CHOOSE-YOUR-OWN ADVENTURE: A LIGHTWEIGHT, HIGH-PERFORMANCE APPROACH TO DEFECT AND VARIATION MITIGATION IN RECONFIGURABLE LOGIC

Raphael Yoram Rubin

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2018

Supervisor of Dissertation	Graduate Group Chairperson
André M. DeHon, Professor Electrical and Systems Engineering	Lyle Ungar, Professor Computer and Information Science

Dissertation Committee:

Jonathan M. Smith, Pompa Professor of Engineering and Applied Science

Boon Thau Loo, Professor of Computer and Information Science

Joseph Devietti, Assistant Professor of Computer and Information Science

Dr. Stephen Trimberger, Xilinx, Inc. (Retired)

CHOOSE-YOUR-OWN ADVENTURE:
A LIGHTWEIGHT, HIGH-PERFORMANCE APPROACH TO DEFECT AND VARIATION MITIGATION IN RECONFIGURABLE LOGIC

COPYRIGHT

Raphael Yoram Rubin

2018

This work is licensed under the Creative Commons
Attribution-NonCommerical-ShareAlike 4.0 International
(CC BY-NC-SA 4.0) License.
To view a copy of this license, visit:
https://creativecommons.org/licenses/by-nc-sa/4.0/

For Poppy.

Acknowledgements

People

I'd like to thank my advisor, André DeHon, and my committee chair, Jonathan M. Smith, for seemingly unlimited patience and support. I am also grateful to the rest of my committee, Boon Thau Loo, Joe Devietti, and Steve Trimberger for their feedback and approval.

My labmate, Ben Gojman, has been my unwavering comrade in arms during our Ph.D. studies, an excellent research collaborator and aesthetic advisor, and a caring friend. Nikil Mehta and Hans Giesen have also been key research contributors and valuable friends.

My parents, Alice Palokoff and David Rubin, have been waiting for far too long to see this dissertation done. I hope the results vindicate their many years of anticipation and encouragement.

My wife, Anne Hanna, always knew that I was going to get this done. Thanks, most of all, to her, for taking significant time out from working on her own dissertation to help drag me across the finish line.

Funding

In addition to the amazing people above, this research was also supported by the following funding sources:

- NSF grants CCF-0403674, CCF-0726602, CCF-0904577, and CNS-1406225
- DARPA grant HR0011-13-C-0005
- ONR grant N00014-15-1-2006
- Toshiba Corporation
- the University of Pennsylvania School of Engineering and Applied Science

All opinions, findings, conclusions, and recommendations expressed in this document are those of the author and do not necessarily reflect the views of any funding agencies.

ABSTRACT

CHOOSE-YOUR-OWN ADVENTURE:
A LIGHTWEIGHT, HIGH-PERFORMANCE APPROACH TO DEFECT AND VARIATION MITIGATION IN RECONFIGURABLE LOGIC

Raphael Yoram Rubin

André M. DeHon

For field-programmable gate arrays (FPGAs), fine-grained pre-computed alternative configurations, combined with simple test-based selection, produce limited per-chip specialization to counter yield loss, increased delay, and increased energy costs that come from fabrication defects and variation. This lightweight approach achieves much of the benefit of knowledge-based full specialization while reducing to practical, palatable levels the computational, testing, and load-time costs that obstruct the application of the knowledge-based approach. In practice this may more than double the power-limited computational capabilities of dies fabricated with 22nm technologies.

Contributions of this work:

- Choose-Your-own-Adventure (CYA), a novel, lightweight, scalable methodology to achieve defect and variation mitigation
- Implementation of CYA, including preparatory components (generation of diverse alternative paths) and FPGA load-time components
- Detailed performance characterization of CYA
  - Comparison to conventional loading and dynamic frequency and voltage scaling (DFVS)
  - Limit studies to characterize the quality of the CYA implementation and identify potential areas for further optimization

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Hypothesis

Pre-computed per-net alternatives provide a lightweight method to exploit the reconfigurability and redundancy of field-programmable gate arrays (FPGAs) to address fabrication defects and variations. The improvements to yield and energy efficiency can more than double power-limited performance of dies fabricated with 22nm technologies.

1.2 The Problem

Moore's Law [62] describes the trend of the doubling of the number of transistors per die every 18 months, largely resulting from the reduction of the size of each transistor (one of many characteristics included in the umbrella term "feature sizes"). Dennard scaling [17], a companion trend of shrinking per-transistor power consumption, enables Moore's Law: smaller transistors can be switched with lower voltage, resulting in constant power density despite the increasing density of transistors enabled by

size-scaling. However, Moore's Law scaling has now driven component sizes down to near-atomic scales, where detailed control over produced structures becomes increasingly difficult, if not impossible. Operating voltages must be raised and clocks must be slowed to compensate for the resulting variations, preventing the voltage from scaling with field-effect transistor (FET) feature size. Consequently, Dennard scaling has ended — energy-efficiency improvements no longer match the pace of transistor area scaling. We can no longer operate as many transistors as we can fabricate in a given area. This scaling mismatch is expected to worsen with continued scaling, leading to a steady increase in the fraction of each die which must remain inactive to meet strict power budgets. This trend/problem is commonly termed "Dark Silicon" [20].

Mehta [61] (elaborated in Mehta [59]) demonstrates that component-specific mapping (CSM) can be used to improve operational energy efficiency for FPGAs. A complementary project, by Gojman [26], demonstrates how to perform the detailed delay characterization of each chip required for knowledge-based CSM. This represents a significant advance beyond Culbertson's [14] defect-only characterization (detection of lost signals, as opposed to the slowing of signals caused by variations). However, Gojman's characterization methodology requires days or even weeks of analysis for a single chip, and utilizing these measurements requires redoing stages of the design compilation for each individual chip. The run times of current place-and-route tools are already considered overly burdensome, often taking hours or even days. Multiplying that cost by the number of deployments, while at the same time adding substantial per-chip characterization costs, is more than sufficient to prevent widespread adoption.

Rather than measuring first and then specializing all aspects of a design for every target chip, Choose-Your-own-Adventure (CYA) produces as-needed specialization at load time using composable pre-computed partial repair solutions and coarse-

grained as-needed testing. This approach avoids the per-chip computer-aided design (CAD), complete characterization, and data management challenges that plague Mehta's knowledge-based approach. Despite having much less chip-specific information, CYA achieves on average 52% of the energy savings of knowledge-based CSM that are specifically attributable to the customization of the mapping (Table 7.5). In addition, both techniques inherently incorporate a form of dynamic frequency and voltage scaling (DFVS) (see Section 1.3). Counting the effects of DFVS, CYA achieves on average 83% of the total energy savings of knowledge-based CSM relative to conventional loading practices (Table 7.6). This dissertation describes the details of the CYA methodology and its implementation, and presents experiments demonstrating its capabilities, sensitivities, and limitations.

1.3 Previous Strategies for Mitigating Post-Fabrication Defects and Variations

Some post-fabrication strategies use low-level physical approaches to effectively remove defects and reduce variations. Body biasing [80, 63, 48] is one such technique. Body biasing divides the chip into control regions and shifts the threshold voltage in each region in order to reverse variation-induced V_{th} shifts. This works well when V_{th} shifts are spatially correlated or uniform across an entire die. However, with random variations (an increasing concern, see [43]), neighboring transistors may vary in opposite directions from nominal V_{th}. In such a case, applying a single correction factor to both transistors will improve one but further degrade the other. The larger the granularity of the control regions, the smaller the potential benefit we can get from uniformly correcting each region. The significant area overhead required to support body biasing negates the utility of this technique at granularities that effectively

address random variations. For a more detailed analysis see Section 3.2.2.

Architectural sparing (including extra resources to replace defective ones), such as the row and column sparing used by Altera [13, 57, 47] and the segment sparing described by Yu [95], is another relatively low-level corrective approach. However, both the extra resources and their supporting infrastructure add area and delay, which also increases energy costs. Moreover, in light of variations, it is difficult to determine a priori which resources to discard or remap when the specific operational voltage is not known. This determination can be made at load-time, but at this point the sparing approach takes on the character of CSM, becoming more of an architectural optimization for that purpose than a distinct solution.

As variations increase, it becomes more important not only to correct or mitigate variations, but also to more precisely quantify what correction or mitigation even means. Specifically, we want to be able to say that, using a given correction protocol, a certain class of designs can be expected to operate reliably on chips with a certain level of variations, within certain voltage, clock speed, and other operating parameter ranges, for the full desired lifetime of the device. I refer to the process of establishing these parameter ranges as "margining".

Acquiring more detailed knowledge about the properties of each individual chip often enables one to certify some chips for use with more demanding operational parameters (i.e., tighter margins). For example, speed/power binning is the practice of testing chips at a few different clock speeds and voltages to determine the rough performance characteristics of each chip. The chips that are reliable at higher frequencies or lower voltages are offered at a premium commensurate with the extra utility to customers, rather than limiting all chips to the safe settings of the worst of the bunch.

Dynamic voltage scaling (DVS) [11], dynamic frequency scaling (DFS), and the

combination, dynamic frequency and voltage scaling (DFVS), take this knowledge-based customization a step further, adjusting the supply voltage, clock speed, or both to save power while maintaining the requirements of a specific application or workload. For FPGAs, DVS/DFS often refer to the determination of an appropriate supply level/clock speed for a specific design on a given chip, rather than continuing to adjust these parameters during operation to match current workloads. DVS/DFS can only optimize the efficiency/speed at which a pre-determined mapping operates on any given chip, and so their effectiveness is limited by the worst of that chip's resources (transistors, wires, etc.) used by the given mapping. This limitation can be addressed with the use of CSM to replace bad resources with better resources, enabling dynamic scaling to achieve lower voltages/higher clock speeds.

At this point in the process, we have accepted the demise of the perfect-chip assumption. The next step now becomes resilient design: creating circuits that will operate correctly on partially defective and otherwise variable hardware. The classic modular redundancy approach [65, 10, 2, 67] provides solutions to high rates of defects, but comes with a high cost. The minimum implementation triplicates (in special cases duplicates) a circuit, and thus triples (or doubles) the energy costs. Alternative approaches can be much more efficient.

Statistical static timing analysis (SSTA) [85, 49, 77, 52, 44] factors delay sensitivity and a model of expected delay variation into the CAD flow. At best this reduces sensitivity to post-fabrication variations and enables some tightening of margins. However, this reduction only goes so far and still requires the margins to account for the fact that any single transistor may vary enough to slow down the design. It would be better if we could entirely avoid using those worst resources, so that we can push further and reduce margins, if not eliminate them entirely, to guard against post-fabrication variations.

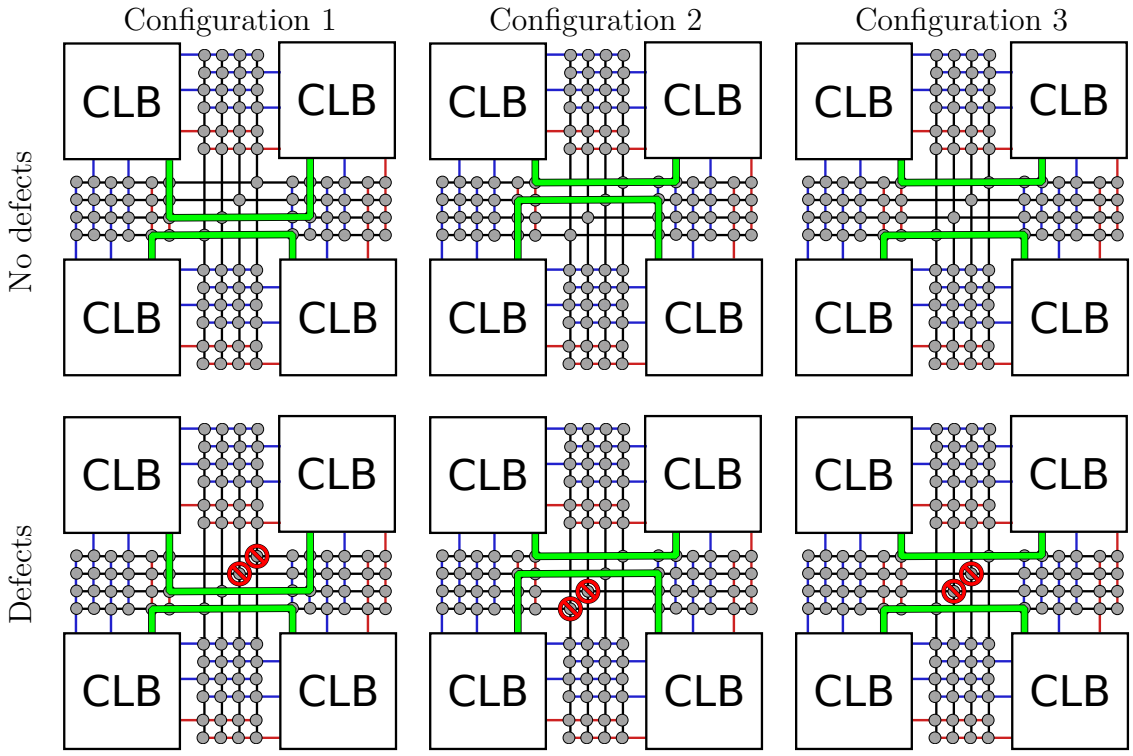


Figure 1.1: The rich pool of interchangeable resources in an FPGA provides many functionally equivalent configurations. For nominal, defect-free FPGAs, arbitrary selection is adequate. When defects are present, CSM enables us to select a configuration that best matches each chip.

Design-specific performance tweaks such as clock skewing [76, 74] and slack stealing can also provide delay improvements for nominal or lightly varying devices. However, these techniques do not address the complete failure of some switches that occurs at more energy efficient voltages.

1.3.1 Component-specific Mapping (CSM)

FPGAs are designed to maximize the range of supported user circuits. This is achieved by providing rich pools of generic, interchangeable resources, almost invariably exceeding the needs of any single application (illustrated in Figure 1.1). For working around defects and variations, this translates to plentiful spare resources and

the freedom to discard bad resources for usable spares. We can use this freedom to adapt the mapping of a user circuit to the unique capacities of each target FPGA, i.e., CSM.

This concept was demonstrated in practice by Hewlett Packard in the context of the TERAMAC project [14]. To make it work, all FPGAs are thoroughly tested to locate defects, and the resulting information stored for later use. When an application is mapped (much like a computer program is compiled) for TERAMAC, the tools use the stored knowledge to work around defects. Mehta [59] demonstrated the ability of this form of knowledge-based specialization to tighten variation-driven energy margins. His results set an energy efficiency target for the present work (see Chapter 7).

This approach pre-emptively performs full custom mapping for each part of a user design for each target FPGA, performing considerable work to specialize nets which may not require the extra attention. In contrast, Lakamraju and Tessier [46] propose an incremental approach: work with the nominal configuration and only reroute nets affected by defects or disturbed by the rerouting of other nets. For small numbers of defects, they show that this approach produces a repair solution in a fraction of the time required for routing from scratch. The speedup comes largely from leveraging common solutions (i.e., the original pre-repair route) which account for most of the configurations of most nets (when defects are few). Taking this concept a step further, individual repair solutions may satisfy the needs of multiple chips with differing defect patterns (see Figure 1.2). Thus, we can save work by reusing repair solutions in addition to portions of the pre-repair configurations. Moreover, these single-net repair solutions can be pre-computed, eliminating the troublesome requirement that the FPGA loader also be capable of performing full routing.

Hyder and Wawrzynek [34] use this idea — they shuffle the various bitstreams

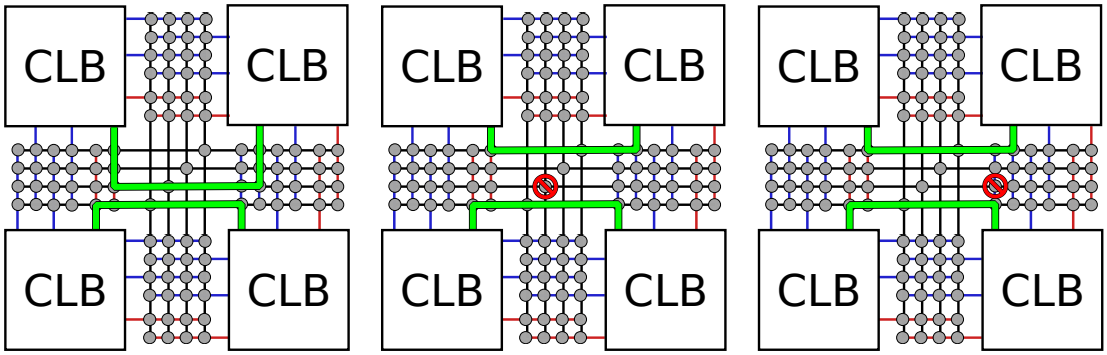
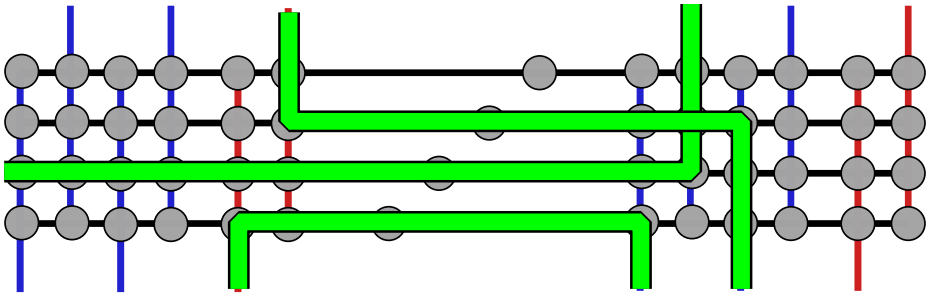
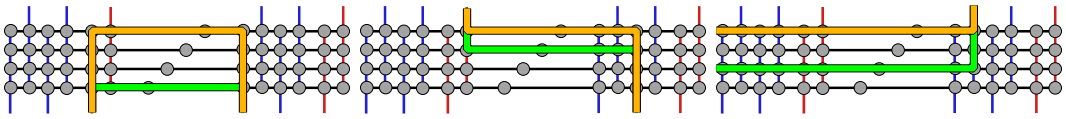


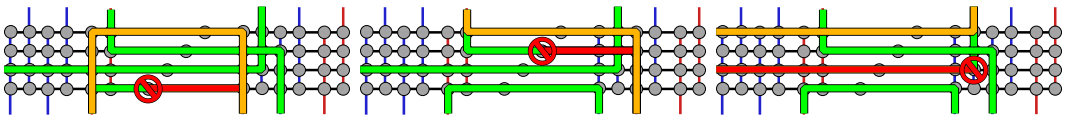
Figure 1.2: (a) shows a design mapped on a healthy FPGA. A single repair solution addresses the different defects in FPGAs (b) and (c).



(a) Conventional FPGA configuration with a single path for each of three shown two-point nets.



(b) Configuration with path multiplicity. Each of the three nets is shown with two different paths (a green original path, and an orange alternative path). When configuring an FPGA, one of the two paths is selected for each net.



(c) Path selection for the three nets, used to work around three different defects.

Figure 1.3: Path multiplicity offers repair flexibility.

(single FPGA configurations) amongst the chips in a multi-FPGA system (5 chips in an experimental BEE2) to find a functional overall configuration. Xilinx uses this idea for the EasyPath™program [42, 50, 86] to recover some value from chips that fail to meet normal production specs. Customer applications are matched to potentially defective FPGAs, using design-specific testing. Chips that pass are sold at a reduced rate to the respective customers with the guarantee of full performance and functionality of a standard FPGA, but only for the one specific application.

With CSM, a single design is compiled into different configurations with equivalent functionality to match the characteristics of specific FPGAs. We can also produce multiple equivalent mappings targeting coverage of avoidable defect patterns (again, consider the three example configurations in Figure 1.1) rather than the specific defect patterns of individual chips. The library of alternative configurations can be used much like the different system components of Hyder and the different applications in EasyPath™. However, in this case, the goal is to produce the same functionality in each chip.

Configuration multiplicity has been studied at a number of different scales [75], including the coarsest (full-chip bitstream multiplicity) [84, 56], coarse block multiplicity [32] and shuffling (an intra-chip variant of Hyder's approach), and multiplicity at the level of paths or "two-point nets" (Trimberger [83], Campergher [9], Rubin [70]). Figure 1.3 shows a simplified example of path multiplicity.

Fine-grained solutions are composable: each repair domain may choose alternatives independently to some degree, resulting in a combinatorially large number of global configurations from relatively few alternatives. This provides a significant advantage in terms of the number of possible solutions produced by a given alternative generation effort, so much so that path multiplicity is capable of achieving most of the benefits of full-knowledge-based CSM despite its significantly lower costs (see

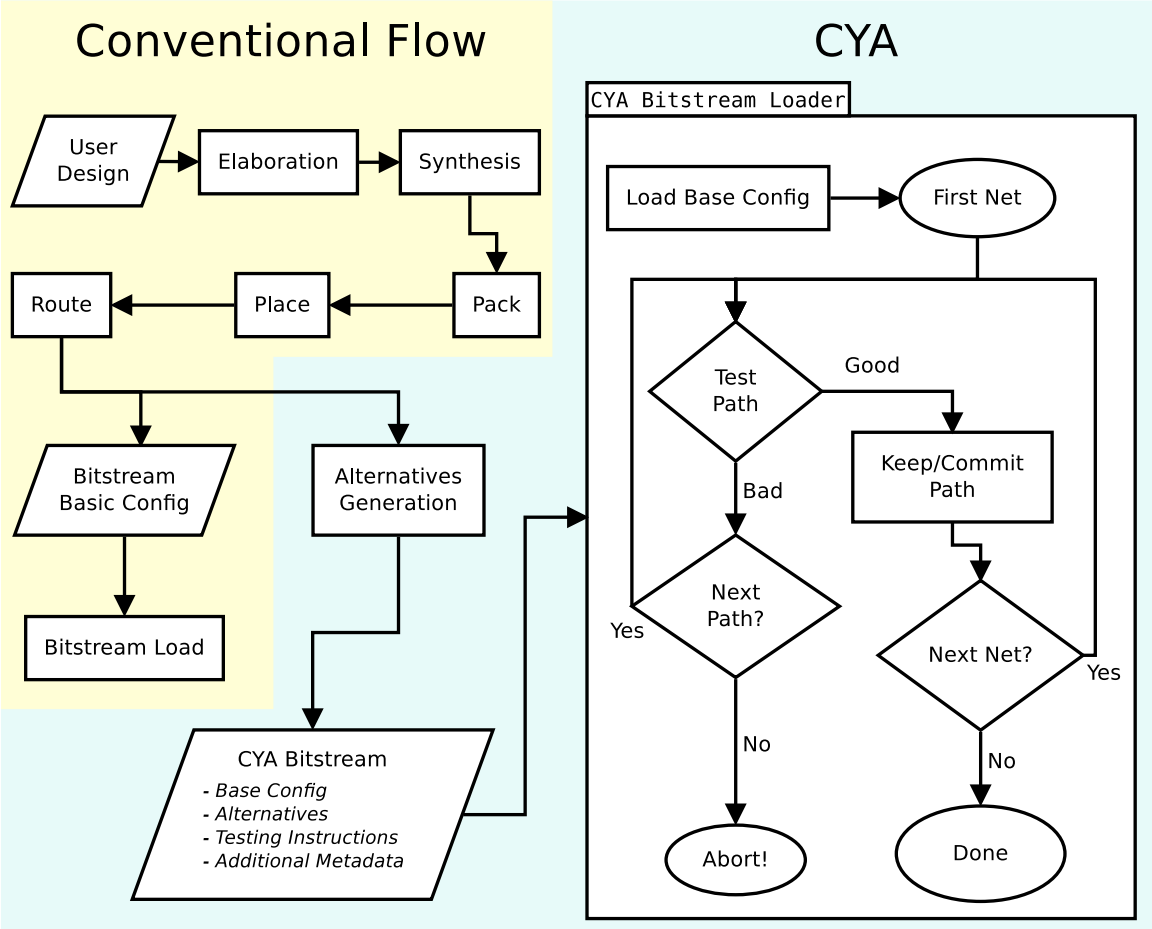


Figure 1.4: CYA CAD flow vs. conventional flow

Chapter 7).

This dissertation describes Choose-Your-own-Adventure (CYA), my novel approach to CSM, details the implementation I used to test it, and demonstrates its advantages.

1.4 CYA: Path-Multiplicity Based CSM

Choose-Your-own-Adventure (CYA) combines path based multiplicity and simple testing into a powerful, lightweight, and practical methodology for CSM.

The CYA process (Figure 1.4) matches conventional FPGA CAD flow until after the routing stage, at which point the conventional flow has a complete and fixed configuration. The next CYA stage is alternatives generation. One two-point net at a time, the alternatives generator searches the FPGA resource graph for other legal paths to connect that source/sink pair. This procedure processes each two-point net until the requested number of alternative paths are discovered, or until reasonable effort fails to reveal additional distinct paths. In general, one would request at least one alternative for each two-point net. In this work, most studies use a target of 64 alternatives.

The core of the alternatives generator was inspired by the popular routing algorithm Pathfinder [58]. It is built around shortest-path searches (Dijkstra [19] or A^* [31]), where costs that shift from one search to the next drive solutions towards specific goals. For alternatives generation, the goals are first to do no harm, and second to produce a diverse set of paths to maximize the probability that at least one will work well for each chip.

To achieve the first goal, I constrain alternate paths to resources that are unused in the base route or only used by the originating net. It should be noted that these nets start with logic blocks (logic elements (LEs) or look-up tables (LUTs)). Therefore, these alternatives provide repair/swapping for both interconnect and logic.

The second goal, alternative diversity, is achieved by penalizing expansions that follow previously recorded alternatives.

$$\begin{aligned} Cost_i = (Cost_{i-1} + NodeCosts) \\ \cdot (1 + PrevAltCount \cdot RepetitionPenaltyFactor) \end{aligned} \tag{1.1a}$$

$$TotalPenalty \geq RepetitionPenaltyFactor^{Length\ of\ Shared\ Prefix} \tag{1.1b}$$

This common-prefix path penalty adds exponential pressure on the search to break away from the previous paths. An additional per-node usage penalty provides linear pressure to avoid previously used nodes, whether or not the paths to those nodes are part of previously recorded prefixes. These searches are repeated until the requested number of alternatives are generated or no new paths are discovered after several attempts, where the penalty factor is increased with each failed attempt. The alternative generation procedures are covered in depth in Section 4.2.3.

Once generated, alternatives are added to the bitstream (the "machine code" streamed into an FPGA to configure it), along with the conventional "base" route. The CYA bitstream may also include supplemental instructions for testing procedures or other information to assist the mapping process.

The CYA bitstream loader builds on the conventional loader, adding the support needed to select alternatives and to act on those decisions. The first step is to map the base configuration, much like a conventional loader. Next, the mapped FPGA is tested to verify proper functionality. Then, broken two-point nets are unmapped and each is repaired individually by mapping and testing its alternatives one after the other until one works. If any net runs out of alternatives, the loader reports failure and aborts.

The pre-computed alternatives focus on routed nets, that is, nets that send signals outside the originating cluster. Local nets, those that terminate entirely within the originating cluster, are simpler to remap on the fly. In the process of evaluating alternatives for a routed net, the loader may remap local nets to any currently available LEs within the cluster.

When focused solely on correcting defects, the loader need only test for functional correctness. During the repair of each two-point net, each candidate path will be tested to determine its suitability. Iterating through each net with the same tests

provides a simple but complete testing regime.

Extending the loader to support testing of the circuit functionality at any specified fixed speed requires only a few minor changes. First, the modeled delays from the original routing are annotated into the bitstream. The annotated delay budgets are then adjusted to allow some nets to operate more slowly than modeled when this will not slow down the entire design (elaborated in Section 5.2.1). Finally, functionality testing is performed at a frequency related to the slack budget for each two point net.

Delay optimization is achieved by performing complete, fixed-speed load/repair at various target delays, following a chosen search pattern (e.g., a binary search for the fastest workable speed). To provide delay budgets for each two point net at different clock frequencies, I simply scale the original budgets by the ratio of the current target delay to the modeled target delay recorded in the bitstream.

To optimize energy, delay optimizing load/repair is performed at various voltages, to select the most energy-efficient voltage at which the design functions properly. Further research may provide models to estimate the most efficient voltage; however, for my results (see Section 6.2), I assume power metering to directly identify the most efficient voltage rather than relying on approximations from other parameters.

1.5 Results

The experiments (discussed in Section 4.4 and forward) follow a set of benchmarks (the "Toronto 20" [6] benchmarks, which are commonly used in FPGA research) and a set of virtual chips (each with a unique pattern of defects and fabrication variations) through a set of CAD flows and loading simulations. Section 4.4.3 details the basic FPGA architecture used for the experiments up through Section 4.6.2, as well as the meanings of the architectural parameters explored. Section 4.6.3 describes an

updated architecture that was used from that point forward.

The CAD stages in these experiments are based on an analogue of traditional CAD flow — synthesis, packing, placement, and routing (the yellow-shaded portions of Figure 1.4) — followed by simulated loading and analysis to measure area, delay, and energy. Additional stages provide data for full-knowledge routing and CYA (the blue-shaded portions of Figure 1.4). This CAD flow is feed-forward and intermediate results are preserved to be used for different experiments at later stages. For example, I only resynthesize a netlist when changing LUT size, but I use fixed synthesis, packing, and placements if I want to examine the impact of changing channel width. Changes in V_{dd} or defect/variation patterns (i.e., comparing different chips) only affect the final stage of the flow (loading for most experiments, routing for full-knowledge).

Sections 4.5 to 5.1 explore the ability of CYA to address post-fabrication defects. Figure 1.5 illustrates the resulting substantial yield improvements for `des` — CYA maintains full yield at defect densities several orders of magnitude beyond both the point at which observable numbers of imperfect chips begin to appear, and the point at which oblivious mapping (use of a single static map) on those imperfect chips begins to experience failures. Across the full Toronto 20 benchmark suite, I saw no failed loads at defect densities as high as 81 defects/cm^2 (Table 4.10). Moreover, it is likely that the rare failures at this defect density, as well as many of the failures at higher defect densities, may be addressed (as future work) with minor improvements to CYA, with complementary solutions (e.g., [46]), or by establishing testing methodologies to catch obviously irreparable combinations of defects. Consequently, it is significant that, in my experiments, CYA maintains 95% yield up to 2400 defects/cm^2 for all Toronto 20 designs.

Moving from hard defect repair to variation tolerance requires only a minimal shift in testing procedures — now, instead of asking only whether a given path is functional

or broken, I ask whether that path functions fast enough to satisfy a prescribed delay budget and use this test to determine which paths are usable or need to be replaced with an alternative. The logic of the CYA repair process otherwise remains unchanged. The results of these experiments are reported in Sections 5.2 and 6.1. At the standard operating voltage for the 22nm technology used in these experiments ($V_{dd} = 0.8$V), CYA provides only a small speed-up. However, at lower voltages, CYA begins to substantially improve both total and parametric yield (yield of chips that operate at a given speed). Figures 5.5 to 5.7 show examples of this behavior for `des`.

These low-voltage parametric yield enhancements enable the most exciting capability of variation tolerance: energy savings achieved through voltage reduction (Section 6.2). Lowering the voltage reduces dynamic energy consumption, but this comes at the cost of increasing variation-related faults and slowdowns. Conventionally-loaded chips are forced to run slower (at increasing cost in static energy), or fail to run at all. CYA repair substantially mitigates these effects, allowing both yield and energy recovery. Table 6.6 shows the energy reduction of CYA relative to conventional loading for all Toronto 20 designs (an average of 61%), while Table 6.7 shows the recovery of the energy lost to variations (an average of 70%).

Chapter 7 compares the delay and energy benefits of CYA to those of more costly CSM techniques, both to estimate the maximum possible gains available with CSM, and to map out the load-time and other costs associated with realizing further gains beyond those offered by CYA. The limited alternative list and greedy selection used by CYA account for roughly half of the difference in energy savings achievable with CYA as compared to the (impractically costly but maximally effective) technique of "full-knowledge" routing. Overall, CYA achieves 52% of the overall energy savings achievable with CSM, and 83% of the portion of that savings that is attributable solely to mapping specialization (rather than simply to the inherently-included DFVS).

Moreover, given that full-knowledge routing requires 7 orders of magnitude more load time to mitigate only 15% more of the energy cost of variations than CYA does (85%, compared to CYA's 70%), it seems clear that CYA represents a cost/benefit sweet spot in the CSM energy savings landscape.

1.6 Pertinent Publications

The following publications of mine cover work discussed in or closely related to this dissertation:

1. Choose-Your-Own-Adventure Routing: Lightweight Load-Time Defect Avoidance [69, 70]
2. Component-Specific Mapping for Low-Power Operation in the Presence of Variation and Aging [27]
3. Timing-Driven Pathfinder Pathology and Remediation: Quantifying and Reducing Delay Noise in VPR-Pathfinder [71, 72]
4. Limit Study of Energy & Delay Benefits of Component-Specific Routing [61]
5. Pitfalls and Tradeoffs in Simultaneous, On-Chip FPGA Delay Measurement [51]
6. Continuous Online Self-Monitoring Introspection Circuitry for Timing Repair by Incremental Partial-reconfiguration (COSMIC TRIP) [22, 23]
7. Quality-Time Tradeoffs in Component-Specific Mapping: How to Train Your Dynamically Reconfigurable Array of Gates with Outrageous Network-delays (DRAGON) [24]
8. Self-Adaptive Timing Repair [25]

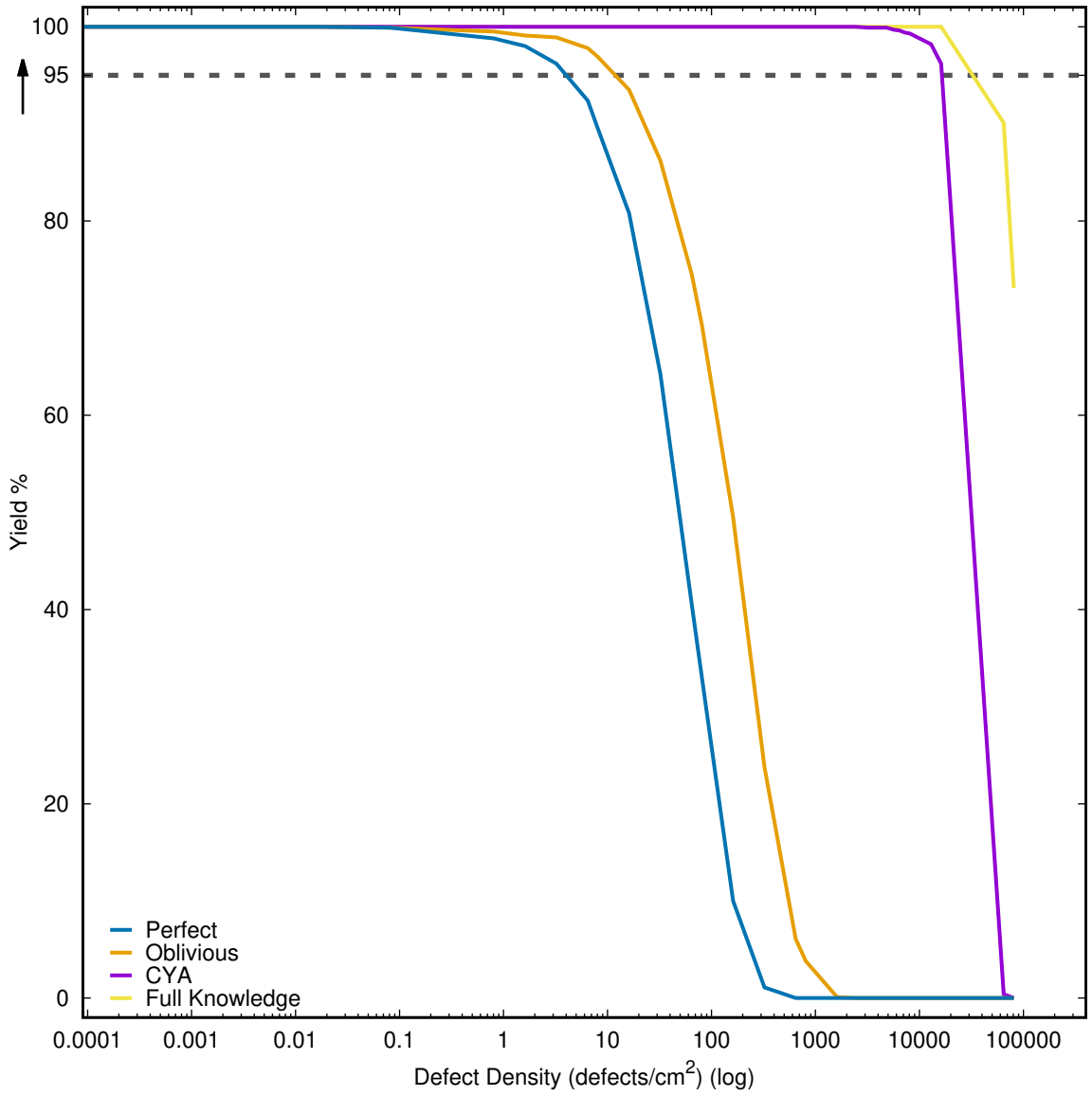


Figure 1.5: Yield comparison of the following routing methodologies: requiring perfect chips ("perfect"), using static mapping on all chips ("oblivious"), using CYA mapping on all chips ("CYA"), and using full-knowledge routing on all chips ("full-knowledge").

Yield figures are for `des` mapped to 1000 chips, using 22nm technology and a die area of 9 million λ^2 ($4400\mu m^2$). All chips had 20% extra base tracks, 16 reserved tracks, and single-driver interconnect, and the CYA mapping used 64 alternatives. Simulations were performed in VPR 5 [53].

The "goodness" arrow next to the yield axis is a reminder that higher yields are preferred.

1.7 Who Did What

The full-knowledge CSM work was a group project. Benjamin Gojman specialized in the implications for nanowire-based technologies. Nikil Mehta specialized in conventional and near-future technologies. Mehta's work provided many of the detailed area, delay, and energy models used to produce the data presented throughout my dissertation.

My own modifications to VPR to support defects, variations, and knowledge-based routing provided a foundation for the full-knowledge research as well as the CYA project. Relevant code files are included with the electronic deposit of this dissertation. Hans Geisen and I jointly modified channel construction in VPR to enable reservation for repair solutions in directional interconnect [24, 23] (see Section 4.6.3). Hans Geisen also extended CYA into incremental [24] and online [22] variants.

My work on noise management [72] for the PathFinder [58] routing algorithm, detailed in [71], made possible the meaningful full-knowledge results in both Mehta's dissertation and this one. I also used these modifications to prepare base routes for CYA (see Section 4.4.2).

1.8 Key Contributions

- CYA, a novel, lightweight, scalable methodology to achieve defect and variation mitigation
- Implementation of CYA, including preparatory components (generation of diverse alternative paths) and FPGA load-time components
- Detailed performance characterization of CYA
  - Comparison to conventional loading and DFVS
  - Limit studies to characterize the quality of the CYA implementation and identify potential areas for further optimization

Chapter 2

Setting

2.1 International Technology Roadmap for Semiconductors

The International Technology Roadmap for Semiconductors (ITRS), as the name implies, is a roadmap for the semiconductor industry. It is produced through the cooperation of companies that manufacture computer chips, companies that supply and produce materials and equipment for the manufacturers, and researchers (public, private, and academic). Its purpose is to set short-term and long-term goals for the industry and to identify the challenges that must be addressed to meet those goals. In short, the roadmap is the predominant authority on the research needs of the semiconductor industry.

The 2001 edition of the roadmap [36] mentioned non-visual defects, parametric defects, and process variation, but did not emphasize these as key challenges facing the industry. By the 2009 edition, this attitude had changed dramatically, with defect and variation issues being viewed as key challenges for near-term technologies. What caused this shift?

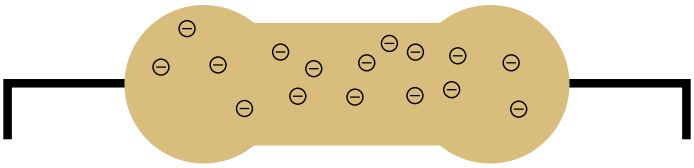


Figure 2.1: Cartoon representation of a resistor. Free charges carry current from one end to the other.

2.2 Transistor Fundamentals

A brief review of the basic concepts of transistor technology will help illustrate the changing nature of the problems which arise with transistors at the 22 nm technology node and beyond.

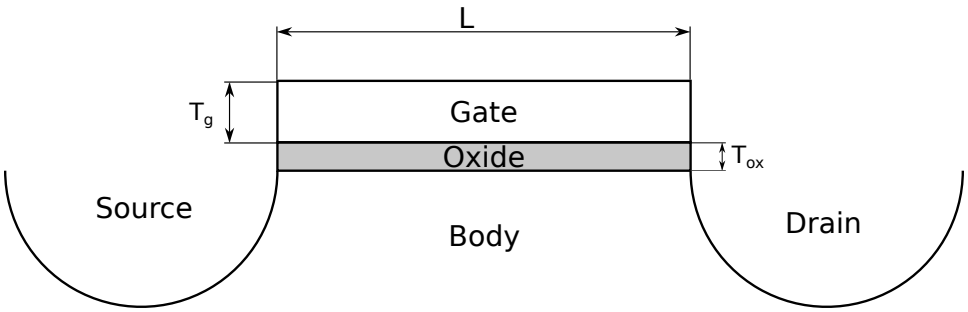
Current computer chips consist mostly of field-effect transistors (FETs) and wiring to connect those transistors. A FET is an analog device, essentially just a variable resistor. In a normal resistor (Figure 2.1), relatively mobile "charge carriers" (specifically, electrons) carry charges from one terminal to another. The resistance R of a resistor is a function (Equation (2.1)) of its length L, its cross-sectional area WH (width times height), the density of charge carriers, and how easy it is to move the charge carriers (mobility). The density and mobility of the charge carriers are jointly represented in the single resistivity parameter ρ, a property of the material of which the resistor is composed.

$$R = \rho \frac{L}{WH} \tag{2.1}$$

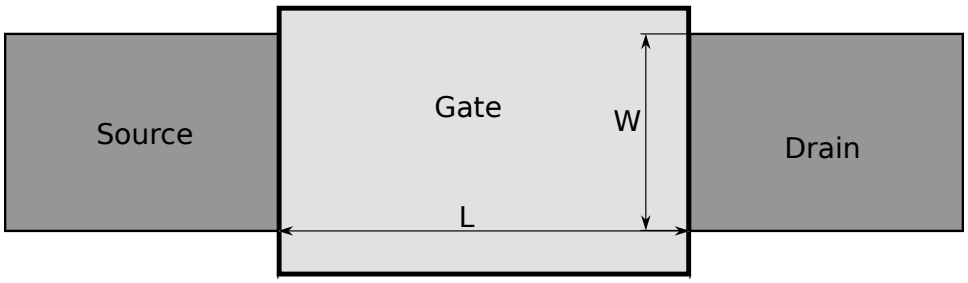
Inserting this equation into Ohm's law ($V = IR$), we get Equation (2.2), which relates the current I through the resistor to the voltage V applied across its terminals.

$$I = \frac{WHV}{\rho L} \tag{2.2}$$

In a FET (Figure 2.2), a "channel" region serves as a variable resistor which con-



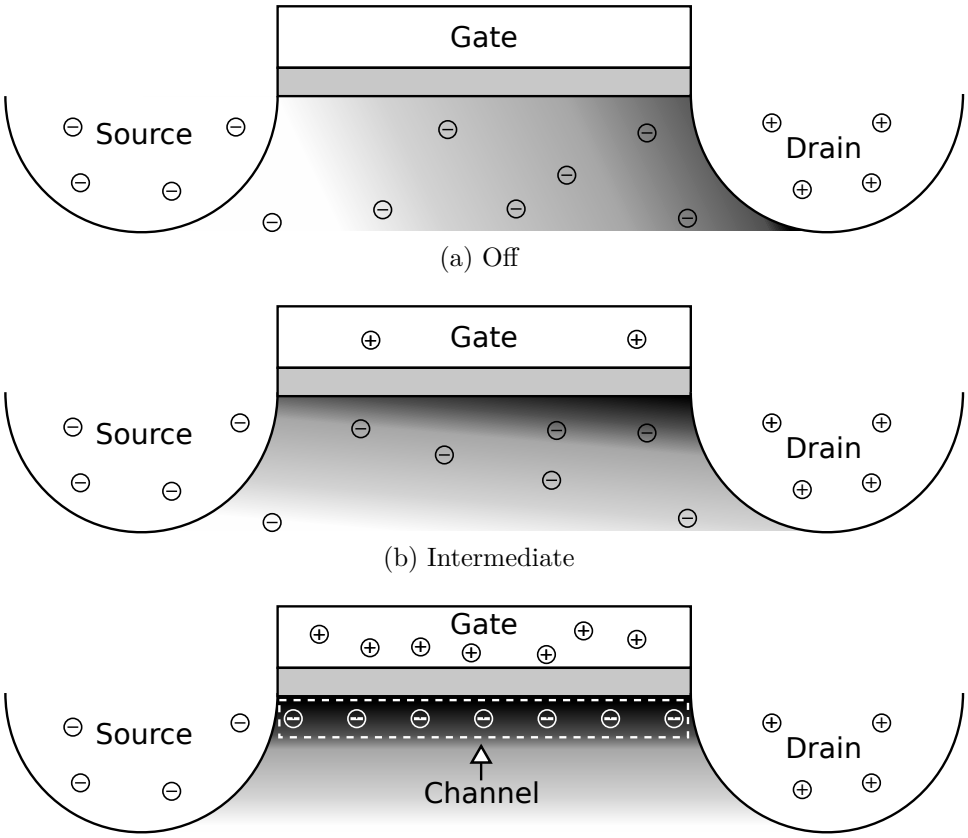
(a) Cross-section



(b) Top-down

Figure 2.2: The basic structure of a metal-oxide semiconductor field-effect transistor (MOSFET).

trols the rate at which current is allowed to flow between the "source" and the "drain". A capacitor serving as a "gate" applies an electric field to the channel to control the availability of charge carriers in the channel, and, thus, its resistance. The gate field pushes charge carriers away from the channel region to increase its resistance or pulls charge carriers into the channel to decrease its resistance (Figure 2.3), effectively disabling or enabling current transmission between the source and the drain. The time it takes to fully charge the gate capacitor is what causes the switching delay of the transistor.



(a) Off

(b) Intermediate

(c) On

Figure 2.3: A MOSFET in three states. (a) With a neutral voltage on the gate, the charge carriers in the body disperse, resulting in a high resistance between the source and drain. (b) As charge builds up in the gate it attracts carriers. (c) Increasing the density of carriers reduces resistance, establishing a channel of relatively good conductivity between the source and drain.

Transistor current and delay can be approximated using Equations (2.3) [59]:

$$I_{sat} = W v_{sat} C_{ox} \left(V_{gs} - V_{th} - \frac{V_{d,sat}}{2} \right)^{\gamma} \tag{2.3a}$$

$$I_{sub} = \frac{W}{L} \mu C_{ox} (n-1) (V_T)^2 e^{\frac{V_{gs}-V_{th}}{nV_T}} \left(1 - e^{-\frac{V_{ds}}{V_T}} \right) \tag{2.3b}$$

$$I_{on} = \begin{cases} I_{sat} \quad for \quad V_{ds} = V_{dd} \geq V_{th} \\ I_{sub} \quad for \quad V_{ds} = V_{dd} < V_{th} \end{cases} \tag{2.3c}$$

$$I_{off} = I_{sub} (V_{gs} = 0) = \frac{W}{L} \mu C_{ox} (n-1) (V_T)^2 e^{-\frac{V_{th}}{nV_T}} \left(1 - e^{-\frac{V_{ds}}{V_T}} \right) \tag{2.3d}$$

$$\tau_p = \frac{C V_{dd}}{I_{on}} \tag{2.3e}$$

These equations describe the current I_{on} that can flow between the source and drain when the voltage V_{ds} between these points (equal to the supply voltage V_{dd}) is above (I_{sat}) or below (I_{sub}) the threshold voltage V_{th} for the transistor. This threshold voltage is defined as the minimum voltage that establishes a conductive channel that spans the full length of the transistor (connecting the source and drain terminals). The delay τ_p is determined by the switched capacitance C, the supply voltage V_{dd}, and the current I_{on}. In these equations, V_{gs} is the gate-source voltage difference and $V_{d,sat}$ is the drain saturation voltage, above which increases in the drain-source voltage no longer affect the current flow.

From this model we can begin to see how variations arise. Topological deviations, such as errors in channel width W or length L, alter resistance. Variation in thickness of the oxide layer (the dielectric between the gate and the body forming a structure with capacitance C_{ox}) alter the strength of the field from the gate in the body, which can also alter V_{th} and the slope factor n. Likewise, a change in the number of charge carriers resulting from effects such as random dopant fluctuation (RDF) will make it easier or harder for the gate to induce a channel, which can alter both V_{th} and

the charge carrier mobility μ. Crystal defects, impurities, and local temperature variations can also affect these parameters, as well as changing the charge carrier saturation velocity v_{sat}, its associated constant γ, and the thermal voltage V_T. Finally, V_{th} can also be changed by the induction of a field in the body (typically from below) which counters or assists the field from the gate. When done deliberately, this last effect is called "body biasing" (as discussed in Section 3.2.2).

Fabrication variations can result not only in parametric deviations, but also in hard defects. For example, if the gate material extends past the oxide, it may short with one of the terminals. A loose particle might block dopant injection and then wash away before the next fabrication step. Errors in the substrate can alter the electrical properties in the body or induce gross geometric errors. In short, transistors require very fine control of the production environment and processes to achieve reliable outcomes.

2.3 What Has Changed?

Conventional semiconductor feature sizes cannot scale below the width of an atom (0.5 nm for a silicon lattice). However, long before this point, the discrete nature and statistical behavior of individual atoms may pose challenges to scaling. Traditional semiconductor doping depends on the statistics of large numbers of dopants to create consistent devices, but variation increases as device size, and hence nominal dopant count, decreases. Small features are more susceptible to movement or displacement of a few atoms and local variations in processing, such as etching and reaction rates.

As a result, we expect increasing parameter variation in devices (e.g., [37] (Table 18, Design chapter), [4, 7]). These variation effects are in addition to the traditionally increasing challenge of avoiding catastrophic photolithographic defects (e.g., [8]).

Year	2005	2006	2007	2008	2009	2010	2011	2012
DRAM $\frac{1}{2}$ Pitch (nm)	80	70	65	57	52	45	40	36
$3\sigma_{V_{dd}}$	10	10	10	10	10	10	10	10
$3\sigma_{V_{th}}$ RDF (min size)	24	29	31	35	40	40	40	58
$3\sigma_{V_{th}}$ total (min size)	26	29	33	37	42	42	42	58
$3\sigma_{V_{th}}$ total (typical logic)			16	18	20	20	20	26
3σ critical dimensions	10	10	12	12	12	12	12	12
3σ circuit performance	41	42	46	48	49	51	60	63

Year	2013	2014	2015	2016	2017	2018	2019	2020
DRAM $\frac{1}{2}$ Pitch (nm)	32	28	25	22.5	20	17.9	15.9	14.2
$3\sigma_{V_{dd}}$	10	10	10	10	10	10	10	10
$3\sigma_{V_{th}}$ RDF (min size)	58	81	81	81	81	112	112	112
$3\sigma_{V_{th}}$ total (min size)	58	81	81	81	81	112	112	112
$3\sigma_{V_{th}}$ total (typical logic)	26	36	36	36	50	50	50	50
3σ critical dimensions	12	12	12	12	12	12	12	12
3σ circuit performance	63	63	63	63	65	66	69	69

Table 2.1: As scaling continues, cost-effective management of process variation is expected to become increasingly difficult. Consequently, if Moore's Law is to continue, an increasing burden of variation tolerance is projected to be left to post-fabrication solutions.

Except for critical dimensions, all variability (σ) values are given as a percentage (%) of nominal. Topological variability is given as a percentage of the dimensions of a minimum-sized transistor as opposed to the design-nominal dimensions for each transistor. This data is compiled from the 2005, 2007, and 2009 ITRS reports.[37, 38, 39]

Dopant implantation is a Poisson process [78]. For large transistors with many dopants, the size of any variations will be negligible relative to the total dopant population. Thus, for older technologies the total number of dopants and their distribution in the body of a transistor are consistent enough that variation is not a concern. Newer technologies are a different story.

RDF is a hard problem to solve and the resulting variation persists through fabrication. Table 2.1 shows this trend in action. At 80 nm (where the ITRS began projecting these parameters), V_{th} variations from RDF are on the same scale as V_{dd}

(system supply voltage)[1] and topological variations. However, process improvements hold V_{dd} and topological variations steady while the impact of RDF grows. In the ITRS projections for 2011, dopant fluctuations are expected to account for half of performance variations [40].

RDF cannot be detected visually, thus visual inspection is insufficient to identify slow circuits. A slow circuit may not fail a slow signal propagation test, as the bits do get through eventually. It also will not trigger a quiescent current test and may even have favorable power characteristics. To catch an RDF-related parametric defect may require testing a circuit in operation at full speed. That means we may lose the cost savings of early detection of bad dies, assuming we can catch such defects at all.

These issues are not a surprise, as the emergence of RDF as a significant limiter was predicted almost half a century ago by Hoeneisen and Mead [33]. If anything, the surprise (at least from a 1971 perspective) is that reliable 14nm circuits (e.g., Intel's Broadwell processors) can be built at all.

2.4 Summary

Killer defects are getting smaller and harder to identify visually. We also must worry about invisible parametric defects that are even harder to identify. In short, "fabrication of chips with 100% working transistors and interconnects becomes prohibitively expensive" [39, Design]. While it is important to continue improving manufacturing, post-fabrication defect tolerance can reduce the burden of perfection, thereby reducing the cost of developing future technologies. Looking forward, these strategies may be key enablers for otherwise impractical or even impossible technologies.

[1] V_{dd} must be raised to compensate for V_{th} variation, thereby increasing the power consumption of an entire chip.

Chapter 3

Current Solutions

Defect tolerance and variation tolerance in field-programmable gate arrays (FPGAs) (and logic in general) have been studied for over a decade (several decades for general logic). Many solutions have been published to address various aspects of these problems.

There are several categories of variations, which are typically addressed with different solutions:

- Die-to-die variations and larger scale (e.g., wafer-to-wafer) variations have a uniform impact on a die. Techniques (including my own) that address within-die variations have little or no impact on these large scale variations.

- Systematic variations are consistent patterns of variation that result from specific design and manufacturing choices. For example, dies around the edge of a wafer may be slower than dies fabricated at the middle of the wafer. Systematic errors, once identified and understood, may be reduced by addressing the source of the variation. For FPGAs, systematic variations can also be addressed using the timing models available in CAD tools.

- Spatially-correlated variations are non-systematic variations that result from sources that affect areas larger than individual transistors and wires. For correlated variations, solutions may leverage statistical knowledge about the variation distribution to avoid multiple likely bad devices at once. For example, if one look-up table (LUT) is identified as particularly slow, a placer might avoid adjacent LUTs, expecting them to be slow as well. Random dopant fluctuation (RDF) is not thought to result in spatially-correlated variations.

- Random variations are the variations that are not systematic and are independent of the variations in nearby devices. Some of the most significant sources of random variation in current and projected future fabrication (such as RDF) operate independently on individual transistors. For FPGAs, random variations and, likewise, random defects, are an important motivation for component-specific mapping (CSM).

In addition to variations (which result in slow components, here referred to as "parametric defects"), my work, along with many of the techniques listed in this chapter, addresses the presence of completely non-functional components ("faults") that result from physical defects. "Fault tolerance" is often used to describe both faults resulting from persistent physical defects and transient faults.

Transient faults, such as single-event upsets (SEUs), do not correspond to physical defects and cannot be addressed by repairing or avoiding a device where a fault occurred. Hence, transient fault tolerance techniques focus on structures that resist disruption (e.g., radiation shielding), techniques to detect errors (e.g., parity checks), and techniques to recover from data corruption (e.g., error correcting codes). Triple modular redundancy (TMR) is the only technique discussed in this chapter that can be used to tolerate transient faults.

Techniques that target persistent faults focus on finding and eliminating defects. TMR can also be used to address persistent faults. My research addresses persistent faults from physical defects and parametric defects (i.e., variations), but not transient faults.

3.1 Chapter Organization

The existing solutions discussed in this chapter are organized into three categories:

1. Solutions that address low-amplitude variations only (Section 3.2).
2. Solutions that will not effectively solve large numbers of defects (Section 3.3).
3. Solutions that are effective for both variations and defects and are therefore more comparable to my work (Section 3.4).

3.2 Techniques That Address Variations Only

Much recent variation tolerance research focuses on techniques that do not migrate logic and routing from bad resources to better resources. These techniques are generally limited to reducing specific small margins, and do not address hard defects. Therefore, these techniques are not fully comparable to my work; however, each is described briefly below.

3.2.1 Statistical Static Timing Analysis (SSTA)

Statistical static timing analysis (SSTA) [85] uses statistical analysis to identify which nets are likely to break timing in the presence of variations. This analysis is used to reduce potential timing failures when a circuit is implemented on physical devices.

For FPGAs, a number of publications examine the value of adding SSTA to the clustering, placement, and routing stages of conventional FPGA CAD [49, 77, 52, 44]. According to Mehta [60], the most optimistic results so far predict a 12.6% improvement in timing yield (the fraction of chips that run correctly at a given frequency) for $3\sigma_{V_{th}}/\mu_{V_{th}} = 10\%$.

3.2.2 Body Biasing

Nabaa et al. [63] explored regional biasing in FPGAs that is similar to the approach proposed for processors in [80]. (Resource overhead makes body biasing too costly to apply to individual transistors.) Nabaa's approach characterizes each physical FPGA tile (a logic block and nearby routing resources) to calibrate for process variation. That calibration is combined with the slack of the logical circuit (the user design loaded onto the FPGA). Body biasing is applied to each tile to improve performance where necessary and to reduce power consumption in regions with extra slack. Because biasing is applied regionally, it is most effective for variations which benefit from uniform tuning (e.g., spatially-correlated variations). With an estimated 1.6% area overhead, Nabaa's simulations achieve a 30% reduction in $\sigma_{V_{th}}$ and a 78% reduction in σ_{power} for the modeled 130nm technology. However, with high degrees of random variation (e.g., RDF), as are seen in smaller-pitch technologies, the advantages of biasing are limited by the most extreme transistors — tuning for one transistor may cause another in the region to fail.

3.2.3 Clock Phase Skewing and Slack Stealing

When the clocked elements (e.g., latches) at the ends of a combinatorial chain are skewed (the clock triggers one slightly earlier than the other), the available period

for signal propagation changes. For example, if a sink latch is slightly early, the signal must propagate from the source to that sink in less than a full clock cycle. In certain circumstances such skews may be used to effectively increase the clock period for a particularly slow chain of logic without slowing down the rest of the chip [76]. Multiple clocks and local phase shifting can be used compensate for variations in logic and the clock distribution network [74]. However, this technique is limited not only to adjustments on the order of the clock period, but also to the available delay that can be shifted from one circuit to another. It cannot compensate for switches that are broken or that slow down the circuit beyond these limits. To address these more extreme cases, it is preferable to be able to avoid the bad switch entirely.

3.3 Techniques That Address Low Defect Rates

3.3.1 Multiple Bitstreams

Typically there are many ways to map a design to an FPGA. We can exploit this freedom to choose a mapping which avoids defective or undesirable devices. At a coarse granularity, we could place and route a design several times to produce multiple bitstreams. Then, when we program a specific FPGA, we can select (by trial and error if necessary) a bitstream from this library that avoids any defects present in that chip [84, 56, 75]. However, this technique does not scale well with increasing defect rates, potentially requiring an exponentially growing number of pre-generated bitstreams.

3.3.2 EasyPath™

The Xilinx EasyPath™ [86, 42] program is similar to the multiple-bitstream approach. Multiple bitstreams are tested on each FPGA to find one that works. However, for EasyPath™, multiple FPGAs are also tested for each bitstream to find FPGAs that match the needs of the customer. The purpose of this program is to reduce waste by finding applications for chips that fail quality testing and cannot be sold as fully functioning FPGAs. While EasyPath™ is successful at achieving this goal, it does not offer a general solution to the problem of mapping an arbitrary application to an arbitrary imperfect chip.

3.4 Techniques That Address High Defect Rates

The techniques discussed in this section attack similar aspects of the defect/variation problem to those addressed by my own work. Many are already used in common practice and show significant benefits; however, Choose-Your-own-Adventure (CYA) has certain distinct advantages relative to each.

3.4.1 Modular Redundancy

A seminal work on defect tolerance is John Von Neumann's "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components" [65], in which he introduced and analyzed the use of redundancy, particularly triple modular redundancy (TMR). TMR continues to be a standard defect tolerance solution, particularly in critical systems and in devices operating in harsh environments where single-event upsets (SEUs) are common.

For FPGAs, TMR is recommended to bolster robustness, particularly to protect against SEUs, a major concern in high radiation environments such as space applica-

tions. Documents from Xilinx [10] and Actel [2] highlight architectural features that aid in the integration of TMR into logical circuits. Pratt et al. [67] developed a tool to automatically add TMR selectively to those portions of a design that will most benefit from redundancy. At the physical circuit level, Actel produced radiation-hardened parts [18] that use TMR to protect particularly sensitive structures. At Xilinx, Trimberger [82] patented specific structures that may be used for TMR or to enhance functionality when redundancy is not needed.

TMR comes with high costs: a minimum of 200% area and power overhead are needed for almost all applications, and the inline voting process also adds delay. TMR also does not scale well with increasing defect rates — Von Neumann's analysis shows that overhead grows exponentially with the number of simultaneous failures the system is designed to tolerate. Consequently, more modern techniques generally prefer solutions with lower overhead and focus on addressing specific failure modalities rather than generic faults.

3.4.2 Hardware-Sparing In-Factory Repair

Another approach to defect tolerance is to perform the defect tolerance behind the scenes, always presenting the end system with the appearance of a perfect component.

Column sparing has long been used to improve yield for memories [21]. Post-fabrication testing identifies columns and rows that contain defects. A permanent alteration (e.g., blowing out a fuse) disables the offending region and remaps the array to fill the gap. The shipped memories appear defect-free to the consumer.

In FPGAs, Altera has employed this kind of behind-the-scenes sparing at the level of rows and columns to enhance yield and enable production of very high capacity components (e.g., [12, 13, 57, 47]). Yu and Lemieux show how to spare at the finer granularity of individual wire tracks [96, 95].

These implementations of sparing result in additional area and delay overhead. For an 80% yield target and an interconnect defect rate of around $2 \cdot 10^{-4}$ defects per wire segment, Yu and Lemieux estimate their design requires 50% area overhead, while the coarse-grained row sparing technique requires over 100% area overhead. By comparison, Figure 4.7 shows that a CYA solution attains the same yield at defect rates an order of magnitude higher, with only ∼20% area overhead. The interconnect defect density in Yu and Lemieux's work is approximately 100 defects/cm^2 in the 45nm technology they used, corresponding to ∼400 defects/cm^2 in my 22nm studies. Table 4.10 shows CYA producing 95% yields at defect densities an order of magnitude higher or more, in simulated chips which contain not only interconnect defects, but also logic defects.

The delay effects of CYA and sparing methods also differ. In both methods, the area overhead imposes an inherent delay burden, so CYA's reduced area offers a relative advantage here. Moreover, unlike sparing methods, CYA does not need to pay the delay costs associated with adding in-path bypass circuitry.

3.4.3 Avoiding Faults

The multiple-bitstreams solution (Section 3.3.1) does not repair defects, nor does it hide them. It prevents faults by using non-defective devices and avoiding defective devices.

HP's TERAMAC project applied component-specific mapping (CSM) to fault avoidance, using extensive testing and component-specific (knowledge-based) CAD. TERAMAC demonstrated the ability to effectively use chips with element defect rates of up to 3–10% [15]. Similarly, Katsuki et al. mapped out the delay of regions of a chip and used that map during placement to keep critical paths away from the slowest resources on the component [41]. One of the weaknesses of this approach is

the long runtime for conventional FPGA mapping tools. To accelerate the mapping process, TERAMAC employed a richer and more regular interconnect architecture than conventional FPGAs [3], trading density for mapping speed.

Local substitution of resources can potentially reduce or avoid the high cost of complete FPGA remapping at the expense of accepting less-optimal mappings. Lach introduced a design style for FPGAs that reserved one or more logic blocks in an $N \times N$ tile so that defects could be repaired with local sparing [45]. Lakamraju and Tessier note that reserving spare LUTs in an island-style cluster also allows local repair of logic [46].

Given the domain [89, 81] style of most FPGAs, one simple way to generate alternatives is to reassign entire domains (e.g., [97]). With a small number of defects, a comparable number of spare domains could be used to replace domains that contain defects with defect-free domains. Domain swapping is very coarse-grained and, similar to the multiple full-bitstreams solution, this large repair granularity does not take full advantage of inherent FPGA configurability.

Campergher et al. [9] and Trimberger [83] describe approaches most similar to my alternatives scheme, in which every path is covered by an alternate path. Although these concepts are clearly a good start, the published descriptions and characterizations are unfortunately extremely limited, making it difficult to perform qualitative or quantitative comparisons. The Trimberger methodology, in particular, is comparable to CYA with only 1 reserved track and 1 alternative. As Figures 4.7 and 4.11 suggest, this limited pool of reserve resources and alternatives leaves significant room for improvement. In addition, as I show in Chapters 5 and 6, CYA provides a practical, customizable framework for more complex forms of variation tolerance, such as parametric repair and delay and energy optimization.

Chapter 4

CYA

This chapter is an introduction to the Choose-Your-own-Adventure (CYA) approach to defect tolerance. I begin with motivation and an intuition-building example (Section 4.1) to demonstrate the key components of CYA. Section 4.2 dives into the details of the basic CYA implementation. Section 4.3 identifies a weakness in the initial algorithm used to pre-compute repair solutions ("alternatives") and provides an improved algorithm. Section 4.4 sets up experimental methodologies to perform an initial assessment of CYA. The results of these assessments and a full analysis are presented in Section 4.5. Section 4.6 looks at the costs of using CYA in terms of the size of the bitstream and estimated load time, and introduces the updated VPR 5-based FPGA model which will used throughout the remainder of the dissertation. Finally, Section 4.7 explores the yield impacts of defects in the various reconfigurable resource classes in an FPGA and adopts a more physically realistic (per-area rather than per-resource) defect density model to elucidate their combined effect.

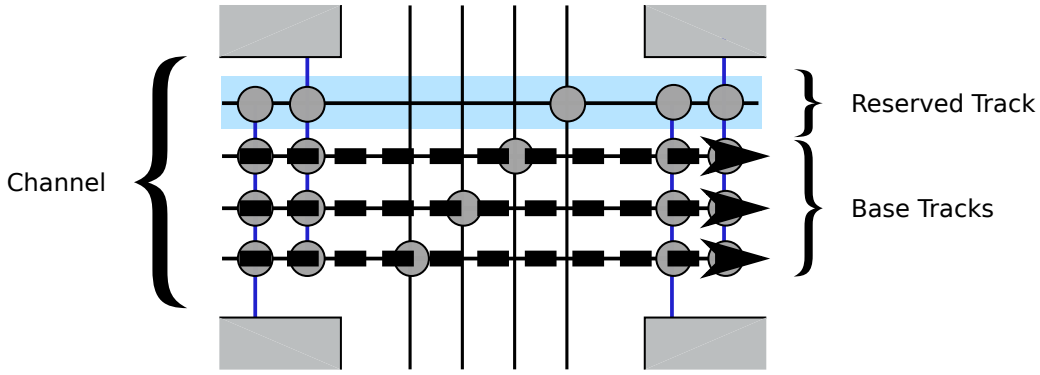


Figure 4.1: Channel with four tracks, one of which is reserved. Three nets saturate the three base tracks.

4.1 Inspiration

The name Choose-Your-own-Adventure is an homage to the eponymous Choose Your Own Adventure novels [66]. Those books are structured around user decisions. This device enables a single immutable hardcopy book to provide many variants of a story, tailoring the plot, on each read-through, to the whims of the reader. Likewise, in Choose-Your-own-Adventure bootstrapping, a single immutable field-programmable gate array (FPGA) bitstream provides choices which result in loaded configurations tailored to the peculiarities of each individual consumer FPGA.

4.1.1 Illustrative Example

Consider a simple FPGA channel (Figure 4.1). We have four tracks and need to route three nets through this channel. Labeling the fourth track as "off limits" (reserved), we route the channel, packing the nets into the first three tracks (base tracks). Then, for each net, we find an alternate path using the fourth track. When programming the FPGA, the loader tries the default route, and, if it is bad, the loader tries the alternate path. If the chip has at most one defect in the channel (or defects in at

most one track), the defect(s) will either be in the fourth track and will not disturb the default routes, or it/they will upset a single route, forcing that route to use the alternate path on the fourth track.

We can expand the notion of tracks to domains. By domains, I refer to architectures where the interconnect network is partitioned into independent sets of routing resources (e.g., [89]). With domains, the initial track choice leaving a source block determines the track that will be used when entering the destination block. The independence property of domains means that when we reserve a domain, we do not affect the rest of the network. Resources that are part of the reserved domain are not available to the base domains and *vice versa*. Now we can look at a whole chip and say that default paths are routed in the base domains and alternatives are routed on the reserved domain and unused portions of the base domains. This allows us to guarantee that an FPGA with a single defect anywhere in the interconnect has an unused alternate route that will avoid the defect.

If we want to provide resilience to any two defects we can add a second domain reserved for alternatives and prepare alternative paths using each domain. We can add additional domains to tolerate more defects. However, we should not need a full alternate domain to tolerate each single defect. In normal FPGA routing, we are able to both find multiple paths between a source and sink in a single domain and route multiple two-point nets in a single domain. Consequently, each additional domain should allow us to tolerate multiple additional defects.

4.2 CYA Components

I now describe CYA bitstream composition, generation, and loading. Figure 4.2 (repeating Figure 1.4) provides a roadmap for this discussion.

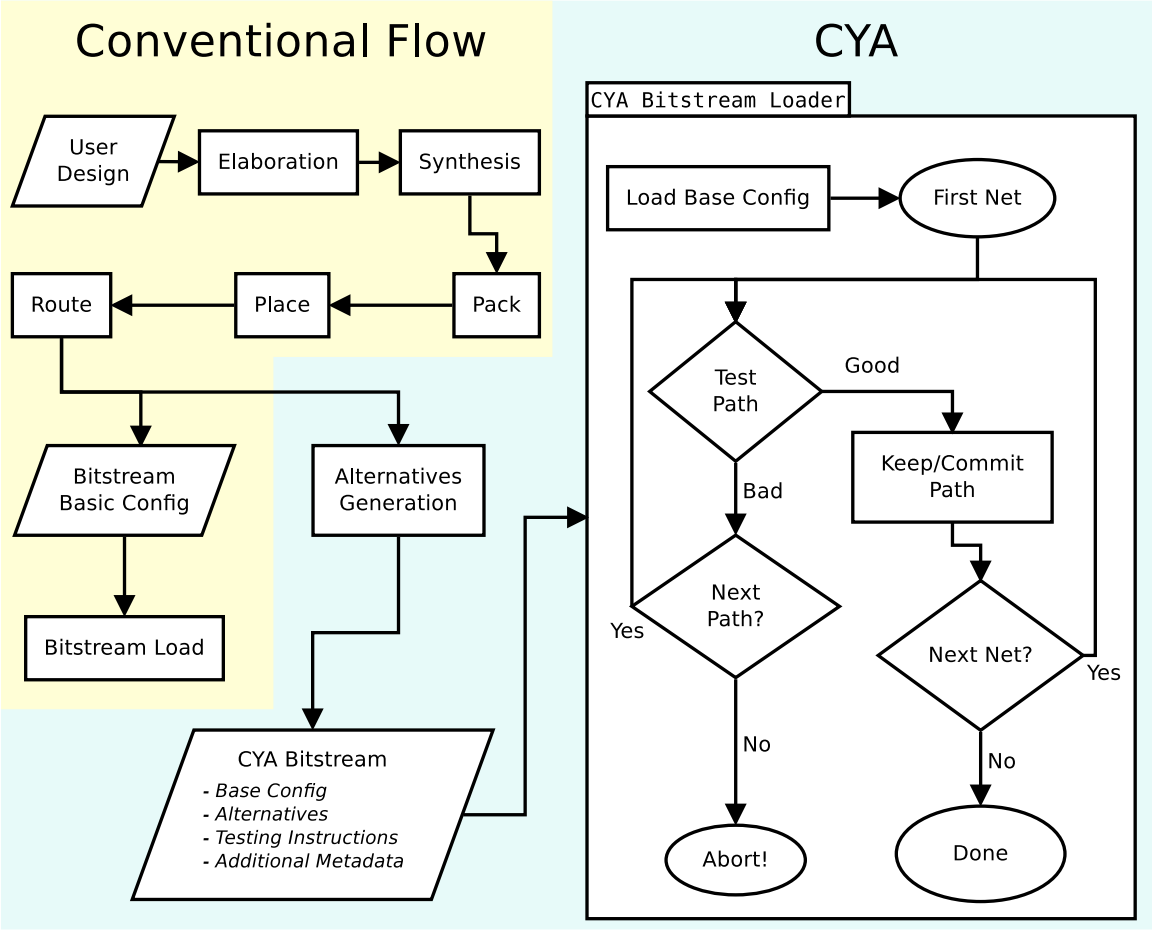
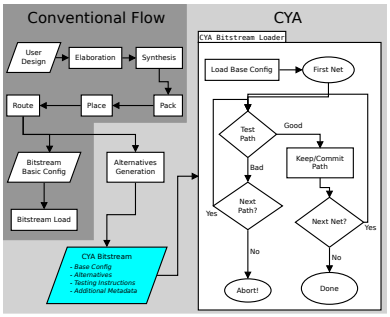


Figure 4.2: CYA computer-aided design (CAD) flow

4.2.1 CYA Bitstream

The CYA bitstream is a list of two-point nets each with:



- a base path configuration
- a list of alternatives
- a set of testing instructions
- optional metadata such as the delay budget (used in Chapters 5 and 6)

This means that bitstream programming is ordered by logical functions rather than by physical location on the FPGA. For fabrics where the configuration bits are not randomly addressable, such as Virtex frames [91, 93], I assume the presence of a translator (see Section 4.2.4) that can add or remove the portion of the input path that applies to the configured frame.

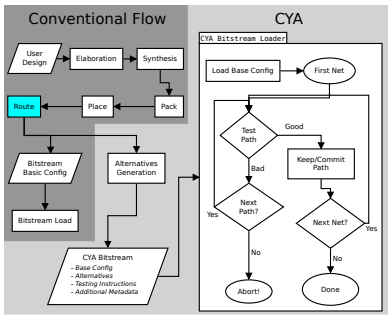
The *base route* is a normal FPGA route prepared using only base tracks.

An *alternative* is a different path from the base route that can be used for defect avoidance. It consists of a set of resources that connects a single source to a single sink (i.e., a two-point net) that differs by at least one resource from the corresponding path found in the base route. Alternative paths may use both the reserved spare resources and any non-reserved resources that were left unused by the base route, and are forbidden from using any resources which are part of the base paths of other nets. This guarantees that we are free to switch from a base path to any of its alternative paths without interfering with any of the base routes.

To detect defective paths, the loader must check the functionality of each path. For a two-point net, the test must simply check that a high-to-low and a low-to-high transition of the source are each correctly observed at the destination. The information needed for this test can be inferred from the path definitions. To generalize

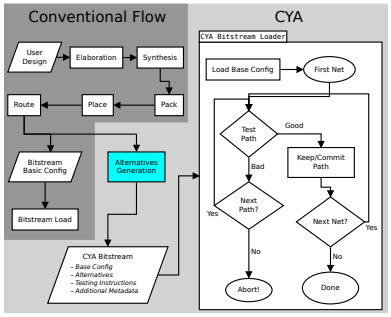
testing for more complex cases (e.g., look-up table (LUT) and Flip-Flop testing, or testing for bridging) and to simplify the design of the loader, I embed *test instructions* in the bitstream. In the simplest scheme described here, multi-point nets are handled by checking each two-point net independently.

4.2.2 Routing



The base route can be generated with a standard FPGA router such as Pathfinder [58]. The only difference is that we may choose to reserve some resources solely for use in the preparation of alternate paths. The router must therefore be capable of acknowledging these resources as being "off limits" to the base route.

4.2.3 Alternatives Generation



Alternatives are paths that must not conflict with the base route. We want a diverse (see Section 4.3) set of alternatives for each base path, to maximize the opportunity for defect avoidance. To encourage this diversity, I associate a cost with resource usage similar to Pathfinder resource costs. This allows me to use repeated calls to a shortest path search to find different routes as long as I update the cost of each resource in the graph as it is used. This *Resource-Cost Algorithm* is shown in Algorithm 4.1. Pathfinder-based routers can easily be modified to add this functionality. During alternative generation, the cost for each resource

is: $cost = alternatives\_using + 1$. In Section 4.3 I note the shortcomings of the Resource-Cost Algorithm and introduce a more sophisticated *Path-Cost Algorithm*.

Algorithm 4.1: Resource-Cost Algorithm for alternatives generation

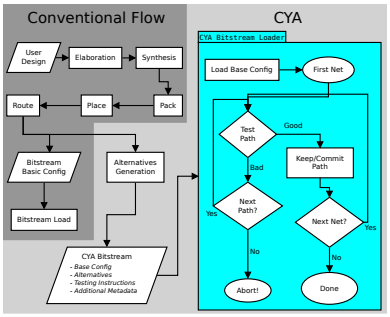
```
input  : Route base_route
output: Bitsteam C

foreach Resource R ∈ {base_route} do Disable R ;

foreach Net N ∈ {base_route.nets} do
    foreach GraphNode sink ∈ N.sinks do
        C.addPath(N, N.base_path(sink));
        foreach Resource R ∈ {all Resources} do
            R.alternatives_using = 0;
        foreach Resource R ∈ {N.base_path(sink)} do
            Enable R;
        for i=1 to number of alternatives do
            Path P ← FindShortest(N.source,sink);
            C.addAlternative(P);
            foreach Resource R ∈ P do
                R.alternatives_using + +;
        foreach Resource R ∈ {N.base_path} do
            Disable R;
return C;
```

4.2.4 Bitstream Loader

I decompose the loader into four components, a programmer, a deprogrammer, a tester, and a controller.



Programmer

By the term "programmer" I refer to that element of any FPGA bitstream loader that sets the configuration bits. In the simplest case, we might have random access to the configuration bits (c.f. Xilinx 6200 [90]). This makes it fast and easy to set each bit but demands greater area overhead for configuration support than conventional configuration chains.

At the cost of more time and work loading the bitstream, we can exploit the frame schemes that exist in modern FPGAs (e.g., [91, 93]). Specifically, we could organize the bitstream to specify the frame address and the address of bits to change within the frame. Then, the loader can: (1) read out the old frame, (2) change the specified bits in the frame, and (3) load the modified frame. This is the same kind of operation used by Xilinx J-bits to perform bitstream modification on Virtex series components [28]. Such a scheme would require no changes to the core of the FPGA. The cost is longer load times, as we must spend an entire frame read/write sequence for every frame touched by an alternative (see Section 4.6.2 and Tables 4.4 and 4.5).

Deprogrammer

When a path fails, the loader must undo the configuration changes to release resources for other paths. Functionally, the deprogrammer must roll back the configuration to its state before the last path was added. One way to accomplish this is to record changes made during programming so that they can be reverted; this has the advantage of demanding no semantic understanding of the bitstream, but requires space

to store the changes. An alternate version might use the same path specification for programming with the configuration sense reversed.

Tester

The tester is responsible for testing each path and reporting the success or failure of the test. The bitstream loader only needs to know if the end-to-end path test fails. The alternatives encoded in the bitstream directly tell the loader what to try next when a test fails. If the bitstream loader does not have random access into the bitstream, the loader will need adequate local space to store the current test specification to be used with the sequence of alternatives.

One simple way to support testing is to drive and recover data using the internal clustered logic block (CLB) flip-flops. These flip-flops can still be used for observability even if they are not used in the design. In some cases, we might reconfigure the CLB logic to facilitate testing. For example, the source CLB would be configured to drive the path under test from a flip-flop and configure the destination LUT as a buffer with its flip-flop enabled. It may be possible to set up the tests using bitstream configuration, trigger transition tests using readback capture, and then view the results using configuration and state readback [92, 93]. End-to-end connectivity tests check that we can see both driven zeros and driven ones at the destination.

Timing tests can be performed using a variant of "launch-from-capture" transition fault testing (e.g., [73, 86]). This simple change from "does the signal get through" to "does the signal arrive on time" can be combined with an outer loop to optimize overall delay, as described in Sections 5.2 and 6.1.

To expand the class of resources covered by this approach, LUTs/subblocks can be tested with conventional procedures (e.g., test patterns) before this path testing. Additionally, it may be necessary to swap LUTs at the start or end of a path before

performing the path test.

Controller

The controller coordinates the other units to implement Algorithms 4.2 and 4.3. It is a very straightforward entity that makes no complex decisions and performs no complex actions. Notably, the controller does not need to understand the FPGA architecture or the semantics of the configuration bits. All the intelligence about the meaning of the bitstream is effectively compiled into the bitstream. The controller only needs to mechanically follow the bitstream load program. With suitable test support, the embedded PowerPC on Virtex devices could be used to run this algorithm, using the internal configuration access port (ICAP) [93] to perform the configuration.

Algorithm 4.2: Bitstream Load Algorithm

```
input  : Bitstream B
output: Boolean configured?

foreach 2PT_Net N ∈ {B} do
    found ← FALSE;
    while (not found) do
        if N.outOfPaths then
            return failure;
        P ← N.nextPath;
        if P.isUsable(P) then
            Program(P);
            if Test(P) then
                found ← TRUE;
            else
                DeProgram(P);
return success;
```

Algorithm 4.3: isUsable Function for Bitstream Load Algorithm

```
input  : Path P
output: Boolean usable

/* First free up the LUT, if required and possible                */
if inUse(P.node) then
    if canMove(currentOccupant(P.node)) then
        move(currentOccupant(P.node));
    else
        return FALSE;

/* To utilize fanout, two-point nets may share common prefixes    */
while P.hasMore and P.next.matches(Current Configuration) do
    P.step;

while P.hasMore do
    if inUse(P.next.node) then
        return FALSE;
    P.step;
return TRUE;
```

4.3 Alternative Diversity

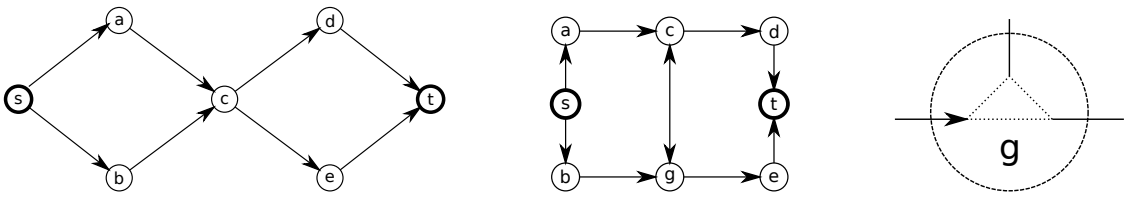
In this section, I identify weaknesses of the Resource-Cost Algorithm (Section 4.2.3) and develop the alternative Path-Cost Algorithm.

4.3.1 Problems with Resource-Cost Alternatives Generation

Figure 4.3 shows two example graphs. In each graph, we suppose that we want to connect the net $s \rightarrow t$. These graphs highlight certain cases where the Resource-Cost Algorithm is incapable of locating all the useful alternatives.

Uniform Paths

In Figure 4.3a, the paths from s to t all have equivalent costs. The router will first identify the path $s \rightarrow a \rightarrow c \rightarrow d \rightarrow t$. On the second pass the nodes used by the



(a) Uniform length paths (b) Longer paths (c) Inside node "g"

Figure 4.3: In each of the above examples, the Resource-Cost alternatives generation algorithm fails to locate all useful alternatives. The improved Path-Cost algorithm is capable of discovering the missed paths.

first pass will all be more expensive. So the partial path $s \to c$ will use b instead of a. Since $c \to e \to t$ is now cheaper than $c \to d \to t$, the second half of the route will use e resulting in $s \to b \to c \to e \to t$. After the second pass the new costs for b and e will be raised and will now match a and d. After that point, since the costs are once again balanced there is nothing to drive decisions on the third pass that are any different from the first, leaving us in a loop that will only find those two paths over and over. However, two paths remain ($s \to a \to c \to e \to t$ and $s \to b \to c \to d \to t$) that are capable of avoiding two defect combinations not covered by the first two paths: $\{b, d\}$ and $\{a, e\}$.

Non-Uniform Paths

Non-minimal length paths can also be valuable. A minor slowdown on nets off the critical paths does not impact performance, and, even for critical paths, a small performance degradation may be preferable to failure. The Resource-Cost Algorithm does not prohibit non-minimum delay paths, but it often fails to find them.

In the second example (Figure 4.3b) we are still trying to route $s \to t$. Again the Resource-Cost Algorithm will find paths along the top ($s \to a \to c \to d \to t$) and bottom ($s \to b \to g \to e \to t$) after which it will find no new unique paths. Any other paths will actually be longer. The paths that cross over in the middle, as in

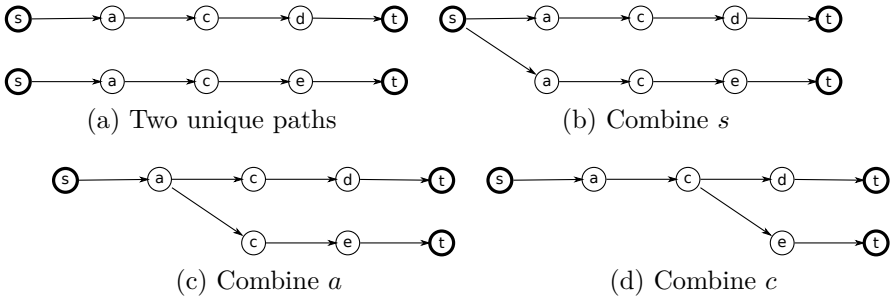


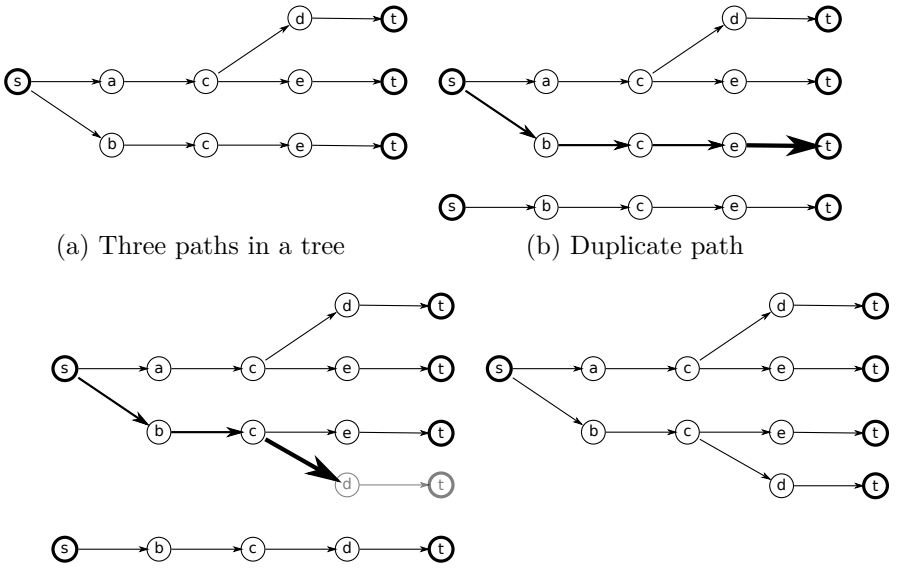
Figure 4.4: The merger of two paths into a tree structure.

the previous example, now must use an extra edge, $c \leftrightarrow g$.

Furthermore, some switch configurations internal to a node (e.g., Figure 4.3c) may have alternate configurations where passing through multiple switches (e.g., the two upper sides of the triangle in Figure 4.3c) can route around a defect in a single direct switch (e.g., the path along the base of the triangle). If we have a partial path $b \rightarrow g \rightarrow e$, the path that skips around the direct switch in g still uses the wires $b \rightarrow g$ and $g \rightarrow e$. When the cost structures do not include a switch congestion factor, there is no incentive to identify the two-switch path that avoids the direct switch.

Paths Tree

A paths tree is a structure that merges the prefix subpaths of multiple paths. Figure 4.4 shows the combination of two paths to form a simple paths tree. Node by node, as long as a new path follows a path already expressed by the tree, nodes are absorbed. When the path diverges from the pre-existing path, it forms a new branch in the tree. Duplicate detection can use the same process, declaring a duplicate if the new path never diverges from the paths tree (Figure 4.5b).



(a) Three paths in a tree

(b) Duplicate path

(c) New path

(d) Full paths tree for the graph in Figure 4.3a

Figure 4.5: Checking paths against a paths tree to identify duplicates and add new paths.

Path-Cost Aware Routing

The above observations of failures in the Resource-Cost Algorithm led me to develop a substitute alternatives generation algorithm that is capable of: (1) assigning costs to paths in additions to resources, (2) crossing between previously discovered paths even when the crossing is not free, and (3) diverting from and returning to an existing path as long as the detour avoids some resource. Furthermore, it is useful to (4) avoid generating duplicate paths to reduce router runtime and (5) require minimal modifications to the router to minimize the cost of adoption. I achieve these goals by using the paths tree to perform early duplicate detection and encourage greater diversity, as shown in Algorithms 4.4 and 4.5.

Consider once again the search for a third path in Figure 4.3a. The weights are still symmetric at s, so the expansion will still get to c through a. While the weights of d and e are balanced, the router recognizes $s \rightarrow a \rightarrow c \rightarrow d$ in the path tree

Algorithm 4.4: Path-Cost Algorithm for alternatives generation

```
input  : Route base_route
output: Bitsteam C

foreach Resource R ∈ {base_route} do Disable R ;
foreach Net N ∈ {base_route.nets} do
    foreach GraphNode sink ∈ N.sinks do
        foreach Resource R ∈ {all resources} do R.alternatives_using = 0 ;
        foreach Resource R ∈ {N.base_path(sink)} do Enable R ;
        TreeNode paths_tree ← new TreeNode(base_path);
        failure_count ← 0;
        pf ← path_factor_base;
        for i = 1 to number of alternatives && failure_count < fail limit do
            Path P ← PathCost.FindShortest(N.source,sink,paths_tree,pf);
            if P = Fail then
                failure_count++;
                pf = pf × path_factor_scale;
            else
                foreach Resource R ∈ P do R.alternatives_using++ ;
                paths_tree.add(P);
        C.addNet(paths_tree);
        foreach Resource R ∈ {N.base_path} do Disable R ;
return C;
```

and charges it a corresponding extra cost. $s \to a \to c \to e$ is not in the path tree prefix and consequently does not pay this cost. In this example, any extra cost from duplicated path prefixes breaks the tie and will result in the discovery of all four paths for this graph.

If we make the cost of continuing along an old path sufficiently high, the costs at c during the search for a third path in Figure 4.3b will be sufficiently unbalanced that the path will be forced to use the $c \to g$ crossover segment, even though g has already been reached by a shorter path via b. To support this behavior, Algorithm 4.5 uses a priority queue of paths to expand and allows nodes to be visited multiple times with different path prefixes. To drive the router to identify full paths to the sink rather

Algorithm 4.5: PathCost.FindShortest Function for Path-Cost Algorithm

```
input  : GraphNode source
input  : GraphNode sink
input  : TreeNode paths_tree
input  : double path_factor
output: a path

PriorityQueue Queue ← new PriorityQueue;
Queue.enqueue(source, paths_tree.root, 0);
while !Queue.empty do
    QueueEntry h ← Queue.dequeue();
    GraphNode g ← h.resource;
    TreeNode t ← h.tree_pointer;
    if g = sink then                                   /* Found the sink */
        if t = unique then
            Return(backTrace(h));                      /* Path is unique */
    else
        foreach GraphNode n ∈ {g.Neighbors} do                 /* Expand */
            TreeNode m;
            if (t.node = g) and [∃m | ((m ∈ {t.Children}) and (m.node = n))]
            then
                m ← matching child;
                        /* m.usage counts alternatives using node m */
                path_cost ← m.usage * path_factor;
            else
                m ← unique;                            /* Path is unique */
                path_cost ← 1;
            cost ← h.cost + pathfinder_cost(g, n) * path_cost;
            if cost < n.old_cost then
                Enqueue(n, m, cost);
                n.old_cost ← cost;
return Fail;
```

than generating all unique prefixes first, I use A^* routing techniques (e.g., [81]) that include an estimate of the remaining cost to the destination in the priority queue ordering function.

The Path-Cost Algorithm identifies more unique alternatives, on average, than the Resource-Cost Algorithm (see [70]). For many nets where the Resource-Cost Algorithm found fewer than 10 unique alternatives, the Path-Cost Algorithm successfully provided the 40 requested alternatives. Furthermore, the Path-Cost Algorithm manages to find better paths and place those paths earlier in the search order than the Resource-Cost Algorithm does. Figure 4.8 shows the impact on defect tolerance of using each alternative-generation algorithm to produce only 5 alternatives. The nearly four orders of magnitude improvement in defect tolerance with the Path-Cost Algorithm demonstrates that its 5 alternatives were significantly more effective than the 5 found by the Resource-Cost Algorithm.

4.4 Foundational Experiments

I designed a set of foundational experiments to characterize several aspects of the CYA approach:

1. How much yield improvement can we get from CYA at each defect rate, and over what range of defect rates is it effective?

2. How many alternatives does CYA need to store? How does this impact bitstream size and load time?

3. How does CYA yield improve with dedicated *reserved* tracks available only for alternatives?

4. How do *extra* base tracks beyond the minimum number required for the design to be routable impact CYA yield?

5. How should additional tracks be partitioned between extra and reserved tracks?

4.4.1 Experimental Framework

Defect Map

I want to characterize the likelihood that a given chip can be made to function correctly in the presence of a given level of initial fabrication defects (the yield). To do this I need to be able to model defects and vary the defect rate. If I simply varied the defect rate and generated independent sets of defects for each experiment, it would: (a) make experiments non-repeatable, (b) prevent direct comparisons between techniques or options (because experiments never see the same set of defects), and (c) create occasional anomalies where an experiment at a higher defect rate outperforms an experiment at a lower defect rate (due to less favorable location of defects in the lower defect rate experiment). To provide clean experiments and avoid these pitfalls, I generate a collection of partially defective virtual "chips" (defect maps) with a tunable defect rate that I reuse across experiments. Specifically, I assign an independent, uniformly distributed, random value to each resource in the chip. Then I apply the target defect rate as a threshold to differentiate good resources from bad. This guarantees that the number of defects in each chip monotonically increases with increasing defect rates. That is, as I apply higher and higher defect rates, this process will simply add more and more defects to the existing set without removing any of the initial defects, making the results at each defect rate easily comparable to each other.

With the exception of Figure 4.14, all figures and tables model both stuck-open

switch and broken-wire defects. In most of the results presented in this chapter, defects in all types of resources share the same per-resource defect rate. However, all figures from Figure 4.26 forward, and all tables from Table 4.11 forward, instead use a constant per-area defect density, reflecting an improved defect model developed for later experiments (see Section 4.7.7).

Simulators

Initially, VPR 4.3 [5] provided the scaffolding for my CYA simulator. I implemented each of the components in Section 4.2 using or modifying existing portions of the VPR 4.3 router. After obtaining promising preliminary results, the CYA enhancements were ported to VPR 5 [53] to add simulation of single-driver interconnect. Variation, delay, and energy modeling based on SPICE [64] (specifically the HSPICE [35] implementation) were also added in the VPR 5 version. (See Section 4.6.3 for additional details.)

Since the simulator has the advantage of global knowledge, the simulator does testing before programming. This does not alter the semantics at a functional level but it did obviate the necessity of simulating the deprogrammer.

VPR (both 4 and 5) uses a simplified CLB representation which does not model the internal structure of each CLB in detail — each net entering a given CLB simply proceeds directly from its corresponding input pin to a common, abstract "sink". (This also implies a fully populated input crossbar.) On the output side of the CLB, there is no crossbar — each subblock is directly tied to a single output pin. Consequently, LUT selection (and in some cases, duplication) is determined by the net emanating from that logic element (LE), not those feeding into it.

4.4.2 Experimental Flow

Placement

I prepare my placements using the unaltered VPR placer, with no extra options. A single placement is generated for each benchmark before all other work. This fixed placement is used for all subsequent operations.

Minimum Channel Width

For experiments where channel width (the number of tracks per channel), W, is a controlled parameter, I needed to find the minimum routable channel width (W_{min}). For this purpose I used VPR with the "`-verify_binary_search`" option. All other options remained at their default settings. Note that this means that the router uses a timing-driven A^* path search.

Extra Base Tracks

Designs are seldom mapped to FPGAs at their absolute minimum channel width. Typically some "extra" tracks are available, making it both easier to find a route (e.g., Swarz [79]) and possible to find faster, more direct routes (e.g., Marquardt [54]). Following the conclusions of these studies, I allocate extra tracks (e.g., $0.2W_{min}$ additional tracks) when preparing the base route. I examine the impact of provisioning extra tracks in Section 4.5.3.

Base Route

To generate the base route, I use VPR's fixed-width router configured with the option "`-max_router_iterations 100`". I route each design using a target channel width determined by the minimum channel width and the number of extra tracks (e.g.,

$W = 1.2W_{min}$ when I allocate 20% extra tracks). I also specify the number of additional reserved tracks, which will be used only during alternative generation.

Generating Alternatives

The base route is passed to the alternatives generator. Initially, all resources are available to the alternative paths, except those belonging to the base route, which are marked as "off limits". The block I/O pins and other resources belonging to a particular base path are also marked as available during the generation of its own alternatives, but they remain off limits to the alternatives associated with any other base path. I generate 40 alternatives for each path. I chose this number to be sufficiently large that this step need be performed only once for each base route. Loading a bitstream with these alternatives onto each defect map will then allow us to determine how many alternatives are needed to successfully load the design, as a function of the defect rate.

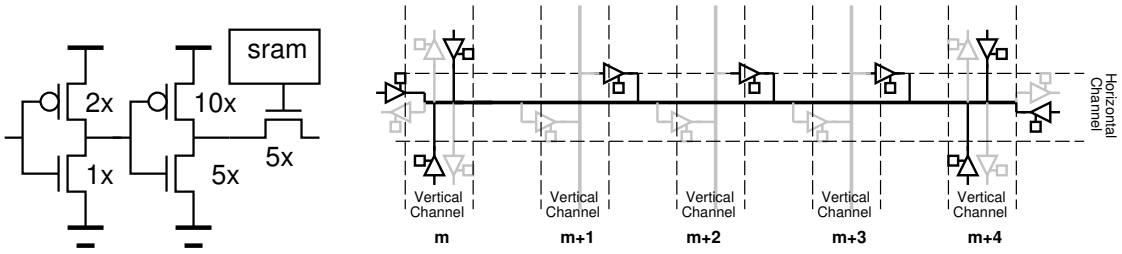
Loading the Bitstream

Once I have generated a defect map, a base route, and a set of alternatives, I can simply feed them back into a bitstream loader simulating Algorithm 4.2, which I added to VPR.

4.4.3 Experimental Architecture

The architecture used for the VPR 4.3 experiments is based on the example architecture file "4x4lut_sanitized.arch" that is included with the stock VPR 4.3 distribution. (Architecture changes made for my VPR 5 experiments are discussed in Section 4.6.3.) The key parameters I kept from that architecture are:

- "subset" switch boxes (S-Boxes)



(a) Buffered switch circuit. (b) Length 4 horizontal route segment (drivers and wire for this segment shown in bold).

Figure 4.6: Buffered switch and wire segment model.

- uniform segment length of 4 ($L_{seg} = 4$)
- 4 LUTs per CLB ($n = 4$, $O = 4$)
- 4 inputs per LUT ($k = 4$)
- 10 input pins ($I = 10$)

To better fit the uniform defect rate model, I use a single buffered switch type (Figure 4.6a) for all S-Box switches, instead of the mix of buffered and unbuffered switch types found in the example architecture.

The default staggering pattern assigned segment offsets in groups that are a fraction of the total channel width. Adding more tracks at the edge of the channel alters the distribution of staggers for the base tracks, resulting in unbalanced alterations to the interconnect topology. To avoid this, my architecture cycles staggers for each track. This maintains fixed connectivity for the existing tracks as I add more tracks to the channels.

Finally, for connection box (C-Box) population, I present results with fully populated C-Boxes ($F_{cin} = 1$ and $F_{cout} = 1$) as well as $F_{cin} = 0.50$ and $F_{cout} = 0.25$, as found in the original architecture files distributed with VPR. Most VPR 4.3 results use fully populated C-Boxes; for these experiments, C-Boxes should be assumed to

be fully populated except when explicitly stated otherwise.

The complete architecture for the fully populated C-Boxes is provided in Appendix B.1, and the depopulated architecture is provided in Appendix B.2.

4.4.4 Experimental Design

I performed all of my tests on the Toronto 20 designs [6], a set of benchmarks commonly used in FPGA research. Each simulation collected the following data: (1) the yield of functioning devices at each defect level, (2) the number of alternatives needed to produce a functioning device, (3) the critical path delay of the repaired logic (Section 4.5.4), and (4) the total path length of all the needed alternatives, used to estimate bitstream costs (Section 4.6).

Data was collected from 20–10000 (depending on the experiment) independently generated defect maps (Section 4.4.1), each of which was used for multiple defect rates ranging up to 10%. For each algorithm and parameter set (number of extra and reserved tracks, number of alternatives, etc.), I estimated the yield probability at that defect rate as the fraction of the defect maps that could be repaired. As with any statistical experiment, the size of the experimental sample set determines the likely error in my estimated results. In particular, it is useful to understand how high of a failure rate can still result in 100% measured yield amongst a certain number of samples. For example, an experiment with a sample size of 20 chips must have a true per-sample success rate of 0.997 or higher to achieve perfect yields 95% of the time ($0.95^{1/20} \approx 0.997$, assuming a binomial success/failure distribution). Table 4.1 summarizes the 95% confidence intervals that can be inferred from full-yield results at all sample sizes used in this dissertation.

Sample size	Full yield 95% confidence interval
20 chips	≥ 0.997439
100 chips	≥ 0.999487
500 chips	≥ 0.999897
1000 chips	≥ 0.999949
10000 chips	≥ 0.999995

Table 4.1: 95% confidence intervals that can be inferred from full-yield results at various chip sample sizes (under the assumption of binomially distributed yield success/failure).

4.5 Initial Results

Figure 4.7 illustrates the yield benefits of CYA. For clarity, this figure, like many figures in this dissertation, shows representative results from just one of the Toronto 20 benchmarks, `des`, rather than for all 20 designs. With 20% extra base tracks and an additional 20% reserved tracks, `des` maintains essentially 100% yield at defect rates five orders of magnitude above the point where defective chips actually begin to appear in our 100-chip sample (shown in the "Perfect Yield" curve). Even with only a single alternative, CYA achieves near 100% yield[2] for defect rates *200 times higher* than the point where one achieves high design-specific yield (i.e., alternatives=0 case; this roughly corresponds to EasyPath™ [42]). The figure further shows how the benefits vary with the number of available alternatives. Note that 40 alternatives provide only a slight improvement in yield over the case of 30 alternatives. This trend suggests that the addition of further alternatives beyond this point will offer little benefit for designs at these scales.

[2]For these experiments, "near 100% yield" implies no failures occurred across all chips in the sample. The experiments in this section employed 100 simulated chips.

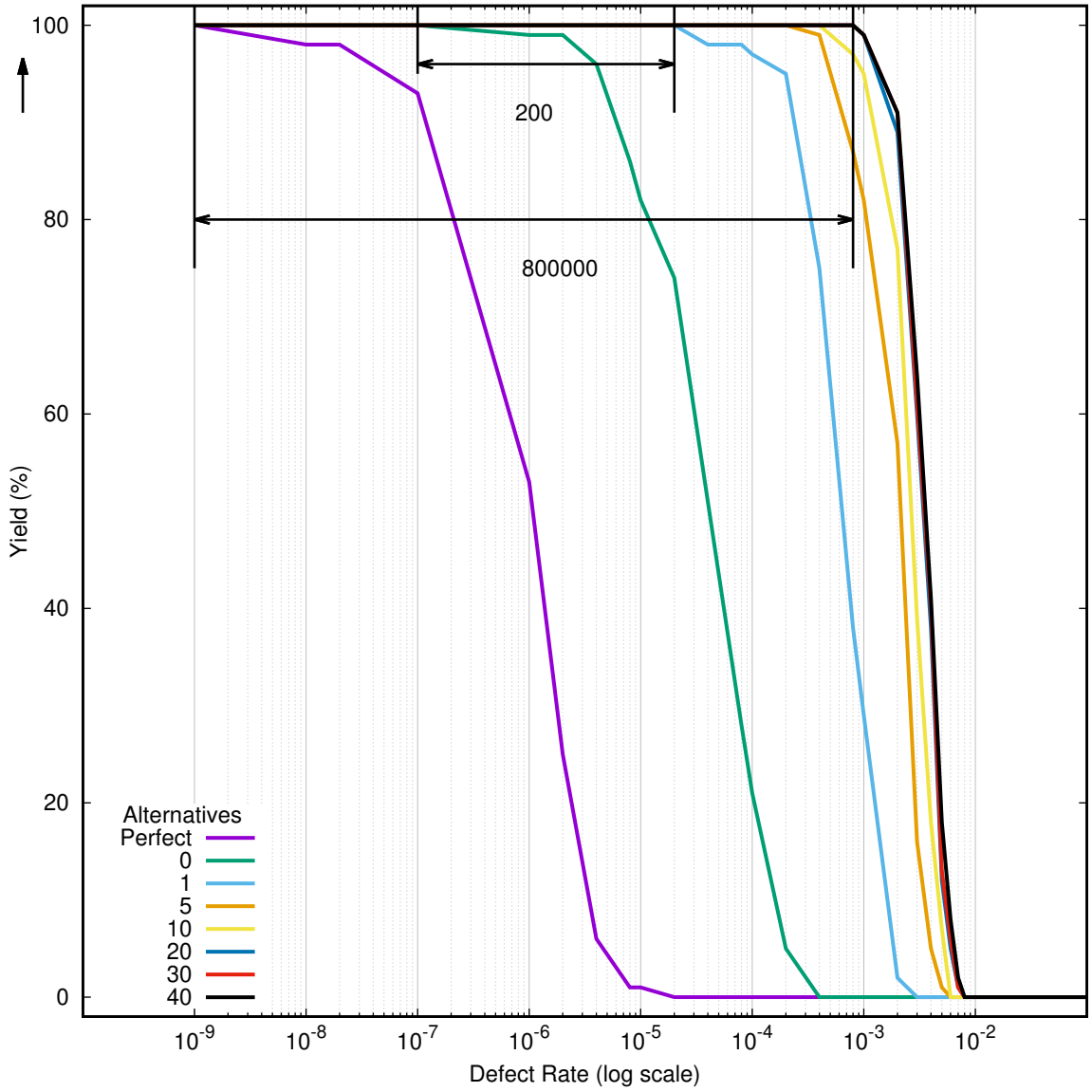


Figure 4.7: Yield vs. defect rate for `des` with 20% extra base tracks, 20% additional reserved tracks, and 1–40 alternatives per net (Path-Cost Algorithm, fully populated C-Boxes, VPR 4.3 [5], multi-driver interconnect, 100 chips). The rates of perfect chip production and of successful use of the base route on imperfect chips (zero-alternatives case) are shown for comparison.

4.5.1 Path-Cost Algorithm vs. Resource-Cost Algorithm

Figure 4.8 shows the difference between the Resource-Cost Algorithm and the Path-Cost Algorithm on `des` when using only 5 alternatives. The shaded area represents improvement in yield for the Path-Cost Algorithm. At this number of alternatives, the Path-Cost Algorithm maintains 100% yield for defect rates nearly 4 orders of magnitude higher than the Resource-Cost Algorithm. All other results in this thesis, including Figure 4.7, use the Path-Cost Algorithm.

4.5.2 C-Box Population

Architectures with depopulated connection boxes (C-Boxes, the crossbars connecting CLBs to the general interconnect) show more modest benefits from CYA (see Figure 4.9). CYA increases yield compared to the no-alternatives case, but we do begin to see some yield loss at defect rates as low as 10^{-6}. It is not surprising that the lower connectivity of depopulation results in lower defect tolerance. Later improvements to the alternatives generation capabilities (specifically, adding the ability to address defective LUTs, which enables the selection of different CLB output pins, see Section 4.7.6) reduced the impact of C-Box depopulation.

4.5.3 Additional Tracks

Since a typical FPGA design is seldom routed at W_{min}, I wanted to understand the impact of larger channel widths. Using a larger channel width means a smaller fraction of the routing resources are occupied by the base route, potentially leaving more resources available for alternatives. This led me to wonder if *extra* tracks might reduce or eliminate the need for *reserved* tracks.

With the assumption of full C-Box population in place, I examined the effects of

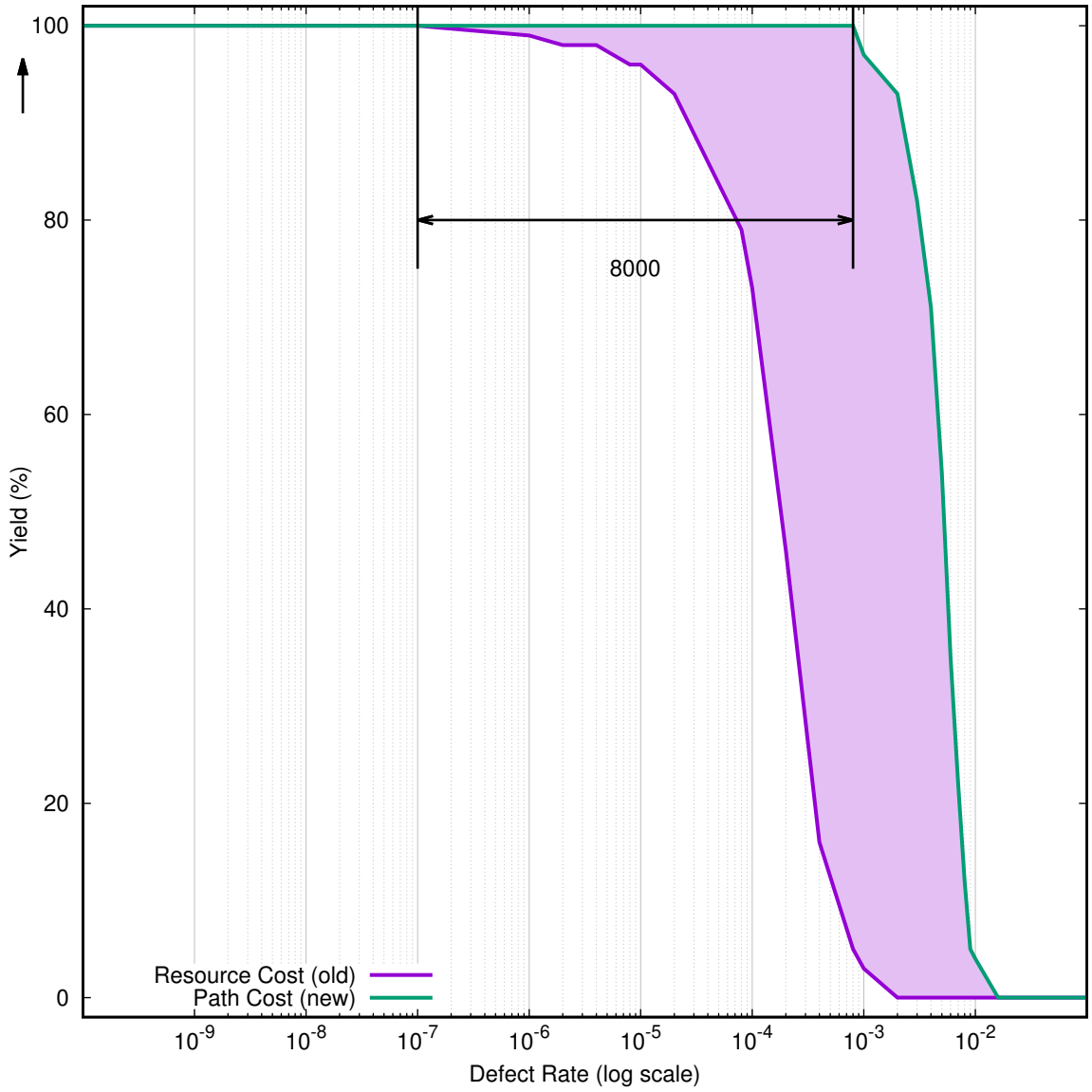


Figure 4.8: Yield vs. defect rate for the Resource-Cost Algorithm and the Path-Cost Algorithm (`des`, no extra base tracks, 20% reserved tracks, 5 alternatives, fully populated C-Boxes, VPR 4.3 [5], multi-driver interconnect, 100 chips).

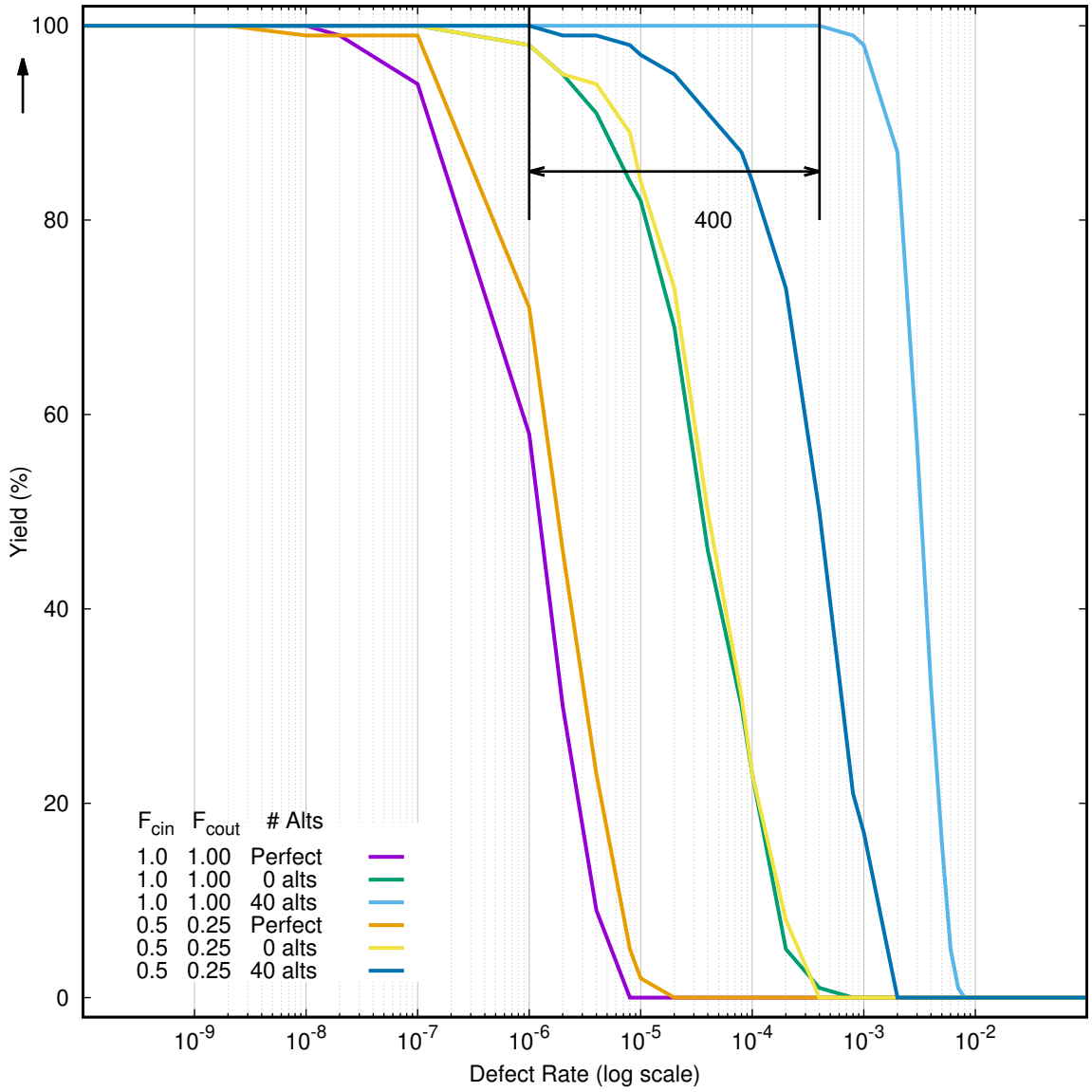


Figure 4.9: Yield vs. defect rate for `des` with depopulated and fully populated C-Boxes (depopulated case sets W_{min}, no extra base tracks, 20% reserved tracks, 40 alternatives, Path-Cost Algorithm, VPR 4.3 [5], multi-driver interconnect, 100 chips).

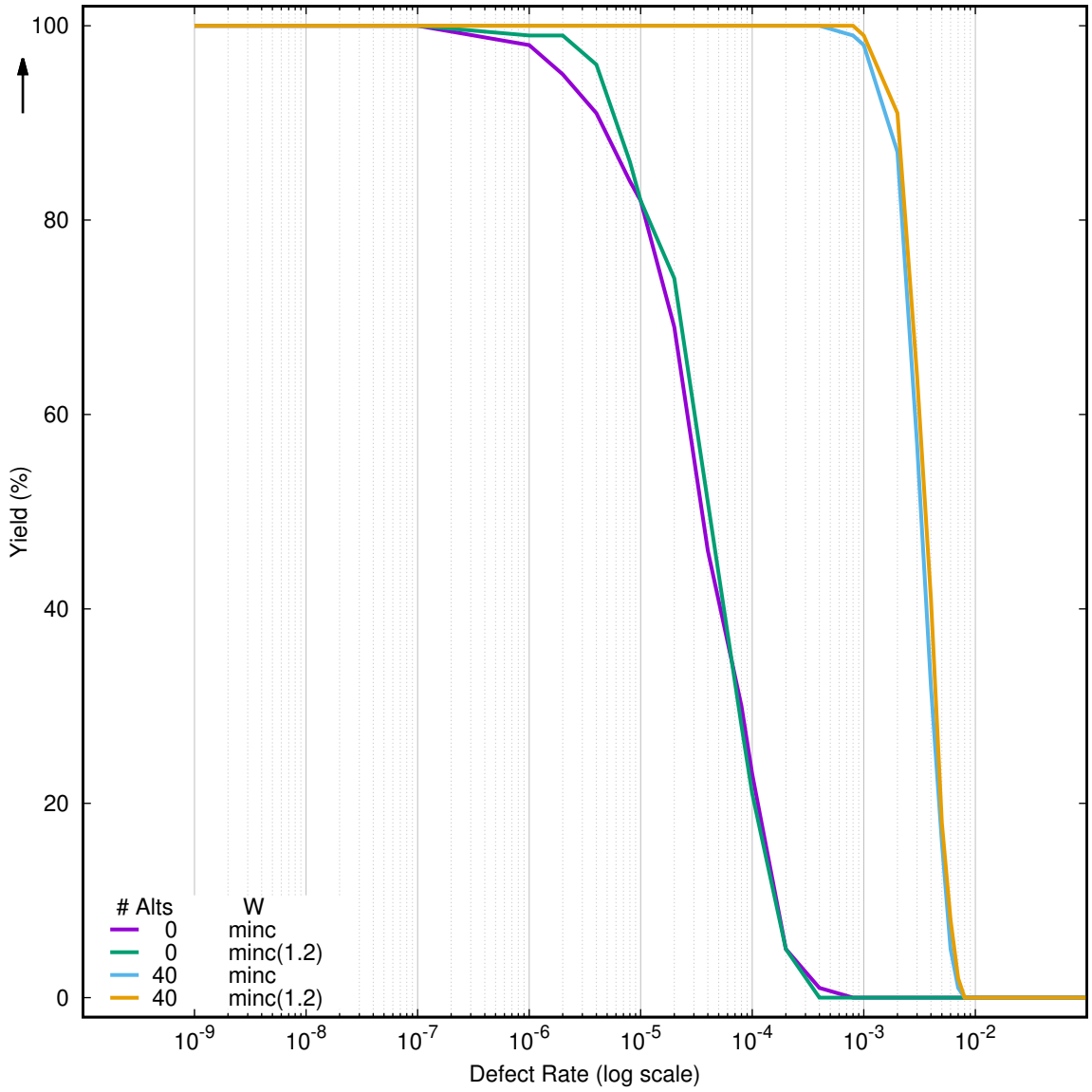


Figure 4.10: Yield vs. defect rate with and without extra base tracks (`des`, 20% reserved tracks, 0 or 40 alternatives, Path-Cost Algorithm, fully populated C-Boxes, VPR 4.3 [5], multi-driver interconnect, 100 chips).

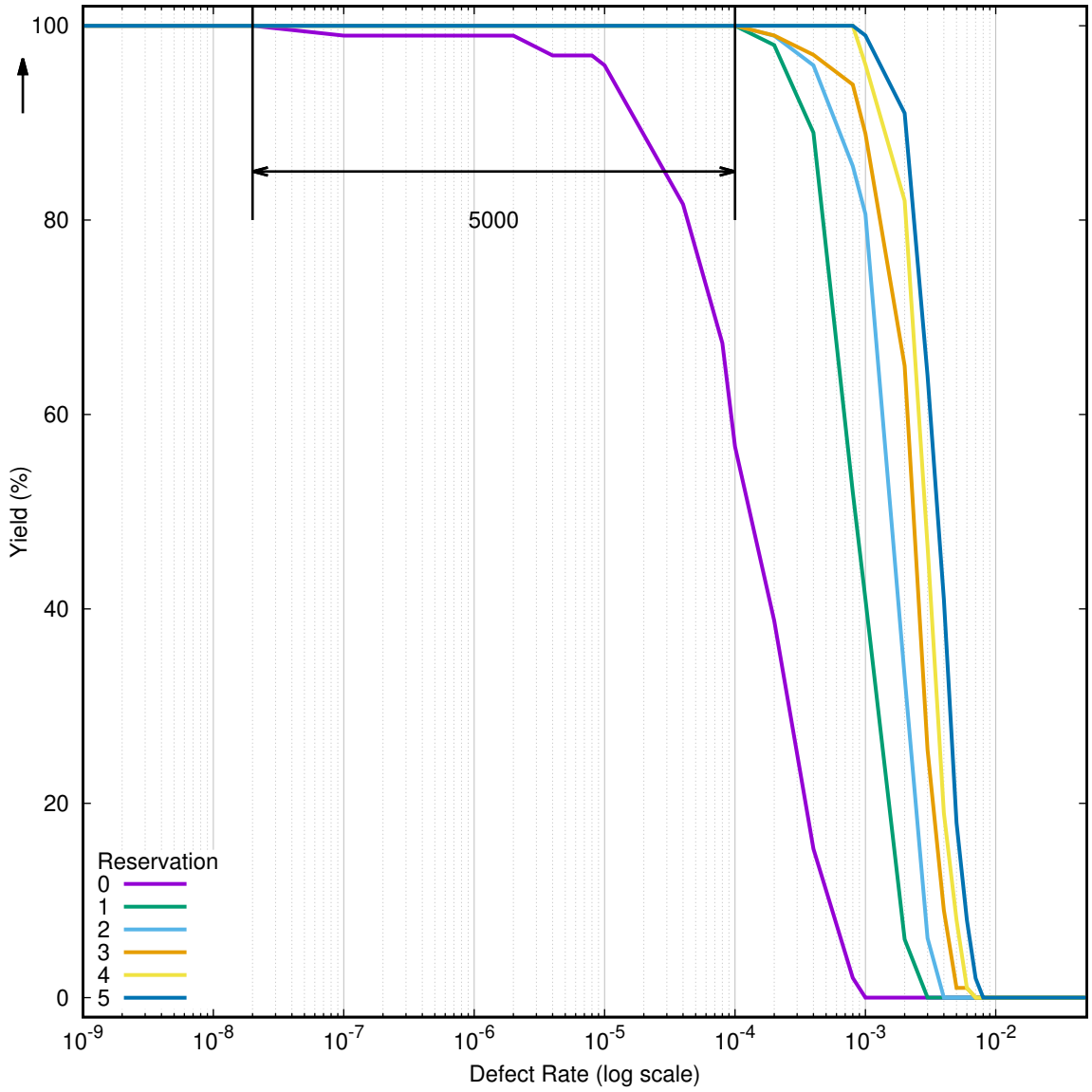


Figure 4.11: Yield vs. defect rate for different numbers of reserved tracks (`des`, 20% extra base tracks, 40 alternatives, Path-Cost Algorithm, fully populated C-Boxes, VPR 4.3 [5], multi-driver interconnect, 100 chips).

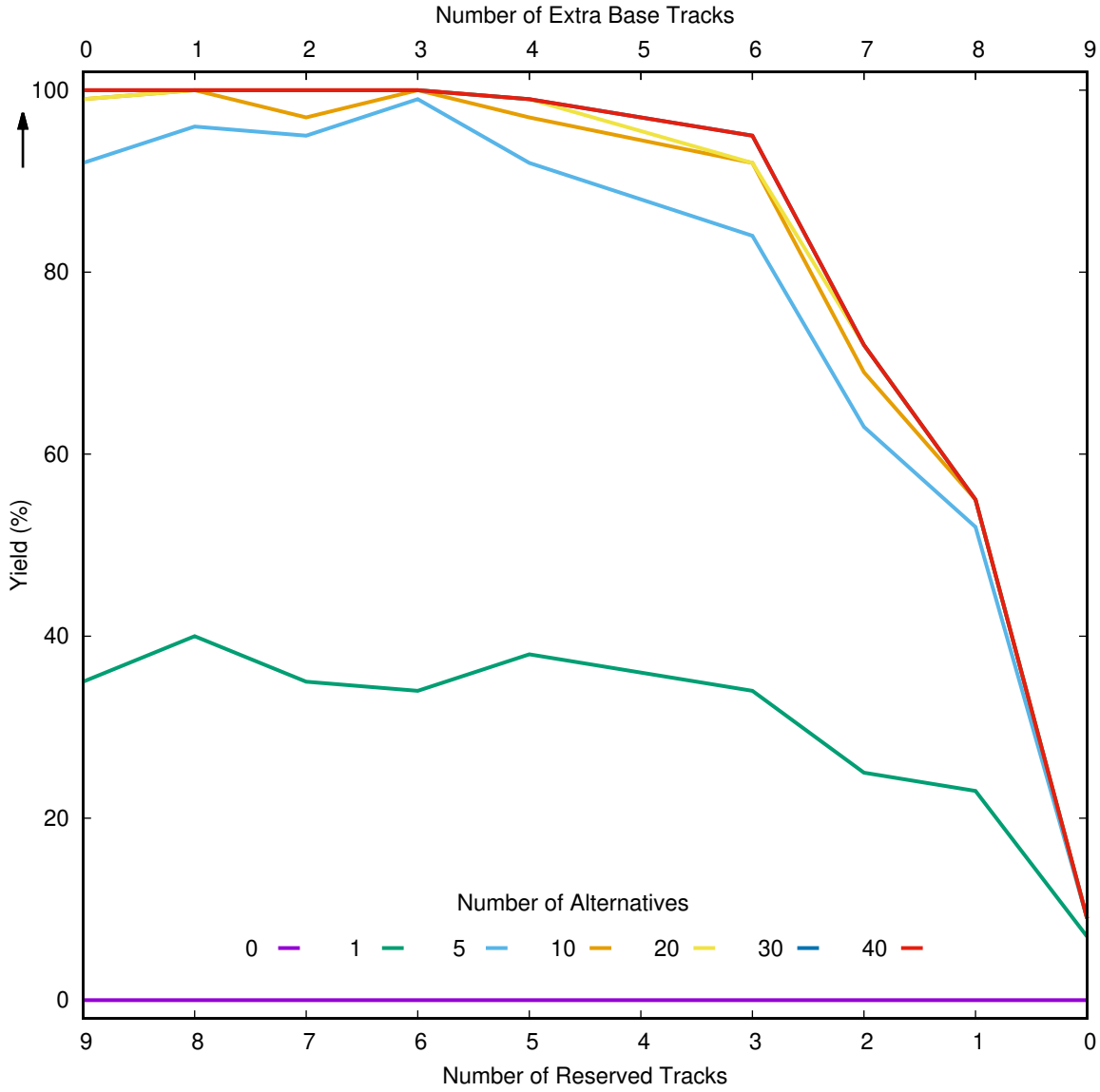


Figure 4.12: Effects of dividing additional tracks between extra base tracks and reserved tracks (`des`, $1.4W_{min}$ total tracks, 0–40 alternatives, Path-Cost Algorithm, fully populated C-Boxes, VPR 4.3 [5], multi-driver interconnect, 100 chips, defect rate 10^{-3}). Note that the number of reserved tracks decreases and the number of extra base tracks increases going from left to right.

varying numbers of extra base tracks and reserved alternative tracks on CYA yields. In Figure 4.10, we see that adding 20% extra base tracks to `des` with 20% reserved tracks provides essentially no benefit compared to a case with no extra base tracks. Conversely, Figure 4.11 shows that *adding a single reserve track per channel to* `des` *with 20% extra base tracks maintains 100% yield at defect rates more than three orders of magnitude above the level at which failures begin to appear in the absence of reserved tracks.* This result demonstrates that there is value in reserving tracks exclusively for alternatives even when there are a number of extra tracks. Alternatives constrained to using only the non-reserved tracks tend to be fragmented by the need to avoid resources used by the base route, making them significantly less effective at providing repair diversity.

Figure 4.12 makes this contrast between the usefulness of extra base tracks vs. reserved alternative tracks more explicit. In this graph, I fix the number of additional tracks above W_{min} at 9 (40%), while varying the division of these tracks between extra base tracks and reserved alternative tracks. These data show that each track which is converted from an extra base track to a reserved track improves the effectiveness of CYA; however, the most significant improvements come from the addition of the first four reserved tracks (roughly $0.2W_{min}$). In practice, the number of useful reserved tracks will be a function of the defect rate. In this case, CYA proved to be too effective to demonstrate the reserved track/extra base track tradeoff at a per-resource defect rate of 10^{-4}, as the presence of even a single reserved track ensured near 100% yield. The figure therefore shows data at a defect rate of 10^{-3}.

4.5.4 Impact on Circuit Delay

Results presented up to this point were produced using CYA algorithms that almost entirely ignored circuit delay. The one exception to this is that the Pathfinder delay

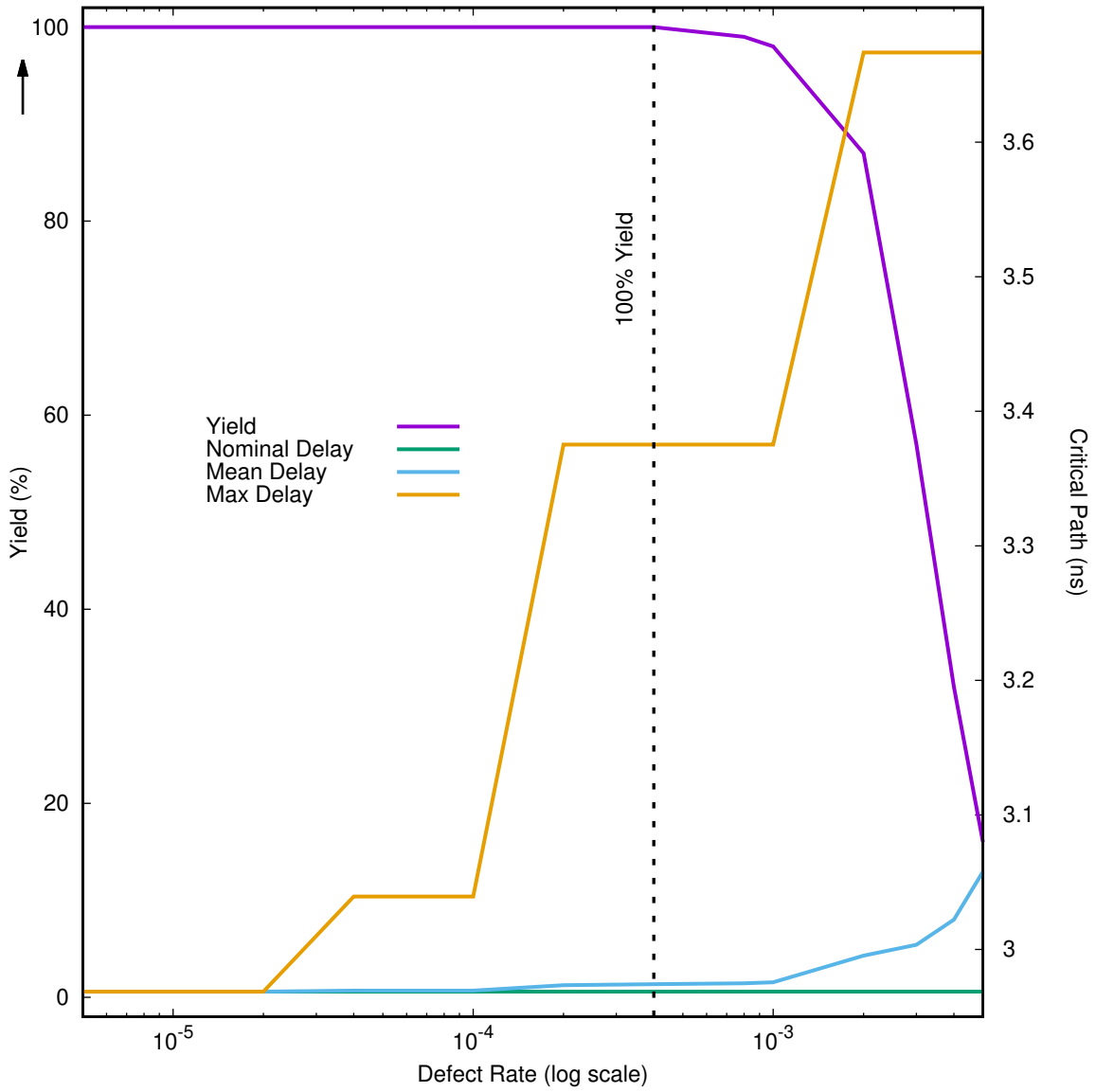


Figure 4.13: Yield and critical path delay vs. defect rate for CYA, as compared to nominal chips (`des`, no extra base tracks, 20% reserved tracks, 40 alternatives, Path-Cost Algorithm, fully populated C-Boxes, VPR 4.3 [5], multi-driver interconnect, 100 chips). CYA preserves 100% yield and near-nominal delays up to per-component defect rates of approximately 10^{-4}.

component of the cost function was used for alternatives generation (see Section 4.2.3). However, in my path generation algorithm, producing a diverse set of alternatives is prioritized over minimizing delay for all alternatives. Thus, we expect the full set of alternatives to contain non-minimal delay paths. Later, in Chapters 5 and 6, the algorithms are expanded to address delay, and the simulation model is enhanced to provide more realistic delay values for nominal chips and to simulate the effects of post-fabrication variations on delay. Given the slight preference of the alternatives generation process for faster paths, the empirical relationship between the delay of repaired circuits and the underlying defect rate provides insight into the adequacy of the resources available for repair, as well as an indication of how much adjustment will be required to avoid significant slow-downs.

In Figure 4.13 the "Max Delay" curve matches the nominal delay perfectly up to a per-component defect rate of around $2 \cdot 10^{-5}$. Up to fairly high defect rates of around 10^{-4}, some unlucky chips begin to suffer a little extra delay ($\sim$2%). This worsens to $\sim$14% before the yield starts to fall off between $4 \cdot 10^{-4}$ and $8 \cdot 10^{-4}$. In the worst cases, we still only see a slowdown of 24% before CYA yield drops to 0. Note that CYA shows no slowdown *past the point where we have no "perfect" chips* and *past the point where oblivious routing yield (the fraction of cases where defects do not occur on any of the resources used in the base route) falls below* 80% (see Figure 4.9 for the non-CYA yield data).

The "Mean Delay" curve tells an even more encouraging story about the impact of CYA repair on delay. Even counting the most unlucky chips, the *average slowdown is less than* 0.2% for even the worst defect rates at which CYA achieves full yield.

Without any further modifications, parametric yield (the fraction of cases where CYA produces a functioning configuration that matches the defect-free delay) is nearly as good as functional yield. My additional enhancements to address delay in the

presence of variations (covered in Section 5.2) further rein in the outliers that appear in Figure 4.13. Figure 6.1 shows the impact of adding delay awareness to CYA.

4.5.5 Impact of Switch and Wire Defects

Figure 4.14 separates out the contributions of switch and wire defects, using `des` as an example. Note that each source-to-sink path for a two-point net will have exactly one more switch than wire segment. That is, except for the final C-box switch to the destination cluster, every switch in the path is followed by a wire segment. Consequently, with equal switch and wire defect rates, we will see a slightly higher likelihood of a path failing due to switch defects than due to wire defects, which is consistent with the results shown.

Our ability to successfully replace a failed base route path with an alternative will depend both on: (a) whether or not the alternative is defect-free, and (b) whether or not some resource needed by the alternative has already been used by a preceding displaced base route path. The likelihood of an alternative being defect-free is the same as any source-to-sink path as discussed above. The likelihood of the alternative being free of already-used resources depends on how good a job CYA does at encouraging resource diversity in the routes. The fact that the broken wire and stuck-open switch curves are so close suggests that it is achieving a similar quality of diversity in both cases.

When we route multi-point nets with efficient fanout trees, the prefixes of many two-point paths will share the same resources in the base route. As a result, defects that occur closer to the source of a multi-point net fanout tree will disrupt more two-point nets than defects that occur closer to the sinks. This effect has roughly the same impact in both the broken wire and stuck-open switch cases, since wires and switches come in pairs along the path.

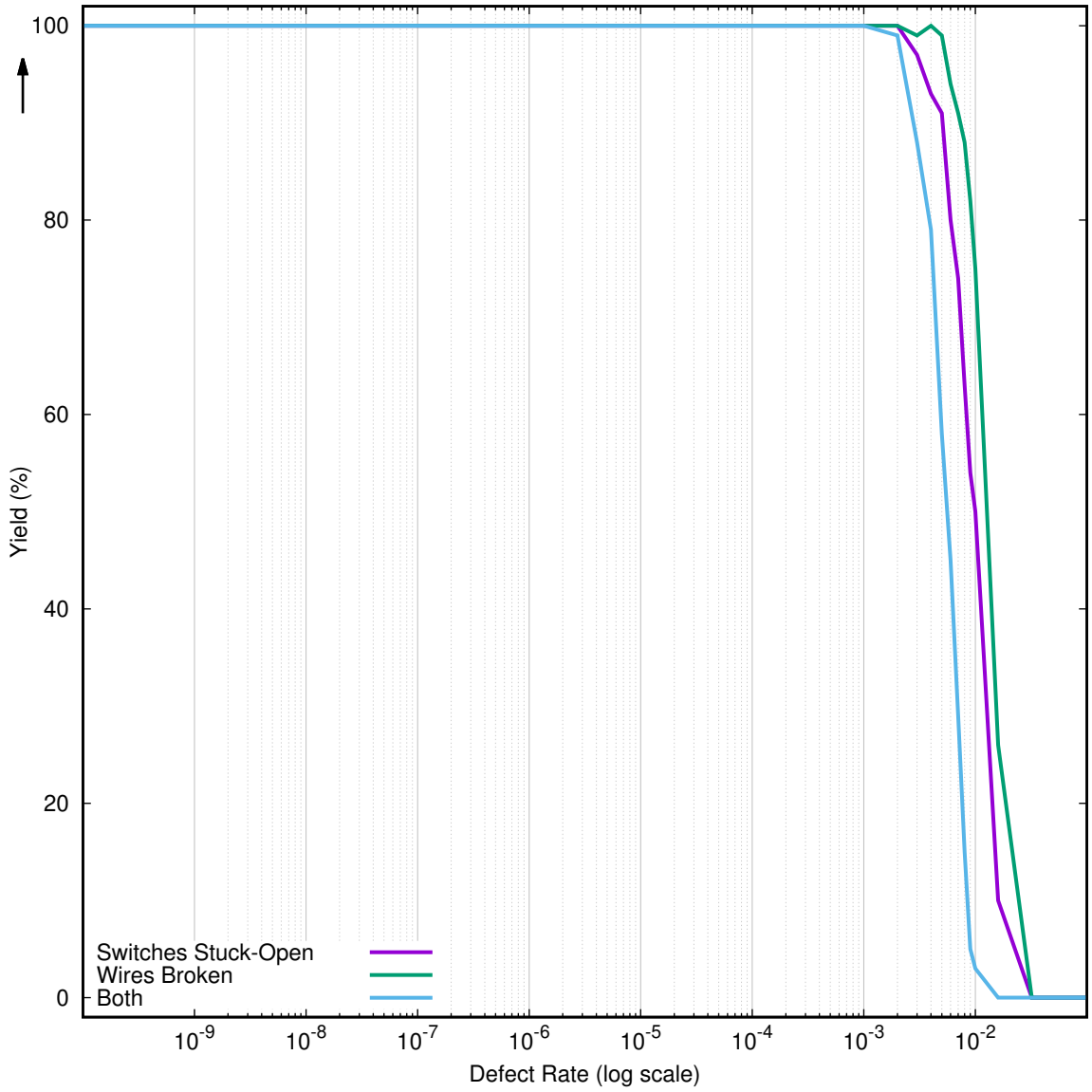


Figure 4.14: Yield vs. defect rate for stuck-open switch and broken-wire defect types (`des`, no extra base tracks, 20% reserved tracks, 40 alternatives, Path-Cost Algorithm, fully populated C-Boxes, VPR 4.3 [5], multi-driver interconnect, 100 chips).

In real chips, wire and switch defects may not occur at the same rates. If one defect type is substantially more common than the other, the composite defect rate will be dominated by that of the resource type with the larger defect rate, so we can understand the primary yield effect using the curve for the dominant defect source. The "both" defects curve in Figure 4.14 shows the composite effect in the case where the defect rates are the same, illustrating how they compose when neither defect rate dominates.

4.5.6 Summary

While my examples above all discuss results for `des` only, the results for the other Toronto 20 designs [6] were similar. Table 4.2 shows the results for experiments with a defect rate of 10^{-4} and no extra base tracks. The larger designs show dramatic yield improvements with the addition of alternatives, improving from zero yield to near-100% yield as we go from 0 to 40 alternatives in the presence of reserve tracks.

4.6 Bitstream Impact

Storing and loading alternatives will make the bitstream larger and lengthen bitstream load time. Tables 4.4 and 4.5 record design and experimental statistics and estimate bitstream sizes and load times for CYA on the Toronto 20 benchmarks [6]. These estimates suggest that the CYA bitstream may be a factor of 2–50 larger than a conventional bitstream depending on the number of alternatives stored (Section 4.6.1) and take 2–200 times longer to load depending on the configuration architecture (Section 4.6.2). In practice, bitstream size and load time are so insignificant that even these large overheads would have little impact for most applications. Moreover, the assumptions that were used for these estimates are intentionally conservative; there-

					Reserved Tracks					
					+0% # alternatives			+20% # alternatives		
Design					0	1	40	0	1	40
name	LUTs	s	W	N_{2pt}	% yield					
tseng	1064	17	29	2069	60	70	70	50	100	100
ex5p	1120	17	48	2516	36	44	44	41	97	100
apex4	1336	19	46	2908	23	36	36	30	99	100
dsip	1372	27	28	3260	29	36	36	31	97	100
misex3	1448	20	41	3215	26	34	34	29	95	100
diffeq	1516	20	32	2987	38	49	49	37	97	100
alu4	1556	20	38	3367	21	30	30	31	96	100
des	1660	32	28	3612	15	35	36	23	98	100
bigkey	1708	27	24	3661	22	35	35	32	95	100
seq	1792	22	44	4000	22	29	29	19	96	100
apex2	1940	23	44	4386	21	31	31	22	99	100
s298	1956	23	32	4107	22	30	30	40	90	100
frisc	3572	30	49	7427	7	11	11	5	84	100
elliptic	3624	31	45	7153	5	11	11	3	87	100
spla	3820	31	60	8757	1	8	8	4	92	100
pdc	4760	35	68	10968	1	4	4	0	78	100
ex1010	4804	35	49	10772	1	4	4	0	86	100
s38417	6440	41	38	12284	0	9	9	1	86	100
s38584.1	6452	41	36	11185	1	9	9	0	90	99
clma	8548	47	58	18266	0	0	0	0	69	98
	Geometric Mean				0	0	0	0	91	99

Table 4.2: CYA yield improvement for the Toronto 20 [6] benchmarks, sorted by size. (No extra base tracks, Path-Cost Algorithm, fully populated C-Boxes, VPR 4.3 [5], multi-driver interconnect, 100 chips, defect rate 10^{-4} for **both** stuck-open switches and wire breaks.) See Table 4.3 for a key to the design parameters.

Symbol	Definition
s	Number of CLBs across the side of the FPGA; s^2 is the total number of CLBs in the FPGA
W	Number of tracks per channel
N_{2pt}	Number of two-point nets in the design
T_{pl}	Total path length of the base configuration: $\sum_{i=1}^{N_{2pt}} (net2pt[i].path\_length)$
t_{alt}^{i}	Number of alternatives tried for two-point net i during configuration
T_{alt}	Total number of paths (base + alts) tried during configuration: $N_{2pt} + \sum_{i=1}^{N_{2pt}} t_{alt}^{i}$
T_{plalt}	Total length of all paths (base + alts) tried during configuration: $T_{pl} + \sum_{i=1}^{N_{2pt}} \sum_{j=1}^{t_{alt}^{i}} (net2pt[i].alt[j].path\_length)$
F_{tch}	Number of frames touched setting and clearing paths

Table 4.3: Bitstream table parameters.

fore, with improved modeling and improvements to storage and loading methodologies, actual overhead may be reduced significantly in practice.

4.6.1 Bitstream Size

I estimate the number of routing configuration bits, B_{conv}, for a conventional, unencoded FPGA bitstream as:

$$B_{conv} = s^2 \cdot W \cdot (F_{cin} \cdot I + F_{cout} \cdot O + 1 + 4/L_{seg}) \tag{4.1}$$

I, O, F_{cin}, F_{cout}, and L_{seg} are defined in Section 4.4.3. Measurements in this section and Section 4.6.2 use $F_{cin} = F_{cout} = 1$ (fully-populated C-Boxes). s and W are defined in Table 4.3. I assume that 5 bits are required to configure the switchpoint at the end of each segment (2 bits to specify which of the 4 sides is the source and 3 bits to specify drive in each of the other 3 directions). 1 additional bit per tile is required to configure the mid-segment switchpoints for each segment.

Design		Bitstream Size				
		Conventional	CYA-1		CYA-40	
name	T_{pl}	Kbits	Kbits	ratio	Kbits	ratio
tseng	6232	131	333	2.5	4448	34
ex5p	7570	217	409	1.9	5504	26
apex4	11644	260	597	2.3	8913	35
dsip	13136	319	689	2.2	10074	32
misex3	12920	257	662	2.6	9891	39
diffeq	8989	200	480	2.4	6416	33
alu4	13580	238	675	2.8	9989	43
des	14458	448	766	1.7	11229	26
bigkey	14867	274	778	2.8	11405	42
seq	16046	333	823	2.5	12283	37
apex2	17646	364	925	2.5	13531	38
s298	16547	265	867	3.3	12690	48
frisc	30115	690	1635	2.4	24306	36
elliptic	35850	676	1855	2.7	29166	44
spla	43890	901	2288	2.5	36057	41
pdc	54965	1302	3005	2.3	46971	37
ex1010	43400	938	2433	2.6	35503	38
s38417	37072	999	2265	2.3	30044	31
s38584.1	45094	946	2527	2.7	36890	40
clma	91901	2002	5145	2.6	79365	40

Table 4.4: CYA vs. conventional bitstream size comparison for the Toronto 20 [6] benchmarks, sorted by size. (No extra base tracks, 20% additional reserved tracks, fully-populated C-Boxes, VPR 4.3 [5], multi-driver interconnect.)

The CYA bitstream contains additional information specifying the configuration of each alternative for each two-point net. Assuming sparse storage where we must provide an address for each configuration bit, I estimate the number of bits required to specify a CYA bitstream as follows:

$$\begin{aligned} B_{alt} = N_{2pt} \cdot \left(\left\lceil \log_2 \left(s^2 I \cdot W \cdot F_{cin} \right) \right\rceil + \left\lceil \log_2 \left(s^2 O \cdot W \cdot F_{cout} \right) \right\rceil \right) \\ + (T_{pl} - 2N_{2pt}) \cdot \left(\left\lceil \log_2 \left(s^2 W \right) \right\rceil + 5 \right) \end{aligned} \tag{4.2a}$$

$$B_{tpath} = N_{2pt} \cdot 5 \left(\left\lceil \log_2 \left(s^2 O \right) \right\rceil + 1 \right) \tag{4.2b}$$

$$B_{cya} = (N_{alt} + 1) \cdot B_{alt} + B_{tpath} \tag{4.2c}$$

B_{alt} is the number of bits required to store one alternative for every two-point net, B_{tpath} is the number of bits required to specify the tests for all the nets, and B_{cya} is the resulting total size of a bitstream with N_{alt} alternatives. N_{2pt} and T_{pl} are defined in Table 4.3.

Each path starts and ends at a C-Box, so the first term in the computation of B_{alt} (Equation (4.2a)) is for specifying the start and end C-Box connections, while the second term is for specifying the S-Box switch settings. I again assume that an S-Box switchpoint requires 5 bits.

To test a two-point net, we need to: (1) set a zero into the driver, (2) set up a one for transition on the driver, (3) read out the result of the zero-one test, (4) set up a zero for transition on the driver, and (5) read out the result of the one-zero test. This sets the multiplier of 5 in the computation of B_{tpath} (Equation (4.2b)).

Table 4.4 compares the bitstream sizes for the Toronto 20 designs ($N_{alt} = 1$ and 40) with conventional bitstream sizes. As expected, the bitstream size increases in rough proportion to the number of alternatives provided. Beyond configuration multiplicity, the overhead in these CYA bitstream size estimates comes from providing

a complete address for every configuration or test resource. Exploiting locality and regularity should enable significant reduction of this cost. My approach here deliberately avoids making assumptions about the structure of the FPGA architecture and bitstream; exploiting architectural structure (e.g., domains [89]) would allow more compact expression of paths and alternatives.

4.6.2 Bitstream Load Time

Conventional bitstream load time is typically limited by load bandwidth:

$$L_{conv} = B_{conv}/BW_{load} \tag{4.3}$$

For concreteness, I assume a system like the Virtex-5 [93] that can load 16-bit values at 50MHz, giving BW_{load} =16b/20ns.

With random access into the stored bitstream, CYA can skip over alternatives that it does not need to load. The number of bits we need to read in this case is the total number of bits required to specify all of the alternatives tried during configuration plus the total number of bits required to specify testing for each of those alternatives:

$$\begin{aligned} R_{cya} = \Big\{ T_{alt} \cdot \Big[\Big\lceil \log_2 \left(s^2 I \cdot W \cdot F_{cin} \right) \Big\rceil + \Big\lceil \log_2 \left(s^2 O \cdot W \cdot F_{cout} \right) \Big\rceil \Big] \\ + (T_{plalt} - 2T_{alt}) \cdot \Big(\Big\lceil \log_2 \left(s^2 W \right) \Big\rceil + 5 \Big) \Big\} \\ + T_{alt} \cdot 5 \left(\Big\lceil \log_2 \left(s^2 O \right) \Big\rceil + 1 \right) \end{aligned} \tag{4.4}$$

Note the correspondence with Equations (4.2a) and (4.2b). T_{alt} and T_{plalt} are defined in Table 4.3. If we have random access to set configuration bits and set and read CLB flip-flops for testing, the time to read R_{cya} bits will determine CYA load time.

Alternately, if we use a frame modification scheme (Section 4.2.4), time will be determined not by the number of bits changed, but by the number of frames touched

Design		Experimental		Load Time				
				Conv	Random Access		Frame Modification	
name	T_{pl}	T_{alt}	T_{plalt}	ms	ms	ratio	ms	ratio
tseng	6232	2079	6272	0.168	0.292	1.7	28	164
ex5p	7570	2533	7633	0.278	0.359	1.3	34	121
apex4	11644	2924	11702	0.333	0.494	1.5	44	131
dsip	13136	3284	13344	0.409	0.580	1.4	50	121
misex3	12920	3245	13100	0.328	0.551	1.7	49	148
diffeq	8989	3013	9123	0.256	0.424	1.7	40	156
alu4	13580	3428	13972	0.304	0.572	1.9	52	170
des	14458	3626	14480	0.574	0.639	1.1	54	94
bigkey	14867	3736	15524	0.350	0.669	1.9	58	164
seq	16046	4033	16202	0.426	0.683	1.6	60	141
apex2	17646	4437	17952	0.466	0.782	1.7	67	143
s298	16547	4166	17004	0.339	0.738	2.2	63	186
frisc	30115	7519	31065	0.882	1.380	1.6	115	130
elliptic	35850	7195	36150	0.865	1.489	1.7	119	138
spla	43890	8792	44170	1.154	1.830	1.6	145	126
pdc	54965	11052	55426	1.666	2.423	1.5	183	110
ex1010	43400	10830	43956	1.201	2.048	1.7	162	135
s38417	37072	12376	37486	1.278	2.006	1.6	164	129
s38584.1	45094	11304	46184	1.211	2.145	1.8	171	141
clma	91901	18501	94432	2.563	4.239	1.7	311	122

Table 4.5: CYA (random or frame-based access) vs. conventional bitstream load time comparison for the Toronto 20 [6] benchmarks, sorted by size. (No extra base tracks, 20% additional reserved tracks, fully-populated C-Boxes, VPR 4.3 [5], multi-driver interconnect, 100 chips, defect rate 10^{-4}.)

and the time to shift and modify each frame. I assume the C-Box connections at the beginning and end of a path are each in one frame. For a conservative estimate, I assume that every S-Box switch in the path touches a separate frame. This means that the path length is equal to the number of frames touched, so I estimate the frames touched as T_{plalt}. When a path is bad, we must unload it, meaning that all but T_{pl} of the frames must be touched twice.

$$F_{tch} = 2T_{plalt} - T_{pl} \tag{4.5}$$

For concreteness, I assume 1312-bit frames, similar to Virtex-5 [93], and define frame load time to match the previous bandwidth assumptions ($T_{frame} = 1312\text{b}/BW_{load}$). Using Equation (4.5), I estimate the CYA load time:

$$L_{frame} = (F_{tch} + 5 \cdot T_{alt}) \cdot T_{frame} \tag{4.6}$$

In this equation, the extra 5 frames loaded per alternative are intended to account for tests of the type included in the random access calculations (Section 4.6.1 and Equation (4.2b)).

As Table 4.5 shows, the frame scheme loads two orders of magnitude slower than a conventional bitstream load. This is the trade-off it makes to guarantee that *no changes* are required to the core of the FPGA architecture. The random read case is a factor of 2–3 slower than the conventional case, and its slowdown is driven largely by the simplistic encoding assumed in my bitstream size estimate. I expect that the overhead for both cases can be reduced significantly with minor modifications.

4.6.3 Updated Bitstream Tables

As mentioned in Section 4.4.1, after obtaining the initial results discussed in Section 4.5, I ported my models to VPR 5 [53] to add simulation of single-driver (direct drive) interconnect, which is more representative of current hardware. For similar reasons, several other parameters were also changed at this point:

- 8 LUTs plus 4 spares per CLB ($n = 12$, $O = 12$)
- 6 inputs per LUT ($k = 6$)
- 27 input pins plus 16 spares ($I = 43$)

The architecture configuration file in Appendix B.3 provides further architecture parameters for these experiments.

VPR 5 only supports direct drive with Wilton S-Boxes [55], not the diamond (subset) S-Boxes used in my VPR 4.3 experiments. The Wilton S-Box topology required modifications to support split channels for the reserved tracks (described in [24, 23]), as it breaks domains by design, which prevents clean track-based reservation.

Repair of logic defects (Section 4.7.6) was added in the CYA algorithm used in the VPR 5 experiments. This made it possible to use more realistically depopulated C-Boxes. The population parameters for these experiments were $F_{cin} = 0.15$, $F_{cout} = 0.2$, $F_{cin,extra} = 0.25$, and $F_{cout,extra} = 0.1$. An improved defect density model (modeling defects per unit area rather than per resource), as well as variation, delay, and energy modeling based on SPICE [64] (specifically, HSPICE [35]) were also developed for later experiments with this model. Routing based on the work described in [71, 72] was also added at this point, to ensure the quality and reliability of delay results computed for base routes.[3]

[3]The congestion-oblivious delay lower bounds discussed in Appendix A.1 are a related application of this work.

Table 4.6 updates the Toronto 20 bitstream parameters given in Table 4.2 to those for my VPR 5 model. Table 4.7 gives the resulting bitstream sizes (compare to Table 4.4 for the VPR 4.3 sizes, and note the changed size ordering). Table 4.8 gives the updated bitstream load times (compare to Table 4.5). Note that VPR 5 experiments also typically used a maximum of 64 alternatives, rather than the 40-alternative cap used in the VPR 4.3 experiments. As shown in Figure 4.7 and Table 4.11, the yield impact of these extra alternatives is not typically significant for this benchmark set.

All results reported from this point forward are based on this VPR 5 model.

name	LUTs	s	W	N_{2pt}
des	556	16	64	1453
ex5p	612	9	88	1487
s298	630	9	80	1642
tseng	663	10	64	1090
diffeq	671	10	80	1391
bigkey	763	14	64	1515
misex3	767	10	88	2002
alu4	794	10	80	1914
apex4	795	11	96	2185
seq	828	11	96	2358
dsip	873	14	64	1729
apex2	922	11	96	2640
elliptic	1815	16	88	3805
frisc	1888	16	112	4474
spla	1927	16	120	5391
pdc	2369	18	136	6781
s38584.1	2475	18	80	4416
ex1010	2519	18	144	7619
s38417	2628	19	80	4233
clma	2957	20	112	7871

Table 4.6: Toronto 20 [6] benchmark parameters updated for VPR 5 [53] CAD flow with single-driver interconnect, sorted by benchmark size. (Compare to the first five columns of Table 4.2 for VPR 4.3 [5].) See Table 4.3 for a key to the parameters.

Design					Bitstream Size				
					Conv	CYA-1		CYA-64	
name	s	W	N_{2pt}	T_{pl}	Mbits	Mbits	ratio	Mbits	ratio
des	16	64	1453	4396	174	241	1.4	4933	29
ex5p	9	88	1487	4477	76	223	2.9	4723	63
s298	9	80	1642	4938	69	243	3.5	5105	74
tseng	10	64	1090	2261	68	131	1.9	2232	33
diffeq	10	80	1391	2812	85	167	2.0	2853	34
bigkey	14	64	1515	4609	133	249	1.9	5078	39
misex3	10	88	2002	6036	93	314	3.4	6507	70
alu4	10	80	1914	5793	85	297	3.5	6124	73
apex4	11	96	2185	6593	123	347	2.8	7247	59
seq	11	96	2358	7112	123	374	3.0	7818	64
dsip	14	64	1729	5247	133	284	2.1	5779	44
apex2	11	96	2640	7961	123	419	3.4	8751	72
elliptic	16	88	3805	11514	239	647	2.7	13410	57
frisc	16	112	4474	13509	304	768	2.5	16014	53
spla	16	120	5391	16285	326	926	2.8	19306	60
pdc	18	136	6781	27238	467	1468	3.1	34156	74
s38584.1	18	80	4416	13424	275	753	2.7	15641	57
ex1010	18	144	7619	30598	494	1649	3.3	38369	78
s38417	19	80	4233	12820	306	749	2.4	15221	50
clma	20	112	7871	23784	475	1421	3.0	29245	62

Table 4.7: CYA vs. conventional bitstream size comparison for the Toronto 20 [6] benchmarks (sorted by size), updated to use VPR 5 [53] with single-driver interconnect and the full defect model. (1 and 64 alternatives, 20% extra base tracks, 16 reserved tracks.) Compare to Table 4.4 for VPR 4.3.

Design		Experimental		Load Time				
				Conv	Random Access		Frame Modification	
name	T_{pl}	T_{alt}	T_{plalt}	ms	ms	ratio	ms	ratio
des	4396	1613	4807	0.219	0.235	1.1	22	100
ex5p	4477	1658	4989	0.095	0.216	2.3	23	241
s298	4938	2015	5989	0.087	0.258	3.0	28	323
tseng	2261	1504	3484	0.086	0.180	2.1	20	234
diffeq	2812	1541	3261	0.107	0.179	1.7	19	178
bigkey	4609	2427	7342	0.168	0.353	2.1	36	215
misex3	6036	2104	6316	0.118	0.289	2.5	28	238
alu4	5793	2060	6405	0.107	0.286	2.7	28	262
apex4	6593	2413	7482	0.155	0.341	2.2	34	219
seq	7112	2473	7492	0.155	0.345	2.2	33	212
dsip	5247	2343	7771	0.168	0.357	2.1	36	215
apex2	7961	2737	8333	0.155	0.383	2.5	37	238
elliptic	11514	4293	13437	0.301	0.653	2.2	60	199
frisc	13509	4828	14684	0.383	0.729	1.9	66	172
spla	16285	5888	18170	0.411	0.896	2.2	81	197
pdc	27238	7209	28991	0.589	1.293	2.2	110	187
s38584.1	13424	4738	14775	0.347	0.719	2.1	65	187
ex1010	30598	9130	36647	0.624	1.635	2.6	145	232
s38417	12820	5002	15881	0.386	0.803	2.1	72	186
clma	23784	8413	25707	0.599	1.348	2.2	114	190

Table 4.8: CYA (random or frame-based access) vs. conventional bitstream load time comparison for the Toronto 20 [6] benchmarks, sorted by size. (20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 1000 chips, 800 defects/cm^2.) Compare to Table 4.5 for VPR 4.3.

4.7 Repair of Different Resource Types

In this section I demonstrate that CYA is capable of repair around defects in all types of resources in the reconfigurable fabric of a typical FPGA. Almost all simulations supporting these results (Figures 4.20 to 4.25) use the same per-resource defect rates for all resource types. Figure 4.26 switches to a more realistic assumption of constant defect *density* per unit area, resulting in different per-resource defect rates for different types of resources.

4.7.1 The Fabric and the Tile

A typical FPGA fabric is composed of a matrix of uniform tiles, as exemplified in Figure 4.15. Zooming in, Figure 4.16 illustrates a single tile, as modeled in my enhanced version of VPR 5 [53]. Note that the number of resources is reduced for clarity.

4.7.2 Channel Wires

Channel wires are modeled as rectangular strips of conductive material with lengths that are an integer multiple of the edge length of one tile in the FPGA. Uniformly distributed random values are sampled to determine if a defect is present. To account for the various wire lengths, measured in terms of the integer number of tiles spanned by the wires, samples are taken for each tile spanned by a given wire.

Breaks are sampled once per tile for each wire. I conservatively model the loss of an entire wire if any defects are present. In practice, a break that is beyond the exit point of a signal may not hinder the signal; indeed, the reduced capacitance may actually speed up signal propagation. However, a partial-use model for broken wires exceeds the scope of this dissertation.

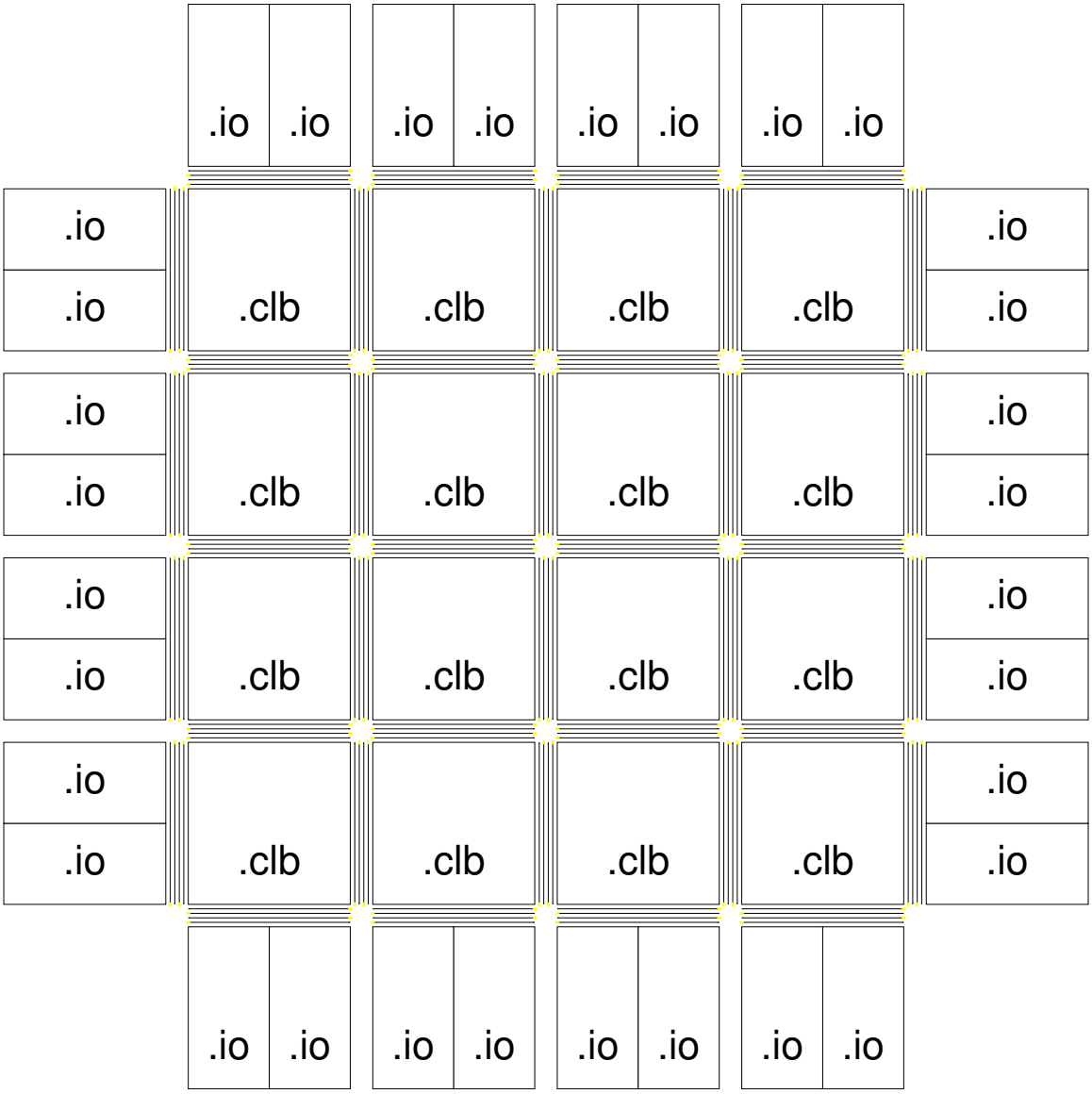


Figure 4.15: The basic organization of a FPGA. The three primary components are logic (the 4x4 array of clustered logic blocks (CLBs) in the middle), input/output blocks around the edge providing external connections, and the interconnect wires (shown here as lines in the gaps between blocks). Figure 4.16 shows a more detailed view of the structure of CLBs and the surrounding interconnect.

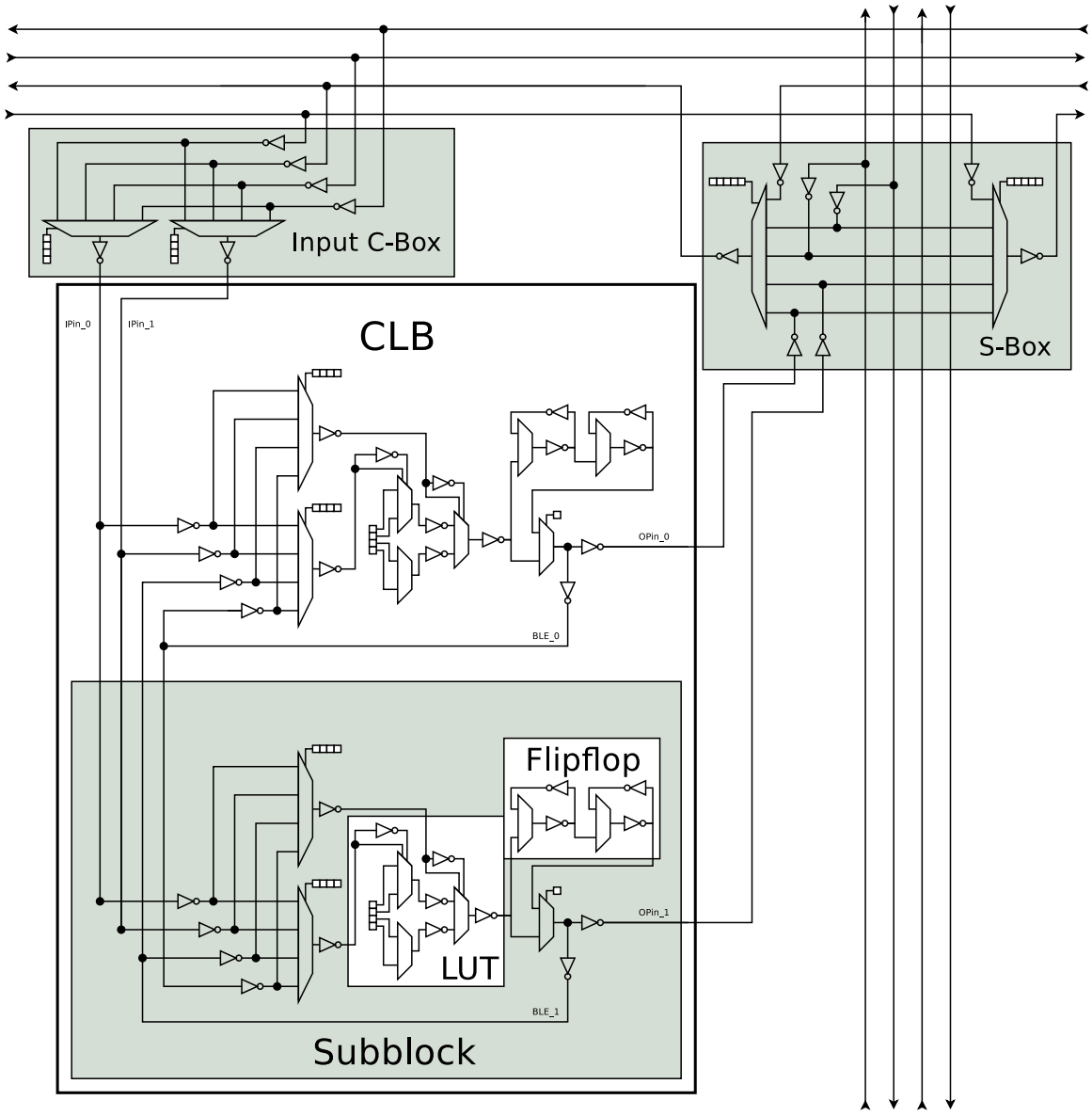


Figure 4.16: An example FPGA tile (a CLB and the surrounding interconnect) as modeled in VPR 5 [53]. This is the model used to specify connectivity and to compute area, delay, and energy in my simulations. The CLB and input C-Box structures shown here are common to both single and multi-driver interconnect architectures. The S-Box structure shown (including the incorporated output C-Box) is only representative of single-driver directional interconnect.

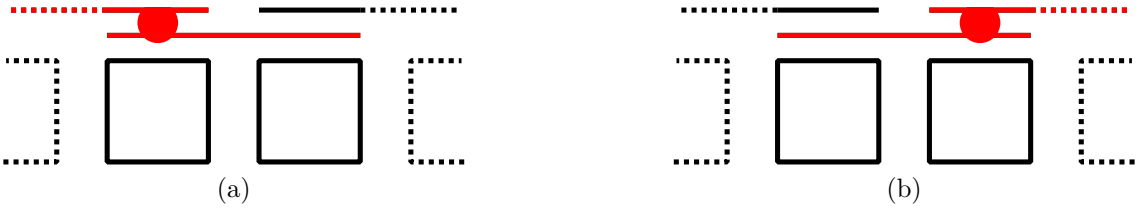


Figure 4.17: Portions of a staggered interconnect affected by bridges (shorting of wires) at different locations along a wire that spans multiple tiles.

Bridges (i.e., shorts) are sampled once per tile for each pair of parallel, adjacent wires spanning that tile. Figure 4.17 highlights the portions of a staggered interconnect (a common architectural practice) affected by bridges at different locations along a wire that is longer than a single tile. For single-driver interconnect (Figure 4.18), bridging two wires effectively shorts the output of two or more drivers, rendering both useless. For multi-driver interconnect (Figure 4.19), there is potential for partial use of bridged wires, if a single signal occupies both wires, but this adds to the capacitive load of the segment and would at very least slow signal propagation. Such partial use is beyond the scope of this dissertation.

Figure 4.20 shows the yield impact of these types of defects in the single-driver interconnect case. Here, I compare the yield of defect-free chips ("nominal" case), the yield of chips for which the base route happens to be functional without use of alternatives ("oblivious" mapping, comparable to EasyPath™ [42]), and the yield of chips which are successfully repaired by CYA.

4.7.3 Input C-Boxes

An input C-Box is modeled as buffers coming from channel wires feeding into a multiplexer (mux) that feeds into a CLB input pin. The yield impact of input C-Box defects, shown in Figure 4.21, comes from independent loss of individual inputs (effectively a stuck-open switch) to that mux.

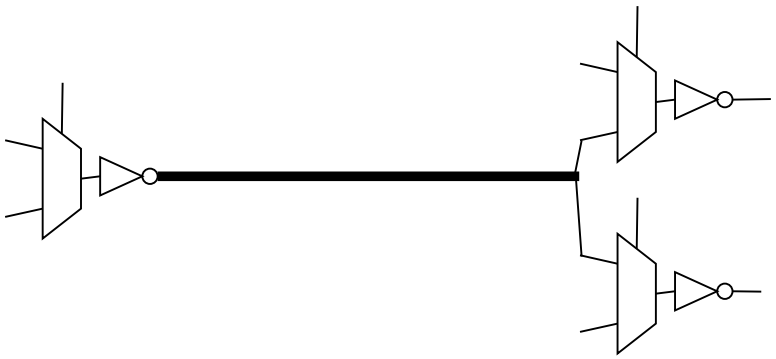


Figure 4.18: A directional (i.e., single-directional), single-driver segment. The propagated signal is selected by a multiplexer before the driver, rather than by choosing amongst several drivers. There are still multiple taps along the length.

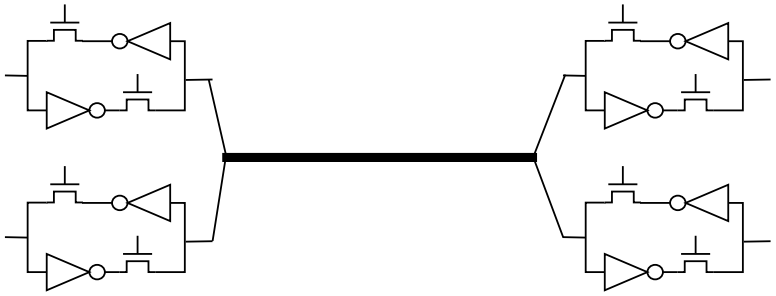


Figure 4.19: A bi-directional, multi-driver segment. Such a segment includes multiple bi-directional switches. Each driver is cut off if it is not used, so we can pick one of many drivers or use none, leaving the segment undriven.

4.7.4 S-Boxes and Output C-Boxes

The crossbars that drive channel wires in FPGAs with single-driver interconnect are S-Boxes which have inputs from both channel wires and CLB output pins. In the older, multi-driver, architectures, these two sets of inputs were often treated as separate crossbars, with S-Boxes for channel-wire-to-channel-wire connectivity and C-Boxes for CLB-to-channel-wire connectivity (and vice versa). Figure 4.22 shows the yield impacts of S-Box defects in a single-driver interconnect.

4.7.5 CLB Pins

Like channel wires, CLB pins are modeled as strips of conductive material. However, their role in the topology of the interconnect and their connectivity differ significantly. In addition, my simulations assume no bridging between CLB pins, only allowing connectivity breaks. Therefore, it is worth exploring the impact of CLB pin defects (Figure 4.23) independently from the impact of channel wire defects (Figure 4.20).

4.7.6 Logic (Subblocks)

Each subblock is modeled as a single entity in my simulations. Thus, a single random sample drawn for each subblock determines whether there are any defects in its resources, or whether all are perfect. Those resources include:

- an input crossbar (a set of buffers and muxes)
- a LUT
- a flip-flop or latch
- a mux to select between clocked and unclocked output from the LUT

- a loopback wire, effectively a CLB input pin driven by the subblock instead of by a selection of channel wires

The presence of any defects in these resources triggers the discard of the entire subblock, making it reasonable to use a single composite defect probability for the entire unit. Though internal repair of subblocks is beyond the scope of this dissertation, it should be noted that DeHon [16] demonstrated that there is considerable flexibility to work around LUT defects through a strategically-chosen set of simple configuration transformations.

The yield impact of these defects is shown in Figure 4.24.

4.7.7 All Resource Types

Figure 4.25 shows the combined impact of defects in all of the domains described in this section. The defect maps for each resource type are identical to those used in the preceding single-resource-type defect simulations.

Simulating the combined impact of all defect types illustrates that CYA can repair defects in all of the modeled types of resources. However, it is unrealistic to assume that each resource type fails with equal probability, meaning that the results in Figure 4.25 are likely somewhat skewed from what would be seen in real-world devices. To get a more realistic perspective, I shift from a per-resource notion of defect probability to a model in which we assume that defects are randomly but uniformly distributed across the die — a constant "defect density" rather than the previous assumption of a constant "defect rate".[4] To enact this shift, the defect threshold for the defect target corresponding to each resource type is scaled by its area. The area of each defect target is shown in Table 4.9. Figure 4.26 shows the yield impact on

[4] I use area models dominated by transistor area. Combined with uniform transistor size, this makes the "defect density" nearly equivalent to the transistor "defect rate".

Target Type	Tech Relative Area (λ^2)	Absolute Area (μm^2)
wire metal (break target)	1100	0.14
wire gap (bridge target)	1100	0.14
C-Box input	24	0.001
C-Box output	24	0.003
S-Box input	24	0.001
S-Box output	8	0.003
subblock	10000	1.2

Table 4.9: The area of each type of defect target, in both absolute units (μm^2) and technology relative units (λ^2), for a 22nm technology.

`des` of this shift to uniform defect density, and Figure 4.27 and Table 4.10 summarize these results across the Toronto 20 [6] benchmarks.

With this improved defect model in hand, Figure 4.28 and Table 4.11 show the overall effectiveness of CYA repair for all Toronto 20 designs across a range of defect densities. The top half of Figure 4.28 shows the yield for each design from 1000 simulated chips (with 64 alternatives available) and the bottom half of the figure shows highest number of alternatives used by any yielding chip for that design. The range of results across all designs suggests the degree to which alternative requirements and yield outcomes are likely to depend on the specifics of the design being mapped.

Table 4.11 shows the growth in CYA yield for each of the Toronto 20 designs as the number of alternatives increases, at a relatively high defect density of 800 defects/cm^2. At such a high defect density, not all designs achieve 100% yield even with 64 alternatives available, but the failure rates are still very low. Even for the worst design, `ex1010`, 995/1000 chips were ultimately repairable under these test conditions.

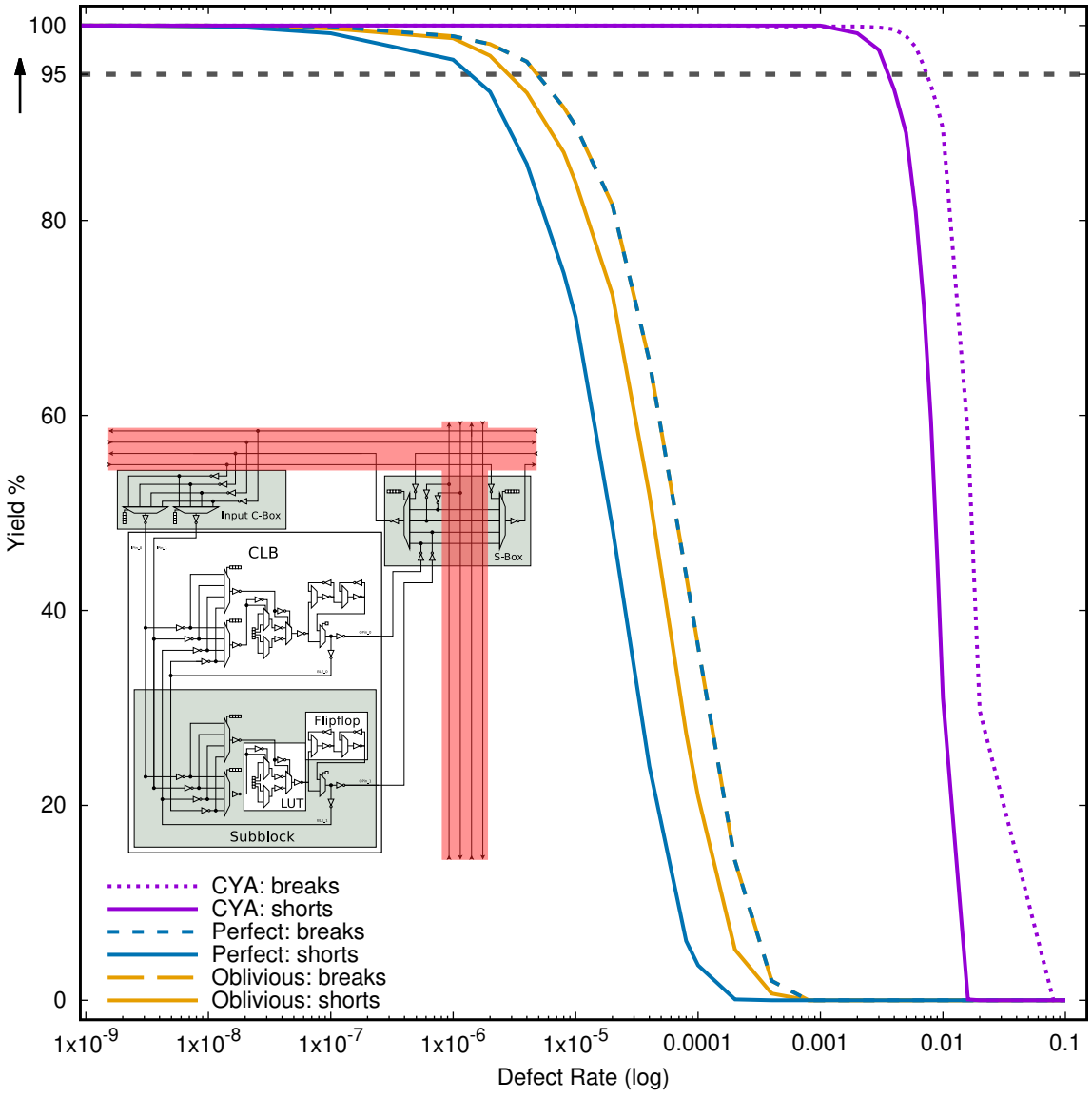


Figure 4.20: Yield vs. defect rate for nominal (perfect chips), oblivious (base route with no alternatives), and CYA routing, comparing the impacts of bridges (shorts) and breaks (`des`, 22nm technology, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 1000 chips).

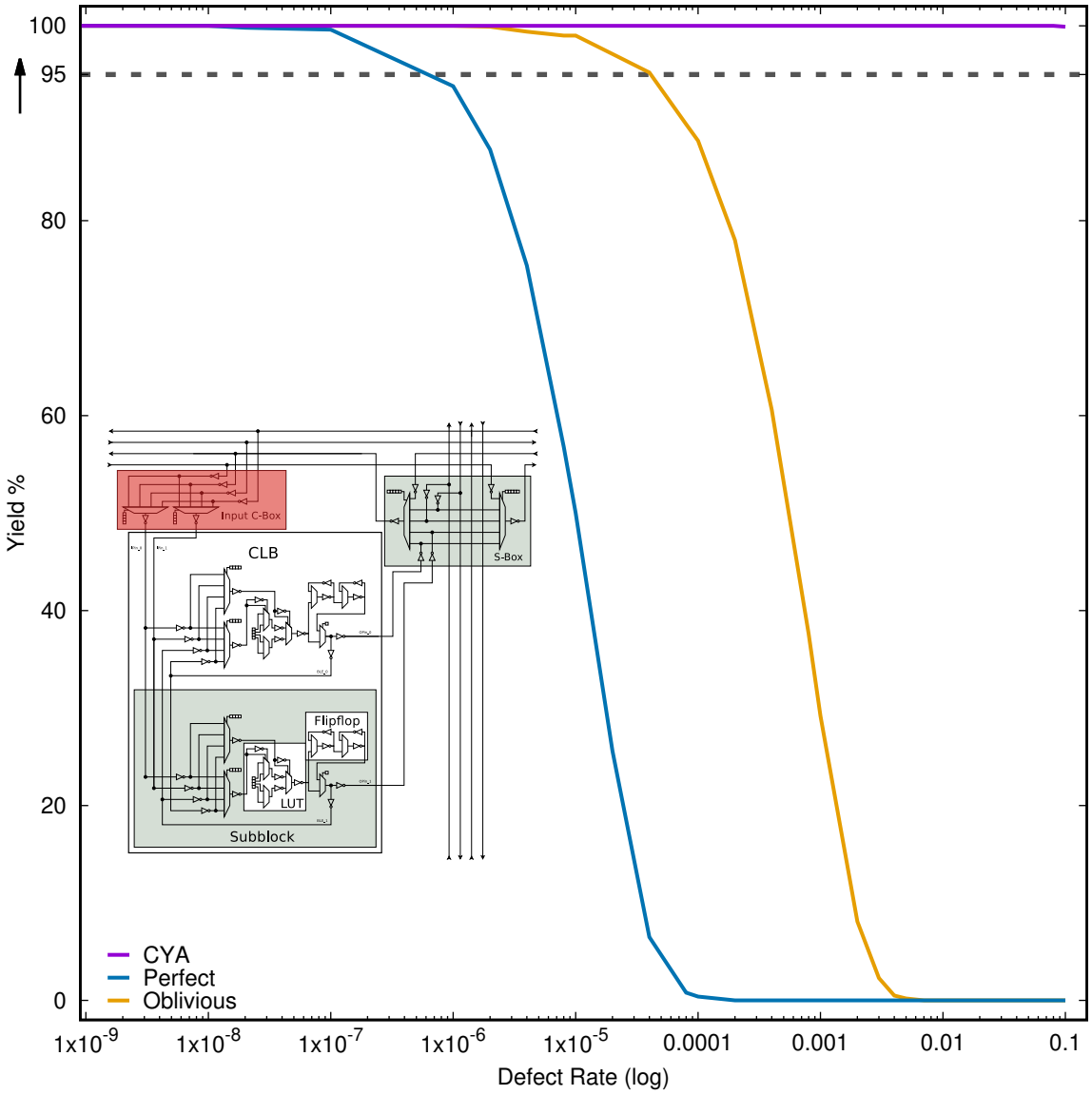


Figure 4.21: Yield vs. defect rate for defects in CLB input buffers (`des`, 22nm technology, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 1000 chips).

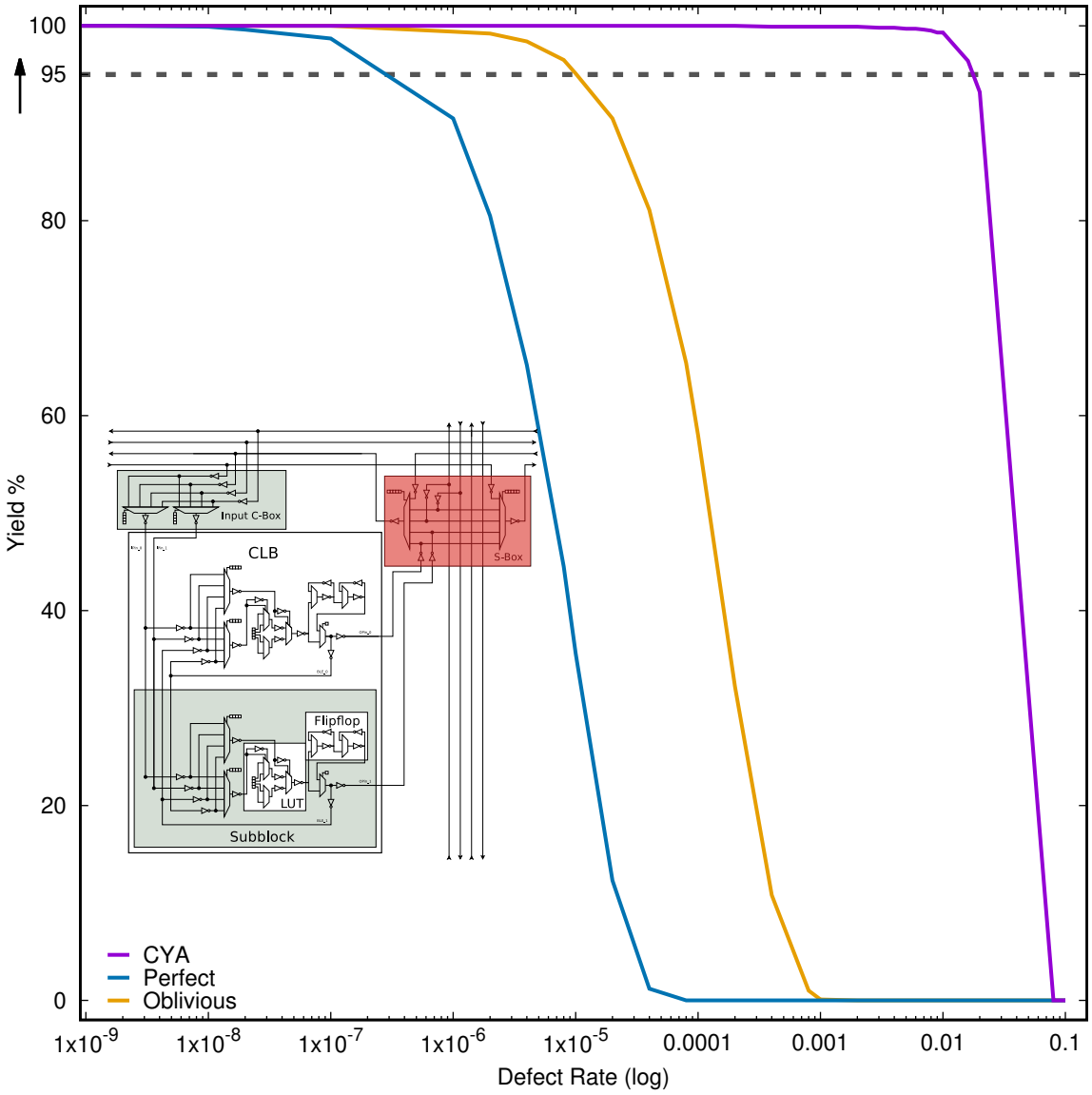


Figure 4.22: Yield vs. defect rate for defects in S-Box buffers (`des`, 22nm technology, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 1000 chips).

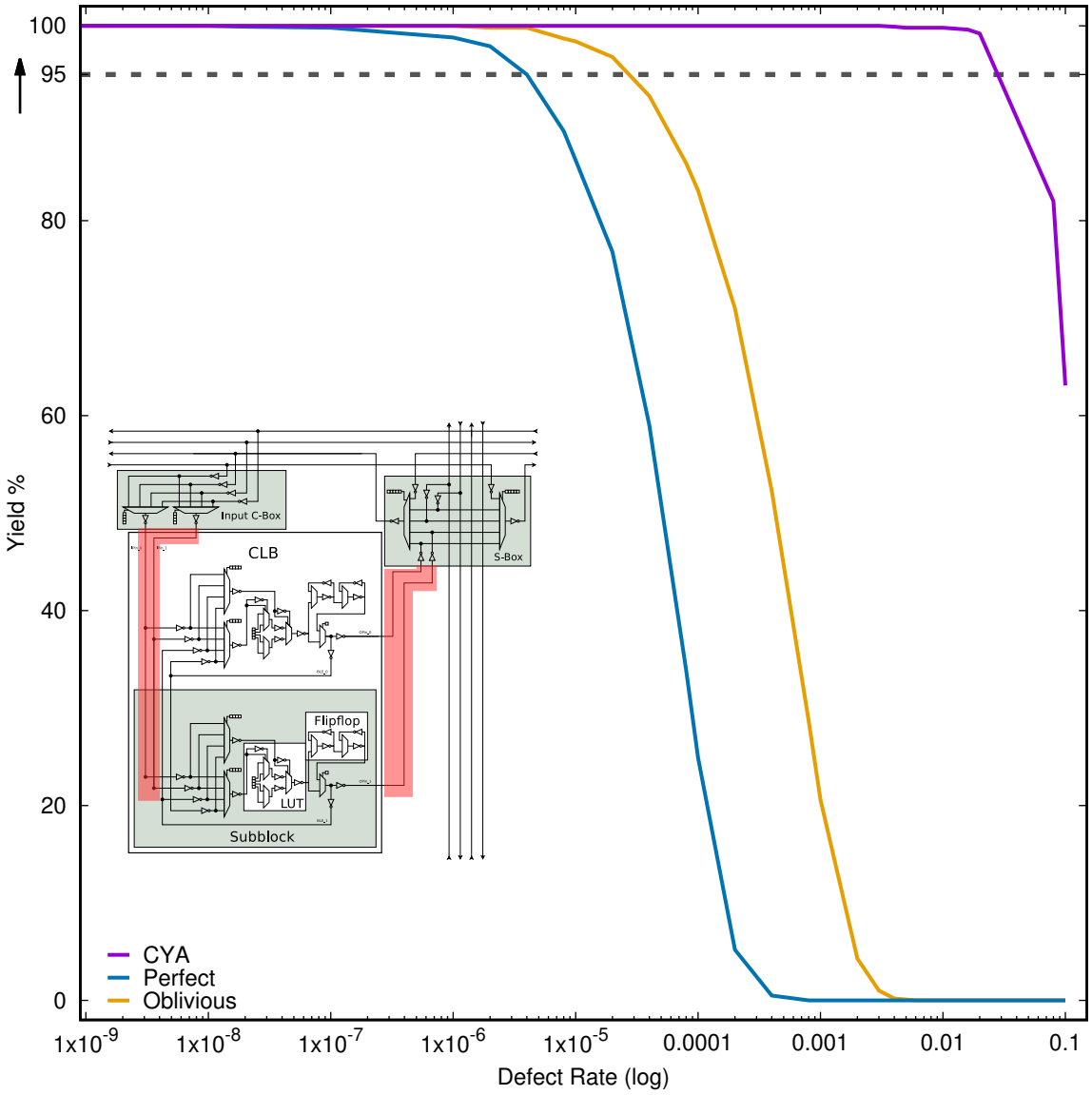


Figure 4.23: Yield vs. defect rate for wire breaks in the input and output CLB pins (`des`, 22nm technology, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 1000 chips).

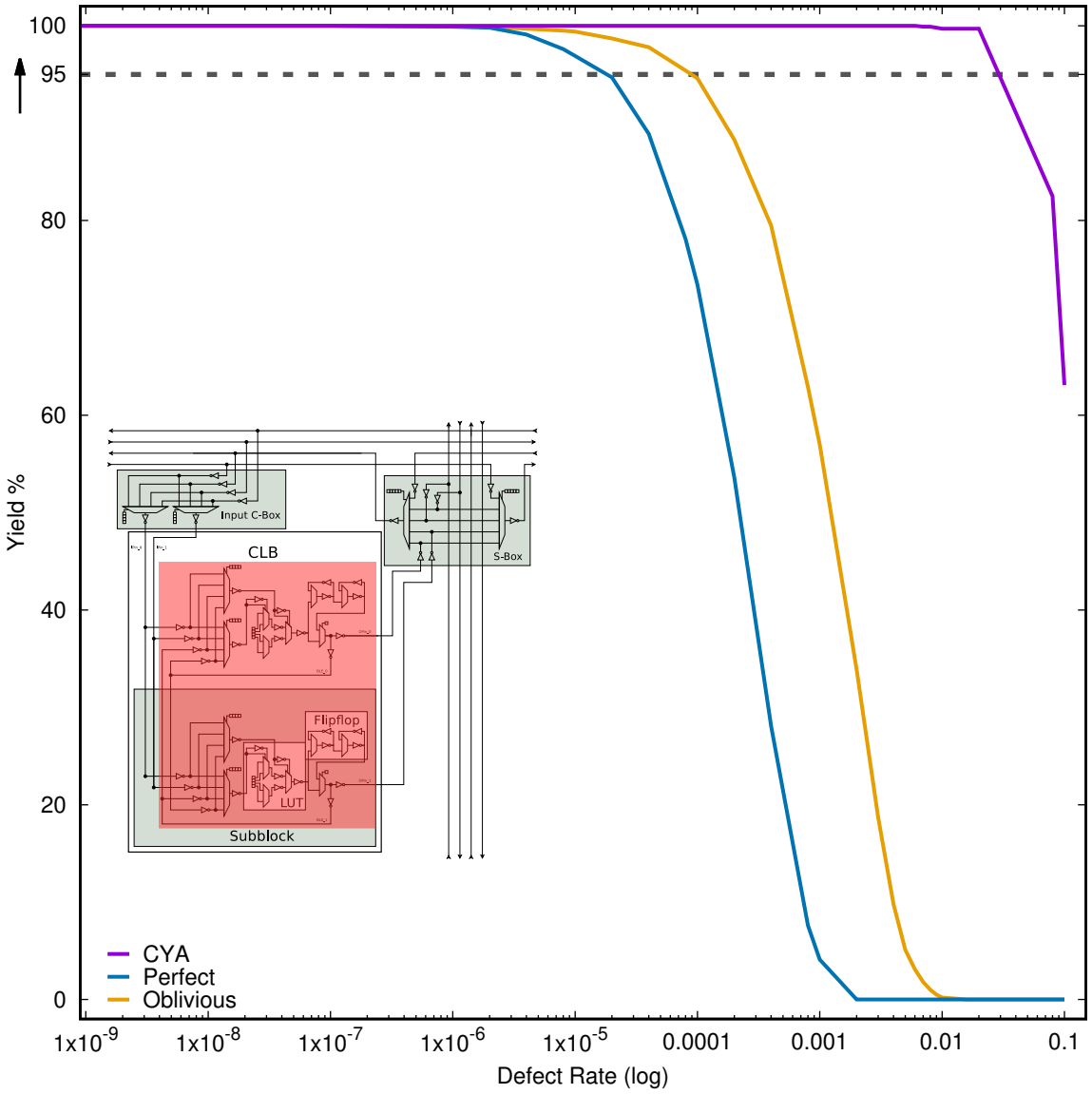


Figure 4.24: Yield vs. defect rate for dead subblocks (`des`, 22nm technology, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 1000 chips).

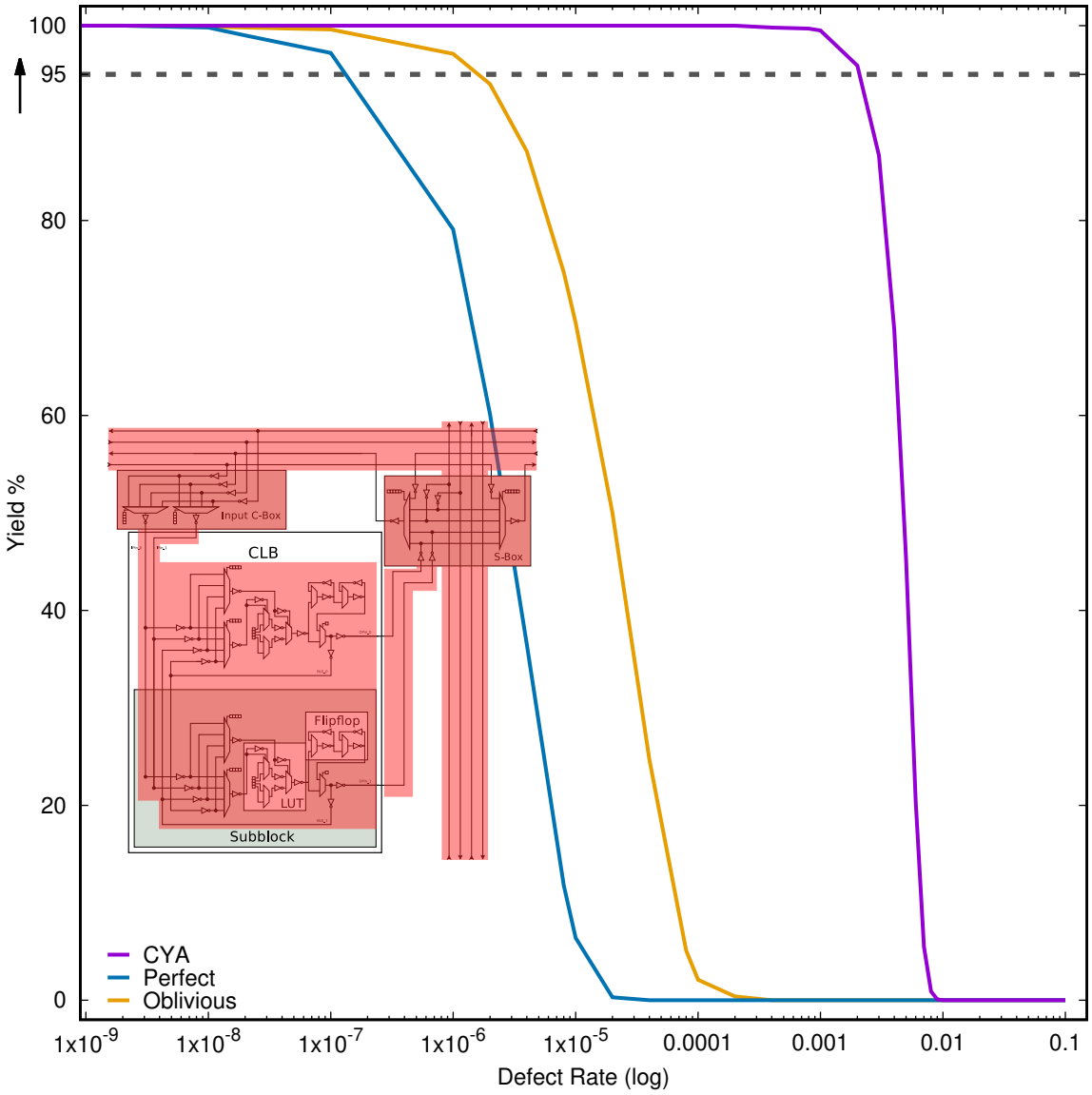


Figure 4.25: The combined impact of defects in all of the resource types modeled in Section 4.7 (`des`, 22nm technology, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 1000 chips).

In these simulations, the defect rate shown on the x-axis represents the identical, independent failure probabilities of each defect target (wire segment, switch, subblock, etc.), regardless of the physical area of the target. This is not intended to present a physically realistic failure model; rather, it is intended to summarize the combined effects of the individual failure types shown in Figures 4.20 to 4.24. See Figure 4.26 for a more realistic failure model.

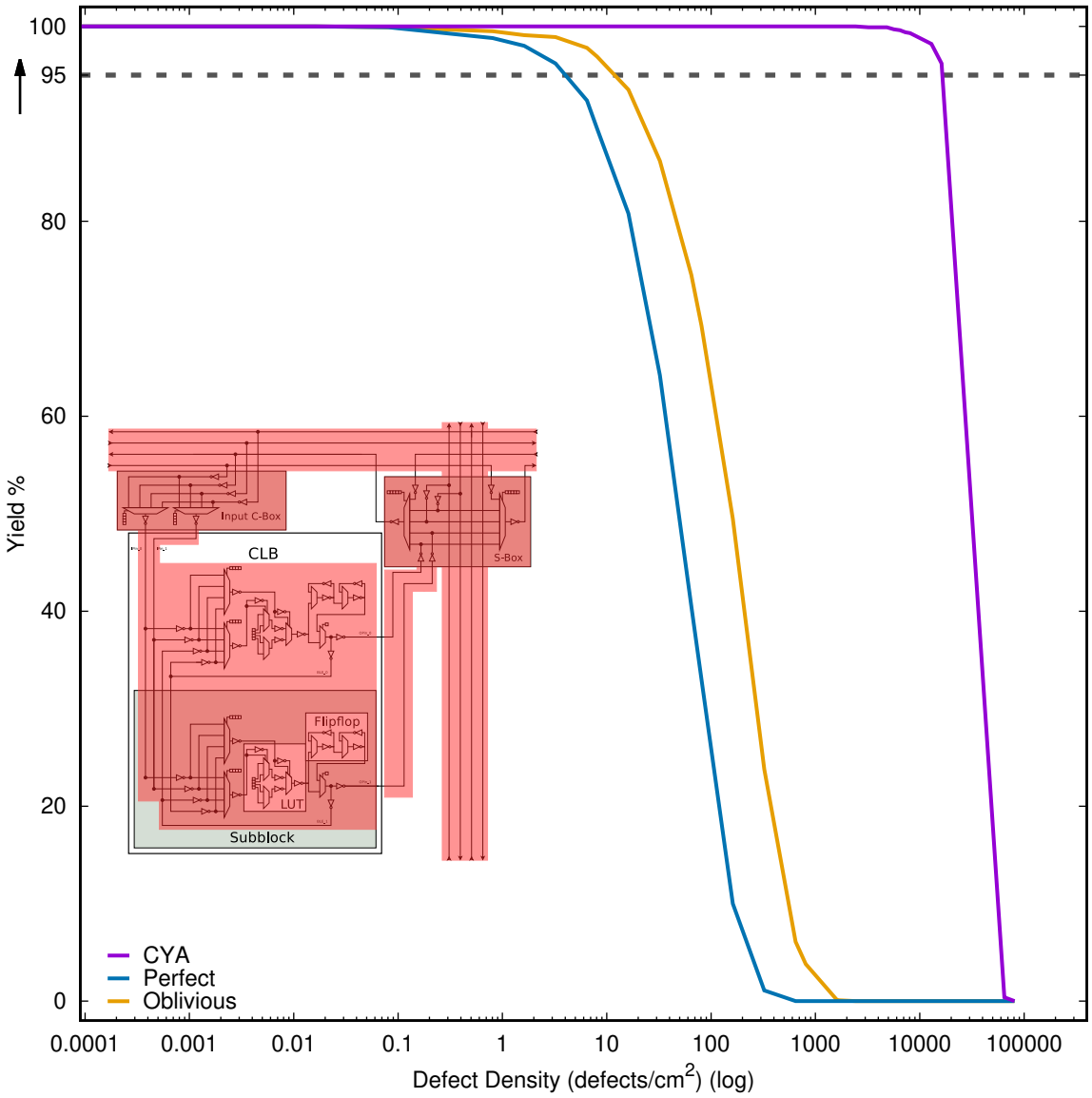


Figure 4.26: The combined impact of defects in all resource types (`des`, 22nm technology, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 1000 chips).

This is a more realistic version of Figure 4.25. Failure probabilities are scaled by the area of each defect target. (See Table 4.9 for the area estimates.) The x-axis shows the defect density in physical units (defects/cm^2).

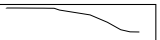
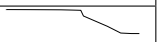
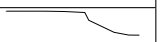
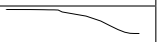
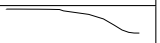
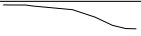
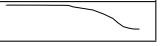
name	Defect Density (defects/cm^2) min	max	mean	median	95% yield	CDF
alu4	2400	48000	25000	24000	8100	
apex2	810	40000	17000	16000	8100	
apex4	810	32000	14000	16000	7300	
bigkey	810	81000	35000	32000	8100	
clma	160	8100	6000	6500	2400	
des	2400	65000	33000	32000	16000	
diffeq	1600	73000	30000	32000	8100	
dsip	1600	65000	31000	32000	8100	
elliptic	81	24000	9700	8100	4000	
ex1010	160	8100	5700	5700	2400	
ex5p	4800	48000	27000	24000	16000	
frisc	570	24000	9200	8100	4000	
misex3	320	48000	21000	24000	8100	
pdc	570	8100	6900	8100	2400	
s298	810	48000	20000	24000	8100	
s38417	570	32000	14000	16000	4800	
s38584.1	400	32000	14000	16000	7300	
seq	730	40000	18000	16000	7300	
spla	810	16000	7500	8100	4000	
tseng	810	81000	34000	32000	8100	
Mean	1070	41200	18900	19200	7150	

Table 4.10: Distribution parameters of maximum CYA-repairable defect density for each Toronto 20 design. (22 nm technology, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 1000 chips)

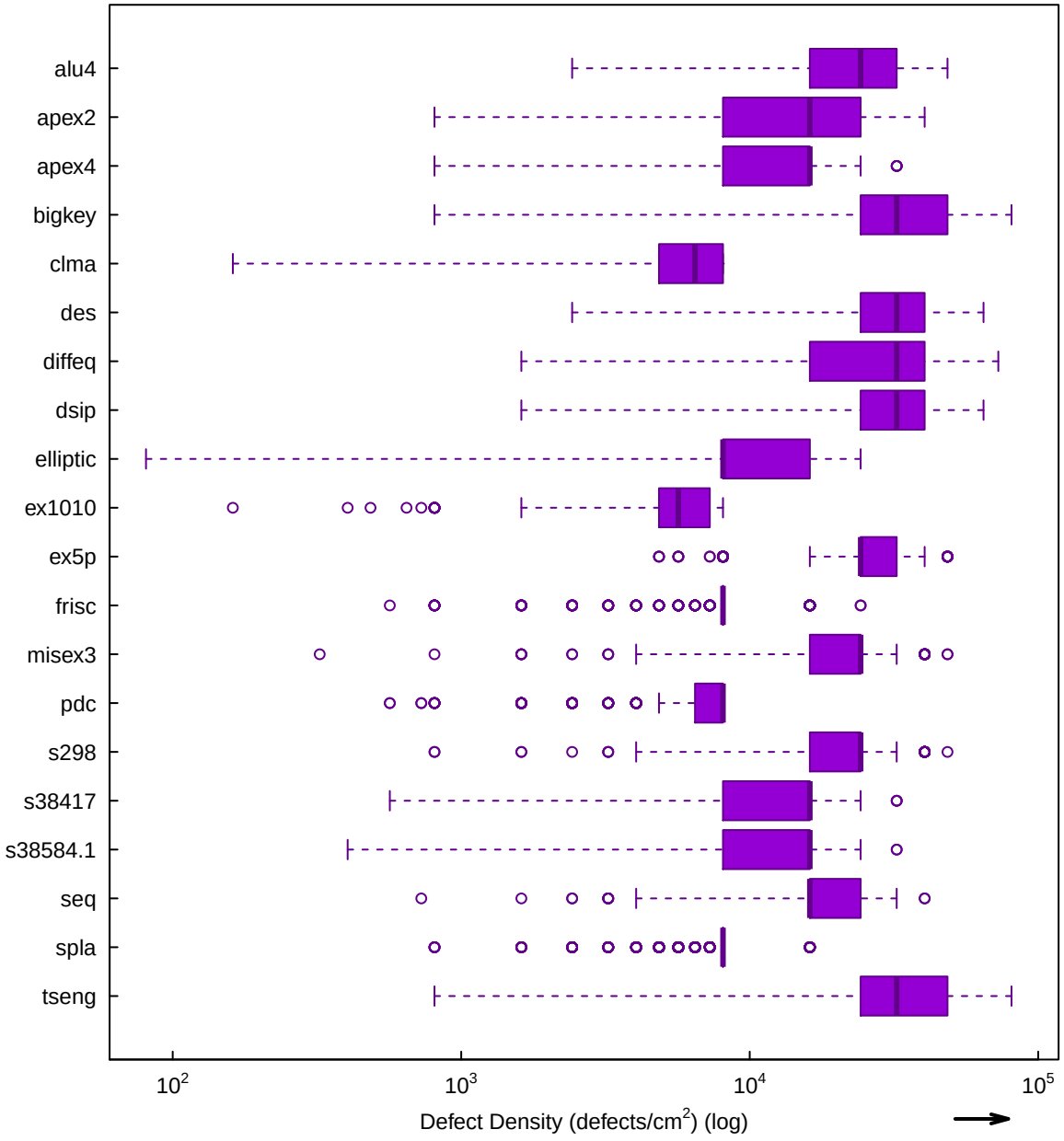


Figure 4.27: Distribution of maximum CYA-repairable defect density for each Toronto 20 design. (22 nm technology, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 1000 chips)

	# alternatives										
name	0	1	2	4	6	8	10	20	30	40	64
des	4	34	54	71	86	93	99	100	100	100	100
ex5p	9	48	64	79	86	94	98	100	100	100	100
s298	10	53	67	84	93	96	97	99	99	99	100
tseng	18	49	59	77	90	99	99	99	100	100	100
diffeq	12	48	65	78	89	99	99	99	99	100	100
bigkey	7	31	49	72	88	96	99	99	100	100	100
misex3	5	41	57	77	89	96	98	99	99	99	99
alu4	7	40	59	82	93	98	99	100	100	100	100
apex4	1	32	49	72	85	93	98	99	100	100	100
seq	2	34	53	74	88	96	98	99	99	99	99
dsip	2	28	44	69	85	95	98	99	100	100	100
apex2	1	34	53	74	85	94	98	99	100	100	100
elliptic	0	12	23	53	75	86	93	98	99	99	99
frisc	0	3	11	37	59	80	91	98	99	99	99
spla	0	3	11	38	66	81	93	98	99	100	100
pdc	0	0	3	28	57	73	87	98	99	99	99
s38584.1	0	2	8	27	50	85	97	99	99	99	99
ex1010	0	0	2	21	51	65	82	95	98	99	99
s38417	0	2	7	30	55	85	96	99	99	99	99
clma	0	0	3	24	50	71	89	98	99	99	99
Mean	4	25	37	58	76	89	95	99	99	99	99.9

Table 4.11: CYA yield improvement for Toronto 20 [6] benchmarks (sorted by size), updated to use VPR 5 [53] with single-driver interconnect and the full defect model. (Compare to Table 4.2 for VPR 4.3 [5]. See Table 4.6 for the updated benchmark parameters.) These data represent simulations of 1000 chips with 20% extra base tracks, 16 reserved tracks, and a defect density of 800 defects/cm^2.

Note that the percentages shown in this table are truncated rather than rounded; the lowest yield at 64 alternatives is `ex1010` with 995 successful chips out of the 1000 sampled. No benchmarks failed to route at defect densities lower than 81 defects/cm^2; the first failure occurs at this density in `elliptic`.

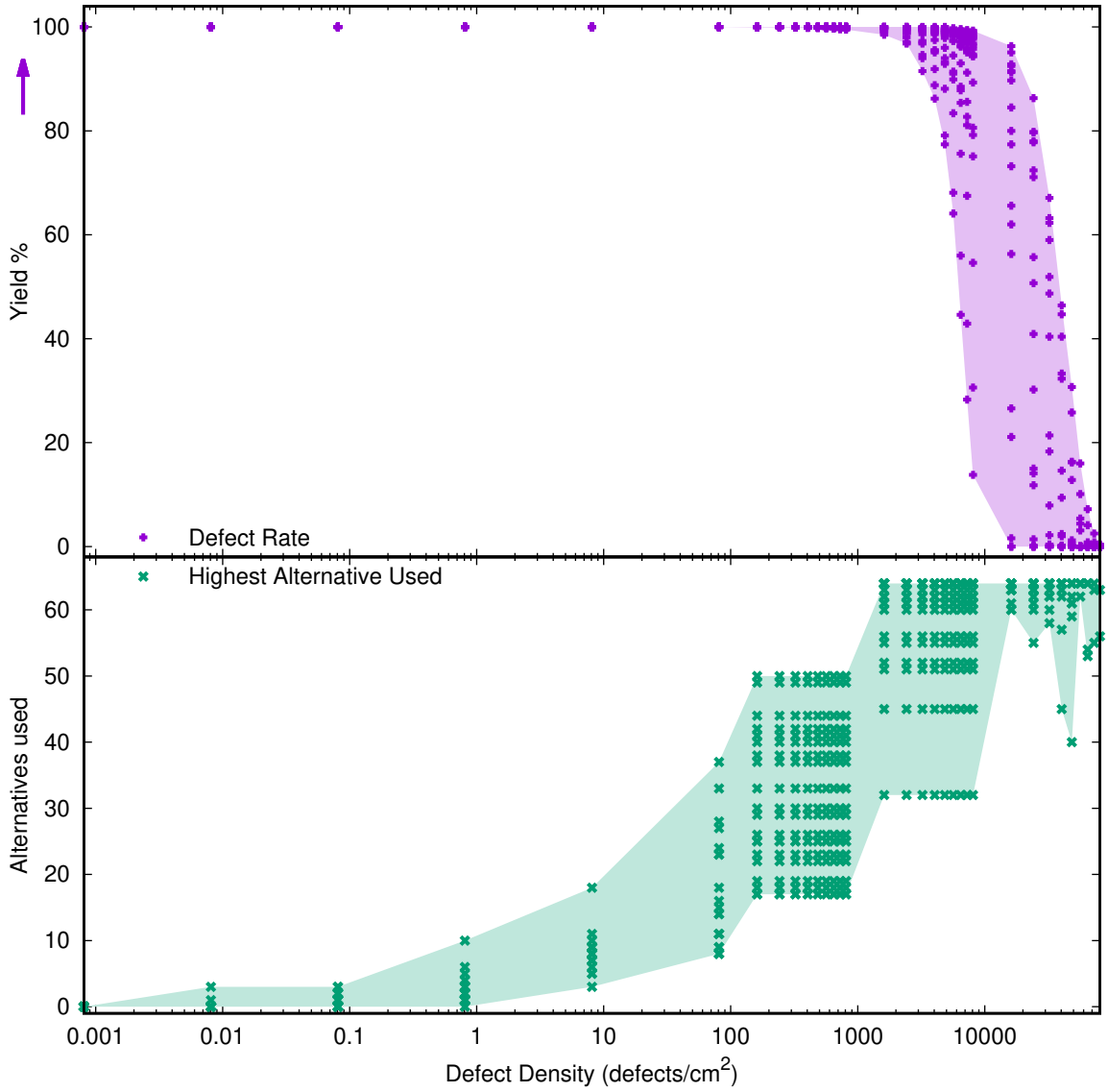


Figure 4.28: CYA repair effectiveness (yield and alternatives needed) vs. defect density, for all Toronto 20 [6] benchmarks (20% extra base tracks, 16 reserved tracks, 64 alternatives available, VPR 5 [53], single-driver interconnect, 1000 chips). Both graphs show points for each benchmark at each sampled defect density, along with a shaded envelope that encompasses all benchmarks.

The upper graph shows the fraction of 1000 simulated chips that were successfully configured in simulation. The lower graph shows the highest index of the alternatives used amongst the yielding chips, which is an estimate of the number of alternatives needed in the bitstream to achieve the yields shown above.

Chapter 5

Failure Modes and Defect Models

In this dissertation, I consider two classes of failure modes: functional faults (incorrect values, Section 5.1), and delay faults (late arrival of correct values, Section 5.2). For both defect types, I consider only unchanging physical conditions where the behavior of a circuit is consistent. Critical defects (e.g., power to ground shorts), intermittent circuits, single event upsets (e.g., particle strikes), environmental conditions, interference (e.g., crosstalk), and aging are beyond the scope of this dissertation. (Choose-Your-own-Adventure (CYA) as a foundation for addressing aging is explored in Giesen [22].)

The work discussed in this dissertation is entirely simulation-based; thus, physical fault testing methodologies are also beyond its direct scope. However, the defect (Section 5.1) and delay (Section 5.2.2) analysis techniques discussed in this chapter are manifestly capable of providing the hardware support needed for practical implementation of CYA. These techniques require no area overhead, and the timing results presented in [24] (and reproduced in part in Chapter 7) suggest entirely tolerable load time requirements.

The delay budgeting procedure described in Section 5.2.1 enables CYA to per-

form parametric repair (repair to a desired speed) of chips that run too slowly in the presence of variations (Section 5.2.3). This parametric repair in turn provides the foundation for the highly effective CYA-based delay and energy optimization procedures discussed in Chapter 6.

5.1 Functional Faults

At the end of a net (a "sink"), a quiesced circuit will produce either a 0 or a 1. This value can be seen as the output of a function, which, when correct, will be the output of the source block (or depending upon the architecture, the inverse of the value sent at the source) and nothing else. However, in the presence of faults, this expected output is only one of several possible types of functions the circuit could produce:

1. The expected function output
2. The exact inverse of the expected output (Section 5.1.1)
3. A constant 0 or 1 output, regardless of the source block output (Section 5.1.2)
4. A variable output, different from the expected function, which depends, in part or in its entirety, on other, unintended sources (Section 5.1.3)

Some of these failure modes may be somewhat physically unlikely; however, the purpose of this discussion is to ensure that we cover the full space of possible incorrect quiescent behaviors with a clear set of testing procedures.

5.1.1 Incorrect Inversions

Signal propagation can be confirmed by showing that the output toggles on the same cycles as the test input. Inversions can be detected by showing that the output is

always the inverse of the testing pattern (Figure 5.1). Alternately, as long as all transitions are verified, only a single value needs to be checked to determine whether the circuit is correct or inverted. Inversions may be treated as a defect, or may be remedied by a trivial permutation of the configuration of the down stream look-up table (LUT).

5.1.2 Outputs That Are Not Functions of the Source

If there is a disconnect between source and sink, the sink will not receive the transitions of the test pattern. This can be discovered by simply toggling the source and noting a lack of transitions at the sink.

5.1.3 Functions With Unintended Inputs

Now that we've considered circuits that are solely functions of the intended inputs, as well as those that produce outputs from no inputs, all that remains are cases where the output is at least in part a function of unintended inputs. For example, at a circuit level, a pair of bridged (i.e., shorted) wires may effectively produce an output that is a logical "or" of the outputs of two logic elements. Alternately, with always-driven segments in modern interconnect, the bridging of two segments may result in contention between two drivers, squelching transitions and only producing clean output when both signals are the same, effectively resulting in a logical "and".

In theory, we can detect these faults by exhaustive testing — transmitting all possible combinations of state from every logic element (LE), as well as all possible transitions between those states, and monitoring all sinks to validate correct behavior. However, this work is about practical implementations, and the exponentially exploding set of test cases for exhaustive validation is clearly not practical. Conse-

quently, this is a good point to step away from the abstract analysis and consider the concrete physical realities of the system.

Realistically, only physically adjacent nets will interfere with each other, reducing the number of interactions we must consider. Testing procedures can then focus on limited subregions of a design. This not only greatly reduces the number of test patterns to consider, but also creates opportunities for parallel testing. The key to parallel testing is to make sure that only paths with sufficient physical separation are tested concurrently. This will avoid direct cross-contamination between tests. However, Linscott [51] shows that indirect, higher-order interactions can skew the results of simultaneous tests of even distant circuits on the same die. Exploration of the implications of these results and development of corrective methodologies for parallel testing are beyond the scope of this dissertation. Thus, all results in this work address only sequential testing, but do take advantage of the ability to assume single-path tests.

Testing efficiency can also be improved by recognizing that it is extremely unlikely that structural flaws will produce arbitrary binary functions. "And" and "Or" are the only realistic faulty but functional (in the sense that the circuit switches at all) pathological behaviors to test [1]. This observation means that testing with all nets other than the net under test (NUT) held high, and again with all other nets held low, will expose these defects. A simple four-phase procedure can perform the necessary tests for each NUT. The four phases consist of low-to-high and high-to-low transitions of the NUT (the same two phases discussed in Section 4.2.1) performed twice, once with the potentially conflicting nets held low and again with those nets held high.

5.1.4 Example Test Circuit

Figure 5.1 provides a sketch of a test circuit which can detect functional defects. Defect detection with this circuit would use a multi-cycle testing procedure, in which the output of the NUT is compared with an input test pattern. A NUT with a correct transition schedule (output always changes with the test pattern such that it always matches or is always perfectly inverted) would be considered functioning or trivially repairable. Irreparable defects or variable influence from other sources would produce an altered transition schedule, i.e., a mix of matched and mismatched outputs.

Using this test circuit, the testing procedure begins by using the reset signal to set both output latches to 0. The desired sequence of testing signals is then sent through the NUT, and the test pattern and NUT output are used as inputs to the circuit. After the entire test pattern is complete, the state of the "Pass" and "Inverted" outputs determines whether the NUT should be considered functional/trivially repairable.

During testing, the input comparison XOR at the left produces a 0 if the output of the test circuit matches the reference test pattern and a 1 if it does not match. The upper latch is set and remains high if the comparison detects even a single mismatch during the entire testing process. The lower latch is set and remains high if the comparison detects even a single match. The circuit will pass testing (the "Pass" XOR) if and only if exactly one of these latches is high at the end of the test (i.e., the output always matched the test pattern or was always inverted), and will fail otherwise (a mix of matches and mismatches occurred).

Detection of any mismatches (upper latch) will also set the "Inverted" output flag. A perfectly inverted signal is trivially correctable with modifications to the logic tables stored in the affected LUTs. If the repair mechanism is not prepared to handle those corrections, the upper output latch and final XOR may be omitted and the output of the lower latch used as the sole measure of correctness.

This circuit could also be adapted to a delay fault testing context (Section 5.2.2) by adding registers between the test circuit and the comparison logic, to better isolate the delay of the test circuit from the measurement logic.

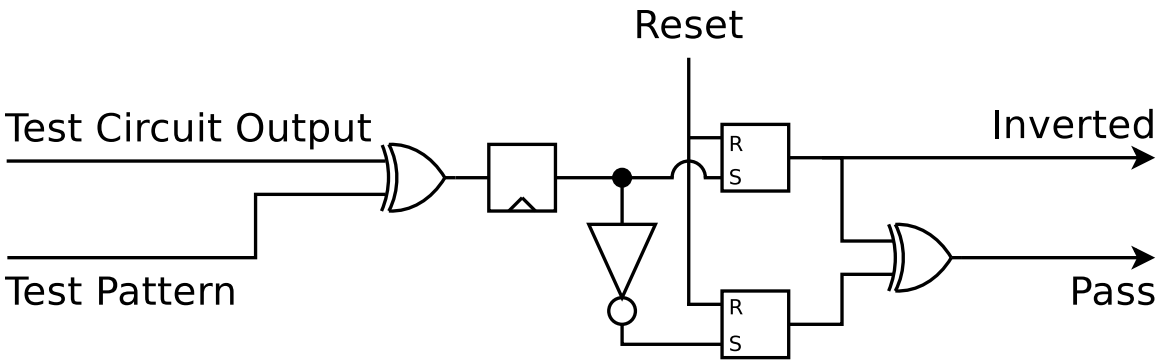


Figure 5.1: A testing circuit which confirms that the output transitions match the input test pattern and adds a flag to indicate whether the output is inverted or not.

5.2 Delay Faults

Tests of quiescent behavior are generally insufficient to ensure satisfactory performance in real-world circuits. In practical applications, we need to guarantee not only that signals are correct, but also that they arrive on time. A delay fault is a late signal, which may result in the propagation of an incorrect value to downstream circuits. A chronically slow circuit is considered a "parametric defect", because for some parameter values (in this case, certain clock speeds) it can act like a hard defect (i.e., it results in an incorrect or unchanging value during some clock cycles).

CYA's support for parametric defect repair demonstrates the power of its modular abstraction. Only two modifications were required to add this capability: a refinement of the definition of a good path (by including the delay budget for each two-point net in the bitstream, as noted in Section 4.2.1 and expanded in Section 5.2.1), and the addition of an updated testing circuit and procedure (to verify that each path is

fast enough). With these two modifications, CYA can repair a circuit to operate at a given clock frequency.

5.2.1 Delay Budgeting

Delay testing takes as input the delay budget for each two-point net. These budgets must be sufficient to guarantee that the overall design will work at the specified speed. Conveniently, a complete delay budget specification, including the design speed and predicted delay for each two-point net, are generated during normal routing.

The net delays produced by the router are the predicted delays from physical models. While the circuit will perform as expected if each net is no slower than these predictions, there is often a certain amount of additional "slack" time available to each net, allowing it to run more slowly than predicted without worsening the overall circuit performance. Figure 5.2 shows the slacks for each net in a sample design. We will always have one or more "critical paths", nets with no slack that determine the maximum speed of the design, but the amount of slack in the remaining nets is often considerable — in this example, almost all non-critical nets can slow down significantly without penalty. We want to represent this freedom in the delay budget, to avoid too-aggressively rejecting paths that are slightly, harmlessly, slower than their modeled delays.

Variables relevant to slack computations are defined in Table 5.1. For this work, I used a relatively simple slack-distribution algorithm similar to the Minimax-PERT algorithm in [94]. For each two-point net, $A_{a,b}$, we can determine the total available slack in terms of $D_{a,b}$, the delay of the longest "long-path" (the full distance a signal travels in one clock cycle) that passes through $A_{a,b}$. The difference between $D_{a,b}$ and the delay of the critical path, T_{crit}, is $s_{a,b}$, the total slack potentially available to $A_{a,b}$. However, this slack must be shared with all of the other nets along that long-path.

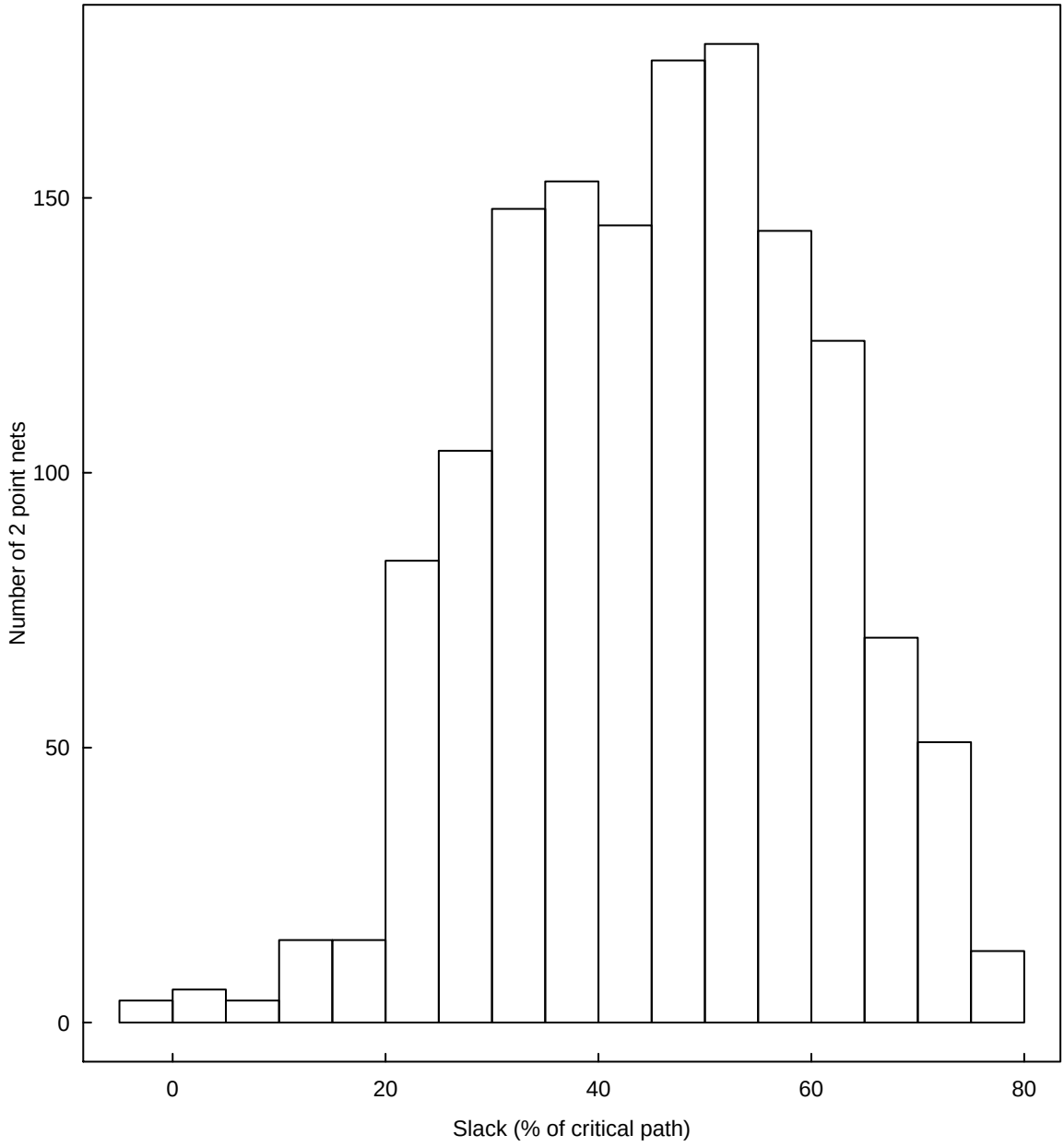


Figure 5.2: Histogram showing the slacks for all two-point nets in a nominal mapping of `des` at 0.8V (22nm technology, VPR 5 [53], single-driver interconnect).

T_{crit}	the delay of the critical path
$A_{a,b}$	the two-point net from source a to sink b
$\tau_{a,b}$	the delay of $A_{a,b}$
$p^j_{a,b}$	the jth "long-path" (the distance a signal travels in one clock cycle) on the list of all long-paths that include $A_{a,b}$
$d^j_{a,b}$	the delay of $p^j_{a,b}$
$D_{a,b}$	the maximum long-path delay through $A_{a,b}$ $(\max_j\{d^j_{a,b}\})$
$s_{a,b}$	the slack of $A_{a,b}$ $(T_{crit} - D_{a,b})$
$B_{a,b}$	the delay budget (permissible delay) for $A_{a,b}$

Table 5.1: Variables for slack calculations.

The simplest way to do this is to give $A_{a,b}$ a share proportional to its share of the delay $D_{a,b}$, as shown in Equation (5.1):

$$share\ of s_{a,b} = s_{a,b} \times \frac{\tau_{a,b}}{D_{a,b}}. \tag{5.1}$$

We then add this on to $A_{a,b}$'s nominal delay, $\tau_{a,b}$, to get an initial estimate for $A_{a,b}$'s delay budget:

$$\begin{aligned} B_{a,b} &= \tau_{a,b} + share\ of\ s_{a,b} \\ &= \tau_{a,b}(1 + \frac{s_{a,b}}{D_{a,b}}) \\ &= \tau_{a,b}(1 + \frac{T_{crit}}{D_{a,b}} - 1) \\ &= \frac{\tau_{a,b} T_{crit}}{D_{a,b}}. \end{aligned} \tag{5.2}$$

Since $s_{a,b}$ is the *minimum* of the slacks of the set of paths $\{p_{a,b}\}$ that pass through $A_{a,b}$, undistributed residual slack remains on less-critical paths. These residual slacks are distributed by repeated iteration of this process until they are reduced to insignificance

(less than $\frac{\tau_{a,b}}{10^7}$ in my experiments), as shown in Equation (5.3) and Algorithm 5.1.

$$B^0_{a,b} = \tau_{a,b} \tag{5.3a}$$

$$B^i_{a,b} = \frac{B^{i-1}_{a,b} T_{crit}}{D^{i-1}_{a,b}} \tag{5.3b}$$

$$\lim_{i \to \infty} B^i_{a,b} \to B^{i-1}_{a,b} \tag{5.3c}$$

Algorithm 5.1: Iterative Slack Distribution Algorithm

```
{settled nets} ← {};

foreach Net N ∈ {all nets} do
    B_N ← D_N;

while ∃_N ∈ {all nets}|N ∉ {settled nets} do
    foreach Net N ∈ {all nets}|N ∉ {settled nets} do
        share ← (s_N * B_n) / (T_crit − s_N) ;
        if share ≥ ε * D_N then
            B_N ← B_N + share;
        else
            {settled nets}.Add(N);
    Update all slacks();
```

The primary design goal for this slack spreading routine was to avoid egregious slack waste while converging with (empirically) negligible runtime. All trials took under 2 seconds, and most were less than 1 second. A more sophisticated algorithm design might provide even greater benefits, particularly in light of existing literature on the subject, but is beyond the scope of this dissertation.

5.2.2 Test Circuit and Procedure

Path delay testing within field-programmable gate arrays (FPGAs) is a well established practice. For example, Harris [30] shows a circuit (Figure 5.3) that compares

the state on the input and output of a flip-flop, at the end of the test circuit, to see if a transition arrived too late to be captured. More refined versions of this technique are used to measure post-fabrication variations in [88] and Gojman [26]. Gojman demonstrates a precision of 1.6 ps and an accuracy of 10–30 ps for path delay measurements on 20 real-world Altera Cyclone III 65 nm FPGAs (see Figure 5.4).

Existing circuits provide adequate testing to satisfy the delay testing requirements of CYA. Additional modeling and simulation of testing circuits is beyond the scope of this dissertation.

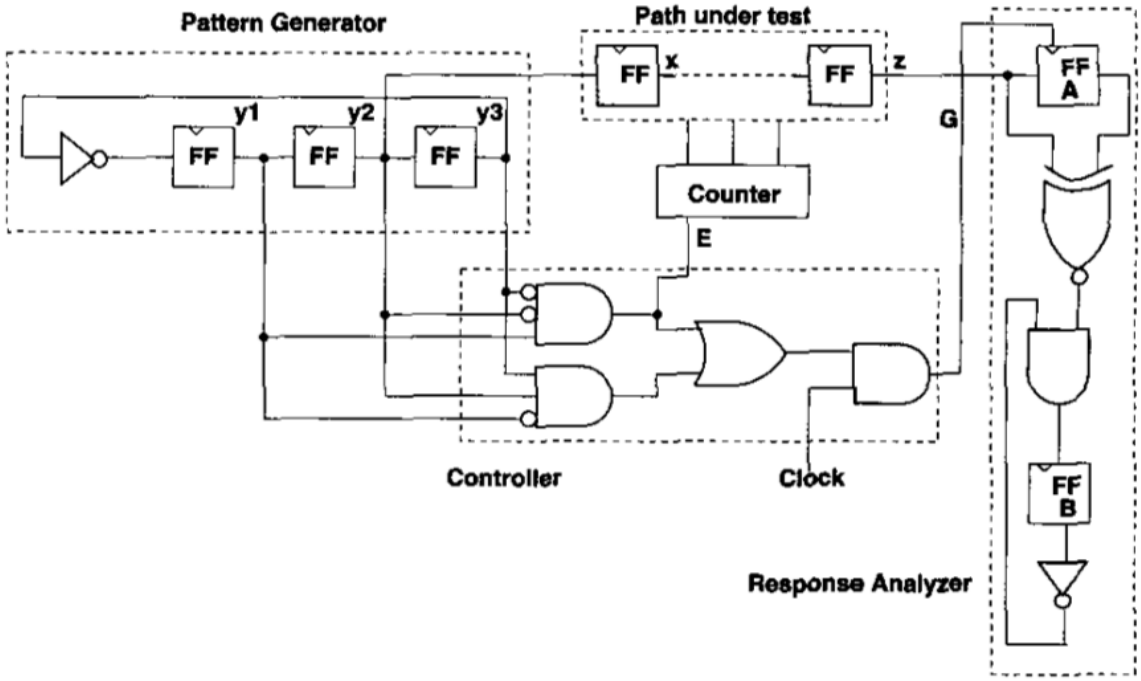
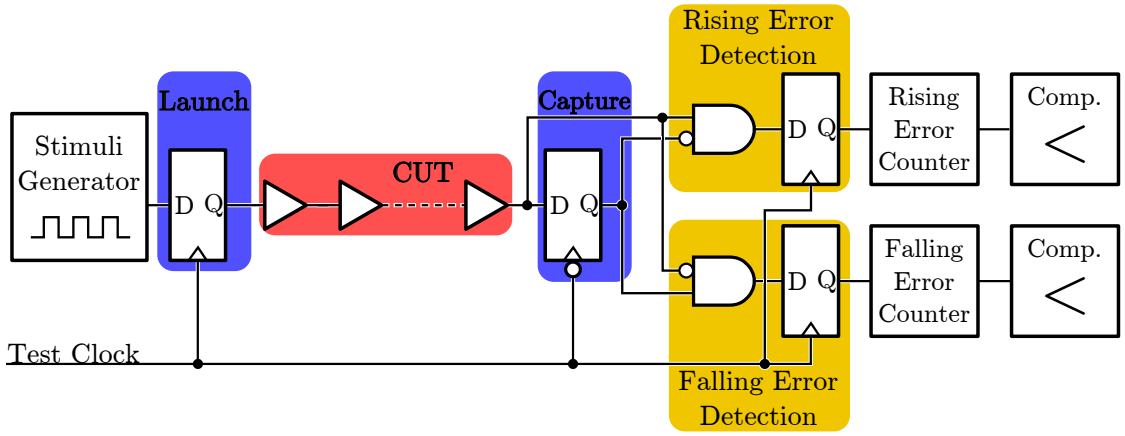
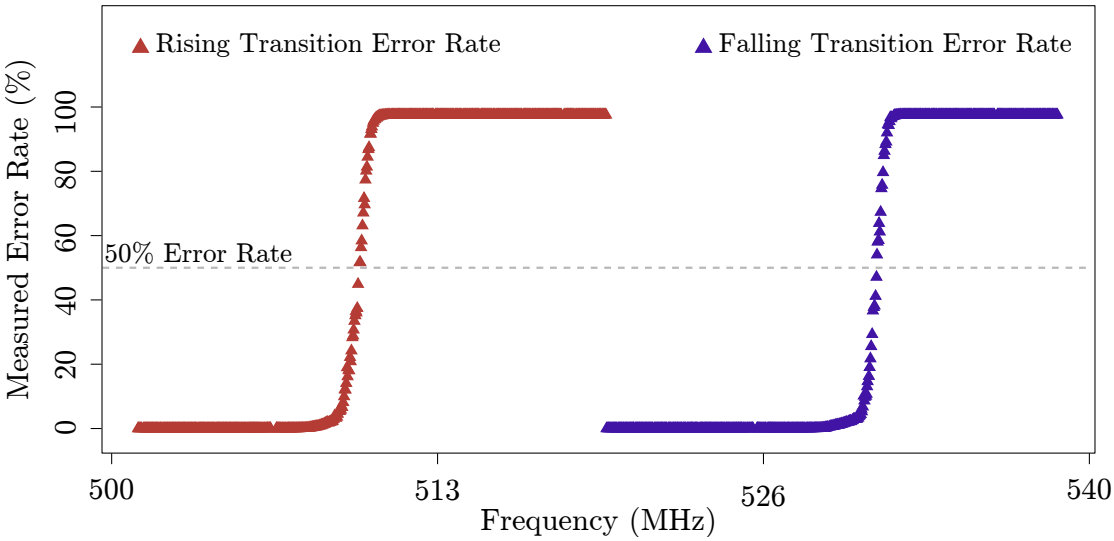


Figure 5.3: Circuit used by Harris [30] for delay self-testing. This circuit can be implemented in ordinary FPGA logic, injected solely for testing, and removed for normal operation. (Image from Harris [30].)



(a) Path-delay measurement circuit from Gojman [26].



(b) Measured rising and falling transition error rates for a path. (Image from Gojman [26].)

Figure 5.4: Gojman delay measurement circuit schematic and a sample of real delays (measured on an Altera Cyclone III).

5.2.3 Single Frequency Delay Results

Figures 5.5 to 5.7 demonstrate that CYA with delay fault repair is, in fact, capable of improving parametric yield (the portion of the simulated chips that are able to function at a given speed). These graphs compare delay yield results for:

1. Nominal: static mapping on perfect chips
2. Oblivious: those same static maps in the presence of variations, with delays optimized using dynamic frequency scaling (DFS)
3. CYA nominal: delay-fault repair CYA on perfect chips, using the above static mapping as the base route
4. CYA: delay-fault repair CYA in the presence of variations, also delay-optimized using DFS
5. CYA normalized: delay-fault repair CYA results scaled by the factor (nominal delay)/(CYA nominal delay)

The purpose of the normalization in Item 5 is to account for a slight systematic error in my experiments that results from the need to implement split channels to enable reservation of resources in the Wilton switch box (S-Box) topology used by VPR 5 [53]. Without this correction, CYA receives a slight unfair delay advantage relative to static mapping, but energy results and comparisons with other CSM methods (Chapters 6 and 7) are not significantly affected. See Appendix A for further details.

As shown in Figure 5.5, CYA's delay reduction is relatively small when variations only have minor effects on performance, as is true at the standard operating voltage for this 22nm technology (V_{dd} = 0.8V). This graph suggests a speedup in operating frequencies of only 1-4%.

This situation changes completely under conditions where variations are more significant, as is true both at lower operating voltages (useful for energy efficiency), and for technologies with smaller feature sizes. The former case, with $V_{dd} = 0.7$V and $V_{dd} = 0.6$V, is shown in Figures 5.6 and 5.7. CYA both recovers much of the delay lost to variations, closing the gap by a median of 66–87%, and restores the ability to achieve 100% yield, even as 2.6–5% of the statically-mapped chips fail to work at any speed. Note that the systematic error, as measured by the gap between the nominal and CYA nominal lines, also decays into insignificance as the voltage is lowered. The advantages of CYA under high-variation conditions become particularly relevant when we are interested in optimizing the energy and delay of each chip. This is the focus of Chapter 6.

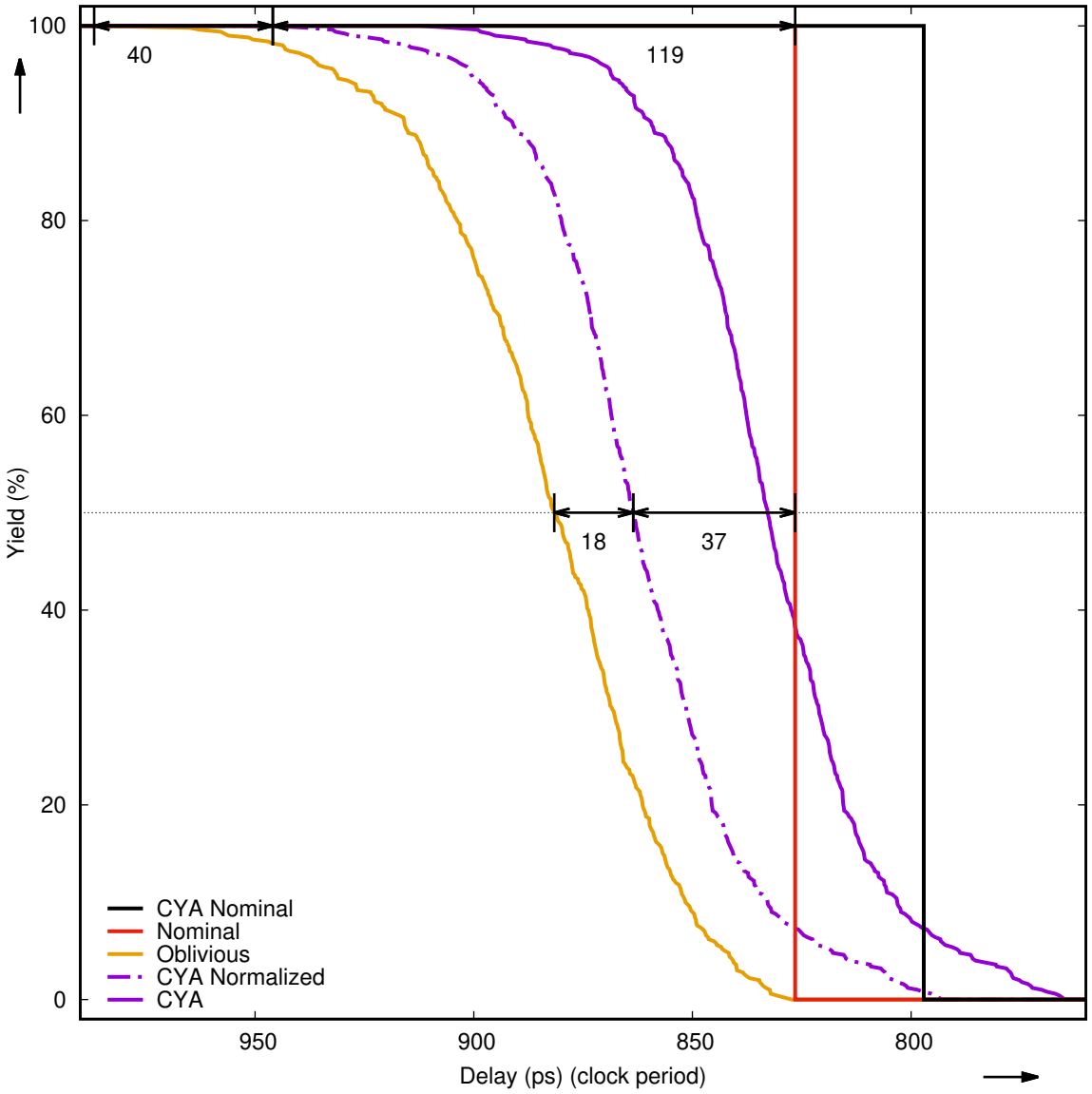


Figure 5.5: At 0.8V, defects slow static mapping by a median of about 6% for `des` (oblivious vs. nominal lines). CYA (normalized) improves upon the oblivious delay by a median of about 2%. Normalized CYA delay values are scaled using the corrective factor (nominal delay)/(CYA nominal delay), to account for a slight systematic error resulting from the implementation of split channels (see Appendix A).

All 500 chips are simulated in VPR 5 [53] using single-driver interconnect, 22 nm technology, and $\sigma_{V_{th}} = 0.0364$V. Mappings are generated with 20% extra base tracks and 16 reserved tracks, and CYA uses 64 alternatives.

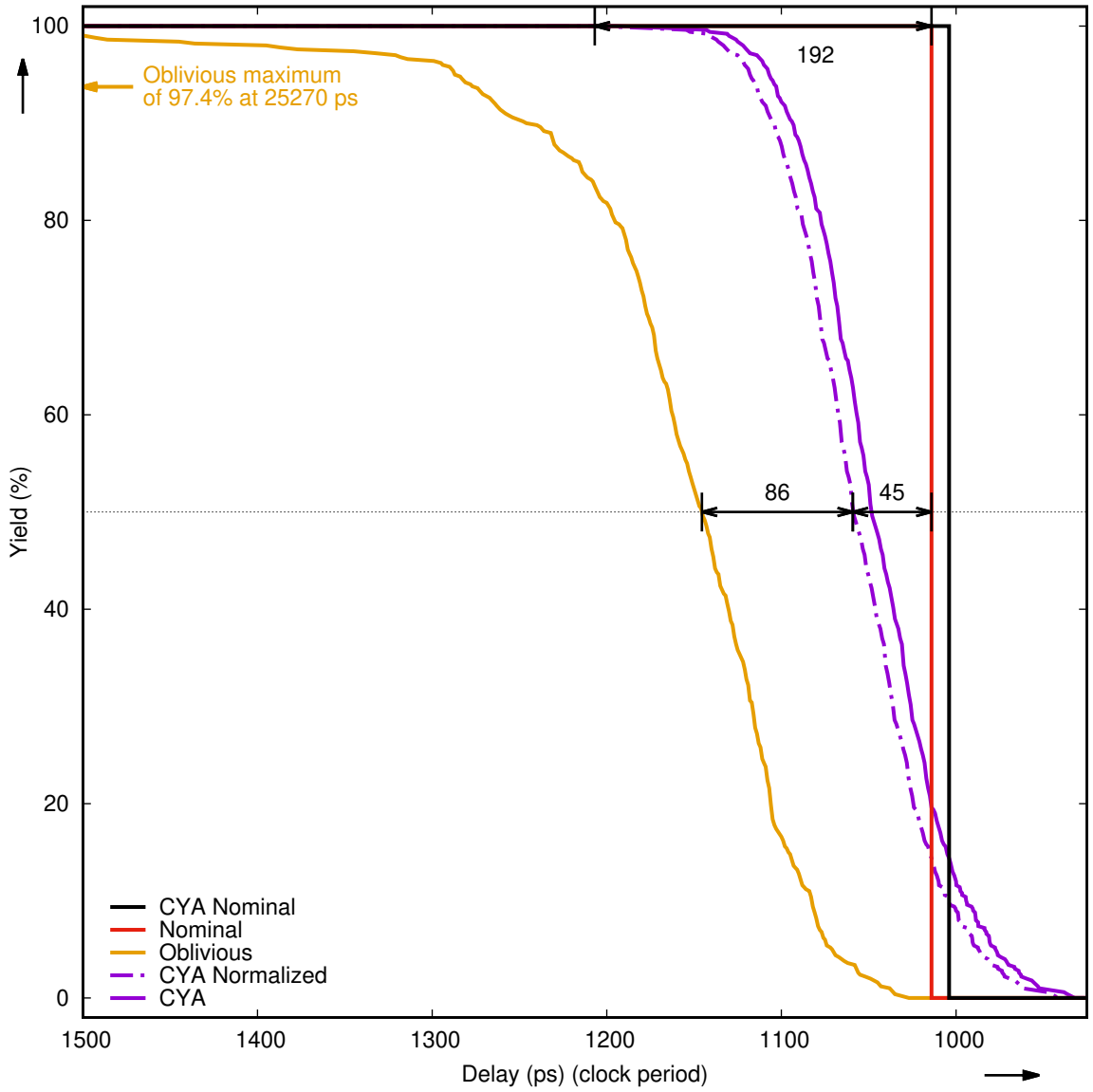


Figure 5.6: At 0.7V, V_{th} variations begin to significantly affect the operating frequency of `des` running on the obliviously loaded chips. 2.6% of chips fail entirely and do not work at any speed. Note that the impact of the systematic delay error (Appendix A) has also decreased significantly for this design at this voltage.

All 500 chips are simulated in VPR 5 [53] using single-driver interconnect, 22 nm technology, and $\sigma_{V_{th}} = 0.0364$V. Mappings are generated with 20% extra base tracks and 16 reserved tracks, and CYA uses 64 alternatives.

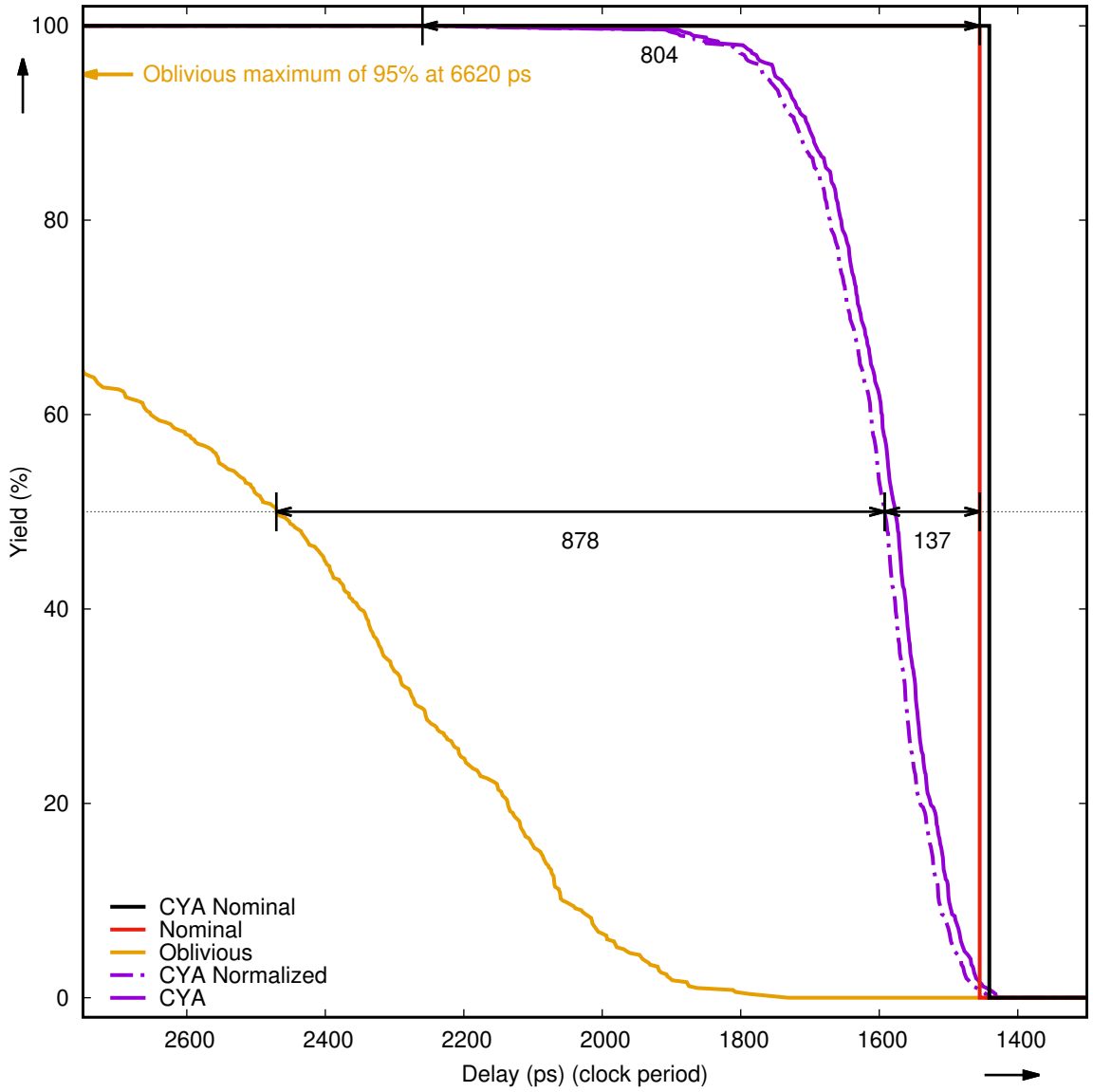


Figure 5.7: At 0.6V, V_{th} variations have an even greater impact on the operating frequency of `des` running on the obliviously loaded chips. 5% of chips now fail entirely. In contrast, CYA repair continues to achieve full yield, albeit at the cost of an approximately 55% increase in delay relative to the nominal case. The systematic delay error (Appendix A) for this design continues to decrease with decreasing voltage.

All 500 chips are simulated in VPR 5 [53] using single-driver interconnect, 22 nm technology, and $\sigma_{V_{th}} = 0.0364$V. Mappings are generated with 20% extra base tracks and 16 reserved tracks, and CYA uses 64 alternatives.

Chapter 6

Delay and Energy Optimization with CYA

In this chapter, I demonstrate the flexibility of Choose-Your-own-Adventure (CYA) as a platform for achieving complex repair and optimization goals. Specifically, I describe how to use CYA to reduce delay and optimize energy usage in the presence of variations.

6.1 Delay Optimization

Section 5.2 describes a CYA-based methodology for addressing parametric defects (i.e., repair to a target speed). The success or failure of any given repair attempt (for a particular circuit on a particular chip at a particular speed) can be treated as a test and used as part of an optimization process (e.g., a binary search) to find the fastest possible speed at which the circuit can be run on that chip. The results of such an optimization process have already been shown in Figures 5.5 to 5.7 of Section 5.2.3 — the parametric yield versus clock period results were derived in this way.

Note that the 0.8V results in Figure 5.5 showed that, at a variation rate appropriate to the 22nm technology modeled, delay-optimizing CYA reduced median delay for `des` by a modest 2%, relative to the results of dynamic frequency scaling (DFS) with a single (oblivious) configuration. This amounts to recovery of about a third of the slowdown attributable to variations. The difference is more pronounced, however, when variations increase, as occurs when moving to smaller feature sizes, or when V_{dd} is reduced. For example, with V_{dd} adjusted to 0.7V, median recovery of lost delay by CYA was close to 2/3, and CYA continued to enable 100% yields (at slower speeds), even as oblivious mapping failed entirely on 2.6% of chips (Figure 5.6). At 0.6V, median delay recovery by CYA increased to 87% and overall CYA yield remained at 100%, while oblivious mapping failure rates increased to 5% (Figure 5.7).

This suggests that delay-optimizing CYA can provide significant benefits for both speedup and yield in newer, higher-variation technologies, as well as for energy optimization (through voltage reduction) in both older and newer technologies. Figures 6.1 and 6.2 show an example of these optimizations acting on a single chip — delay-optimizing CYA provides greater and greater delay and energy benefits relative to both oblivious mapping (static mapping with DFS) and defect-only CYA (with DFS) as the voltage is decreased.

Note that defect-only CYA continues to yield on this chip well below the voltage where oblivious mapping fails. However, this does not result in meaningful energy improvements on its own because, without delay optimization, CYA is content to select resources just as slow as those used by the oblivious map. With delay optimization, CYA achieves a minimum energy (at 0.55V) of 41% lower than the minimum achieved by both oblivious mapping and defect-only CYA (which occurs at 0.7V). This suggests that we may, in general, be able to obtain significant energy savings using a CYA-based energy optimization strategy, as I will explore further in Section 6.2.

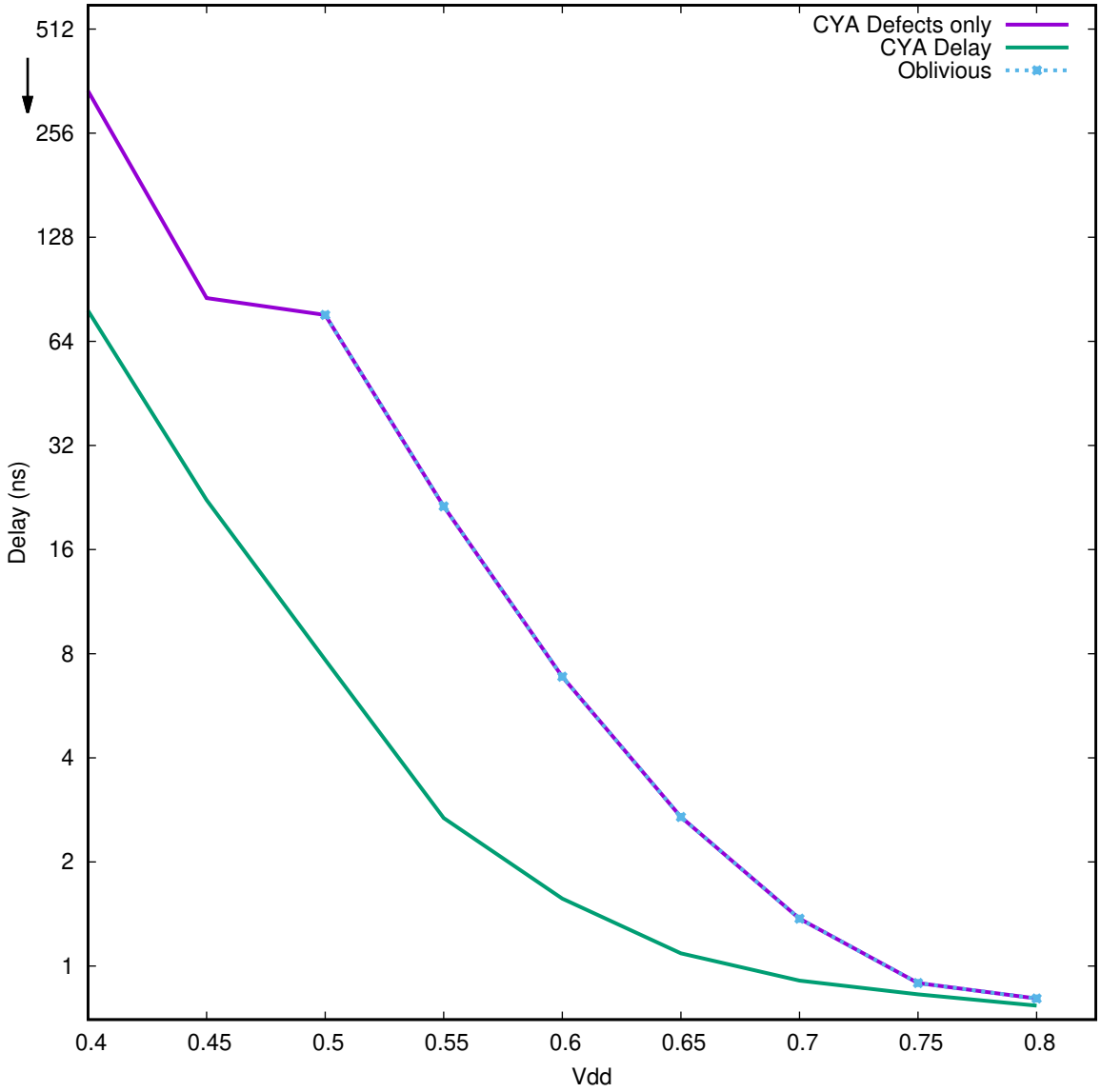


Figure 6.1: Impact of voltage reduction on delay for `des` running on a single simulated chip. This illustrates the difference between CYA with and without optimization of delay.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364V$, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect.

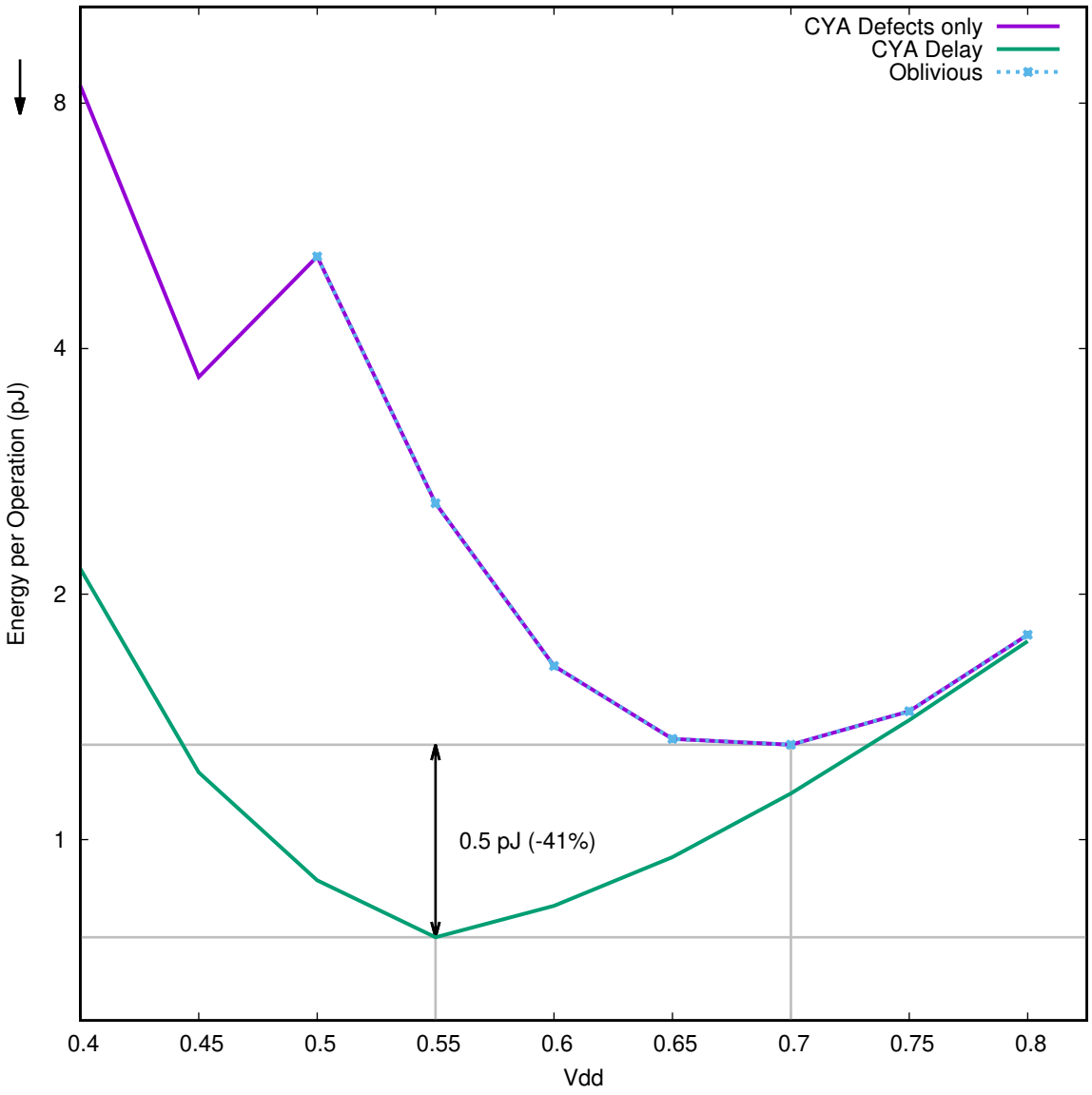


Figure 6.2: Impact of voltage reduction on energy usage of `des` running on a single simulated chip, again showing the difference between CYA with and without optimization of delay. The delay-agnostic algorithm freely selects slow paths as long as they are functional. The slowness of the mapping negates dynamic energy savings.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364V$, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect.

6.2 Energy Optimization

Energy-optimizing CYA, largely enabled by delay-optimizing CYA (Section 6.1), reduces energy consumption of the Toronto 20 benchmarks at variation rates appropriate to 22nm technology by an average of 61% relative to conventional bitstream loading practice (Table 6.6) and recovers an average of 70% of the energy lost to variations (see Table 6.7). Read on for more details.

Optimizing the energy usage of a design has a number of clear benefits. Most obviously, it can reduce operating costs for stationary systems, lower battery requirements for portable devices, and reduce environmental impacts. In addition, lowering power requirements could help alleviate the "Dark Silicon" problem (see Section 1.2) by reducing the amount of waste heat that needs to be dissipated per operation and thus increasing the possible active transistor density. Even if energy reduction comes at the cost of increased delay, the ability to activate more processing units in parallel could enable net performance improvements.

6.2.1 Energy Impacts of Voltage Reduction and Variations

Tuning V_{dd} is an essential part of optimizing energy usage. In a perfect chip, voltage reduction (up to a point) reduces dynamic energy costs (the cost of flipping bits) more than it increases static energy costs (the cost of current leakage), thus reducing the total energy consumed to complete a computation (energy per operation). For the 20 benchmarks I use (Toronto 20 [6]), Table 6.1 shows the energy consumption of perfect chips running at 0.8V (the normal operating voltage for the 22nm technology used in these simulations) and the energy at the optimal voltage.

In a chip with post-fabrication variations, lowering voltage begins to result in excessive delays and failures, reducing or negating energy savings, as was shown in

	Energy (pJ)		Energy Reduction	Optimal V_{dd}
	$V_{dd} = 0.8V$	At optimal V_{dd}	(%)	(V)
alu4	0.90	0.20	78	0.30
apex2	1.19	0.28	77	0.35
apex4	0.81	0.21	74	0.35
bigkey	0.95	0.20	79	0.30
clma	2.62	0.73	72	0.40
des	1.25	0.30	76	0.35
diffeq	0.26	0.07	73	0.40
dsip	1.44	0.27	82	0.30
elliptic	1.09	0.30	73	0.40
ex1010	3.39	0.93	72	0.40
ex5p	0.62	0.15	76	0.35
frisc	1.19	0.32	73	0.45
misex3	0.93	0.21	77	0.35
pdc	3.63	0.98	73	0.40
s298	0.81	0.18	78	0.30
s38417	1.96	0.48	75	0.35
s38584.1	2.69	0.62	77	0.35
seq	1.10	0.26	76	0.35
spla	2.85	0.76	73	0.40
tseng	0.28	0.07	75	0.35
Mean	1.50	0.38	76	0.36

Table 6.1: Energy consumption by variation-free chips at standard ($V_{dd} = 0.8$V) and energy-optimized (per-design) voltages. These results demonstrate the potential for energy savings through voltage reduction if chips can be fabricated free of variations.

Experiment parameters: 22nm technology, 20% extra base tracks, 16 reserved tracks, VPR 5 [53], single-driver interconnect.

Figures 6.1 and 6.2. By avoiding slow and non-functional resources, component-specific mapping provides a method to retain much of the voltage-reduction energy efficiency available with perfect chips.

Variations also result in a distribution of leakage energy and delay values, from the level of individual components all the way up to critical delays of entire circuits [61]. Figure 6.3 (from [61]) shows the delay distribution for individual 22nm inverters (two transistors) at $V_{dd} = 0.3$V, while Figure 6.4 shows the evolution of this distribution with V_{dd} for a more complex structure type (C-DUKs, see [26]). These effects compound further to create variations in the static energy and critical path delay values of entire statically mapped circuits, rather than the single result that arises in the variation-free case. Table 6.2 shows this impact at 0.8V for each of my benchmarks across 10000 chips.

To understand the effects of variations and voltage reduction in more detail, it is useful to recall the transistor current and delay equations first presented in Section 2.2 as Equation (2.3):

$$I_{sat} = W v_{sat} C_{ox} \left(V_{gs} - V_{th} - \frac{V_{d,sat}}{2} \right)^{\gamma} \tag{6.1a}$$

$$I_{sub} = \frac{W}{L} \mu C_{ox} (n-1) (V_T)^2 e^{\frac{V_{gs}-V_{th}}{nV_T}} \left(1 - e^{-\frac{V_{ds}}{V_T}} \right) \tag{6.1b}$$

$$I_{on} = \begin{cases} I_{sat} \quad for \quad V_{ds} = V_{dd} \geq V_{th} \\ I_{sub} \quad for \quad V_{ds} = V_{dd} < V_{th} \end{cases} \tag{6.1c}$$

$$I_{off} = I_{sub} (V_{gs} = 0) = \frac{W}{L} \mu C_{ox} (n-1) (V_T)^2 e^{-\frac{V_{th}}{nV_T}} \left(1 - e^{-\frac{V_{ds}}{V_T}} \right) \tag{6.1d}$$

$$\tau_p = \frac{C V_{dd}}{I_{on}} \tag{6.1e}$$

These equations have important consequences for energy consumption, especially in

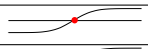
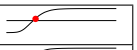
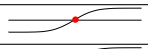
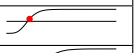
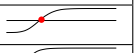
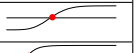
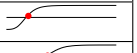
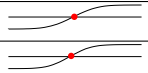
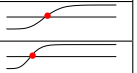
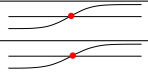
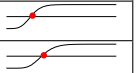
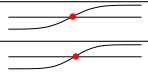
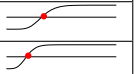
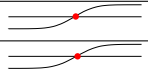
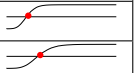
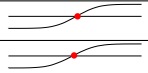
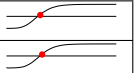
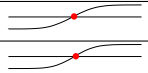
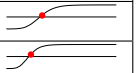
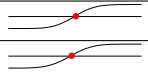
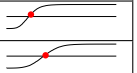
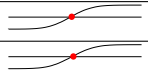
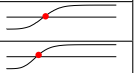
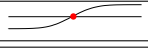
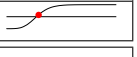
name	E_{dyn} (pJ)	Leakage ($pJ/\mu s$) min	med	max	CDF	Critical Path (ns) min	med	max	CDF
alu4	0.8	500	524	550		0.9	0.9	1.1	
apex2	1.0	711	740	769		1.0	1.0	1.3	
apex4	0.7	711	740	770		1.1	1.1	1.3	
bigkey	0.9	801	830	860		0.5	0.5	0.7	
clma	1.6	2670	2720	2780		2.0	2.1	2.3	
des	1.1	1050	1080	1110		0.8	0.9	1.1	
diffeq	0.2	499	524	550		0.5	0.6	0.8	
dsip	1.4	801	830	859		0.5	0.5	0.6	
elliptic	0.7	1380	1420	1450		1.3	1.5	1.6	
ex1010	2.2	2730	2790	2840		2.1	2.3	2.5	
ex5p	0.5	444	465	488		0.9	0.9	1.1	
frisc	0.6	1720	1760	1800		1.7	1.8	2.0	
misex3	0.8	547	570	595		0.9	1.0	1.3	
pdc	2.6	2600	2640	2690		1.9	2.1	2.4	
s298	0.7	406	429	451		0.9	1.0	1.4	
s38417	1.6	1780	1820	1870		0.9	1.0	1.1	
s38584.1	2.4	1600	1640	1680		0.9	1.0	1.2	
seq	0.9	711	740	769		1.0	1.1	1.3	
spla	2.2	1830	1870	1920		1.8	1.9	2.2	
tseng	0.2	413	433	453		0.5	0.6	0.7	
Mean	1.2	1190	1230	1260		1.1	1.2	1.4	

Table 6.2: Variation impact on the components of energy consumption at 0.8V. The effects of variations on dynamic energy (E_{dyn}) are higher-order only, and so are ignored in my models. Both leakage power and cycle duration (i.e., critical path delay) are affected.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, VPR 5 [53], single-driver interconnect, 10000 chips.

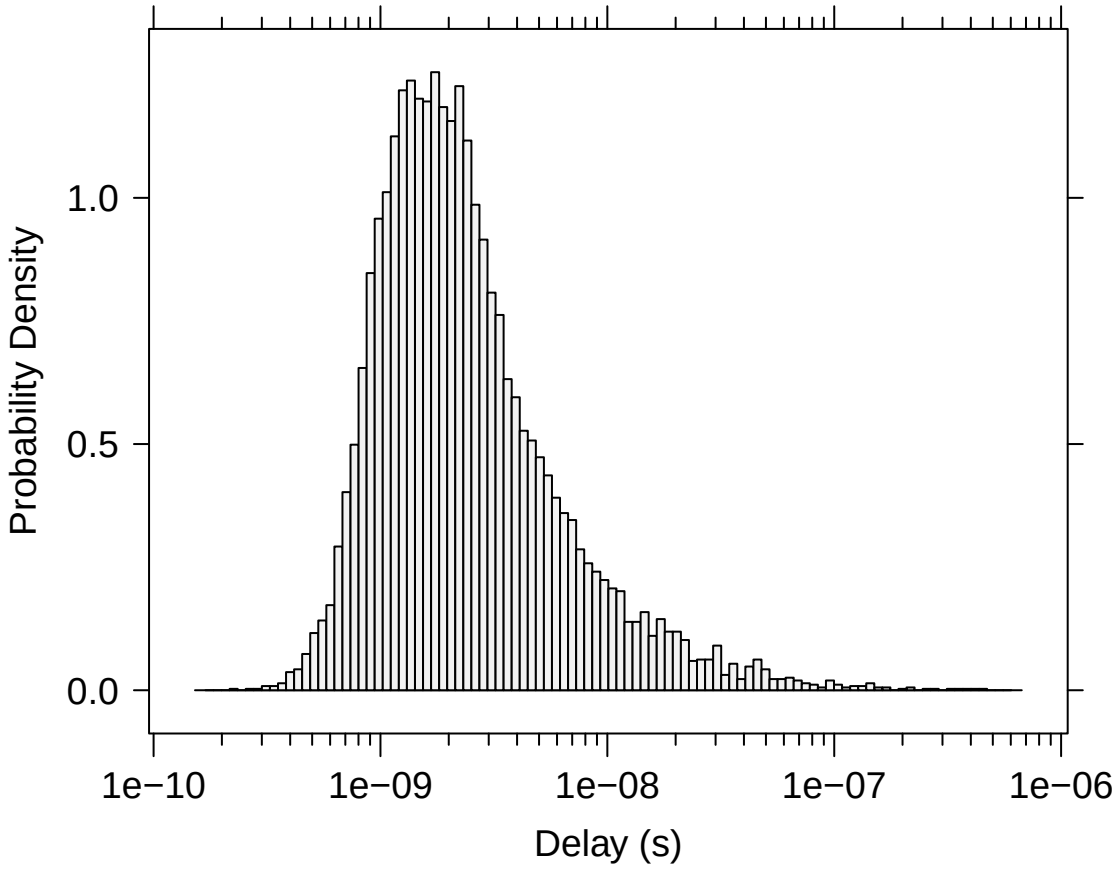


Figure 6.3: Inverter delay distribution under variation (W_p=W_n=L=22nm, V_{dd} = 0.3V). (Source: Mehta [61])

the presence of variations.[5]

As can be seen from Equation (6.1d), leakage current (I_{off}) decreases as V_{dd} is lowered, due to reduction of the $\left(1 - e^{-V_{ds}/V_T}\right)$ factor in Equation (6.1d). However, in the sub-threshold voltage region ($V_{dd} < V_{th}$), this trend competes with an exponential growth in delay, caused by the exponential decline of the drive current $\left(\text{in Equation (6.1b), } V_{gs} = V_{dd} \text{ results in } I_{on} = I_{sub} \propto e^{V_{dd}/nV_T}\right)$. This tends to produce a U-shaped static energy curve, as exemplified by the dash-dotted green line in

[5] See [87] or Mehta [61] Section 2.2 for more details beyond those discussed below.

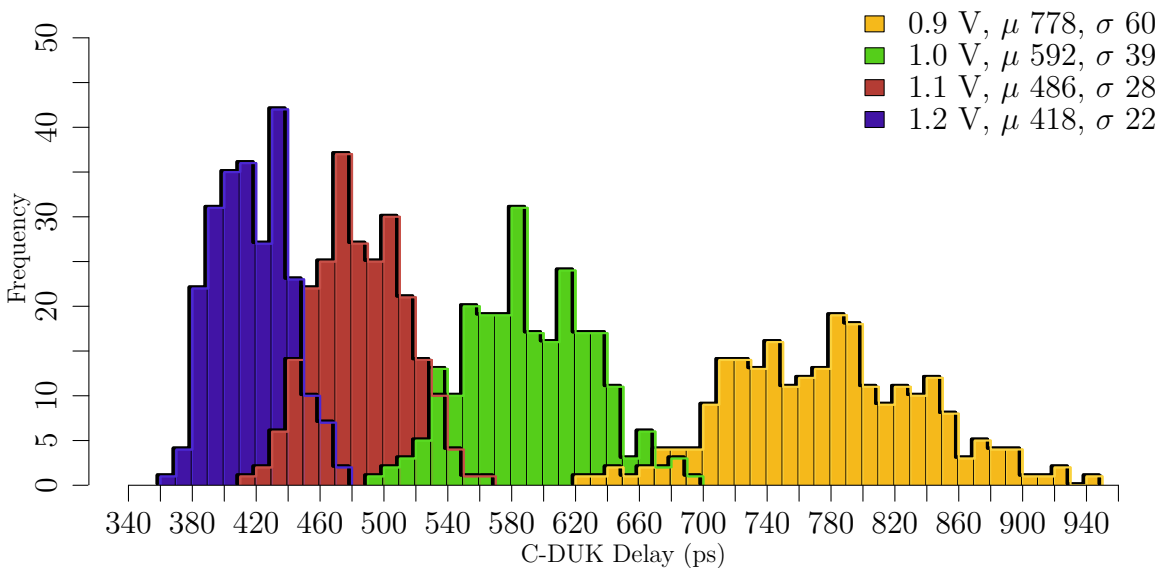


Figure 6.4: Real-world measurements illustrating both the increasing delay and widening distribution of delays resulting from the reduction of V_{dd}. These measurements were performed on a 65nm technology with a standard operating voltage of 1.2V. (Source: Gojman [26].)

Figure 6.6.

The effects of V_{dd} reduction on dynamic energy are more straightforward: reducing V_{dd} decreases the swing of gates, reducing the number of electrons that need to move around to turn a gate on or off (a linear to quadratic effect, see [59]) and lowering the dynamic energy consumption. Figures 6.5 and 6.6 (note the logarithmic energy scales in both) show that the contributions of static and dynamic energy dominate the total energy usage in different V_{dd} ranges, generally resulting in a total energy minimum somewhere in the subthreshold voltage region [29]. This is why subthreshold operation can provide significant energy savings in nominal chips.

However, a chip with variations will have transistors in which V_{th} deviates both above and below its nominal value. Transistors with higher V_{th} will have lower I_{on} at any given V_{dd}, meaning that they will take longer to drive transitions in downstream logic. Clock speeds will need to be further slowed to compensate, exponentially so in

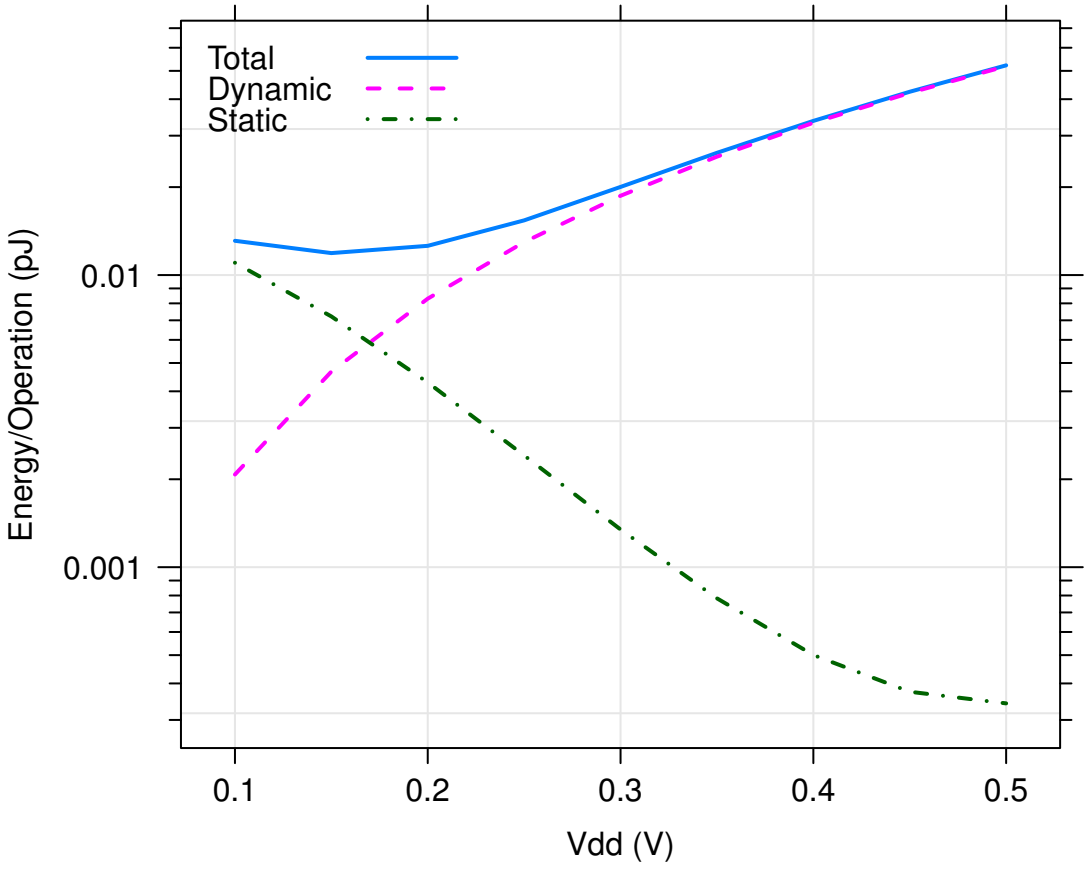


Figure 6.5: Static/dynamic breakdown of energy per operation for a 16-bit multiplier at 22nm ($V_{th} = 300$mV). (source: Mehta [61])

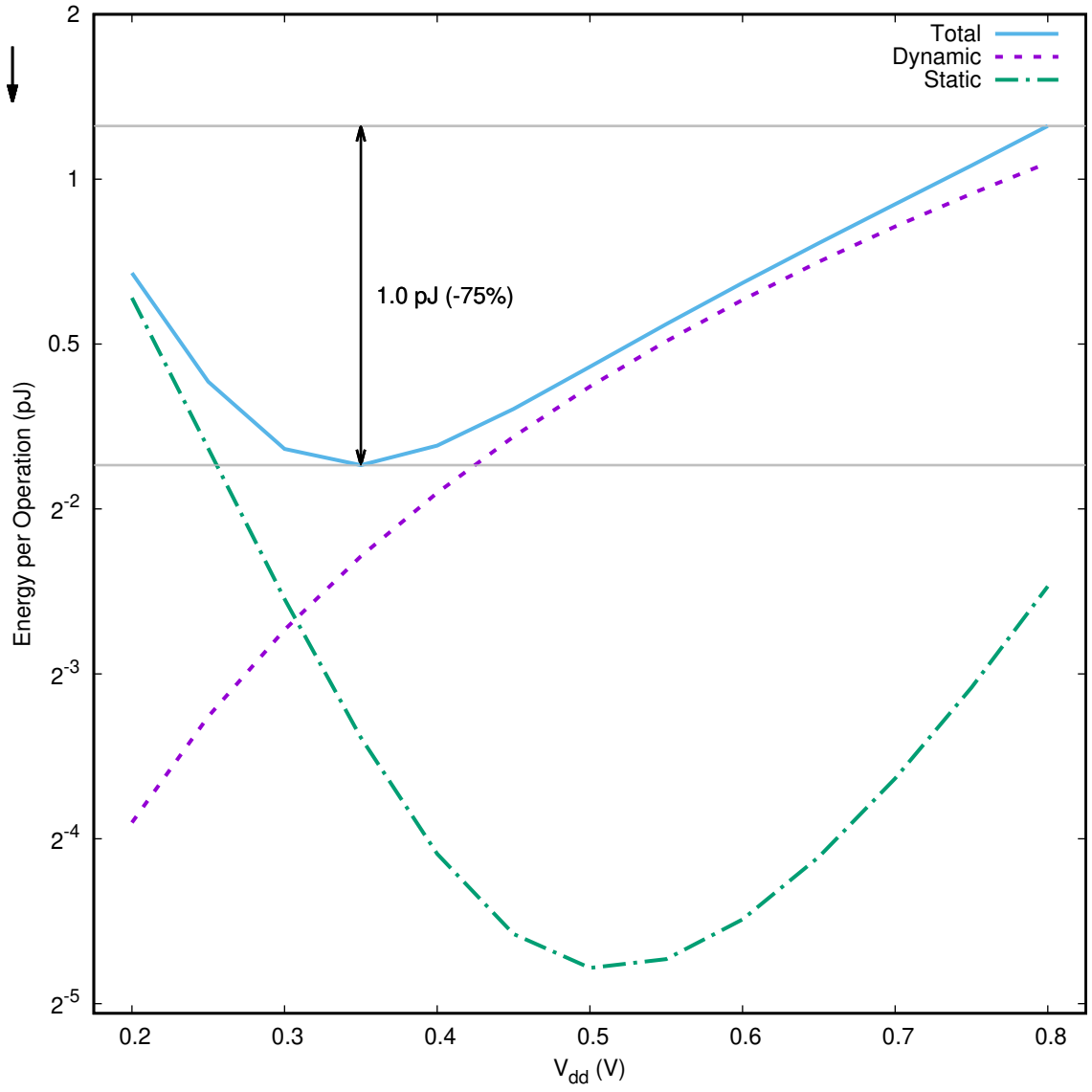


Figure 6.6: Energy consumption of `des` at 22nm, without variations. As in Figure 6.5, lowering V_{dd} below the standard voltage (0.8V), deep into the sub-threshold range, reduces energy consumption. Minimum energy consumption occurs at 0.35V. Static energy dominates at lower voltages and dynamic energy dominates at higher voltages.

subthreshold operations, resulting in additional static energy penalties.

Compounding the problem, there may also be transistors with lower than nominal V_{th}. These transistors will have exponentially higher I_{off} (leakage current), which will also be active over the exponentially longer clock cycles, further increasing the static energy consumption of the circuit.

Dynamic energy, on the other hand, is not strongly affected by variations.[6] Thus, if component-specific mapping can mitigate the static energy penalties associated with variations, voltage reduction will continue to offer energy savings through reduction of dynamic energy expenditures, just as it does in nominal chips. The experiments below show that CYA performs this task well.

6.2.2 Experiments

In conventional bitstream loading, manufacturers provide, for each line of chips, a set of timing guarantees which apply to standard operating conditions only (including V_{dd}). To map any given design, the user specifies the desired operating speed and computer-aided design (CAD) tools attempt to produce a single static map that will work at that clock speed if the timing guarantees hold true. In my simulations, I model conventional loading by running all chips at the standard voltage (0.8V) and using the slowest critical path from all 10000 simulated chips to represent the fastest speed available to this method. This significantly overestimates the speed and energy efficiency available to conventional loading, as manufacturers' timing guarantees necessarily provide for significant safety margins to ensure that advertised speeds will always be achieved, and to account for aging.

The energy distribution at $V_{dd} = 0.8$V resulting from conventional loading in the presence of variations is shown in Table 6.3 for all 20 of my benchmarks. We can

[6]The effects are higher-order at most and are not included in my models.

see from this table that the energy distribution is not very broad, but the average energy consumption is 6–11 times that which can be achieved in nominal chips. The question is how much of this lost energy efficiency can be restored by using CYA.

To optimize energy usage, CYA-based energy optimization essentially implements a component-specific version of dynamic frequency and voltage scaling (DFVS). Standard DFVS, like conventional loading, uses a single static map, but customizes the clock speed and voltage for each individual chip. It first seeks the fastest usable clock speed at each tested voltage, and then uses these results to find the most energy-efficient operating voltage overall. Using CYA rather than static mapping allows chips to run faster and have higher functioning yield at each voltage. This makes CYA-configured chips more efficient at any given voltage and allows even lower voltages to be accessed.

Voltage and energy distributions for DFVS ("oblivious mapping") are shown in Table 6.4. Comparable results for CYA are shown in Table 6.5. As is clear from these tables, and from the comparison statistics given in Tables 6.6 and 6.7, CYA significantly improves the energy efficiency relative not only to conventional loading (Table 6.3), but also relative to oblivious mapping. On average, across all benchmarks, oblivious mapping reduces energy to 54% of that required by conventional loading (absolute energy reduction), while CYA reduces energy to 39% of the conventional loading requirement. This corresponds to a recovery by CYA of 70% of the energy lost to variations (relative energy reduction), as compared to only a 54% recovery by oblivious mapping (Table 6.7). Because CYA can route around defective resources, its energy costs are also substantially less variable than those achieved with DFVS, as can be seen by comparing the standard deviation columns of Tables 6.4 and 6.5 or by examining the box plots in Figure 6.7.

Figure 6.7 shows a graphical comparison of nominal, conventional, oblivious, and

CYA energy results for a single benchmark (`des`).

While these improvements are substantial, CYA still does not completely recover the full 76% energy savings that can be obtained in nominal chips by optimal tuning of the operating voltage (see Table 6.1). Of course, variation-free fabrication is not realistic at modern feature sizes with any known technology, so it is useful to explore the best possible results that can be obtained in practice. The next chapter, Chapter 7, explores the theoretical limits of optimization in chips with variations, using full-knowledge customized mapping for each individual chip. We will see that CYA achieves most of the benefits of full-knowledge mapping at only a small fraction of the cost.

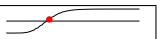
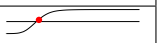
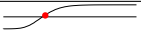
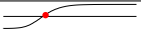
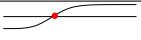
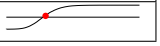
name	V_{dd} (V)	Energy (pJ) min	med	max	mean	s.d.	ΔE	% ΔE	CDF
alu4	0.80	1.25	1.29	1.38	1.29	0.01	1.09	546	
apex2	0.80	1.77	1.82	1.99	1.83	0.02	1.55	560	
apex4	0.80	1.42	1.48	1.63	1.49	0.02	1.28	619	
bigkey	0.80	1.26	1.31	1.49	1.31	0.02	1.11	555	
clma	0.80	6.87	7.24	7.95	7.25	0.13	6.52	896	
des	0.80	1.94	2.02	2.22	2.02	0.03	1.72	572	
diffeq	0.80	0.48	0.51	0.61	0.51	0.01	0.44	649	
dsip	0.80	1.73	1.78	1.90	1.78	0.02	1.52	571	
elliptic	0.80	2.57	2.75	3.03	2.76	0.06	2.45	821	
ex1010	0.80	8.10	8.58	9.25	8.59	0.14	7.65	821	
ex5p	0.80	0.93	0.96	1.07	0.97	0.01	0.82	563	
frisc	0.80	3.51	3.77	4.17	3.77	0.08	3.44	1064	
misex3	0.80	1.34	1.39	1.56	1.39	0.02	1.17	548	
pdc	0.80	7.72	8.10	8.85	8.11	0.12	7.12	727	
s298	0.80	1.12	1.16	1.31	1.16	0.01	0.98	548	
s38417	0.80	3.27	3.40	3.70	3.41	0.05	2.92	605	
s38584.1	0.80	3.86	4.00	4.30	4.00	0.05	3.38	546	
seq	0.80	1.68	1.73	1.94	1.73	0.02	1.47	565	
spla	0.80	5.50	5.78	6.24	5.79	0.09	5.02	660	
tseng	0.80	0.46	0.49	0.56	0.49	0.01	0.42	597	
Mean	0.80	2.84	2.98	3.26	2.98	0.05	2.60	652	

Table 6.3: The energy per operation with variations, using conventional loading for each benchmark. That is, V_{dd} is 0.8V for all chips, and the clock period for each benchmark is determined by the slowest of the 10000 chips for that benchmark. ΔE numbers are relative to the optimal energy achievable by adjusting V_{dd} in nominal chips (Table 6.1). Static energy (leakage rate) is the only component of the energy calculations affected by the variations of each chip.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, VPR 5 [53], single-driver interconnect, 10000 chips.

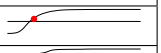
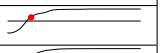
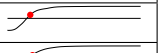
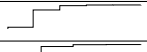
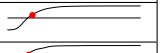
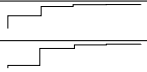
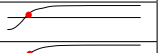
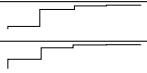
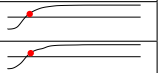
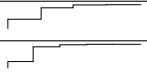
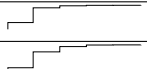
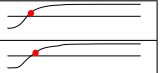
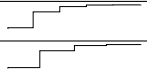
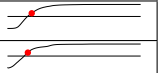
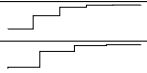
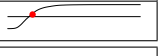
	Voltage (V)			Energy (pJ)					
name	min	max	CDF	min	med	max	mean	s.d.	CDF
alu4	0.55	0.80		0.58	0.70	1.32	0.72	0.07	
apex2	0.55	0.80		0.83	1.01	1.92	1.04	0.11	
apex4	0.55	0.80		0.67	0.82	1.60	0.83	0.09	
bigkey	0.55	0.80		0.58	0.73	1.42	0.75	0.08	
clma	0.60	0.80		2.80	3.47	7.18	3.57	0.41	
des	0.55	0.80		0.86	1.05	2.13	1.07	0.11	
diffeq	0.55	0.80		0.22	0.27	0.55	0.28	0.03	
dsip	0.50	0.75		0.81	1.01	1.78	1.03	0.11	
elliptic	0.55	0.80		1.04	1.30	2.75	1.33	0.15	
ex5p	0.55	0.80		0.44	0.52	0.99	0.54	0.06	
frisc	0.60	0.80		1.38	1.73	3.83	1.77	0.21	
misex3	0.55	0.75		0.61	0.74	1.52	0.76	0.08	
pdc	0.60	0.80		3.45	4.20	8.41	4.32	0.46	
s298	0.55	0.80		0.50	0.61	1.18	0.62	0.06	
s38584.1	0.55	0.80		1.74	2.17	4.04	2.22	0.24	
seq	0.55	0.75		0.76	0.94	1.89	0.96	0.10	
spla	0.55	0.80		2.42	2.88	5.82	2.94	0.33	
tseng	0.55	0.75		0.20	0.25	0.52	0.26	0.03	
Mean	0.56	0.79		1.10	1.36	2.71	1.39	0.15	

Table 6.4: Oblivious (DFVS) energy statistics for 10000 simulated chips for each benchmark.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, VPR 5 [53], single-driver interconnect, 10000 chips.

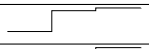
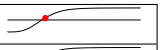
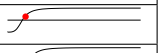
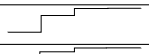
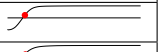
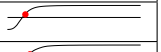
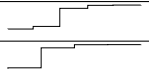
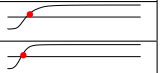
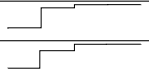
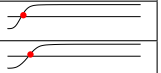
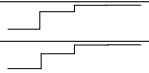
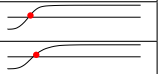
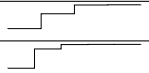
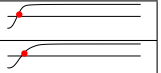
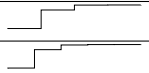
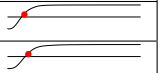
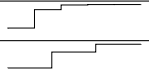
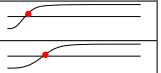
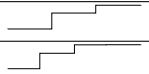
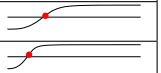
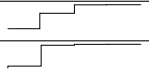
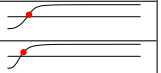
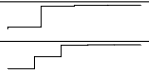
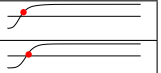
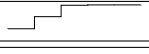
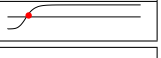
	Voltage (V)			Energy (pJ)					
name	min	max	CDF	min	med	max	mean	s.d.	CDF
alu4	0.45	0.60		0.48	0.53	0.66	0.53	0.02	
apex2	0.45	0.60		0.68	0.77	0.98	0.77	0.03	
apex4	0.50	0.65		0.56	0.61	1.00	0.62	0.03	
bigkey	0.45	0.70		0.45	0.54	0.99	0.55	0.04	
clma	0.50	0.70		2.26	2.53	4.14	2.56	0.16	
des	0.45	0.65		0.69	0.76	1.22	0.76	0.03	
diffeq	0.45	0.70		0.17	0.20	0.40	0.20	0.02	
dsip	0.40	0.65		0.62	0.74	1.42	0.75	0.06	
elliptic	0.50	0.70		0.83	0.96	2.00	0.97	0.05	
ex5p	0.45	0.65		0.36	0.39	0.57	0.40	0.01	
frisc	0.50	0.70		1.07	1.27	2.08	1.29	0.09	
misex3	0.45	0.70		0.51	0.56	1.11	0.56	0.03	
pdc	0.50	0.70		2.89	3.22	5.72	3.25	0.18	
s298	0.45	0.70		0.41	0.47	0.79	0.47	0.03	
s38584.1	0.45	0.60		1.44	1.64	2.16	1.64	0.08	
seq	0.45	0.65		0.64	0.70	1.04	0.71	0.03	
spla	0.50	0.70		1.98	2.14	3.47	2.16	0.09	
tseng	0.45	0.70		0.16	0.19	0.36	0.19	0.01	
Mean	0.46	0.67		0.90	1.01	1.67	1.02	0.06	

Table 6.5: CYA energy statistics for 10000 simulated chips for each benchmark.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 10000 chips.

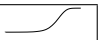
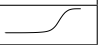
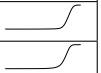
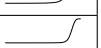
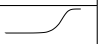
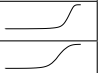
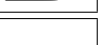
	Oblivious					CYA				
name	min	med	max	mean	CDF	min	med	max	mean	CDF
alu4	0	46	55	44		49	59	63	59	
apex2	0	45	54	43		46	58	63	58	
apex4	0	45	54	44		34	59	63	58	
bigkey	0	44	55	43		25	59	66	58	
clma	0	52	60	51		44	65	70	65	
des	0	48	57	47		39	62	67	62	
diffeq	0	47	58	46		28	61	69	60	
dsip	5	43	55	42		21	59	65	58	
elliptic	0	53	60	52		27	65	70	65	
ex5p	0	46	54	44		41	59	64	59	
frisc	0	54	62	53		47	66	73	66	
misex3	3	47	56	45		19	60	65	59	
pdc	0	48	57	47		31	60	65	60	
s298	0	48	56	46		34	60	65	59	
s38584.1	0	46	56	44		47	59	64	59	
seq	2	46	56	45		40	59	64	59	
spla	0	50	58	49		41	63	67	63	
tseng	2	48	58	47		30	62	67	61	
Mean	1	47	57	46		36	61	66	61	

Table 6.6: Percentage energy reduction relative to conventional loading. The "oblivious" columns show the energy savings achieved with DFVS using a single bitstream for all 10000 chips. The "CYA" columns show the greater benefits of energy optimization with CYA.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 10000 chips.

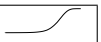
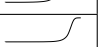
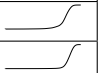
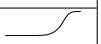
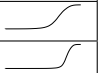
	Oblivious					CYA				
name	min	med	max	mean	CDF	min	med	max	mean	CDF
alu4	0	54	65	53		58	70	75	70	
apex2	0	53	64	51		54	68	74	68	
apex4	0	52	63	51		39	68	73	68	
bigkey	0	52	65	51		29	69	78	69	
clma	0	58	67	56		50	72	78	72	
des	0	57	67	55		46	73	78	73	
diffeq	0	54	66	53		32	70	79	70	
dsip	5	51	64	50		24	69	77	68	
elliptic	0	59	68	58		31	73	78	73	
ex5p	0	54	64	52		48	70	74	70	
frisc	0	59	68	58		51	72	79	72	
misex3	3	55	66	53		22	71	75	70	
pdc	0	55	65	53		35	69	73	68	
s298	0	56	67	55		40	71	76	70	
s38584.1	0	54	67	53		56	70	76	70	
seq	3	54	66	52		47	70	75	70	
spla	0	58	67	57		47	73	77	72	
tseng	2	56	68	55		35	72	79	72	
Mean	1	55	66	54		41	71	76	70	

Table 6.7: Percentage of energy lost due to variations (relative to nominal) recovered (mitigated) by oblivious (DFVS) and CYA mapping.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 10000 chips.

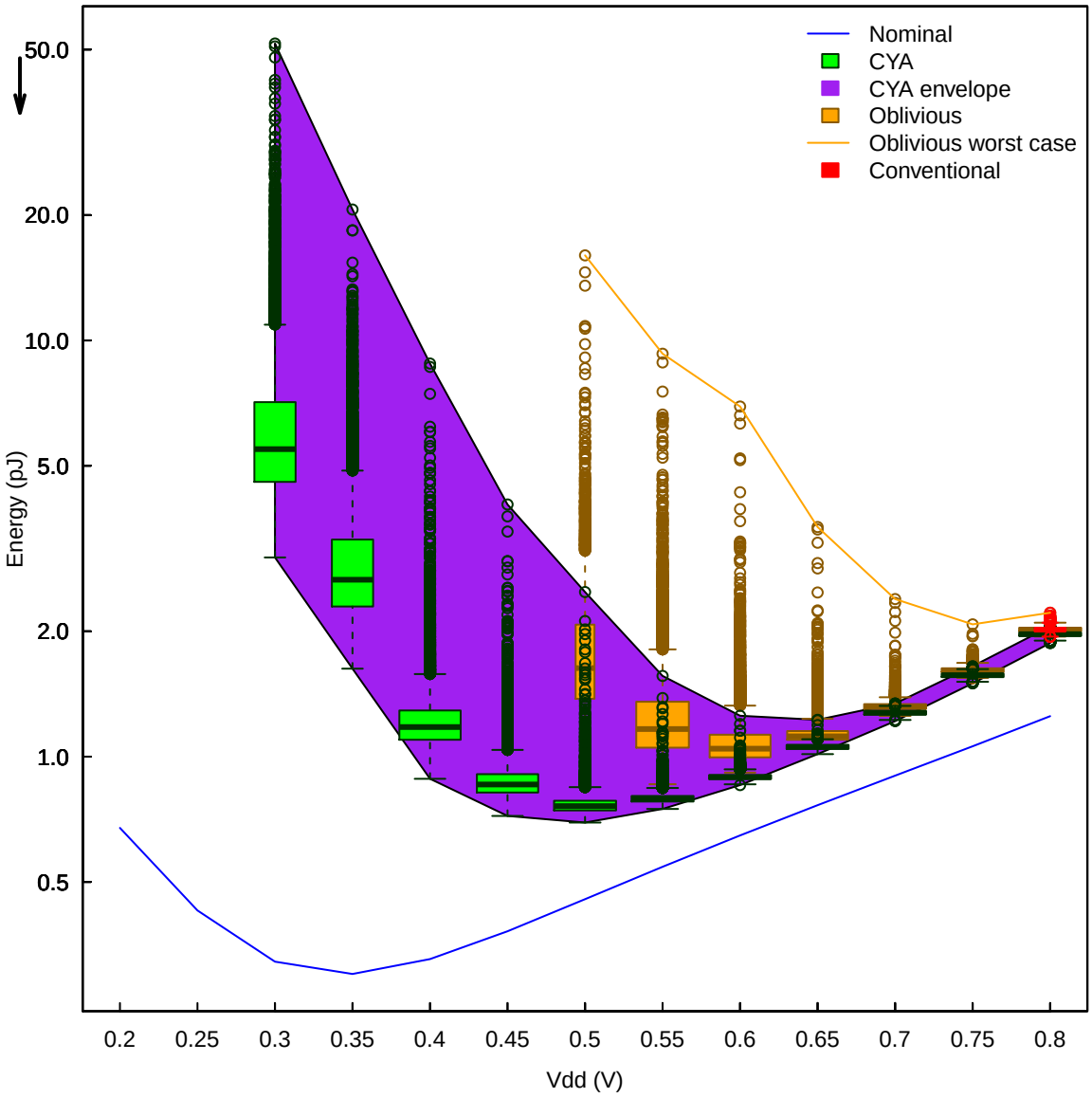


Figure 6.7: Distribution of energy per operation of `des` vs. V_{dd}, for all loading methods.

Nominal results (static mapping on perfect chips) fall along a single (blue) line, using less energy than all other methods. Conventional results (red) are all mapped at 0.8V and the range of energy variation is too small to see on this graph. Oblivious (orange) and CYA (green/purple) mapping produce a substantial range of energy values at each voltage, as shown in the box plots, the "oblivious worst case" line, and the outlined "envelope" of the CYA results. Box widths correspond to the fraction of chips that were mapped successfully at each voltage; oblivious mapping failed to map any chips below 0.5V.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 10000 chips.

Chapter 7

Limit Studies

The goal of Choose-Your-own-Adventure (CYA) is to make practical the benefits of component-specific mapping (CSM) for delay and energy savings. This chapter examines the tradeoffs associated with the CYA approach and explores possible future avenues of improvement.

A number of costs have slowed adoption of CSM, including those associated with:

1. Per-chip testing/measurement
2. Per-chip, per-design computer-aided design (CAD)
3. Increases in load time and/or in-situ computation (e.g., increased bitstream size and load time due to alternatives)

CYA mitigates these costs by making certain compromises between delay/energy optimization and CAD/loading efficiency:

1. It commits to using all base paths that pass testing (see Section 7.3).
2. It provides only a limited set of alternative routes for defect and delay repair (see Section 7.2).

3. It greedily selects amongst those alternatives during repair (see Section 7.1).

In this chapter I strip away these layers of compromise, one after the other, to show the delay and energy costs (relative to full-knowledge routing) and CAD/loading efficiency benefits of this approach. The results of these tests are discussed individually in Sections 7.1 to 7.3. Section 7.4 brings all of these results together for comparison.

7.1 Pathfinder Negotiated Alternative Selection

For this first study, the initial mapping and testing of the base route remains intact, leaving a common starting point for repair. In addition, the same limited selection of alternatives is generated as in CYA. The only change is that the "first to work" greedy selection of alternative paths for repair is now replaced with a Pathfinder-based heuristic congestion negotiation process. As the challenge of selecting amongst alternatives is closely similar to that of free-form routing, the capabilities of Pathfinder are a natural fit for this task.[7]

For a net requiring repair, the Pathfinder-based alternative selector works by first filtering the available alternatives using the same acceptance tests as CYA. The acceptable paths for each net are collected into a paths tree (Section 4.3.1), as in path-cost aware alternatives generation (Section 4.3). These paths trees are then used as constraints for a shortest path search — only expansions that follow the trees are permitted, but no following penalty is applied to the cost function. This is in contrast to the approach taken in path-cost routing, in which following a paths tree is strongly penalized.

As shown in [24] and Figure 7.3, this results in a large jump in load-time computational overhead relative to CYA, requiring a fair bit of computational power and

[7]It has been suggested that alternative selection might also be an interesting application for SAT solvers. This may be a promising experiment for future work.

memory to support Pathfinder. Load times increase by two orders of magnitude, from a few seconds to several minutes; see the referenced paper for details of the timing model. There is also a notable increase in the number of paths tested — CYA usually tries only a small number of the theoretically acceptable paths because it almost always finds a working solution within the first few alternatives tested. Some paths may also never be tried by CYA because they conflict with alternatives already selected to repair other nets. However, despite the increased costs of using Pathfinder to choose alternatives, this process is still extremely light-weight relative to the detailed measurements required to perform knowledge-based routing (Section 7.3).

Figures 7.1 and 7.2 show that, in exchange for these additional costs, Pathfinder alternative selection does improve both delay and energy efficiency compared to CYA's greedy selection. Over all the Toronto 20 designs, the median energy consumption is lower by approximately 8% (see Table 7.4). As shown in Table 7.5, on average, about 12% of the energy savings that full-knowledge routing achieves relative to oblivious (dynamic frequency and voltage scaling (DFVS)) loading can be restored by using Pathfinder alternative selection rather than CYA's greedy selection. Other, more efficient alternative selection methods may be able to improve on the timing costs of the Pathfinder-based algorithm while still retaining much of its delay and energy advantage.

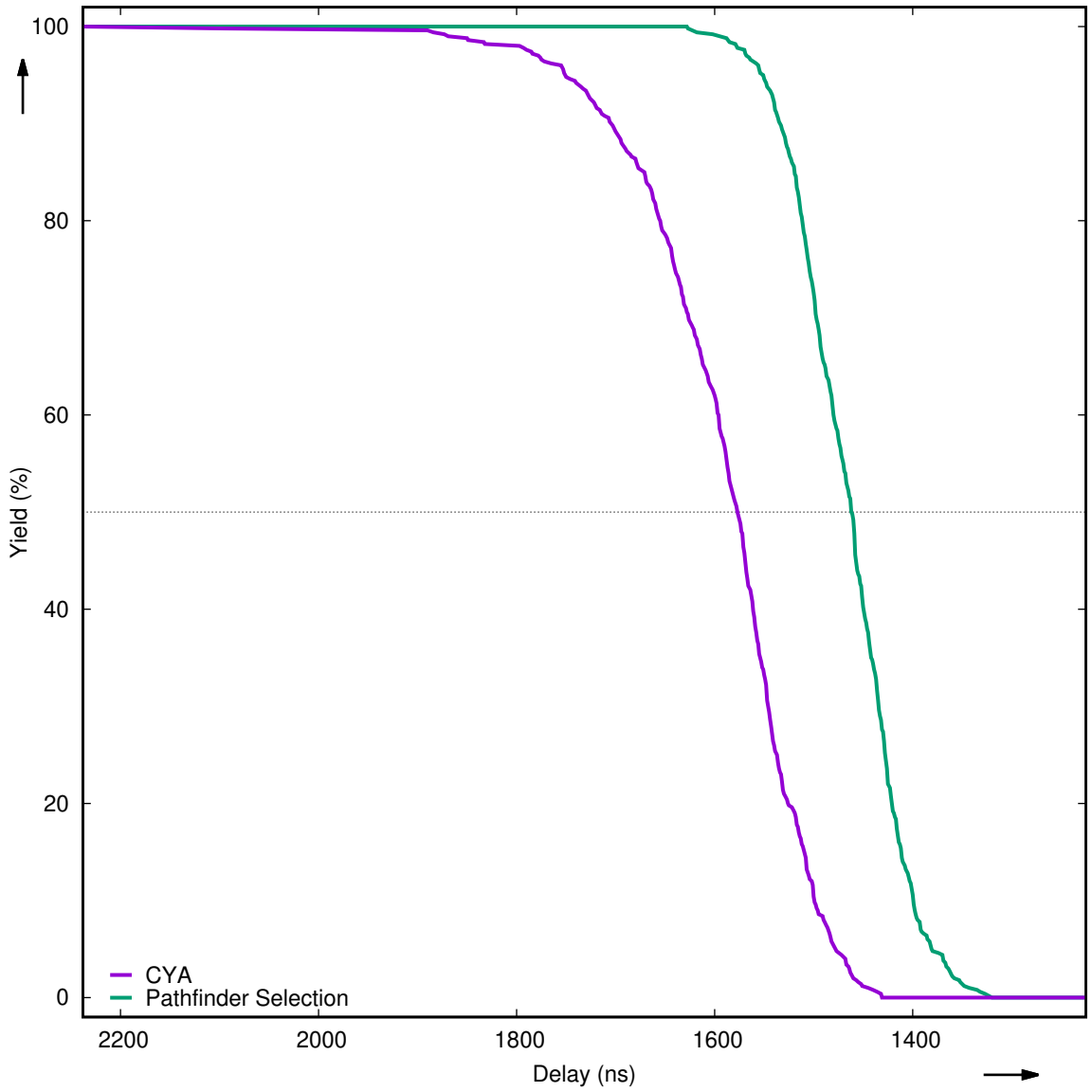


Figure 7.1: Delay parametric yield for **CYA and Pathfinder-based alternative selection**.

Experiment parameters: V_{dd} = 0.6V, `des`, 22nm technology, $\sigma_{V_{th}}$ = 0.0364V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

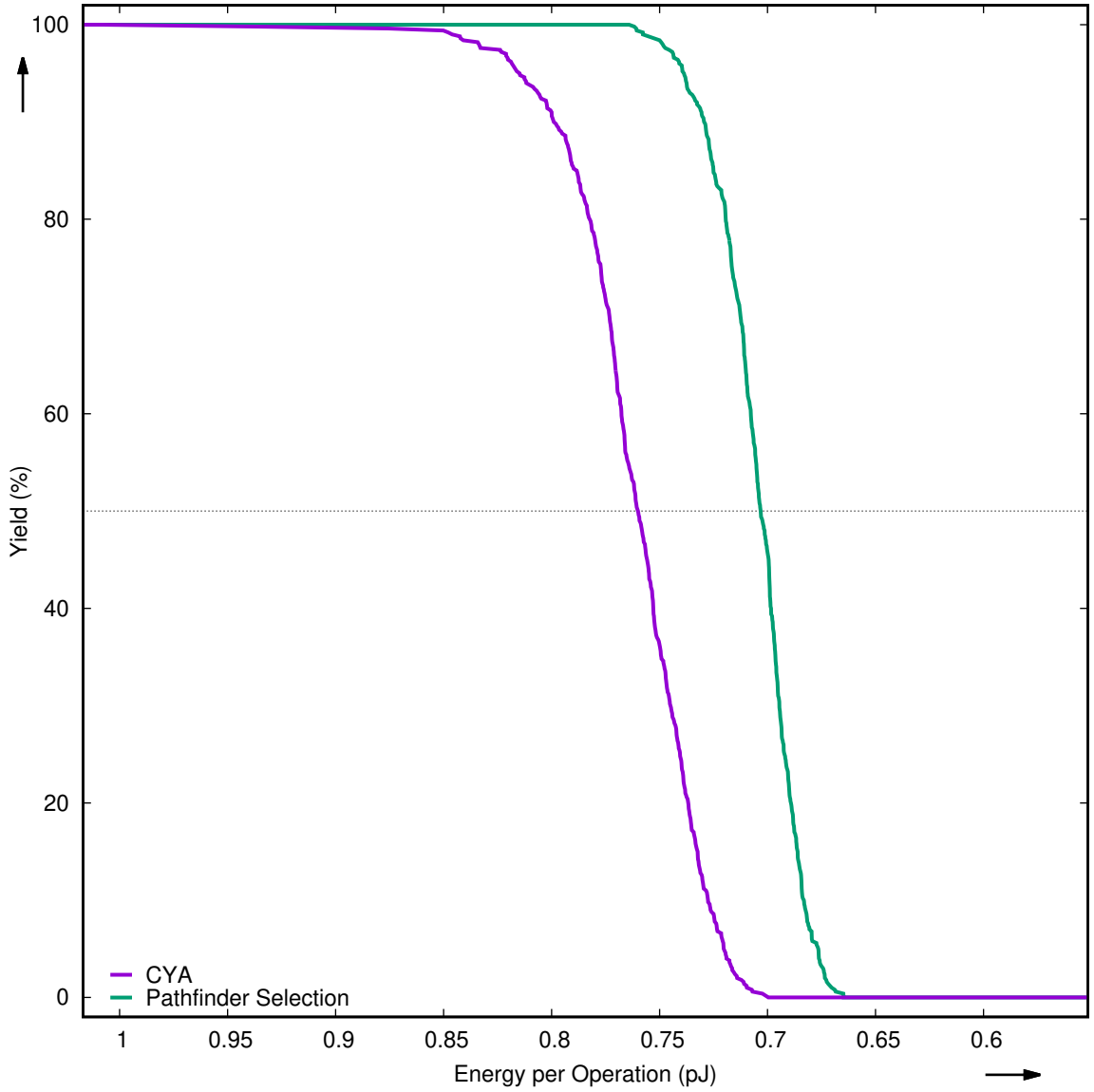


Figure 7.2: Energy parametric yield for **CYA and Pathfinder-based alternative selection**.

Experiment parameters: energy-optimizing V_{dd}, `des`, 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

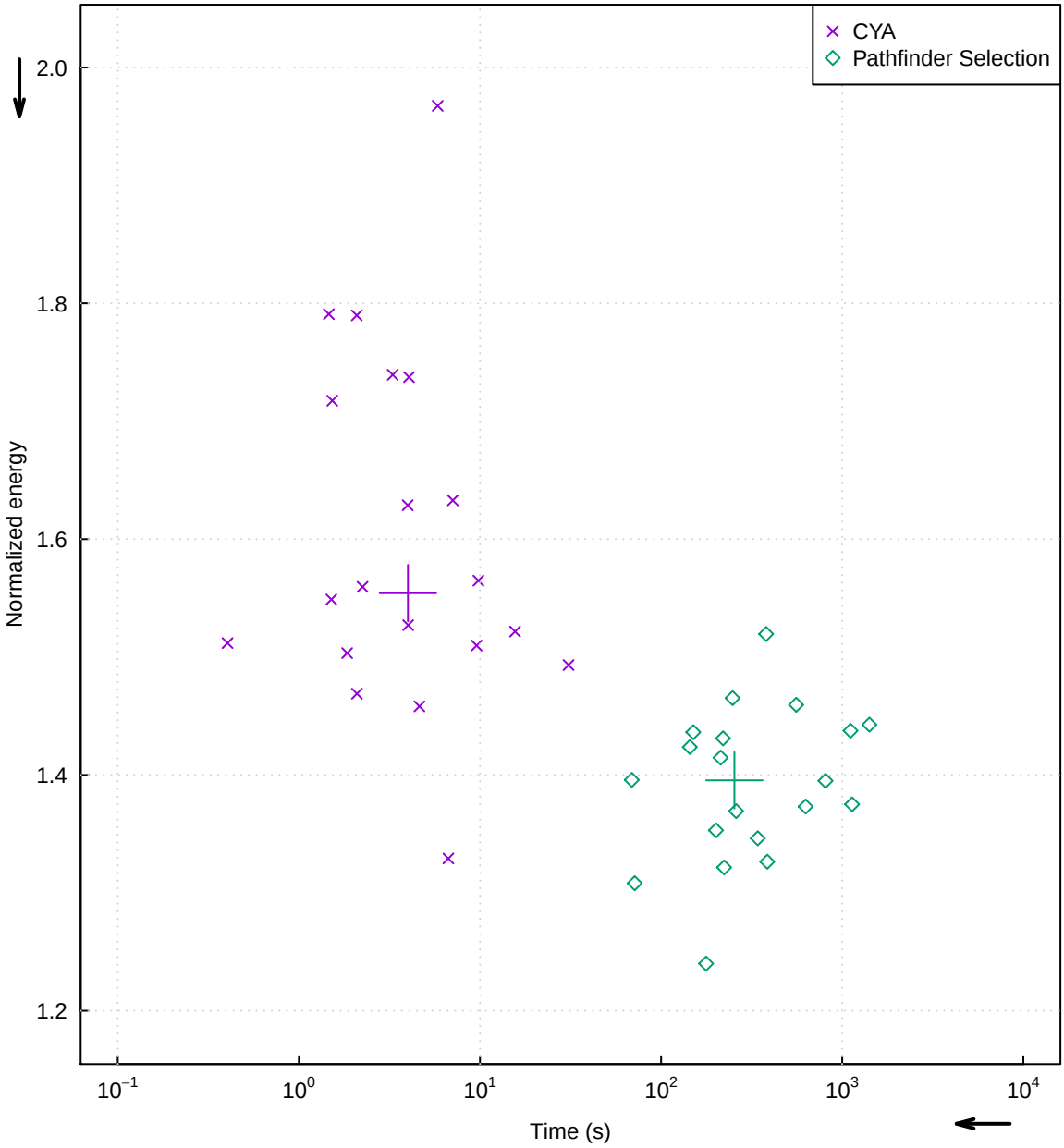


Figure 7.3: Energy vs. CAD/loading time for **CYA and Pathfinder-based alternative selection**. Each point is the 95th percentile of normalized energy consumption amongst 20 chips, for one of the Toronto 20 [6] benchmarks. Normalization for each chip was performed with respect to the energy consumption achievable for that same chip using full-knowledge routing. See [24] (graph source) for details of the timing model. The crosshairs show the median time and energy for each alternative selection method.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 20 chips.

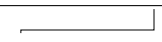
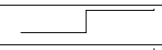
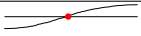
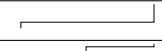
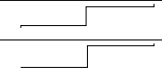
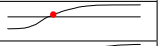
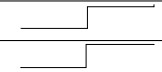
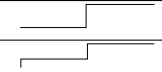
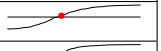
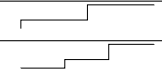
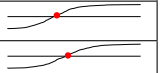
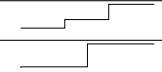
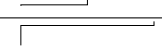
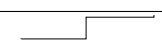
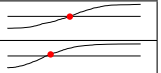
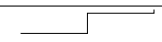
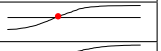
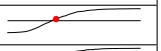
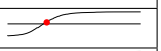
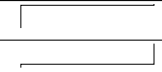
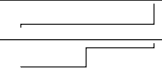
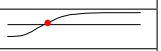
	Voltage (V)			Energy (pJ)					
name	min	max	CDF	min	med	max	mean	s.d.	CDF
alu4	0.45	0.50		0.46	0.49	0.53	0.49	0.01	
apex2	0.45	0.55		0.67	0.71	0.76	0.71	0.02	
apex4	0.50	0.55		0.54	0.58	0.62	0.58	0.01	
bigkey	0.45	0.55		0.45	0.51	0.65	0.52	0.03	
clma	0.50	0.60		2.12	2.30	2.52	2.31	0.07	
des	0.45	0.55		0.67	0.70	0.76	0.70	0.02	
diffeq	0.50	0.60		0.16	0.18	0.22	0.18	0.01	
dsip	0.40	0.55		0.61	0.69	0.80	0.69	0.03	
elliptic	0.50	0.60		0.80	0.86	0.98	0.86	0.03	
ex1010	0.55	0.60		2.77	2.98	3.20	2.97	0.08	
ex5p	0.45	0.55		0.35	0.37	0.42	0.37	0.01	
frisc	0.50	0.60		1.01	1.12	1.30	1.12	0.05	
misex3	0.45	0.55		0.49	0.52	0.56	0.52	0.01	
pdc	0.50	0.60		2.78	2.94	3.26	2.95	0.08	
s298	0.45	0.55		0.41	0.43	0.49	0.44	0.01	
s38417	0.50	0.60		1.21	1.31	1.57	1.32	0.05	
s38584.1	0.45	0.55		1.40	1.52	1.68	1.52	0.06	
seq	0.50	0.55		0.61	0.65	0.70	0.65	0.02	
spla	0.50	0.55		1.91	2.00	2.23	2.00	0.04	
tseng	0.45	0.55		0.15	0.17	0.19	0.17	0.01	
Mean	0.48	0.57		0.98	1.05	1.17	1.05	0.03	

Table 7.1: Statistics of minimum energy and energy-minimizing voltage for **Pathfinder-based alternative selection**.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

7.2 Pathfinder Knowledge-Based Repair

The next question is how much repair capability and energy efficiency is lost by limiting the set of paths available for repair. To test this, I proceed similarly to Section 7.1, but remove the constraint that repairs must use only the precomputed alternatives. Essentially this constitutes a limited version of full-knowledge routing (i.e., routing based on complete characterization of all resource variations), constrained to the space of broken nets and resources not used in the functional portions of the base path.

It should be noted that, because of the use of Pathfinder for repair, the results of this test will *not* be the same as the results we would obtain through greedy selection amongst all possible alternative paths. As discussed in the previous section, CYA's greedy alternative selection trades speed in finding an acceptable path for the possibility of missing out on a better path. So, as before, Pathfinder repair with unlimited path choices will try more paths than CYA with unlimited alternatives, but it may find better paths as a result.

As illustrated by Table 4.11, precisely because of CYA's greedy selection of the first acceptable alternative, adding alternatives to CYA shows diminishing returns after a certain point. Thus, completely removing alternative limits, rather than simply increasing some finite alternative count, is likely beneficial primarily when negotiated alternative selection is already in use, as in the present tests.

Since Pathfinder-based alternative selection from an unlimited choice of paths is essentially limited full-knowledge routing, we should expect that the cost of this approach will begin to more closely resemble that of full-knowledge routing. Specifically, it requires that all variations in repair resources (but not in resources that are unavailable for repair, such as non-shareable, consumed base resources, unused blocks, and out-of-bounds interconnect) be completely characterized, in order to en-

able Pathfinder to select the best repair pathways. As discussed in Section 1.2, this kind of comprehensive characterization can take hours to weeks per chip. However, some savings are still present on the routing side, due to the fact that only broken nets need to be routed and these nets can only be routed onto reserve and unused base resources. Timing results are shown in Figure 7.6 — note the roughly four order of magnitude increase in time costs compared to Pathfinder alternative selection.

In exchange for this significant increase in CAD/loading time, we see in Figures 7.4 and 7.5 that free-form routing for repair does provide noticeable delay and energy improvements compared to smart selection amongst pre-computed alternatives. Median energy costs improve by about 10% across all benchmarks (Table 7.4). Table 7.5 shows that an additional 14% of the energy savings that full-knowledge routing achieves relative to oblivious loading can be restored by this step of removing the limit on alternatives.

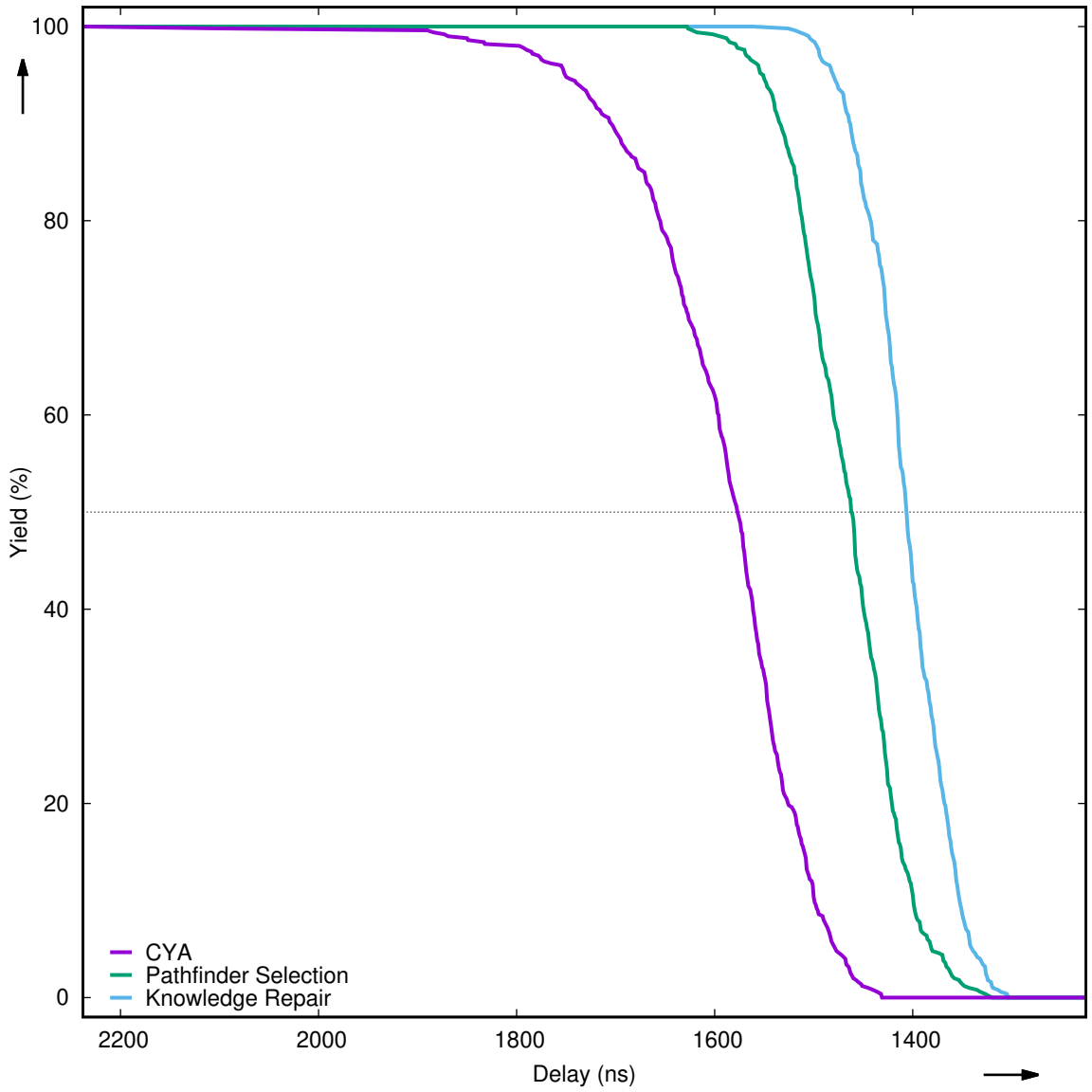


Figure 7.4: Delay parametric yield for **CYA, Pathfinder-based alternative selection, and Pathfinder knowledge-based repair.**

Experiment parameters: V_{dd} = 0.6V, `des`, 22nm technology, $\sigma_{V_{th}}$ = 0.0364V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

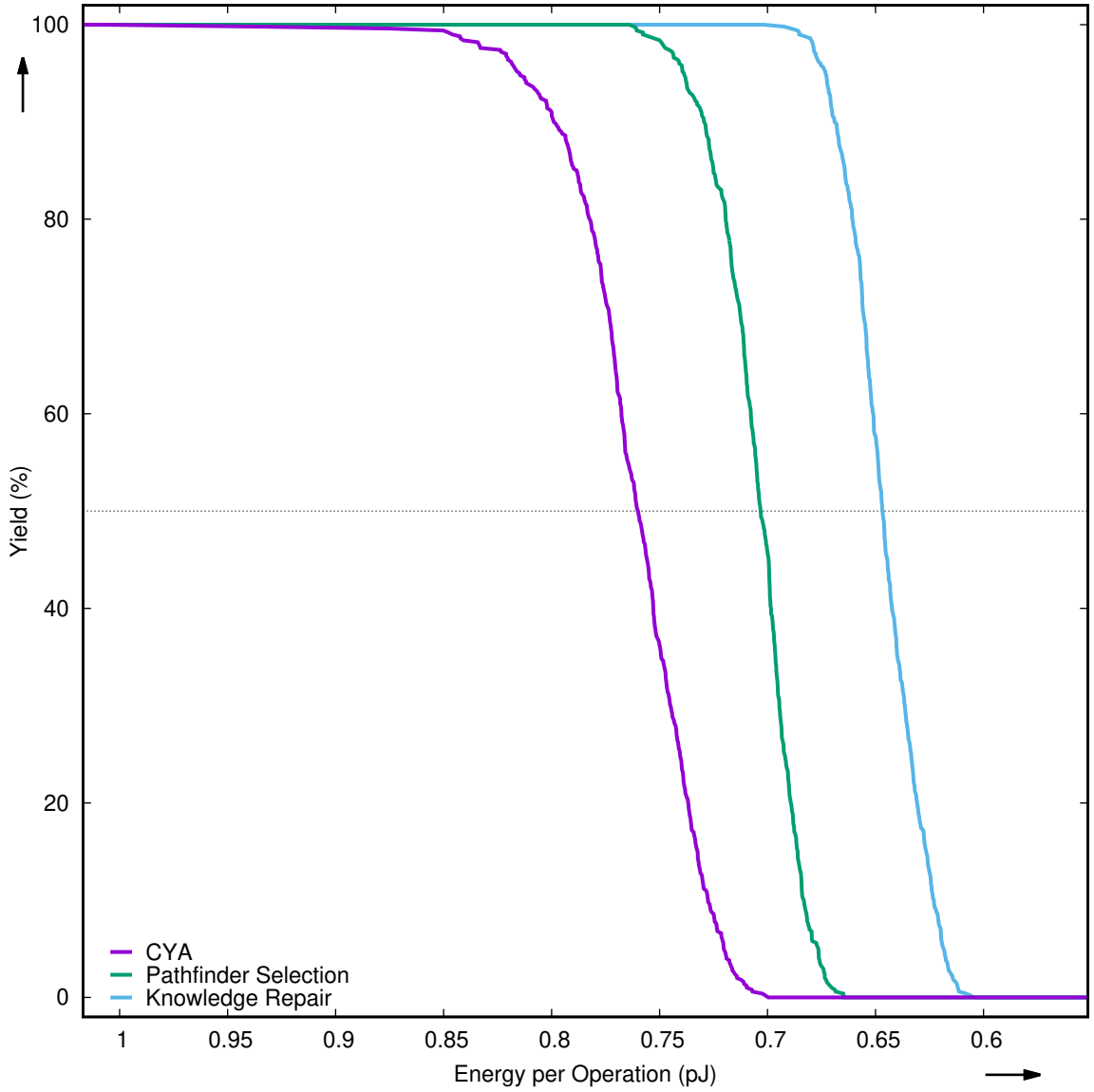


Figure 7.5: Energy parametric yield for **CYA, Pathfinder-based alternative selection, and Pathfinder knowledge-based repair.**

Experiment parameters: energy-optimizing V_{dd}, `des`, 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

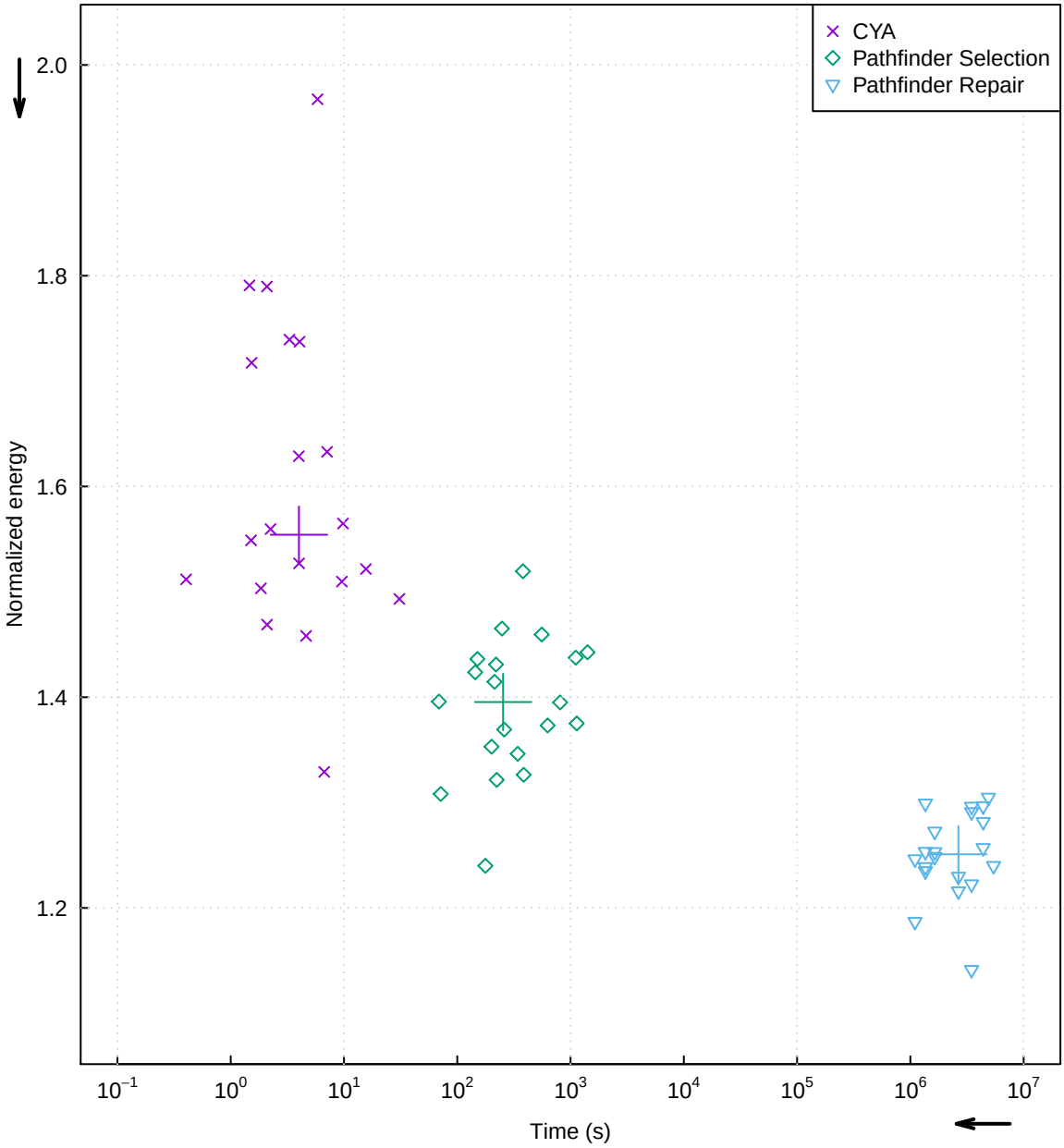


Figure 7.6: Energy vs. CAD/loading time for **CYA, Pathfinder-based alternative selection, and Pathfinder knowledge-based repair**. Each point is the 95th percentile of normalized energy consumption amongst 20 chips, for one of the Toronto 20 [6] benchmarks. Normalization for each chip was performed with respect to the energy consumption achievable for that same chip using full-knowledge routing. See [24] (graph source) for details of the timing model. The crosshairs show the median time and energy for each alternative selection method.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 20 chips.

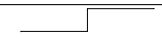
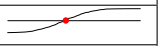
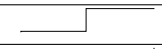
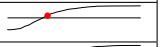
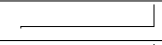
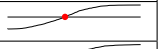
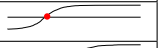
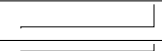
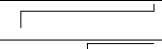
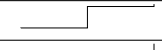
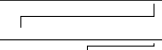
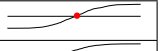
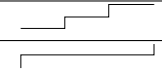
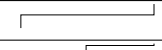
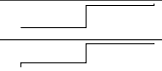
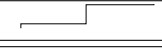
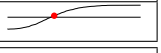
	Voltage (V)			Energy (pJ)					
name	min	max	CDF	min	med	max	mean	s.d.	CDF
alu4	0.40	0.50		0.41	0.44	0.47	0.44	0.01	
apex2	0.45	0.50		0.60	0.64	0.68	0.64	0.01	
apex4	0.45	0.55		0.48	0.51	0.55	0.51	0.01	
bigkey	0.40	0.50		0.41	0.44	0.52	0.44	0.02	
clma	0.50	0.55		1.97	2.08	2.24	2.08	0.04	
des	0.40	0.50		0.60	0.65	0.70	0.65	0.02	
diffeq	0.45	0.55		0.15	0.17	0.18	0.17	0.01	
dsip	0.35	0.50		0.54	0.59	0.73	0.59	0.02	
elliptic	0.50	0.55		0.73	0.80	0.89	0.80	0.02	
ex1010	0.50	0.55		2.57	2.68	2.85	2.68	0.05	
ex5p	0.45	0.50		0.31	0.33	0.35	0.33	0.01	
frisc	0.50	0.60		0.92	0.99	1.11	0.99	0.03	
misex3	0.40	0.50		0.43	0.47	0.50	0.47	0.01	
pdc	0.50	0.55		2.54	2.69	2.83	2.69	0.04	
s298	0.40	0.50		0.36	0.39	0.42	0.39	0.01	
s38417	0.45	0.55		1.05	1.14	1.26	1.14	0.03	
s38584.1	0.40	0.55		1.26	1.37	1.57	1.37	0.04	
seq	0.45	0.50		0.56	0.59	0.64	0.59	0.01	
spla	0.45	0.55		1.72	1.84	1.98	1.84	0.04	
tseng	0.45	0.55		0.14	0.16	0.18	0.16	0.01	
Mean	0.44	0.53		0.89	0.95	1.03	0.95	0.02	

Table 7.2: Statistics of minimum energy and energy-minimizing voltage for **Pathfinder knowledge-based repair**.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

7.3 Full-Knowledge Routing

At this time, Pathfinder is still considered the highest-quality practical router available for general field-programmable gate arrays (FPGAs). Combined with extensive physical characterization of each chip and the noise reduction techniques in Rubin [71, 72], Pathfinder routing is the current most effective solution for CSM. Full-knowledge full-design routing of this type is not currently a practical real-world approach, due to the prohibitive time cost of comprehensive characterization (hours to weeks) and individualized CAD (hours to days) for each chip, as well as the cost of storing large amounts of per-chip data. However, the difference between oblivious loading (DFVS) and full-knowledge full-design Pathfinder routing does provide an excellent measure for the maximum possible utility of specializing each design for each chip.

Figures 7.7 to 7.9 show the delay/energy improvements and extra CAD/load time costs resulting from using full-knowledge full-design routing instead of simply knowledge-based repair. For all 20 benchmarks, the median energy savings relative to knowledge-based repair are around 12% (see Table 7.4), at a cost of approximately 1.5 orders of magnitude more load time. The switch to full-design routing accounts for the remaining 22% of the utility of component specialization (see Table 7.5).

We can now answer the key question of this chapter: just how much of the benefit of component specialization do we achieve using lightweight CYA? The answer is that CYA provides about 52% of full-knowledge's energy savings relative to oblivious (DFVS) loading (Table 7.5), and about 83% of its savings relative to conventional loading (Table 7.6).

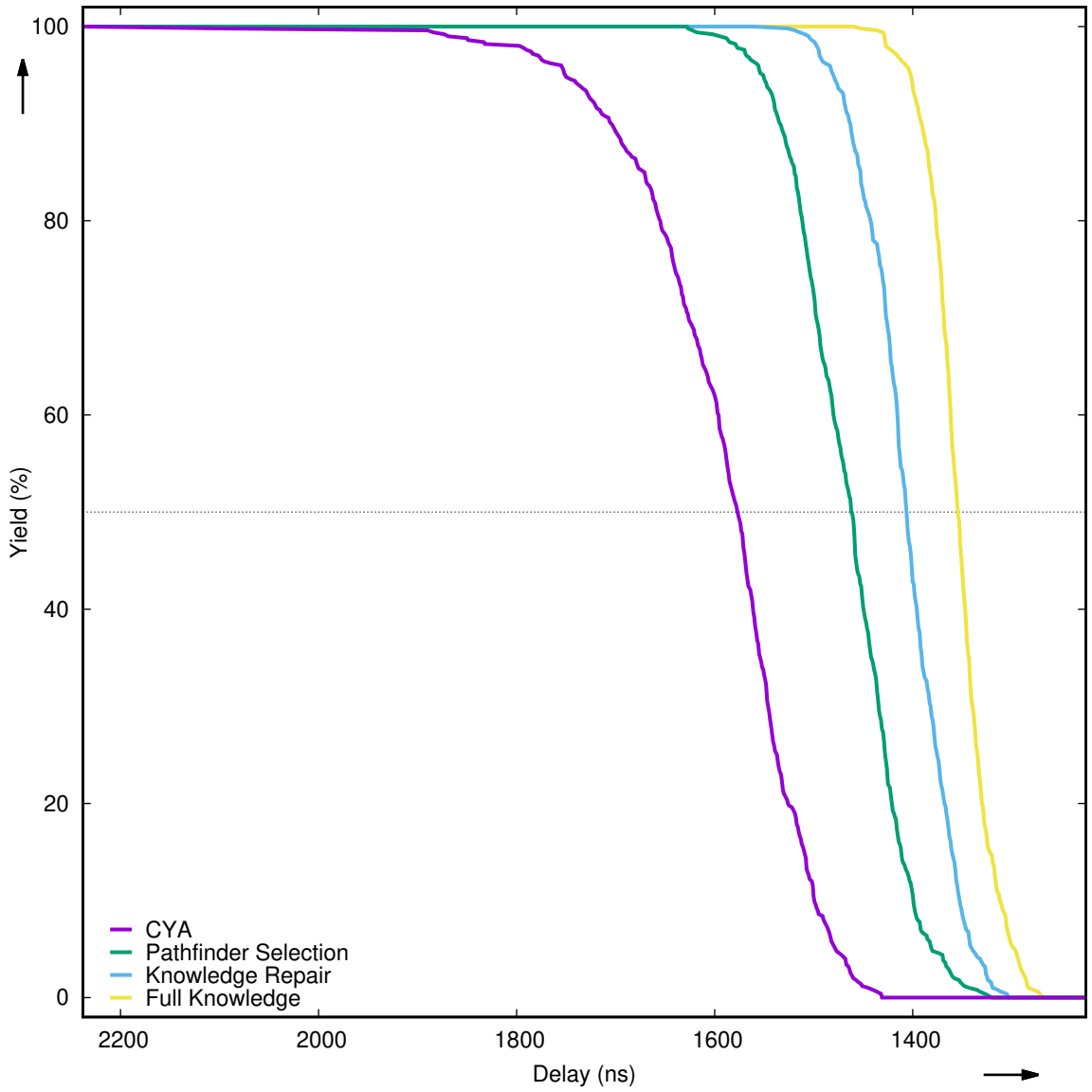


Figure 7.7: Delay parametric yield for **CYA, Pathfinder-based alternative selection, Pathfinder knowledge-based repair, and full-knowledge routing.**

Experiment parameters: V_{dd} = 0.6V, `des`, 22nm technology, $\sigma_{V_{th}}$ = 0.0364V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

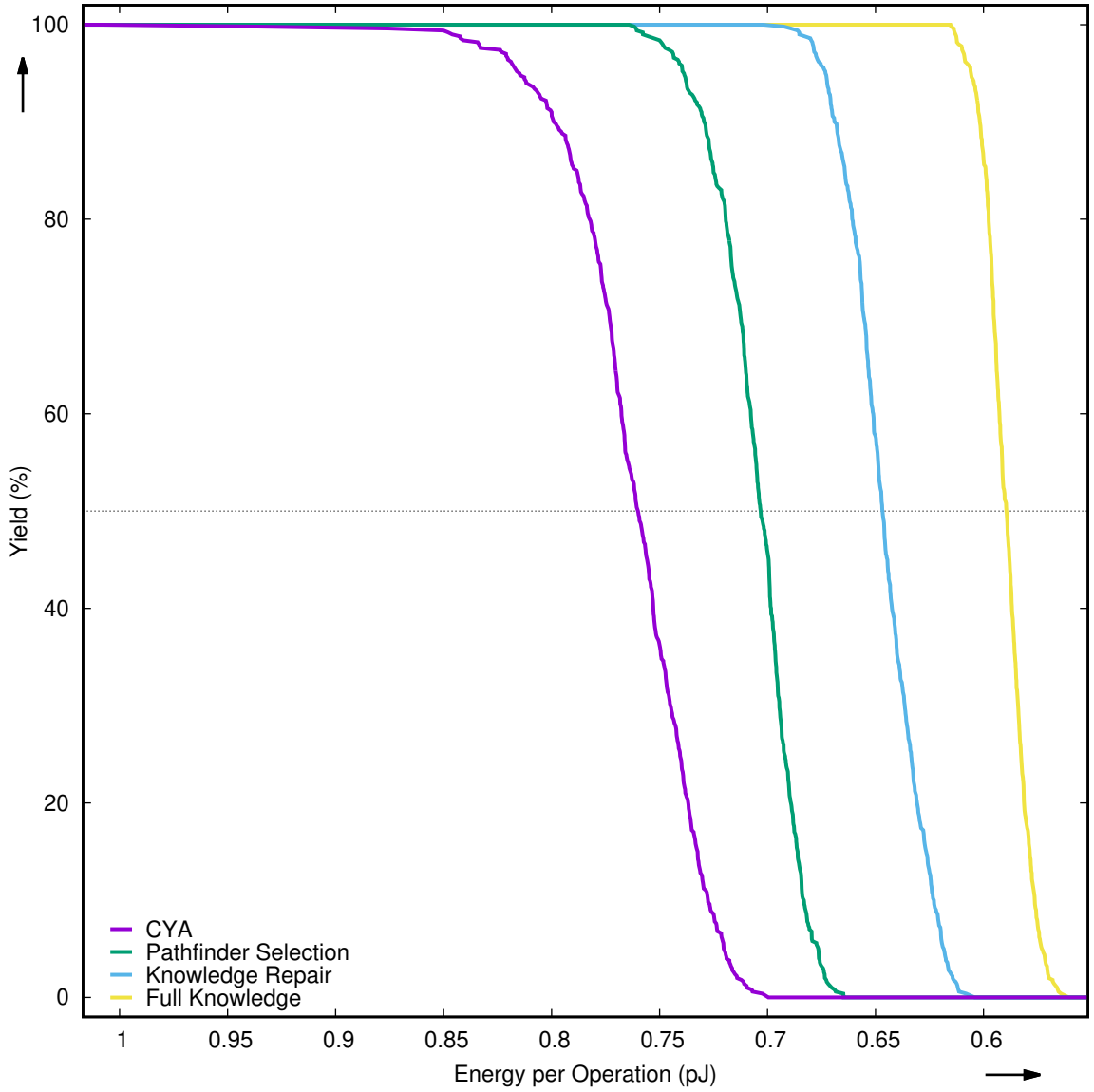


Figure 7.8: Energy parametric yield for **CYA, Pathfinder-based alternative selection, Pathfinder knowledge-based repair, and full-knowledge routing.**

Experiment parameters: energy-optimizing V_{dd}, `des`, 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

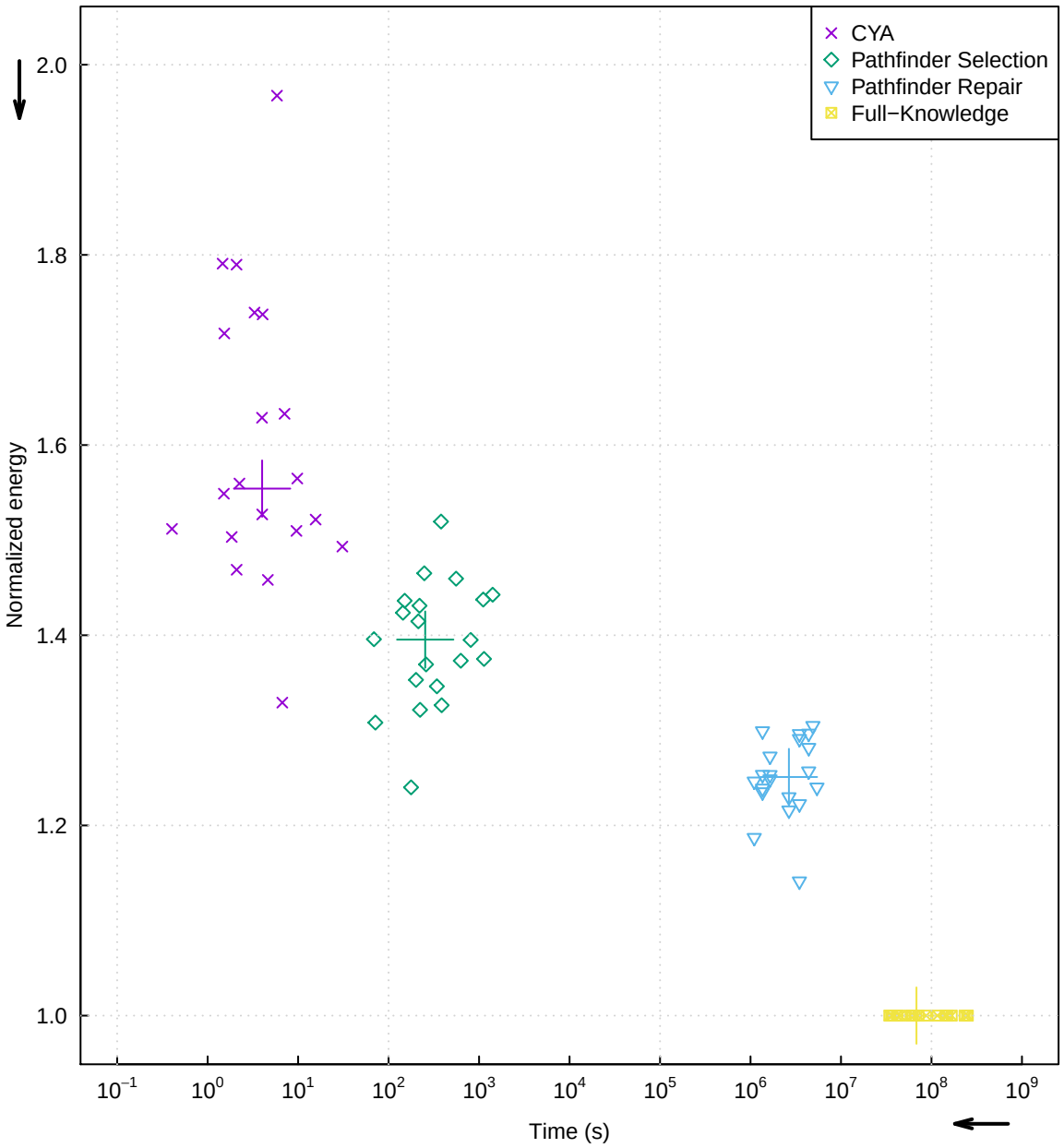


Figure 7.9: Energy vs. CAD/loading time for **CYA, Pathfinder-based alternative selection, Pathfinder knowledge-based repair, and full-knowledge routing**. Each point is the 95th percentile of normalized energy consumption amongst 20 chips, for one of the Toronto 20 [6] benchmarks. Normalization for each chip was performed with respect to the energy consumption achievable for that same chip using full-knowledge routing. See [24] (graph source) for details of the timing model. The crosshairs show the median time and energy for each alternative selection method.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 20 chips.

name	Voltage (V) min	max	CDF	Energy (pJ) min	med	max	mean	s.d.	CDF
alu4	0.40	0.45		0.36	0.37	0.39	0.37	0.00	
apex2	0.40	0.45		0.50	0.52	0.55	0.52	0.01	
apex4	0.45	0.50		0.40	0.41	0.43	0.41	0.01	
bigkey	0.40	0.45		0.36	0.38	0.40	0.38	0.01	
clma	0.45	0.50		1.60	1.68	1.77	1.68	0.03	
des	0.45	0.45		0.56	0.59	0.61	0.59	0.01	
diffeq	0.45	0.50		0.12	0.13	0.14	0.13	0.00	
dsip	0.35	0.40		0.48	0.50	0.53	0.50	0.01	
elliptic	0.45	0.50		0.63	0.67	0.71	0.67	0.01	
ex1010	0.45	0.55		2.00	2.09	2.26	2.10	0.04	
ex5p	0.40	0.50		0.27	0.28	0.30	0.28	0.01	
frisc	0.45	0.50		0.75	0.78	0.85	0.79	0.02	
misex3	0.40	0.45		0.37	0.38	0.40	0.39	0.01	
pdc	0.45	0.50		2.04	2.11	2.26	2.11	0.03	
s298	0.40	0.45		0.30	0.31	0.33	0.31	0.01	
s38417	0.45	0.50		0.87	0.90	0.96	0.90	0.01	
s38584.1	0.40	0.45		1.12	1.16	1.22	1.16	0.01	
seq	0.40	0.45		0.47	0.49	0.51	0.49	0.01	
spla	0.45	0.50		1.40	1.46	1.52	1.47	0.02	
tseng	0.45	0.50		0.12	0.13	0.14	0.13	0.00	
Mean	0.42	0.48		0.74	0.77	0.81	0.77	0.01	

Table 7.3: Statistics of minimum energy and energy-minimizing voltage for **full-knowledge, full-design routing**.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

7.4 CYA and the Costs and Benefits of Component-Specific Mapping

At this point it is useful to review and summarize what this series of experiments can teach us about the achievements and limitations of the present implementation of CYA, particularly in terms of what we can learn about the capabilities and costs of CSM. This will enable us to both establish the likely envelope of potential future advances and, in Chapter 8, contemplate further directions for exploration and development.

We begin with some summary data and visualizations. Table 7.4 shows the median energy usage for each loading method and each design across all 500 simulated chips. The steady energy efficiency improvements are clear as we proceed from conventional loading, to oblivious (DFVS) loading, to CYA, through more costly CSM methods, all the way to perfect chips.

Figure 7.10 and Table 7.5 report the degree to which CYA, Pathfinder-based alternative selection, and Pathfinder knowledge-based repair achieve the energy savings offered by full-design full-knowledge routing relative to oblivious (DFVS) loading. This enables us to distinguish the portion of the energy savings specifically attributable to each component of CSM.

Figure 7.11 and Table 7.6 report a similar comparison, only now savings are measured relative to the gap between full-knowledge routing and conventional loading. The advantage of oblivious loading/DFVS over conventional loading is also shown in these two charts. This demonstrates the advantages offered by CSM above and beyond the baseline utility of DFVS, which is implicitly part of all tested CSM strategies.

Finally, Figure 7.12 and Table 7.7 report the degree to which each loading scheme

mitigates the conventional-loading energy impact of variations, relative to energy optimization on defect-free chips. These comparisons illustrate the degree to which conventional loading is leaving energy savings on the table relative to all tested variation-mitigation strategies. They also allow us to begin to contemplate the value of these types of defect mitigation compared to fabrication-time defect reduction in improving energy efficiency.

As discussed in Chapter 6, CYA achieves a 61% reduction in total energy usage, compared to that of conventional loading (Table 6.6), and recovers about 70% of the energy lost due to variations (Table 6.7). This accounts for 83% of the savings relative to conventional loading that are achieved by full-knowledge routing (Table 7.6).

Focusing specifically on the benefits of mapping specialization (above and beyond the contributions of DFVS to CYA's success), we can examine Table 7.5 to peel away the layers of the onion one by one. From this table, we see that CYA achieves about half (52%) of the total benefit of CSM, switching to "smarter" alternative selection grants another 12% improvement, and removing the limits on alternatives saves yet another 14%. This leaves only 22% of the benefits of CSM that we miss out on by not doing unfettered full-knowledge routing of the entire design.

name	Conv	Obliv	CYA	Pathfinder Selection	Knowledge Repair	Full Knowledge	Nom
alu4	1.30	0.70	0.53	0.49	0.44	0.37	0.20
apex2	1.80	1.00	0.77	0.71	0.63	0.52	0.28
apex4	1.50	0.81	0.61	0.58	0.51	0.41	0.21
bigkey	1.30	0.74	0.54	0.51	0.44	0.38	0.20
clma	7.20	3.50	2.50	2.30	2.10	1.70	0.73
des	2.00	1.00	0.76	0.70	0.65	0.59	0.30
diffeq	0.51	0.27	0.20	0.18	0.16	0.13	0.07
dsip	1.80	1.00	0.74	0.69	0.59	0.50	0.27
elliptic	2.80	1.30	0.96	0.86	0.80	0.67	0.30
ex1010	8.60	4.20	3.20	3.00	2.70	2.10	0.93
ex5p	0.96	0.53	0.39	0.37	0.33	0.28	0.15
frisc	3.80	1.70	1.30	1.10	0.99	0.78	0.32
misex3	1.40	0.74	0.56	0.52	0.46	0.38	0.21
pdc	8.10	4.20	3.20	2.90	2.70	2.10	0.98
s298	1.20	0.61	0.46	0.44	0.39	0.31	0.18
s38417	3.40	1.90	1.50	1.30	1.10	0.90	0.48
s38584.1	4.00	2.20	1.60	1.50	1.40	1.20	0.62
seq	1.70	0.93	0.71	0.65	0.60	0.49	0.26
spla	5.80	2.90	2.10	2.00	1.80	1.50	0.76
tseng	0.49	0.25	0.19	0.17	0.16	0.13	0.07
Mean	2.98	1.53	1.14	1.05	0.95	0.77	0.38

Table 7.4: Median energy usage (in pJ) of each of the Toronto 20 designs with $\sigma_{V_{th}} = 0.0364$V under conventional loading, oblivious loading (DFVS), CYA, Pathfinder-based alternative selection, Pathfinder knowledge-based repair, and full-knowledge routing, compared with optimized energy usage on nominal chips.

Note that the smaller sample size of the experiments in this chapter (due to the high computational costs of full-knowledge routing) means that the conventional, oblivious, and CYA results in this table are slightly different from those reported in Tables 6.3 to 6.5.

Experiment parameters: 22nm technology, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

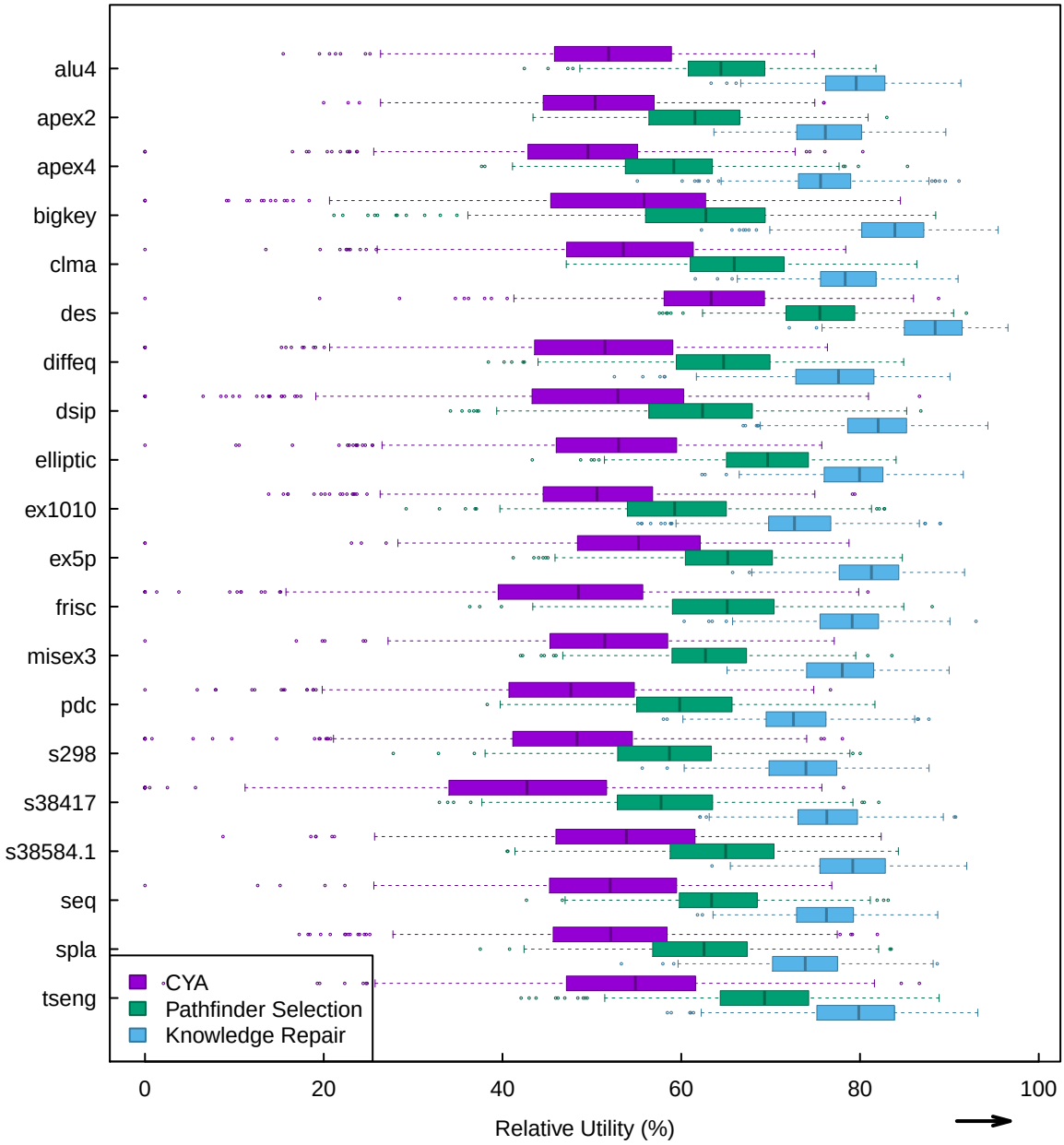


Figure 7.10: Percentage energy savings relative to the gap between **full-design full-knowledge routing and oblivious loading (DFVS)**. For each benchmark, the box-and-whisker and outliers are shown for CYA, Pathfinder-based alternative selection, and Pathfinder knowledge-based repair. This graph shows that CYA achieves a substantial portion of the benefits available to more costly CSM techniques. Table 7.5 records the median energy savings for each loading method.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

name	Oblivious	CYA	Pathfinder Selection	Knowledge Repair	Full Knowledge
alu4	0	52	64	80	100
apex2	0	50	62	76	100
apex4	0	50	59	76	100
bigkey	0	56	63	84	100
clma	0	54	66	78	100
des	0	63	76	88	100
diffeq	0	51	65	78	100
dsip	0	53	62	82	100
elliptic	0	53	70	80	100
ex1010	0	51	59	73	100
ex5p	0	55	65	81	100
frisc	0	48	65	79	100
misex3	0	51	63	78	100
pdc	0	48	60	73	100
s298	0	48	59	74	100
s38417	0	43	58	76	100
s38584.1	0	54	65	79	100
seq	0	52	63	76	100
spla	0	52	63	74	100
tseng	0	55	69	80	100
Mean	0	52	64	78	100

Table 7.5: Median percentage energy savings relative to the gap between **full-design full-knowledge routing and oblivious loading (DFVS)**. These data show that CYA achieves a substantial portion of the benefits available to more costly CSM techniques. A graphical representation of the full distributions is shown in Figure 7.10.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

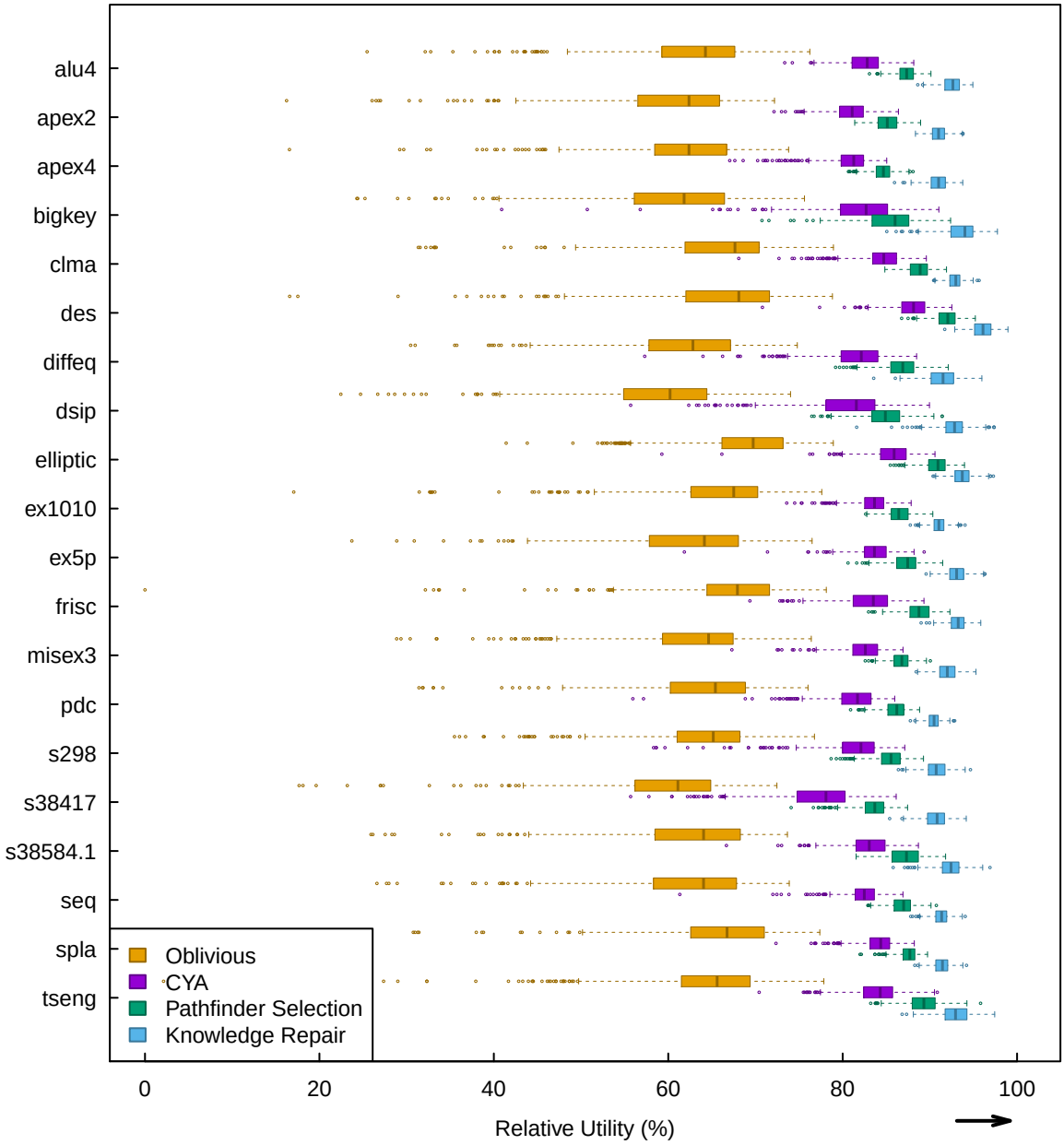


Figure 7.11: Percentage energy savings relative to the gap between **full-knowledge routing and conventional loading**. For each benchmark, the box-and-whisker and outliers are shown for oblivious loading (DFVS), CYA, Pathfinder-based alternative selection, and Pathfinder knowledge-based repair. This graph highlights the portion of each technique's energy savings that are directly attributable to mapping specialization, above and beyond the benefits of DFVS. Table 7.6 records the median energy savings for each loading method.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

name	Oblivious	CYA	Pathfinder Selection	Knowledge Repair	Full Knowledge
alu4	64	83	87	93	100
apex2	62	81	85	91	100
apex4	62	81	85	91	100
bigkey	62	83	86	94	100
clma	68	85	89	93	100
des	68	88	92	96	100
diffeq	63	82	87	92	100
dsip	60	82	85	93	100
elliptic	70	86	91	94	100
ex1010	68	84	86	91	100
ex5p	64	84	87	93	100
frisc	68	84	89	93	100
misex3	65	83	87	92	100
pdc	65	82	86	90	100
s298	65	82	86	91	100
s38417	61	78	84	91	100
s38584.1	64	83	87	92	100
seq	64	82	87	91	100
spla	67	84	88	91	100
tseng	66	84	89	93	100
Mean	65	83	87	92	100

Table 7.6: Median percentage energy savings relative to the gap between **full-knowledge routing and conventional loading**. These data highlight the portion of each technique's energy savings that are attributable to mapping specialization, above and beyond the benefits of DFVS. A graphical representation of the full distributions is shown in Figure 7.11.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

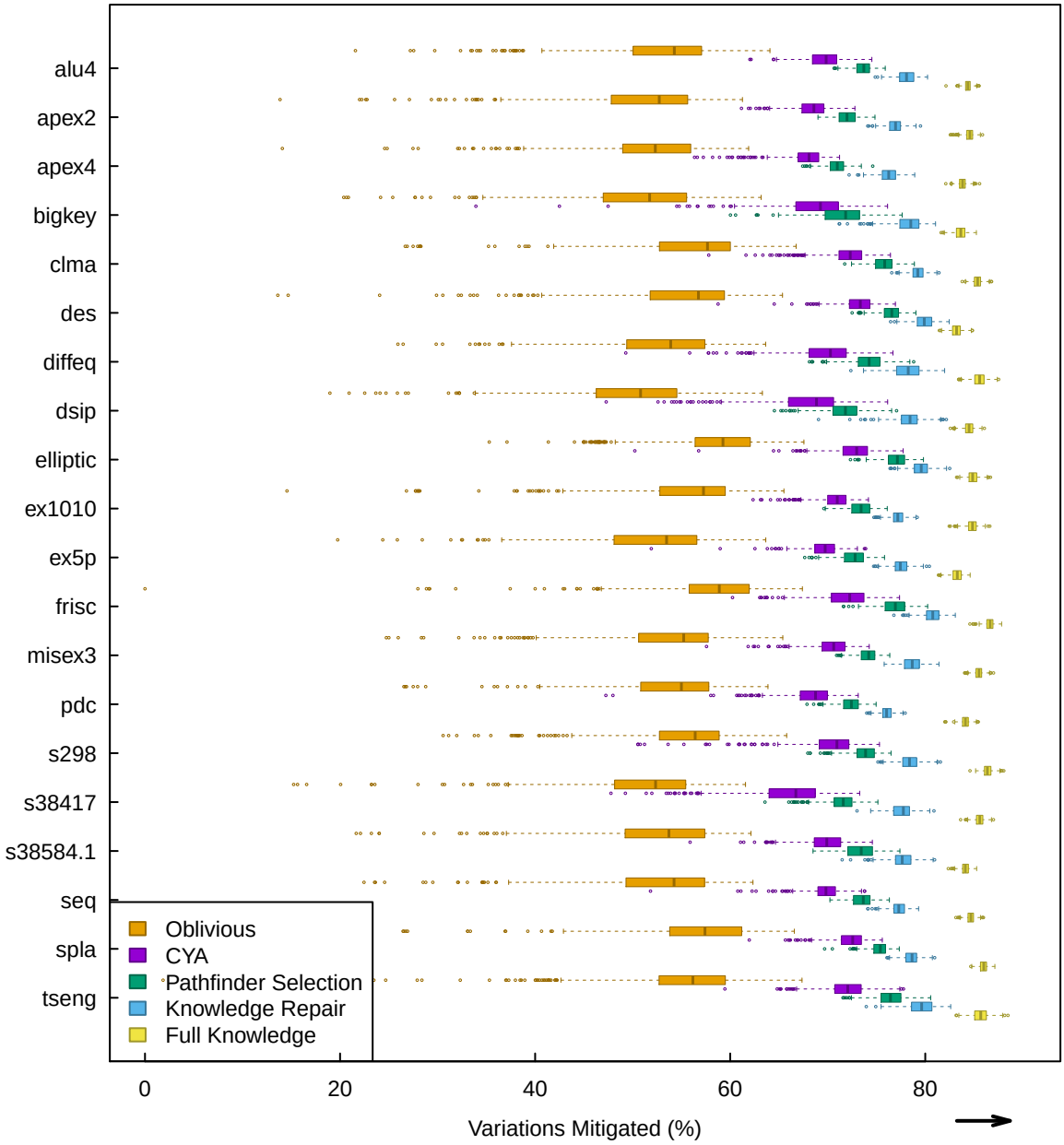


Figure 7.12: Percentage energy savings relative to the gap between **nominal chips and conventional loading with variations**. For each benchmark, the box-and-whisker and outliers are shown for oblivious loading (DFVS), CYA, Pathfinder-based alternative selection, Pathfinder knowledge-based repair, and full-knowledge routing. These comparisons illustrate that a significant portion of the energy lost to variations that can be recovered using relatively lightweight variation-mitigation strategies. Table 7.7 records the median energy savings for each loading method.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

name	Oblivious	CYA	Pathfinder Selection	Knowledge Repair	Full Knowledge
alu4	54	70	74	78	84
apex2	53	69	72	77	85
apex4	52	68	71	76	84
bigkey	52	69	72	79	84
clma	58	72	76	79	85
des	57	73	77	80	83
diffeq	54	70	74	78	86
dsip	51	69	72	78	84
elliptic	59	73	77	80	85
ex1010	57	71	73	77	85
ex5p	53	70	73	77	83
frisc	59	72	77	81	87
misex3	55	71	74	79	86
pdc	55	69	72	76	84
s298	56	71	74	78	86
s38417	52	67	72	78	86
s38584.1	54	70	73	78	84
seq	54	70	74	77	85
spla	57	73	75	79	86
tseng	56	72	76	80	86
Mean	55	70	74	78	85

Table 7.7: Median percentage energy savings relative to the gap between **nominal chips and conventional loading with variations**. These comparisons illustrate that a significant portion of the energy lost to variations that can be recovered using relatively lightweight variation-mitigation strategies. A graphical representation of the full distributions is shown in Figure 7.12.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 500 chips.

Chapter 8

Conclusions and Future Prospects

Choose-Your-own-Adventure (CYA) is a powerful and practical tool for mitigation of defects and variations in field-programmable gate arrays (FPGAs).

With computer-aided design (CAD) overheads that are a function of the design rather than of the number of chips to be programmed, and with a load-time overhead of seconds, CYA is able to reliably produce working mappings for arbitrary designs on chips with hard-defect densities of up to 81 defects/cm^2, and maintains 95% yields up to 2400 defects/cm^2 (22nm technology, Toronto 20 benchmarks [6], Section 4.7.7). Moreover, many of the mapping failures at this defect rate are likely addressable in future work by minor improvements to CYA (permuting LUT inputs and outputs, allowing logic to move across CLBs, etc.), complementary solutions (such as [46]), and lightweight testing methodologies to detect and reject those chips with the most severe defect clusters at the factory.

In the presence of parametric defects (variations), CYA reduces energy costs by approximately 61% compared to conventional loading (22nm technology, Toronto 20 benchmarks). This amounts to a recovery of 70% of the energy lost to variations (see Chapter 6). Moreover, despite its low overhead, CYA attains half (52%) of the

energy savings offered by the most extreme (and impractically expensive) form of component-specific mapping (CSM), full-design full-knowledge routing (see Chapter 7). These advantages are likely to increase even further with the use of higher-variation technologies.

These benefits mean that CYA, even without any further improvements, has the ability to address many important current challenges. Low-overhead CSM opens up the possibility of using high-variation technologies and architectures that would otherwise be impractical or impossible to deploy, such as smaller feature scales and nanoPLAs, and of using more cheaply-fabricated (i.e., higher-defect/higher-variation) versions of existing technologies. Moreover, CYA has already inspired and serves as the foundation for effective methodologies for repair of aging-induced variation (COSMIC TRIP [22]) and for quick FPGA loading with incremental online optimization (DRAGON [24]).[8]

In addition to these possible applications for CYA in its current form, there are several interesting research areas for further development, to increase its effectiveness for these and other purposes. More sophisticated slack budgeting tools or moving from two-point net to long-path slack timing could offer additional speedups during delay repair and optimization. Bitstream storage and loading overhead can be reduced and/or the breadth of alternative options increased by implementing composable paths (partial paths that can be combined in multiple ways) and other compression techniques. Parallel testing methodologies to test multiple nets simultaneously (as mentioned in Section 5.1.3) can be developed to reduce bootstrap time. Statistical modeling of likely types of defect combinations may help generate more effective sets of alternatives and efficiently select those alternatives most likely to repair a given net. As discussed in Chapter 7, Pathfinder-based alternative selection makes more effective

[8]These projects were collaborations with Geisen, Gojman, and DeHon.

use of the available alternatives than CYA's current single-pass, stateless, greedy selection model, albeit at somewhat greater load-time compute requirements; further effort along these lines can develop algorithms with intermediate cost/intelligence profiles to optimize this trade-off.

Architecture-level adjustments can also make CYA in particular and CSM in general cheaper and more effective. As discussed in Section 4.2.4, one possibility along these lines might be to implement directly addressable configurations, instead of block-based loading. Other possibilities might include modifying topologies to enable easy reservation of repair channels (to avoid the types of switch box (S-Box) difficulties discussed in Appendix A.2), or making architectures more testable (for general CSM).

Perhaps the most important avenue for future study is the development of CSM-aware quality control methodologies — how do we decide which imperfect chips can still be certified "CYA-mappable"? Relatedly, we may wish to ship with additional spare resources for lifetime repair of aging-related defects and variations (as is already common practice for storage technologies); development of proper budgeting methods for these resources will also be critical.

With these sorts of tools in hand, CYA can serve as a transition pathway towards a world in which high-variation chips are the norm and CSM of many types becomes the new standard. Scaling can continue to push beyond the yield and energy barriers which are presently emplaced by static mapping (see [61]) and only partly resolved by large-scale core sparing. In the long term, CYA and related cheap CSM techniques can even open the field for newer and higher-variation technologies which would otherwise be inconceivable.

Appendices

Appendix A

Topological Systematic Error

It is important, in establishing the benefits of the component-specific mapping (CSM) strategies discussed in this thesis, to confirm that the demonstrated benefits do not result, in part, from other factors. In particular, CSM could possibly receive an unfair advantage in comparisons with static maps from the addition of reserved tracks for repair (Section 4.1.1) which are not available to the base (static) route. Possible advantages could arise either from relief of congestion by the increase in routing options or from the slightly different topology of the reserved tracks (due to the way the reservation was specified). This topological difference results in reserved tracks having both a slightly higher speed and slightly different connectivity with each other and with the base tracks, which may provide different routing options. The purpose of this appendix is to explore the effects of the presence of these repair resources on CSM/static mapping comparisons.

In order to test these effects, I separately examine, first, the effects of increasing the number of tracks (by expanding the base channel width), second, the effects of the repair resource speeds (by examining a simplified design), and, third, the effects of the differing topology of the repair resources (by unlocking those resources for

base routing). Finally, I bring all of these effects together to enable calibration of comparisons between Choose-Your-own-Adventure (CYA) and static mapping delay and energy measurements. All tests in this appendix are performed on nominal chips to isolate the effects of resource quantity/speed/topology in the absence of variations.

Figure A.1 demonstrates that the effects of repair resource quantity on the delay are largely eliminated by using a base channel width of $1.2W_{min}$. For 14 of the 20 benchmarks, using this "relaxed" channel width causes delay to attain its lower bound value, showing that having more (reserved) resources available will not offer additional delay advantages. For the remaining 6 benchmarks, the data suggest that any delay advantage offered by those additional resources is likely to be fairly small.

Speed tests (Figure A.2) show that the repair resources are at most 1% faster than the base resources, so this difference is not significant in CYA/static mapping comparisons.

The topological differences between the reserved resources and the base resources could in principle have a more noticeable effect. This can be seen by comparing static mapping delays with reserved resources "locked" and "unlocked" (Figure A.3) — the median speedup of the "unlocked" routes is about 9%. However, CYA does not in fact receive much unfair advantage from this difference (Figure A.4), as it is constrained to use the locked base route when not repairing a defect.[9] Thus, the median speedup for CYA relative to relaxed locked static routes is only about 2%.

For energy calculations, the impacts of the reserved resources are even smaller. The resource differences have significant effects on energy consumption only through the connection between static energy (leakage) and delay. In the most relevant voltage range for energy minimization in the presence of variations ($V_{dd} \geq 0.4$V)[10], 75% of the

[9] Full-knowledge routing, on the other hand, *will* benefit from the topological differences.

[10] All energy minima in all experiments occurred at or above $V_{dd} = 0.4$V, except in 2% of the `dsip` full-knowledge routes, which achieved minimum energy at $V_{dd} = 0.35$V (Table 7.3).

benchmarks show less than 1% energy skew, with the worst outlier (`frisc`) showing only 8% skew at 0.4V.

As a consequence, the potential systematic error in my reporting of the energy efficiency advantages of CYA in particular and CSM in general is almost always less than one tenth of the utility shown in my data tables in Chapters 6 and 7. The few outliers noted in this appendix (`frisc` being the worst by a large margin) are not strong outliers in those tables (e.g., Table 6.6), suggesting that the true impacts of this systematic error are even less significant overall.

The following sections discuss more details of my systematic error tests.

A.1 Resource Quantity

My first set of tests addresses the impact of the quantity of resources available to routing.

It is common practice in field-programmable gate array (FPGA) research to begin analysis by finding the minimum-sized FPGA (minimum number of look-up tables (LUTs) and minimum channel width) that fits a given computation or benchmark. However, at this minimum channel width, it is highly likely that some nets will take non-minimal paths, in terms of both Manhattan distance and delay. Often, the nets so affected either are intrinsically critical paths or become critical paths as a result of the non-minimal routing. In order to eliminate the distorting influence of this routing "stress", typically one adds extra tracks to the minimal channel width to "relax" the route [79, 54]. The goal is to provide sufficient routing resources to ensure that the critical path will follow a delay-optimal route.

To reliably estimate the effectiveness of this relaxation, it is useful to compute a congestion-oblivious delay lower bound [71, 72]. Each net is assigned a delay-optimal

path as if it were routed in isolation, without having to contend for resources with other nets. If a route achieves this delay after full routing (congestion negotiation, etc.), then adding more resources of the same type cannot reduce delay.

Figure A.1 shows that, as expected, the high-stress routes frequently fail to achieve the congestion-oblivious lower bound delay. With the addition of approximately 20% extra tracks, 14 of the 20 benchmarks achieve the lower bound delay. Even for the remaining 6 benchmarks, the excess delay (achieved delay minus lower bound) is small. This suggests that, provided that all CSM/static comparisons using these benchmarks are done at a "relaxed" channel width of $1.2W_{min}$ or higher, the distorting effects of CSM having access to a few additional reserved channels should mostly be ignorable.

It should be noted that the additional tracks in these experiments were added by expanding the simulated chip. Adding resources in this fashion lengthens wires and increases the connectivity of some resources (more inputs to multiplexors and higher fanout (stubs) from each wire). This does increase the overall delay and energy consumption of the chip relative to the stressed route, even at the lower bound, but it does not affect my ability to draw comparisons to CYA routes performed on chips at the same (relaxed) size.

A.2 Resource Properties

My remaining tests address the fact that all resources may not be equal. Reserving tracks for repair that are unavailable for base routing may result in CSM having access not only to more resources, but also to potentially faster resources (due to a reduction in capacitance resulting from the topological changes needed to reserve tracks). In addition, the slightly differing connectivity of the repair resources could potentially

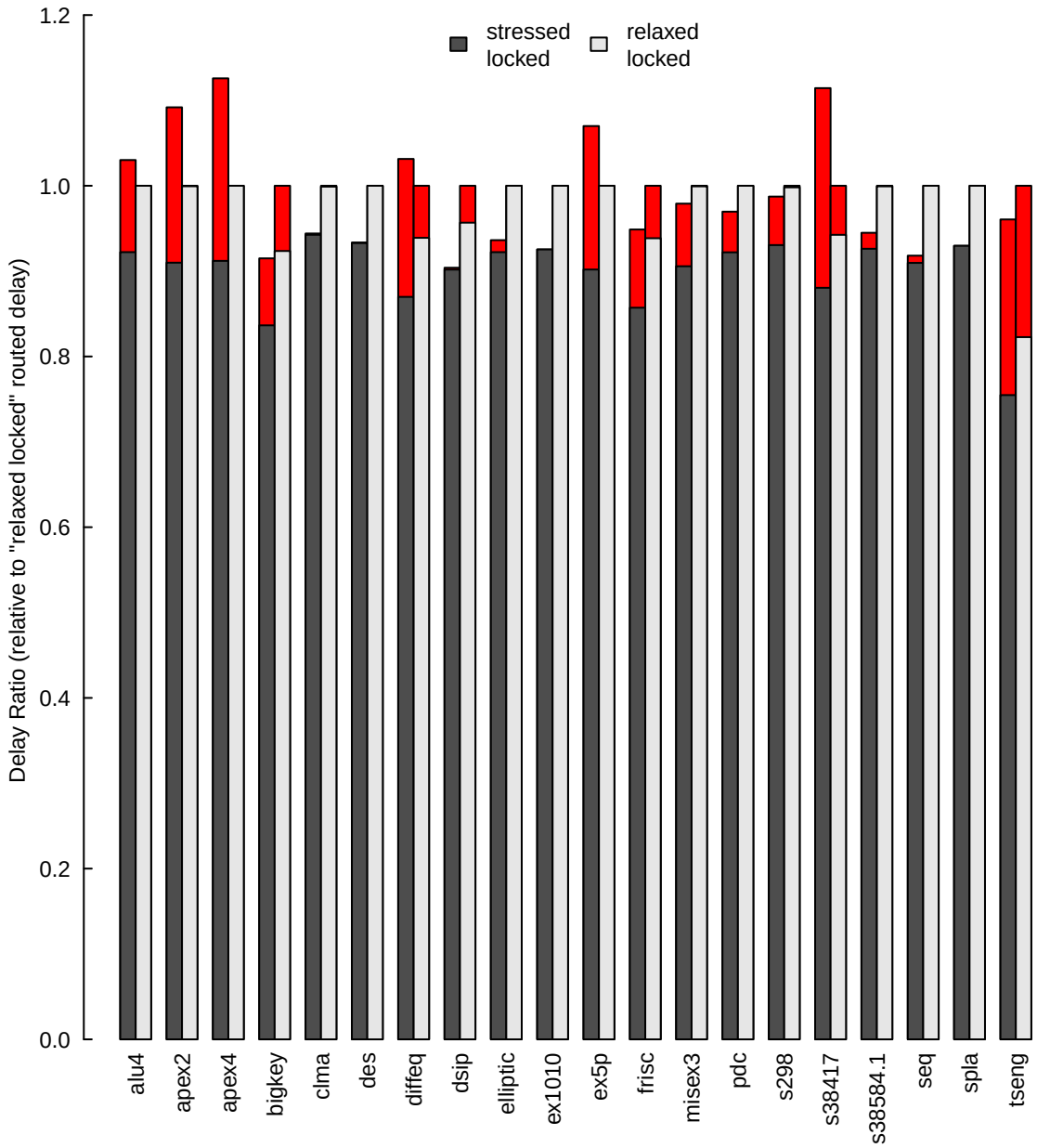


Figure A.1: Relative delay of stressed (W_{min}) vs. relaxed ($1.2W_{min}$) static routes at 0.8V, with reserved resources present but locked. Grey bars show the delay lower bound for each design at each channel width, while red bars stacked on top indicate the excess delay of the actual route. Note that in most cases relaxing the route cures the delay excess entirely, and the excess delay is small in the other cases.

Experiment parameters: 22nm technology, 16 reserved tracks, VPR 5 [53], single-driver interconnect, nominal chips.

provide an advantage to routes permitted to use these resources. In this section I examine how these issues arise in my experimental framework and demonstrate that their impacts on my results are nevertheless negligible.

There may be some subtle differences amongst tracks in the base resources, but these differences are addressed by the lower bound analysis discussed in Appendix A.1. Differences between the base resources as a whole and those resources reserved for repair may be more significant. Stock VPR 5 [68] supports only Wilton switch boxes (S-Boxes) [55] for single driver architectures. By design, these S-Boxes break domains, which prevents clean track-based reservation. Consequently, in a joint effort with Hans Giesen, I added support for a disjoint "shadow channel" for reserved resources [24, 23]. While this reserved sub-channel is in many ways similar to the base channel, tracks in this (smaller) channel have slightly different connectivity and slightly lower capacitance (due to a reduced number of stubs) than the base tracks.

The resulting greater speed of the reserved resources can be cleanly demonstrated by mapping two clocked LUTs to opposite ends of the same row of an FPGA, at varying chip sizes, and comparing the delay of base vs. reserved track mappings of a two point net joining them. The reserved track speedup, while not zero, is always less than 1%, and so is essentially negligible for the purpose of my experiments (see Figure A.2).

The topological differences between base and reserved resources can result in more significant delay reduction for nets routed through reserved resources. To see the effects of this speedup, I compare the routed delays for each of the Toronto 20 [6] benchmarks, as computed with the reserved resources either "locked" (unavailable for use) or "unlocked" (available). Figure A.3 shows that unlocking the reserved resources for static mapping results in a median 9% reduction in delay across all V_{dd} values. However, a similar comparison between relaxed locked static mapping and

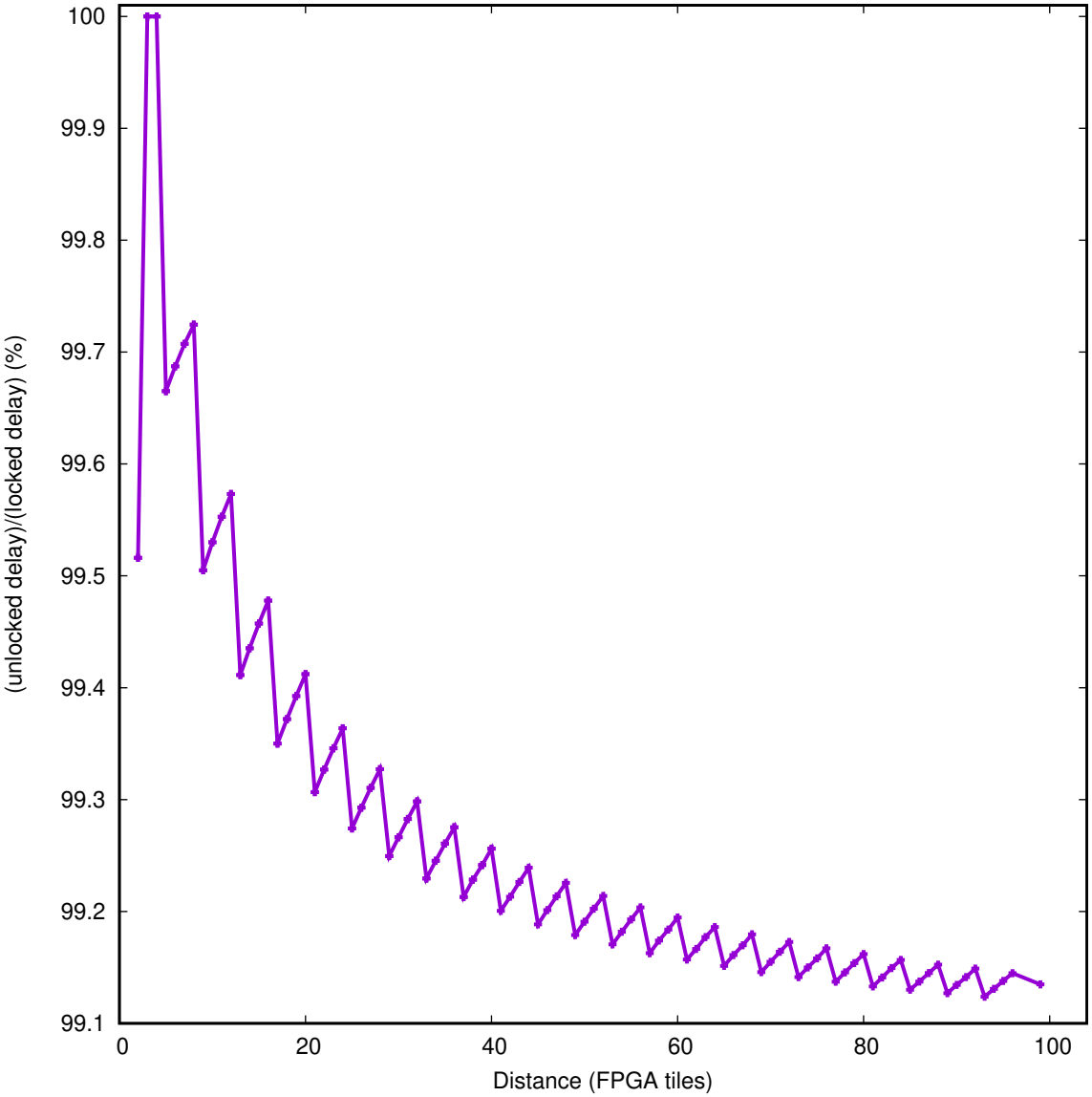


Figure A.2: Relative speed of reserved ("unlocked") vs. base ("locked") resources at 0.8V, for a single net routed between two LUTs at varying separations in the same row of an FPGA. The reserved path is always faster, but never by more than 1%.

Experiment parameters: 22nm technology, 16 reserved tracks, VPR 5 [53], single-driver interconnect, nominal chips.

CYA (Figures A.4 and A.5) shows that CYA actually receives little unfair benefit (median 2% across all voltages) from the topology difference. This is most likely because CYA does not make fundamental structural changes to the base ("locked") route, but rather updates it only incrementally and greedily, as needed.

As a result, in most cases of interest, CYA receives little unfair delay benefit from the reserved channel topology. In the few cases in the main text where this potential benefit is relevant (in Section 5.2.3), raw CYA results are shown alongside the same results rescaled by the ratio of relaxed locked to relaxed CYA delay in nominal chips.

Energy impacts are even less significant, and are mediated primarily through the relation of static energy to delay. As static energy is only a small component of the total energy throughout most of the voltage range in which energy minima are found, the energy advantage CYA receives from its repair resource topology relative to relaxed locked static mapping is less than 1% for the vast majority of benchmarks (see Figure A.6).

The outliers in this graph, such as `frisc`, are particularly instructive when I compare the nominal energy results measured in this appendix with the variation results presented in Chapter 6. Table A.1 shows that, in the presence of variations, none of the outliers in the nominal CYA vs. relaxed unlocked comparison are noticeably different from the other benchmarks. (Table 7.7 suggests that the same is true for Pathfinder alternative selection, Pathfinder repair, and full-knowledge full-design CSM methodologies.) This implies that the effects of the reserved resources become even more unimportant in the face of variations.

In consequence, it appears that any potential systematic error associated with the use of reserved resources for CSM has a *negligible effect on the key results* reported in this dissertation.

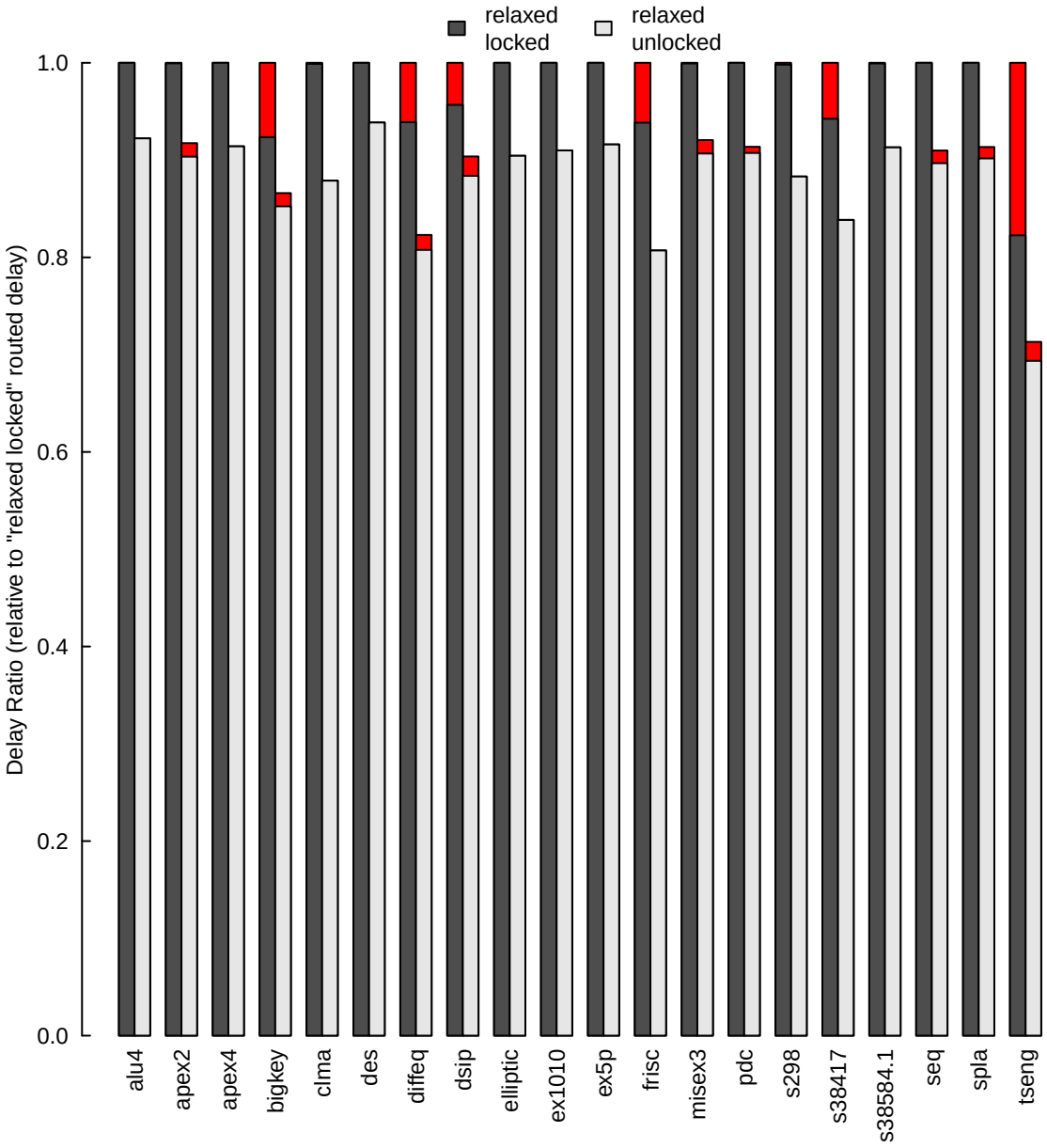


Figure A.3: Relative delay of unlocked (reserved resources available for routing) vs. locked (reserved resources unavailable) static routes at 0.8V, with channel widths relaxed to $1.2W_{min}$. Grey bars show the delay lower bound for each design under each test condition, while red bars stacked on top indicate the excess delay of the actual route. Because relaxation causes most designs to achieve their delay lower bound, the difference between the locked and unlocked delays is primarily attributable to the effects of the different properties of the reserved resources.

Experiment parameters: 22nm technology, 20% extra base tracks, 16 reserved tracks, VPR 5 [53], single-driver interconnect, nominal chips.

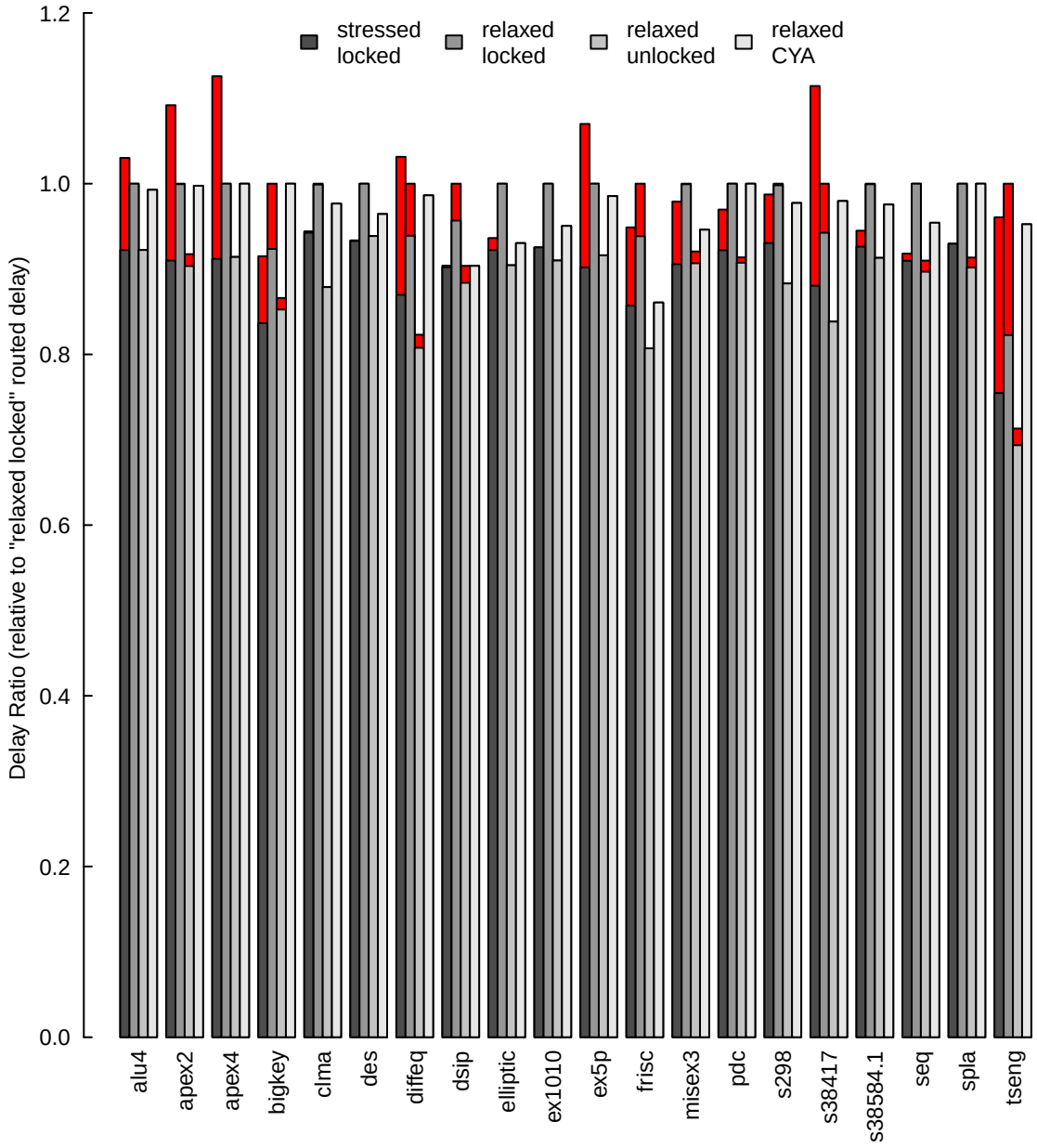


Figure A.4: Relative delay at 0.8V of stressed locked static mapping, relaxed unlocked static mapping, and $1.2W_{min}$ CYA, compared to relaxed locked static mapping. Grey bars show the delay lower bound for each design under each test condition, while red bars stacked on top indicate the excess delay of the actual route. Comparing CYA to relaxed locked static mapping results allows us to estimate the systematic advantage CYA receives from its use of reserved resources.

Experiment parameters: 22nm technology, 0% or 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, nominal chips.

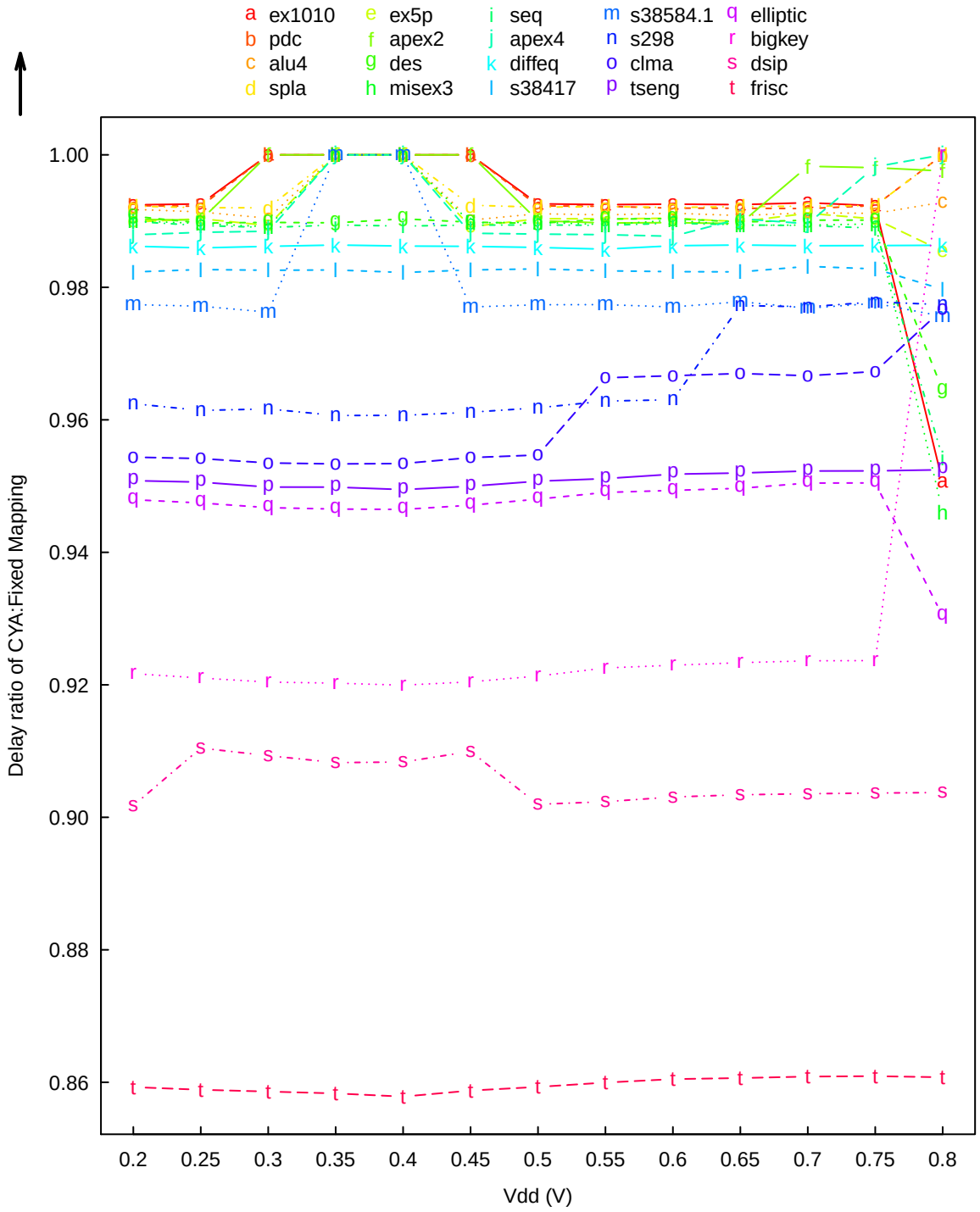


Figure A.5: Ratio of relaxed CYA delay to relaxed locked static mapping delay for each benchmark, across the full range of V_{dd} values tested.

Experiment parameters: 22nm technology, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, nominal chips.

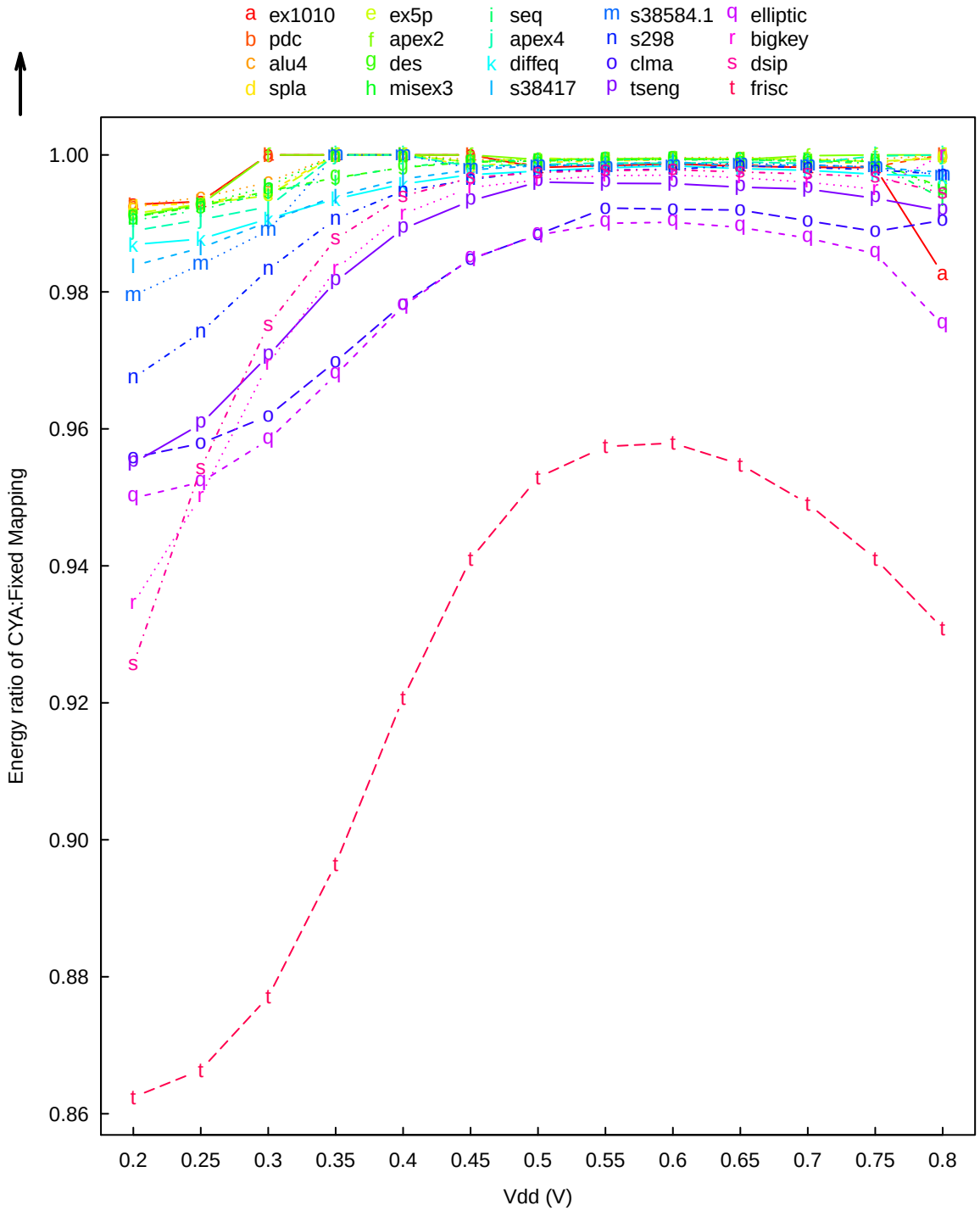


Figure A.6: Ratio of relaxed CYA energy to relaxed locked static mapping energy for each benchmark, across the full range of V_{dd} values tested.

Experiment parameters: 22nm technology, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, nominal chips.

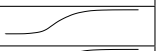
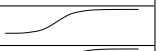
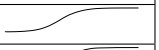
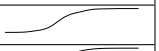
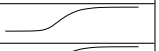
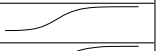
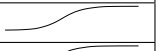
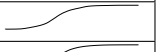
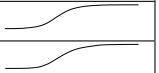
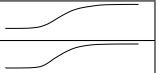
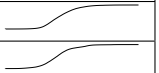
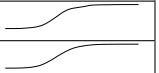
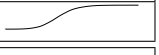
name	min	med	max	mean	CDF
alu4	0	24	61	25	
apex2	0	24	61	25	
apex4	0	24	61	25	
bigkey	0	26	62	26	
clma	0	27	65	27	
des	0	27	63	28	
diffeq	0	26	65	26	
dsip	0	26	60	26	
elliptic	0	26	66	26	
ex5p	0	25	61	26	
frisc	0	27	67	27	
misex3	0	25	64	26	
pdc	0	24	60	24	
s298	0	23	59	24	
s38584.1	0	24	58	25	
seq	0	25	61	26	
spla	0	25	64	26	
tseng	0	26	65	26	
Mean	0	25	62	26	

Table A.1: Percentage energy savings of CYA relative to dynamic frequency and voltage scaling (DFVS) mapping for all Toronto 20 [6] benchmarks.

Experiment parameters: 22nm technology, $\sigma_{V_{th}} = 0.0364$V, 20% extra base tracks, 16 reserved tracks, 64 alternatives, VPR 5 [53], single-driver interconnect, 10000 chips.

Appendix B

Architecture Files

B.1 4x4_fcin_1.00_fcout_1.00.arch

```
#based on 4lut_sanitized and 4x4lut_sanitized from the vpr
   4.3 dist
# with our computations of 22nm characteristics

# Uniform channels.  Each pin appears on only one side.
io_rat 4
chan_width_io 1
chan_width_x uniform 1
chan_width_y uniform 1

# Cluster of size 4, with 10 logic inputs.
inpin class: 0 bottom
inpin class: 0 left
inpin class: 0 top
inpin class: 0 right
inpin class: 0 bottom
inpin class: 0 left
inpin class: 0 top
inpin class: 0 right
inpin class: 0 bottom
inpin class: 0 left
outpin class: 1 top
outpin class: 1 right
outpin class: 1 bottom
```

```
outpin class: 1 left
inpin class: 2 global top   # Clock, global -> routed on a
   special resource.

# Class 0 -> logic cluster inputs, Class 1 -> Outputs, Class
   2 -> clock.

subblocks_per_clb 4
subblock_lut_size 4

#parameters needed only for detailed routing.
switch_block_type subset
Fc_type fractional
Fc_output 1
Fc_input 1
Fc_pad 1

# orig 1lut clb version, 4lut blocks multiply block size
   dependent factors by 2
#segment frequency: 1 length: 1 wire_switch: 0 opin_switch: 0
   Frac_cb: 1. \
#         Frac_sb: 1. Rmetal: 195 Cmetal: 0.7e-15
# R and C are per unit length so only double to compensate
   for clb size
# note only 1 seg type cause I'm staying simple and want
   fully buffered
segment frequency: 1 length: 4 wire_switch: 0 opin_switch: 0
   Frac_cb: 1. \
         Frac_sb: 1. Rmetal: 390 Cmetal: 1.4e-15


# Switch used as a tri-state buffer within the routing, and
   also as the
# output buffer used to drive from a CLB output pin to a
   routing wire.

# same as seg len 1 22nm, switches left untouched
switch 0  buffered: yes  R: 6553 Cin: 0.2e-15  Cout: 0.2e-15 \
       Tdel: 24e-12
#switch 0  buffered: yes  R: 7207 Cin: 0.2e-15  Cout: 0.2e-15
   \

# Used only by the area model.

# R_minW_nmos 1967
```

```
# R_minW_pmos 3738
R_minW_nmos 4000
R_minW_pmos 8000

# Timing info below.

C_ipin_cblock 0.2e-15


#ignored
T_ipin_cblock 72e-12

T_ipad 24e-12 # clk_to_Q + 2:1 mux
T_opad 24e-12   # Tsetup
T_sblk_opin_to_sblk_ipin 48e-12  # No local routing
T_clb_ipin_to_sblk_ipin 24e-12   # No local routing
T_sblk_opin_to_clb_opin 0.

# Delays through the BLE (LUT and a FF)
T_subblock T_comb: 24e-12 T_seq_in: 36e-12 T_seq_out: 24e-12
T_subblock T_comb: 24e-12 T_seq_in: 36e-12 T_seq_out: 24e-12
T_subblock T_comb: 24e-12 T_seq_in: 36e-12 T_seq_out: 24e-12
T_subblock T_comb: 24e-12 T_seq_in: 36e-12 T_seq_out: 24e-12
# device model parameters
Vdd 0.600                   # supply voltage (V)
Vth 0.202                   # threshold voltage (V)
Transistor_Width_Factor 4   # ratio of actual PMOS/NMOS width
   to minimum width
Transistor_Length 11        # effective channel length (nm)
```

B.2 4x4_fcin_0.50_fcout_0.25.arch

```
#based on 4lut_sanitized and 4x4lut_sanitized from the vpr
   4.3 dist
# with our computations of 22nm characteristics

# Uniform channels.  Each pin appears on only one side.
io_rat 4
chan_width_io 1
chan_width_x uniform 1
chan_width_y uniform 1

# Cluster of size 4, with 10 logic inputs.
inpin class: 0 bottom
inpin class: 0 left
inpin class: 0 top
inpin class: 0 right
inpin class: 0 bottom
inpin class: 0 left
inpin class: 0 top
inpin class: 0 right
inpin class: 0 bottom
inpin class: 0 left
outpin class: 1 top
outpin class: 1 right
outpin class: 1 bottom
outpin class: 1 left
inpin class: 2 global top   # Clock, global -> routed on a
   special resource.

# Class 0 -> logic cluster inputs, Class 1 -> Outputs, Class
   2 -> clock.

subblocks_per_clb 4
subblock_lut_size 4

#parameters needed only for detailed routing.
switch_block_type subset
Fc_type fractional
Fc_input 0.5
Fc_output 0.25
Fc_pad 1

# orig 1lut clb version, 4lut blocks multiply block size
```

```
   dependent factors by 2
#segment frequency: 1 length: 1 wire_switch: 0 opin_switch: 0
   Frac_cb: 1. \
#         Frac_sb: 1. Rmetal: 195 Cmetal: 0.7e-15
# R and C are per unit length so only double to compensate
   for clb size
# note only 1 seg type cause I'm staying simple and want
   fully buffered
segment frequency: 1 length: 4 wire_switch: 0 opin_switch: 0
   Frac_cb: 1. \
         Frac_sb: 1. Rmetal: 390 Cmetal: 1.4e-15


# Switch used as a tri-state buffer within the routing, and
   also as the
# output buffer used to drive from a CLB output pin to a
   routing wire.

# same as seg len 1 22nm, switches left untouched
switch 0  buffered: yes  R: 6553 Cin: 0.2e-15  Cout: 0.2e-15 \
       Tdel: 24e-12
#switch 0  buffered: yes  R: 7207 Cin: 0.2e-15  Cout: 0.2e-15
   \

# Used only by the area model.

# R_minW_nmos 1967
# R_minW_pmos 3738
R_minW_nmos 4000
R_minW_pmos 8000

# Timing info below.

C_ipin_cblock 0.2e-15


#ignored
T_ipin_cblock 72e-12

T_ipad 24e-12 # clk_to_Q + 2:1 mux
T_opad 24e-12   # Tsetup
T_sblk_opin_to_sblk_ipin 48e-12  # No local routing
T_clb_ipin_to_sblk_ipin 24e-12   # No local routing
T_sblk_opin_to_clb_opin 0.
```

```
# Delays through the BLE (LUT and a FF)
T_subblock T_comb: 24e-12 T_seq_in: 36e-12 T_seq_out: 24e-12
T_subblock T_comb: 24e-12 T_seq_in: 36e-12 T_seq_out: 24e-12
T_subblock T_comb: 24e-12 T_seq_in: 36e-12 T_seq_out: 24e-12
T_subblock T_comb: 24e-12 T_seq_in: 36e-12 T_seq_out: 24e-12
# device model parameters
Vdd 0.600                    # supply voltage (V)
Vth 0.202                    # threshold voltage (V)
Transistor_Width_Factor 4    # ratio of actual PMOS/NMOS width
   to minimum width
Transistor_Length 11         # effective channel length (nm)
```

B.3 One_alt_guaranteed_seg_len_4.xml

```
<!--
# VPR 5 Architecture File
#
# authors: Nikil Mehta <nikil@caltech.edu>
#          Hans Giesen <giesen@seas.upenn.edu> (ITR changes)
#
# Generated using the following command line:
# ./arch.py \-\-extra\-input\-pins 16 \-\-extra\-output\-pins
   4 \-\-lseg 4 \-\-fcin 0.15 \-\-fcout 0.2
   \-\-fcin\-extra\-pins 0.25 \-\-fcout\-extra\-pins 0.1
   \-\-power lp One_alt_guaranteed_seg_len_4_smaller.xml
-->

<architecture>

    <layout auto="1.0"/>

    <device>

        <!--
            my technology parameters.
        -->
        <tech tech="22" vdd="0.8" vth_sigma="0.0364"
           p_to_n_ratio="1"/>
        <nmos cdrain="3.036e-17" cgate="2.024e-17"
           csource="2.2e-17" vth0="0.385"/>
        <pmos cdrain="3.058e-17" cgate="1.782e-17"
           csource="2.662e-17" vth0="-0.412"/>
        <wire resistivity="6.01" cap_per_length="1.5"
           half_pitch="33" aspect_ratio="2"/>

        <!--
            sbox type.

            type:   subset/wilton/universal (always wilton
               for unidir switches)
            fs:     number of segments that a switch outputs
               to (always 3 for unidir switches)
        -->
        <switch_block type="wilton" fs="3"/>

        <!--
            cbox isolation buffer.
```

```
    vpr assumes there is an isolation buffer between
       track and cbox
    input. the buffer is shared if a track connects
       to cboxes above
    and below.  e.g., one buffer per *track* (and per
       cbox location).
    NOTE: I ignore this, and actually figure out the
       cin and tdel of
    the cbox myself.  Set this to 0.

    capacitance:    c_in for the buffer (or c_in of
       cbox if there is no buffer)
    timing:         delay of track -> clb input (so,
       isolation buffer + cbox switch delay)
-->
<timing C_ipin_cblock="0" T_ipin_cblock="0"/>

<!--
    channel width distributions.

    width:  width of channels between pads and core,
       relative to widest core channel
    x:      relative width of x channel relative to y
    y:      relative width of x channel relative to x
-->
<chan_width_distr>
    <io width="1.000000"/>
    <x distr="uniform" peak="1.000000"/>
    <y distr="uniform" peak="1.000000"/>
</chan_width_distr>

<!--
    area estimation.
    NOTE: I ignore this, since I calculate my own
       areas.

    R_minW:                 resistance of minimum
       width nmos/pmos transistors
    ipin_mux_trans_size:    size of each transistor
       in ipin mux
    grid_logic_tile_area:   for estimating functional
       block area
-->
<sizing R_minW_nmos="1" R_minW_pmos="1"
```

```
        ipin_mux_trans_size="0.000000"/>
    <area grid_logic_tile_area="0.000000"/>

</device>

<switchlist>

    <!--
        switch details.

        important:
        name:            unique identifier for switch
           (used by segment definitions)
        type:            buffered (bidir) or mux (unidir)
        R:               equivalent resistance of switch
        Cin:             input capacitance of switch
        Cout:            output capactiance of switch
        Tdel:            intrinsic switch delay (includes
           both mux + buffer for unidir case)

        only for area estimation:
        buf_size:        size of buffer in minimum sized
           transistor units (UNIDIR ONLY)
        mux_trans_size: size of each transistor in mux in
           minimum size transistor units (UNIDIR ONLY)

        my stuff:
        <type>_input_inverter_sizes:    list of '_'
           separated sizes for input inverter chain of
           switch
        <type>_output_inverter_sizes:   list of '_'
           separated sizes for output inverter chain of
           switch

        NOTE: <type> can be cin, cout and sbox for
           selective sizing.

        NOTE: I ignore the nominal values of R, Cin,
           Cout, Tdel and the
        areas since I calculate all of that using my
           models.
    -->
    <switch name="nominal" type="mux" R="0" Cin="0"
       Cout="0" Tdel="0" buf_size="1" mux_trans_size="1"
       cin_input_inverter_sizes="1"
```

```
        cin_output_inverter_sizes="1"
        cout_input_inverter_sizes="1"
        cout_output_inverter_sizes="1"
        sbox_input_inverter_sizes="1"
        sbox_output_inverter_sizes="1"/>

</switchlist>

<segmentlist>

    <!--
        segment details.

        length:          number of logic blocks spanned by
           this segment (longline = entire fpga)
        type:            bidir or unidir
        freq:            fraction of routing tracks
           composed of this segment
        rmetal:          resistance per unit length (in
           terms of *logic blocks*, e.g. 10 ohms/block)
        cmetal:          capacitance per unit length (in
           terms of *logic blocks*, e.g. 10 F/block)

        NOTE: I ignore rmetal and cmetal since I
           calculate them myself.
    -->
    <segment length="4" type="unidir" freq="1.000000"
       Rmetal="498.759" Cmetal="2.71123e-15">

        <!--
            segment switch type.

            mux_name:        name of switch type used to
               drive this segment (UNIDIR ONLY)
            wire_switch:     name of switch type used to
               drive this segment from segment (BIDIR
               ONLY)
            opin_switch:     name of switch type used to
               drive this segment from clb/pad output
               pins (BIDIR ONLY)
        -->
        <mux name="nominal"/>

        <!--
            segment population.
```

```
            depopulation pattern for segments.  binary
               list that specifies connections.
            note that sb string has (length + 1) digits;
               cb string just has length
            take segment length 6 for example:

            sb type: '1 0 1 0 1 0 1'    segment corner
               turns at every other intersection
            cb type: '0 0 0 1 1 1'      connect to last 3
               cboxes only
        -->
        <sb type="pattern">1 1 1 1 1</sb>
        <cb type="pattern">1 1 1 1</cb>

    </segment>

</segmentlist>

<typelist>

    <type name=".clb">

        <!--
            cbox connectivity.

            fc_in:      number of tracks in each
               bordering channel to which each logic
               input pin connects
            fc_out:     number of tracks in each
               bordering channel to which each logic
               output pin connects
        -->
        <fc_in type="frac">0.15</fc_in>
        <fc_out type="frac">0.2</fc_out>
        <fc_in_extra_pins
           type="frac">0.25</fc_in_extra_pins>
        <fc_out_extra_pins
           type="frac">0.1</fc_out_extra_pins>

        <!--
            clb/lut inputs.

            max_subblocks:          luts per clb (N)
            max_subblock_inputs:    inputs per lut (K)
```

```
    T_comb:                  delay from lut input
       to lut output when this lut is only
       combinational
    T_seq_in:                delay from lut input
       to flop input for seqeuential mode (lut
       delay + t_setup)
    T_seq_out:               delay from flop input
       to flop output for seqeuential mode
       (clk_to_q)

    NOTE: I ignore this, since I calculate my own
       delays.
-->
<subblocks max_subblocks="8"
   max_subblock_inputs="6">
    <timing>
        <T_comb>
            <trow>0</trow>
            <trow>0</trow>
            <trow>0</trow>
            <trow>0</trow>
            <trow>0</trow>
            <trow>0</trow>
        </T_comb>
        <T_seq_in>
            <trow>0</trow>
        </T_seq_in>
        <T_seq_out>
            <trow>0</trow>
        </T_seq_out>
    </timing>
</subblocks>

<!--
    lut timing.

    T_sblk_opin_to_fb_opin:     delay from lut
       output to clb output
    T_sblk_opin_to_sblk_ipin:   delay from lut
       output to lut input
    T_fb_ipin_to_sblk_ipin:     delay from clb
       input to lut input

    NOTE: I ignore this, since I calculate my own
```

```
        delays.
-->
<timing>
    <tedge type="T_sblk_opin_to_fb_opin">0</tedge>
    <tedge
        type="T_sblk_opin_to_sblk_ipin">0</tedge>
    <tedge type="T_fb_ipin_to_sblk_ipin">0</tedge>
</timing>

<!--
    clb pin numbers.

    note: order must match the .net file!
-->
<pinclasses>
    <class type="in">0 1 2 3 4 5 6 7 8 9 10 11 12
        13 14 15 16 17 18 19 20 21 22 23 24 25 26
        -36 -37 -38 -39 -40 -41 -42 -43 -44 -45
        -46 -47 -48 -49 -50 -51</class>
    <class type="out">27 28 29 30 31 32 33 34 -52
        -53 -54 -55</class>
    <class type="global">35</class>
</pinclasses>

<!--
    clb pin locations.
-->
<pinlocations>
    <loc side="bottom"> 0 4 8 12 16 20 24 28 32
        36 40 44 48 52</loc>
    <loc side="left"> 1 5 9 13 17 21 25 29 33 37
        41 45 49 53</loc>
    <loc side="top"> 2 6 10 14 18 22 26 30 34 38
        42 46 50 54</loc>
    <loc side="right"> 3 7 11 15 19 23 27 31 35
        39 43 47 51 55</loc>
</pinlocations>

<!--
    clb locations.

    columns of the fpga that use this block
        (type=fill, priority=1 means only use this
        block)
-->
```

```
            <gridlocations>
                <loc type="fill" priority="1"/>
            </gridlocations>

        </type>

        <!--
            io pad timing.

            capacity:   number of actual IOs per IO pad
            t_inpad:    delay through input pad (clk_to_q +
               2:1 mux)
            t_outpad:   delay through output pad (t_setup)
            fc_in:      number of tracks in each bordering
               channel to which each input pad connects
            fc_iout:    number of tracks in each bordering
               channel to which each output pad connects
        -->
        <io capacity="8" t_inpad="0" t_outpad="0">
            <fc_in type="frac">1.000000</fc_in>
            <fc_out type="frac">1.000000</fc_out>
        </io>

    </typelist>

</architecture>
```

Acronyms

C-Box Connection box, the set of switches that connect a CLB to the surrounding network. 59, 60, 62–68, 70, 73, 75–78, 80–82, 89, 90, 92, 94

CAD Computer-Aided Design. 3, 5, 10, 11, 13, 14, 41, 84, 132, 142, 143, 147, 150, 153, 155, 158, 169

CLB Clustered Logic Block. 46, 56, 59, 63, 79, 82, 88–90, 92, 93, 96, 98, 169

CSM Component-Specific Mapping. 2–7, 9, 10, 15, 16, 19, 30, 36, 117, 142, 155, 160, 161, 163, 164, 170, 171, 173, 175, 176, 180

CYA Choose-Your-own-Adventure. vi, vii, 2, 3, 10–12, 14–17, 19, 20, 34, 36–39, 41, 42, 54–56, 61, 63, 69–72, 74, 75, 77–82, 85–87, 90, 93–95, 102–107, 110, 114, 117–126, 134, 136, 138–147, 149, 151–153, 155–158, 160–165, 167, 169–171, 174–176, 180, 182–185

DFS Dynamic Frequency Scaling. 4, 5, 117, 123

DFVS Dynamic Frequency and Voltage Scaling. vi, 3, 5, 16, 19, 134, 136, 137, 139, 140, 144, 155, 160–167, 185

DVS Dynamic Voltage Scaling. 4, 5

FET Field-Effect Transistor. 2, 22

FPGA Field-Programmable Gate Array. vi, 1, 2, 5–9, 11–13, 16, 19, 29, 30, 32–35, 37–39, 45, 60, 87–89, 92, 114, 115, 155, 169, 170, 175, 178, 179

ICAP Internal Configuration Access Port. 47

ITRS International Technology Roadmap for Semiconductors. 21, 27, 28

LE Logic Element. 11, 12, 56, 109

LUT Look-Up Table. 11, 14, 30, 37, 43, 46, 56, 59, 63, 82, 92, 93, 108, 169, 175, 178, 179

MOSFET Metal-Oxide-Semiconductor FET. 23, 24

NUT Net Under Test. 110

RDF Random Dopant Fluctuation. 25, 27, 28, 30, 32

S-Box Switch box, the set of switches at the intersection between two channels. 58, 59, 78, 81, 82, 89, 92, 94, 97, 117, 171, 178

SEU Single-Event Upset. 30, 34

SSTA Statistical Static Timing Analysis. 5, 31

TMR Triple Modular Redundancy. 30, 31, 34, 35

Bibliography

[1] M. Abramovici and P.R. Menon. A practical approach to fault simulation and test generation for bridging faults. *IEEE Transactions on Computers*, C-34(7):658–663, July 1985.

[2] Actel. *Design Techniques for Radiation-Hardened FPGAs.* Actel, Inc., 955 East Arques Avenue, Sunnyvale, CA 94086, 1997. Dual and TMR Application Note AC128 `https://www.actel.com/documents/Des_Tech_RH_AN.pdf`.

[3] Rick Amerson, Richard Carter, W. Bruce Culbertson, Phil Kuekes, and Greg Snider. Plasma: An FPGA for million gate systems. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 10–16, February 1996.

[4] K. Bernstein, D.J. Frank, A.E. Gattiker, W. Haensch, B.L. Ji, S.R. Nassif, E.J. Nowak, D.J. Pearson, and N.J. Rohrer. High-performance CMOS variability in the 65-nm regime and beyond. *IBM Journal of Research and Development*, 50(4/5):433–449, July/September 2006.

[5] Vaughn Betz. VPR and T-VPack: Versatile Packing, Placement and Routing for FPGAs. `http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html`, March 27 1999. Version 4.30.

[6] Vaughn Betz and Jonathan Rose. FPGA Place-and-Route Challenge. `http://www.eecg.toronto.edu/~vaughn/challenge/challenge.html`, 1999.

[7] Shekhar Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, November–December 2005.

[8] Nicola Campregher, Peter Y.K. Cheung, George A. Constantinides, and Milan Vasilko. Yield modelling and yield enhancement for FPGAs using fault tolerance schemes. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*, 2005.

[9] Nicola Campregher, Peter Y.K. Cheung, George A. Constantinides, and Milan Vasilko. Reconfiguration and fine-grained redundancy for fault tolerance in FPGAs. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*, 2006.

[10] Carl Carmichael, Michael Caffrey, and Anthony Salazar. *Correcting Single-Event Upsets Through Virtex Partial Configuration*. Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124, 2000. XAPP 216, `http://www.xilinx.com/bvdocs/appnotes/xapp216.pdf`.

[11] C.T. Chow, L.S.M. Tsui, Philip H.W. Leong, Wayne Luk, and Steve J.E. Wilton. Dynamic voltage scaling for commercial FPGAs. In *Proceedings of the International Conference on Field-Programmable Technology*, pages 173–180, 2005.

[12] Richard G. Cliff, Rina Raman, and Srinivas T. Reddy. Programmable logic devices with spare circuits for replacement of defects. United States Patent Number: 5,434,514, July 18 1995.

[13] Richard G. Cliff, Rina Raman, and Srinivas T. Reddy. Implementation of redundancy for a programmable logic device. United States Patent Number: 5,498,975, March 12 1996.

[14] W. Bruce Culbertson, Rick Amerson, Richard Carter, Phil Kuekes, and Greg Snider. The Teramac custom computer: Extending the limits with defect tolerance. *Defect and Fault-Tolerance in VLSI Systems, IEEE International Symposium on*, 0:2, 1996.

[15] W. Bruce Culbertson, Rick Amerson, Richard Carter, Phil Kuekes, and Greg Snider. Defect tolerance on the TERAMAC custom computer. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pages 116–123, April 1997.

[16] André DeHon and Nikil Mehta. Exploiting partially defective LUTs: Why you don't need perfect fabrication. In *Proceedings of the International Conference on Field-Programmable Technology*, pages 12–19, December 2013.

[17] Robert H. Dennard, Fritz H. Gaensslen, Hwa-Nien Yu, V. Leo Rideout, Ernest Bassous, and Andre R. LeBlanc. Design of ion-implanted MOSFETs with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, October 1974.

[18] Fethi Dhaoui, Zhigang Wang, John McCollum, Richard Chan, and Vidyadhara Bellippady. Radiation-tolerant flash-based FPGA memory cells, May 2008.

[19] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.

[20] Hadi Esmaeilzadeh, Emily Blem, Rene St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings*

of the International Symposium on Computer Architecture, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

[21] B.F. Fitzgerald and E.P. Thoma. Circuit implementation of fusible redundant addresses on rams for productivity enhancement. *IBM Journal of Research and Development*, 24(3):291–298, May 1980.

[22] Hans Giesen, Benjamin Gojman, Raphael Rubin, and André DeHon. Continuous Online Self-Monitoring Introspection Circuitry for Timing Repair by Incremental Partial-reconfiguration (COSMIC TRIP). In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 111–118, 2016.

[23] Hans Giesen, Benjamin Gojman, Raphael Rubin, Ji Kim, and André DeHon. Continuous Online Self-Monitoring Introspection Circuitry for Timing Repair by Incremental Partial-reconfiguration (COSMIC TRIP). *ACM Transactions on Reconfigurable Technology and Systems*, 11(1):3:1–3:23, January 2018.

[24] Hans Giesen, Raphael Rubin, Benjamin Gojman, and André DeHon. Quality-time tradeoffs in component-specific mapping: How to train your Dynamically Reconfigurable Array of Gates with Outrageous Network-delays. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 85–94, February 2017.

[25] Hans Giesen, Raphael Rubin, Benjamin Gojman, and André DeHon. Self-adaptive timing repair. *IEEE Design & Test*, 34(6):54–62, December 2017.

[26] Benjamin Gojman. *GROK-FPGA: Generating Real On-chip Knowledge for FPGA Fine-Grain Delays using Timing Extraction.* PhD thesis, University of Pennsylvania, 2014.

[27] Benjamin Gojman, Nikil Mehta, Raphael Rubin, and André DeHon. Component-specific mapping for low-power operation in the presence of variation and aging. In *Low-Power Variation-Tolerant Design in Nanometer Silicon*, chapter 12, pages 381–432. Springer, 2011.

[28] Steve Guccione, Delon Levi, and Prasanna Sundararajan. JBits: Java based interface for reconfigurable computing. In *Proceedings of the International Conference on Military and Aerospace Programmable Logic Devices*, 1999.

[29] Scott Hanson, Bo Zhai, Kerry Bernstein, David Blaauw, Andres Bryant, Leland Chang, Koushik K. Das, Wilfried Haensch, Edward J. Nowak, and Dinnis M. Sylvester. Ultralow-voltage, minimum-energy CMOS. *IBM Journal of Research and Development*, 50(4–5):469–490, July/September 2006.

[30] I.G. Harris, P.R. Menon, and R. Tessier. BIST-based delay path testing in FPGA architectures. In *Proceedings of International Test Conference*, pages 932–938, November 2001.

[31] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[32] Chen He, Margarida F. Jacome, and Gustavo de Veciana. A reconfiguration-based defect-tolerant design paradigm for nanotechnologies. *IEEE Design and Test of Computers*, 22(4):316–326, July-August 2005.

[33] B. Hoeneisen and C.A. Mead. Fundamental limitations in microelectronics–i. mos technology. *Solid-State Electronics*, 15(7):819–829, 1972.

[34] Zohair Hyder and John Wawrzynek. Defect tolerance in multiple-FPGA systems. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*, pages 247–254, August 2005.

[35] Synopsys, Inc. HSPICE. `http://www.synopsys.com/Tools/Verification/AMSVerification/CircuitSimulation/HSPICE`, 2010.

[36] 2001 International Technology Roadmap for Semiconductors. `https://www.dropbox.com/sh/vxigcu48nfe4t81/AACuMvZEh1peQ6G8miYFCSEJa?dl=0`, 2001.

[37] 2005 International Technology Roadmap for Semiconductors. `https://www.dropbox.com/sh/2urwqghq1gzk511/AADuZE5F68lz2DYGpA3TspSna?dl=0`, 2005.

[38] 2007 International Technology Roadmap for Semiconductors. `https://www.dropbox.com/sh/floxh3swiynur47/AAAwTAwf1RUzyNu8qv-PMfiUa?dl=0`, 2007.

[39] 2009 International Technology Roadmap for Semiconductors. `https://www.dropbox.com/sh/ia1jkem3v708hx1/AAB1fo1HrYIKClJNk0dB7YrCa?dl=0`, 2009.

[40] 2011 International Technology Roadmap for Semiconductors. `http://www.itrs2.net/2011-itrs.html`, 2011.

[41] Kazuya Katsuki, Manabu Kotani, Kazutoshi Kobayashi, and Hidetoshi Onodera. A yield and speed enhancement scheme under within-die variations on 90nm LUT array. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 601–604, 2005.

[42] Gokal Krishnan. Flexibility with EasyPath FPGAs. *Xcell Journal*, 0(4):96–98, 2005.

[43] K.J. Kuhn. CMOS transistor scaling past 32nm and implications on variation. In *2010 IEEE/SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, pages 241–246, July 2010.

[44] Akhilesh Kumar and Mohab Anis. FPGA Design for Timing Yield Under Process Variations. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(3):423–435, March 2010.

[45] John Lach, William H. Mangione-Smith, and Miodrag Potkonjak. Efficiently Supporting Fault-Tolerance in FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 105–115, February 1998.

[46] Vijay Lakamraju and Russell Tessier. Tolerating operational faults in cluster-based FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 187–194, New York, NY, USA, 2000. ACM.

[47] Christopher Lane, Ketan Zaveri, Hyun Yi, Giles Powell, Paul Leventis, David Jefferson, David Lewis, Triet Nyguen, Vikram Santurkar, Michael Chan, Andy Lee, Brian Johnson, and David Cashman. Programmable logic device with redundant circuitry. United States Patent Number: 6,965,249, November 15 2005.

[48] G.V. Leming and K. Nepal. Low-power FPGA routing switches using adaptive body biasing technique. In *Circuits and Systems, 2009. MWSCAS '09. 52nd IEEE International Midwest Symposium on*, pages 447–450, August 2009.

[49] Yan Lin, Lei He, and Mike Hutton. Stochastic physical synthesis considering prerouting interconnect uncertainty and process variation for FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(2):124, 2008.

[50] Zhi-Ming Ling, Jae Cho, Robert W. Wells, Clay S. Johnson, and Shelly G. Davis. Method of using partially defective programmable logic devices. United States Patent Number: 6,664,808, December 16 2003.

[51] Timothy A. Linscott, Benjamin Gojman, Raphael Rubin, and André DeHon. Pitfalls and tradeoffs in simultaneous, on-chip FPGA delay measurement. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 100–104, February 2016.

[52] Gregory Lucas, Chen Dong, and Deming Chen. Variation-aware placement for FPGAs with multi-cycle statistical timing analysis. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 177–180, New York, New York, USA, 2010. ACM.

[53] Jason Luu, Ian Kuon, Peter Jamieson, Ted Campbell, Andy Ye, Wei Mark Fang, and Jonathan Rose. VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '09, pages 133–142, New York, NY, USA, 2009. ACM.

[54] Alexander Marquardt, Vaughn Betz, and Jonathan Rose. Timing-driven placement for FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 203–213, 2000.

[55] M. Imran Masud and Steve Wilton. A new switch block for segmented FPGAs. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*, pages 274–281, 1999.

[56] Yohei Matsumoto, Masakazu Hioki, Takashi Kawanami, Hanpei Koike, Toshiyuki Tsutsumi, Tadashi Nakagawa, and Toshihiro Sekigawa. Suppression of intrinsic delay variation in FPGAs using multiple configurations. *ACM Transactions on Reconfigurable Technology and Systems*, 1(1), March 2008.

[57] Cameron McClintock, Andy L. Lee, and Richard G. Cliff. Redundancy circuitry for logic circuits. United States Patent Number: 6,034,536, March 7 2000.

[58] Larry McMurchie and Carl Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 111–117, 1995.

[59] Nikil Mehta. *An Ultra-Low Energy, Variation Tolerant FPGA Architecture Using Component-Specific Mapping*. PhD thesis, California Institute of Technology, 2013.

[60] Nikil Mehta and André DeHon. Variation and aging tolerance in FPGA. In *Low-Power Variation-Tolerant Design in Nanometer Silicon*, chapter 11, pages 365–380. Springer, 2011.

[61] Nikil Mehta, Raphael Rubin, and André DeHon. Limit Study of Energy & Delay Benefits of Component-Specific Routing. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 97–106, 2012.

[62] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, page 4, 1965.

[63] Geoges Nabaa, Navid Azizi, and Farid N. Najm. An adaptive FPGA architecture with process variation compensation and reduced leakage. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 624–629, 2006.

[64] Laurence W. Nagel and D.O. Pederson. SPICE (Simulation Program with Integrated Circuit Emphasis). Technical Report UCB/ERL M382, EECS Department, University of California, Berkeley, April 1973.

[65] John Von Neumann. Probabilistic logic and the synthesis of reliable organisms from unreliable components. In Claude Shannon and John McCarthy, editors, *Automata Studies*. Princeton University Press, 1956.

[66] Edward Packard. *The Cave of Time*. Bantam Books, 1979.

[67] Brian Pratt, Michael Caffrey, Paul Graham, Keith Morgan, and Michael Wirthlin. Improving FPGA design robustness with partial TMR. In *Proceedings of the IEEE International Reliability Physics Symposium*, pages 226–232, 2006.

[68] Jonathan Rose et al. VPR and T-VPack: Versatile Packing, Placement and Routing for FPGAs. `http://www.eecg.utoronto.ca/vpr/`, 2009.

[69] Raphael Rubin and André DeHon. Choose-Your-Own-Adventure Routing: Lightweight Load-Time Defect Avoidance. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 23–32, 2009.

[70] Raphael Rubin and André DeHon. Choose-Your-Own-Adventure Routing: Lightweight Load-Time Defect Avoidance. *ACM Transactions on Reconfigurable Technology and Systems*, 4(4), December 2011.

[71] Raphael Rubin and André DeHon. Timing-Driven Pathfinder Pathology and Remediation: Quantifying and Reducing Delay Noise in VPR-Pathfinder. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 173–176, 2011.

[72] Raphael Rubin and André DeHon. Timing-Driven Pathfinder Pathology and Remediation: Quantifying and Reducing Delay Noise in VPR-Pathfinder, supplemental materials. `http://ic.ese.upenn.edu/publications/pf_fpga_2011/`, 2011.

[73] Jayashree Saxena, Kenneth M. Butler, John Gatt, R. Raghuraman, Sudheendra Phani Kumar, Supatra Basu, David J. Campbell, and John Berech. Scan-based transition fault testing — Implementation and low cost test challenges. *Proceedings of International Test Conference*, pages 1120–1129, 2002.

[74] P. Sedcole, J.S. Wong, and P.Y.K. Cheung. Modelling and compensating for clock skew variability in FPGAs. In *ICECE Technology, 2008. FPT 2008. International Conference on*, pages 217–224, December 2008.

[75] Pete Sedcole and Peter Y.K. Cheung. Parametric yield modeling and simulations of FPGA circuits considering within-die delay variations. *ACM Transactions on Reconfigurable Technology and Systems*, 1(2), June 2008.

[76] Deshanand P. Singh and Stephen D. Brown. Constrained clock shifting for field programmable gate arrays. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 121–126. ACM, 2002.

[77] Satish Sivaswamy and Kia Bazargan. Statistical analysis and process variation-aware routing and skew assignment for FPGAs. *ACM Transactions on Reconfigurable Technology and Systems*, 1(1):1–35, 2008.

[78] P.A. Stolk, F.P. Widdershoven, and D.B.M. Klaassen. Modeling statistical dopant fluctuations in MOS transistors. *Electron Devices, IEEE Transactions on*, 45(9):1960–1971, September 1998.

[79] Jordan S. Swarz, Vaughn Betz, and Jonathan Rose. A Fast Routability-Driven Router for FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 140–149. ACM/SIGDA, February 1998.

[80] R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. Mitigating parameter variation with dynamic fine-grain body biasing. In *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, pages 27–42, December 2007.

[81] Russell Tessier. Negotiated A* Routing for FPGAs. In *Proceedings of the 5th Canadian Workshop on Field Programmable Devices*, June 1998.

[82] Stephen M. Trimberger. Method and apparatus for multiple context and high reliability operation of programmable logic devices, May 2007.

[83] Stephen M. Trimberger. Structures and methods of overcoming localized defects in programmable integrated circuits by routing during the programming thereof. United States Patent Number: 7,251,804, July 31 2007.

[84] Stephen M. Trimberger. Utilizing multiple test bitstreams to avoid localized defects in partially defective programmable integrated circuits. United States Patent Number: 7,424,655, September 9 2008.

[85] C. Visweswariah, K. Ravindran, K. Kalafala, S.G. Walker, S. Narayan, D.K. Beece, J. Piaget, N. Venkateswaran, and J.G. Hemmett. First-order incremental block-based statistical timing analysis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(10):2170–2180, October 2006.

[86] Robert W. Wells, Zhi-Ming Ling, Robert D. Patrie, Vincent L. Tong, Jae Cho, and Shahin Toutounchi. Application-specific testing methods for programmable logic devices. United States Patent Number: 6,817,006, November 9 2004.

[87] Neil H.E. Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley, third edition, 2005.

[88] Justin S. Wong, Pete Sedcole, and Peter Y.K. Cheung. Self-measurement of combinatorial circuit delays in FPGAs. *ACM Transactions on Reconfigurable Technology and Systems*, 2(2):1–22, June 2009.

[89] Yu-Liang Wu, Shuji Tsukiyama, and Malgorzata Marek-Sadowska. Graph based analysis of 2-D FPGA routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(1):33–44, January 1996.

[90] Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124. *XC6200 FPGA Advanced Product Specification*, version 1.0 edition, June 1996.

[91] Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124. *Xilinx Virtex-4 Family Overview*, June 2005. DS112, `http://direct.xilinx.com/bvdocs/publications/ds112.pdf`.

[92] Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124. *Virtex FPGA Series Configuration and Readback*, March 2005. XAPP 138.

[93] Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124. *Virtex-5 FPGA Configuration User Guide*, September 2008. UG191, `http://www.xilinx.com/bvdocs/userguides/ug191.pdf`.

[94] H. Youssef and E. Shragowitz. Timing constraints for correct performance. In *Proceedings of the International Conference on Computer-Aided Design*, pages 24–27, Nov 1990.

[95] Anthony J. Yu and Guy G. Lemieux. Defect-tolerant FPGA switch block and connection block with fine-grain redundancy for yield enhancement. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*, pages 255–262, 2005.

[96] Anthony J. Yu and Guy G. Lemieux. FPGA defect tolerance: Impact of granularity. In *Proceedings of the International Conference on Field-Programmable Technology*, pages 189–196, 2005.

[97] Sugihara Yuuri, Kume Youhei, Kobayashi Kazutoshi, and Onodera Hidetoshi. Track swapping on critical paths utilizing random variations for FPGAs to enhance speed and yield. *IEICE Technical Report*, 107(340):13–18, 2007.