c_{x,i,y,j} \quad \forall x, y \in O \quad i, j \in U \tag{5.4}$$

Note that the above constraint is not linear, but because b is a binary value, it can be rewritten as a set of linear constraints.

From these prerequisite constraints we can begin to incorporate physical timing information. If we only want to find a binding that meets timing constraints with no negative slack, the following constraint could be used:

$$\sum\nolimits_{i,j} c_{x,i,y,j} D_{i,j} \leq P_{x,y} \quad \forall x, y \in O \tag{5.5}$$

This constraint finds the relevant physical delay $D_{i,j}$ between modules i and j based on what physical modules x and y are bound to via $c_{x,i,y,j}$ and ensures its less than the schedule-defined delay constraint $P_{x,y}$. Note that we assume every operation's data dependencies happen in previous clocks. If multiple operations can happen in the same clock, 5.5 can be modified by considering a sequence of nodes that occur in the same clock instead of a single node x.

The MILP as-written so far with a objective function of 0 would force the solver to find a binding that meets timing as-is, with any bindings with negative slack being infeasible solutions. However, if the physical placement is especially bad or the target clock frequency too constrained, the feasible set may end up being empty. Additionally, we would like some kind of cost function to guide our modified placement tool. A few choice objective functions are available to us:

Total Negative Slack: Total negative slack captures the total slacks of all failing timing paths, which can be a better indicator of overall placement quality [39]. The negative slack of each module-module path $s_{i,j}$ can be found by modifying 5.5 to be:

$$\sum_{i,j} c_{x,i,y,j} D_{i,j} - P_{x,y} <= s_{i,j} \quad \forall x, y \in O \quad \forall i, j \in U \tag{5.6}$$

Note that we also constrain $s_{i,j}$ to be non-negative to prevent positive slacks from being captured. The optimization objective then is

$$\min_{b,c,s} \sum_{i,j} s_{i,j} \tag{5.7}$$

subject to eqs. (5.2) to (5.4) and (5.6).

Worst Negative Slack: A similar objective function can be used for worst negative slack. First we add a new constraint to find the worst negative slack value ws:

$$s_{i,j} <= ws \quad \forall i, j \in U \tag{5.8}$$

The objective function is simply

$$\min_{b,s,ws} ws \tag{5.9}$$

subject to eqs. (5.2) to (5.4), (5.6) and (5.8).

Number of Failing Timing Paths: The number of failing timing paths can be a useful global metric for determining how many paths need fixing. Finding the number of failing timing paths (i.e. paths between modules with negative slack) is equivalent to minimizing the zero "norm" of s, which is not convex. A common relaxation of the zero norm is to use the 1-norm, however the 1-norm s is equivalent to TNS, so TNS should be used instead.

5.2.3 Timing-Weighted Wirelength Cost Function

Our MILP formulation generates a valid datapath architecture along with timing constraints at connections between modules. Based on these timing constraints, we can estimate the longest wire that can reasonably accommodate a timing constraint for each net. For each net $e \in E$, we have a target net length $target_e$. Total wirelength is the most common metric used for determining the quality of a global placement. The most commonly used wirelength metric for global placers $W(\mathbf{x})$ is the HPWL:

$$W(\mathbf{x}) = \sum\nolimits_{e \in E} \mathrm{HPWL}_e(\mathbf{x}) \tag{5.10}$$

$$= \sum\nolimits_{e} \mathrm{HPWL}_{e_x}(\mathbf{x}) + \mathrm{HPWL}_{e_y}(\mathbf{x}) + \mathrm{HPWL}_{e_z}(\mathbf{x}) \tag{5.11}$$

We propose a flexible dynamic weighting method that increases the weight of timing-constrained nets as the wirelength approaches the timing-derived wirelength target by using an exponential function:

$$W(\mathbf{x}) = \sum\nolimits_{e} HPWL_e(\mathbf{x})(1 + \exp((HPWL_e(\mathbf{x}) - target_e)/\gamma) \tag{5.12}$$

where γ controls the penalty of exceeding a particular timing-derived wirelength target $target_e$. For 3D ICs, we account for the delay contribution of TSVs by weighting the Z dimension by the expected RC delay of a single TSV. The advantage of using this weighting method over previous timing-driven placement methods such as [39] is that we can directly incorporate a timing budget into the placer's cost function without needing to go through a costly STA step.

Because we are working with wire delays and not path delays, we can determine values of $target_e$ before we begin module placement, and the fully differentiable nature of our weight function means the modified wirelength term is flexible enough to be applied to any placer that uses wirelength in its cost function. The next two sections demonstrate how we apply this weight to the two major 3D components of ePlace-3D applicable to modules: the simulated annealing-based macro legalization step and the force-directed 3D global placement step.

Timing-Weighted Wirelength for Simulated Annealing: The cost function used for low temperature simulated annealing in ePlace is similar to that of the analytic global placer's cost function (without the overlap term):

$$\text{HPWL}(\mathbf{x}) + \mu_D D(\mathbf{x}) \tag{5.13}$$

where $D(\mathbf{x})$ is the electrostatics-based density function and μ_D is statically set to $\text{HPWL}(\mathbf{x})/D(\mathbf{x})$. Incorporating our timing-based wirelength weight is as simple as modifying the HPWL term to include the exponential weight for timing-constrained nets:

$$\sum\nolimits_e \text{HPWL}_e(\mathbf{x})(1 + \exp((\text{HPWL}_e(\mathbf{x}) - target_e)/\gamma) + \mu_D D(\mathbf{x}) \tag{5.14}$$

Since the simulated annealing step directly calculates the HPWL for determining the total improvement at every movement step, no further modifications are necessary, greatly simplifying implementation.

Timing-Weighted Wirelength for Force-Directed Placement ePlace-based global placers determine the direction to move each placement object by finding the gradient of the cost function. Since HPWL is not differentiable everywhere, approximations of it have been proposed, the leading of which is the weighted-average $WA(\mathbf{x}$ approximation [20]. We can apply our weighting term to the cost function (5.1) by substituting HPWL with its WA approximation:

$$\sum_{e \in E} \mathrm{WA}_e(\mathbf{x})(1 + \exp((\mathrm{WA}_e(\mathbf{x}) - target_e)/\gamma) + \lambda D(\mathbf{x}) \tag{5.15}$$

Since our weight is continuous everywhere when applied to the WA approximation, a closed form expression for the gradient of our weighted wirelength term $W(\mathbf{x})$ can be found:

$$\nabla \mathrm{WA}(\mathbf{x})(1 + \exp((\mathrm{WA}_e(\mathbf{x}) - target_e)/\gamma)(1 + \mathrm{WA}(\mathbf{x})/\gamma) \tag{5.16}$$

making implementation relatively straightforward. Additionally, the gradual change in gradient as WA approaches $target_e$ means wirelengths cannot "rubber-band" and oscillate around $target_e$.

5.3 The Overall Co-Design Flow

Algorithm 5.1 describes our entire 3D IC co-design flow. We first initialize module-module delays to 0 (line 1), so that the MILP solver can see if the target clock period is even feasible with just the known module propagation delays. In the

Algorithm 5.1: Timing-directed co-design algorithm

Input: SDFG, resource allocation, clock period in

Output: placement out

Initialization :

1: Initialize placement delays to 0

Optimization Loop:

2: **while** iteration limit not reached **do**

3: binding = bind(calculate_delays(placement)

4: macro_netlist = generate_datapath(binding)

5: placement = ePlace-3D(macro_netlist)

6: **if** TNS(placement) == 0 **then**

7: **return** placement

8: **end if**

9: **end while**

10: **return** placement

loop, we first call the binder (line 3) and find a binding that minimizes TNS. After binding, we generate values for $target_e$ based on the length of a wire with the same RC delay as the timing budget for each net. We pass the generated datapath and target wirelengths to the 3D force-directed placer (line 5). If the placement returned by the placer has 0 TNS at this point, we can return this placement (lines 6, 7). Otherwise the loop continues until we reach an iteration limit (line 2).

The designer would need to provide a scheduled DFG, a resource allocation, and module outlines and delay estimates for each allocated resource. The output will be a timing-optimized placement along with estimated slack values along each net connecting modules in the floorplan. As we operate at a higher abstraction level than standard cells, we assume TSVs have zero physical width and do not take up any placement area. The placement can then be passed along to more detailed physical design tools.

5.4 Results

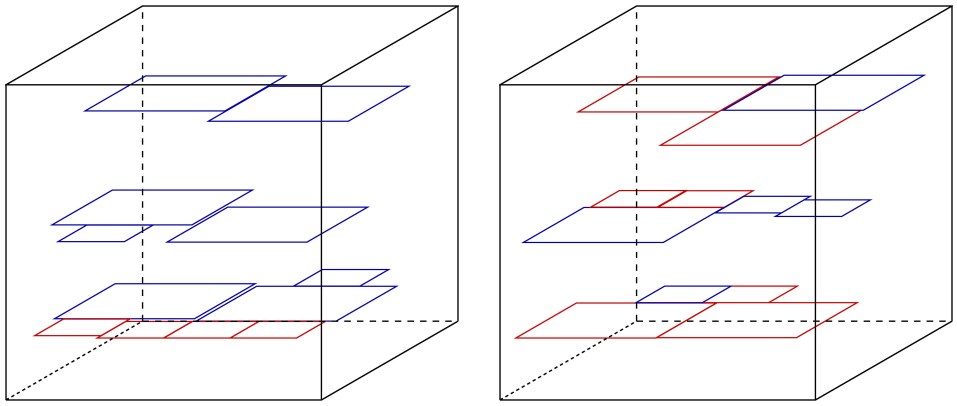
To evaluate our binding and module placement co-design flow, we extract functions from the MediaBench benchmark suite [37] for use with HLS. The DFG of each benchmark was extracted with SUIF and scheduled with a path-based scheduler [48]. As the DFGs extracted from these benchmarks (and hence the required module allocation) were small, we increased their size by duplicating the base DFG and appending them together to create a larger benchmark to stress test our method. We used a security-aware binding algorithm as our baseline reference binding method [87]. We limited the number of iterations of our co-design flow to 10 to keep runtimes manageable.

We used ePlace-3D [43] as our true 3D placer, although we note that any placer that utilizes HPWL (including RePlAce [11]) in its cost function can be used. As ePlace-3D is designed for mixed-size 3D placement, we modify its placement flow to tailor it for module-level placement by skipping the 3D and 2D global standard

cell placement steps. Data for the conventional method used our module-optimized ePlace-3D implementation with no wirelength targets set. We empirically found that a γ value of 20000 yielded the best results for our benchmarks without causing numerical issues. As we are performing high level module placement and not a more detailed global placement step, we set the pin locations of all module-module nets to be at the center of each module's macro, and the macro's length/width is added to HPWL targets accordingly. In a similar vein, we model TSVs as having zero volume.

Gurobi was used to implement the MILP binding optimization program with a timeout limit of 20 minutes. Cadence Genus and Innovus were used to generate module macro blocks from RTL synthesized with Nangate45 [66] with an aspect ratio of 1. OpenSTA [1] was used to characterize each functional module's timing delay. OpenROAD [3] was used to initialize the die outline with a placement density of 70%, and RosettaStone [24] was used to translate LEF/DEF files from OpenROAD to the academic Bookshelf format used by ePlace-3D.

Figure 5.3 compares how our co-design technique compares against a conventional datapath synthesis and placement flow over all clock period targets tested. Data are for 3D floorplans are normalized against the TNS obtained from the conventional approach in 3D, and similarly for 2D. Our co-design flow was able to significantly reduce TNS in all cases, especially in 3D floorplanned test cases, while TNS reduction in the 2D case was less. This is due to the lesser degree of placement freedom in 2D, causing higher congestion and longer wirelengths which has been well documented in previous works [44].



(a) Baseline 3D Placement (b) Co-designed 3D Placement

Figure 5.2: 3D Module Placements for the `motion2` benchmark. An aggressive clock period of 3ns was set for the co-design flow. Large modules are 32-bit multipliers, small modules are 32 bit adders.

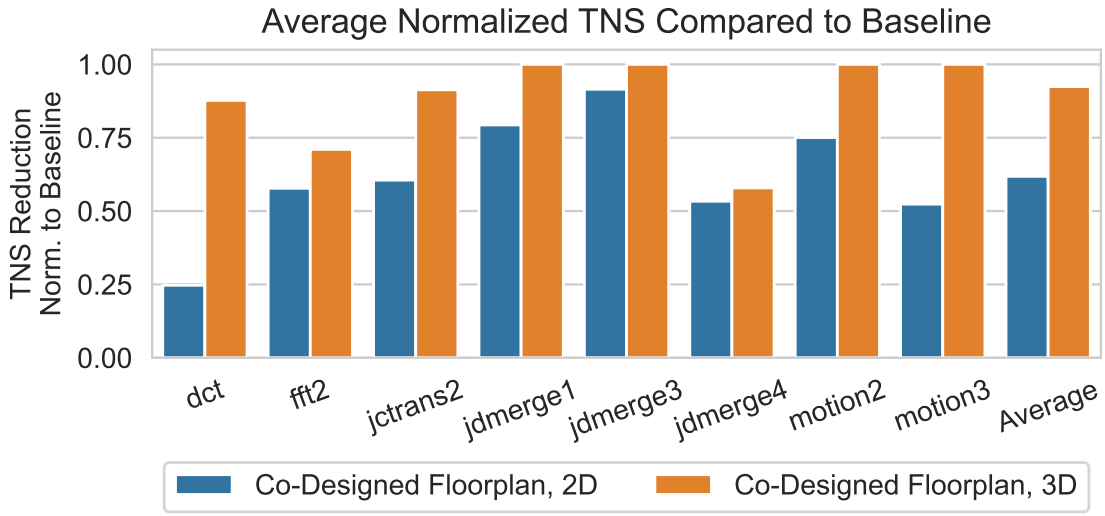
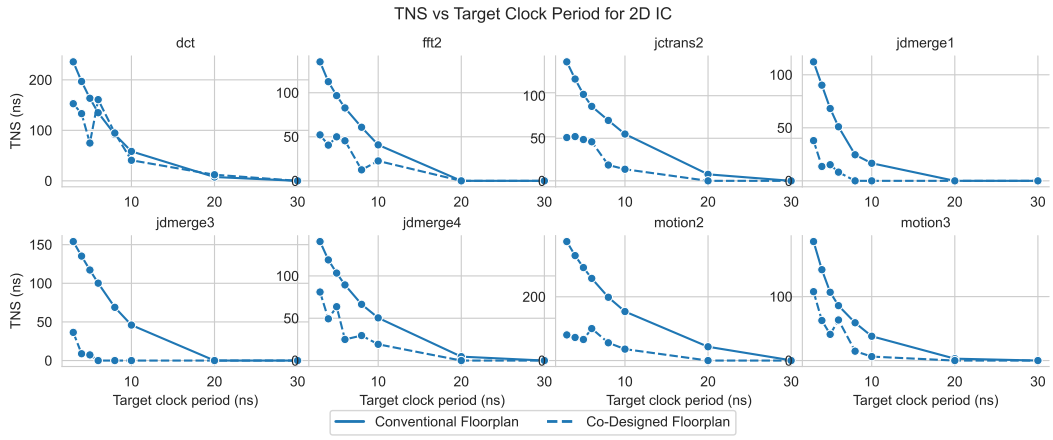
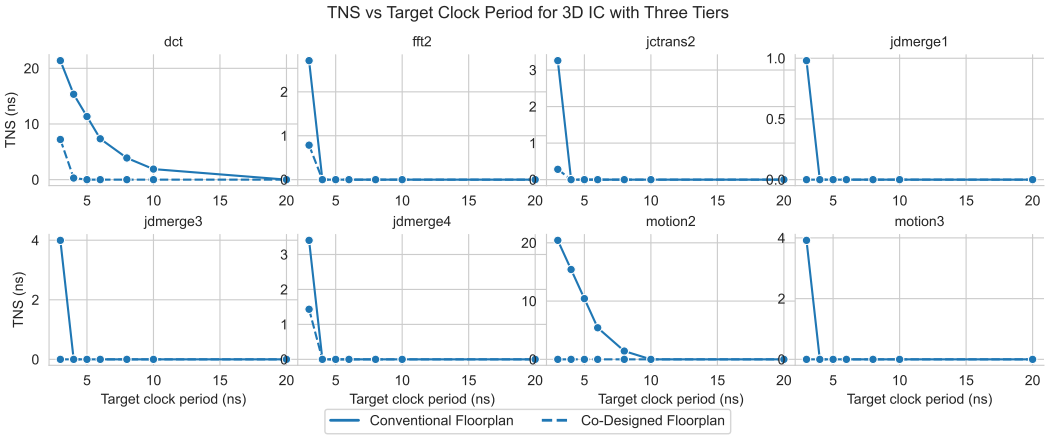


Figure 5.3: Reduction in total negative slack using a co-designed datapath and physical placement normalized to the corresponding 2D or 3D conventional binding and module placement approach. Data shown are averaged over all clock periods tested.



(a) 2D Results



(b) 3D Results

Figure 5.4: Breakdown of how TNS increases as the target clock period decreases for both our co-design approach and a conventional binding and module placement toolflow when applied to a 2D chip. All units are in nanoseconds.

Figures 5.4a and 5.4b show how TNS changes with tighter clock period constraints for 2D and 3D respectively. In all cases except one, our method was able to meet or beat the TNS of a conventional flow. Moreover, our co-design flow was able to reach 0 TNS in some cases where the conventional approach could not. The only case where the co-design flow did worse than the baseline was the `dct` benchmark at a clock period of 6ns for a 2D design, which upon further investigation was due to the iteration limit of 10 we used. By increasing the iteration limit to 20 we were able to find a module placement that yielded a TNS of 133ns, marginally better than the baseline TNS of 135ns. The small difference in TNS between the conventional binding and placement and our method for the other target clock periods in 2D `dct` tests also indicates that the conventional method was already close to timing optimal.

5.5 Summary

In this chapter we present a HLS and physical placement co-design flow for use with 3D ICs. Our method improves TNS significantly by proposing a physically-aware timing-optimized binding method in conjunction with timing-informed module placement. We apply the co-design flow on DFGs extracted from MediaBench and compare TNS values for a conventional flow where HLS and placement take place separately and our flow which integrates physical placement with HLS. Overall our flow was able to reduce TNS by 62% and 92% on average when compared to the conventional approach for 2D and 3D floorplans respectively.

Chapter 6: High Level 3D IC Co-Design for Dynamic Power Minimization

As devices become increasing small, particularly in mobile and edge applications, power and energy consumption becomes increasingly important, especially for devices with limited battery capacity and constrained cooling solutions. Additionally, the low-hanging power optimization strategies (Dennard scaling, Vdd lowering) have either run out, or are traded off for lower performance and power.

Previous works have also investigated low power HLS. [62] proposed a scheduling method based on integer linear programming (ILP) for minimizing module level peak switching power and and LP-based binding method for minimizing switching activity at the inputs of modules. [84] proposed a low power HLS binding method by finding binding assignments that lower switching power consumed by both hardware resources and interconnects using empirically-derived cost functions and module floorplanning information.

Previous works on low power high-level design have either approached optimal power optimization without considering the physical realities of interconnect power [62] or used heuristics derived from physical placement data in the loop with HLS [84]. No work to our knowledge has tackled interconnect-aware power-optimized high level and physical design with a mathematically rigorous approach.

This chapter introduces a novel interconnect-aware binding method that incorporates a mathematically formal model for estimating and optimizing total power consumption given a particular physical module placement. We then use this binding information to directly guide force-directed and simulated-annealing based placers to low-power placement solutions while maintaining timing correctness. Our method is particularly suitable for 3D ICs as the power dissipation of TSVs can be substantial, and thus should be captured and incorporated into high level architectural decision making. The contributions of this chapter are:

1. A mathematically rigorous ILP formulation of the interconnect aware switching activity-based power minimization binding problem with timing constraints.

2. A new wirelength-based power aware cost function for global placers that adjusts the wirelength penalty weight of each net by the net's expected switching activity while maintaining timing correctness.

3. A co-optimization loop integrating the two methods together in order to find a physically realizable power-minimal and timing valid datapath architecture and module placement.

4. Evaluations of the integrated method on scheduled DFGs extracted from MediaBench [37] that demonstrate 31% and 28% switching power reduction for 2D and 3D ICs respectively.

6.1 Preliminaries

6.1.1 Mutual Switching Activity Graphs for Low Power Binding

[62] originally proposed a binding method to optimally minimize switching activity within datapath modules. The method relies on a multistage graph that encodes all possible mutual input switching activities between all pairs of operations that can be scheduled in sequence on the same resource. For this chapter we generalize the originally proposed multistage graph into a similarly defined structure, the mutual switching activity graph (MSAG). Similar to the previously proposed multistage graph, the MSAG is a comparability graph based on the operation schedule that contains all possible orderings of operations bound to the same functional resource.

Given a scheduled DFG (SDFG) $G(V, E, s)$ and system test traces, a MSAG is a acyclic directed graph $G_A(V_A, E_A, \alpha)$ that uses operations as nodes along with a $START$ and END node (i.e. $V_A = V \cup \{START, END\}$). The $START$ node has no incoming edges and the END node has no outgoing edges. An edge between two nodes $o, p \in V_A$ exists if p is scheduled after o (i.e. $s(o) < s(p)$. Every edge has an associated edge weight $\alpha_{o,p}$ which represents the average input switching activity caused by binding p immediately after o to the same module, calculated from system traces.

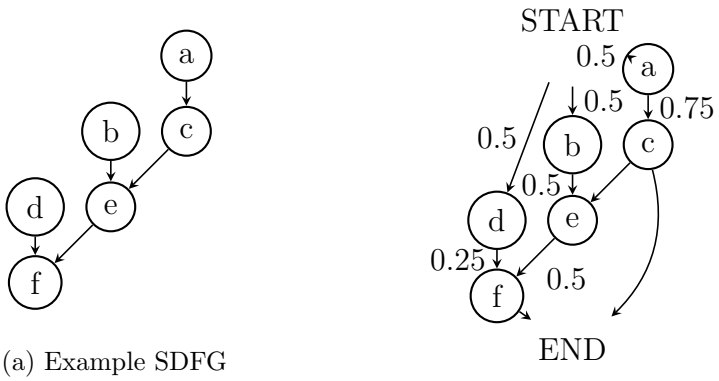
6.1.2 HPWL and Timing-weighted Placement

A common metric used to evaluate global placement quality is the total half-perimeter wirelength (HPWL) of all nets in a design. For an individual net the HPWL is defined as half of the perimeter of the smallest bounding box for that net. Global placers can target minimizing the total HPWL of a synthesized design by utilizing it directly as a cost function (as is the case with simulated annealing based methods) or by using smooth approximations such as the weighted average (WA) approximation [20].

The previous chapter proposed dynamically weighting the HPWL and WA approximation cost functions used by global placers based on how close each net's length was to violating timing constraints. For a net that connects two modules u, v together and a timing budget for that net, an estimated target wirelength $target_{u,v}$ with the same expected RC delay as the timing budget can be calculated from the expected technology parasitic characteristics. The cost function used by the placer given a placement is $\mathbf{x}$ then changed to be:

$$\sum_{(u,v)\in N} HPWL_{u,v}(\mathbf{x})(1+\exp((HPWL_{u,v}(\mathbf{x})-target_{u,v})/\gamma) \tag{6.1}$$

where N is the set of nets in a design, $HPWL_{u,v}(\mathbf{x})$ is the HPWL for the net connecting modules u and v, and γ controls how rapidly the net weight increases as $HPWL_{u,v}(\mathbf{x})$ approaches $target_{u,v}$.



(a) Example SDFG

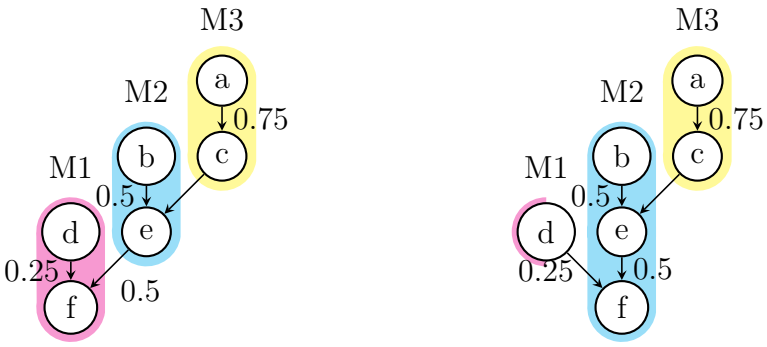
(b) Example MSAG

Figure 6.1: Example SDFG and associated MSAG. Most edges of the MSAG are not shown and only some MSAG edge labels are shown for readability.

6.2 Motivating Example: High Level Physical Placement

In a traditional design flow, physical design is typically done after after architectural and logical synthesis, so design flexibility is limited. Any physical design details considered at higher design abstractions are usually limited to slicing floorplans [84, 65, 10].

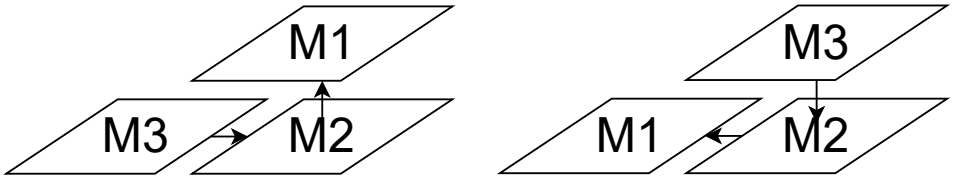
In this section we will use an example SDFG to illustrate how incorporating both module placement during high level synthesis *and* high level architecture details during physical placement can influence switching power, especially in a 3D IC context.



(a) Binding 1

(b) Binding 2

Figure 6.2: Two potential binding solutions for the same SDFG. Switching activities between some operations are shown as edge labels.



(a) 3D Placement 1

(b) 3D Placement 2

Figure 6.3: Two potential placement solutions for three modules placed on two tiers in a 3D IC.

6.2.1 Placement Informed Binding

Using the switching activities for the given SDFG, the interconnect unaware method proposed in [62] may result in the binding shown in Fig. 6.2a, since this minimizes the sum total switching activities *within* each module. However, this would neglect interconnect capacitance, which is non-negligible.

Suppose the three modules are already placed on a two-tier 3D floorplan shown in Fig. 6.3a. Modules M2 and M3 are located in the same tier and are connected by a net with a total wire capacitance of 1. Module M1 is located in the next tier up, and the total TSV capacitance connecting the two tiers is 3. We also assume each module has an intrinsic switchable capacitance of 1. For this example we assume the placement does not cause any timing violations.

Since operation c has an outgoing data edge to operation e, a wire must connect M3 to M2. Switching activity at the input of M3 will therefore switch not just the intrinsic capacitance of M3 but also the interconnect capacitance of the M3-M2 wire. Including interconnects, the total switchable capacitance for modules {M1, M2, M3} is $\{1, 1+3, 1+1\}$. Additionally, given the binding in Fig. 6.2a and the MSAG, the total switching activities at each module (including the switching activity due to coming out of reset) is $\{0.5+0.25, 0.5+0.5, 0.5+0.75\}$. The total expected switched capacitance, which is directly proportional to dynamic power, is then $1(0.75) + 4(1.0) + 2(1.25) = 7.25$[1].

[1]Note that we assume the α module switching activity values can also be used for interconnects, which may not be the case in practice. Separate α values can be used for interconnects without significantly changing our cost function.

If we instead change the binding to that shown in 6.2b without changing module placement, we change not just the switching activity profile of each module but also the way modules are connected to each other, and therefore what capacitances are switched by each module. Since M1 now sends data to M2, the associated capacitance of the TSV that connects the two will now be switched by M1, and hence the contribution to total expected switched capacitance will be due to the switching activity of M1 (and not M2 like in the original binding). The switchable capacitances of each module are now $\{1+3, 1, 1+1\}$ and the total switching activity of each module is now $\{0.5, 0.5+0.5+0.5, 0.5+0.75\}$, therefore the total expected switched capacitance is $4(0.5)+1(1.5)+2(1.25)=6$, a 17% reduction.

Although the amount of switching activity at M2 increased, it also experienced a decrease in switched capacitance due to the change in connectivity resulting in the high capacitance TSV being switched by M1, which, due to the change in binding, now also experiences less switching activity. This illustrates how informing the binding algorithm of physical module placement and interconnect lengths can lead to reductions in total switching power.

6.2.2 Architecture-Informed Placement

Bringing high level switching activity details from HLS down to physical design can also yield significant power improvements. We will again consider the binding shown in Fig. 6.2a. Two possible physical module placements are shown in Fig. 6.3.

Assuming every module has the same outline, *both placements have the same total HPWL*, and therefore a placer using only total HPWL to assess placement quality will not be able to distinguish the two.

Since we have access to a full behavioral description of the overall architecture from the SDFG (not just a structural RTL-level understanding), switching activities for each wire/TSV connecting modules together can be estimated. As wirelength is directly proportional to capacitance, the placer can use binding-derived switching activities to directly estimate overall switched interconnect capacitance and therefore the system interconnect's contribution to overall dynamic power.

For module placement 1, the switched capacitance for each module (including both module-intrinsic capacitance and wire/TSV capacitance) is again $\{1, 1+3, 1+1\}$. Combined with the switching activities of each module for binding 1, the total estimated switched capacitance (which is proportional to dynamic power) is $1(0.75)+4(1.0)+2(1.25) = 7.25$. For placement 2, the capacitances are $\{1, 1+1, 1+3\}$ and the corresponding total switched capacitance is $1(0.75)+2(1.0)+4(1.25) = 7.75$, an increase of 7%. Therefore, the module placer should be made aware of wire switching activities and focus optimization effort on wires that switch more heavily in order to reduce total switching power.

6.3 Low Power System Design and Placement

This section will describe our interconnect-aware ILP binding method for optimizing dynamic power and our power-aware physical placer. We then propose an optimization loop that jointly finds a power-minimal system architecture and module placement. We conclude the section with a short usage guide.

6.3.1 Interconnect-aware Low Power Binding with ILP

All constants and variables used in our formulations are shown in Table 6.1 to ease readability. Note that many constraints use logical AND operators, which as-written are not linear. However, all variables used are binary, and therefore each AND operator can be rewritten as a set of linear constraints.

We first start with basic constraints on operation binding. Every functional module can execute at most one operation per clock,

$$\sum\nolimits_{o \in V, s(o)=t} x_{o,u} \leq 1 \quad \forall u \in U, t \in 0, ..., T-1 \tag{6.2}$$

and every operation must be bound to a module:

$$\sum\nolimits_{u \in U} x_{o,u} = 1 \quad \forall o \in V_A \tag{6.3}$$

Note that we use V_A as the set of operations instead of V in (6.3), so $START, END$ nodes of the MSAG can be captured and "bound" to all modules. This is necessary for calculating the total switching activity of each module later on.

Table 6.1: ILP variables, constants, and definitions

Notation	Definition
$G(V, E, s)$	The scheduled DFG
$s(o)$	The clock cycle operation o is scheduled for
$\alpha_{o,p}$	Mutual switching activity between operations o, p
$G_A(V_A, E_A)$	The mutual switching activity graph
$pred_A(o)$	The set of predecessor nodes of o in G_A
$succ_A(o)$	The set of successor nodes of o in G_A
U	The set of allocated functional modules
$D_{u,v}$	Physically realizable delay between modules $u, v \in U$, including propagation delay through module u.
C_u^M	The intrinsic capacitance of module u
$C_{u,v}^N$	The estimated total capacitance of the wires/TSVs needed to connect module u to module v.
λ	Clock period
T	Length of the schedule in clocks
$t_{u,v,o,p}$	Captures interconnect switching. 1 if there is a wire connecting module u to module v and a switching edge $(o, p) \in E_A$ where o and p are bound to FU u
$x_{o,u}$	Captures operation binding. 1 if operation o is bound to module u, 0 otherwise. Only defined for valid o, u bindings.
$y_{u,o,p}$	Captures operation binding-derived switching activity. 1 if operation p is a direct successor of operation o in G_A, both o and p are bound to module u, and p is the next operation to run on u after o.
$z_{u,v}$	Captures module connectivity. 1 if $u \neq v$, there exists two operations o, p such that $(o, p) \in E$ and o is bound to u and p is bound to v

To make the binding method interconnect-aware, we use DFG data edges E and the binding x to determine what the module connectivities z will be in the datapath architecture:

$$x_{o,u} \wedge x_{p,v} \leq z_{u,v} \quad \forall u, v \in U, u \neq v, (o,p) \in E \tag{6.4}$$

To calculate switching activities at each module, we modify the optimization program introduced by [62] to capture not only the sequence of operations on a module, but also the specific module that operation sequence is bound to. First we determine potential orderings of operations on each module based on the binding:

$$x_{o,u} \wedge x_{p,u} \geq y_{u,o,p} \quad \forall u \in U, (o,p) \in E_A \tag{6.5}$$

Note that the direction of this constraint's inequality: while we want $y_{u,o,p}$ to be 1 if and only if operation p occurs directly after operation o in module u and thus trace out a binding sequence through the MSAG for each module (similarly to how $x_{i,j}$ defined binding paths in [62]), the MSAG is not a multistage graph as in previous works, so such a relationship cannot be determined directly from the binding alone. Like [62], we introduce constraints based on KCL to enforce that relationship.

The first KCL constraint ensures each operation on a binding path through the MSAG indicated by $y_{u,o,p}$ can only have one predecessor (or else the path would not be a path):

$$\sum\nolimits_{n \in pred_A(o), u \in U} y_{u,n,o} = 1 \quad \forall o \in V \tag{6.6}$$

Another constraint does the same for successors:

$$\sum\nolimits_{p \in succ_A(o), u \in U} y_{u,o,p} = 1 \quad \forall o \in V \tag{6.7}$$

We also treat $START$ and END nodes in V_A as pseudo-operations that are bound to every module (i.e. $x_{START,u} = 1,\ x_{END,u} = 1\ \forall u \in U$), and modify their KCL constraints accordingly:

$$\sum\nolimits_{p \in succ_A(START)} y_{u,START,p} = 1 \quad \forall u \in U \tag{6.8}$$

$$\sum\nolimits_{n \in pred_A(END_u)} y_{u,n,END_u} = 1 \quad \forall u \in U \tag{6.9}$$

With these KCL constraints, $y_{u,o,p}$ will be 1 if and only if o and p are bound to u and o, p occur in sequence at u. Therefore, the switching activity due to the specific sequence of operations at each module is known and the corresponding total switching activity of each module can be calculated.

Next, we determine the interconnects that the operations at each module will switch:

$$z_{u,v} \wedge y_{u,o,p} \leq t_{u,v,o,p} \quad \forall u, v \in U, u \neq v, (o,p) \in E_A \tag{6.10}$$

This constraint captures two key aspects of interconnect switching activity: what interconnects will be switched (via the $z_{u,v}$ variable) and the operations at each module that switches them (via $y_{u,o,p}$). Note that we only consider **non-gated** interconnects, so switching activity at a module's input will also lead to switching of **all** of the module's outgoing interconnects. We leave further power optimization via wire gating to future work.

Finally, we only consider binding solutions that can meet timing given a particular physical placement solution. Similar to the previous chapter, we ensure all DFG data transfers have positive timing slack:

$$(x_{o,u} \wedge x_{p,v}) D_{u,v} \leq (s(p) - s(o))\lambda \quad \forall u, v \in U, u \neq v, (o,p) \in E \tag{6.11}$$

Our objective function is to minimize the total expected switched capacitance of both modules and the interconnects that join them:

$$\min_{t,x,y,z} \sum_{o,p \in V_A, u \in U} \alpha_{o,p} [C_u^M y_{u,o,p} + \sum_{v \in U, v \neq u} C_{u,v}^N t_{u,v,o,p}] \tag{6.12}$$

where $\alpha_{o,p}$ captures the expected switching activity due to change in output values between operations o and p and C_u^M, $C_{u,v}^N$ represent the switchable capacitances of modules and interconnects respectively, subject to all of the above constraints.

6.3.2 Power Weighted Module Placement

Once a particular binding solution/datapath architecture is found, we can use the switching activities at each wire as weights on each net when performing module-level global placement to guide the placer towards optimizing $C_{u,v}^N$ terms in (6.12).

For each wire that exists between modules u, v for a given binding, we can find the α values that make up the coefficient for the corresponding $C_{u,v}^N$ term in (6.12) and use that as a net wirelength weight value $w_{u,v}$

$$w_{u,v} := \beta \sum\nolimits_{o,p \in E_A} \alpha_{o,p} t_{u,v,o,p} \quad \forall u, v \in \{a, b | z_{a,b} = 1\} \tag{6.13}$$

where values of z, t are the optimal solution to the ILP presented in the previous section. A higher $w_{u,v}$ value indicates more switching activity on the wire between modules u, v, and therefore will effectively be treated as a longer wire to the global placer. An additional β parameter is added to control how aggressively the placer will optimize for interconnect power. We integrate this weight into the timing-aware

cost function (6.1) as an additional non-exponential weighting term in the overall placer cost function:

$$\sum_{u,v \in N} WL_{u,v}(\mathbf{x})(1 + w_{u,v} + \exp((WL_{u,v}(\mathbf{x}) - target_{u,v})/\gamma) \tag{6.14}$$

Since the cost function remains differentiable, it can be utilized in both simulated annealing based placers and electrostatics based global placers like RePlAce [11].

6.3.3 The Overall Co-Design Flow

Algorithm 6.1: calculate_MSAG

Input: $G(V, E, s)$, system traces

Output: $G_A(V_A, E_A, \alpha)$

1: $V_A = V \cup \{START, END\}$

2: $E_A = \emptyset$

3: **for** $o, p \in V_A, o \neq p$ **do**

4: **if** $s(o) < s(p)$ **then**

5: $E_A = E_A \cup \{(o, p)\}$

6: Calculate $\alpha_{o,p}$ from system traces

7: **end if**

8: **end for**

9: **return** G_A

Algorithm 6.3 describes our co-design flow integrating our placement aware ILP binding technique and binding-aware placement methodologies together. First, all net capacitances C^N (and correspondingly all net delays) are initialized to zero

Algorithm 6.2: place

Input: binding, α, G

Output: module placement

1: calculate $w_{u,v}$ weights from eq. (6.13)

2: calculate $target_{u,v}$ values for each net based on SDFG timing constraints, technology file

3: call 3D placer with $w_{u,v}$ weights and $target_{u,v}$ HPWL targets

4: **return** 3D IC module placement

(line 1) and the MSAG is calculated from system traces as described in Algorithm 6.1 and Section 6.1.1 (line 2). In the loop, the algorithm first reads and parses net lengths, capacitances, and delays from the most recent placement solution if it exists, otherwise the initial capacitance/delay values set in line 1 are used. The ILP binding method proposed in Section 6.3.1 is then run (line 8) and either a valid binding returned or the ILP is reported as infeasible. If any stop conditions have been met, then the current best binding and placement solution is returned. Otherwise, if the current binding results in the best achieved power thus far, it is saved (lines 9-13). Power and timing weighted placement is then performed (line 14, Algorithm 6.2) and the iteration count increased (line 15).

To run the overall co-design flow, a designer will need to first perform HLS scheduling and resource allocation steps so that a SDFG $G(V, E, s)$ and module allocation U are available. Additionally, a clock period λ that meets performance targets with the given schedule should be given. Good estimates of switching activity

Algorithm 6.3: Power-optimized co-design algorithm

Input: G, U, λ, system traces, C^M

Output: module placement, binding

1: Initialize C^N to 0

2: G_A = calculate_MSAG(G, system traces)

3: current_best_place = NONE

4: **while** TRUE **do**

5: **if** placement exists **then**

6: parse placement's net lengths, C^N, D

7: **end if**

8: binding = bind($C^M, C^N, G, G_A, U, \lambda, D$)

9: **if** iteration limit not reached, binding didn't change, ILP infeasible **then**

10: return current_best_place

11: **else if** placement solution power better than current_best_place **then**

12: update current_best_place

13: **end if**

14: placement = place(binding, α, G)

15: increment iteration count

16: **end while**

between operations in the DFG must be provided (e.g. from system traces) and the intrinsic power and capacitance of each allocated module should be characterized. The co-design flow will then iteratively find binding and placement solutions that minimize total switched capacitance and will return the best solution found. If the clock period is set too aggressively low, then constraint (6.11) may render the ILP infeasible. The system designer will then need to reevaluate timing constraints.

6.4 Results

To evaluate our power-optimized binding and placement co-design flow, we test our methods on functions extracted from the MediaBench [37] suite of multimedia benchmark suite. The DFG of each tested benchmark was extracted with SUIF and scheduled with a path-based scheduling algorithm [48]. We used the binding method introduced in [87] as the baseline comparison for our power-optimized binding method.

We used the timing-aware version of the electrostatics-based ePlace-3D global placer presented in the previous chapter as our baseline 3D module placer. The modified version of ePlace-3D does not perform standard cell placement, only force-directed and simulated annealing based module/macro placement. A γ value of 20000 was empirically found to give good results without causing numerical issues for our technology library. Similarly, setting β to 10 resulted in a good balance between wirelength and power, and the timing-aware HPWL weight was limited to not exceed 50 to prevent numerical overflow. We set pin locations of all module

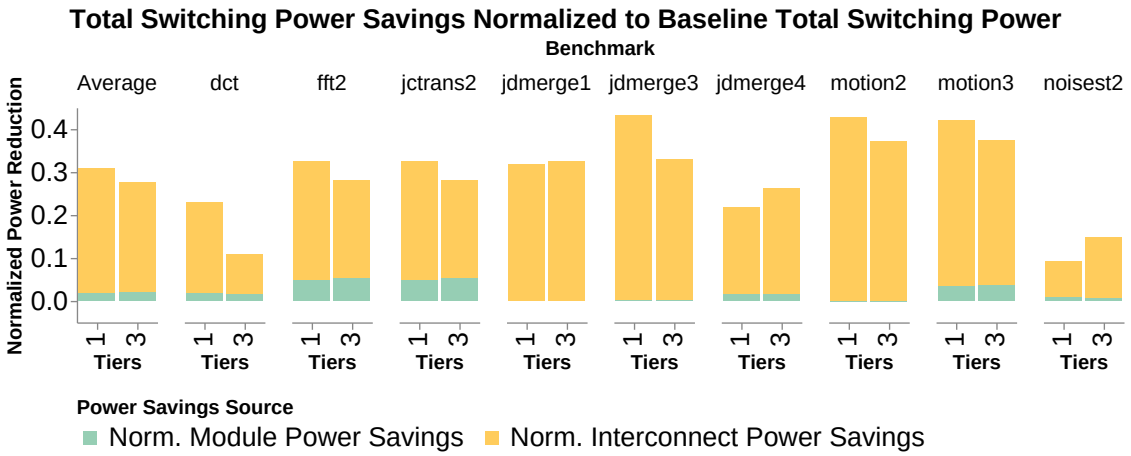


Figure 6.4: Power reductions for tested benchmarks, normalized to a typical binding and placement flow. Reductions are further broken down by source.

interconnects to the center of every module and increase the timing-based *target* wirelength values accordingly. Similarly, we use ePlace-3D's existing approach of modeling TSVs as having zero volume. For 3D floorplans, three tiers were allocated with a target density of 0.7. A relaxed clock period of 30ns was used as well since we consider low power applications where high performance is not as critical.

Gurobi was used to implement the ILP. Cadence Genus and Innovus were used to synthesize and perform power characterization of representative macro blocks built on top of Nangate45 [66]. OpenSTA and OpenROAD [3] were used to perform module-level timing characterization and die outline initialization. RosettaStone [24] was used to convert the DEF format used by OpenROAD to the bookshelf format expected by ePlace-3D.

Fig. 6.4 shows how much our method reduces overall dynamic power normalized to the conventional binding and timing-aware-only placement approach. In all test cases the vast majority of the reduction in overall power is due to reductions in

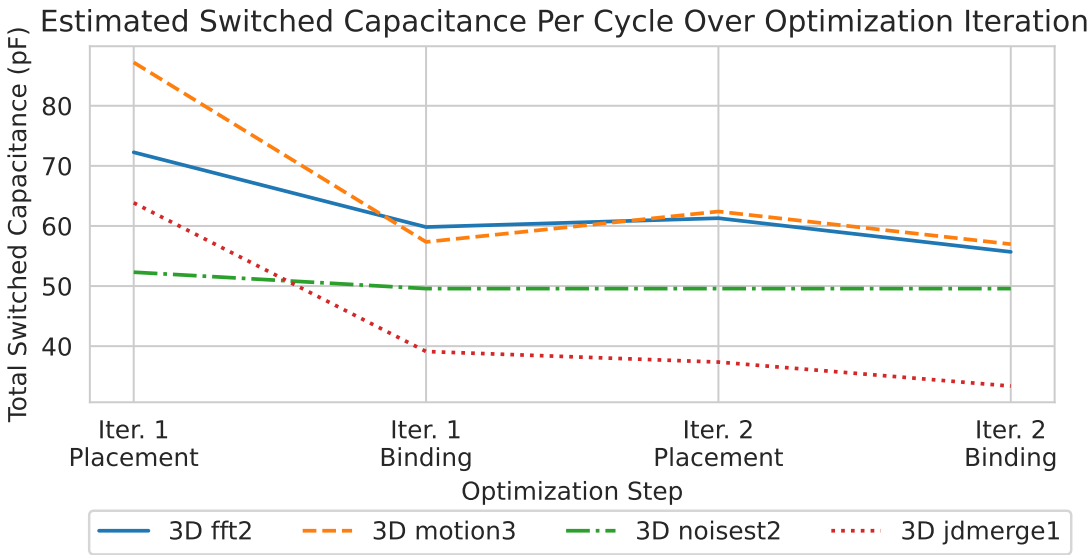


Figure 6.5: Total capacitance values as the optimization loop progresses for selected 3D benchmarks.

interconnect power, with minimal reduction contribution from changes in module switching. Most cases also showed lesser reduction in capacitance for 3D floorplans due to the relatively small capacitance value for TSVs used natively in ePlace-3D (30 fF).

We further show how the objective value changed as the co-design loop progresses for a few selected benchmarks in Fig. 6.5. Two values per iteration are reported: the first is calculated after placement, and the second is calculated after binding. Note that before the first iteration is run, the ILP binding method is run (not shown in our figures) with intermodule distances (and hence capacitances) set to zero in order to jump start placement.

Overall capacitance after first running placement is close to the baseline value since the architecture had to be decided with no placement information, and thus could not be optimized for interconnect power. Only after an initial placement

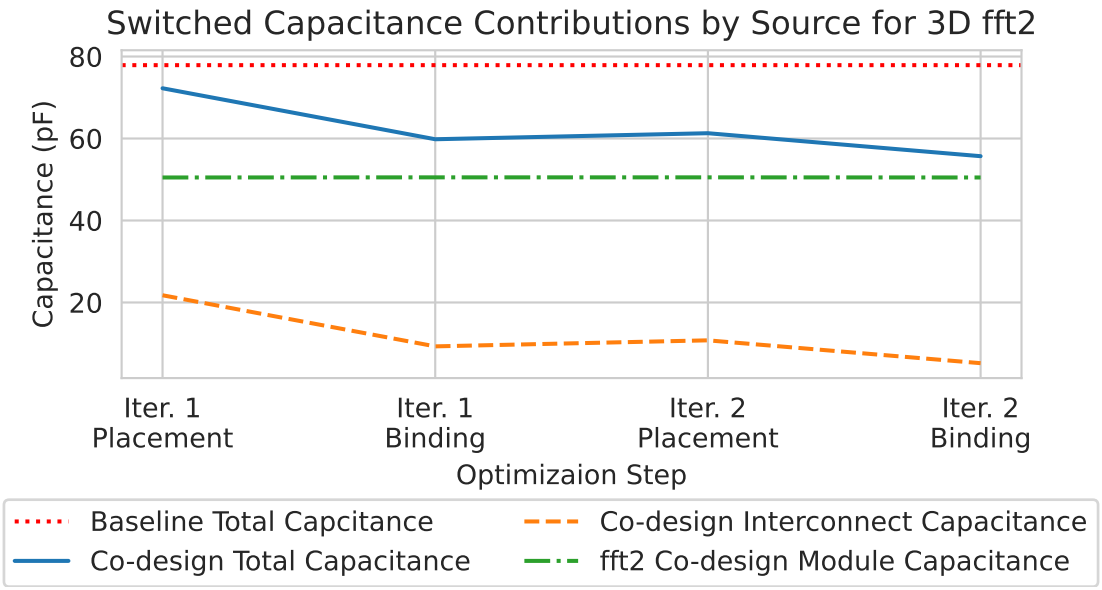


Figure 6.6: Optimization progress of 2D `motion2` as optimization progresses with further breakdown by sources of switching capacitance (module and interconnect).

is generated can our binding method make use of placement-derived capacitance values, subsequently significantly lowering the overall switched capacitance as shown in Fig. 6.5. In the second iteration, placement is run with adjusted interconnect weights. Another round of binding further lowers switched capacitance.

We can dig even further and examine how interconnect and module capacitance contribute to overall total capacitance during the co-design loop for the 2D `motion2` benchmark in Fig. 6.6. Interestingly, module capacitance does not change much. Coupled with the low reduction in power due to module switching shown in Fig. 6.4, we can conclude that the baseline binding method was already close to optimally minimizing module switching power, again highlighting the importance of interconnect-aware design.

6.5 Summary

In this chapter, we propose a 3D- and interconnect-aware power-optimized HLS binding and physical placement co-design flow. Our method minimizes switching activity both within modules and at module-module interconnects by performing physically-aware power-minimized binding and power-aware module placement together in the same loop. We apply our co-design method on MediaBench DFGs and find that our proposed approach was able to reduce switching power by 31% and 28% for 2D and 3D designs respectively and show that the majority of power savings is due to reductions in interconnect switching power.

Chapter 7: Architectural-Physical Co-Design for TSV Area Optimization

Since TSVs are usually placed during physical design, guiding architecture synthesis towards architectures that better utilize TSVs while making sure TSV usage and placement are within physical placement constraints is a challenge. Previous chapters outline similar co-design approaches, integrating HLS binding with global module placement for timing (chapter 5) and dynamic power (chapter 6), they do not consider TSV area overhead and assume TSVs take up zero logic area.

The co-design approach presented in this chapter uses information provided by physical placement to allocate and place TSVs in available whitespace while guiding architecture synthesis towards better operation schedules and bindings that more effectively utilize area allocated for TSVs through TSV sharing. By directly providing a physically-aware HLS tool TSV size and logic placement information, the number of TSVs can be significantly reduced while constraining other aspects of the design including timing correctness and schedule latency. The contributions of this chapter are as follows:

- An exact mathematical representation of the scheduling, binding, coarse-grained TSV placement and TSV sharing problem while incorporating physical timing constraints through integer linear programming (ILP).

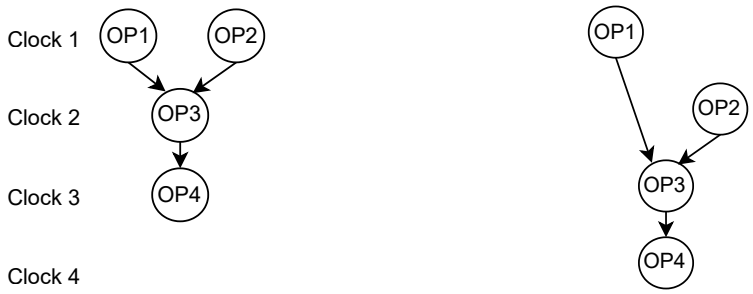
- Multiple optimization objectives based on the proposed constraints. Besides total TSV allocation, our formulation can also be used to optimize for different types of timing slack and schedule latency.

- As HLS updates the scheduling and binding, we show how our approach can be integrated into a architectural-physical co-design loop with 3D module placement that updates the placement for timing-optimal total wirelength between modules

- When tested on synthesized MediaBench benchmarks, our approach reduces the number of TSVs by 73% when compared to a timing-only co-design approach of the previous chapter. We also show that simplifying the model to only consider binding still results in an average of 41% TSV savings.

7.1 Placement-Informed Scheduling and Binding for TSVs

Placement information can be used to model wire lengths during high level synthesis, which can then be used to estimate wire delays (as shown in chapter 5) and interconnect power consumption (6). However, depending on the technology, TSVs needed to connect modules together in a 3D IC can take up significant on-die area so TSVs need to be placed such that they do not overlap with logic and other TSVs on the bottom die. Given a fixed module placement, the remaining white space can then be reserved for TSV placement.



(a) Scheduled DFG 1

(b) Scheduled DFG 2

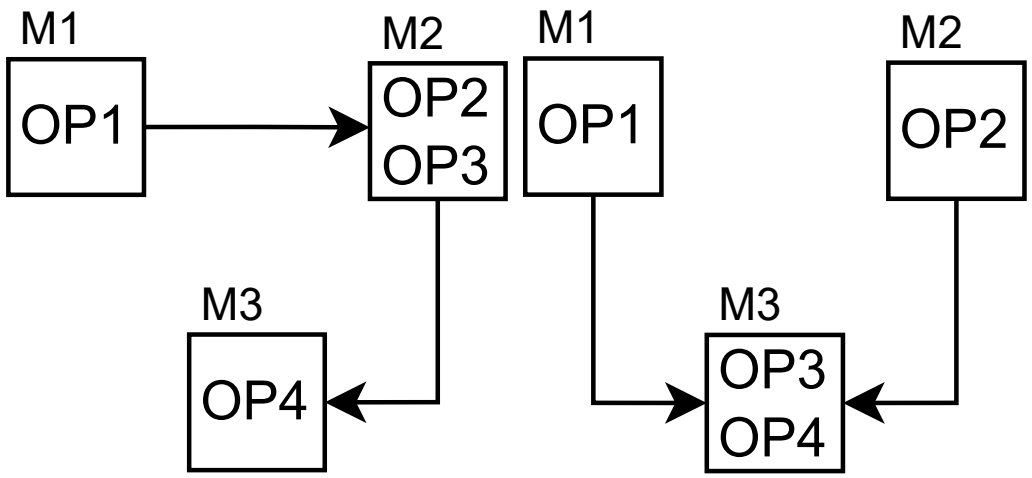
Figure 7.1: Example schedules for the same DFG

While this can be handled during placement and routing, we propose incorporating coarse-grained TSV placement into architectural decisions so that operation scheduling and binding can optimize module connectivity and data transfers to work within TSV placement constraints. This also enables TSVs to be shared by time-multiplexing data transfers when not enough die space is available for TSVs. To illustrate how we incorporate coarse grained TSV placement into architectural synthesis, we first present a simplified example.

7.1.1 An Illustrative Example

We will use the DFG shown in Fig. 7.1 to illustrate how our HLS physically-aware TSV area optimization approach works.

Suppose the schedule shown in Fig. 7.1a is used, where operations OP1 and OP2 are executed in parallel in the first clock and the last two operations are executed in separate subsequent clocks. Two example bindings are shown in Figure



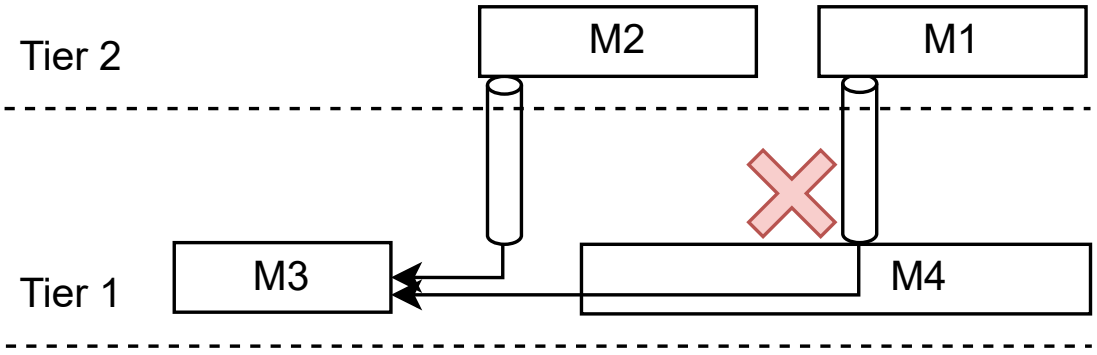
(a) Binding 1 (b) Binding 2

Figure 7.2: Example operation bindings

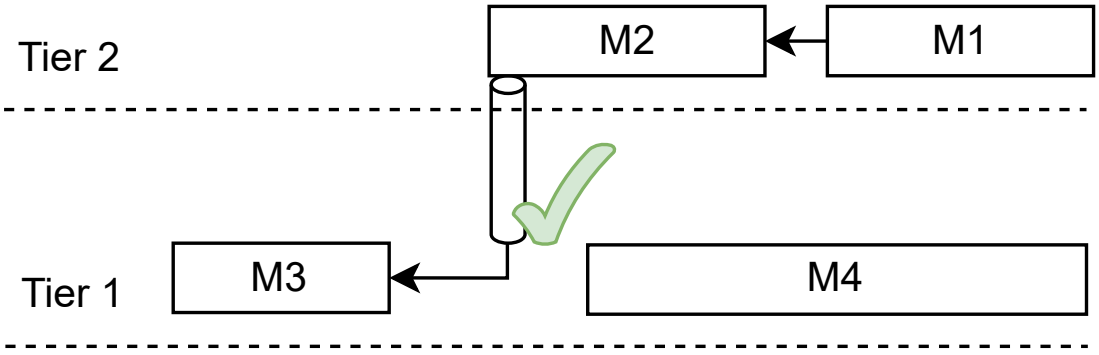
7.2, one where OP3 is bound to module 2 (Fig. 7.2a and one where it OP3 is bound to module 3 (Fig. 7.2b). Module interconnects for each binding are represented by arrows.

Assuming the modules are placed as shown in the 2D cross section in Fig. 7.3, where modules 1 and 2 are placed on the top die, module 3 is placed in the bottom die, and a fourth module placed underneath both M1 and M2, then the first binding requires only one set of TSVs to connect modules 2 and 3 (Fig. 7.3b) together while the second binding requires two sets to connect modules 1 and 2 to module 3 (Fig. 7.3a). However, the TSV connecting M2 to M3 actually cannot be legally placed due to a blockage by M4 (Fig. 7.3a)[1]. Thus, only the first binding is a valid one,

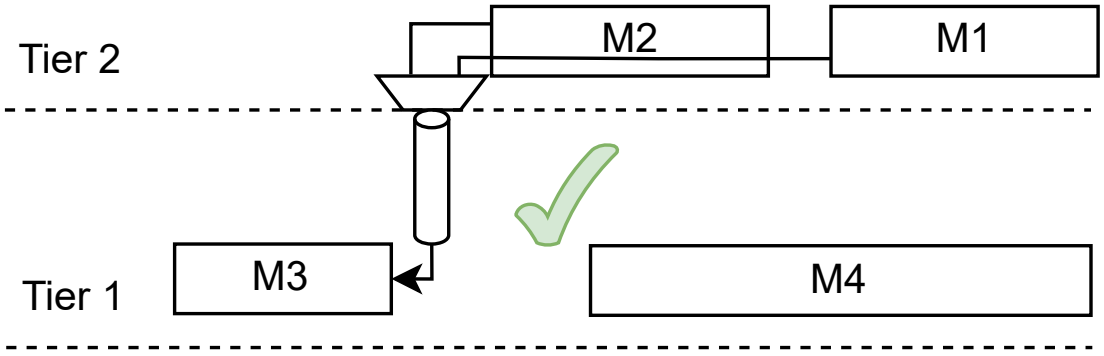
[1]The TSV could be placed to the left of M3. However, this would lead to greater wirelength and therefore higher delay on the M1-M3 net.



(a) Invalid TSV placement



(b) Valid TSV placement



(c) Valid placement with TSV sharing

Figure 7.3: Example module placement with different TSV placements.

with the corresponding TSV placed as shown in Fig. 7.3b. This illustrates how placement informed binding can lead to an overall design with not only fewer TSVs but also *placable* TSVs.

On the other hand, we can also choose the binding shown in Fig. 7.2b and adjust the schedule instead. To resolve the TSV blockage by M4 we can instead change the schedule to that shown in Fig. 7.1b such that OP1 and OP2 are scheduled on different clocks. Then the data transfers to module 3 are no longer concurrent (assuming data transfers take place within one clock cycle) and can therefore share the same TSV to facilitate data transfers by multiplexing the transfers as shown in Fig. 7.3c. The tradeoff for sharing the TSV is the additional clock cycle of latency schedule 1 has over schedule 2. This illustrates how informing scheduling and binding with placement information can result in better and valid TSV placements, especially in area-constrained designs.

7.2 An ILP Model for TSV-Aware Scheduling and Binding

This section will describe our mathematical approach to the physically aware HLS scheduling, binding, TSV placement, and TSV sharing problem. First, we will describe how we partition a 3D floorplan into TSV *parcels*, before describing the HLS formulation. Then we describe how timing constraints from a solution to the model can be extracted and given to a timing-aware global placer. We conclude the section describing how placement and binding can be integrated together into the same optimization loop.

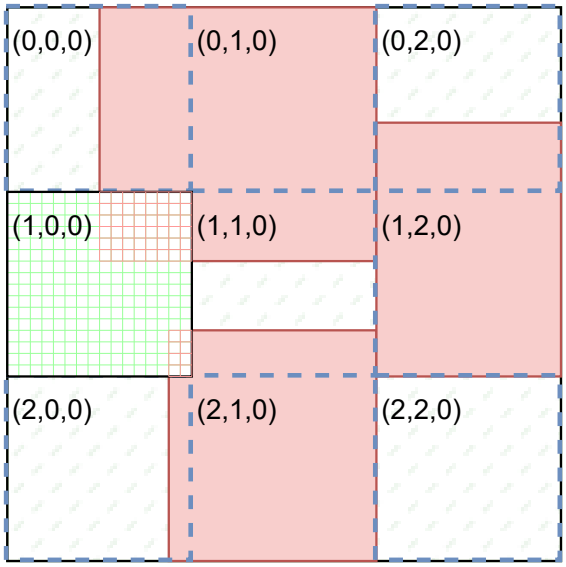


Figure 7.4: TSV parceling example for $n_x = n_y = 3$ on a single tier. Each parcel contains space for 16×16 TSVs. Modules are highlighted in red and unused areas are shown in hatched green. Assuming each module interconnect is 32 bits wide, parcel $(1, 0, 0)$ has enough open space for a parcel capacity of $C_{1,0,0} = \lfloor (256 - 56)/32 \rfloor = 6$ TSV bundles after accounting for overlaps with placed modules.

7.2.1 TSV Parcels

Since we want to place TSVs at the architectural level, we uniformly divide the floorplan into parcels that we allocate TSVs to. An example is shown in Fig. 7.4. We first take a 3D IC floorplan and partition the placement region into a uniform $n_x \times n_y \times n_z$ grid Q, where n_x, n_y are user-determined[2] and $n_z + 1$ is the number of 3D IC tiers[3].

[2]Of note is that our formulation can be devolved into a 3D tier partitioning problem by setting $n_x = n_y = 1$, i.e. each tier is a single TSV parcel.

[3]For face-to-back 3D ICs, TSVs do not take logic area on the top die, so parcels are not needed on the top die.

Instead of considering individual TSVs, we group TSVs together into *bundles*, each of which represent a single module to module connection. Each parcel $(x, y, z) \in Q$ then has an associated *TSV bundle* capacity $C_{x,y,z}$ which is the number of module-module connections that can be formed by TSVs in the parcel. This will depend on the size of the parcel and the bit width of each module-module connection, along with minimum TSV pitch constraints[4]. Additionally, given a 3D module placement, we scale the TSV bundle capacity of each parcel down based on how much of the parcel has been occluded by placed modules, as shown in Fig. 7.4.

7.2.2 Constraints

To keep our formulation easy to follow, Table 7.1 lists all the variables and constants used in this section. First we start with basic scheduling constraints. Every operation must be scheduled

$$\sum_{t \in T} a_{t,o} = 1 \quad \forall o \in V \tag{7.1}$$

and operations must be scheduled such that data dependencies in the DFG G are obeyed

$$\begin{gathered} a_{t_o,o} + \sum_{t_p \in [\text{ASAP}(p), t_o]} a_{t_p,p} \leq 1 \\ \forall (o,p) \in E, \ \forall t_o \in \text{valid range of } o \end{gathered} \tag{7.2}$$

[4]In our formulation we assume all module-module interconnects have the same bit width. However, our formulation can be adapted to handle different bit widths by using a smaller TSV bundles and adding additional connectivity between modules with higher connectivity requirements in the following ILP.

Table 7.1: ILP Variables and Constants

Notation	Definition
	Constants
$G(V, E)$	Input DFG.
$C_{x,y,z}$	TSV capacity for 3D floorplan parcel located at (x, y, z) with module-induced blockages.
λ	Clock period.
P_u	Intrinsic propagation delay of module u. Included in $P_{u,v}$
$P_{u,v}$	Estimated delay between modules u and v calculated from the 3D Manhattan distance between u and v.
Q	a $n_x \times n_y \times n_z$ grid over the 3D placement region for module and TSV locations.
T	Schedule maximum delay constraint.
U	Set of allocated modules.
V	Set of operations in the DFG G.
	Variables
$a_{t,o}$	Binary variable capturing scheduling decisions. 1 if operation o is scheduled for clock t, 0 otherwise.
$b_{o,u}$	Binary variable describing operation-module binding decisions. 1 if operation o is bound to module u, 0 otherwise.
$c_{u,v}$	Binary variable describing module connectivity. 1 if module u transfers data to module v, i.e. if a wire connects u to v.
$d^{\text{up}}_{x,y,z}$	Nonnegative integer variable tracking number of TSVs placed at TSV parcel x, y in tier z bringing data upward out of tier z.
$d^{\text{down}}_{x,y,z}$	Nonnegative integer variable tracking number of TSVs placed at TSV parcel x, y in tier z bringing data downward into tier z.
$d^{u,v}_{x,y,z}$	Binary variable, 1 if the net connecting module u to v (where u is the source module) has a TSV placed at TSV parcel x, y in tier z.
$e_{u,v,o,p}$	Binary variable that describes module connectivity and the associated data transfers on them.
$f_{u,v,t}$	Binary variable, 1 if a data transfer from module u to module v is live/ongoing on the wire connecting u to v at clock t.
$g_{o,p,t1,t2}$	Binary variable describing every pair of scheduling possibilities. 1 if operation p depends on operation o and (o, p) is scheduled for $(t1, t2)$.
$h_{u,v,o,t}$	Binary variable describing data transfers and the source operation that drives them. 1 if operation o is bound to module u, a wire exists connecting modules u to v, and the data transfer of the output of operation o to module v is physically ongoing at clock t.
$lastclock$	Nonnegative integer variable that encodes the last clock of the schedule.
$s_{u,v}$	Nonnegative real-valued variable that captures the negative slack on the net between modules u and v, if it exists.
s	Nonnegative real-valued variable that captures the worst slack among all u, v nets.

where ASAP(p) is the ASAP schedule [46] for operation p, and the *valid range* of an operation is the set of clock cycles that an operation could be scheduled for without a data dependency violation, i.e. $[\text{ASAP}(o), \text{ALAP}(o)]$.

Additionally, only one operation can be executed in any particular clock:

$$\sum_{o \in V} a_{t,o} \wedge b_{o,u} \leq 1 \quad \forall u \in U \ \forall t \in T \tag{7.3}$$

Note that while the logical AND operation in constraint 7.3 is not linear, since all variables are binary, the constraint can be rewritten as an equivalent set of integer linear constraints:

$$\left.\begin{aligned} a_{t,o} + b_{o,u} - 1 &\leq c_{t,o,u} \\ c_{t,o,u} &\leq a_{t,o} \\ c_{t,o,u} &\leq b_{o,u} \end{aligned}\right\} \forall u \in U,\ \forall t \in T,\ \forall o \in V$$

$$\sum_{o \in V} c_{t,o,u} \leq 1 \quad \forall u \in U,\ \forall t \in T$$

Every operation must be bound to exactly one module:

$$\sum_{u} b_{o,u} = 1 \quad \forall o \in V \tag{7.4}$$

The following constraint encodes connectivity and data transfer information between modules based on the binding and the dataflow graph into the variable e:

$$b_{o,u} \wedge b_{p,v} = e_{u,v,o,p} \quad \forall (o,p) \in E,\ u, v \in U,\ u \neq v \tag{7.5}$$

Similarly to eq. 7.3, this can be rewritten as a set of linear constraints.

From e we can derive the necessary connectivity needed to facilitate data transfers between modules and store it in variable c:

$$e_{u,v,o,p} \leq c_{u,v} \quad \forall (o,p) \in E,\ u, v \in U,\ u \neq v \tag{7.6}$$

Since TSVs have high parasitics and therefore have higher delay, we allow data transfers to take place over multiple clock cycles, using a latch to hold output values on wires as necessary. Since preliminary physical placement information for each module is known, we can derive an estimate on the physical wire/TSV delay between every pair of modules $P_{u,v}$. When combined with operation binding information e, the clock cycle each source operation is scheduled for $a_{t_o,o}$, and the target clock period λ, we can derive which clocks t that a data bus connecting modules u and v is busy with a data transfer caused by o:

$$\begin{aligned} e_{u,v,o,p} \wedge a_{t_o,o} \leq h_{u,v,o,t} \; & \forall (o,p) \in E, \; u,v \in U, u \neq v, \\ & \forall t_o \in \text{valid range of } o, \\ & \forall t \in \{t_o ... t_o + \lceil P_{u,v}/\lambda \rceil - 1\} \end{aligned} \tag{7.7}$$

We constrain h such that only one operation can cause a wire to be busy at a time, effectively a resource binding constraint on the interconnect sending data from module u to module v that ensures the wire is only supporting one data transfer at a time:

$$\sum_{o \text{ bindable to } u} h_{u,v,o,t} \leq 1 \quad \forall u,v \in U, u \neq v, \forall t \in T \tag{7.8}$$

Now that interconnect resource contention has been constrained away, we can then encode which clock cycles each wire is busy for into f:

$$h_{u,v,o,t} \leq f_{u,v,t} \quad \forall u,v \in U, u \neq v, \forall t \in T \tag{7.9}$$

To ensure no negative slack on operation-operation data transfers between different modules, we additionally capture every *scheduled* data edge in the DFG into g such that the timing on each data edge can be tracked:

$$\begin{aligned} a_{t_o,o} \wedge a_{t_p,p} = g_{o,p,t1,t2} \; \forall t_o \in \text{valid range of o}, \\ \forall t_p \in \text{valid range of p}, \\ t_o < t_p, \forall (o,p) \in E, \end{aligned} \tag{7.10}$$

We can then create a constraint that ensures each (potentially) multi-cycle data transfer does not take less time than physically realizable and cause negative slack:

$$\begin{aligned} e_{u,v,o,p} P_{u,v} - (t_p - t_o) g_{o,p,t_o,t_p} \lambda - (1 - g_{o,p,t_o,t_p})(\text{Inf}) \leq 0 \\ \forall u, v \in U, u \neq v, \; \forall (o,p) \in E, \\ \forall t_o \in \text{valid range of } o, \; \forall t_p \in \text{valid range of } p, \; t_o < t_p \end{aligned} \tag{7.11}$$

where Inf is a large number greater than the maximum allowable schedule latency, λ is the clock period, and $P_{u,v}$ are placement-derived estimates of the total delay between every pair of modules.

From here we can start mapping each u, v wire to physical TSVs. Recall from Sec. 7.2.1 that we partition each tier of the 3D floorplan into a uniform $n_x \times n_y$ grid of parcels. Nominally, every parcel is allocated TSV bundles based on the number of TSVs that can fit in a parcel divided by the bit width of a module interconnect. Since module placement information is given to HLS, we proportionally reduce the TSV bundle capacity of each parcel by how much each parcel is occluded by placed modules. All of this is encoded in $C_{x,y,z}$.

First, we note that every pair of modules placed on different 3D IC tiers can be used to define the corners of a 3D bounding box. In parcel coordinates, we add a constraint that specifies that a TSV bundle must be allocated somewhere on each layer inside that bounding box if u and v are on separate layers and are connected together:

$$\begin{aligned} \sum_{(x,y)\in[x_u,x_v]\times[y_u,y_v]} d^{u,v}_{x,y,z} &= c_{u,v} \\ \forall u, v \in U, u \neq v, & \\ \forall z \in \begin{cases} [z_u, z_v - 1] & \text{if } v \text{ above } u \\ [z_v, z_u - 1] & \text{if } u \text{ above } v \end{cases} & \end{aligned} \tag{7.12}$$

where $(x_u, y_u, z_u) \in Q$, $(x_v, y_v, z_v) \in Q$ are the locations of modules u and v on the 3D TSV parcel placement grid Q respectively, and $[x_u, x_v] \times [y_u, y_v]$ denotes the set of (x, y) parcel coordinates inside the bounding box created by modules u and v.

Additionally, we want to ensure that TSVs are placed such that the 3D Manhattan distance of the wire connecting every pair of modules u, v remains minimal so that HPWL delay estimates remain accurate. Since there are many possible TSV placements that preserves this minimal wirelength property, we constrain the TSV placement at each tier such that each TSV can only be placed such that each TSV is within both the 3D bounding box defined by the location of each module u and v as well as the bounding box defined by the (up to) two closest TSVs on the same

u, v net:

$$d^{u,v}_{x_a,x_b,i} + \sum_{(x,y)\in[x_u,x_v]\times[y_u,y_v]\setminus[x_a,x_v]\times[x_b,y_v]} d^{u,v}_{x,y,i+1} \leq c_{u,v}$$

$$\forall u, v \in U, u \neq v,$$

$$\forall (x_a, x_b) \in [x_u, x_v] \times [y_u, y_v], \tag{7.13}$$

$$\forall i \in \begin{cases} [z_u, z_v - 2] & \text{if } v \text{ above } u \\ [z_v, z_u - 2] & \text{if } u \text{ above } v \end{cases}$$

where $\times$ is the Cartesian product and $[x_u, x_v] \times [y_u, y_v]$ is inside the 2D bounding box defined by the x,y coordinates of modules u and v. Note that the structure of this constraint is similar to that of constraint (7.2), except instead of data dependencies in the DFG, we constrain each TSV such that the 3D wirelength between u to v remains minimal.

Based on the direction each placed $d^{u,v}_{x,y,z}$ TSV bundle is moving data in, we can then determine the number of TSV bundles that needs to be allocated in each parcel of the floorplan. Additionally, we also time-multiplex TSVs and share them between different data transfers on different modules by setting the TSV allocation for each parcel to the maximum number of simultaneous data transfers at each parcel in any clock, similar to the approaches used in [70, 36]. There are two sets of constraints, one for TSVs that move data upwards through the 3D stack

$$\sum_{u,v\in U, \text{v above u}, u\neq v} d^{u,v}_{x,y,z} \wedge f_{u,v,t} \leq d^{\text{up}}_{x,y,z} \tag{7.14}$$

$$\forall x, y, z \in Q, \forall t \in T$$

and one for TSVs that move data downwards

$$\sum_{u,v \in U, \text{u above v}, u \neq v} d^{u,v}_{x,y,z} \wedge f_{u,v,t} \leq d^{\text{down}}_{x,y,z} \quad \forall x, y, z \in Q, \forall t \in T \tag{7.15}$$

Finally, we enforce the TSV capacity of each parcel based on the number of upwards and downwards TSVs in each parcel

$$d^{\text{up}}_{x,y,z} + d^{\text{down}}_{x,y,z} \leq C_{x,y,z} \quad \forall x, y, z \in Q \tag{7.16}$$

7.2.3 Objective Functions

Since an ILP representation is flexible, there are many possible objective functions to optimize for. In this section, we describe four objective functions: schedule length, total TSV count, total negative slack (TNS), and worst negative slack (WNS). In all cases, the optimal solution to the ILP describes the operation schedule (a), the operation binding (b), the required connectivity needed between modules (c), as well as the coarse-grained allocation of all TSVs (d^{up}, d^{down}) which, combined with module-specific TSV placements ($d^{u,v}_{*}$), the schedule and binding, also gives TSV sharing information.

7.2.3.1 Schedule length

Schedule delay is a common scheduling optimization objective and can be expressed in our formulation. First, another constraint is added to constrain a new integer variable *lastclock* to be at least as large as the final clock:

$$ta_{t,o} \leq lastclock \quad \forall t \in T, \ \forall o \in V \tag{7.17}$$

The objective function is simply to minimize *lastclock*:

$$\min_{a,b,c,d,e,f,g,lastclock} lastclock \tag{7.18}$$

subject to all constraints in Section 7.2.2 and constraint 7.17.

7.2.3.2 TSV Minimization

Besides taking up logic area on the bottom die, TSVs also add complexity to post-fabrication packaging and testing[75]. Increasing the number of TSVs increases the amount of testing that needs to be done after integration and can reduce yield, both of which can increase cost. Minimizing the number of TSVs is therefore a common optimization objective of many 3D placers.

Since $d^{\text{up}}_{x,y,z}$ and $d^{\text{down}}_{x,y,z}$ together directly describe the number of allocated TSV bundles in each (x, y, z) floorplan parcel, minimizing the total number of TSVs allocated can be easily captured through this program:

$$\min_{a,b,c,d,e,f,g} \sum_{x,y,z \in \text{placement area}} d^{\text{up}}_{x,y,z} + d^{\text{down}}_{x,y,z} \tag{7.19}$$

subject to all constraints in Section 7.2.2.

Since the ILP can simultaneously optimize both binding (which affects which module-module interconnects are needed, and therefore how many TSVs are needed) and scheduling (which increases both module resource sharing and TSV resource sharing), it has the potential to reduce TSV count significantly more than scheduling and binding alone.

7.2.3.3 TNS and WNS

Instead of constraining the ILP to only consider schedules and operation mappings with no negative slack, we can instead consider schedules and bindings that minimize WNS or TNS. For TNS, we define an additional nonnegative real-valued slack variable $s_{u,v}$ that captures the negative slack on each u, v net. Constraint 7.11 is then replaced with

$$e_{u,v,o,p}P_{u,v} - (t_p - t_o)g_{o,p,t_o,t_p}\lambda - (1 - g_{o,p,t_o,t_p})(\text{Inf}) \leq s_{u,v}$$
$$\forall u, v \in U, u \neq v,\ \forall (o,p) \in E, \tag{7.20}$$
$$\forall t_o \in \text{valid range of } o,\ \forall t_p \in \text{valid range of } p,\ t_o < t_p$$

Minimizing TNS is then equivalent to

$$\min_{a,b,c,d,e,f,g,s} \sum_{u,v \in U} s_{u,v} \tag{7.21}$$

Subject to constraints (7.1) to (7.10), (7.12) to (7.16), and (7.20).

WNS is defined in a similar way. Instead of tracking the negative slack on every net ($s_{u,v}$), only the worst negative slack across all nets needs to be defined and captured, so only one s variable is needed. We replace constraint (7.20) with

$$e_{u,v,o,p}P_{u,v} - (t_p - t_o)g_{o,p,t_o,t_p}\lambda - (1 - g_{o,p,t_o,t_p})(\text{Inf}) \leq s$$
$$\forall u, v \in U, u \neq v,\ \forall (o,p) \in E, \tag{7.22}$$
$$\forall t_o \in \text{valid range of } o,\ \forall t_p \in \text{valid range of } p,\ t_o < t_p$$

The objective function is to just directly minimize the worst negative slack s:

$$\min_{a,b,c,d,e,f,g,s} s \tag{7.23}$$

Subject to constraints (7.1) to (7.10), (7.12) to (7.16), and (7.22).

7.2.4 Timing Aware Physical Design

Given the optimal objective values to the ILP above, we can then derive timing constraints on each u, v module wire based on when operations that transfer data between the two modules are scheduled. First, we define $DT_{u,v}$ as the set of edges $(o, p) \in E$ of the DFG where operation o is bound to module u and operation p is bound to module v, i.e. the set of data transfers between modules v and v. More formally,

$$DT_{u,v} := \{(o, p) \in E | e_{u,v,o,p} = 1\} \tag{7.24}$$

The wire delay timing constraint $D_{u,v}$ between modules u and v is then just the minimum schedule delay of the data transfers between u and v

$$D_{u,v} = \min(\{\lambda(t_{p_i} - t_{o_i}) | (o_i, p_i) \in DT_{u,v}, a_{t_{o_i},o} = 1, a_{t_{p_i},p} = 1\}) - P_u \tag{7.25}$$

with the intrinsic execution delay of module u, P_u, removed. This timing constraint can then be given to timing aware global placement tools like the dynamic weighting approach proposed in chapter 5, where each module net is weighted by an exponential function such that net weights increase when a wire's delay starts to approach timing constraint.

7.2.5 Overall Co-Design Loop

Algorithm 7.1 describes the overall co-design optimization loop that connects our ILP-based HLS optimization back to module placement. First, we initialize delays P and parcel capacities with null values so that we can perform an initial

Algorithm 7.1: Optimization loop

Input: G, Q, λ, T, U

Ensure: schedule, binding, module placement,

TSV placement, TSV sharing

Initialize P to 0

Initialize C to Inf

while TRUE **do**

  if modules have been placed **then**

    get module placement, calculate delays P, module blockages C

  end if

  schedule, binding, TSV placement = ILP$(G, C, P, \lambda, Q, T, U)$

  if iteration limit reached, binding didn't change, ILP infeasible **then**

    return current_best

  else if placement solution latency better than current_best_place **then**

    update current_best

  end if

  perform timing-aware placement

  increment iteration count

end while

architecture synthesis (lines 1,2). Then we check if placement has already been performed (line 4) and if so, read out the placement solution and calculate estimated wire delays between each module $P_{u,v}$ as well as where modules overlap TSV placement parcels $C_{x,y,z}$ (line 5). Inside the loop, we perform placement-informed scheduling, binding, TSV placement, and TSV sharing using the ILP described in sections 7.2 and returns a valid schedule, binding, and TSV placement (line 7). We then check if any stop conditions have been met. If so, the best (measured in terms of the objective function, either total number of TSVs or schedule latency) architecture is returned. Otherwise we perform global placement (line 13) and increment the iteration count (line 14).

A system designer would need to specify the functionality being synthesized via a DFG (derived from a HLS compilation tool) as well as a module allocation that will implement the functionality. Modules should have been characterized in terms of their delay as well as 2D die outline. A target clock period as well as a maximum schedule latency is also required. The output will be a datapath architecture and schedule that implements the DFG as well as a timing-optimized module placement and coarse-grained TSV placement. When combined with schedule and binding information, TSV sharing parameters are also given. The placement can then be passed along to downstream physical design tools for refinement and final GDSII generation.

7.3 Results

We evaluate our approach on dataflows extracted from C functions in the MediaBench suite [37] by SUIF and compared our approach against a baseline approach consisting of a schedule generated by a path-based scheduler [48] and a binding generated by the tsv-agnostic co-design binding method proposed in chapter 5.

We used a timing-aware version of ePlace-3D [43] presented in chapter 5 as our 3D global placer for our co-design experiments. Pin locations for all modules were set to be the center of each module for wire length calculations and TSV capacity calculations. A square 3D floorplan with three tiers was used at a target density of 0.7 to allow for sufficient whitespace for TSV insertion. We use 5μm wide TSVs with a keep-out zone of 15 μm based on the reported TSV parameters in [57]. 32 bit wide interconnects were used to connect modules together, so the overall area of a TSV bundle is 7200μm^2. A 3-by-3 TSV parcel grid was used ($n_x = n_y = 3$) since it was empirically found to provide a good trade-off between placement resolution and solver performance. In the optimization loop, an initial aggressive clock period of 2ns was first used with the binding approach proposed in chapter 5 in WNS optimization mode in order to determine the fastest clock that could be used before the rest of the optimization was run. To keep TSV numbers comparable between the baseline approach and our improved approach, the schedule length limit T was kept constant for all runs on the same benchmark.

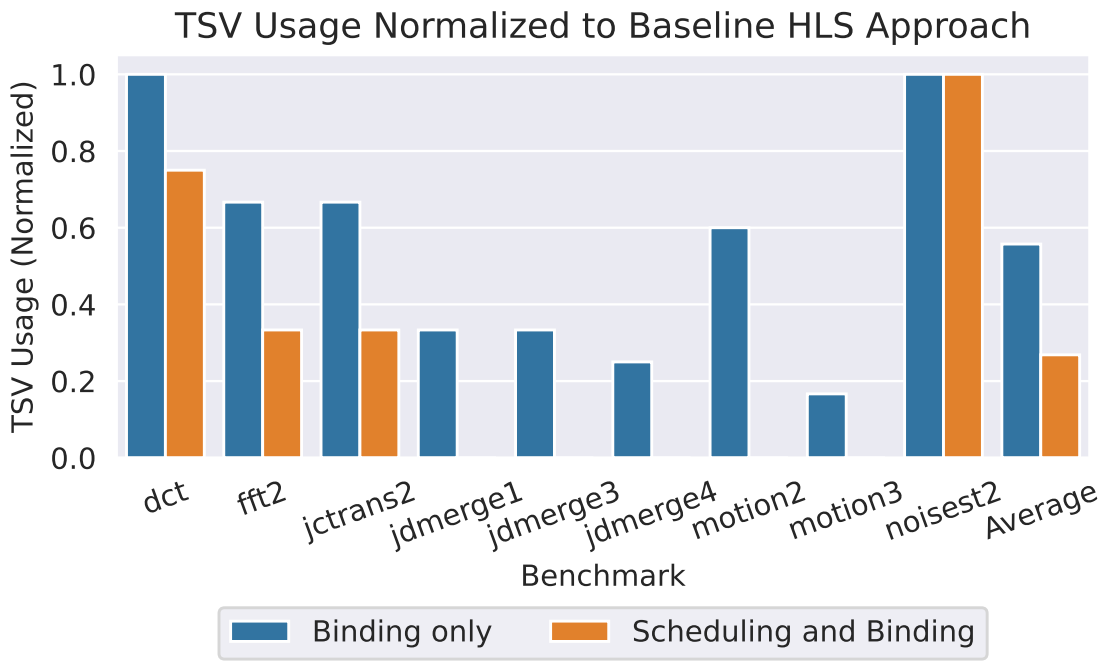


Figure 7.5: TSV savings over a baseline scheduling and binding approach for selected benchmarks.

Gurobi [18] was used to model and solve the ILP. Module design was performed with Cadence Genus and Innovus using standard cells built on top of Nangate45 [66]. OpenSTA [1] was used to generate timing characteristics and OpenROAD [3] was used to perform floorplan initialization. RosettaStone [24] was used to translate standard DEF/LEF physical design formats generated by OpenROAD into the academic Bookshelf format used in ePlace-3D.

Figure 7.5 shows the overall reduction in TSV count when compared to the non-TSV aware architectural-physical co-design approach proposed in chapter 5, which only focused on timing. Since each benchmark design has different DFGs and resource allocations, we normalize the total TSV count by the amount used by the baseline co-design approach. In almost benchmarks, the flexibility of controlling

TSV utilization through architectural scheduling and binding decisions resulted in a significant reduction of allocated TSVs, on average by 73%, and sometimes reducing TSV counts to zero. In benchmarks that resulted in zero allocated TSVs, this indicates that the ILP was able to find an operation binding that only used a single layer of the 3D floorplan, and therefore no TSVs are needed to implement the design (i.e. the design could be implemented with a monolithic 2D IC). Additionally, all tested benchmarks were able to find architectures that respected TSV capacity constraints within each TSV parcel.

We also examined how limiting our approach to only consider binding would affect TSV usage. We forced our optimization to use the same schedule as the baseline co-design approach (and therefore run in binding-only mode) by fixing schedule variables $a_{t,o}$ to the correspond to the same schedule as the baseline. When only considering binding, TSV usage was only reduced by 40%, less than the 77% reduction when simultaneously optimizing binding and scheduling. Since the operation schedule determines when data transfers occur, and therefore when TSVs are actively carrying signals between modules, constraining/fixing the schedule decreases opportunities for TSV sharing between modules, increasing TSV usage.

7.4 Conclusion

We propose a new integrated co-design approach that enables HLS scheduling and binding to consider TSV placement constraints and utilization optimization during datapath synthesis. Our ILP-based approach is able to optimally find a

schedule and module binding that satisfies data transfer timing constraints as well as TSV placement constraints by sharing TSVs and tailoring the schedule and binding for a better overall architecture for a given module placement. Multiple optimization objectives, including negative slack and total TSV usage, are presented. We applied our optimization program to extracted Mediabench dataflows and was able to reduce TSV usage on average by 77% while meeting placement-imposed TSV capacity and blockage constraints, timing constraints on nets, as well as schedule latency constraints.

Chapter 8: Conclusions and Future Research Directions

In this dissertation, we explored how optimizations enabled by a system level perspective can result in significant design quality improvements. Chapter 2 derived a RU resource sharing approach that significantly minimized the overhead of SFLL logic while increasing system-level locking impact. Chapter 3 expanded upon this by relaxing the requirement that the designer have access to comprehensive system traces and treating restore units as an operation in its own right, giving designers the flexibility to tradeoff area overhead of RUs with latency. Chapter 4 further extends this by realizing that locked FSCs do no useful work when the input is a PIP, and we demonstrate how the RU can be repurposed as a LUT-based clock gating mechanism to save dynamic power while maintaining the security primitives of SFLL, with additional gating opportunities enabled by examining data dependencies between operations and hardware modules.

Chapter 5 examined how considering physical placement of modules during HLS binding can lead to better estimates of wire delays between allocated modules, and therefore guide placement towards architectures that minimize timing violations. Chapter 6 considers how the long inter-module wires needed in 3D ICs contribute significantly to dynamic power consumption, and proposes a low dynamic

power binding optimization. Chapter 7 extends 5 to consider the logic area taken up by TSVs and the placement constraints imposed by placed modules in order to arrive at schedules and operation bindings that respect timing constraints while minimizing TSV usage.

8.1 Future Work

The hardware security and 3D IC work presented in this dissertation can be expanded in multiple directions. The rest of the chapter will discuss future research avenues that can be explored.

8.2 Low Overhead Scheduling and Binding for SFLL RU Sharing

Chapter 2 discussed our initial work on sharing SFLL restore units by using application traces to identify "compatible" modules for RU sharing. Chapter 3 discussed an alternative approach that removed the potentially burdensome requirement of access to exhaustive test traces in exchange for the potential for delay overhead by inserting static delays into an existing schedule. Our future work aims to further reduce the delay overheads from inserting static delays by treating SFLL restores as regular HLS operations with data dependencies such that they can be scheduled like any other operation.

Considering scheduling alongside binding for RU sharing essentially gives HLS another degree of control over resource contention avoidance when determining how RUs should be shared, which should further reduce the delay penalty introduced in Chapter 3. We will additionally combine our improved scheduling/binding sharing method with our LUT gating approach introduced in Chapter 4 to further reduce power consumption not only by RUs but also by locked modules.

8.3 Thermal Co-Optimizations for 3D ICs

One challenge with stacked 3D ICs is thermal issues with active silicon buried in the 3D stack, far away from any heat sink. Since the thermal resistance from buried active silicon to the heat sink is high, power management and careful placement is crucial to ensure the stack functions correctly.

A natural extension of the power optimization work in chapter 6 is to consider peak/average temperature optimization, since our 3D co-design approaches . Since the floorplan and power profile of each module is known, a simplified thermal model could be incorporated into the optimization. Additionally, metal TSVs have higher thermal conductance than bulk silicon, so TSV placement optimization be an additional factor in the optimization.

8.4 Secure System Level Design of 3D ICs

Heterogeneous integration (HI) enables designers to choose chiplets in order to compose a larger system-in-package (SiP). HI offers a multitude of benefits over monolithic 2D systems, including lowering cost through the use of smaller dies, using different process technologies for different chiplets for cost and custom circuits, and lower PCB footprint. However, the supply chain issues for conventional ICs extend to HI as well. Chiplets may be sourced from untrusted or unverified third parties, exposing the end SiP to vulnerabilities embedded in its chiplets such as trojans, which can cause performance degradation, side channel information leakage, and denial of service [72].

We propose exploring new system-level methods for chiplet-based HI for 3D ICs that mix and match chiplets from trusted and untrusted sources. The objective is to ensure that the resulting 3D system architecture remains secure even if certain components are sourced from untrusted suppliers.

Bibliography

[1] Parallax Static Timing Analyzer, November 2023. original-date: 2018-09-28T15:45:57Z.

[2] Nicolas Bohm Agostini, Serena Curzel, Vinay Amatya, Cheng Tan, Marco Minutoli, Vito Giovanni Castellana, Joseph Manzano, David Kaeli, and Antonino Tumeo. An MLIR-based Compiler Flow for System-Level Design and Hardware Acceleration. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '22, pages 1–9, New York, NY, USA, December 2022. Association for Computing Machinery.

[3] T Ajayi, D Blaauw, R Dreslinski, Mateus Fogaca, S Hashemi, A Ibrahim, A B Kahng, M Kim, J Li, Z Liang, U Mallappa, P Penzes, G Pradipta, S Reda, A Rovinski, K Samadi, S S Sapatnekar, L Saul, C Sechen, V Srinivas, W Swartz, D Sylvester, D Urquhart, L Wang, M Woo, and B Xu. OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain. *Proceedings of Government Microcircuit Applications and Critical Technology Conference*, 2019.

[4] Yousra Alkabani and Farinaz Koushanfar. Active hardware metering for intellectual property protection and security. In *USENIX security symposium*, 2007.

[5] Kimia Zamiri Azar, Hadi Mardani Kamali, Houman Homayoun, and Avesta Sasan. SMT Attack: Next Generation Attack on Obfuscated Circuits with Capabilities and Performance Beyond the SAT Attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 97–122, 2019.

[6] K. Banerjee, S.J. Souri, P. Kapur, and K.C. Saraswat. 3-D ICs: a novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration. *Proceedings of the IEEE*, 89(5):602–633, May 2001. Conference Name: Proceedings of the IEEE.

[7] Shivam Bhasin and Francesco Regazzoni. A survey on hardware trojan detection techniques. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2021–2024, May 2015. ISSN: 2158-1525.

[8] Abhishek Chakraborty, Nithyashankari Gummidipoondi Jayasankaran, Yuntao Liu, Jeyavijayan Rajendran, Ozgur Sinanoglu, Ankur Srivastava, Yang Xie, Muhammad Yasin, and Michael Zuzak. Keynote: A disquisition on logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[9] Prabuddha Chakraborty, Jonathan Cruz, and Swarup Bhunia. Sail: Machine learning guided structural analysis attack on hardware obfuscation. In *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 56–61. IEEE, 2018.

[10] Yibo Chen, Guangyu Sun, Qiaosha Zou, and Yuan Xie. 3DHLS: Incorporating high-level synthesis in physical planning of three-dimensional (3D) ICs. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1185–1190, March 2012. ISSN: 1558-1101.

[11] Chung-Kuan Cheng, Andrew B. Kahng, Ilgweon Kang, and Lutong Wang. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(9):1717–1730, September 2019. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[12] Armin Darjani, Nima Kavand, Shubham Rai, Mark Wijtvliet, and Akash Kumar. ENTANGLE: An Enhanced Logic-locking Technique for Thwarting SAT and Structural Attacks. In *Proceedings of the Great Lakes Symposium on VLSI 2022*, GLSVLSI '22, pages 147–151, New York, NY, USA, June 2022. Association for Computing Machinery.

[13] W Rhett Davis, John Wilson, Stephen Mick, Jian Xu, Hao Hua, Christopher Mineo, Ambarish M Sule, Michael Steer, and Paul D Franzon. Demystifying 3d ics: the pros and cons of going vertical. *IEEE Design & Test of Computers*, 22(6):498–510, 2005.

[14] Milind Dawande, Pinar Keskinocak, Jayashankar M Swaminathan, and Sridhar Tayur. On Bipartite and Multipartite Clique Problems. *Journal of Algorithms*, 41(2):388–403, November 2001.

[15] Krithika Dhananjay, Prachi Shukla, Vasilis F. Pavlidis, Ayse Coskun, and Emre Salman. Monolithic 3D Integrated Circuits: Recent Trends and Future Prospects. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(3):837–843, March 2021. Conference Name: IEEE Transactions on Circuits and Systems II: Express Briefs.

[16] Sophie Dupuis and Marie-Lise Flottes. Logic Locking: A Survey of Proposed Methods and Evaluation Metrics. *Journal of Electronic Testing*, 35(3):273–291, June 2019.

[17] Hans Eisenmann and Frank M. Johannes. Generic global placement and floorplanning. In *Proceedings of the 35th annual conference on Design automation conference - DAC '98*, pages 269–274, San Francisco, California, United States, 1998. ACM Press.

[18] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022.

[19] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring Network Structure, Dynamics, and Function using NetworkX. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[20] Meng-Kai Hsu, Valeriy Balabanov, and Yao-Wen Chang. TSV-Aware Analytical Placement for 3-D IC Designs Based on a Novel Weighted-Average Wirelength Model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(4):497–509, April 2013. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[21] Frank Imeson, Ariq Emtenan, Siddharth Garg, and Mahesh Tripunitara. Securing computer hardware using 3d integrated circuit (ic) technology and split manufacturing for obfuscation. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 495–510, 2013.

[22] Richard Wayne Jarvis and Michael G McIntyre. Split manufacturing method for advanced semiconductor circuits, March 27 2007. US Patent 7,195,931.

[23] A.B. Kahng, J. Lach, W.H. Mangione-Smith, S. Mantik, I.L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe. Watermarking techniques for intellectual property protection. In *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175)*, pages 776–781, June 1998.

[24] Andrew B. Kahng, Minsoo Kim, Seungwon Kim, and Mingyu Woo. RosettaStone: Connecting the Past, Present, and Future of Physical Design Research. *IEEE Design & Test*, 39(5):70–78, October 2022. Conference Name: IEEE Design & Test.

[25] A. Kennings and I. Markov. Analytical minimization of half-perimeter wirelength. In *Proceedings 2000. Design Automation Conference. (IEEE Cat. No.00CH37106)*, pages 179–184, January 2000.

[26] Dae Hyun Kim, Krit Athikulwongse, and Sung Kyu Lim. Study of Through-Silicon-Via Impact on the 3-D Stacked IC Layout. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(5):862–874, May 2013. Conference Name: IEEE Transactions on Very Large Scale Integration (VLSI) Systems.

[27] Myung-Chul Kim and Igor L. Markov. ComPLx: A Competitive Primal-dual Lagrange Optimization for Global Placement. In *Proceedings of the 49th Annual Design Automation Conference*, pages 747–752, San Francisco California, June 2012. ACM.

[28] Myung-Chul Kim, Natarajan Viswanathan, Charles J. Alpert, Igor L. Markov, and Shyam Ramji. MAPLE: multilevel adaptive placement for mixed-size designs. In *Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design*, pages 193–200, Napa California USA, March 2012. ACM.

[29] D. Kirovski and M. Potkonjak. Local watermarks: methodology and application to behavioral synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(9):1277–1283, September 2003. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[30] Farinaz Koushanfar. Provably secure active ic metering techniques for piracy avoidance and digital rights management. *IEEE Transactions on Information Forensics and Security*, 7(1):51–63, 2011.

[31] Bon Woong Ku, Kyungwook Chang, and Sung Kyu Lim. Compact-2D: A Physical Design Methodology to Build Commercial-Quality Face-to-Face-Bonded 3D ICs. In *Proceedings of the 2018 International Symposium on Physical Design*, ISPD '18, pages 90–97, New York, NY, USA, March 2018. Association for Computing Machinery.

[32] C. Lattner and V. Adve. LLVM: a compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pages 75–86, March 2004.

[33] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 2–14, February 2021.

[34] John H. Lau. Evolution, challenge, and outlook of TSV, 3D IC integration and 3d silicon integration. In *2011 International Symposium on Advanced Packaging Materials (APM)*, pages 462–488, October 2011. ISSN: 1550-5723.

[35] John H. Lau. Recent Advances and Trends in Advanced Packaging. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 12(2):228–252, February 2022. Conference Name: IEEE Transactions on Components, Packaging and Manufacturing Technology.

[36] Byunghyun Lee and Taewhan Kim. High-level TSV resource sharing and optimization for TSV based 3D IC designs. In *2013 IEEE International SOC Conference*, pages 153–158, September 2013. ISSN: 2164-1706.

[37] Chunho Lee, M. Potkonjak, and W.H. Mangione-Smith. MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of 30th Annual International Symposium on Microarchitecture*, pages 330–335, December 1997. ISSN: 1072-4451.

[38] Chunho Lee, Miodrag Potkonjak, and William H Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *International Symposium on Microarchitecture*. IEEE, 1997.

[39] Peiyu Liao, Siting Liu, Zhitang Chen, Wenlong Lv, Yibo Lin, and Bei Yu. DREAMPlace 4.0: Timing-driven Global Placement with Momentum-based Net Weighting. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 939–944, March 2022. ISSN: 1558-1101.

[40] Nimisha Limaye, Animesh B Chowdhury, Christian Pilato, Mohammed TM Nabeel, Ozgur Sinanoglu, Siddharth Garg, and Ramesh Karri. Fortifying rtl locking against oracle-less (untrusted foundry) and oracle-guided attacks. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 91–96. IEEE, 2021.

[41] Yuntao Liu, Michael Zuzak, Yang Xie, Abhishek Chakraborty, and Ankur Srivastava. Strong Anti-SAT: Secure and Effective Logic Locking. In *2020 21st International Symposium on Quality Electronic Design (ISQED)*, pages 199–205, March 2020. ISSN: 1948-3287.

[42] Jingwei Lu, Hao Zhuang, Pengwen Chen, Hongliang Chang, Chin-Chih Chang, Yiu-Chung Wong, Lu Sha, Dennis Huang, Yufeng Luo, Chin-Chi Teng, and Chung-Kuan Cheng. ePlace-MS: Electrostatics-Based Placement for Mixed-Size Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(5):685–698, May 2015. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[43] Jingwei Lu, Hao Zhuang, Ilgweon Kang, Pengwen Chen, and Chung-Kuan Cheng. ePlace-3D: Electrostatics based Placement for 3D-ICs. In *Proceedings of the 2016 on International Symposium on Physical Design*, pages 11–18, Santa Rosa California USA, April 2016. ACM.

[44] Yi-Chen Lu, Sai Surya Kiran Pentapati, Lingjun Zhu, Kambiz Samadi, and Sung Kyu Lim. TP-GNN: A Graph Neural Network Framework for Tier Partitioning in Monolithic 3D ICs. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, July 2020. ISSN: 0738-100X.

[45] Mohamed El Massad, Jun Zhang, Siddharth Garg, and Mahesh V. Tripunitara. Logic Locking for Secure Outsourced Chip Fabrication: A New Attack and Provably Secure Defense Mechanism, March 2017. arXiv:1703.10187 [cs].

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

[47] Md Rafid Muttaki, Roshanak Mohammadivojdan, Mark Tehranipoor, and Farimah Farahmandi. Hlock: Locking ips at the high-level language. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 79–84. IEEE, 2021.

[48] S. Ogrenci Memik, E. Bozorgzadeh, R. Kastner, and M. Sarrafzadeh. A super-scheduler for embedded reconfigurable systems. In *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)*, pages 391–394, November 2001. ISSN: 1092-3152.

[49] Shreepad Panth, Kambiz Samadi, Yang Du, and Sung Kyu Lim. Shrunk-2-D: A Physical Design Methodology to Build Commercial-Quality Monolithic 3-D ICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(10):1716–1724, October 2017. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[50] Christian Pilato, Animesh Basak Chowdhury, Donatella Sciuto, Siddharth Garg, and Ramesh Karri. Assure: Rtl locking against an untrusted foundry. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(7):1306–1318, 2021.

[51] Christian Pilato, Luca Collini, Luca Cassano, Donatella Sciuto, Siddharth Garg, and Ramesh Karri. On the optimization of behavioral logic locking for high-level synthesis. *arXiv preprint arXiv:2105.09666*, 2021.

[52] Christian Pilato, Luca Collini, Luca Cassano, Donatella Sciuto, Siddharth Garg, and Ramesh Karri. Optimizing the Use of Behavioral Locking for High-Level Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(2):462–472, February 2023.

[53] Jeyavijayan Rajendran, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Security analysis of logic obfuscation. In *Proceedings of Design Automation Conference*, 2012.

[54] Jeyavijayan Rajendran, Huan Zhang, Chi Zhang, Garrett S. Rose, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Fault Analysis-Based Logic Encryption. *IEEE Transactions on Computers*, 64(2):410–424, February 2015. Conference Name: IEEE Transactions on Computers.

[55] Masoud Rostami, Farinaz Koushanfar, and Ramesh Karri. A Primer on Hardware Security: Models, Methods, and Metrics. *Proceedings of the IEEE*, 102(8):1283–1295, August 2014. Conference Name: Proceedings of the IEEE.

[56] Jarrod A. Roy, Farinaz Koushanfar, and Igor L. Markov. EPIC: Ending Piracy of Integrated Circuits. In *2008 Design, Automation and Test in Europe*, pages 1069–1074, March 2008. ISSN: 1558-1101.

[57] Sandeep Kumar Samal, Deepak Nayak, Motoi Ichihashi, Srinivasa Banna, and Sung Kyu Lim. Monolithic 3D IC vs. TSV-based 3D IC in 14nm FinFET technology. In *2016 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, pages 1–2, October 2016.

[58] Abhrajit Sengupta, Mohammed Nabeel, Nimisha Limaye, Mohammed Ashraf, and Ozgur Sinanoglu. Truly Stripping Functionality for Logic Locking: A Fault-Based Perspective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(12):4439–4452, December 2020. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[59] Abhrajit Sengupta, Mohammed Nabeel, Muhammad Yasin, and Ozgur Sinanoglu. ATPG-based cost-effective, secure logic locking. In *2018 IEEE 36th VLSI Test Symposium (VTS)*, pages 1–6, April 2018. ISSN: 2375-1053.

[60] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z Pan, and Yier Jin. Appsat: Approximately deobfuscating integrated circuits. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 95–100. IEEE, 2017.

[61] Yuanqi Shen and Hai Zhou. Double dip: Re-evaluating security of logic encryption algorithms. In *Great Lakes Symposium on VLSI 2017*, 2017.

[62] Wen-Tsong Shiue and C. Chakrabarti. ILP-based scheme for low power scheduling and resource binding. In *2000 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 3, pages 279–282 vol.3, May 2000.

[63] Deepak Sirone and Pramod Subramanyan. Functional analysis attacks on logic locking. In *Design, Automation & Test in Europe Conference & Exhibition*, 2019.

[64] Dominik Sisejkovic, Farhad Merchant, Lennart M. Reimann, Harshit Srivastava, Ahmed Hallawa, and Rainer Leupers. Challenging the Security of Logic Locking Schemes in the Era of Deep Learning: A Neuroevolutionary Approach. *J. Emerg. Technol. Comput. Syst.*, 17(3):30:1–30:26, May 2021.

[65] A. Stammermann, D. Helms, M. Schulte, A. Schulz, and W. Nebel. Binding allocation and floorplanning in low power high-level synthesis. In *ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No.03CH37486)*, pages 544–550, November 2003.

[66] James E. Stine, Ivan Castellanos, Michael Wood, Jeff Henson, Fred Love, W. Rhett Davis, Paul D. Franzon, Michael Bucher, Sunil Basavarajaiah, Julie Oh, and Ravi Jenkal. FreePDK: An Open-Source Variation-Aware Design Kit. In *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*, pages 173–174, June 2007.

[67] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the security of logic encryption algorithms. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 137–143. IEEE, 2015.

[68] Benjamin Tan, Ramesh Karri, Nimisha Limaye, Abhrajit Sengupta, Ozgur Sinanoglu, Md Moshiur Rahman, Swarup Bhunia, Danielle Duvalsaint, R. D., Blanton, Amin Rezaei, Yuanqi Shen, Hai Zhou, Leon Li, Alex Orailoglu, Zhaokun Han, Austin Benedetti, Luciano Brignone, Muhammad Yasin, Jeyavijayan Rajendran, Michael Zuzak, Ankur Srivastava, Ujjwal Guin, Chandan Karfa, Kanad Basu, Vivek V. Menon, Matthew French, Peilin Song, Franco Stellari, Gi-Joon Nam, Peter Gadfort, Alric Althoff, Joseph Tostenrude, Saverio Fazzari, Eric Breckenfeld, and Kenneth Plaks. Benchmarking at the Frontier of Hardware Security: Lessons from Logic Locking. *arXiv:2006.06806 [cs]*, June 2020. arXiv: 2006.06806.

[69] Fatemeh Tehranipoor, Nima Karimian, Mehran Mozaffari Kermani, and Hamid Mahmoodi. Deep RNN-Oriented Paradigm Shift through BOCANet: Broken Obfuscated Circuit Attack. In *Proceedings of the 2019 Great Lakes Symposium on VLSI*, GLSVLSI '19, pages 335–338, New York, NY, USA, May 2019. Association for Computing Machinery.

[70] Wen-Pin Tu, Yen-Hsin Lee, and Shih-Hsu Huang. TSV sharing through multiplexing for TSV count minimization in high-level synthesis. In *2011 IEEE International SOC Conference*, pages 156–159, September 2011. ISSN: 2164-1706.

[71] Pruek Vanna-Iampikul, Chengjia Shao, Yi-Chen Lu, Sai Pentapati, and Sung Kyu Lim. Snap-3D: A Constrained Placement-Driven Physical Design Methodology for Face-to-Face-Bonded 3D ICs. In *Proceedings of the 2021 International Symposium on Physical Design*, pages 39–46, Virtual Event USA, March 2021. ACM.

[72] Nidish Vashistha, Md Latifur Rahman, Md Saad Ul Haque, Azim Uddin, Md Sami Ul Islam Sami, Amit Mazumder Shuo, Paul Calzada, Farimah Farahmandi, Navid Asadizanjani, Fahim Rahman, and Mark Tehranipoor. ToSHI - Towards Secure Heterogeneous Integration: Security Risks, Threat Assessment, and Assurance, 2022. Publication info: Preprint.

[73] Yang Xie and Ankur Srivastava. Mitigating sat attack on logic locking. In *Conference on Cryptographic Hardware and Embedded Systems*, 2016.

[74] Yuan Xie. Processor Architecture Design Using 3D Integration Technology. In *2010 23rd International Conference on VLSI Design*, pages 446–451, January 2010. ISSN: 2380-6923.

[75] Qiang Xu, Li Jiang, Huiyun Li, and Bill Eklow. Yield enhancement for 3D-stacked ICs: Recent advances and challenges. In *17th Asia and South Pacific Design Automation Conference*, pages 731–737, January 2012. ISSN: 2153-697X.

[76] Xiaolin Xu, Bicky Shakya, Mark M Tehranipoor, and Domenic Forte. Novel bypass attack and bdd-based tradeoff analysis against all known logic locking attacks. In *International conference on cryptographic hardware and embedded systems*, pages 189–210, 2017.

[77] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan J V Rajendran, and Ozgur Sinanoglu. SARLock: SAT attack resistant logic locking. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 236–241, May 2016.

[78] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. Removal Attacks on Logic Locking and Camouflaging Techniques. *IEEE Transactions on Emerging Topics in Computing*, 8(2):517–532, April 2020.

[79] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan (JV) Rajendran, and Ozgur Sinanoglu. Provably-Secure Logic Locking: From Theory To Practice. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1601–1618, New York, NY, USA, October 2017. Association for Computing Machinery.

[80] Muhammad Yasin, Abhrajit Sengupta, Benjamin Carrion Schafer, Yiorgos Makris, Ozgur Sinanoglu, and Jeyavijayan JV Rajendran. What to lock?: Functional and parametric locking. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017.

[81] Muhammad Yasin, Chongzhi Zhao, and Jeyavijayan JV Rajendran. SFLL-HLS: Stripped-Functionality Logic Locking Meets High-Level Synthesis. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–4, November 2019. ISSN: 1558-2434.

[82] Hanchen Ye, Cong Hao, Jianyi Cheng, Hyunmin Jeong, Jack Huang, Stephen Neuendorffer, and Deming Chen. ScaleHLS: A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 741–755, April 2022. ISSN: 2378-203X.

[83] Hanchen Ye, Hyegang Jun, and Deming Chen. HIDA: A Hierarchical Dataflow Compiler for High-Level Synthesis. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, volume 1 of *ASPLOS '24*, pages 215–230, New York, NY, USA, April 2024. Association for Computing Machinery.

[84] Lin Zhong and N.K. Jha. Interconnect-aware low-power high-level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(3):336–351, March 2005. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[85] Hai Zhou, Amin Rezaei, and Yuanqi Shen. Resolving the trilemma in logic encryption. In *International Conference on Computer-Aided Design (ICCAD)*, 2019.

[86] Michael Zuzak, Yuntao Liu, and Ankur Srivastava. Trace logic locking: Improving the parametric space of logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[87] Michael Zuzak, Yuntao Liu, and Ankur Srivastava. A Resource Binding Approach to Logic Obfuscation. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 235–240, December 2021. ISSN: 0738-100X.