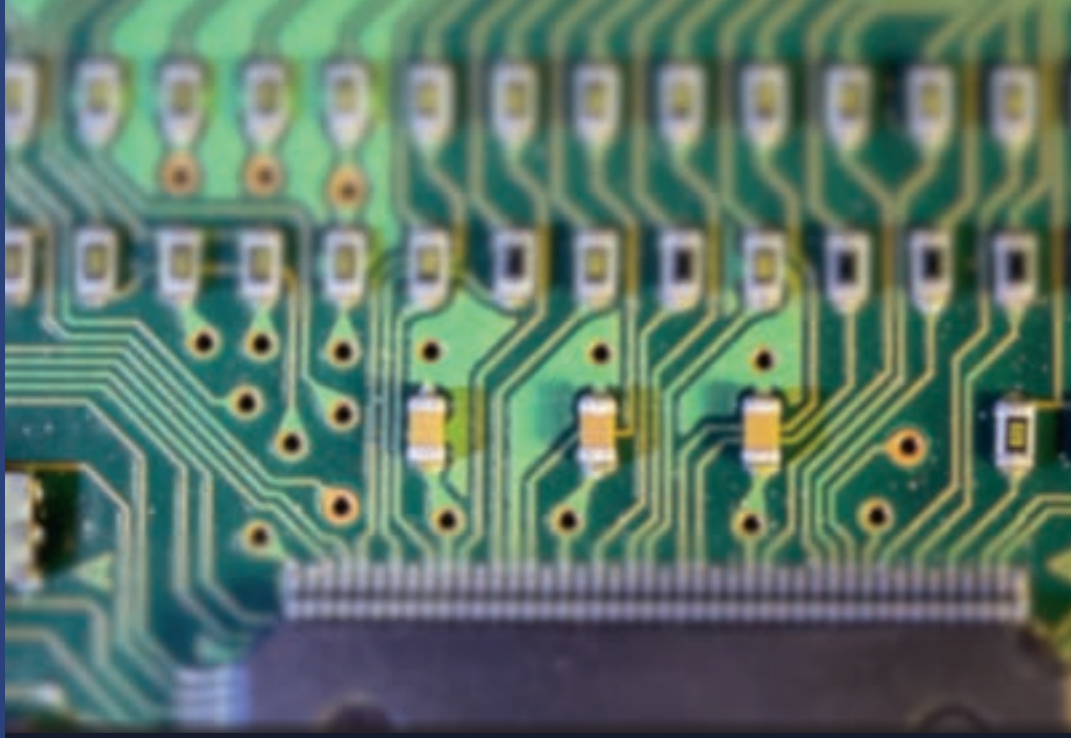


Johannes Pfau

RFET Reconfigurable Devices:

Power Aware FPGA Architectures and Toolflow

RFET Reconfigurable Devices: Power Aware FPGA Architectures and Toolflow

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von der KIT-Fakultät für Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie (KIT)

angenommene

Dissertation

von

M.Sc. Johannes Pfau

Tag der mündlichen Prüfung:	25.06.2024
Hauptreferent:	Prof. Dr.-Ing. Dr. h. c. Jürgen Becker
Korreferent:	Prof. Dr.-Ing. Klaus Hofmann

RFET Reconfigurable Devices:
Power Aware FPGA Architectures and Toolflow

First edition: September 2024
DOI: 10.5445/IR/1000174452

The latest digital edition of this work is available at
dx.doi.org/10.5445/IR/1000174452 .

This page intentionally left blank

This page intentionally left blank

Abstract

In recent years, power consumption in modern Integrated Circuits (ICs) has become increasingly important. One special kind of IC, Field Programmable Gate Arrays (FPGAs), is especially hampered by high power consumption. The main reason for this being that in FPGA, the real application is not known during chip manufacturing time and will only be programmed later, in the field. Because of this, FPGAs will have unused resources, as applications rarely use all of them. Such unused resources do not use dynamic power, but they will use static power due to leakage currents. Classical mitigation approaches, such as power gating, can not easily be used in FPGA, as the non-utilized resources are often spread over the whole chip area. In addition to this, Process-, Voltage-, Temperature-Variation and Aging (PVTA) effects also affect FPGAs worse: Unlike in Application Specific Integrated Circuits (ASICs), logic placement and density are not known during manufacturing time. Voltage drop and temperature analysis can therefore not be performed before chip manufacturing. FPGA architectures must therefore assume uniform applications and worst-case PVTA effects instead. With some transistor technologies, it is however possible to trade-off leakage power with logic performance. In this case, safety margins in performance prevent further reduction in power usage.

As a solution to these problems, this thesis proposes the Power Aware Reconfigurable FPGA Architecture (PARFAIT). This FPGA is divided into power regions, that can be controlled individually by power controllers. The Electronic Design Automation (EDA) toolflow for user application synthesis is modified to determine the required performance in each region. Additionally, instead of simply assuming worst-case PVTA, a measurement system obtains real propagation delays at runtime. Combining those two approaches allows to reduce the power in each region at runtime, until the performance matches the determined requirements. Such a system also implicitly compensates dynamic changes in PVTA values.

To realize this architecture, a transistor technology that enables a trade-off between performance and leakage current is needed. For comparison, this thesis evaluates all results in Silicon on Insulator (SOI) technology with body

biasing. The main technology investigated however is Reconfigurable (Ambipolar) FET (RFET) technology with program gate voltage scaling. To fully utilize this technology, various changes to standard FPGA toolflow and architecture are proposed: A standard cell library and evaluation for RFET enables realization of non-reconfigurable logic. For reconfigurable logic, RFET based Universal Logic Modules (ULMs) are investigated as a less area intensive alternative for Lookup Tables (LUTs). Based on this, the architecture with power regions and region controller is introduced. The architecture further introduces logic invasion, a novel method to repurpose parts of the reconfigurable logic for performance measurement at runtime.

Performance requirements are calculated for each power region during application synthesis with the adjusted EDA toolfow. These values are then used in a final hardware / software co-simulation, demonstrating the functionality and determining achievable power savings. For the simulation, technology models determining propagation delay from PVTA parameters will be derived. Additionally, various scenarios to describe the PVTA parameters in the evaluation are developed. Final results show that a static power reduction down to 49.1 % of original power consumption is possible with RFET. Due to the different behavior of SOI technology, this technology even allows power to be reduced down to 2.76 %. This large difference seems to suggest that leakage power sensitivity for the control parameter needs to be improved in RFET technology. As will be explained in the thesis though, the absolute leakage currents of RFETs are already significantly smaller, so that further optimization may actually not be required. RFET technology propagation delay is also more sensitive to changes in the control parameter, which enables a wider range of compensation.

Kurzfassung

Der Stromverbrauch in Informationsverarbeitungssystemen hat in den letzten Jahren mehr und mehr an Bedeutung gewonnen, weswegen eine Reduktion der Verlustleistung auch in ICs immer relevanter wird. Im Vergleich zu ASICs haben FPGAs hierbei aufgrund ihrer besonderen Eigenschaften oft eine besonders hohe Verlustleistung: Da die Endanwendung bei der Chipfertigung noch nicht bekannt ist, müssen FPGAs generisch für verschiedene Anwendungen ausgelegt werden. Konkrete Endanwendungen benötigen jedoch selten alle FPGA-Ressourcen, was zu ungenutzten Ressourcen und erhöhtem Energiebedarf führt. Die Verlustleistung in diesen Ressourcen wird primär durch statische Leckströme erzeugt, dynamische Verlustleistung ist in ungenutzten Ressourcen irrelevant. Viele Lösungsansätze, die üblicherweise in ASICs verwendet werden, sind in FPGAs nicht realisierbar: So ist beispielsweise Power-Gating schwer umzusetzen, da unbenutzte Ressourcen zur Fertigungszeit unbekannt, und oft über die Chip-Fläche verteilt sind. Aus denselben Gründen sind FPGAs auch von PVTA Effekten besonders betroffen: Da mit der Logikplatzierung auch die Logikdichte zur Fertigungszeit unbekannt ist, können Analysen zu Spannungsabfällen und Temperaturverteilungen nicht vorab durchgeführt werden. FPGAs müssen also unter Annahme von Worst-Case-Szenarien entwickelt werden. Werden zur FPGA-Realisierung Technologien verwendet, die eine Abwägung zwischen Geschwindigkeit und Verlustleistung erlauben, verhindert die Nutzung solcher Worst-Case-Szenarien eine weitere Reduktion der Verlustleistung.

Zur Lösung dieser Probleme wird in dieser Dissertation die Power Aware Reconfigurable FPGA Architecture (PARFAIT) vorgestellt. Diese unterteilt den FPGA in mehrere Regionen mit dazugehörigen Controllern zur Regelung der Verlustleistung und Geschwindigkeit. Zur Bestimmung des maximalen Propagation Delays, das als Maß für die Geschwindigkeitsanforderungen in jeder Region dient, werden Open-Source EDA Programme angepasst. Anstatt ein Worst-Case-Szenario anzunehmen, wird zusätzlich ein System entwickelt, um das tatsächliche Propagation Delay in jeder Region zu erfassen. Durch die Kombination der beiden Ansätze wird ein System realisiert, dass die Verlustleistung dynamisch reduziert und dabei sicherstellt, dass

die Anforderungen an die Schaltungsgeschwindigkeit erfüllt werden. Weiterhin lassen sich mit diesem System dynamische PVTA Effekte kompensieren.

Zur Umsetzung dieser Ansätze werden Schaltungstechnologien verwendet, die eine Abwägung zwischen Verlustleistung und Geschwindigkeit erlauben. Hierfür werden eine kommerzielle SOI Technologie mit Body Biasing (BB) als Referenz, und eine RFET Technologie mit Program-Gate basiertem Threshold-Voltage-Scaling, evaluiert. Weiterhin erfordern die Konzepte auch Anpassungen an EDA Tools und der FPGA Architektur: So wird in dieser Arbeit eine Standardzellenbibliothek für RFETs zur Realisierung von digitaler Logik eingeführt. Zur Umsetzung der rekonfigurierbaren Logik werden RFET-basierte Universal Logic Modules (ULMs) als Alternative für LUTs untersucht. Darauf aufbauend wird die FPGA Architektur mit Power Regionen und Region Controller vorgestellt. Unter anderem wird hier das Konzept der Logic-Invasion eingeführt, das eine Charakterisierung der Schaltungsgeschwindigkeit durch Mitnutzung der bereits vorhandenen rekonfigurierbaren Logikelemente ermöglicht.

Die Anforderungen an die Schaltungsgeschwindigkeit in jeder Region werden durch die angepassten EDA Tools berechnet. Für verschiedene Benchmarks werden diese Anforderungen dann in einer Hardware/Software-Kosimulation verwendet, um die mögliche Reduktion der Verlustleistung zu bestimmen. Dafür werden für die untersuchten Technologien Simulationsmodelle eingeführt, die eine Abschätzung des Propagation Delay in Abhängigkeit von PVTA ermöglichen. Abschließend werden mehrere Szenarien zu Änderungen der PVTA Parameter evaluiert. Hierbei wird gezeigt, dass mit der RFET Technologie eine Reduktion auf bis zu 49 % der ursprünglichen Verlustleistung möglich ist. Für die untersuchte SOI Technologie ergibt sich eine Reduktion auf bis zu 2 %. Diese Ergebnisse zeigen einerseits, dass der Einfluss des Program-Gates auf die Leckströme in der RFET Technologie noch verbessert werden kann. Anderseits sind die absoluten Leckströme in der RFET Technologie bereits um Größenordnungen geringer. Zusätzlich ist die Abhängigkeit der Schaltungsgeschwindigkeit von dem Kontrollparameter in RFET Technologie stärker, was eine bessere PVTA Kompensation ermöglicht.

Acknowledgements

Writing a dissertation and the research involved in it is a complex, long-term task. I couldn't have completed it without various people's help, both technical help and support in general.

First, I'd like to thank my supervisor Jürgen Becker: When I joined the institute, I intended to work in longer established projects. Luckily you convinced me to join the Power Aware Reconfigurable FPGA Architecture (PARFAIT) project. As the project progressed, you always provided invaluable feedback and ideas, e.g. during meetings and interim presentations. Despite this guidance, you still offered me complete freedom to shape the research according to my interests. I'm very grateful for the trust in me and for providing the opportunity to take responsibility early on. I'd also like to thank the members of my examination board: Klaus Hofmann for providing help and feedback as my secondary advisor and as project partner in PARFAIT. I still fondly recall the productive discussions and unique ideas we developed in project meetings. Peter Rost for chairing the examination and for the reassuring words in the preparation meeting and Jasmin Aghassi-Hagmann and Laurent Schmalen for investing their time to be part of the committee.

Invaluable insights were also provided by PARFAIT project partners and the co-authors of my research papers: Maximilian Reuter started his dissertation research at essentially the same time as I did. It was a pleasure to discuss ideas, organize work plans, write joint papers and carry out research and dissertation basically in lockstep. Tillmann Krauss introduced me to the world of RFETs. Thank you for the patience when teaching me RFET working principles and for the dedication to still offer advice years after you left university. Jens Trommer joined the second phase of the PARFAIT project. Being the "new Till" for me, you had to explain RFET circuits such as RGATEs. Your help with the PARFAIT 2 project proposal enabled a large part of the research in this thesis. Giulio Galderisi was the one carrying out device measurements and improvements. For a long time I thought this dissertation would have to be based solely on simulated transistors. You made it possible to actually use real, measured transistor characteristics in the evaluation. In addition, there have been many more people involved to get the PARFAIT project up and

running. There are too many of you to list everyone, but I certainly did not forget your valuable help. This also includes all students whose master or bachelor thesis I supervised. I always loved to gain new perspectives and to discuss various technical details with you.

I also want to express my gratitude to all colleagues and friends at the institute: A special thanks goes to Tobias Dörr, with whom I shared an office for almost seven years. Whenever I had any question about anything, you always provided advice and often a complementary perspective, which I could not get to on my own. More thanks go to Hannes Stoll and Timo Sandmann. For a short time, we set up some server things at the institute and since then, you're my go-to experts for everything network related. Similarly, thanks go to the "dynamic duo" Fabian Lesniak and Tim Hotfilter: We did not only set up servers but also locomotives and coffee machines. Time went by way too fast when we plotted various plans, and I'll for sure continue to annoy you with music genre discussions. In almost seven years at the institute I met many more colleagues, most of which I consider good friends by now. There's not enough space here to thank everyone individually, but anyone who was part of the cinema group, shared discussions in coffee breaks or at lunch, was part of PhD hat building projects or any other craft projects: You're what makes the institute special. Thank you for all the shared laughs and the great atmosphere in the last seven years.

I'd also like to thank everyone involved in my research stay in Kobe, especially Kentaro Sano who so kindly received me as his guest researcher. Thanks to the whole team in Kobe as well, I learned so much from you. The opportunity to see a culture which is sometimes so different from the German one, was probably a once-in-a-lifetime experience. Special thanks go to Carlos Cortes for being an awesome friend, tour guide and for always finding all the best places to eat in Japan.

An apology to my non-institute friends and family: You didn't get to see me a lot when I was writing the thesis and I am sorry for missing one or the other event. Special thanks to everyone in the board game group, for offering an escape from sometimes boring day-to-day life: To Valentin and Sara for always offering shelter, Benni for figuring out the best strategies, Jannika for being the best hiking-buddy one can think of, Feli and Steven for always having an ear for me ranting about something, and Pia and Reiner for always coming back the long distance to Karlsruhe to meet everyone. Thank you all for the countless fun hours. Thanks also go to my small family core: To my parents, Hans and Gaby, who always supported me on my way to the PhD. You always believed in me and let me go my own way, even though academia is not really

your world. To my grandma Anne-Marie, who still can't believe that sitting in front of a screen in home-office is real work, and to my brother Mathias, who has to do the real work while I'm sitting in front of screens. Also thanks to my aunt, uncle and cousin Johanna, Kurt and Monika. I'll promise to join family meetings more often again.

Last but not least, I'd like to thank the giants on whose shoulders I'm standing on: This includes all the countless scientists, who did the previous research that enabled the work I did as part of this thesis. It also includes all those, that invested their time to teach me those scientific concepts in 13 long years of school and five years of university. All those motivated teachers who put in unpaid extra hours, true humanists burning for education and enabling social advancement through the promotion of knowledge. Thanks to all those who still do more than "Dienst nach Vorschrift" and burn for something more than ever more money in these neoliberal times.

This page intentionally left blank

To all my friends, present, past and beyond
— Pennywise

This page intentionally left blank

Contents

This page intentionally left blank

Notation

This chapter introduces the notation and symbol types which are used in this thesis.

General notation

Numbers	1.0	Numbers are set like this: 1.0.
Code	`code`	Inline program code is set in `typewriter` font.
Variables	`a`	Variables are set in `typewriter` font.
PDKs	*XT018*	PDK names are referred to in *italics*.
Gates	*XOR*	Gates are referred to in *italics*.
Blocks	*CTRL*	References to blocks in diagrams are given *italics*.
File	`.vhd`	File extensions are set in `typewriter` font.
Tools	*Genus*	Tool names are set in *italics*.
Sets	$x_0..x_N$	Simple sets are abbreviated as $x_0..x_N$, meaning $\{x_i \mid i \in \mathbb{Z} \wedge 0 \leq i \leq N\}$.
Intervals	$[n,m]$	Intervals are abbreviated as $[n,m]$, describing an interval of real numbers including the limits n and m.

Terminology

Technology	Process and/or transistor type used.
Hard Logic	Logic realized in non-reprogrammable ways.
Soft Logic	Logic realized in reprogrammable logic ways.
Faster Logic	Higher application frequency, i.e. less propagation delay in the critical path.

Symbols

α	Switching activity.
$C_{\mathbf{in}}$	Input capacitance.
$C_{\mathbf{L}}$	Load capacitance.
$C_{\mathbf{tot}}$	Total load capacitance in a path.
$f_{\mathbf{clk}}$	Clock frequency.
$I_{\mathbf{D}}$	The current flowing through the drain contact of a FET.
$I_{\mathbf{off}}$	Off current of a FET: The maximum current when the transistor off voltage is applied to the respective gate.
$I_{\mathbf{on}}$	On current of a FET: The maximum current when the transistor on voltage is applied to the respective gate.
$L_{\mathbf{eff}}$	Effective transistor channel length, i.e. including process variation.
μ	Charge carrier mobility in a FET.
T	Temperature.
$t_{\mathbf{arr}}$	Actual arrival time: Actual time a signal arrives at a Flip-Flop (FF) input.
$T_{\mathbf{clk}}$	Clock period: Time between two rising clock edges.
$t_{\mathbf{f}}$	Fall time: Time for an output to fall from 80 % of its steady state high value to 20 %.
$t_{\mathbf{hold}}$	Hold time: Time a signal at a Flip-Flop input must be stable after the clock edge arrived.
$t_{\mathbf{PD}}$	Propagation delay: Time for a signal to propagate through a cell.
$t_{\mathbf{r}}$	Rise time: Time for an output to rise from 20 % of its steady state high value to 80 %.
$t_{\mathbf{req}}$	Required arrival time: Latest time a signal must arrive at a FF input at for timing closure.
$t_{\mathbf{setup}}$	Setup time: Time a signal at a Flip-Flop input must be stable before the clock edge arrives.

$t_{\mathbf{skew}}$	Clock skew: Time a local clock is shifted compared to a global, virtual time reference clock.
$t_{\mathbf{slack}}$	Slack: Difference between required arrival time t_{req} and actual arrival time t_{arr}.
$t_{\mathbf{WD}}$	Wire delay: Part of propagation delay caused by parasitic effects of connected wires.
V	Supply Voltage.
VDD	Supply Voltage Positive Potential.
VSS	Supply Voltage Ground Potential.
ΔV	Variation in Supply Voltage.
$V_{\mathbf{IH}}$	Input voltages above this threshold are stable high signals.
$V_{\mathbf{IL}}$	Input voltages below this threshold are stable low signals.
$V_{\mathbf{BS}}$	Body Bias voltage, as potential difference between bulk and source contacts.
$V_{\mathbf{th}}$	Threshold Voltage of a FET.
$V_{\mathbf{th,FG}}$	Threshold Voltage of the Front Gate.
x	Input of a logic gate.
y	Output of a logic gate.

Part I

Prologue

This page intentionally left blank

Chapter 1

Introduction

This thesis describes the overall design and some specific features of the PARFAIT. It was derived by the author as part of the PARFAIT research project, which focuses on the use of RFET technology in FPGAs. Whereas project partners focus on development of RFET transistor devices [1, 2] and circuits [Reu20, Reu21], this thesis discusses system level aspects related to use of RFET technology in FPGA architecture design. Although this thesis provides a short introduction to the device and circuit aspects in chapters 2 and 3, the reader is referred to additional publications for a complete overview of the PARFAIT research project.

Power Reduction In recent years, power efficiency of embedded systems has become more and more important. Reducing power consumption is a challenge especially for FPGAs, which are commonly considered to be less energy efficient than ASICs or Central Processing Units (CPUs). The power losses in ICs are generally classified into static and dynamic power, as will be explained in section 2.3. The dynamic power part is dominated by switching power, which is proportional to the clocking frequency.

In FPGAs, user applications usually do not make use of all available resources. Unused resources do not have switching transistors and therefore do not exhibit dynamic power loss. They however are affected by static power losses, especially by static leakage paths. These paths are caused by transistors in off state conducting leakage current, thus essentially forming paths between power supply rails. CPUs and ASIC use well-established techniques to reduce power in such contexts: Solutions such as power-gating can turn of unused parts of ICs. These approaches have not seen widespread application in FPGA though: Which resources are unused depends primarily on the FPGA user application. This information is not known at the time the FPGA is manufactured, rendering many of the ASIC power management solutions unusable. The primary objective of this work is to investigate approaches to reduce static leakage power suitable for FPGAs.

PVTA This dissertation also addresses PVTA in FPGAs. As will be explained in detail in section 2.4, PVTA effects describe variations of transistor characteristics because of various causes. Such varying transistor characteristics lead to varying propagation delays of logic cells (section 2.3). These effects therefore need to be taken into account when designing ICs: Process variation causes differences in transistor characteristics introduced by the IC manufacturing process. These variations can cause both differences between ICs and differences between different logic cells in one IC. In FPGAs, again many of the established solutions for ASICs are not applicable: When a user application is synthesized to a bitstream, it is expected to work on any FPGA IC. Whereas ASICs can be measured and binned after production, this is therefore not readily possible for FPGAs. Each FPGA IC must work with any bitstream, regardless of the actual resources used in the bitstream. Speed grading in FPGAs therefore can not be based on specific resources on the device, but must always consider all resources.

Voltage and temperature variation can cause similar effects: Locally increased power consumption can lead to a localized drop in supply voltage or temperature hotspots. As the use of FPGA resources is not known during manufacturing time, solutions such as locally increasing power grid density can not be used. Unlike process variation, the effects on the user application can however be estimated during synthesis of the user application. In addition to those effects, aging describes wear of devices over time.

When performing Static Timing Analysis (STA) for FPGA applications, EDA tools have to assume the worst-possible delay a IC could have, effectively using the worst case process variation. In reality, not all resources on each IC will be affected by worst-case process variation. The application may therefore have a larger timing slack in practice then predicted during STA. This unused performance may also be interpreted as a waste of power: Solutions which trade-off transistor performance with power could enable power reduction, if the available timing slack was known.

Performance Power Trade-Off The performance of ICs is mainly characterized by the propagation delay through the used logic cells. As mentioned, this delay relates to the drain current I_{D} of used transistors. Both this on-current I_{D} and the off-current leakage I_{D} are affected by the threshold voltage V_{th}. Trade-offs between performance and leakage power can therefore be achieved if V_{th} can be modulated.

Depending on the technology used, there are various effects that can be used to modulate V_{th}. In bulk silicon technology and more effectively in

SOI, body biasing can be used (see section 2.1). For the RFET technology used in PARFAIT, a similar effect can be achieved through voltage scaling on the Program Gate (PG), an additional gate introduced in RFET devices. A quick introduction to this effect will be given in section 2.2. The newly introduced PG also enables novel approaches for reconfigurable logic. This dissertation will therefore also discuss the use of LUT replacements based on RFET technology in FPGAs.

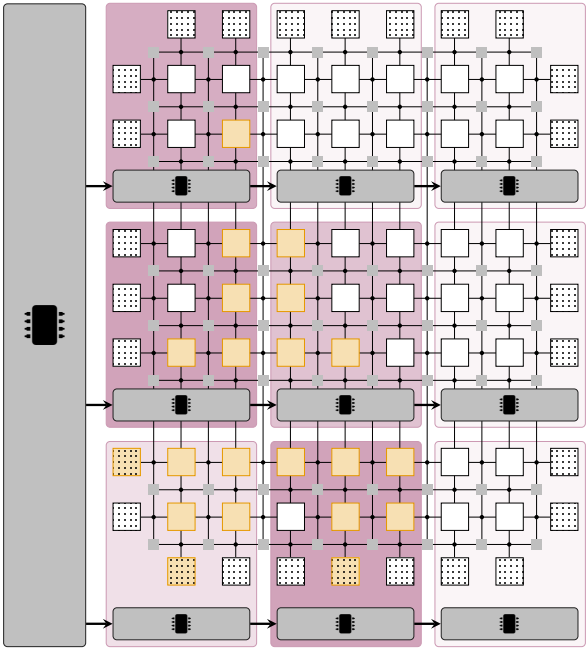


Figure 1.1: The PARFAIT FPGA architecture with the shade of red in each region representing locally adjusted performance using PVTA compensation. Also shown are region controllers and the main controller, which orchestrate logic invasion and measurement.

Power Regions Scaling V_{th} instead of VDD enables connecting regions with different performance: In VDD scaling, when gates of transistors in one region are being driven by transistors in another region, certain restrictions for the voltage levels arise. With V_{th} scaling, all gate and supply voltages are identical, avoiding these problems. V_{th} scaling therefore enables fine-grain power regions as shown in figure 1.1.

With power regions introduced in chapter 7, each region enables a local trade-off between performance and power. When additionally supporting dynamic adjustment of this trade-off, the system can also be used to counter PVTA effects. For this, in addition to knowing the application slack in each region (section 8.1), the FPGA architecture also has to determine the current real performance under PVTA effects. Chapter 8 describes the logic invasion method introduced as part of this thesis: It invades application resources to periodically characterize the performance of each resource in the FPGA, without interrupting the user application.

Logic Cells For logic cells to use V_{th} scaling, special RFET reconfigurable cells are used. Those can provide more area efficient configurable logic than LUTs, but come with certain limitations: Most notably, those ULM can often not realize all functions of N input variables, but only a certain subset. Chapter 6 explains how such cells can be used in an FPGA architecture and what changes are necessary. In addition, chapter 5 describes how RFET cells can be used to realize standard cell based applications.

Evaluation In order to evaluate the PARFAIT FPGA architecture, section 9.1 introduces optimizations for the Virtual FPGA (VFPGA), enabling evaluation on commercial FPGAs. Such a VFPGA-based evaluation can however not simulate the PVTA effects. Because of this, the main evaluation of this thesis' results is based on simulation. Section 4.6 introduces the technology models that derive propagation delay t_{PD} from PVTA parameters. Models are then extended to also consider V_{th} scaling using a control parameter. Two models are introduced: One for a reference SOI technology, based on Scarpato's work [3]. This model is extended to support V_{th} scaling and then parametrized for the used technology. In addition, the Scarpato model is adjusted for the RFET technology characterized in [2]. Here, some changes in modeling are necessary. The model is then fitted to the measured RFET characterization data of [2] to obtain propagation delays.

To obtain t_{PD}, the delay model requires PVTA parameter inputs. Section 4.7 describes the scenario models used to derive realistic PVTA parameters for the evaluation. The control parameter is calculated by region controllers in the FPGA architecture, as shown in figure 1.1. To simulate these Very High Speed Integrated Circuit Hardware Description Language (VHDL) based controllers and the delay model, a co-simulation framework will be introduced in section 9.4. The final evaluation in chapter 10 then uses a set of benchmark user FPGA applications introduced in section 9.5. Those are first placed on the PARFAIT FPGA with different regions sizes, then the slack factors for all

regions are calculated. Using these factors and the scenario models, the co-simulation evaluates the region controller responses. The relative power and the control voltage in each region, as well as the achieved delay factors, will be presented in the results chapter.

Novel contributions As part of this thesis, various new topics have been investigated or advanced significantly beyond state of the art. The following list provides an overview, referencing the relevant section in the dissertation.

- Parametrization of the Scarpato delay model for SOI technology and extensions to model body biasing: Section 4.6.
- Modification of the Scarpato delay model for RFET technology and parametrization according to measurements: Section 4.6.
- Evaluation of large digital circuits in RFET technology using a custom standard cell library: Section 5.3.
- Derivation of relative power and delay metrics, that can be used in absence of absolute power and delay information: Sections 4.6 and 9.2.
- Derivation of a toolflow to map FPGA applications to ULMs: Section 6.2.
- Design of Controllable Logic Blocks (CLBs) based on ULMs and metrics and methodology to evaluate the designs: Section 6.4.
- Introduction of Logic Invasion, efficiently reusing CLBs for t_{PD} characterization of all CLBs with little resource overhead: Section 8.2.
- Introduction of slack factors to describe the available timing slacks in power regions, and algorithms to derive those: Section 8.1.
- Introduction of a co-simulation framework to evaluate power aware architectures and derivation of related PVTA scenarios: Section 9.4.
- Evaluation of power saving in the PARFAIT: Section 10.4.
- Introduction of manual placement techniques for VFPGA: Section 9.1.

This page intentionally left blank

Chapter 2

Fundamentals

This chapter reviews necessary prerequisites for understanding of the PARFAIT architecture. Explanations are kept succinct, as the reader is assumed to possess knowledge of electrical and information systems engineering. If further explanations are desired, the end of each chapter includes references to relevant standard works discussing the topics in more breadth and depth.

2.1 Classic Silicon Semiconductors

Silicon semiconductors used for CMOS circuits are realized in various different physical device implementations, mostly depending on target feature size and other application characteristics. In this section, only basic device implementations will be discussed, as this is sufficient to describe the functional difference between devices for classic Complementary Metal Oxide Semiconductor (CMOS) and RFET.

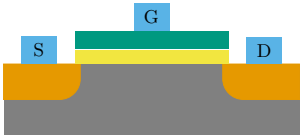
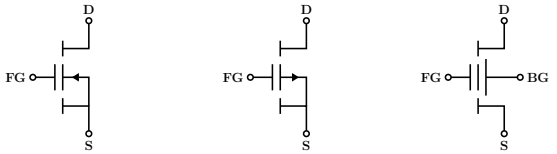


Figure 2.1: Cross-section of a basic bulk-silicon type Metal Oxide Semiconductor FET (MOSFET). The bulk (or body) silicon is electrically connected to the source (S) terminal. Apart from this, the source (S) and drain (D) terminals are symmetrical. Materials for N-Channel MOSFET (NMOS) / P-Channel MOSFET (PMOS) are n/p doped bulk (black), n+/p+ doped source and drain regions (orange), some insulator for the gate oxide (yellow), polysilicon for the gate region (green) and metal for the wiring contacts (blue).

Figure 2.1 on the preceding page shows the most basic MOSFET, a bulk type device. It is stacked onto the bulk or body silicon, which is provided by the wafer itself and usually lightly doped. Source and drain regions of the wafer are n+ doped for NMOS devices. An oxide layer – in the simplest form silicon oxide – is then used to isolate the channel regions from the polysilicon gate region. Gate, source and drain are connected to the metal wiring layers to form circuits. The body contact connecting to the bulk is not shown in figure 2.1. It is often globally connected to the source terminal of the respective device type and not available as an individual terminal. In a simple explanation, the electric field between gate and bulk forms a depletion region and causes charge carriers to form a conducting channel below the gate oxide. When this channel has been established, a current flow between gate and source can be induced when a respective potential difference is applied across the source and drain terminal. In this current flow, generally only one type of charge carrier is involved. The distinction between source and drain contacts is needed because of the potential applied to the bulk terminal.

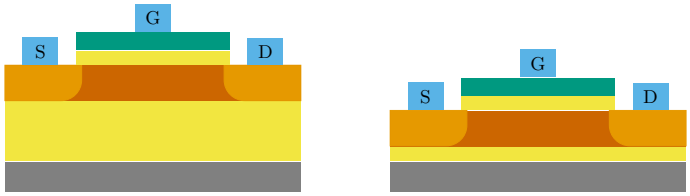


(a) NMOS (b) PMOS (c) 4-T MOSFET

Figure 2.2: Symbols and Terminals for MOSFET Devices.

In the bulk MOSFET, the potential of the device body is usually applied globally to the bulk substrate, so individual control of the potential per transistor is not possible. The device can therefore be described as a three-terminal device, as indicated by the commonly used symbols shown in figures 2.2a and 2.2b. Other device types allow more fine-grain access to the body potential and can thus support four-terminal device control as shown in figure 2.2c. An example for such a technology are SOI MOSFETs as shown in figure 2.3a on the next page. Unlike bulk MOSFETs, the semiconductor layers are not etched and doped into the bulk silicon substrate. In SOI devices, the substrate is instead electrically isolated using a Buried Oxide (BOX), onto which semiconductor layers are deposited.

A SOTB device, a special realization of SOI, is shown in figure 2.3b on the facing page. Compared to normal SOI devices, SOTB devices feature a thinner BOX. The main benefit of a thin oxide is better modulation of the device channel



(a) SOI MOSFET (b) SOTB MOSFET

Figure 2.3: Cross-sections of SOI MOSFETs. The transistor layers are similar to the bulk MOSFET, but all layers are stacked onto a buried oxide layer. **(a)** The conventional SOI MOSFET with thick BOX. **(b)** Silicon on Thin BOX (SOTB) device with thinner BOX layer. These devices differ in manufacturing and in electrical characteristics.

from the substrate below the BOX, including more effective body biasing [4, p. 14]. In a generalized abstract view, a contacted substrate can act similarly to the transistor gate. To be able to better describe these contacts in these devices, gates are usually distinguished as Front Gate (FG) and Back Gate (BG).

In conventional designs, the substrate is connected to the same voltage potential as the source terminal. Body Biasing (BB) designs use a potential difference between substrate and source to enable more control over the threshold voltage V_{th} and the on (I_{on}) and off (I_{off}) currents. For bulk devices, the body effect is commonly described as a shift in V_{th}:

$$V_{\text{th}} = V_{\text{th0}} + \gamma\left(\sqrt{|-2\Phi - V_{\text{BS}}|} - \sqrt{|-2\Phi|}\right) \tag{2.1}$$

Here, V_{BS} is the body biasing voltage, V_{th0} the threshold voltage without body biasing, Φ the substrate Fermi potential and γ is the body effect coefficient, a technology specific constant. The maximum potential difference in bulk silicon designs is limited, leaving most benefits of body biasing for SOI devices. As the substrate is isolated from the channel by the BOX, the terminal used for body biasing in these technologies is commonly referred to as Back Gate (BG). Whereas BB conceptually makes the MOSFET a four-terminal devices as shown in figure 2.2c on the preceding page, the fourth terminal's purpose is usually limited to power saving. Unlike four-terminal RFETs introduced in the next chapter, the classic MOSFET structure with n or p channels is kept. Only one type of charger carrier, electrons in the NMOS or holes in PMOS, is involved in current flow.

SOI devices can be divided into Partially Depleted SOI (PDSOI) and Fully

Depleted SOI (FDSOI) types. For FDSOI, the channel material is intrinsic silicon. As the channel material is not doped, the channel is fully depleted. For PDSOI, the channel material is doped like in bulk devices. PDSOI is commonly used for thick BOX devices, whereas FDSOI commonly goes with SOTB. FDSOI technology has recently been adopted in industry and designs have demonstrated speed improvements and power reductions for both logic and memory applications [5, 6, 7].

The drain current I_{D} of the transistor plays an important rule in the development of digital circuits. It depends on the technology type and its formulaic description changes depending on the region of the transfer characteristic, the transistor is used in. For example, in the Shockley model, there is a cut-off, a linear and a saturation region. Static operation of digital circuits uses transistors in the saturation region. Drain current in the Shockley model is described for the saturation regions as below [8]:

$$\begin{aligned} I_D &= 0.5K(V_{\mathrm{GS}} - V_{\mathrm{th}})^2 \\ &= C_s(V_{\mathrm{GS}} - V_{\mathrm{th}})^2 \end{aligned} \tag{2.2}$$

The Shockley model has been found to be insufficient to describe modern Field Effect Transistors (FETs), as it does not model short-channel effects. It is therefore commonly replaced by a more general formula, the alpha-power law. This thesis will later on make use of the Scarpato model to describe circuit delay depending on PVTA. As the Scarpato model is based on the alpha-power-law, this necessitates its introduction [3]. Digital circuits operate in the pentode region of that model, where the drain current is defined as follows [8]:

$$\begin{aligned} I_D &= \frac{W}{L_{\mathrm{eff}} P_C}(V_{\mathrm{GS}} - V_{\mathrm{th}})^{\alpha} \\ &= C_{\alpha}\mu(V_{\mathrm{GS}} - V_{\mathrm{th}})^{\alpha} \end{aligned} \tag{2.3}$$

Both models make use of different constants, but only the temperature dependency of those is relevant for this thesis. This dependency was analyzed by Dasdan and Hom, especially focusing on the Inverted Temperature Dependence (ITD) effect [9]. They note that the carrier mobility μ and threshold voltage V_{th} depend on the temperature like this:

$$\mu(T) = \mu(300)\left(\frac{300}{T}\right)^{m} \tag{2.4}$$

$$V_{\mathrm{th}}(T) = V_{\mathrm{th}}(300) - \kappa(T - 300) \tag{2.5}$$

It can therefore be seen that rising temperature T leads to decreasing mobility μ and therefore decreasing I_D. On the other hand, rising temperature causes decreasing threshold voltage V_{th} and therefore increasing I_D. Which effect is dominant, and whether I_D ultimately increases or decreases with T, depends on the supply voltage VDD [9].

Further Reading

A broad range of textbooks are available discussing various aspects of classic silicon technology. For a broad overview of topics concerning digital circuit design, see [Rab03]. For an introduction into Metal Oxide Semiconductor (MOS) technology and details on analog circuit design, see [Raz16]. An introduction into Ultra-Thin-Body MOSFETs and SOI in general can be found in [Fos13]. For details about manufacturing and devices in SOI, [Kon14] may be used as a reference. Articles describing various body biasing techniques for FDSOI devices at various granularity have been collected in [Cle20].

[Rab03] Rabaey, Jan M.; Chandrakasan, Anantha P. and Nikolić, Borivoje: Digital integrated circuits: A design perspective. 2. ed. Prentice Hall electronics and VLSI series. Upper Saddle River, NJ: Prentice Hall, 2003.

[Raz16] Razavi, Behzad: Design of analog CMOS integrated circuits. Second editon. New York: McGraw Hill Education, 2016.

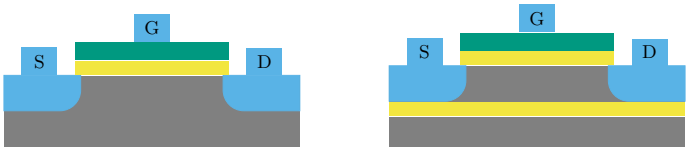
[Fos13] Fossum, Jerry G. and Trivedi, Vishal P.: Fundamentals of ultra-thin-body MOSFETs and FinFETs. Cambridge: Cambridge Univ. Press, 2013.

[Kon14] Kononchuk, O. and B-Y., Nguyen, eds.: Silicon-on-insulator (SOI) technology: Manufacture and applications. Woodhead Publishing series in electronic and optical materials. Cambridge, UK: Woodhead Pub, 2014.

[Cle20] Clerc, Sylvain; Di Gilio, Thierry and Cathelin, Andreia, eds.: The Fourth Terminal: Benefits of Body-Biasing Techniques for FDSOI Circuits and Systems. 1st ed. 2020. Integrated Circuits and Systems. Cham: Springer International Publishing and Imprint Springer, 2020.

2.2 Ambipolar Silicon Semiconductors

In all devices introduced in section 2.1, current through the channel involves only one type of charge carrier. The type of carrier is fixed at manufacturing time by selection of materials, usually by the doping of the source and drain regions. Devices where current flow through the channel involves both electrons and holes, are called ambipolar devices. With early ambipolar devices, the ambipolarity was not actively used to enable additional features and devices were designed to suppress the ambipolar behavior [10, 11]. A recent review publication summarizes various approaches which have been taken in this respect [12]. Active use of ambipolarity was initially described in [13] and is the main idea of Reconfigurable (Ambipolar) FETs (RFETs), which use the ambipolar behavior to switch device polarization. It should therefore be noted that not all ambipolar devices qualify as RFET.



(a) SBFET (b) Planar RFET

Figure 2.4: Cross-sections of basic ambipolar MOSFETs. Unlike bulk and SOI MOSFETs, the shown devices employ source and drain region materials with metallic behavior. Junctions between these regions and the channel region show Schottky junction behavior. **(a)** The basic Schottky Barrier FET (SBFET) device is derived from the bulk MOSFET. **(b)** A basic planar Reconfigurable (Ambipolar) FET (RFET) device derived from the SOI MOSFET.

Figure 2.4a shows the cross-section of the Schottky Barrier FET (SBFET) device. The SBFET's structure is similar to the bulk MOSFET of figure 2.1 on page 9, with the main difference being the use of different material for source and drain regions. Instead of using p- or n- doped silicon, SBFETs use materials with metallic behavior, usually a silicide. The use of these materials creates Schottky junctions between the channel and these regions. The effect of the gate can then now longer be described as forming a conducting channel through accumulation of charge carriers. Instead, a more complex analysis has to consider band diagrams in the device and of the junctions, and how the electric field formed by the gate contact interacts with those junctions. For the use of the devices in this thesis, a detailed understanding of such effects is not necessary and the reader is referred to literature [1].

The use of Schottky contacts leads to ambipolar behavior, i.e. both charge carrier types are involved in current flow. The SBFET is a conventional three terminal device, so the ambipolarity is usually not used to enhance the FET functionality.

Figure 2.4a shows the idea of a simple, planar RFET device. From a technology and manufacturing perspective, this device is related to the basic SOI and SOTB devices. Like those, it builds on top of a BOX, where usually a thin-BOX concept is used to enhance electrostatic control using the back gate. Like the SBFET, it uses source and drain region materials with metallic behavior, but unlike the SBFET it uses four or more terminals. In RFET concepts, the polarity of the channel can be changed. For the device shown here, applying a certain voltage potential bias can either make the device act like a PMOS or a NMOS device. The analogy is limited to functional level, in particular the polarity of the front gate threshold voltage $V_{\mathrm{th,FG}}$. More detailed analysis of channel currents will show differences in device physics and behavior when compared to classic NMOS of PMOS devices. For the architecture-level analysis in this thesis, these details are not relevant. The reader is referred to [1] for more details.

Some types of RFET devices, such as e.g. the PARFAIT device, do not use any doped materials. Instead, they rely only on intrinsic silicon, poly silicon, oxides and silicides [14]. Apart from removing reducing manufacturing steps, this also allows devices to be used across a wider temperature range: Freeze-out at low temperature and intrinsic behavior at high temperature are not an issue with these RFET concepts [15]. As the functionality of the device is determined by the BG potential instead of by doping, the process of configuring the device using a static voltage potential is called electrostatic-doping [16]. From fabrication point of view, electrostatic doping poses similar requirements as body biasing in SOI devices [17]. If reconfiguration is not required, RFETs can be configured statically as PMOS or NMOS devices by statically connecting the back gates of all respective devices to the respective supply [18]. Alternatively, the back gates can be connected to variable voltage rails to enable similar effects as body biasing in SOI.

Figure 2.5 shows the final RFET device developed in the PARFAIT 1 project. To enhance device characteristics, the FG has been split into three gates. Two TG are directly placed above the Schottky junctions and are used to configure the device polarity. The FG is located in the center of the channel and is used as the main control gate for this device. Both TG are usually connected internally to reduce external wiring overhead and therefore influence both junctions identically. In its most general configuration, the PARFAIT RFET

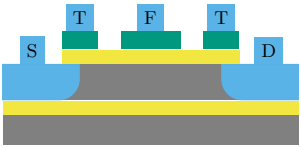
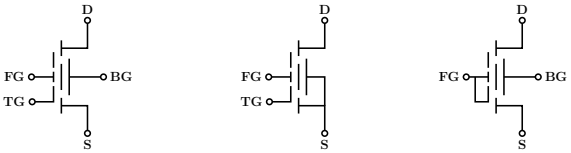


Figure 2.5: Cross-section of the PARFAIT RFET, adapted from [19]. Compared to the basic planar RFET, the PARFAIT RFET splits the top gate into three independent gates. The Top Gates (TGs) (T in the figure) are internally connected and exposed as one contact.

is a five-terminal device as shown in figure 2.6a. It features the well-known source and drain terminals connected to the channel, the back gate below the BOX and one contact for TG and FG each.

To reduce complexity both in connecting the devices using metal layers and in circuit design, certain device variations with combined terminals have been devised: Figure 2.6b shows a variant where the BG has been connected to the source terminal. This design is based on the bulk MOSFET design, where the substrate or well is connected to the source terminal. As thin-BOX technology is not always available, the BG is often separated from the channel through a larger oxide than TG. Because of this, TG often provides a more efficient way to manipulate the Schottky junction, justifying this device design. For details, please refer to [19] and [1]. Figure 2.6c shows a variant where the TG and FG has been connected. This configuration resembles the more traditional design of figure 2.4 on page 14 with one front and one back gate. It can be used for circuits originally devised for these less complex devices.



(a) 5-T (b) BG-S Combined (c) FG-TG Combined

Figure 2.6: Symbols and Terminals for the PARFAIT RFET Devices.

Whereas this introduction derived ambipolar transistors from planar SBFETs, ambipolar behavior was chronologically first explored as part of other device technologies. A summary of early publications can be found in [20] and more current ones are summarized in [21]. Early devices were mostly

one-dimensional devices and include silicon [22, 23, 24, 25, 26] or germanium nanowires [27, 28] as well as carbon nanotubes [13]. Later on, two-dimensional devices using graphene [29], dichalcogenides [30, 31, 32, 33] and black phosphorus [34] materials were introduced. Planar, silicon based RFETs on the other hand can be considered as an extension of existing transistor technology. They can be realized with comparatively small changes to existing manufacturing processes [18]. As shown in the deduction of the PARFAIT RFET from SOI and SBFET devices, the main difference is in the material of source and drain regions. As silicides are already available in existing processes, the technology can be adapted for most FDSOI processes, but a thin-BOX process is advantageous for good electrostatic behavior. This has lead planar RFETs to now entering the world of commercial manufacturing processes. For example, devices have recently been integrated into GlobalFoundries FDX22 [17] 22 nm FDSOI technology with minimal changes to the manufacturing process [35, 36].

From a circuit perspective, RFETs can be divided into two groups [18]: Those with independent control of the source and drain junctions and those with only combined control over both interfaces. The PARFAIT device in figure 2.5 on the preceding page is an example of the second category: Although it does provide two TGs, those are internally connected. The BGs of figures 2.4b and 2.5 on page 14 and on the preceding page also control both junctions at the same time, and therefore also belong to category two. In the remaining thesis, when RFET technology is referred to, a device of the second category will be assumed.

Compared to conventional silicon technology, RFETs offer three primary benefits which will be leveraged in this thesis: As already mentioned, RFETs offer reconfiguration, providing a way to change the polarity of the device not only during manufacturing, but also in the field. This technological advantage enables the design of various circuits with fewer transistors, potentially reducing area and power of circuits [18]. Enabling hardware reconfiguration, this immediately benefits FPGAs and will therefore be discussed in detail as part of this thesis. Whereas these logic cells for FPGA based on RFET technology will be introduced in section 2.6 on page 53, an overview of reconfiguration and further applications can be found in literature [18, 26, 37, 38, 39]. The second major benefit of RFETs is the increased temperature range [40]. As the devices do not require doping, wide temperature ranges [15, 41] allow targeting cryogenic and high-temperature applications. Whereas this enables the devices to function over a wide range, temperature-dependent change of device performance is still likely. This thesis therefore will introduce a mechanism to counter temperature variation induced effects and discuss

them in more detail in section 2.4 on page 27. The third and most important advantage is the ability to adjust the threshold voltage of the device [37, 38, 42, 43]. Whereas for basic RFETs changing V_{th} happens with the same terminal that programs the polarity of the device, more advanced designs such as the PARFAIT 1 RFET allow independent control using additional terminals. An adjustment of the threshold voltage changes the static leakage current and affects the dynamic power as well. On the other hand side, it modifies the on-current I_{on}, affecting the delay times and therefore performance of the circuits. A detailed derivation of these effects will be given in section 2.3. The trade-off between power and performance forms the main aspect of this thesis.

The planar PARFAIT device is also compatible with CMOS processes, which enables mixed circuits made of both RFET and normal CMOS devices. This allows for more rapid integration and will be used in this work to reduce the complexity of the RFET standard cell library presented in chapter 5 on page 161. RFETs further possess some advantageous properties for analog and RF design, which are not further explored here [Reu22]. The reader is referred to the PARFAIT project publications and related work for more information on those topics.

Further Reading

As academic research initially focused on suppression of ambipolarity instead of on active use, most information about ambipolar transistor development is currently found in research articles. Most of those articles have narrow scope and are therefore not useful as overview works. They have be referenced in the bibliography section instead. The review articles [Ren19] for ambipolar devices in general and [Hu21, Fei22] for two-dimensional devices serve as a good overview of recent developments. For general information about design of SBFETs devices, refer to the summary in [Rud23]. Additionally, an introduction into characteristics and manufacturing of the specific RFET device used in PARFAIT 1 can be found in [Kra19]. As there has been further research since the finalization of [Kra19], readers are advised to also review the PARFAIT research literature referenced in this section.

[Ren19] Ren, Yi; Yang, Xiaoyang; Zhou, Li; Mao, Jing–Yu; Han, Su–Ting and Zhou, Ye: "Recent Advances in Ambipolar Transistors for Functional Applications". In: *Advanced Functional Materials* 29.40 (2019), p. 1902105. doi: 10.1002/adfm.201902105.

[Hu21] Hu, Wennan; Sheng, Zhe; Hou, Xiang; Chen, Huawei; Zhang, Zengxing; Zhang, David Wei and Zhou, Peng: "Ambipolar 2D Semiconductors and Emerging Device Applications". In: *Small methods* 5.1 (2021), e2000837. doi: 10.1002/smtd.202000837.

[Fei22] Fei, Wenwen; Trommer, Jens; Lemme, Max Christian; Mikolajick, Thomas and Heinzig, André: "Emerging reconfigurable electronic devices based on two–dimensional materials: A review". In: *InfoMat* 4.10 (2022). doi: 10.1002/inf2.12355.

[Rud23] Rudan, Massimo: Springer Handbook of Semiconductor Devices. Springer Handbooks. Cham: Springer International Publishing AG, 2023.

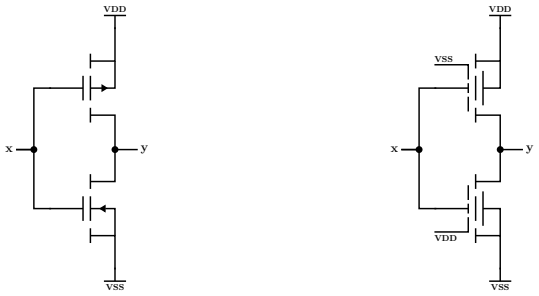
[Kra19] Krauss, Tillmann A.: "Planare elektrostatisch dotierte rekonfigurierbare Schottky-Barriere FDSOI Feldeffekttransistor Strukturen". Dissertation. Darmstadt: Technische Universität Darmstadt, 2019.

2.3 CMOS Circuit Technology

In order to realize logic using transistor devices, those have to be connected in specific ways. There are various logic technologies, but the most commonly used one for digital logic is CMOS technology. In CMOS, logic gates consist of a pull-up network used to connect the output to VDD and a pull-down network, which connects to VSS. To avoid short circuits, it must be ensured that only one of the networks is conducting at a time, i.e. the networks have to be complementary.

This simplified view only applies to the static, steady state case, where all inputs of a cell are constant. Observation of the dynamic behavior, as explained below, will show a current flowing through the pull-up and pull-down networks during the output transition. It should also be noted that some optimized cells – such as multiplexers and LUTs – are commonly realized using other logic, e.g. using pass transistors or transmission gates.

CMOS Cells



(a) Traditional Inverter (b) RFET Inverter

Figure 2.7: Inverter circuit in MOSFET CMOS technology and in statically configured RFET CMOS technology. The MOSFET inverter realizes the classical inverter circuit using PMOS and NMOS devices [44]. The RFET inverter uses a static configuration, where the TG of the inverters are directly connected to the VSS and VDD voltages [Reu19]. In this configuration, RFETs always behave as either PMOS or NMOS transistors and can not be reconfigured.

Figure 2.7 shows a simple inverter as an example for a gate in CMOS logic. Implementations are shown for both SOI MOSFET and RFET technology. As

one of the most basic gates, the inverter features only one input x and one output y, which simplifies the introduction of the required concepts. In the RFET case in figure 2.7b, the RFETs are statically configured by connecting their TGs to VDD or VSS [Reu19]. This way, the transistor shown in the upper part of the figure will behave as a PMOS transistor, the lower transistor as NMOS. As this circuit is not reconfigurable, from a system perspective it behaves similar to the SOI inverter in 2.7a. From a circuit perspective, electrical characteristics such as the voltage transfer characteristic of course vary.

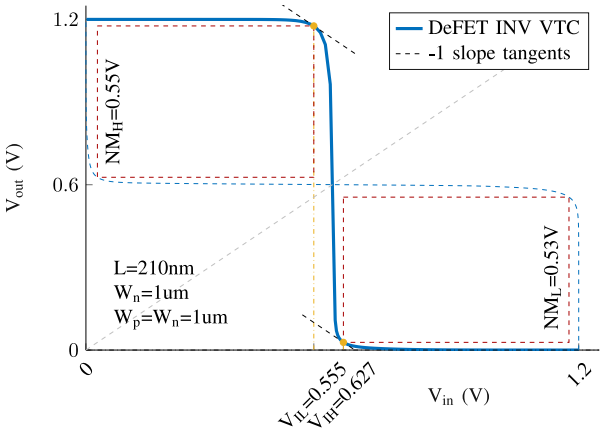


Figure 2.8: Inverter voltage transfer characteristic, showing the output voltage depending on input voltage for the RFET inverter of figure 2.7b. The figure depicts noise margins, slope tangents and the low and high levels derived from those. Taken from [Reu19].

Power: Figure 2.8 shows the voltage transfer characteristic for a statically configured RFET inverter, closely resembling those of an SOI inverter [Reu19]. As can be seen from the figure, an input voltage between V_{IL} and V_{IH} will cause an output voltage somewhere in between the low and high levels. In this range, both the pull-up and the pull-down network are conducting, leading to a current flow from VDD to VSS. For static operation, a circuit designer has to ensure that the input voltage is out of this range. For dynamic operation, when the input is switching from low to high or vice versa, the input voltage will be in this range for some time.

In order to quantify the energy loss during such a transition, a closer look at power loss in CMOS gates is necessary. In general, dynamic power loss in

nanometer CMOS is caused not only by short-circuit currents, but mostly by charging and discharging of load capacitance. This load capacitance is constituted of parasitic input capacitance of the gates connected to an output, as well as of the capacitance of the metal connection. In addition to those dynamic effects, which occur only during transitions of input signals, there are also static power dissipation effects, occurring even when signals are constant. These effects include subthreshold leakage, gate leakage, junction leakage and contention current. The following formulas summarize the power dissipation in CMOS circuits [45]:

$$P_{\text{total}} = P_{\text{dynamic}} + P_{\text{static}} \tag{2.6}$$

$$P_{\text{dynamic}} = P_{\text{switching}} + P_{\text{short circuit}} \tag{2.7}$$

$$P_{\text{static}} = \left(I_{\text{sub}} + I_{\text{gate}} + I_{\text{junct}} + I_{\text{cont}}\right) V_{\text{DD}} \tag{2.8}$$

Switching power is given as:

$$P_{\text{switching}} = \alpha C_{\text{L}} V_{\text{DD}}^2 f \tag{2.9}$$

Where C_{L} is the load capacitance, VDD is the supply voltage, and the clock frequency f_{clk} multiplied by the activity factor α gives (half) the amount of transitions per second. The switching power therefore is directly proportional to the frequency f and the activity rate, which specifies in what proportion of the cycles the signal is actually switching. Switching power is therefore ultimately dependent on the number of transitions. Short circuit current has become mostly negligible in nanometer processes [45].

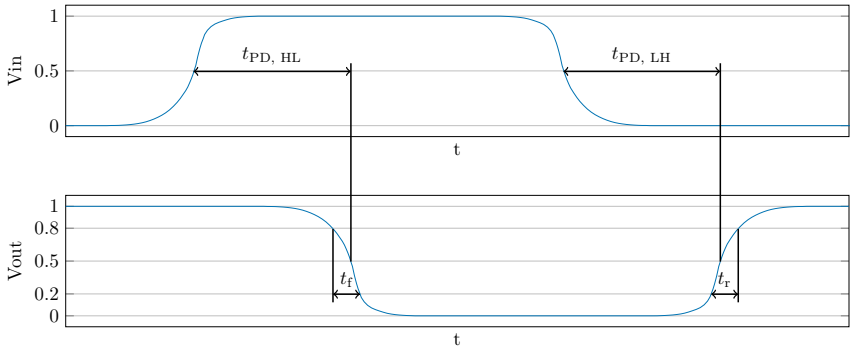


Figure 2.9: Definition of the propagation delay t_{PD} for both high-to-low and low-to-high transition at output of an inverter. Also shown are the definitions of the rise and fall times t_{r} and t_{f}.

Delay: Figure 2.9 on the preceding page shows the definition of the propagation delay, a value used to characterize dynamic timing behavior of cells [45]. If there is a change of the value of a cell input that leads to a change of the value of a cell output, the propagation delay is the time between the input being at 50 % of its high potential and the output reaching 50 % of the high potential. Propagation delay can vary for different transitions. For an inverter, the only two possible transitions, a falling output and a rising output, are shown in the figure. More complex gates with multiple inputs and outputs accordingly need more propagation delay values to be described completely. The figure also shows definitions of rise and fall times, the time needed for an output to transition between 20 % and 80 % of its steady state high value. Those again depend on the combination of input and output used.

Furthermore, the exact values of those variables also depend on the load connected to the output. Assuming a load capacitance C_L, the propagation delay can be specified based on the time needed to load this capacitance. Using the alpha-power model, this leads to [8]:

$$t_\mathrm{PD} = \left(\frac{1}{2} - \frac{1-\nu_\mathrm{T}}{1+\alpha}\right) t_\mathrm{T} + \frac{C_\mathrm{L} V_\mathrm{DD}}{2 I_\mathrm{D}} \tag{2.10}$$

Neglecting the constant addend which depends on the input signal, Scarpato simplifies this to the following proportionality [3]:

$$t_\mathrm{PD} \propto \frac{C_\mathrm{L} V_\mathrm{DD}}{I_\mathrm{D}} \tag{2.11}$$

Using the alpha-power law model of equation (2.3) for the drain current, this can be written as:

$$t_\mathrm{PD} \propto \frac{C_\mathrm{L} V_\mathrm{DD}}{\mu (V_\mathrm{GS} - V_\mathrm{th})^\alpha}, \tag{2.12}$$

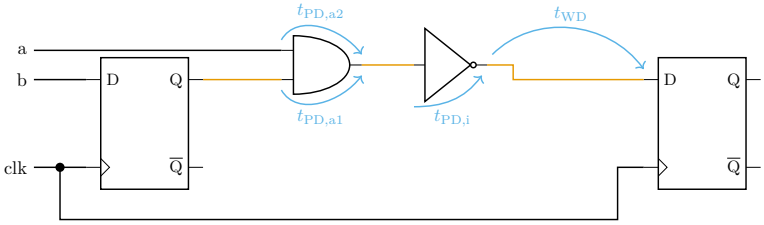
where the temperature dependency of μ and V_th are given in equations (2.4) and (2.5). Scarpato then extends this model for chains of gates, assuming that I_D and VDD are constant and using a replacement capacitance C_tot as the sum of all load capacitances in a path [3]:

$$t_\mathrm{PD,Path} \propto \frac{C_\mathrm{tot} V_\mathrm{DD}}{\mu (V_\mathrm{GS} - V_\mathrm{th})^\alpha} \tag{2.13}$$

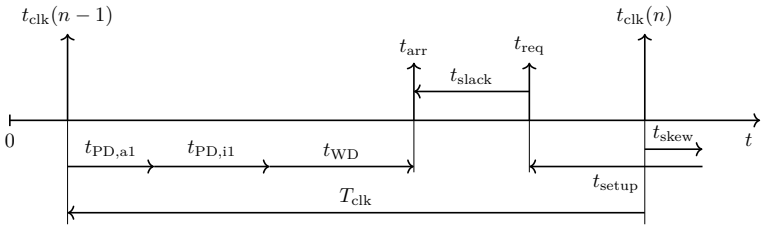
Static Timing Analysis

Figure 2.10a shows a sequential (clocked) circuit consisting of two FFs and a combinational part with one inverter and one *AND* gate. In order for the

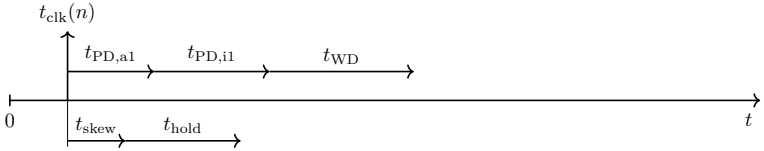
FFs to properly latch the input signal (avoiding zero and double clocking), certain constraints regarding setup and hold times must hold. Conceptually, the setup time is the duration the signal at a FF input must be stable before the clock edge arrives at the FF. Hold time on the other hand is the duration the input signal must be stable after the clock edge arrived. Satisfying these constraints for all FFs in a circuit achieves timing closure and the process of verification is called STA [46].



(a) Example Circuit



(b) Setup Time Constraints



(c) Hold Time Constraints

Figure 2.10: Example circuit and explanation of times used in STA.

Various definitions for the setup time calculations of the example circuit are shown in figure 2.10b. The markers at t_{clk} denote the clock edges of a global, virtual reference clock with period T_{clk}. Because of delay caused by clock routing, the clock may not arrive at exactly the same time at all FFs. In that case, t_{skew} describes the delay of the clock arriving at one FF, compared to the virtual global clock. It essentially "shifts" the clock edges in the figure. In the example, the skew for the first FF is assumed to be zero for simplicity,

i.e. a clock perfectly aligned to the virtual reference. The setup time itself, t_{setup}, describes how long before the clock edge the input signal needs to be stable. This point in time is shown as t_{req}, the required arrival time. The actual arrival time t_{arr} can be calculated by summing all signal delays in the path. This includes the propagation delays t_{PD} of all gates in the path, as well as wire delay t_{WD}. Wire delay usually describes the increase of propagation delay of the driving gate due to parasitic capacitance of the interconnect. In some cases, it also includes propagation delays of buffers inserted by EDA tools to drive long wires. The difference between t_{req} and t_{arr} is called slack t_{slack}. If it is positive, the signal arrives early enough and setup time constraints are adhered to. Putting all this together leads to a formula for the maximum achievable clock frequency and minimum clock period:

$$T_{\text{clk}} \geq t_{\text{comb}} + t_{\text{setup}} + t_{\text{skew}} \tag{2.14}$$

Definitions for the hold time are shown in figure 2.10c, where the local clock may again be delayed by t_{skew}. The hold time t_{hold} denotes the hold time required by the FF. To analyze hold time constraints, the signal propagation time needs to be assessed relative to the same clock edge at the first FF. The formula for a minimum combinational delay is then:

$$t_{\text{comb}} \geq t_{\text{skew}} + t_{\text{hold}} \tag{2.15}$$

As shown in this example, the combinational path is usually formed out of multiple independent elements, ARCs. STA usually represents the complete circuit as a graph and validates the mentioned constraints for all possible paths. As previously explained, t_{PD} varies for different transitions. STA therefore has to consider the worst case (or the worst possible combination) of all propagation delays in a path. Furthermore, cell speed differs depending on temperature, supply voltage and manufacturing effects. Process Design Kit (PDK) vendors therefore usually provide different parametrized sets of values. Fixing the temperature, voltage and process parameters allows to obtain one set of values, a so-called corner. STA in common realizations uses the worst case value, leading to pessimistic results [46, p. 224].

Real circuits will possess many timing paths, but for circuit designers, primarily the critical paths are important. These are paths which either have a negative slack, or the ones with the smallest slack value, where the one with the smallest slack is called critical path. Manual optimizations of combinational delay are usually only needed for setup time constraints. For the hold time constraint, EDA tools can automatically increase the delay of the

combinational path, if necessary. Furthermore, clock skew can be adjusted deliberately to achieve timing closure [46].

Further Reading

CMOS circuit fundamentals have been discussed on different abstraction levels in various standard works. Streetman focuses largely on technology aspects and transistor devices, but also gives a quick introduction to inverter characteristics [Str15]. Razavi also summarizes semiconductor physics, but focuses more on circuit level aspects [Raz21]. He also covers an introduction to CMOS circuits, focusing on noise margins and transition times. Baker also explains circuits such as oscillators, rather than only basic gates [Bak19]. He also covers physical aspects of cell design such as transistor sizing and layout. Weste covers CMOS design from a system perspective [Wes11]. After quickly introducing MOSFET devices and basic concepts such as delay and power in CMOS circuits, he covers delay models, interconnect aspects, combinational and sequential circuit design, EDA and verification. Sakurai focuses on SOI and circuit design for this technology [Sak06]. He also includes a chapter on CMOS circuit design with focus on low-power realization in SOI. An introduction to timing analysis can be found in [Wes11], but a more thorough introduction to this topic is given in [Kah22]. The actual algorithms used in EDA tools are described in more detail in [Ger99].

[Str15] Streetman, Ben: Solid State Electronic Devices: Global Edition. 7th Edition. Harlow: Pearson, 2015.

[Raz21] Razavi, Behzad: Fundamentals of microelectronics: With robotics and bioengineering applications. Third edition. Hoboken: Wiley, 2021.

[Bak19] Baker, Russel Jacob: CMOS circuit design, layout, and simulation. Fourth edition. Vol. 22. IEEE Press series on microelectronic systems. Piscataway, NJ and Hoboken, New Jersey: IEEE Press and Wiley, 2019.

[Wes11] Weste, Neil H. E. and Harris, David Money: CMOS VLSI design: A circuits and systems perspective. 4. ed. Boston, Mass.: Addison-Wesley, 2011.

[Sak06] Sakurai, Takayasu: Fully-Depleted SOI CMOS Circuits and Technology: For Ultralow-Power Applications. Boston, MA: Springer, 2006. doi: 10.1007/978-0-387-29218-2.

[Kah22] Kahng, Andrew B.; Lienig, Jens; Markov, Igor L. and Hu, Jin: VLSI Physical Design: From Graph Partitioning to Timing Closure. Cham: Springer International Publishing, 2022. doi: 10.1007/978-3-030-96415-3.

[Ger99] Gerez, Sabih H.: Algorithms for VLSI design automation. Chichester and Weinheim: Wiley, 1999.

2.4 PVT Variation and Aging

As one of its primary features, the PARFAIT FPGA architecture enables local power adjustments. These can be used to tune the performance of the implemented circuit toward user application requirements: For example, the power and performance of paths which are not critical for timing can be reduced. However, in order to guarantee that paths still meet timing requirements, it is necessary to not only know the expected path delay, but also the actual delay in the manufactured circuit. Here it is important to notice that currently used models for STA do not represent the real path delay accurately, as they ignore various operating condition effects. Additionally, corner-based STA analysis for ASIC can conceive various combinations of operating conditions, balancing device yield and productivity loss by overly strict requirements. As will be shown here, FPGAs however generally have to use the worst-case corner, leading to pessimistic STA results.

In the following chapter, the four main operating conditions which affect circuit delays will be introduced:

- **Process Variation:** Due to manufacturing effects,
- **Voltage Variation:** Due to locally varying power supply,
- **Temperature Variation:** Due to locally varying temperature and
- **Aging:** Due to degradation of circuits over time.

It will be explained what causes those effects and how representative models for simulation can be derived from literature. In addition, it will be shown how existing models can be adapted for RFET devices. Following that, the influence of transistor-level effects on circuit and system level metrics will be explained and how modeling for those effects will be introduced. In later chapters, these models will be used to simulate the PARFAIT architecture with changing operating conditions.

Process Variation

Process variation describes the fact that transistor parameter values are not identical for all transistors on a manufactured IC, even if they have been designed with nominally same values. This variation of transistor parameters is caused by manufacturing and illustrated in figure 2.11. It mainly affects the threshold voltage V_{th} and effective channel length L_{eff}, but other parameters

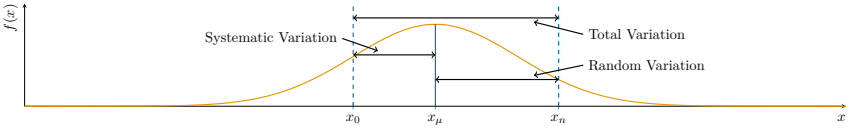


Figure 2.11: Process Variation affecting a transistor parameter x. The figure shows the probability density of a parameter relative to its nominal value x_0. x_μ shows the effective mean value including variation, x_n is one instance of a measured device. Adapted from [47].

are affected as well. This variation is usually assumed to be Gaussian and can then be split into an offset of the mean and into zero-mean Gaussian variation. To understand process variation in transistor manufacturing, it is illustrative to review the main processing steps introducing variations. For an overview of Very Large Scale Integration (VLSI) device manufacturing, please refer to literature [48, 49, 50].

Process Variation Sources: In device manufacturing, photolithography is used to project patterns from masks onto the wafer. The process consists of multiple steps, including deposition of a resist film, mask alignment, exposure of the resist, development and baking. As structure sizes have decreased, various complex forms of photolithography have been introduced. Photolithography introduces process variation mainly through two effects: Line-Edge Roughness (LER) is a variation of the channel length along the width, or the width along the length. It is caused by optical effects in during exposure and mainly affects the V_{th} of devices. The second effect is the Optical Proximity Effect (OPE): Because of diffraction effects, exposing a structure is actually dependent on neighboring elements. OPE correction has been developed, but can not completely counter the effect. OPE mostly leads to width variation and affects a device's V_{th}. In addition, mask alignment issues during lithography also affect device width and V_{th}.

Etching, which removes unneeded structures according to the mask, is another step introducing process variation. Depending on the method used (wet / dry / plasma / sputtering), there can be over- or underetching. These effects depend on the mask layout and therefore introduce variation. Doping is also a main contributor to process variation. It is used to insert dopants into a grid to form semiconductors. Commonly used methods for doping include ion implantation and diffusion. The process is prone to Random Dopant Fluctuation (RDF), causing locally varying doping concentrations, which mainly leads to varying V_{th}.

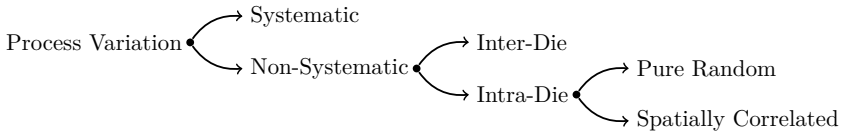


Figure 2.12: Process Variation taxonomy according to [51]. Adapted from [52].

Deposition of materials is one more step which can cause process variation. As the threshold voltage depends on the gate oxide thickness, layer deposition of the gate oxide can cause variation. Methods commonly used for deposition are Chemical Vapor Deposition (CVD), Physical Vapor Deposition (PVD) and atomic layer deposition. One more processing step introducing variation is planarization of surfaces. As the wafer has to be as planar as possible for various production steps, it is planarized repeatedly. Whereas there are various etchback techniques, Chemical Mechanical Polishing (CMP) is the most commonly use planarization method. It is prone to two effects causing variation: Dishing happens when the wafer is over-polished and too much metal is removed between dielectric parts, as it is softer than those. Erosion is similar, but includes both dielectric and metal loss. Again it is more marked in locations where there is lots of metal compared to dielectric, i.e. in locations with dense metal wiring. Both effects reduce the metal layer width, increasing resistance in the interconnect.

Process Variation Taxonomy: According to [51, 52], process variation effects can be classified into the categories shown in figure 2.12. *Systematic* or deterministic process variation is caused by deterministic effects, which affect all produced devices similarly. These manufacturing effects are identical across wafers, but could for example depend on the circuit layout instead. The most common cause of such effects occurs in photolithography, with optical proximity effects being the main contributor to variation. As these effects are systematic, they can be compensated at least partially.

Apart from systematic effects, all other effects can be categorized as *Non-Systematic* or random effects [52]. As these variations are arbitrary, modeling generally uses statistical approaches. Non-systematic effects can then be further separated into *Inter-Die* or global variation, and *Intra-Die* or local variation. Inter-Die Variation affects transistors on each single die in the same way, but does produce variations across different dies. Such effects are generally introduced by manufacturing variations which concern at least the whole die in the same way. Here die-to-die variations can occur due to

mask alignment issues, wafer-to-wafer variations due to wafer positioning, lot-to-lot variations because of changes in equipment or source material and fab-to-fab variations due to differences in different fabs. Inter-Die variation is usually countered by binning devices [47]: Device performance is measured and devices are classified into different speed grades according to their performance. As Inter-Die variation affects all transistors and therefore logic paths on a wafer similarly, there is little use in trying to locally compensate effects on the produced chips. Inter-Die variation can be modeled using a statistical distribution of affected device parameters, in the simplest case assuming a Gaussian distribution. As these effects do not have spatial correlation, all transistors can be modelled in the same way.

Intra-Die variation on the other hand can be differentiated into pure random variation and spatially correlated variation. Most notably, these changes can affect the feature size of different transistors on one die, leading to varying electrical characteristics for those. In currently used, modern technologies, most variation within produced devices is intra-die variation. Pure random intra-die variation represents effects which do not have any spatial correlation, i.e. affect nearby transistors in the same way as transistors which are at a larger distance.

According to [52], these effects are mainly caused by random-dopant fluctuation and line edge roughness: Random-dopant fluctuation is less of an issue for RFET technology, as it does not use any doping steps. Line edge roughness however affects RFET technology as well, as photolithography is used for RFET manufacturing. Spatially correlated variation is location dependent and describes an effect of the correlation between transistor characteristics depending on their relative distance, with larger correlation for transistors more closely located on the die. Intuitively, this means that transistors close to each other on a die are more likely to have "similar" characteristics. Various manufacturing steps can introduce such effects, most notably photolithography, etching and chemical-mechanical publishing. All of those can change gradually along the die. Intra-die variation affects mostly the channel length, width and oxide thickness of transistors.

Process Variation Modeling: This thesis will mainly address intra-die variation, but inter-die effects can be added using an additional Gaussian parameter in the variation model. In that case, mean and standard deviation of the inter-die variation distribution has to be known. The most commonly used way to model process variation of some parameter X in literature, is to consider it as a random variable. Then a linear combination of individual

components can be used to describe the final random variable for the total variation [52, 53, 54, 55]:

$$\begin{aligned} X &= X_0 + \Delta X_{\text{D2D}} + \Delta X_{\text{WID}} \\ &= X_0 + \Delta X_{\text{D2D}} + \Delta X_{\text{WID,c}} + \Delta X_{\text{WID,r}} \end{aligned} \tag{2.16}$$

In equation (2.16), X_0 represents the nominal value of the parameter, ΔX_{D2D} inter-die variation and ΔX_{WID} intra-die variation with correlated part $\Delta X_{\text{WID,c}}$ and pure random part $\Delta X_{\text{WID,r}}$. Assuming X to be normally distributed and considering ΔX_{D2D}, $\Delta X_{\text{WID,c}}$ and $\Delta X_{\text{WID,r}}$ as normally distributed and independent, mean μ_X and variance σ_X^2 of X are given as:

$$\mu_X = X_0 \tag{2.17}$$

$$\sigma_X^2 = \sigma_{X_{\text{D2D}}}^2 + \sigma_{X_{\text{WID,c}}}^2 + \sigma_{X_{\text{WID,r}}}^2 \tag{2.18}$$

Which of the parts contributes to the parameter variation X differs for each parameter, depending on how it is affected by the variation sources [52]. Equations (2.17) and (2.18) don't necessarily apply for some recent models which use non-normally distributed variables.

VARIUS Model: Models for process variation usually need technology dependent parameters. As those are not always available from fabs, this can make adaption of those models difficult. The thesis therefore first introduces the VARIUS process variation model from [53], as it is a simple model requiring few parameters and provides exemplary parameter values which match the empirical study in [56]. It can easily be adjusted for device data from other studies such as [57, 58], or be tailored to a custom device characterization. The model is physically motivated, i.e. derived from transistor equations, and considers two main factors in process variation: Threshold voltage V_{th} and the effective gate length L_{eff}.

VARIUS describes $\Delta X_{\text{WID,r}}$ and $\Delta X_{\text{WID,c}}$ in equation (2.16) as independent normal distributions. It models spatial correlation between two points on the chip with the correlation

$$corr(X_{\vec{x}}, X_{\vec{y}}) = \rho(r) \qquad r = |\vec{x} - \vec{y}| \tag{2.19}$$

and a spherical model for the correlation function $\rho(r)$:

$$\rho = \begin{cases} 1 - \frac{3r}{2\Phi} + \frac{r^3}{2\Phi^3} & (r \leq \Phi) \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

It notes that the spatially correlated component of L_{eff} is correlated to V_{th} and derives it accordingly:

$$L_{\text{eff}} = L_{\text{eff}}^0 \left(1 + \frac{V_{\text{th}} - V_{\text{th}}^0}{2V_{\text{th}}^0}\right) \tag{2.21}$$

The publication also offers parameter values $\Phi = 0.5$, $\sigma/\mu = 0.063$ for random and systemic V_{th}, and $\sigma/\mu = 0.032$ for random L_{eff}. Those can be used to reproduce the results for the transistor technology of [56].

Scarpato Model: In [3], Scarpato presents a complete approach to derive parameters from fab data for his propagation delay model. Compared to the VARIUS model, the model directly represents propagation delay instead of transistor parameters. It provides a complete model not only for process variation, but also for voltage and temperature variation as well as aging. Parameter values for this model are obtained through SPICE simulation of the intended target technology. As test circuit for the simulation, a representative path is selected from the target application [3]. To model process variation, the Scarpato model first characterizes the technologies voltage and temperature dependencies. First, various voltage and temperature values are simulated. Then the parameters in the following equation are fitted to the SPICE simulation results:

$$t_{\text{pd}}(V,T) = p_\beta + (C_1 + k_1 T^{n_1}) \frac{V}{(V - (C_2 - k_2 T^{n_2}))^{p_\alpha}} \tag{2.22}$$

To model the process dependency, the simulation is repeated for various corners. To be able to get results for process variation between corners, some sort of interpolation is necessary. Scarpato therefore introduces parameter shifts depending on the process corner. Here he notes that there seems to be no meaningful dependency when all the parameters are varied between corners. Therefore, the model uses constant values for all parameters except for C_2 and p_α. Those parameters can then be considered to be a linear function between best and worst values:

$$C_2(P) = C_{2,FF} + m_{c_2} \cdot P \tag{2.23}$$

$$p_\alpha(P) = p_{\alpha,FF} + m_{p_\alpha} \cdot P \tag{2.24}$$

With P being a normalized process parameter from fastest (0) to slowest (1), $C_{2,FF}/p_{\alpha,FF}$ the base values at the fastest corner and m_{c_2}/m_{p_α} the slopes of the linear functions. Inserting into equation (2.22) yields:

$$t_{\text{pd}}(P,V,T) = p_\beta + (C_1 + k_1 T^{n_1}) \frac{V}{(V - (C_2(P) - k_2 T^{n_2}))^{p_\alpha(P)}} \tag{2.25}$$

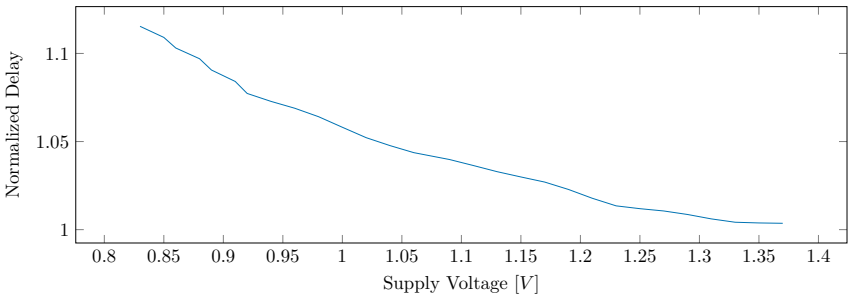


Figure 2.13: Delay increase due to supply voltage drop. The figure shows the effect of a 5 mV drop depending on the supply voltage. Adapted from [3].

The Scarpato model thereby provides delay for the estimated circuit at one operating point described by the (P,V,T) tuple. It does not model local variation directly, but can be used to obtain locally varying delays. When the local variation of the parameters $f : (x,y) \mapsto (P,V,T)$ has been modelled, the Scarpato model can be used to obtain the local delay.

Other Models: Various other models have been proposed in literature. Simple, grid based models have been described as early as in [59]. As process variation grows increasingly complex, the amount of parameters can be expensive for simulation times. Because of this, [60] proposes a model to reduce the amount of modelled parameters, which is especially important for Monte-Carlo simulations. Other models enable description of manufacturing parameters as non-gaussian and correlated parameters [61, 62]. An overview of commercially used models is given in [63, 64]. Other recent publications have focused on describing the variation as a skewed distribution instead of normally distributed [65]. The methods shown in this thesis can be used for all of those models. In order to simulate all four PVTA parameters, a model encompassing all of those is necessary. As the mentioned models focus only on process variation, this thesis' evaluations will be based on a Scarpato model variant extended for local variation.

Voltage Variation

As demonstrated in figure 2.13, changes in the supply voltage also cause changes in circuit delay. Supply voltage variation can be classified in two categories: Steady-state effects and dynamic effects [3]. Steady-state effects

are caused by IR-drop, the voltage drop caused by the parasitic resistance R of the power network when a current I flows [66]. Varying voltage drops can therefore be caused by both variation in resistance R and variation in I. Local variation of the resistance can be caused by process variation. In addition, it can also be systematically caused by unbalanced power network design. Changes in current can be caused by both density of transistors within a certain area and switching activity. Varying switching activity through clock gating or frequency scaling results in varying local power usage and IR-drop [67]. Switching activity is furthermore data-dependent, but for analysis of steady-state effects, considering the mean switching activity is sufficient. Whereas the transistor density can be taken into account when designing the power network, switching activity can not be compensated easily, as it may not be constant over time. This is especially the case in FPGAs designs, where both the final switching activity and density of actively used transistors depends on the application bitstream. Both effects can therefore not be estimated until the chip programmed in the field, making power network based compensation largely impossible. Dynamic effects are mostly caused by di/dt noise [66] and can cause supply voltage variations of up to 10 % [3]. They can be classified according to duration of the drops they cause [68]: The deepest drops are usually short and in the range of nanoseconds. They are caused by package inductance and capacity of the power distribution network. Effects in the range of hundreds of nanoseconds and slower effects in the range of microseconds usually have external cause. They can be mitigated by better package decoupling on the Printed Circuit Board (PCB) integrating the IC.

Effects on FPGAs: For FPGAs, voltage variation is an especially difficult issue to address. In ASICs, the power supply network can be adapted to the application as it is manufactured together with the circuit. In FPGAs, the power grid has to be pre-defined at the manufacturing time of the FPGA IC. The application is known only much later however, when programming the bitstream. It is therefore not readily possible to adapt the grid to the transistor density and switching activity of the target application. Because of that, the power network in FPGAs needs to be designed defensively. In addition, when mapping the target application and calculating timing slacks, certain safety margins have to be included. Voltage drops are the reason for the largest part of the timing margins and cause overly pessimistic margins in general [69]. Because it is difficult to address this issue in the FPGA architecture, publications such as [69] instead provide advice for FPGA application designers. For example, careful floor planning of the user application can reduce the issue. The analysis in [69] also shows that out of Process-, Voltage-, Temperature-

Variation (PVT), dynamic voltage variation contributes most to increase of path delay, followed by process variation, temperature variation and static voltage variation at last.

Voltage Variation Modeling: Influence on the circuit timing can be estimated with a *PVT* model such as equation (2.25) or the alpha-power law [66]:

$$t_{\mathrm{pd}}(V+\Delta V) \propto \frac{V+\Delta V}{(V+\Delta V-V_{\mathrm{th}})^{\alpha}} \tag{2.26}$$

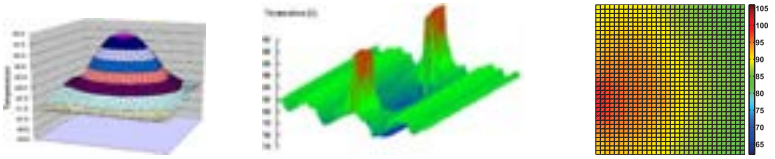
Here it can be seen that the closer V and V_{th}, the larger the influence of ΔV [3]. This effect can also be seen in figure 2.13 on page 33, where a voltage drop of 5 mV has no effect at a supply voltage of 1.4 V, but causes 10 % additional delay at 0.86 V. With such a model, a model describing the voltage variation on the IC, $f:\ (x,y) \mapsto (V)$, is needed. Modeling of such power variation differs for steady-state and dynamic variation. For ASICs, steady-state variation is usually analyzed using deterministic methods. If the application design and switching frequencies are known, Computer Aided Design (CAD) tools support calculation and analysis of *IR* drop in power networks. Similarly, for FPGAs power drop effects could be estimated if details about FPGA power network, the application bitstream and switching activity are known. As this drop can be analyzed deterministically for applications, there are no ready to use models which represent a "typical" application.

Research on dynamic voltage drops, especially on FPGAs, on the other hand largely focuses on security issues. Voltage drops can leak data through side-channels or could be used for Denial Of Service (DOS) attacks in cloud FPGA hosts [70]. Models for these effects have so far only been provided on transistor level [71]. It has also been shown that power drop effects are usually spatially correlated over the IC [72], as voltage drops effectively occur in the power network. They therefore always affect multiple devices connected to that local part of the network. The author of this work is not aware of any system-level model combining these aspects for representable applications on FPGA. The example evaluations in this work will therefore only consider static effects, but the designed simulation framework will be adaptable to dynamic models as well.

Temperature Variation

Temperature variation can be a large issue in modern VLSI ICs. Early works have demonstrated that there can be local hotspots in CPUs of up to 120 °C

[67]. Furthermore, it has been shown that both transistor device and interconnect performance are affected by temperature effects [67]. Whereas this effect is an issue for classic circuits as well, energy saving Near-Threshold Voltage Operation (NTVO) is affected in a special way: In this situation, on-chip temperature is often close to ambient temperature. Therefore, fluctuations in ambient temperature have immediate effect on circuit performance [73].



(a) Soft Logic [74] (b) Hard Blocks [75] (c) Simulation [76]

Figure 2.14: Temperature variation on FPGAs as reported by previous works in measurements and simulation models. **(a)** Temperature profile for an application using only programmable soft logic. **(b)** Temperature profile for an application using hard blocks on a platform FPGA. **(c)** Temperature profile as obtained using a simulation model.

Whereas in ASICs a temperature profile can be devised by CAD tools before manufacturing the chip, in FPGA such an analysis can only be performed when implementing the user application. Mitigation techniques which improve cooling of the chip locally, can therefore not be applied. Furthermore, recent work shows that heating can also be excessive on FPGAs. Specially designed FPGA applications were able to heat commercial FPGAs to temperatures of 50 °C to 120 °C [77]. Standard FPGA designs also cause temperature variation, especially in platform FPGAs (figures 2.14a and 2.14b). In those, especially hard blocks can cause local heating [75]. An exemplary analysis of CPU circuits implemented on FPGAs has further found temperature hotspots caused by cache and memory interfaces [78]. Other works have inserted temperature sensors into certain FPGA applications to verify predictions of hotspot locations or monitor those at runtime [74].

Placement Strategies: One way to counter these hotspot effects in FPGA is using a thermal aware placement strategy [75]. In those works, the thermal effects are usually estimated using a thermal simulator and the placement is then modified accordingly. Apart from larger modifications of placement, even small changes such as swapping the inputs of LUTs can reduce the thermal hotspots [79]. Lu et al. provided a slightly different approach where the

initial temperature profile is not obtained through simulation, but through measurements [80]. They first perform a normal placement, but embed temperature sensors. The user application design then needs to be executed on the FPGA, so the embedded sensors can be used to obtain the temperature profile. The proposed algorithm then perform a new placement for the application, taking the temperature profile into account. Overall, this resulted in a 13.9 % increase of uniformity in temperature distribution over the FPGA.

Modeling Approaches: To evaluate the effectiveness of all those mitigations for temperature variation in simulation, a model for this variation is needed. Temperature variation is usually modelled deterministically, based on the concrete circuit. Whereas CAD tools for ASICs integrate such temperature modeling, for FPGAs usually only an overall temperature for the whole chip is estimated by vendor tools. Academia has therefore focused on deriving custom workflows to generate temperature profiles. Equations (2.27) and (2.28) demonstrate that the chip temperature is related to leakage power and dynamic power [76]:

$$P_{\text{leak}} = P_0 e^{\frac{-k}{T}} \tag{2.27}$$

$$T = T_A \Theta (P_{\text{leak}} + P_{\text{d}}) \tag{2.28}$$

It is therefore necessary to derive dynamic and leakage power to estimate temperature profiles. Whereas dynamic power can be obtained from switching activity, leakage power is often not reported by FPGA vendor tools. In [76], the authors therefore propose an iterative approach to derive the leakage power. Once those components are known, the temperature distribution can be obtained using thermal simulators such as HotSpot or ISAC. An example for such a simulated heatmap is shown in figure 2.14c. Validating their measurements using thermal cameras, [76] shows that this strategy can estimate temperature with a precision of up to 1 °C. The author of this dissertation is not aware of any application-independent, generic, temperature model. All models focus on the behavior of a given application, but there are none which predict the behavior of an exemplary application. As such, temperature modelling for the purpose of this thesis needs to be performed using the modelling approach presented above: The dynamic and leakage power need to be derived, and the temperature distribution can then be obtained using thermal modeling for each specific application.

Effect on Circuit Delay: The effect of varying temperature depends largely on the actual transistor technology used [81]. Previous studies found transistor

drive current to decrease by 4 % and interconnect delay to increase by 5 % for a temperature increase of 10 K [67, 81]. Various parameters in the transistor equation are temperature dependent, including the channel mobility and charge, as well as the gate overdrive voltage through the dependency of the threshold voltage. For FinFET and other SOI devices, reduced short channel effects and self heating effects might have additional influence [81]. For bulk MOSFET, temperature effects can be physically understood using the BSIM3 and BSIM4 transistor models. These models contain various parameters with temperature dependencies. The delay variation characteristics change, depending on which parameter is dominant in each specific case [82]. As an example, reduced supply voltage can cause an inversion of the temperature dependency: In older technology nodes, carrier mobility variation dominated, increasing circuit delay with rising temperature. In 45 nm technology, gate overdrive becomes the dominant parameter and circuit delay now decreases with rising temperature [83].

Because of this technology dependent behavior, there are no physically based models for system level delay estimation. Simulation is commonly based on the device level SPICE models, which is accurate yet computationally infeasible for large circuits. Alternatively, mathematical models that abstract from the physical device behavior and use parameter fitting can be used. This is the approach used in the Scarpato model, fitting parameters of a mathematically motivated model to SPICE simulations. Here the dependency of the temperature variation on the dominant physical parameter is hidden in the fitted model parameters. This simplifies the equations and allowing for efficient computation using a generic model, adapting to various technologies. The resulting model is only valid as long as the temperature and other parameters stay within the limits used in the fitting process [3].

Aging

Apart from process variation, which affects circuit performance independently of the environment conditions during operation, and environment conditions such as voltage and temperature, another aspect to consider in circuit performance is IC aging. IC aging can be caused by Front End of Line (FEOL) effects, i.e. by transistors, by Back End of Line (BEOL) effects, i.e. metal layers, and on package level and back end [84]. Aging effects can cause gradual degradation in performance or complete breakdown of functionality. Effects can also be temporary, allowing recovery under certain conditions,

or permanent. For the analysis in this thesis, both permanent and temporary gradual degradation effects will be addressed. Effects causing complete breakdown of functionality will not be considered.

FEOL effects: There are three main effects belonging to the FEOL category: Hot Carrier Injection (HCI) is an effect where charges get trapped in the dielectric of the transistor. It is caused by current flow in the transistor channel, letting "hot" carriers enter the dielectric. Whereas it also affects carrier mobility and current flow, its main result is a permanent change in V_{th} [3]. Signal activity is the main factor affecting HCI severity. It is also dependent on the supply voltage of the transistors, but mostly independent of the temperature [85, 86]. It is generally most relevant for analog circuits and less of an issue in digital designs [87].

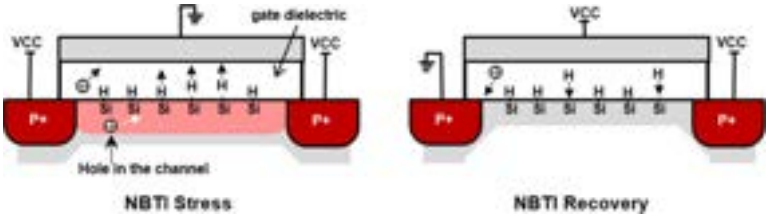


Figure 2.15: Negative Bias Temperature Instability (NBTI) stress and recovery in PMOS transistors, taken from [87]. Left: Charge carriers entering dielectric and breaking Si-H bonds under stress (negative V_{th}). Right: Charge carriers leaving dielectric and Si-H bonds restoring under recovery conditions (zero V_{th}).

The second effect in FEOL is Bias Temperature Instability (BTI) and its variants NBTI and Positive Bias Temperature Instability (PBTI). NBTI as shown in figure 2.15 occurs in PMOS devices and is the more common of the two effects. Like HCI, it is an effect where charge carriers enter the dielectric and causes an increase in V_{th}. Unlike HCI however, the cause for BTI is the electric field perpendicular to the channel. The effect is therefore mainly caused by the voltage applied to the gate and the severity of BTI depends on the signal switching probability [3]. Some of BTI is reversible: Charge carriers entering the dielectric can leave it again if reduced gate voltage is applied. BTI can however also lead to permanent effects, e.g. through broken Si-H bonds [3]. The effect is highly temperature dependent [88, 89] and is the dominant aging effect in digital circuits [87].

The third common aging effect in FEOL is Time Dependent Dielectric Break-

down (TDDB). TDDB is a mostly permanent and often destructive effect. It causes a conductive path in the dielectric, which leads to increased leakage currents. When the hard breakdown occurs, a conducting path is created from gate to substrate, making the transistor unusable [3]. As destructive effects won't be addressed in this thesis, TDDB effects will not be examined further.

BEOL effects: BEOL aging effects are those effects, which are caused in metal layers. The most prominent failure mechanism for metal layer based interconnects is Electromigration (EM). EM is caused by the electric field enabling the current flow in a conductor. This field causes atoms in the conductor to slowly migrate, as shown in figure 2.16. As a result, wires can develop voids, causing increasing resistance on the wire. This effect is not necessarily destructive. In addition, migrated atoms accumulate on certain spots, forming hillocks. Hillocks can cause short-circuits between neighboring wires, leading to permanent defects [90].

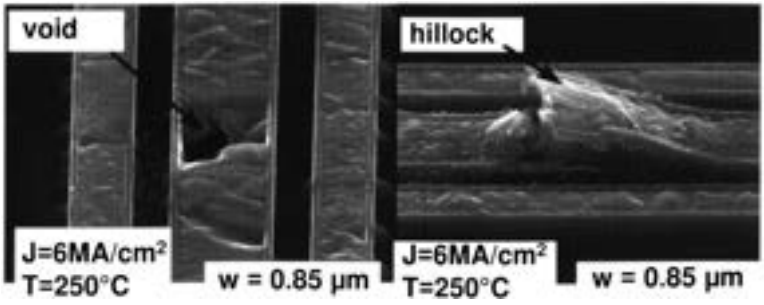


Figure 2.16: EM effects observed using a scanning electron microscope. Picture taken from [91]. Left: A void in the metal wire connection, increasing connection resistance. Right: A hillock in the wire connection, potentially causing short circuits.

Metal layers are further affected by mechanical stress, e.g. caused by thermal cycling [3]. More aging effects concern the packaging of the IC. As those are usually destructive effects, they are not further elaborated here.

Aging in FPGAs: For FPGAs as primarily digital circuits, BTI is the most critical aging cause. When considering both BTI and HCI, it should be noted that treating them as completely independent effects can result in pessimistic estimations [3]. When modelling BTI, it should also be considered that it varies

locally. This local variation is however largely independent of the process variation [3]. As BTI causes increased V_{th} and therefore increased delay in circuits [87], it will also affect FPGA logic. Due to the FPGA's late configuration, it is again not possible to consider application specific countermeasures during circuit design. Solutions therefore could use adding safety margins during timing analysis of FPGA applications. Local variation and dependency on switching activity will however lead to pessimistic results. Alternatively, aging monitoring can detect delay degradation. Such systems can also handle local degradation, if they can counter effects using some mechanism to locally increase performance.

Modeling Approaches: Models for the aging processes have been proposed at various layers of abstraction. At the lowest abstraction, physical models can be used to enhance SPICE simulations with aging information. Until recently, fabs have provided EDA specific simulation models and libraries for their respective technologies [92]. More recently, the Silicon Integration Initiative's Compact Model Coalition has standardized an Application Programming Interface (API) to develop heating and aging models [84, 93]. These models are then used in combination with the standard SPICE transistor models, such as BSIM. Factory provided models for aging can be both empirical or physical models, trading of performance and accuracy. For physical models, reaction-diffusion [94] and trapping-detrapping [95] models have been proposed. A summary of these models and the corresponding equations are given by Khoshavi et al. and can be found in [87].

To provide faster models for simulation, circuit level predictive aging models have been proposed [87]. For example, long-term HCI and BTI effects in circuits can be modeled using these equations [96, 97]:

$$\Delta V_{\text{th\_BTI}} = a(TSP \cdot t)^n \tag{2.29}$$

$$\Delta V_{\text{th\_HCI}} = b(TSwP \cdot t)^m \tag{2.30}$$

In these equations, a and b are technology and temperature specific constants, TSP is the transistor stress probability, $TSwP$ is the transistor switching stress probability, t is time, and n and m are time exponential constants. As the switching stress probabilities are application dependent, a work load profile for the application has to be provided [98]. For the different effects, both the signal probability (time a signal assumes a certain value) and transition density has to be known. In addition, these models require temperature and supply voltage profiles as inputs [99]. Furthermore, stochastic models can take non-deterministic effects into account [100]. For some effects, it is however

important that process variation and aging are not modelled independently, as this could yield to pessimistic results [87].

Another kind of aging models on a yet higher level of abstraction are architecture models [87]. For example, [101] proposed to select an "representative" transistor for a certain area and use it to estimate the values of all transistors in this area. [102] simulates a certain workload pattern and derives voltage and temperature profiles. It then uses this information to derive gate delays. [103] extends all these considerations to instantaneous BTI effects. Application specific models are of limited use in FPGAs, as the application is not known during FPGA design time. As the FPGA transistor sizes are predetermined, these models can be used for analysis, but are of limited use for mitigation.

Scarpato Model: One variant of a circuit level aging model is the one proposed in [3]. As an extension of the PVT model presented in the same work, it is primarily an empirical, mathematical model: Parameters in the model equation are fitted to results of SPICE simulations. This approach therefore requires existing aging models that can be used in a SPICE simulator. The main benefit of the Scarpato model is its reduced computation complexity when compared to the SPICE model. The Scarpato PVT model presented earlier in this section was therefore analyzed to determine which parameters change with BTI and HCI effects. EM effects where not considered in this model, but extending it accordingly is possible. To reduce the model complexity, aging variation has been reduced to an aging-dependent shift of a single parameter in the model.

$$t_{\mathrm{pd}}(V,T) = p_\beta + p_{\mu^{-1}}(T)\left(\frac{V}{\left(V - p_{V_{\mathrm{th}}}(T)\right)^{p_\alpha}}\right) \tag{2.31}$$

$$t_{\mathrm{pd}}(V,T,t) = p_\beta + p_{\mu^{-1}}(T)\left(\frac{V}{\left(V - \left(p_{V_{\mathrm{th}}}(T) + \Delta p_{V_{\mathrm{th}}}(V,T,t)\right)\right)^{p_\alpha}}\right) \tag{2.32}$$

$$\Delta p_{V_{\mathrm{th}}}(T) \propto e^{\frac{-E_\alpha}{kT}} \tag{2.33}$$

$$\Delta p_{V_{\mathrm{th}}}(V,T) = A \cdot V^\gamma \cdot e^{\frac{-E_\alpha}{kT}} \tag{2.34}$$

Revisiting equation (2.31), a short-hand version of equation (2.22), $p_{V_{\mathrm{th}}}$ was identified as the parameter affected most by aging shift. Introducing the new $\Delta p_{V_{\mathrm{th}}}$ parameter into this equation yields the aging-aware model in equation (2.32). Scarpato then continues to describe the temperature and supply voltage dependencies. Various alternatives are examined for

the voltage dependency, but ultimately equation (2.33) is used in the model.

To introduce the time dependency of the aging parameter, Scarpato introduces trapping-detrapping and reaction-diffusion models into equation (2.34). He then evaluates which variant of the equation yields a better fit of the SPICE simulations. Ultimately, the final model chosen for $\Delta p_{V_{\text{th}}}$ is given as:

$$\Delta p_{V_{\text{th}}}(V,T,t) = \left(C_1 t^{n_1+a_1 log(V)} + C_2 t^{n_2+a_2 log(V)}\right) \cdot V^{\gamma} \cdot e^{\frac{-E_\alpha}{kT}} \tag{2.35}$$

Equation (2.31) does not include process variation effects, but Scarpato showed that process variation is uncorrelated to aging. The parameter can therefore be determined once and be reused for different corners in equation (2.22) to include process variation.

As mentioned in the introduction of the physical aging effects, workload described by switching frequency and signal probability also affects the aging process. As it was found that workload can not be simply reduced to a fixed number of parameters, the $\Delta p_{V_{\text{th}}}$ parameter is instead being estimated for different workloads using different SPICE simulations. The model can also include dynamic variation, i.e. changing supply voltage, temperature and workload over time. Refer to [3, p. 74] for details.

Further Reading

Various textbooks cover the VLSI IC fabrication process in detail. For example, a recent take can be found in [Gen17]. For process variation, [Cha18] provides a detailed introduction from a circuit perspective. After introducing classical STA and explaining the need for statistical analysis, it introduces mathematical foundations needed for the modeling. It further includes an overview of process variation sources and a classification of those. It also extends the analysis to gate and path delays for CMOS circuits. Additional information about process variation and modelling can be found in [Chi07, Die12, Huf20]. A detailed introduction to voltage variation can be found in [Wir13] and an introduction to aging, including sources, models and optimization techniques, is provided by [Tan19]. Additionally, [Ye21] presents methods to deal with aging effects. The PVTA simulation model by Scarpati, which is primarily used in this thesis to simulate the developed FPGA architecture, is described in detail in his dissertation [Alt17].

[Gen17] Geng, Hwaiyu: Semiconductor Manufacturing Handbook, Second Edition. 2nd edition. New York, N.Y.: McGraw-Hill Education and McGraw Hill, 2017.

[Cha18] Champac, Victor and Garcia Gervacio, Jose: Timing Performance of Nanometer Digital Circuits Under Process Variations. Vol. 39. Cham: Springer International Publishing, 2018. doi: 10.1007/978-3-319-75465-9.

[Chi07] Chiang, Charles C. and Kawa, Jamil: Design for manufacturability and yield for nano-scale CMOS. Series on integrated circuits and systems. Dordrecht: Springer, 2007.

[Die12] Dietrich, Manfred and Haase, Joachim: Process Variations and Probabilistic Integrated Circuit Design. New York, NY: Springer New York, 2012. doi: 10.1007/978-1-4419-6621-6.

[Huf20] Huff, Michael: Process Variations in Microsystems Manufacturing. 1st ed. 2020. Microsystems and Nanosystems. Cham: Springer International Publishing and Imprint Springer, 2020. doi: 10.1007/978-3-030-40560-1.

[Wir13] Wirnshofer, Martin: Variation-aware adaptive voltage scaling for digital CMOS circuits. Vol. 41. Springer series in advanced microelectronics. Dordrecht: Springer, 2013. doi: 10.1007/978-94-007-6196-4.

[Tan19] Tan, Sheldon; Tahoori, Mehdi Baradaran; Kim, Taeyoung; Wang, Shengcheng; Sun, Zeyu and Kiamehr, Saman: Long-Term Reliability of Nanometer VLSI Systems: Modeling, Analysis and Optimization. Springer eBook Collection. Cham: Springer, 2019. doi: 10.1007/978-3-030-26172-6.

[Ye21] Ye, Wei; Alawieh, Mohamed Baker; Hsu, Che-Lun; Lin, Yibo and Pan, David Z.: "Dealing with Aging and Yield in Scaled Technologies". In: *Dependable Embedded Systems*. Ed. by Henkel, Jörg and Dutt, Nikil. Embedded Systems. Cham: Springer International Publishing, 2021, pp. 409–429. doi: 10.1007/978-3-030-52017-5_17.

[Alt17] Altieri Scarpato, Mauricio: "Digital circuit performance estimation under PVT and aging effects". Thesis. Université Grenoble Alpes, 2017. url: https://theses.hal.science/tel-01773745.

2.5 FPGA Logic Generators

The CMOS circuits introduced previously have their functionality determined and fixed at manufacturing time of the circuit. As manufacturing is a long and expensive process, this approach is not suitable for prototyping or if only a few devices are needed. A solution to this issue lies in field programmable devices, which have their ultimate functionality decided after manufacturing, "in the field". Over the last decades, various approaches have emerged to realize the main component of such systems, the programmable logic cell.

Logic Matrices The first commercially available devices that provided programmable cells have been Programmable Read Only Memorys (PROMs) [104]. Whereas they were mainly meant to be used as memory, connecting input signals to address lines and output signals to the data output of the memory realizes programmable logic. This approach works like an early implementation of a LUT, but is rather inefficient due to the need for a complete address decoder.

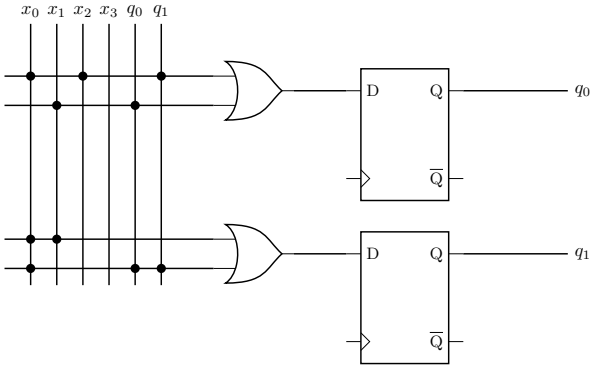


Figure 2.17: Programmable Array Logic (PAL) structure according to [104]. Like Programmable Logic Arrays (PLAs), PALs realize two level logic, but unlike those, they do not support configuration of the *OR* stage. As depicted in the left part of the picture, all configuration happens in the *AND* plane.

The next step in the evolution of programmable logic cells were PLAs. PLAs realize two-level logic, where the first level is implemented as a wired *AND* and the second level as a wired *OR*. Due to this structure, this circuits can directly realize the Sum-of-Products (SOP) form of an equation. PLAs were mainly

used to implement combinational logic and to replace multiple discrete gates on a PCB. PLAs had comparatively wide inputs in both configurations stages, causing the main drawback of early PLAs: Comparatively large delay caused by two levels of configurable cells and complex manufacturing[104]. To solve these issues, PALs as shown in figure 2.17 were introduced. Compared to PLAs, these systems kept the configurable *AND* plane, but fixed the functionality of the *OR* stage. In addition, PALs also introduced FFs on the outputs of the combinational logic cell. As those are also available as inputs for the *AND* matrix, these programmable logic circuits enable implementation of sequential circuits.

Later on, combinations of multiple PLAs or PALs were put in a single chip, connected using some kind of interconnect [104]. These chips, called Complex Programmable Logic Device (CPLD), enable realization of more complex sequential logic such as state machines. The basic element in all these mentioned logic circuits are programmable matrices, usually a programmable *AND* matrix. They have to support a wide number of inputs and are therefore usually implemented in specific ways, which are more efficient than naive CMOS implementation. As programmable matrix structures will not be employed in this thesis, the transistor level implementation details will not be discussed further.

AND-Inverter Logic Another possible realization of configurable logic has been introduced in 2012 by Parandeh-Afshar and Benbihi[105]: So-called AND-Inverter Cones (AICs) are configurable logic cells consisting of *AND* and *INVERTER* gates. Unlike PALs and PLAs, AICs do not realize the configurable logic through specific assignment of inputs to gates. Instead, connections are hardwired and reconfiguration is achieved by changing the logic function in the logic element. Figure 2.18 shows a simplified 3-level AIC implementation. Each node in the shown graph is an *AND* with an optional *INVERTER* and can therefore either realize an *AND* or a *NAND* function. Each cell provides multiple outputs, allowing to derive multiple outputs from one block. This feature may also be used to implement multiple functions in a single block, enabling a fracturable logic system.

AICs are inspired by modern synthesis tools, which represent circuits as graphs of *AND* and *INVERTER* nodes. Their area scales linearly with the number of inputs, which is a benefit compared to LUTs which scale exponentially. Furthermore, the delay scales logarithmically, whereas it scales linearly for LUTs. Because of those advantages, Parandeh-Afshar et al. introduced the logic cell in a hybrid FPGA, combining both LUTs and AICs. They achieved

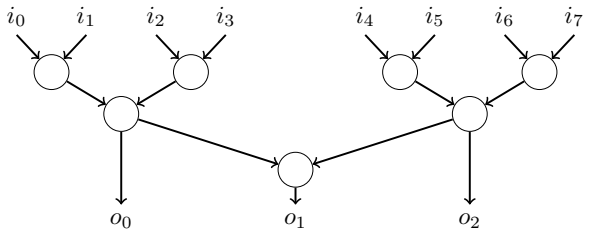


Figure 2.18: AIC-3 structure adapted from [105]. Each node represents a logic cell which can either realize an *AND* or a *NAND* function. Shown here is a simplified 3-stage design, where the AICs used in practice usually consist of 6 levels. Intermediate signals are also available as outputs to make the cell fracturable and enable reuse of intermediate values.

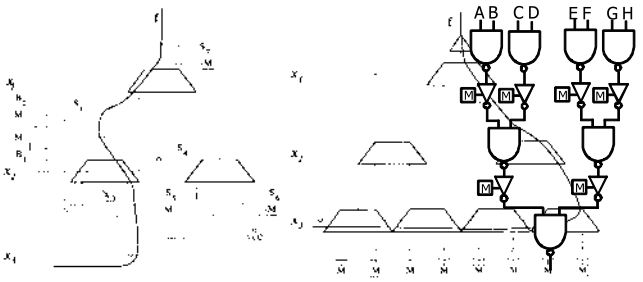
a 16 % decrease in area and up to 32 % decrease in delay compared to their LUT-only reference. [105]

AICs are less expressive than LUT, as they can not realize every possible function of their inputs. Because of that, numerous AICs needs to be used, which causes additional interconnect pressure. To avoid routing congestion in the interconnect, multiple AICs are combined within one cluster using internal loopback connections. In 2014, Zgheib et al. implemented the proposed AICs FPGA in 40 nm technology [106]. They found that because of the large number of inputs and outputs, a comparatively large crossbar is required. This crossbar requires almost twice as much area as the LUT system used for comparison. Furthermore, authors show that reduced delays compared to LUTs only were achieved for short critical paths.

In 2020, Thummler et al. proposed new, optimized mapping algorithms for AIC based programmable logic systems [107]. Their algorithm reduces area by up to 16.4 %, but the general issue with large crossbars persists to date. Because of this, authors have proposed alternative solutions with small, reconfigurable cells as summarized in [108]. The most common replacement cell design changes functions between *NOR* and *NAND*. Compared to LUTs or ULMs explained in the next section, the number of functions that can be realized by the cell are still small.

Universal Logic Modules Whereas AICs realize two logic functions in logic cells, ULMs take this idea one step further. Initial ULM research was carried out decades before the introduction of first FPGAs [109]. An early definition of the ULM was given in [110] for example: An $ULM.m$ is a function $U(z_0..z_n)$

that realizes all possible functions $f(x_0..x_m)$ by substitution of $z_0..z_n$ with any of $\{x_0..x_m, \overline{x_0}..\overline{x_m}, 0, 1\}$. Strictly speaking, a $ULM.m$ must therefore be able to implement all functions of m input variables. Later works however also refer st



(a) ULM.3 implementation [109] (b) 8-COGRE Logic Cell [111]

Figure 2.19: Exemplary ULM implementations proposed in literature. **(a)** ULM.3 designed by Ziic et al. using Binary Decision Diagrams (BDDs). **(b)** 8-input logic cell manually designed by Iida et al.

Figure 2.19a shows two examples which were especially designed for FPGA application. The adaption of ULMs for FPGA happened later, starting with Thurber et al. in 1999 [113]. This publication adapted the previous ULM work, considering that equivalent functions can be combined and inputs can be swapped in FPGAs. It then provided an algorithm to derive sets of $ULM.m$ and recommended implementations to the Actel multiplexer based cell. The main goal of that publication was to generate set of suitable functions.

Ziic et al. expanded from this work in 2006 [111]. Unlike Thakur et al., this work uses BDDs to design the ULMs. It also provides concrete 3 and 4 input example implementations, where the $ULM.3$ has been reproduced in figure 2.19a. This publication also first separated combinational and general purpose inputs. Configuration inputs are then driven by Static Random Access Memory (SRAM) or similar memory cells, reducing pressure on the global interconnect. Ziic et al. also note how the number of configuration bits for their 4-input cell was reduced to 13, but compared to 16 bit needed for a 4-input LUT. Some years later, new algorithms for ULM design were proposed to design cells with more inputs [113].

A set of cell implementations using even less area has been presented in [111], the COGRE cells. Strictly speaking, those cells are not ULMs, as they do not

cover all possible functions. Taking advantage of that design decision, the authors state that their 8-input cell uses 75.19 % less area and 68.27 % less configuration bits than an 8-input LUT. An implementation of the 8-input COGRE cell is shown in figure 2.19b. Even though there has been continued research on ULMs themselves, there are few publications and no commercial systems actually using them in FPGAs.

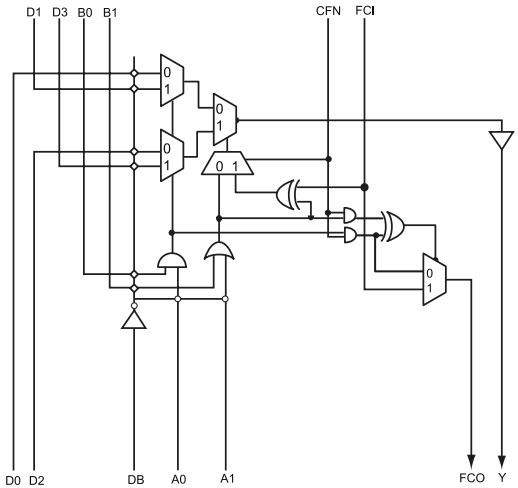


Figure 2.20: Microsemi Axcelerator C-Cell Logic Block [114], cited via [115]. This block realizes a Multiplexer (MUX) based Logic Element (LE), but combines the multiplexer with hard logic gates to achieve higher expressiveness.

Multiplexer Logic An alternative to ULMs are MUX-based logic cells. Compared to LUTs, MUX-based cells route input signals to both select and data inputs of multiplexers. LUTs are similarly based on multiplexers. As they route input signals to the select inputs of multiplexers only, data inputs in LUTs are however always constant. They are directly connected to memory, usually SRAM.

A two data input multiplexer can realize six functions, when it uses variables and constants as inputs. In general, MUX-based logic cells can therefore achieve large expressiveness with few transistors. This benefit however is diminished by the introduction of a large number of inputs, compared to LUT based cells. Additional inputs increase resource demand and complexity in the interconnect, making the use of small routing switches especially important. In practical realizations, MUX-based logic cells have therefore

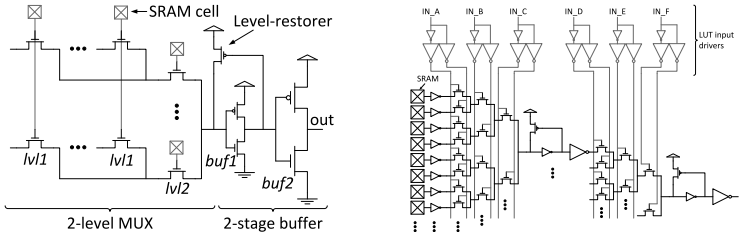
often been combined with flash or anti-fuse technology [116]. To realize cells with more inputs, MUXs are often combined with logic gates to yield more complex functions. Figure 2.20 shows an example of such a cell which was used in a commercial architecture released in the early 1990s, the Actel ACT3 [114]. In addition to the MUXs, it includes hard logic *AND* and *OR* gates to realize more functions. Another early example of MUXs cells are early FPGAs by Quicklogic [104]. Both early commercial variants are Antifuse based, making use of small switches to reduce the wiring overhead. MUX-based logic cells have gone out of fashion in recent FPGAs architectures, which use LUTs almost exclusively. This is often attributed to the comparatively complex tools needed to map user applications to such logic. For LUT-based FPGAs in comparison, the mapping step is reduced to a less-complex graph-covering problem [117].

In 2016, Chin et al. presented a more modern variant of a MUX-based logic cell. Their variant realizes a 6-input LE using a 4-input MUX. Using both the data and the select inputs of these multiplexers they obtain a LE with the same amount of inputs as in modern LUT-based LE. The authors' LE requires only 15 % of the area of a comparable 6-input LUT. When used in a hybrid architecture mixed with LUTs, the hybrid FPGA architecture still used 8 % less area than a LUT-only architecture. The logic cell can map all functions of 2 and 3 inputs and some functions of up to 6 inputs. For the logic mapping, authors explain in detail how the Shannon expansion can be used to fit logic functions to the LE.

Lookup Tables LUTs are the most common logic generator used in recent, commercial FPGAs [118, p.4ff]. An N-LUT combines a 2^N-to-1 MUX and a 2^N bit memory, usually SRAM. Figure 2.21b depicts the transistor level implementation of such a basic LUT. Memory outputs are connected to the data inputs of the multiplexer, whereas its select inputs are external inputs for the logic function's parameters. The figure also includes signal buffers in various parts of the LUT, similar to the buffered MUX shown in figure 2.21a. Depending on the requirements of output signal levels and the characteristics of transistors and technology, some buffers may not be required.

To realize a function $f(x_0..x_n)$ in a LUT, its truth table needs to be obtained and stored in the memory. Inputs $x_0..x_n$ are connected to the select inputs and choose the matching output value from the truth table for the given input combination. Like in MUX logic, the function table can again be obtained using the Shannon Composition $f(x_0, x_1..x_n) = x_0 \cdot f_0(x_1..x_n) \vee x_1 \cdot f_1(x_1..x_n)$. An important difference compared to MUX logic is that the composition

needs to be applied repeatedly until all data inputs of the multiplexers are constant.



(a) 2-Level Buffered MUX [119] (b) 6-LUT (Partial View) [119]

Figure 2.21: Exemplary MUX and LUT implementations proposed in literature. Implementations shown are models used in COFFEE, which automatically determines transistor widths for FPGA elements [119]. **(a)** A two-level multiplexer in pass-transistor logic configured by SRAM cells. The output contains a level restore buffer to compensate for the voltage drop across pass transistors. **(b)** An implementation of a LUT using pass-transistors. Internal level restore buffers are needed after three pass transistors to ensure signal integrity. Buffered multiplexer select inputs are driven externally, whereas buffered data inputs are driven by SRAM cells.

LUTs can describe all functions of N inputs and have therefore been called "computational heart" of the FPGA [118, p.4 ff.]. They allow for a higher logic density and large digital designs, causing a historical paradigm shift [104]: Large digital systems can now be realized in small quantities, avoiding the initial setup cost involved in ASIC manufacturing. The best size of a LUT, i.e. the number of select inputs, depends on the mapped user applications and has been extensively researched. Literature suggests 4-input LUTs, but commercial architectures commonly use 6-input LUT as well. Larger LUTs can realize more complex functions, but cause higher propagation delay [118, p. 4]. Whereas delay of LUTs grows linearly with the number of inputs, its area grows exponentially [116]. Because of this, useful sizes of LUTs are limited. An alternative way to realize large functions is to divide the function into multiple parts, which are then realized in multiple LUTs[118].

Recent commercial LUTs often include enhanced functionality, such as being useable as memory or shift registers [120, p. 37]. These extensions are usually realized on transistor level, allowing for an area and delay efficient implementation. Other extras which are often implemented on cell level to complement the LUT include dedicated signals for carry chains and external adders or *XOR* gates [121, p. 25].

Further Reading

Information about CPLDs, PLAs and PALs can be found in an early work of Brown et al. [Bro96]. This work also gives a quick summary of commercial architectures available in the late 1990s. In a more recent overview, Kuon et al. give an overview of commercial architectures available in 2007 [Kuo07]. [Yan14] can be used as a starting point for information on AICs. Information about LUTs can be found in most FPGA textbooks, as these are the most commonly used logic generators. Some examples include [Vas07] which summarizes various architecture studies, [DeH07], which includes a discussion of granularity, and [Ama18], which discusses performance trade-offs and LUT size. The 2020 textbook of Rodríguez-Andina provides an overview of commercial FPGA architectures and used logic generators in 2020 [Rod20]. When it comes to most recent logic generator recent research, [Rai21] provides a short overview. Apart from discussing AICs again, this work also introduces logic generators which make efficient use of various novel transistor technologies. Those generators will be discussed in detail in the next section.

[Bro96] Brown, S. and Rose, J.: "FPGA and CPLD architectures: a tutorial". In: *IEEE Design & Test of Computers* 13.2 (1996), pp. 42–57. DOI: 10.1109/54.500200.

[Kuo07] Kuon, Ian; Tessier, Russell and Rose, Jonathan: "FPGA Architecture: Survey and Challenges". In: *Foundations and Trends® in Electronic Design Automation* 2.2 (2007), pp. 135–253. DOI: 10.1561/1000000005.

[Yan14] Yang, Haigang; Zhang, Jia; Sun, Jiabin and Le Yu: "Review of advanced FPGA architectures and technologies". In: *Journal of Electronics (China)* 31.5 (2014), pp. 371–393. DOI: 10.1007/s11767-014-4090-x.

[Vas07] Vassiliadis, Stamatis, ed.: Fine- and coarse-grain reconfigurable computing. Dordrecht: Springer, 2007.

[DeH07] DeHon, André and Hauck, Scott: Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation. 1. Aufl. Systems on Silicon. s.l.: Elsevier professional, 2007.

[Ama18] Amano, Hideharu, ed.: Principles and structures of FPGAs. Singapore: Springer, 2018.

[Rod20] Rodríguez-Andina, Juan José; La Torre-Arnanz, Eduardo de and Valdés Peña, María Dolores: FPGAs: Fundamentals, advanced features, and applications in industrial electronics. First issued in paperback. Boca Raton: CRC Press, 2020.

[Rai21] Rai, Shubham; Nath, Pallab; Rupani, Ansh; Vishvakarma, Santosh Kumar and Kumar, Akash: "A Survey of FPGA Logic Cell Designs in the Light of Emerging Technologies". In: *IEEE Access* 9 (2021), pp. 91564–91574. DOI: 10.1109/ACCESS.2021.3092167.

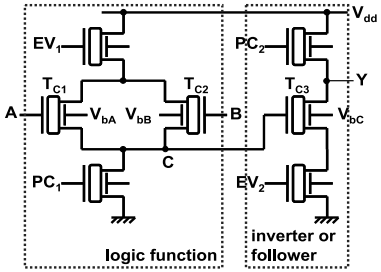
2.6 Ambipolar Reconfigurable Cells

The logic generators discussed so far have been realized mostly in traditional CMOS silicon technologies. The development of those logic generators has therefore been limited to designs which can be efficiently realized in that technology. Furthermore, those works focus on the design of efficient cells for FPGAs leveraging flexibility given by the technology. In recent years, a new type of logic generators has been proposed in literature: Logic cells which make use of – and are optimized for – ambipolar transistor technologies. As these technologies enable efficient implementation of reconfigurable FETs, the adaption in larger, reconfigurable cells seems natural. The realization of such cells however has to obey different technology constraints, leading to new challenges. Most notably, the cell implementations with low area usually also provide low flexibility or expressiveness [122]. Ambipolar reconfigurable cells have been realized in both dynamic and in static logic styles. In dynamic logic, the cell needs to be clocked to transition between various internal states over time. Usually, such additional states are used to pre-charge certain capacitances in the circuit. Apart from the need of a clocking system for the cells, another drawback of dynamic logic is reduced energy efficiency [123]. Static cells on the other hand do not need any clocking input and can therefore replace LUTs in common FPGA architectures more easily.

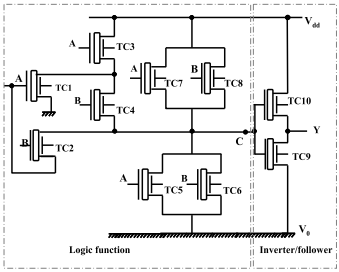
A main differentiating characteristic of ambipolar reconfigurable cells is the number of functions they can realize "in the field" [18] and the number of transistors used [124]. Here, Mikolajick et al. first classify cells according to their configuration inputs as "hard-wired" or "soft-wired". "Hard-wired" logic cells connect their configuration inputs to voltages statically, i.e. during manufacturing. Such devices do not offer any customization in the field [18]. They can however allow for more cost-efficient manufacturing of ASIC, as the functionality of the circuit can be changed through modification of only metal layers. The semiconductor layers can therefore be manufactured using the same masks for different applications. Furthermore, it is also possible to develop standard cells with completely fixed functionality for these ambipolar technologies [125].

More interesting for this thesis is the second group of "soft-wired" logic cells. These cells enable changing of functionality "in the field" and can therefore be used to realize FPGAs architectures. The most basic of those cells can realize only two different functions. They are usually implemented using complimentary pull-up and pull-down networks and switch between complimentary

functions [18]. In this category, NAND/NOR, AND/OR, XOR/XNOR and other cells have been proposed [26]. Other works in literature have also combined multiple of these cells using multiplexers, forming more expressive cells, e.g. a 6-function static-logic cell [126].



(a) DRLC_7T Cell [123, 127]



(b) CNT_SRC_10T Cell [123]

Figure 2.22: Reconfigurable cells based on ambipolar devices as proposed in literature. Circuits were originally realized using Dual Gate CNTFET (DGCNTFET) devices [123], but the structure has been adapted to other ambipolar technologies. **(a)** A 7-transistor cell realizing 14 functions of two inputs. This realization uses a dynamic logic approach. **(b)** A 10-transistor cell realizing all 16 functions of two inputs. The circuit has been realized as static logic.

More advanced cells can be grouped into almost ULMs, ULMs and novel LUTs classes. An example of an almost ULM is shown in figure 2.22a. The original version of this circuit was presented in [127] and described an 8-function cell. It was derived from [128], which is the first published "soft-wired" ambipolar reconfigurable cell. The circuit is implemented using DGCNTFET in dynamic logic. The three programming inputs *A*, *B* and *C* are programmed with positive and negative supply voltages. In [123], the concept was extended to use three values for the programming voltages. Using an additional zero level voltage allows turning off RFET devices, and enables realization of 14 functions in total. Jabeur et al. also proposed a similar static logic cell, shown in figure 2.22b. This cell is an example of a full ULM, as it can realize all possible 16 functions of 2 input signals. Whereas a static cell is easier to use, the main drawback of this specific cell is the large number of configuration inputs: A total of 9 configuration inputs with three possible values each require more configuration storage than the 4 bit needed for a two-input LUT. Because of that, later works such as [129] explicitly try to reduce the needed configuration storage: The cell proposed by Kato et al. reduces configuration input values back to only two voltages, but it is a dynamic logic design. Some designs of similar

cells, such as a 6-function 2-input cell or a 13-function 3-input cell, can be found in [122]. The 6-input static cell presented there uses only 150 transistors compared to 648 needed for a similar LUT.

The fourth class, LUT-like cells, can also realize all possible functions for a certain input configuration. Unlike ULM, these devices however implement a memory based logic generator, just like traditional LUTs. Examples of such cells include the ones proposed by Kumar et al. [130] and improved by Guo et al. [131]. Both works use memristors to implement LUTs in less area than is needed for a silicon LUT realization. They mainly achieve this through combination of the data storage and the MUX or decoder network. Furthermore, they replace the SRAM for configuration storage with memristor elements. Such logic cells have been proposed as two, four and six input variants. Both Kumar's and Guo's implementations are similar, with the main difference being a reduced number of transistors in Guo's variant.

Another way to classify ambipolar logic generators is according to the way the logic elements have been designed. Here, Cheng et al. distinguish four classes [122]: CMOS like structures are based on complementary networks and include designs like the simple 2-function cells of [26]. Stack based cell architectures stack multiple rows, where each row consists of two transistors. Inverted output architectures are those which feature an inverter at the output, such as the designs of figure 2.22. BDD based cells are designed using BDDs, enabling the designer to specify the functions to be realized [132]. Yet another way to classify those cells is chosen by Rai et al. and groups devices according to technology [108]: The realizations by Jabeur et al. [123] and Cheng et al. [124] are DGCNTFET based, whereas Gaillardon' works are based on silicon nanowires [125]. Rai also presents Spintronic based cells as well as memristor based LUT realizations. As some of those ambipolar reconfigurable cells provide only limited expressiveness, some works have investigated efficient interconnect concepts to combine multiple of those cells. Yakymets et al. propose an efficient interconnect matrix [133], whereas Cheng et al. focus on the implementation of efficient cell clusters [124].

Further Reading

As there are no textbooks available on reconfigurable ambipolar logic cells, information has to be obtained largely from original publications. Some review papers are available and summarized in the following: [Che13] provides an overview of cell designs published until 2013. It analyzes the number of transistors used as well as the number of functions realized. In a publication

published three years later, Cheng et al. categorize various 2-input cells and analyze the realized functions in detail. They also discuss efficient combination of multiple cells in matrix structures [Che17]. Mikolajick et al. provide an overview of RFET device technology in 2017, including a short section on applications [Mik17]. They describe various published reconfigurable cells and cover the topic of Multiple Independent Gate FETs (MIGFETs), which allow for further optimizations in cell design. The most recent review on reconfigurable ambipolar cells was published by Rai et al. in 2021 [Rai21]. It summarizes both traditional FPGA logic generators and RFET based ones, as well as memristor and Spintronic cells. Rai et al. also evaluate these cells in terms of area, delay and power for certain FPGA benchmarks.

[Che13] Cheng, Kevin; Le Beux, Sebastien and O'Connor, Ian: "Am/IDG-FET based reconfigurable cells versus LUTs: Characteristics description and analysis". In: *2013 25th International Conference on Microelectronics (ICM)*. IEEE, 2013, pp. 1–4. DOI: 10.1109/ICM.2013.6734987.

[Che17] Cheng, Kevin; Le Beux, Sebastien and O'Connor, Ian: "Hybrid Topologies for Reconfigurable Matrices Based on Nano-Grain Cells". In: *2017 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2017, pp. 1–8. DOI: 10.1109/ICRC.2017.8123639.

[Mik17] Mikolajick, T.; Heinzig, A.; Trommer, J.; Baldauf, T. and Weber, W. M.: "The RFET—a reconfigurable nanowire transistor and its application to novel electronic circuits and systems". In: *Semiconductor Science and Technology* 32.4 (2017), p. 043001. DOI: 10.1088/1361-6641/aa5581.

[Rai21] Rai, Shubham; Nath, Pallab; Rupani, Ansh; Vishvakarma, Santosh Kumar and Kumar, Akash: "A Survey of FPGA Logic Cell Designs in the Light of Emerging Technologies". In: *IEEE Access* 9 (2021), pp. 91564–91574. DOI: 10.1109/ACCESS.2021.3092167.

2.7 FPGA System Architecture

Previous sections have introduced reconfigurable cells that can be used to realize small combinational functions. In order to design larger digital circuits, these basic cells need to be integrated in a larger, reconfigurable system. This thesis will focus on one specific type of reconfigurable systems, Field Programmable Gate Arrays (FPGAs). FPGAs are "field programmable", which means that they can be reconfigured "in the field", i.e. without manufacturing of customized hardware. This is in contrast to ASICs and mask-programmable devices, where customizing the function means developing at least some custom masks and manufacturing new ASIC devices using those. FPGAs are fine-grain reconfigurable devices: They are optimized to provide efficient manipulation of single bits, whereas coarse-grain systems operate on whole words – often even floating point values – at once [134]. As FPGAs enable the implementation of customized digital ICs without expensive manufacturing, they are widely used for prototyping and small-scale series production, where manufacturing ASICs is not cost-efficient. Commercial devices are available from various vendors with different performance, power and cost characteristics. The following section will summarize quickly how FPGAs are realized using the reconfigurable cells introduced previously.

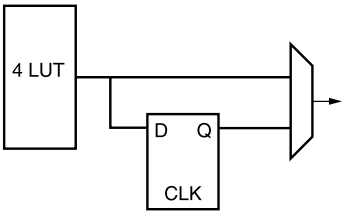


Figure 2.23: Combination of an D-FF and the reconfigurable cell in a LE [118].

Logic Elements LEs are based on the previously shown reconfigurable cells, but extend them in various ways. In commercial devices, LEs are usually realized using LUTs, but any of the previously introduced reconfigurable cells could be used. As this cell is the base element of FPGAs, it has to feature some way to reconfigure the logic function it realizes in-the-field. In addition, to support arbitrary user application logic functions, the FPGA must be able to provide a functionally complete set of boolean operations. Although this usually means one type of LE has to support such a complete set, it is also possible to use multiple types of LEs in an FPGA. Alternatively, some logic operations could be integrated within device interconnect, e.g.

an inversion as part of an inverting buffer. As long as the combined set of reconfigurable elements in the FPGA realizes a functionally complete set of operations, basic FPGA functionality can be achieved. Larger logic functions can then simply be decomposed into operations supported by the LEs using EDA tools.

In addition to the reconfigurable cells, which enable implementation of combinational logic, LEs usually feature a storage element [118]. In most implementations, this element is a simple D-FF connected to the output of the reconfigurable cell, as shown in figure 2.23 on the previous page. It is needed to implement sequential circuits, commonly used in Finite State Machine (FSM) implementations. The FF is usually not directly connected to the output of the LE, but via a user-configurable MUX: This design allows to optionally bypass the FF and output the combinational signal of the reconfigurable cell. It is mostly useful to realize large combinational functions using multiple LE. Similarly, a bypass for the reconfigurable cell, which is often realized using the identity function in LUTs, allows to use the LE as a basic memory storage element.

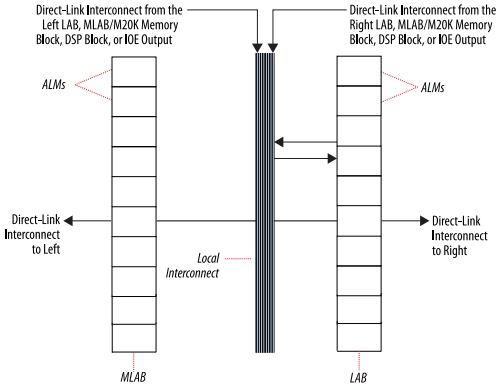


Figure 2.24: The logic cluster as used in the commercial Intel Aria 10 architecture [135]. The cluster consists of MLAB and LAB, each consisting of 10 ALMs.

Logic Clusters Multiple LEs are commonly combined in clusters, called Logic Array Block (LAB) in Intel and CLB in Xilinx FPGAs [120, 136]. An example of such a cluster, the LAB used in Aria 10 FPGAs, is shown in figure 2.24. A LAB consists of 10 Adaptive Logic Module (ALM), where "ALM" is Intel's term for LE [135]. As can be seen in the figure, logic clusters commonly share a local interconnect, which is their defining characteristic: The interconnect

is commonly the largest single contributor to FPGA area and application delay, with up to 50 % of both being caused by it [137]. Because of that, FPGA designers introduce one level of hierarchy using a local interconnect. This way, not all signals have to be routed using the global interconnect. Xilinx uses similar clusters called CLBs. As an example, in their Ultrascale+ architecture, this cluster consists of 8 6-input LUTs, 16 FF and additional carry chain logic [138].

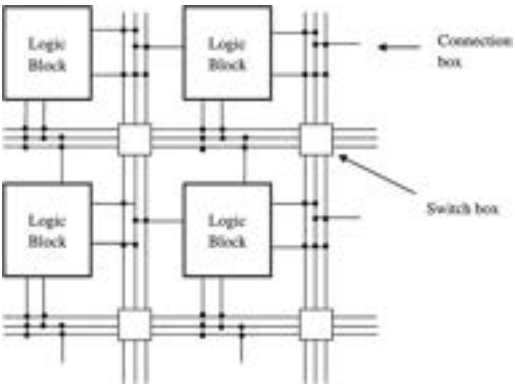


Figure 2.25: Defining characteristics of an island-style FPGA architecture [134]. The interconnect is routed in a 2D-grid, logic blocks are islands within the interconnect.

Global Architecture Logic Clusters are then combined in the FPGA system as depicted in figure 2.25. The main feature at this abstraction level is the FPGA's programmable interconnect. It allows to connect the logic clusters in a way that is configured by the user when programming the FPGA. The interconnect itself used to be an active area of research. Commonly, the Switch Boxs (SBs) have been altered to only support some connections instead of all, in an effort to reduce their area [120]. Similarly, Connection Boxs (CBs) often do not connect logic clusters to all signals in a channel, but only to a few. For the spatial placement of logic clusters, multiple variants have been used over time: Whereas row-based systems used to be common for early FPGAs, modern systems are largely island-style architectures. Those have shown to be efficiently implementable, as their defining characteristic, a regular 2D grid of wires, is easy to manufacture [137]. In addition, commercial systems tend to divide the FPGA into regions, where some resources such as clock signals, are constrained to individual regions. Commercial systems usually do not only consist of configurable logic clusters, but they include

hard logic such as memory blocks, Digital Signal Processing (DSP) blocks, optimized Input / Output (IO) and other Intellectual Property (IP) blocks [120]. Hard logic in this case describes logic which is directly realized on the chip, as opposed to soft logic, which is using the programmable logic resources.

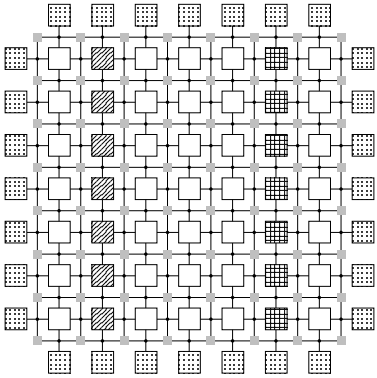


Figure 2.26: Simple FPGA architecture as used in the remaining thesis. IO blocks are at the periphery. Central blocks are logic clusters (white), memory (diagonal lines) and compute elements (grid). SBs are denoted as small gray blocks, CBs as dots.

Academic FPGA architectures on the other mostly focus on research of LE and interconnect. They are usually simple island-style architectures as shown in figure 2.26, with IO at the periphery. In real ICs this is difficult to realize, as large pin-count demand IO pads all over the chip area. Furthermore, hard logic for high-performance IO needs to be physically close to the IO pads, which may cause issues if pads are only at the periphery. Because of the research focus, such architectures commonly do not include any hard logic blocks. Figure 2.26 shows an illustration of an island-style FPGA architecture which will be used in the rest of this thesis. It has been simplified to show interconnect as simple dots and squares, avoiding visual clutter.

Programming & Storage Whether based on LUTs or other reconfigurable cells, FPGA also need reconfigurable storage. To date, three types of memory have been commonly used for that [134]: The most commonly used storage is SRAM storage. It is commonly used in high-performance commercial FPGAs and is relatively easy to integrate in manufacturing, as it has little demands

on device technology. Its main drawbacks are relatively large size of up to 6 or 7 transistors per bit and the relatively high power usage. Furthermore, the volatility of the storage necessitates additional non-volatile storage, which is used to program the SRAM storage in the final system. One way to avoid this external storage is to directly use Flash storage in the FPGA. This approach is not commonly used though, as it poses certain requirements on the technology used. Efficient realization of Flash cells requires thick oxide layers to prevent discharge of stored charges. Additionally, programming of the storage cells requires high voltages, needing additional circuits and making dynamic reconfiguration more difficult. As a slight variation, Dynamic Random-Access Memory (DRAM) FPGAs have been proposed in academia: They store data similarly to Flash based FPGAs, but need periodic refreshes of the stored charges. Whereas this relaxes some requirements on device technology, it also makes storage volatile again. The third category of memory is Anti-Fuse based memory. This approach has been used in older commercial devices. Its main benefit is the non-volatility and power efficiency of the storage. Programming again needs to happen using high voltages. This is then used to break down fuses, often realized in between metal layers to avoid area overhead on the semiconductor layers [139]. Whereas this storage is nonvolatile, programming is also permanent and does not enable any reprogramming. Furthermore, scalability to smaller devices nodes is limited.

Further Reading

Various textbooks on FPGA design cover FPGA architectures and some good survey papers are available as well. [DeH07] provides an extensive overview of all FPGA related topics, especially compute models and programming. As it was published 2007, it also features description of older commercial architectures. For an overview of PLA and PAL architectures, [Kuo07] can be used. In addition to those works, [Vas07] provides an overview of research architectures available in 2007. For a more recent introduction to FPGA architectures, [Ama18] can be recommended. It is supplemented by [Rod20] which focuses on use of commercial FPGA in industry contexts and [Bou21], which provides an overview of research FPGA architectures in 2021.

[DeH07] DeHon, André and Hauck, Scott: Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation. 1. Aufl. Systems on Silicon. s.l.: Elsevier professional, 2007.

[Kuo07] Kuon, Ian; Tessier, Russell and Rose, Jonathan: "FPGA Architecture: Survey and Challenges". In: *Foundations and Trends® in Electronic Design Automation* 2.2 (2007), pp. 135–253. doi: 10.1561/1000000005.

[Vas07] Vassiliadis, Stamatis, ed.: Fine- and coarse-grain reconfigurable computing. Dordrecht: Springer, 2007.

[Ama18] Amano, Hideharu, ed.: Principles and structures of FPGAs. Singapore: Springer, 2018.

[Rod20] Rodríguez-Andina, Juan José; La Torre-Arnanz, Eduardo de and Valdés Peña, María Dolores: FPGAs: Fundamentals, advanced features, and applications in industrial electronics. First issued in paperback. Boca Raton: CRC Press, 2020.

[Bou21] Boutros, Andrew and Betz, Vaughn: "FPGA Architecture: Principles and Progression". In: *IEEE Circuits and Systems Magazine* 21.2 (2021), pp. 4–29. doi: 10.1109/MCAS.2021.3071607.

2.8 Synthesis and Implementation

The following section gives a quick introduction to EDA tools. This thesis will introduce both standard cell design for RFETs and new approaches for application mapping to RFET-based FPGAs, so both FPGA and ASIC design flows will be explained. Due to the similarities of the design flows, and FPGA tools being more relevant for this thesis, the FPGA flows are used as an example and relevant differences to ASICs will be noted inline. Figure 2.27 shows involved EDA tools for FPGA application design. This figure includes host integration, as in the general case, FPGA may be used in combination with a host processor [140]. This aspect is of no particular relevance to this work though and therefore won't be elaborated further.

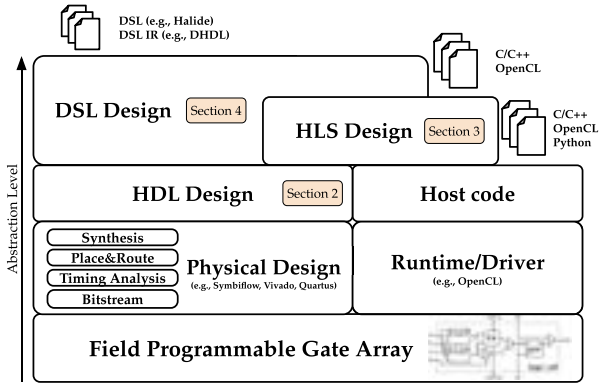


Figure 2.27: Design flow for FPGA applications, including design capture using High Level Synthesis (HLS) and Domain Specific Languages (DSLs), as well as integration with host code [140].

Design Capture The first step in designing an FPGA application is design capture [141]. In this step, the user describes the application in a machine-readable way. For higher productivity, DSLs and HLS have been introduced as ways to enable higher-level descriptions than the traditionally used Hardware Description Languages (HDLs) VHDL and Verilog. A detailed overview of such high-level approaches is given in [140], but the applications used in this thesis will start on HDL level.

Synthesis After a HLS design has been compiled to HDL or a user has developed a HDL design, the first step is to transform it to a technology-independent netlist in Register Transfer Level (RTL) form. The RTL form

consists of only registers and combinational logic, allowing direct mapping to the reconfigurable and non-reconfigurable logic cells introduced previously. The transformation step, technology-independent synthesis, replaces high level language constructs such as processes, and emits a structured netlist. As a technology-independent description of combinational logic, netlists often include function tables with an arbitrary number of inputs. The tools commonly used in the open source community for this synthesis step are ODIN II [142] and *Yosys* [143]. In addition, various commercial tools are available. In general, technology independent synthesis does not differ for ASICs and FPGA targets and tools like *Yosys* have been used for both. Nevertheless, FPGA vendors often include tools specific to their FPGAs in their tools, such as in Xilinx *Vivado*. Similarly, tools like Cadence Genus are mostly used for ASIC design. Often, technology independent synthesis and technology mapping, are combined in one tool.

Technology Mapping Technology Mapping is the step of converting the technology independent RTL netlist to a technology specific netlist. For ASICs, the tools have to ensure to only use logic cells available in a standard library. For LUT based FPGA, the problem is easier, as LUT can realize any function. The only limitation in that case is the number of available inputs of the LUT, reducing the problem to a covering problem which can be solved using e.g. dynamic programming [118]. Both standard cell mapping and LUT mapping is handled by *ABC* in open source design flows. *ABC* converts netlists into an internal And-Inverter Graph (AIG) representation, performs technology independent logic optimization, and also maps the internal presentation to cells or LUTs [144]. Commercial tools again include FPGA specific vendor tools such as *Vivado* and tools such as Cadence Genus for ASICs.

ASIC tools can in general be used with different technology nodes and fabs. For a tool to work with their technology, technology vendors provide PDKs to be used with those tools. For the technology mapping, PDKs provide Liberty or *.lib* files, describing which standard cells are available and their timing information. The format developed by Synopsys is text based and can therefore easily be generated manually [145]. Apart from information about the cells themselves, such as names and pins, the file includes all possible timing ARCs for the cells. Furthermore, it includes a wire-load model to be used to derive an approximation of the capacitive load caused by wires connecting cells. This load information is then used to look up the timing arcs, which are usually parametrized on this load. Traditional standard cell libraries may provide multiple cells with the same function for different wire-loads. The synthesis then performs STA as explained earlier and chooses

a cell which minimizes the path delay or area, depending on optimization goal and criticality of the path. Whereas STA for ASICs works as previously explained, STA for FPGA applications is simplified: For LUT based FPGAs, the delay introduced by logic elements is always the same. However, in FPGA the interconnect, delay is more important: Unlike in ASICs it is not only caused by parasitic wire loads. It also includes the delays caused by interconnect switches. For both targets, synthesis will perform STA even before placement to obtain estimates of expected delays and guide synthesis. The longest, critical path will limit the maximum achievable frequency, which has to match or surpass the target frequency set by the application designer. In ASIC, STA is explicitly performed on different corners because of the delay variations due to PVTA. Usually, results of the worst corner have to be used, pessimizing the timing results. Even though FPGA tools often do not expose various corners for STA, they are also affected by PVTA and internally assume worst case delays of cells.

Packing Packing or Clustering is a step performed before the placement step, reducing that step's complexity [118]. It is most commonly used to cluster primitives which need to be placed in related locations on the final FPGA. An example for this is Versatile Place and Route (VPR)'s VPACK algorithm: It is primarily used to cluster LUT and FF which usually are realized in one LE in FPGAs. Furthermore, it also allows packing multiple LE within a logic cluster [146]. Apart from handling some parts of the placement process, packing also handles some placement legality constraints, simplifying the implementation of placement algorithms [118].

Placement Placement can be realized using structured or unstructured approaches. Structured approaches use information about the design hierarchy, whereas unstructured approaches view the circuit as a large unstructured network of logic blocks and FFs. Structured approaches include datapath oriented placement as well as user guided placement variants. The most commonly used placement algorithms however are unstructured [118]. For ASICs, placement usually means placing rectangular elements on a 2D grid or placing standard cells within predefined rows. For this, information about the physical dimensions of a cell is needed, which is provided as a `.lef` file. Placement for ASICs can be separated in global and detailed placement. Global placement may then generate illegal results, such as small overlaps between cells, which need to be sorted out in detailed placement [46]. Apart from stochastic placement algorithms, ASICs sometimes use analytic approaches: Here, a global function describing the total wire length is used to analytically find a minimum [118]. FPGAs usually use stochastic placement algorithms,

where the most commonly used ones are based on simulated annealing [118]. For example, VPR's placer operates in three phases [146]: In the first, it produces a random placement. In the second it performs random pairwise swaps and in the third reevaluates costs. Using the standard deviation of these initial placements, an initial temperature for the annealing is derived. As a last step, the annealing itself is performed with a specific temperature schedule. For an open source tool placement tool targeting commercial FPGAs, refer to the nextpnr tools [143].

Routing Routing finds legal connections between placed logic elements, according to the netlist. In ASICs, routing is often split into global and detailed routing phases [46, 147]. Global routing in this case determines routing between certain predefined regions, whereas detailed routing then finalizes the connections for each single network. FPGA tools usually combine global and detailed routing, as resources in routing channel are more limited than in ASICs [118].

Figure 2.28: Tools and file formats used in common FPGA design flows [148]. The figure focuses on open source tools, but includes some commercial tools for reference. Highlighted is the *fasm* format, which is used to as a text-based bitstreams description for various FPGA architectures.

Bitstream Generation After a legal placement and routing has been obtained, the reconfigurable cells and interconnect of the FPGA have to be configured to realize this result. For that, usually a file containing the configuration bits for all reconfigurable units is created [118]. For LUTs, this means directly storing the function table, whereas for other elements, the configuration has to be encoded somehow. Regarding the interconnect, selected multiplexer bits are usually stored directly. Figure 2.28 shows tools and file formats used in open source design flows for FPGAs. Bitstream generation is generally an FPGA dependent task, but the FPGA ASM (FASM) format allows to encode this information in a common text format [148]. FASM data can be generated by VPR as a result of the place and route phases

[149] and can then be used in custom tools to transfer the result to a binary format.

Further Reading

EDA tools and algorithms have been covered for both ASICs and FPGAs in literature. For DSLs, HLS and programming paradigms for FPGAs in general, refer to [Del23]. An in-depth treatment of mapping, placement, routing and bitstream generation for FPGA is given in [DeH07]. For a shorter, more recent summary of those topics, refer to [Ama18]. Examples and case studies of FPGA architectures and tools can be found in [Vas07], whereas an industry oriented introduction covering mixed signal simulation and debugging in general is given by [Rod20]. ASIC EDA topics are covered in detail in [Ger99], with a shorter, more recent summary given in [Kah22]. For an overview of design flows and mixed signal aspects, refer to [Wes11].

[Del23] Del Sozzo, Emanuele; Conficconi, Davide; Zeni, Alberto; Salaris, Mirko; Sciuto, Donatella and Santambrogio, Marco D.: "Pushing the Level of Abstraction of Digital System Design: A Survey on How to Program FPGAs". In: *ACM Computing Surveys* 55.5 (2023), pp. 1–48. doi: 10.1145/3532989.

[DeH07] DeHon, André and Hauck, Scott: Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation. 1. Aufl. Systems on Silicon. s.l.: Elsevier professional, 2007.

[Ama18] Amano, Hideharu, ed.: Principles and structures of FPGAs. Singapore: Springer, 2018.

[Vas07] Vassiliadis, Stamatis, ed.: Fine- and coarse-grain reconfigurable computing. Dordrecht: Springer, 2007.

[Rod20] Rodríguez-Andina, Juan José; La Torre-Arnanz, Eduardo de and Valdés Peña, María Dolores: FPGAs: Fundamentals, advanced features, and applications in industrial electronics. First issued in paperback. Boca Raton: CRC Press, 2020.

[Ger99] Gerez, Sabih H.: Algorithms for VLSI design automation. Chichester and Weinheim: Wiley, 1999.

[Kah22] Kahng, Andrew B.; Lienig, Jens; Markov, Igor L. and Hu, Jin: VLSI Physical Design: From Graph Partitioning to Timing Closure. Cham: Springer International Publishing, 2022. doi: 10.1007/978-3-030-96415-3.

[Wes11] Weste, Neil H. E. and Harris, David Money: CMOS VLSI design: A circuits and systems perspective. 4. ed. Boston, Mass.: Addison-Wesley, 2011.

This page intentionally left blank

Chapter 3

Related Work

This chapter covers related work for the various aspects covered in this thesis. It starts with ambipolar standard cell libraries, continues with ambipolar FPGAs, introduces works related to dynamic reconfiguration in FPGA, summarizes some PVTA compensation works and power management techniques and concludes with an overview of synthesis approaches for reconfigurable cells.

3.1 Ambipolar Standard Cell Libraries

To develop digital circuits based on ambipolar devices in an automated, standard cell based EDA flow, a matching PDK including a set of standard cells is required. Traditionally, PDKs have been largely shipped as closed source IP and have been provided by the foundries which developed the technology nodes and ultimately manufacture the ICs. In recent years, there have been releases of open-source PDKs for commercially available CMOS technologies. Two famous examples include SkyWater's 130 nm *SKY130* [150] and GlobalFoundries 180 nm *GF180MCU* [151] technology.

Predictive PDKs It is possible to develop PDKs without having a complete manufacturing process for a technology. Those PDKs are called Predictive PDK (PPDK), as they describe a predicted technology. Traditionally, academia did not have access to the small-feature size technologies, such as recent FinFET technology. Because of that, researchers build PPDKs for these CMOS technologies, enabling them to prototype and test their circuit designs on modern technology nodes without access to the foundries commercial PDK. One of the earliest example for a standard CMOS PPDK is the 45 nm *FreePDK* [152]. Some years later, *FreePDK15* [153] was developed to target a functional 15 nm FinFET technology. A basic PDK such as *FreePDK15* usually includes

various rules and definitions: A set of design rules specifies required distances between traces and similar geometric rules. The layer definitions determine what layers are available and what properties they possess. This includes active layers for the design of transistor devices as well as metal layers for wiring. Furthermore, a PDK usually includes SPICE simulation models for the transistor devices. More advanced PDKs may also provide support for Parasitics Extraction (PEX) to determine parasitics from the layout and Layout vs. Schematic (LVS) to verify that the layout matches the schematic. With those parts, PDKs can be used to design and simulate analog circuits, including the standard cells themselves.

View	File	Description
Technology Library	`.lib`	Provide logic, timing, power and area information of the cells in the library.
Geometric Library	`.lef`	Provide information about the physical layout of the library in plain text, including design rules and abstract information about the cells.
Simulation Library	`.v`	Provide a behavioral information of the cells for simulation intents.
Cell Layouts	`.gds`	Provide information about planar geometric shapes, text labels, and other layout information in a binary format.
Cell Netlists	`.spi`	Provide an instance-based transistor netlist, representing instances, nets, and some attributes.
OpenAccess	`.oa`	Provide a database containing layouts and netlists.

Table 3.1: Views and file formats to describe a standard cell library for use in common commercial EDA tools. The table shows the views supported by the *FreePDK15* standard cell library and was taken from [154]. Not all functions in the EDA flow require all views.

Standard cell libraries are often not included with the PDKs themselves, but offered as an additional package. For example, for *FreePDK15*, a standard cell library has been provided by Martins et al. [154]. Depending on the actual EDA tool and the task to be performed, different information grouped in "Views" is needed. Table 3.1 shows the views provided by the *FreePDK15* library, which is a mostly complete set. Here, the `.lib` file is used for synthesis, `.lef` for physical synthesis, `.v` is used for simulation, the `.gds` file for streamout and the `.spi` file for timing characterization of the standard cells. The `.oa` file is a database which combines some of these individual files. Standard cell libraries vary in the number and type of cells they provide, as well as in the

functions those cells can realize. For example, [154] provides 76 cells with 21 logic functions. Some cells are available in different drive strengths, so that cells with higher drive strength can be used in high-fanout situations. In addition, buffer and inverter cells are usually available in more drive strengths than usual cells. The library also provides sequential cells such as FFs, scan-flops and latches. In addition to such logic cells, libraries provide helper cells: Antenna cells, tie high and tie low cells and filler cells. The implementation of cells commonly follows a template, as described in [154]: Apart from the physical height of cells, usually positions of the power supply pins as well as some well dimensions are the same for all cells. Such regularity ensures that the user application design can later be routed more efficiently by the EDA tools. Timing and power information are ultimately stored as simple tables in the respective views, where values may be interpolated by the EDA tools if necessary. Nevertheless, some methodologies have been established for the characterization of cells to obtain these tables: As presented in [154], the non-linear delay model is commonly used, the Composite Current Source model is used by Synopsys and the Effective Current Source model is used by Cadence. In general, tools like Cadence Liberate can provide an automated characterization of CMOS standard cells: Given a netlist for a cell, these tools automatically create SPICE netlists to simulate the cell. The simulations are then performed using the SPICE model of the PDK and the results are used to derive the values for the `.lib` file. This approach can also be used to characterize RFET based cells, as it is independent of the cell layout. It however requires fully functional SPICE models.

RFET Specific Optimization When it comes to logic cells for RFET, few publications provide a complete set of standard cells. Most of the publications focus on some specific issues and optimization instead: For example, Rai et al. propose a set of six functionally enhanced logic gates [126]: Compared to commonly used standard-cells, these integrate XOR operations, which can be efficiently realized using RFET. In another publication, Rai et al. introduce the concept of self-dual functions [155]: They first observe, that a specific class of reconfigurable cell can be obtained by switching of pull-up and pull-down networks, including the power rails. Such an operation makes a cell switch between its dual functions. For example, a traditional CMOS *NOR* becomes a *NAND* it its pull-up network and pull-down network are switched. Of course, for traditional technology, switching means keeping the network topology but using the correct transistor type in each network. For ambipolar transistors used in RFET circuits, the transistors in pull-up and pull-down network are identical and switching their polarities and power rails can indeed realize the opposite circuit. Based on this idea, Rai proposes a workflow

which finds such pairs of functions, called self-dual in the publication. The workflow then automatically derives a set of standard cells which implements these special, reconfigurable cells. In benchmarks, Rai et al. could show a decrease in used area of 13 % and a decrease in delay of 11.5 %. Another RFET specific optimization for standard cells has been proposed by Krinke et al. in [156]: After observing that the additional configuration gates used by RFET circuits add additional wiring overhead, Krinke et al. propose various solutions to this issue. Apart from optimizing the layout of reconfigurable cells and relative positions of these additional gate contacts, they also introduce special buffer drivers to drive the gates in a reconfigurable cell. Furthermore, they investigate placement of parts of the cells in specific power regions to reduce power.

RFET PPDKs and Standard Cells To date, few standard cell libraries have been presented for RFET PPDK. The first one was presented by Ben-Jamaa et al. in [157] for Carbon Nanotube FETs (CNTFETs). They developed a library based on *NOR, NAND, AOI* and *OAI* gates, but extended it with "non-conventional" gates embedding *XOR* functionality. They also realized 46 functions with one base cell design to achieve a reduction of 26 % in delay and 32 % in area compared to using only unipolar CNTFET gates. The library characterization was performed mostly manually. For area, the authors provided relative estimates based on a weighted device count: They normalized the area of each used transistor in respect to a unit transistor and then summed the normalized area of all transistors in a cell. They did however not consider the complete layout of each single cell. For the delay, Ben-Jamaa et. all used SPICE simulation to obtain the FO4 delay, where each gate is driving 4 instances of the same gate at its output. For evaluation, the authors used the open source ABC tool to perform technology mapping. They developed a custom genlib file to specify the available gates to ABC and mapped various benchmark circuits, achieving a 6.9 % average speedup.

In 2018, a new set of standard cell libraries was presented by Rai et al., this time for Silicon Nanowire (SiNW) technology [158]. Unlike Ben-Jamaa, Rai et al. designed full cell layouts for seven cells and characterized their library in `.lib` and `.lef` files. Their library does not contain any sequential elements. To obtain the timing characterization, the authors simulated the SPICE netlist with a Verilog-A table model for the silicon nanowire transistors. Area was obtained from the gate layouts. Using those files, they evaluated the MCNC benchmarks using the qflow flow, which internally uses Yosys and ABC. Their circuits required in average 13 % more area than a comparable CMOS implementation, which used an scaled version of *FreePDK45*.

Yet another PDK was presented by Gore et al. in 2019 [159]. The PDK features Verilog-A SPICE models, design rule manuals and Design-Rule Check (DRC) and LVS support for 10 nm silicon-nanowire Three-Independent-Gate FET (TIGFET). As common for these PPDK, the SPICE model was derived as a table model from Technology CAD (TCAD) simulation. Whereas this initial publication did not include a standard cell library, it was presented in 2022 by Gauchi [160] and Keyser [161]. Using the analog features of the PDK, Gauchi designed layouts for 11 cells in Cadence Virtuoso. The cells where then characterized using Cadence Liberate and the results where analyzed using benchmarks. Whereas the analog flow and design of the libraries used Cadence tools, the benchmarks where synthesized using open source tools such as Yosys. When analyzing the synthesis results of the PicoRV32 RISC-V processor, Gauchi reported 2.3 times less area and 5.7 times less energy compared to the GlobalFoundries 12 nm technology. Figure 3.1 shows the cell characterization flow for this technology as described bey Keyser [161]. Keyser's description focuses more on physical aspects of the cells. After layout, DRC and LVS, they authors also perform PEX to assess parasitics of the devices. They then use Cadence Liberate to obtain the `.lib` files.

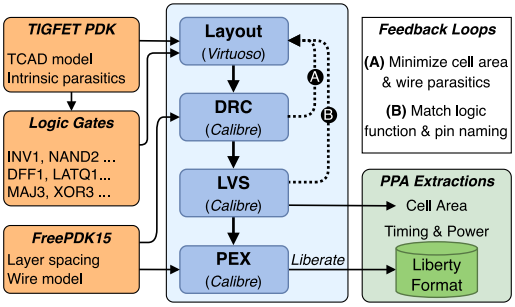


Figure 3.1: Design flow for a TIGFET standard cell library, taken from [161]. Gates are designed using the analog parts of the TIGFET PDK of [159]. Functionality is then verified in DRC and LVS checks. The gates are then combined with wire models taken from *FreePDK15* to obtain parasitics for the final characterization.

The latest PDK was presented by Quijada et al. in 2022 [162]. It targets the germanium nanowire technology and again derived its SPICE model as a Verilog-A table model from TCAD simulation. The authors analyze various cells in SPICE, but do not provide any `.lib` or `.lef` files for a complete standard cell library. The authors claim that parasitic effects can easily be derived, because the nanowire design is based on a commercial FinFET process and values can

Author	Ben-Jamaa	Rai	Gore Gauchi Keyser	Quijada
Year	2011	2018	2019 – 2022	2022
Publication	[157]	[158]	[159] [160, 161]	[162]
Device Used	CNTFET	SiNW	SiNW TIGFET	Germanium Nanowire
Sequential Cells	no	no	no	no
Characterization	Manual	Silicon Smart	Liberate	no
Synthesis	ABC	Yosys ABC	Yosys	no
Wireload	?	?	*FreePDK15*	no
Si-Compatible	no	no	no	no

Table 3.2: Overview of previously published PDKs for ambipolar transistor devices. The table compares the technology used, whether sequential cells such as FFs are provided, how `.lib` files are obtained, whether the PDK supports standard cell synthesis for digital circuits, whether it integrates a wireload model and whether it can be mixed with existing silicon standard cells.

be adapted. But before standard cells can be provided, the PDK needs to be extended with DRC and LVS support first.

3.2 Ambipolar FPGA Architectures

Although various publications have covered ambipolar reconfigurable cells and FPGAs are commonly mentioned as the main use case, few publications on ambipolar device based FPGAs are available. Part of the reason for this is that a circuit level implementation and evaluation requires a full PPDK and EDA integration. So far, only one such PPDK for RFET has been published and only in 2022: The SiNW TIGFET PDK by Gore, Gauchi and Keyser [159, 160, 161]. Without such a PDK, full circuit level implementation, simulation and evaluation is not possible. Nevertheless, some previous works have investigated partial aspects of ambipolar FPGA architectures.

Novel non-RFET FPGAs Some publications have investigated FPGA architectures using novel LEs, but still using conventional technology. An example for this is the work by Parandeh-Afshar et al. in 2012 [105]: In this publication, LUTs have been replaced with AICs. As a motivation, the authors argue that programmable devices have previously been designed to fit EDA tools: As those tools mostly used SOP representation of logic, PALs and similar devices have focused on *AND* and *OR* gates. With the adaption of AIGs in EDA tools, the introduction of AICs enables a simple technology mapping implementation. As AICs are less expressive than LUTs, a larger number of AICs has to be used, increasing demands on the interconnect. To avoid congestion, the authors propose the introduction of AIC clusters, where they also evaluate various different sizes. For the local interconnect in these clusters, the authors evaluate various depopulation levels for crossbars to reduce the area usage. For their architecture, they found 75 % populated crossbars to provide the best trade off. In addition to a purely AIC based system, the authors also introduce a hybrid system which mixes LUTs and AIC. Little details are provided for the hybrid system, but ultimately, the authors claim this architecture reduced delay by up to 32 % and area by 16 %.

Another novel, non-RFET FPGA architecture has been proposed by Gonçalves et al. in 2013 [163]. In this architecture, the authors replaced normal SRAM-based LUTs with Magnetoresistive Random-Access Memory (MRAM) backed ones. This architecture is intended to be used in space systems, where radiation is causing issues with SRAM based storage. The 2 input LUT, a combination of MRAM for long-term storage and DRAM for permanent access to the stored data, has been manufactured and analyzed. The authors have not discussed the system architecture of their FPGA or any implications for EDA tools. As the new architecture is however still using LUTs, it can be assumed that few or no changes are necessary.

Ambipolar FPGA Architectures The first and to the knowledge of the author only complete description of an FPGA architecture based on ambipolar devices has been given in various publications by Ben-Jamaa and Gaillardon. In 2011, they first described their reconfigurable logic cell, a DGCNTFET based replacement for LUTs [164]. In addition to the reconfigurable cell, the authors also introduced a way to allow permutation of power lines, which may be needed if nets change between pull-up and pull-down functionality. Special to this initial publication is that configuration inputs of the cell are not directly connected to storage. They are realized as normal inputs instead and routed using the interconnect. The authors present a EDA flow based on ABC, VPACK and VPR and evaluate various benchmarks for their architecture.

They report a 13 % speedup compared to a reference architecture with LUTs, but at an area overhead of 10 %. The authors have not made use of any other benefits of RFET and have not introduced any novelties in the FPGA system architecture.

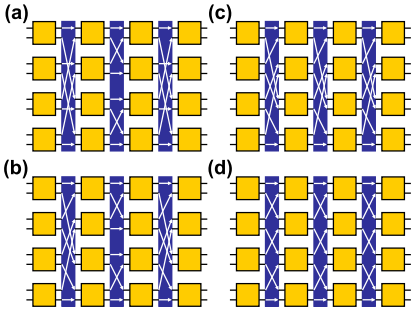


Figure 3.2: Introduction of the MCluster and different internal routing architectures [165]. MClusters reduce the amount of signals which are routed on the global interconnect and therefore reduce routing congestion.

A slightly modified variant of this FPGA architecture has been presented in the same year by Gaillardon and Ben-Jamaa [166]. The main novelty of this publication is the introduction of MClusters, shown in figure 3.2. Using 3x3 MClusters, the authors managed to reduce the area by 62 % compared to LUTs. As MClusters use a special interconnect implementation, there are more changes required in EDA tools, which are explained in detail in the publication. Again, the publication does not make use of any other features of RFET, e.g. for power reduction. The final version of this architecture has been presented 4 years later, in [165]. It is based on the same dynamic logic reconfigurable cell as the previous publications and again uses MClusters. This publication however puts even more focus on EDA aspects and evaluates a larger set of benchmarks. In addition, the authors perform an evaluation of various granularity levels for MClusters.

Whereas the FPGA architecture by Gaillardon et al. did not make use of special RFET features, a publication by Park et al. in 2017 [167] explicitly focuses on one such feature: The technology used enables the storage of configuration data on the BGs of the device. The BGs therefore effectively work as a flash memory, as shown in figure 3.3. Programming of the BGs is carried out using high voltage pulses, where the pulse duration determines drain current and pulse potential determines threshold voltage. Unfortunately, the publication provides little additional information: The transistor technology

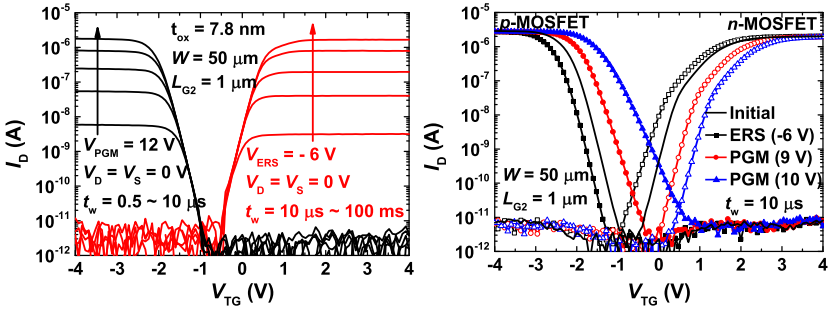


Figure 3.3: Programming of a transistor using charge storage on the BG [167]. **Left:** Programming pulse polarity determines device polarity. Pulse duration determines drain current. **Right:** Variation of programming voltage affects threshold voltage.

has not been presented in detail. It is based on poly-silicon and the measured devices have gate lengths of 1 µm. The authors did not explain whether they expect their technology to be shrinkable to smaller features sizes. Similarly, little information is given about the reconfigurable cell or the system FPGA architecture. Although the publication envisions high-density reconfigurable devices, it does not present such a system. The publication rather focuses on the description of the device on transistor level.

3.3 Dynamic Reconfiguration

Modern FPGA architectures do not only allow to configure FPGAs once at startup, but provide more advanced configuration features. As most commercial FPGA are SRAM based, various advanced applications using the ability to quickly modify memory have been developed. In general, reconfiguration, the process of programming an FPGA with a bitstream, has been distinguished in different categories [168]: Reconfiguration in general describes a replacement of the configuration of the whole FPGA. In contrast, partial reconfiguration replaces only a part of the FPGA bitstream. Both techniques can be further distinguished as static or dynamic reconfiguration: In static reconfiguration, non-reconfigured logic is kept in a reset, or stopped state. In dynamic reconfiguration on-the-other hand, those areas which are not reconfigured contain active logic and FPGA applications mapped to those areas are not interrupted by reconfiguration. Most commercial FPGA now either provide only static

reconfiguration of the whole FPGA, or they provide Partial Dynamic Reconfiguration (PDR), allowing to replace parts of the logic while keeping other parts of the application working.

A summary of recent academic and commercial reconfiguration architectures has been collected in 2019 by Vipin et al. [168], whereas older architectures have been described in [169]. PDR enables various new application patterns: Logic can be time-multiplexed, which allows execution of large circuits on smaller FPGAs. As reconfiguration is often implemented using serial register chains and serial programming, the data transfer rate in reconfiguration is often limited. Reconfiguration can therefore be performed faster in general, if only a part of the bitstream is changed. Another benefit of PDR is that a part of the user application circuit can be kept active during reconfiguration. This can be useful to keep a peripheral link active and is commonly used with Peripheral Component Interconnect Express (PCIe) connections to host computers.

Reconfiguration architectures can be distinguished into architectures supporting fine-grain reconfiguration and coarse grain reconfiguration. Whereas fine-grain reconfiguration allows reconfiguring individual programmable elements, most architectures combine multiple elements to be reprogrammed at the same time into Partially Reconfigurable Regions (PRRs). Grouping elements in this way allows reducing area overhead of the configuration network at the cost of reduced flexibility. Most architectures support reading back the configuration SRAM, but this feature is often not exposed. Cardona et al. used this feature to read back frames, modify single LUTs and write back the bitstream [170]. This enabled them to support fine-grain reconfiguration on an architecture which originally only supports frame-based reconfiguration. In addition to reconfiguration, some architectures support relocation: In the relocation case, a user application can be placed onto a location on the FPGA for which it has not originally been synthesized. Such concepts are most common when multiple applications are to be executed on one FPGA. In such a case, it might not be known during synthesis time which applications will be running on the FPGA later on, and the location can not be determined ahead of time. Most of the commercial toolchains still require defining possible target areas ahead of time. An application can then be placed onto any such block, but not onto freely chosen locations. Yet another conceptual difference can be found in reconfiguration time: Whereas specialized early architectures enabled reconfiguration in one clock cycle, for recent commercial architectures, reconfiguration is a slower process.

In an abstract view, the configuration memory of an FPGA can be thought

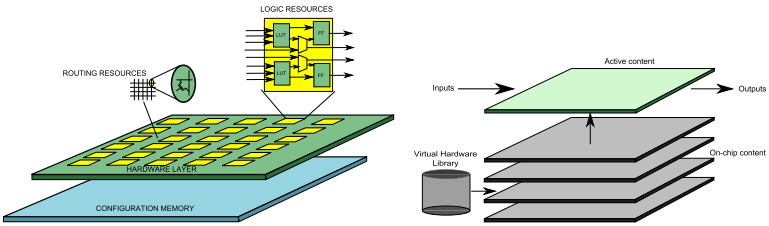


Figure 3.4: Conceptual view of reconfiguration in FPGA [168]. **Left:** Configuration memory as a virtual layer, independent of the hardware layer. **Right:** Extending the concept to multiple memory layers leads to multi-context FPGAs.

to be independent of the hardware logic, as shown in figure 3.4. The figure also shows a conceptional view of multi-context FPGAs, which was an active topic of research in the late 1990s. As FPGA were limited in size, such a concept allowed to split larger circuits in a time-multiplexed manner. To realize this, multi-context FPGAs store multiple independent bitstreams and switch between those, usually on a fixed clock-cycle schedule. Switching between different bitstreams leads to lots of changing signals, which ultimately causes a large activity factor for most nets in the FPGA. This leads to high power consumption, which ultimately caused this concept to be no longer used when large FPGAs became available [168].

When it comes to adoption in commercial architectures, Xilinx and Intel support PDR on a coarse grain level. Xilinx introduces frames as the smallest reconfigurable unit. In early Xilinx architectures, frames used to cover complete columns in the FPGA. They have become smaller in more recent architectures though. National Semiconductor, Lattice and Actel initially supported PDR, but removed it from later architectures. In general, PDR has not been adopted by a larger audience [168]. Nevertheless, some applications where it has been used will be explained in more detail:

Task Based Reconfiguration Apart from domain-specific examples, PDR has been mostly investigated as part of task based reconfiguration systems. Such systems take the idea of tasks as used in software Operating System (OS) and transfer it to FPGA. As shown in figure 3.5, hardware tasks are usually realized as partial bitstreams describing a local region of the overall FPGA.

Publications for FPGA tasks can be roughly sorted into four categories: Task-Mapping, Task-Scheduling, OS and Hardware-Software integration. Most

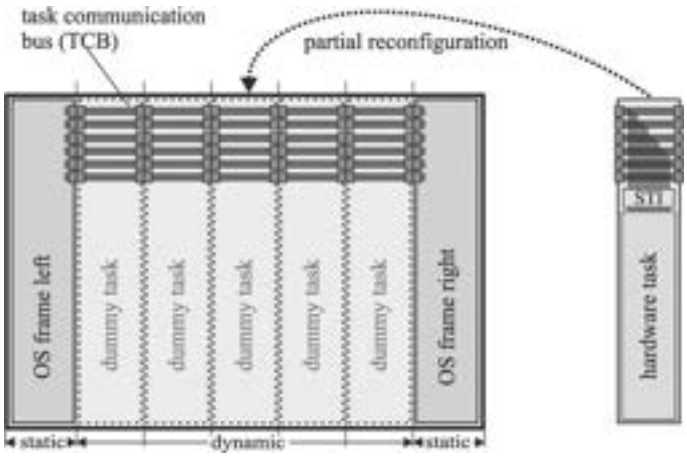


Figure 3.5: Hardware tasks in a 1D area model [171]. In the 1D model, tasks are always full-height and vary only in width, simplifying task placement. Tasks need predefined communication interfaces to be relocatable. Steiger's architecture also reserves some area for OS support.

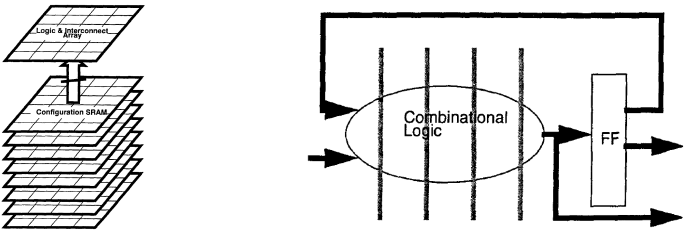
publications do not describe any changes in FPGA architecture, but they make certain assumptions on the reconfiguration system. When it comes to task mapping, one of the earliest publications was published by Diessel in 1997 [172]. The publication assumes a homogeneous FPGA which allows arbitrary relocation of logic in two dimensions. For such an architecture, the authors presented an algorithm which can find free space to efficiently map tasks. If not enough space is available, tasks can be moved to reduce gaps, called compaction or defragmentation. In [173], Walder et al. use a similar approach, but propose a different algorithm to efficiently keep track of free space. A more recent take of task mapping is given by Sidiropoulos et al. in [174]. The authors use an architecture with multiple independent FPGA cores and combine those with a host computer. The host computer keeps track of the unused logic resources within each FPGA core. When an application needs to be mapped, the system uses the netlist to actually place and route the design on demand. It takes into account information about already used resources and therefore enables tasks to overlap in area. Compared to other systems which are based on pre-placed and pre-routed tasks, this allows for better resource usage. For large FPGA designs, the reconfiguration time can however become excessive when a place and route step has to be performed on demand.

Task scheduling has been discussed starting with [175]. In this publication, authors describe a system which allows to choose the location of tasks at runtime. They provide an algorithm for efficient arrangement and packing, assuming a homogeneous FPGA architecture. For recent, commercial FPGAs, Sterpone et al. introduce a custom routing system [176]. As this system is aware of reconfiguration frames, it routes the design in a way to minimize the number of frames and bitstream size. This allows for more efficient reconfiguration of tasks. Another scheduling algorithm has been presented by da Silva et al. and focuses on streaming applications [177]. It uses runtime task scheduling and introduces a performance model to enable prediction of the speedup.

The third topic, OS for FPGA systems, has been lead by Steiger et al. In 2003, they first described the idea of an FPGA OS as shown in figure 3.5, including a scheduler, placer and loader for tasks [178]. The publication's main focus however is on algorithms for planning and placement. In the followup publication [171], Steiger et al. then describe the OS in detail. It introduces online scheduling with hard realtime guarantees. It further asserts that as FPGA systems do not always allow arbitrary relocation of logic, tasks need to use a pre-defined communication scheme. In such a scheme, the location of communication logic within a task is fixed.

More recently, with the introduction of FPGA in data centers and cloud computing, hardware-software co-design aspects have been investigated. In such systems, a host computer is used to configure the FPGA with various tasks. These tasks are then used to accelerate certain specific operations, but the main application logic is still executed in software on the host computer. Publications include [179], which was one of the first to discuss the topic and to combine software and hardware tasks. Janßen et al. then expanded on the concept by providing a predefined library of hardware accelerators, so-called hardware overlays [180, 181]. Their system was on of the first to be integrated with the PYNQ software stack. Another hardware-software integration framework which provides automated design-space exploration to find efficient trade-offs is TaPasCo. It focuses on parallel computation of tasks and ease of use in software [182].

Fine Grain Reconfiguration Previously mentioned publications have not made any changes to FPGA architecture, although some of them have assumed fine-grain reconfigurability with relocation support. In the following, custom architectures with novelties in the reconfiguration system are reviewed quickly.



(a) (b)

Figure 3.6: Overview of a time multiplexed FPGA as described by [183]. **(a)** Working principle with multiple independent memory layers time multiplexed for one logic layer. **(b)** Splitting combinational logic into multiple contexts.

Figure 3.6 shows one of the first multi-context FPGA [183]. It is reconfigurable in one cycle and allows to choose one out of eight stored configurations. To support quick saving and restoring of the state, the architecture introduced micro registers, which can store CLB outputs. The authors also explore various usage patterns of such architecture in detail. One of those, Logic Engine mode, is shown in figure 3.6b. It depicts how the system can be used to time-multiplex a single design: Combination logic is split into multiple parts and executed in multiple cycles.

A similar architecture was proposed by Li et al. [184]. Based on multi-context FPGAs, they introduce configuration caching: To reduce the time spent in reconfiguration, the authors propose various optimizations to reduce the number of reconfigurations. They also propose caching algorithms with work efficiently for relocation and defragmentation.

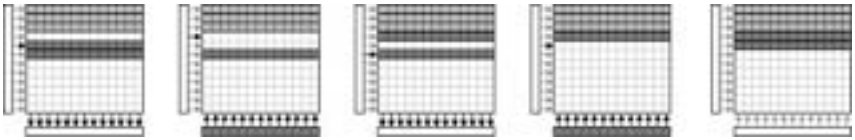


Figure 3.7: Row based defragmentation as proposed by Compton [185]. Configuration data and state of each single row is first read into the row buffer, then stored into the new location.

Figure 3.7 shows row-based defragmentation, a concept proposed by Compton et al. [185]. Their architecture is a homogeneous FPGA with homogeneous and virtual IO to enable moving of logic. As 2D defragmentation is an algorithmically complex problem, the authors focus on 1D defragmentation instead.

Through introduction of a row buffer, reconfiguration of the device always happens one row at a time. In addition, the current configuration and state can be read back into the row buffer. When multiple applications have been started and stopped, some empty rows may reside between applications. To free up this space into a larger region, a defragmentation system is introduced. This system moves applications on the FPGA by copying them into the row buffer and then to their new location row-by-row. A slightly modified and extended variant of this architecture has been presented by Brebner et al. [186]. The author's architecture enables relocation without a host computer, implementating all the relocation logic in hardware. For this, they extend the Compton architecture's row-based defragmentation to quickly find free rows in hardware. Based on this free-space search, they implement on-chip compaction of running FPGA tasks.

In 2004, Koch et al. introduce hardware extensions for preemptive task scheduling and defragmentation [187]. FFs can already be configured as part of the bitstream, as those can have defined initialization values. Freezing the current application state can therefore be realized without hardware extensions, but it requires reading back the complete configuration memory, including LUT configuration which does not change. The authors therefore introduce an additional scan chain which only connects the FFs in the logic cells. This way, a smaller amount of data needs to be saved and restored. The authors also present an extension to enable two-dimensional defragmentation through shifting.

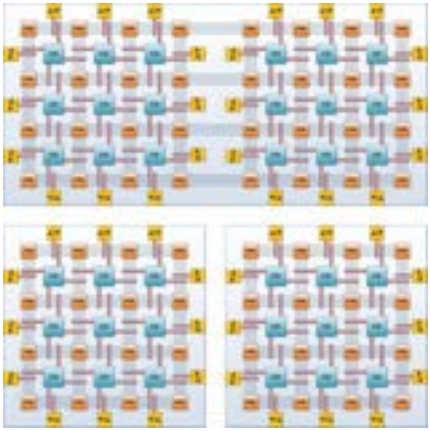


Figure 3.8: FPGA core fusion as proposed by Figuli [188]. Multiple independent FPGA cores can be combined to place larger tasks.

Figure 3.8 shows an approach to task mapping presented by Figuli et al. in 2011 [188]. This FPGA architecture is divided into multiple identical cores, which behave independently. Application size is limited to be multiples of the core size and applications are mapped to the cores using a controller. When an application is too large for one core, the system supports core fusion: In core fusion, the IO peripherals on one side are disabled and interconnect switches instead connect to the next core. To enable routing of IO pins in such an architecture, the authors also provide virtual IO adapters. Defragmentation can be performed on a coarse level using this system: The architecture supports freezing and restoring state of a complete core. This allows applications to be moved between cores, enabling defragmentation on core level.

A completely different approach to fine-granular reconfiguration presented by Bozzoli et al. in 2019 is shown in figure 3.9 [189]. This architecture allows reconfiguration on single LUT level through introduction of the "Reconfigurable Multipotent Cell", ReM. This cell combines logic, memory and reconfiguration: It enables distributed reconfiguration, as each cell can trigger reconfiguration of its neighbors. Unfortunately, the authors have not demonstrated how existing, regular applications can make use of such a novel architecture.

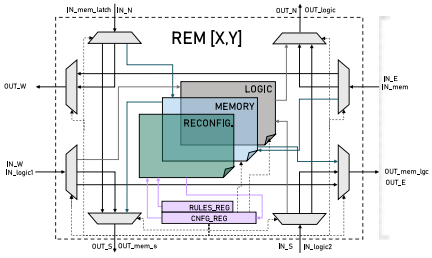


Figure 3.9: Basic cell of the distributed reconfigurable architecture by Bozzoli et al. [189]. Each cell can trigger reconfiguration of its neighboring cells.

An extension for relocation support on commercial FPGAs has been presented by Adetomi et al. [190]. Its main contribution is the use of clock lines to communicate between applications. As clock lines are not part of static wiring in Xilinx architectures, this can reduce routing issues when relocating applications. This approach is mostly useful when an existing architecture must be used and can not be modified. When novel FPGA architectures are

designed, it is possible to include dedicated wiring such as Network-on-Chip (NoC) instead.

3.4 PVTA Compensation

Traditional PVTA Suppresion Methods As explained in the previous chapter, various PVTA sources affect the propagation delay of logic gates, including the reconfigurable logic elements in FPGA. Similar effects also affect the propagation delay of wires and the FPGA interconnect in general. When working with average or typical process values and the corresponding propagation delays, circuits can fail: When one of the critical paths contains gates with worse delay than assumed in EDA STA, the circuit may actually cause setup time violation, even though this was not visible in STA. Commonly used solutions include speed binning and worst corner design: In speed binning, it is accepted that some produced ICs may fail at their nominal clock frequency. Therefore, each single IC is measured either statically after fabrication or dynamically at runtime. Depending on the results, it may be used with clock frequencies which are lower than the nominal frequency. Similarly, ICs with lower propagation delays may be used at a higher frequency. Such an approach however requires detailed measurements of the IC, which often needs test structures requiring additional area on the chip. An alternative to this is worst-corner design: In this case, STA is performed with the worst case values for all possible variation sources. This includes process, voltage and temperature variation as well as aging. As process variation is largely random, the worst case combination of all sources is unlikely to occur, especially all the time and affecting the total IC area. Worst-case corner design therefore introduces overly pessimistic guard bands [191]. This leads to additional design effort in otherwise unnecessary circuit optimization. It also means that circuits often are clocked at a lower frequency than theoretically possible. For FPGAs, this issue occurs in exactly the same way. A commonly proposed solution to improve this situation, is Statistical STA (SSTA) [51]. SSTA does not work with single values for propagation delays, but operates on statistical distributions instead. As a result, SSTA yields a distribution of circuit delay paths. This allows to estimate the amount of circuits which can operate within a certain performance region. It therefore allows to shape the performance distribution during design, which increases the yield after binning.

Applicability for FPGAs Whereas FPGA suffer from performance variation due to PVTA just like ASICs, the SSTA solution is unfortunately not easily

applicable. An approach was described in [192], but it suffers from various usability issues: In FPGAs, the critical paths are not known during design and manufacturing of the FPGA itself. SSTA therefore has to be performed on the user application. However, at this time after FPGA ICs have already been manufactured and sold, a yield optimization is difficult. In essence this would mean that a produced bitstream works only on some FPGAs. Whereas this is already undesirable for FPGA users, it would also be more difficult to realize binning for FPGA bitstream programming. To make matters worse, whether a bitstream operates properly on some FPGA IC is also highly dependent on the placement. An FPGA IC which works for one version of the design might not work anymore when minor changes in the design demand a rerun of the EDA flow and yield a changed placement. Design synthesis could then be repeated until the design is found to work on an FPGA, but again, this only optimizes for a single IC. All in all, SSTA approaches therefore are less suited for FPGAs: As binning becomes infeasible due to previously manufactured devices and because of device reuse using reconfiguration, the utility of statistical approaches is limited.

FPGA Specific PVTA Handling Solutions to mitigate PVTA effects on FPGA therefore look different from ones used for ASICs. They are usually making use of the reconfigurability of the FPGAs in some way. Most of the time, solutions are dynamic, adjusting various aspects of the implemented circuit during circuit operation. A few completely static solutions, which do not perform any operation at runtime, but only modify the EDA flow, have been proposed. Usually those rely on a previous characterization of the target IC and make use of that information during placement. Dynamic solutions also rely on such a measurement of device characteristics, but perform these online. In addition, they also include some aspects to compensate the PVTA effects during runtime. In some cases, solutions also characterize the user application: This allows to not only compensate PVTA to reach nominal operating conditions, but to also accept reduced performance in areas where the user application has sufficient slack and can tolerate larger delays. Compared to nominal conditions, this concept allows for further energy saving optimization. In the following, publications covering those individual aspects will be presented. Publications combining these topics to yield a full compensation system similar to the one in this thesis will be introduced at the end of this section.

Critical Path Identification

Identification of critical paths in an application design can be obtained in two ways: One approach is special handling or detection of paths during synthesis and implementation in the EDA tools. The other is to detect critical paths or violation of their timing constraints directly in a circuit.

In 1990, Kaenel et al. [193] presented a system for global voltage reduction. To estimate the available timing slack of the critical path, they built a circuit which emulates this path. This "Equivalent Critical Path" is measured and depending on available slack, the global power supply of the circuit is adjusted. To build this equivalent path, the authors identify and extract the critical path using EDA tools.

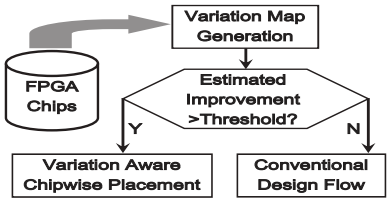


Figure 3.10: Variation aware chipwise placement as proposed by Cheng et al. [194]. Placement uses a chip specific variation map to optimize critical path location.

A different approach was taken by Cheng et al. [194]. Instead of regulating performance dynamically, they characterize each FPGA IC individually: Using test circuits configured onto the FPGA, they obtain performance variation maps for some defined regions in their ICs. They then integrate this information in the EDA flow as shown in figure 3.10 to customize the placement step for each FPGA IC. As the process variation is known, critical paths can be placed into regions with smaller propagation delay. Overall, this allowed to achieve a 12 % improved performance. The main drawback of the system is that the application bitstream has to be regenerated for each FPGA IC, which is a time intensive operation.

A related approach was taken by Ghosh et al. [195] for low-power design. In such systems, optimization such as gate sizing can increase the number of critical paths. In order to compensate process variation, the authors therefore describe a way to reduce critical paths in circuits: Using modified EDA tools, they customize the Shannon Expansion step in a way to shape critical paths.

Using this, they then confine the critical paths to certain logic cofactors and isolate them. They then switch to two-cycle operation at runtime when those critical paths are activated. A different solution was given by Ebrahimi et al. [196]. Their compensation approach is based on the common critical path replica idea, but the selection of the critical path is special: Whereas there have already been publications describing how to select those critical paths, which are most likely to be affected by aging, for ASICs, the authors adapt this idea for FPGAs. As the transistor-level design of commercial FPGAs is not publically known, ASIC models can not be used and the authors derive an FPGA model, taking e.g. static and dynamic stress into account.

Elgebaly et al. noted, that the critical path in a circuit might change over time due to PVTA [197]. They therefore propose to track a changing, emulated path. This emulated path is designed to exhibit the same behavior as the actual critical path under all PVT conditions. Their path emulation covers both interconnect and combinational logic.

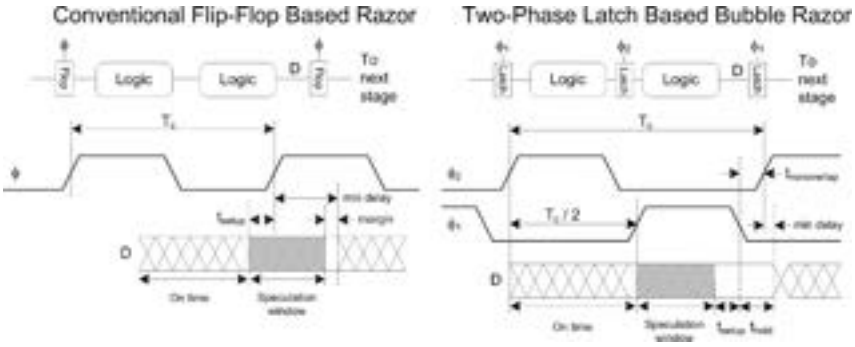


Figure 3.11: Razor (left) and Bubble Razor (right) timing violation detector [198]. Special hardware monitors in all paths detect violations of setup time constraints.

A different style of compensation systems has been published based on the Razor system. Figure 3.11 shows a comparison of Razor and similar systems and the Bubble Razor system. Instead of detecting critical paths ahead of time, razor like systems extend combinational logic to include detectors for setup time violation. The original razor system detects such a violation by checking whether a transition on the data signal occurs shortly after a clock transition. Razor therefore however adds a constraint on the minimal hold time within a circuit. Bubble Razor on the Other hand introduces two latches in between combinational paths and uses them to detect changes within

the combinational logic at invalid times. When integrated into PVTA compensation systems, those systems operate the application at almost-failure frequency. When a failure occurs, the system needs to invoke restore logic to replay the operation.

A novel approach for critical path replica in ASIC was recently published by Miro-Panades et al. [199]. It operates the application circuit in two phases: In the first phase, it detects the available timing margins. In the second phase, it actually operates the application normally. The remaining slack is estimated using a timing fault sensor, emulating critical paths. Unlike previous publications, this sensor is not fixed-function but configurable. It can therefore adapt to different application designs.

Device Characterization

Apart from detecting the critical path in applications, PVTA compensation systems usually also characterize the device in some way. Often, a circuit is characterized using a replica of the critical path in the application. In other cases however, process, voltage, temperature variation and aging are measured directly. Most of those sensors proposed for FPGAs are based on delay measurements. As all of those physical values are correlated with delay, designing sensors for one of them requires compensation of the others. The following section will present a short overview of publications on those topics.

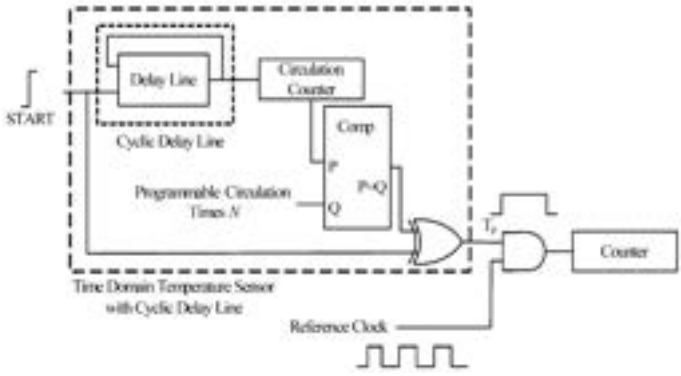


Figure 3.12: Fully digitally temperature sensor as presented by Chen et al. [200]. All signal processing is performed in the digital domain.

Yu et al. describe a way to measure process variation, which they use for a variation aware design approach [201]. They measure the delay of a ring-oscillator based circuit to estimate both LE and wire delay. The design is implemented for FPGA and only uses resources available in standard FPGA architectures. Gnad et al. analyzed voltage fluctuations in commercial FPGAs and used a similar time-to-digital sensor to measure the voltage [69]. They placed multiple sensors over the whole FPGA area and analyzed temporal and spatial effects depending on the application design.

Various publications have used similar sensors for temperature sensing. One of the first such temperature sensors was described by Chen et al. and is shown in figure 3.12. It consists of a cyclic delay line and uses the system clock as a time reference. The whole design fits into 140 LEs and achieves an error between −1.5 K and 0.8 K With 260 µs conversion time, the sensor can be used for dynamic measurements. Franco et al. describe a similar, ring oscillator based temperature sensor for Virtex 5 [202]. They explicitly discuss the voltage sensibility of the sensor and how to address it. Happe et al. use 144 similar sensors on a Virtex 6 architecture [203]. Using this dense set of sensors, they derive a thermal model which they use for thread mapping onto CPU cores. For characterization and dynamic modelling, a set of heating elements on the IC produces temperature gradients which are measured using the sensors. Calibration is performed using a temperature measurement diode which is integrated in the FPGA.

Aging and process variation is monitored similarly. Agarwal et al. describe a system for circuit failure prediction [204]. They mostly focus on PMOS aging and NBTI effects and implement and aging detector integrated into a FF. Huard et al. use aging monitors to perform adaptive wearout management [205]. They argue that replica elements work well for global effects, but less so for local effects. In-situ monitors integrated within the application design and embedded in the physical area of those application parts enables more direct monitoring of delays of real paths. Another ring oscillator based system to detect aging was presented by Sengupta et al. [206]. Their publication focuses on BTI and HCI effects and explains the required sensor circuit calibration.

A sensor directly developed to measure delay was presented by Zick et al. [207]. They implemented an FPGA based sensor node in 8 LUTs. The sensor element was therefore able to fit within a single Virtex 5 CLB. For usage scenarios, they explain how the sensor can measure delay, temperature and IR drop or voltage variation.

PVTA Compensation Architectures

Based on critical path identification and device characterization, various solutions for management and compensation of PVTA have been proposed. Most systems target ASICs and are often not directly applicable for FPGAs, as the critical path is not known at IC manufacturing time. Some ideas presented in these systems are however general and have been adapted to FPGAs.

An analysis focusing on circuit and transistor level aging solutions was presented by Alam et al. [208]. The authors suggest considering aging effects already in the design of circuits. Transistor sizes and other physical parameters would then also be chosen according to aging requirements. Unlike this static compensation approach, Gupta et al. presented TRIBECA, a dynamic compensation system mostly targeting processor systems [209]. The authors provide an analysis of variation sources, then propose a system for error detection and compensation: In order to correct errors, they introduce an error detection unit and operation replay support. As their solution is local, it can also handle spatial variation effects. The proposed system is tightly integrated with the CPU architecture presented and can therefore not be used for general purpose applications. When it comes to more general compensation systems for ASICs, various systems have been proposed over time. An overview of those systems can be found in recent survey publications [87, 210, 211]. Khoshavi et al. provide an overview of measurement and monitoring approaches. They compare static guard banding and dynamic approaches. Static approaches include design aware balancing, which balances critical paths according to aging criteria and other EDA based solution such as the one by Alam. Dynamic approaches presented focus mostly on voltage and frequency scaling. Another summary, focusing mostly on CPU systems, was published by Mittal et al. [210]. They mostly include solutions which address process variation, including block selection and error management techniques. Compared to other publications, they also summarize works which investigate scheduling of tasks on CPU under process variation. A similar, but more extensive survey is given by Rahimi et al. [211]. It focuses on processor systems as well, but covers variability mitigation from circuit to software level. In the following, some individual works which are closely related to the work in this thesis will be presented in more detail.

Adaptive Body Biasing Similar to RFET technology, BB in SOI devices enables fine-grain adjustment of transistors threshold voltages V_{th}. As SOI

technology is readily available in commercial manufacturing processes, various systems using BB at different granularity have been proposed. When the amount of BB is adjusted at runtime, these systems are called Adaptive Body Biasing (ABB) systems. Figure 3.13 shows such an ABB based compensation system, as presented by Tschanz et al. [212]. This early work focused on die-to-die chip variation and as such uses BB on a chip-wide scale. The system is dynamic, in that it finds the optimum BB voltage for each chip during runtime of the application. To determine whether the circuit speed needs to be improved or whether the circuit can get slowed down, the system constantly monitors a replica of a critical path. It then adjusts the bias of a circuit block accordingly. In the test chip shown in the figure, this circuit block is not a complete processor, but only consists of some extracted processor paths for simplicity. The authors also propose an extension to use multiple instances of this system on one IC to compensate intra die variations. The granularity of this ABB approach is therefore at chip level, the proposed extension operates at region level.

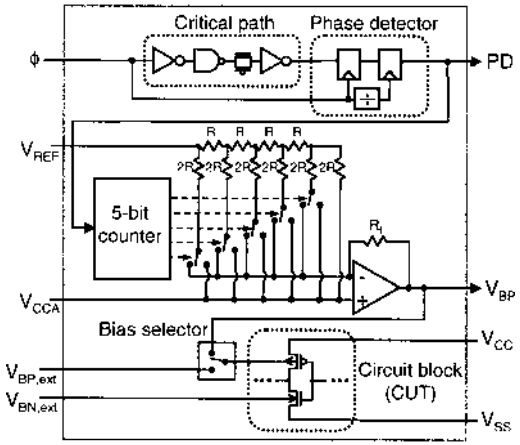


Figure 3.13: Adaptive Body Biasing test chip presented by Tschanz et al. [212]. The chip contains a critical path replica used for characterization as well as a circuit block, which simulates a CPU.

A similar system proposed by Teodorescu et al. is explicitly tailored to mitigate process variation for processors [213]. This work considers fine-grain BB, although in this case granularity is also only at region, not at transistor scale. It also combines the BB system with Dynamic Voltage and Frequency Scaling (DVFS) of the processor, to reduce power consumption even more. In addition,

the authors propose an in-chip V_{th} variation model for the manufacturing process they use.

Yet another ABB system was proposed by Mauricio et al. [214]. As shown in figure 3.14, this work again devides a chip into regions, where each region contains one biasing system. These systems track runtime variations in the circuits in a region using a delay sensor, which is not further specified. The main focus of the paper is on an area efficient bias generator for the individual regions. Using a charge pump in this bias generator allows generating bias voltages above and below the power supply voltages. Here, the authors claim their approach to require 70 % less area than a comparable system using Digital to Analog Converters (DACs).

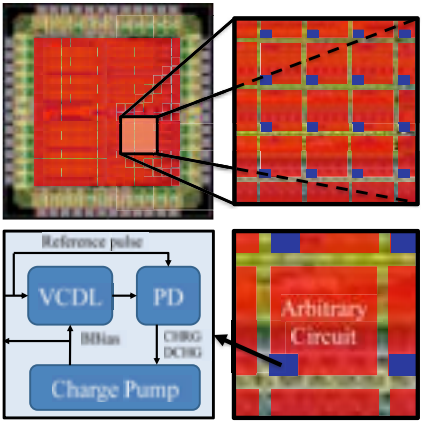


Figure 3.14: Body Biasing Islands in a compensation system proposed by Mauricio et al. [214]. Each island contains the compensation circuit, which consists of a delay detector and charge pump.

Whereas previous systems have been proposed for general purpose ASIC application, a system closer to solutions for FPGAs has been proposed by Matsushita et al. [215]. This work focuses on Coarse Grain Reconfigurable Arrays (CGRAs) and primarily aims to reduce leakage power using BB on SOTB technology. To enable BB, the authors first divide the CGRA into multiple regions of a certain size. In CGRAs, just like in FPGAs, some details about critical paths may only be known after configuration with a user application. The authors therefore determine the biases for each region as part of the application EDA flow, after the CGRA user application has been placed. As there is no runtime management system, the authors' approach is completely

static. To find the optimal size, the grain size is then varied in a design space exploration. The authors claim to achieve a 40 % reduction of static leakage using this approach, at an area overhead of 6 %.

FPGA Solutions In addition to the generic compensation systems presented so far, some specialized FPGA systems have been described in literature as well. Chow et al. proposed one of the first systems to realize Dynamic Voltage Scaling (DVS) on commercial FPGAs. Their proposed system uses the delay sensor in figure 3.15 to determine the current speed of the FPGA logic. It then scales the supply voltage globally for the whole FPGA to reduce power as far as possible. The system was mainly meant to be used to reduce power usage, not to counter PVTA. As such, there is only one sensor for the whole IC and there is no local adjustment of voltages. The authors therefore claim a 54 % reduction in power.

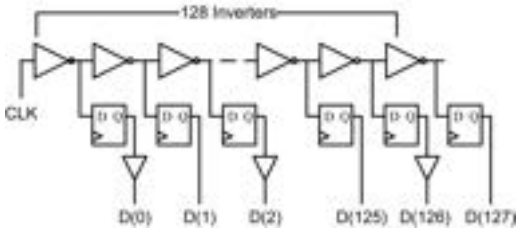


Figure 3.15: Logic delay measurement circuit proposed by Chow et al. [216]. Registers are clocked using the same signal as the data input of the inverter chain. The falling clock edge will trigger signal changes in the inverter chain, which will be captured by the FF.

A different goal was pursued by Nabaa et al. [217]: Their system primarily targets compensation of process variation, requiring a more fine-grain, localized measurement of delays. The main novelty in this publication is the fact that the characterizer circuit shown in figure 3.16 is placed only once on the IC. To measure delays in different areas of the FPGA, the authors iteratively route the signal to be measured through different LEs on the FPGA. They then use BB to slow regions which positive slack, achieving a 3-times reduction in leakage power. As the system can only characterize LE as long as they are not in use by the application, the authors propose to run the characterization phase once, before starting the user application. This semi-dynamic compensation can therefore not address variation during application runtime, including temperature variation and voltage variation. Long-term aging can be compensated, if the user application is stopped periodically.

Hioki et al. focus on reduction of leakage power in their SOTB test chip [218]. Using a fine granular approach with 57 V_{th} domains per FPGA region, they achieve up to 50 times reduction in leakage. Due to this fine-grain approach, they however report area overhead of 26 % and addition 10 % area which needs to be unused for separation. There is no runtime measurement of path delay. Instead, the authors modify their FPGA tools to adjust the bias for all power domains after placing the user application. As such, not only the critical path but all paths individually are considered. The resulting circuit characterization is used to tune the biases in the power domains of the device. As this approach is static, it can not address temperature and voltage variation, as well as aging. It could compensate process variation, if the chips are characterized before configuration. This option was however not explored by the authors.

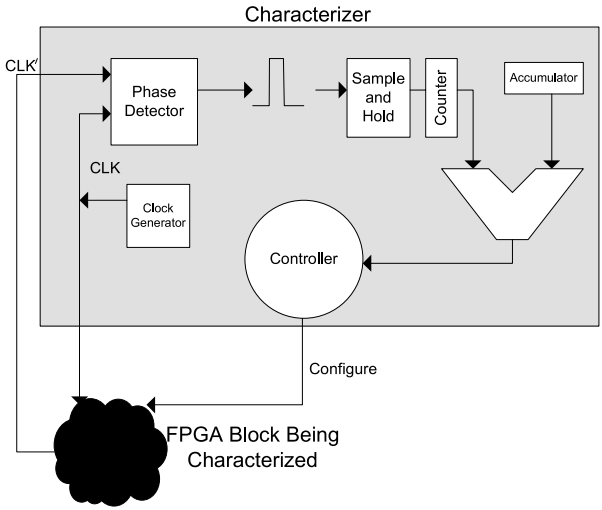


Figure 3.16: FPGA block characterizer used in the system proposed by Nabaa et al. [217]. A clock signal is routed through the block to be characterized to a phase detector. The phase detector also gets a direct connection to the clock, so that the measured phase difference characterizes the delay of the FPGA block and wire delay.

A slightly different approach has been taken in the thesis by Burmester Campos [219]. His work focuses on EDA algorithms and proposes variation aware optimization algorithms. These approaches are validated on the PAnDA architecture, an FPGA with reconfigurable transistor widths.

Focusing on FPGA runtime again, Maragos et al. proposed a PVT system for commercial FPGAs[220]. It places dozens of delay sensors onto an FPGA embedded into the user application. The sensor then uses a logic chain to measure the delay and compares it to a configurable, acceptable delay. This acceptable delay is predetermined in the EDA flow for the user application. This information is then used to dynamically adapt the global power supply of the FPGA. The authors achieve up to 27 % reduction in power, at a resource overhead of 1.6 %. The main limitation of this approach is that DVS can only happen globally in commercial FPGA.

Apart from these compensation systems presented in academia, commercial FPGA vendors have started to make use of BB in SOTB FPGAs. In 2019, Lattice Semiconductor presented the Crosslink-NX FPGA, the first based on the Nexus platform [221]. These devices are based on a FDSOI process and enable BB on a chip-wide level. The devices also only allow for two performance levels to be selected. So far, no dynamic management system has been proposed, further limiting the use for PVTA compensation.

3.5 Power Management Techniques

The previous section has included some systems, which do not only focus on PVTA compensation, but also feature power reduction. As the mechanisms for performance measurement and adjustment are the same in both cases, solutions are often similar as well. Previously presented publications were however more focused on technology-level details, such as BB in SOI devices. The power management schemes discussed in this section focus more on system-level aspects.

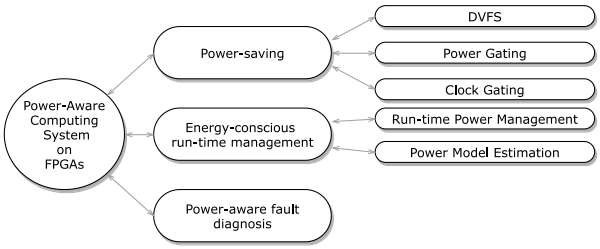


Figure 3.17: Classification scheme for power aware FPGA architectures and techniques as proposed by Akgün et al. [222].

Figure 3.17 shows a classification of such power aware FPGA techniques, as recently proposed by Akgün et al. [222]. The authors distinguish between three main topics: Power-saving techniques, runtime management and fault diagnosis. In the following, we will mainly consider the first topic, which includes system architecture solutions. The second aspect mostly focuses on application specific solutions which are of less relevance to this thesis. Error detection and recovery is also not relevant to this work.

EDA Techniques In addition to hardware approaches, power saving approaches have also been introduced in EDA tools. Early works by Sutter et al. have focused on more power efficient FSM state encoding [223]. Apart from binary and one-hot encoding, they also analyzed two-hot encoding and a custom scheme which is intended to minimize switching activity. Using the best technique reduces the power consumption by 57 %. Whereas this approach considers one small detail, more systemic EDA power saving solutions have been proposed by Singh et al. [224]. Their work considers clustering and placement under power reduction constraints. Using a specialized logic clustering algorithm enabled power reduction by up to 13 %. To achieve that, the authors proposed a new algorithm for routeability estimation between logic cluster block. Gayasen et al. introduced power regions in their FPGA architecture [225]. In order to reduce leakage power, they added sleep transistors to all those regions and turn of unused ones. To improve power reduction results, they also customized the placement phase in their EDA tools. Using a vertical or horizontal direction based placement allowed to maximize the number of unused regions. Further details on the placement algorithm were not given by the authors. They also quickly note that their region based power management, essentially a power gating scheme, can also be used at runtime. This however requires explicit activation or deactivation of regions by the user application.

Power Gating Many other publications have focused on clock gating on an architectural level. In 2007, Tuan et al. described a low-power version of the Xilinx Spartan 3 architecture [226]. Apart from various general optimizations, such as lowering the core voltage and reducing leakage in configuration SRAM, they also include a power gating scheme. As SRAM leakage has already been optimized, the authors do not power gate the configuration storage. Whereas power gating was originally mostly used for unused logic, keeping configuration storage also enables a standby mode in the architecture: In this mode, the logic in the design can be power gated dynamically and powered on later on again. The authors propose this could be used with a user-provided central power controller, which is itself never power gated.

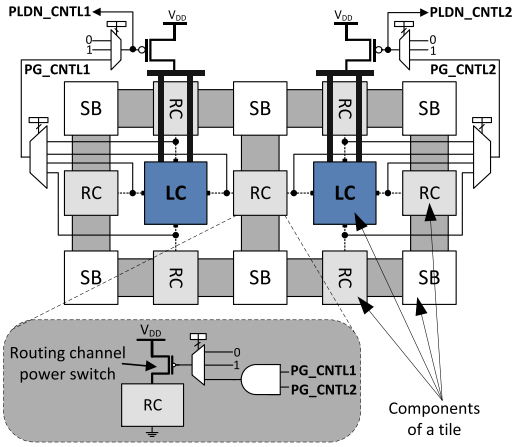


Figure 3.18: Fine grain power gating for FPGA as proposed by Bsoul et al. [227]. All logic blocks (LC) and neighboring routing channels can be power gated individually.

As the area overhead of power gating solutions can be significant, Bsoul et al. carried out a design space exploration of various sizes of gating regions [228]. The authors use a dynamically power-gated design and study the area vs. power saving trade-off. For their architecture, they find tiles of 3x3 CLB to be most efficient, reducing leakage by 40 % at an area overhead of only 1 %. The proposed architecture allows runtime adaption as well, but the power controller needs to be implemented by the FPGA application designer. In a follow-up publication, Bsoul et al. extend the architecture [227] to allow for fine-grain power gating, as shown in figure 3.18. Apart from the finer granularity for SBs, the authors also introduce a modified EDA tool flow. To introduce a power domain aware routing algorithm, they modify the cost function of VPR's router to penalize routing through regions which do not belong to the fan-ins or fan-outs of the net. This is intended to reduce routing through otherwise unused tiles, making more tiles available for power gating. With these changes, the authors managed to reduce leakage by 83 % in total.

Whereas Bsoul et al. modified CAD algorithms, a different approach is taken by Seifoori et al. [229]. In an attempt to more efficiently select the multiplexers which are part of one power region, the authors first place and route a set of benchmarks for a baseline FPGA architecture. They then perform k-means clustering and a custom utilization similarity clustering to find multiplexers

and SBs which are commonly used together. Instead of modifying the CAD tools, this approach shapes the power-gating regions according to the results of the CAD tools for certain benchmark circuits.

DVS and DVFS Another commonly used power management technique on FPGA is DVS. As previously mentioned, one of the first such systems was the setup by Chow et al. [216]. It used an off-chip voltage controller and a commercial FPGA to realize global DVS for the complete IC. A more recent publication by Nabina et al. replicated the idea on a Virtex 5 architecture [230]. Using a LEON3 processor, the authors realize an Application Specific Integrated Processor (ASIP), where some FPGA resources are used to implement a reconfigurable part. For the processor, the authors also implement dynamic frequency scaling. For the reconfigurable module, the authors use an adaptive voltage scaling scheme, measuring the logic delay using a delay measurement circuit.

Figure 3.19 on the next page shows a DVS system proposed by Nunez-Yanez [231]. This system also uses an external power regulator, but uses an in-situ detector to determine whether the voltage should be scaled up or down. The detector consists of two FFs, a main FF and a slow FF. The main FF directly replaces a FF in user application logic, whereas the slow FF connects to the same input, but with an additional delay. When propagation delay increases due to voltage reduction and there is a setup time violation, this is first detected in the slow FF. As the main FF still operates correctly, the user application will not be affected. The DVS controller uses the information about failing slow FFs to stop further reduction of the supply voltage. To integrate these detectors within the application logic, the author has developed an EDA tool to automatically replace FFs in critical paths.

Another DVS system was proposed by Ahmed et al. [232]. This system changes the supply voltage according to a calibration lookup table. This table contains the minimum operation voltage for the circuit at various operation points. It is obtained through a custom EDA tool which automatically generates a calibration bitstream based in the user application. The calibration bitstream then has to be programmed once to each FPGA IC to obtain the device specific calibration. In [233], this idea was extended by the authors to use multiple calibration bitstreams.

Recent systems realizing DVFS have also been proposed by Levine, Zhao and Taka. Levine's system performs online slack measurement using shadow registers. These registers have to be added to critical paths using a custom EDA tool [234]. Apart from enabling global power scaling, the proposed system

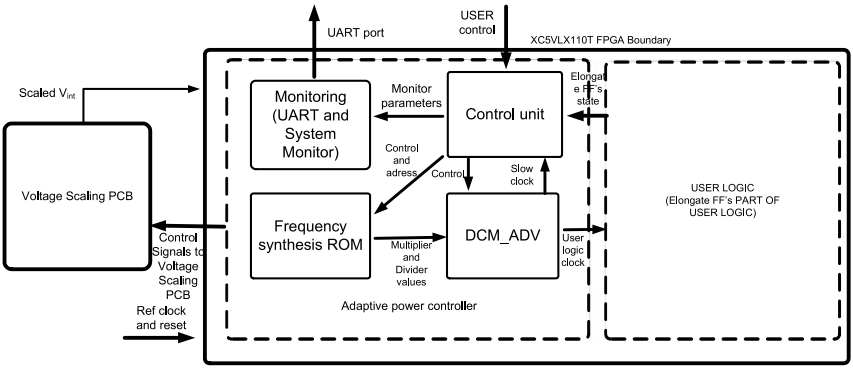


Figure 3.19: DVS system for FPGA proposed by Nunez-Yanez [231]. Voltage scaling is performed off-chip, as the used commercial FPGA does not provide any on-chip voltage configuration.

also scales one global clock. Zhao's system performs offline self-calibration instead, finding the frequency and voltage limits for an application before it is actually active [235]. To realize this, the authors extract the critical paths of the user application and build a calibration design based on these critical paths. The custom FPGA bitstream then performs the self-calibration step. The authors also scale power and one global clock and evaluated their design using an Finite Impulse Response (FIR) filter. Taka's design focuses on one specific application only, a RISC-V processor[236]. To detect whether the processor is operating correctly at a certain frequency and voltage, the authors slowly increase the frequency of the processor for each tested voltage. The authors then run a test application at all frequency steps and verify the applications output. If the application output is incorrect, the system assumes the frequency has been increased too much. As the system requires running software, the concept is limited to processors.

Other publications attempted to reduce power on FPGAs using a dual-VDD design [237]: In such architectures, voltage scaling is not global and each CLB can be programmed to use a high or a low supply voltage. For this, Gayasen et al. changed EDA tools to assign CLBs to one of the supply rails. After evaluation various techniques, one was found to provide an average power reduction of 61 % in the MCNC benchmarks. Historically, a few such dual-VDD designs have been explored. With the availability of SOI technology, fine-grain techniques have focused more on BB, as explained in the previous section.

3.6 Synthesis for Reconfigurable Cells

Whereas various ambipolar reconfigurable cells have been described in literature, there are few publications focusing on EDA flow integration to use these cells in FPGA. As most of these cells are experimental and have not been manufactured in large-scale FPGA devices, there was so far no demand for EDA tools. Nevertheless, a few publications have proposed solutions for ambipolar reconfigurable cells or for ULMs and logic macro cells, which need similar EDA flows. For DGCNTFET, Zukoski et al. have described not only their logic cell, but also gave a quick introduction to the EDA flow they use [238]. They use ABC for optimization and tech mapping. For the tech mapping itself, they provide a custom library of gates which represent the functions that can be realized by the reconfigurable cell. The place and route steps for use in a final FPGA architecture are not further explained.

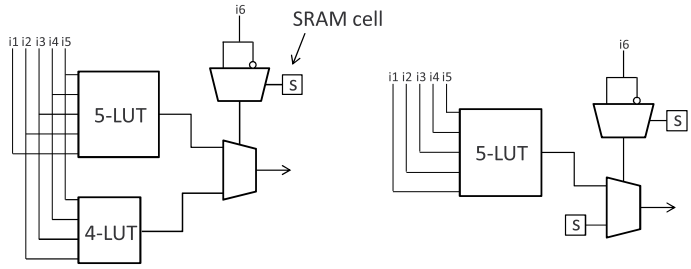


Figure 3.20: Asymmetric 4+5 LUT (left) and extended 5 LUT (right) by Anderson et al. [239]. The asymmetric LUT was developed for trimming input optimization, the extended LUT for gated input optimization.

Publications on EDA for ULM and macro-cell based reconfigurable architectures have been proposed in the late 1990s and early 2000s. Due to their age, these publications use older synthesis approaches, based on SOP representation. For example, Lin et al. use the SIS mapper for their reconfigurable architecture [240]. They first map a set of benchmarks to a generic 3 input LUT architecture. They then analyze which functions are commonly used and derive a set of custom ULM modules for this information. Synthesis for these ULM modules is not explained in detail. Another EDA flow has been presented by Cong et al., focusing on synthesis for k/m cell reconfigurable logic [241]. k/m cells are PLA-like reconfigurable cells and can be described by the number of inputs, k, and the number of product terms that can be realized, m. Unlike a PLA, k/m cells have always exactly one output signal. Optimization and mapping for this cell has been implemented as a custom

algorithm in SIS and verified using the MCNC benchmarks. For place and route, VPR has been used.

Figure 3.20 on the preceding page shows a special reconfigurable cell derived by Anderson et al. [239]. The architecture has been developed to fit a specific, custom EDA flow efficiently: The authors introduce the notion of trimming inputs and gated inputs. A trimming input an input, which when used in a Shannon decomposition yields a co-factor with more than one variable less than the original function. When applied to functions with 6 inputs, such a decomposition can be realized by the reconfigurable element on the left-hand side of figure 3.20. A gating input is a special trimming input, which yields one constant cofactor in the Shannon decomposition. Such a decomposition has been realized with the cell in the right-hand side of the figure. The authors observe, that such special inputs can be found quickly in AIGs by finding non-inverting paths. The mapping algorithm can therefore be implemented in tools like ABC.

Apart from special EDA solutions for specific architectures, literature has also proposed solutions for hybrid architectures. These architectures do contain LUTs, which are primarily used when the hard logic macro can not represent a function. Such an approach was used by Hu et al. [242]: The authors first analyze which functions are commonly used in a set of benchmarks and then design a macro cell to realize those. They then propose an FPGA architecture using both these cells and LUT. For EDA, they use a standard cut-based mapping technique for LUTs first. They then determine which LUTs in the netlist could be replaced by a macro cell. Furthermore, they use a modified packing and repacking scheme to recover area. This step consists of realizing macro functions using a LUT instead of a macro cell, if many free LUT are available.

A similar approach to combine LUTs and ULMs, also called universal logic generator, has been presented by Luo et al. [243]. The publication focuses on the description of the complete EDA flow, as shown in figure 3.21. It uses ABC for optimization and technology mapping to standard 4 input LUTs. In a post-processes pass, the netlist is modified to replace LUT instances with ULMs when possible. Packing, placement and routing is performed in VPR as usual. The authors however note that a LUT can also realize the ULM function, but not the other way round. They therefore specify LUTs as multimode cells in the VPR architecture, enabling VPR to also place logic mapped to ULMs into LUTs.

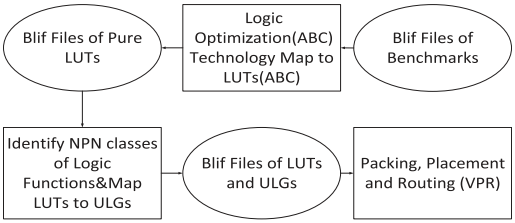


Figure 3.21: EDA flow for a hybrid FPGA consisting of LUTs and universal logic generators (ULG) [243]. Applications are first mapped to LUT as usual, then eligible LUT are replaced by ULGs.

3.7 Summary

Table 3.3 shows a quick summary of the related works which are most closely related to this dissertation. Table columns show: Whether a non-LUT LE is used, whether moving and freezing of data is supported, whether power reduction is supported, whether PVTA compensation and measurement happens once or repeatedly, whether compensation is local or global and whether a delay measurement system is used and if it reuses existing logic. This dissertation combines three aspects, which has not been done in any related work: RFET standard cell usage, system level FPGA power management and PVTA compensation and low level reconfiguration aspects. As there is little overlap, the first aspect was not included in the table, but is explained in section 3.1. As the table clearly shows, no previous work has combined the low-level reconfiguration system with PVTA and power reduction techniques. Combining these aspects is the main novelty of this thesis, as it allows for fully dynamic, local PVTA with limited resource overhead. And whereas some works do use FPGA resources for measurement, none is able to share the resources with an application. To the best of the author's knowledge, the transparent logic invasion concept has not been proposed anywhere previously.

Source	Non-LUT	Reconfiguration		Power	PVTA			Characterization	
		Move	Freeze		Static	Dynamic	Local	Measure	Reuse
[105]	✓	✗	✗	✗	✗	✗	✗	✗	✗
[163]	✗[1]	✗	✗	✗	✗	✗	✗	✗	✗
[164]–[166]	✓	✗	✗	✗	✗	✗	✗	✗	✗
[185]	✗	✗[2]	✗	✗	✗	✗	✗	✗	✗
[186]	✗	✓	✗	✗	✗	✗	✗	✗	✗
[188]	✗	✓	✓	✗	✗	✗	✗	✗	✗
[193]	✗	✗	✗	✓	✗	✓	✗	✓	✗
[194]	✗	✗	✗	✗	✓	✗	✓	✗	✗
[195]	✗	✗	✗	✓	✓	✓	✓	✓	✗
[201]	✗	✗	✗	✓	✓	✗	✓	✗	✗
[69]	✗	✗	✗	✗	✓	✗	✓[3]	✓	✗[4]
[202]	✗	✗	✗	✗	✓	✗	✗	✓	✗[4]
[216]	✗	✗	✗	✓	✗	✓	✗	✓	✗[4]
[217]	✗	✗	✗	✓	✓	✗	✓	✓	✓[5]
[218]	✗	✗	✗	✓	✓	✗	✗	✗	✗
[220]	✗	✗	✗	✗	✗	✓	✗	✓	✓[4]
This	✓	✓	✓[6]	✓	✓	✓	✓	✓	✓

Table 3.3: Overview of related works with most similarity to this thesis. RFET works presented in section 3.1 do not include any system level FPGA features, so they are not shown here.

[1] Uses MRAM-based LUTs. [2] Requires a host computer. [3] Local analysis, but not compensation. [4] Uses reconfigurable logic, but doesn't share with application. [5] Only once, at application startup. [6] Partially, implicitly during movement.

Part II

PARFAIT Architecture

This page intentionally left blank

Chapter 4

Overall System

In the following chapters, state-of-the-art PVTA compensation and power saving measures in FPGAs will be investigated to be used with ambipolar transistor devices. To keep the overall system independent of technology, it will also be adapted and evaluated for a state-of-the-art SOI technology. This chapter will first introduce a standard FPGA architecture for PARFAIT, which will be used as the baseline in the evaluation. It furthermore introduces the simulation models and parameters for the technologies used. The chapter concludes with a quick preview of topics which will be investigated in more detail in following chapters.

4.1 FPGA Base Architecture

Top Architecture Figure 4.1 shows the top-level view of the base FPGA architecture used in this thesis. It is based on *k6_frac_N10_40nm*, which is one of the reference architectures shipped with Verilog to Routing (VTR). These architectures model a 40 nm technology FPGA, trying to match the Stratix IV architecture closely. Area, wire segment lengths, capacitances and resistances have been modelled to match this commercial FPGA architecture. *k6_frac_N10_40nm* is based on the main VTR flagship architecture, but with the memory blocks, DSP blocks and carry chains removed: As these elements are not relevant for this thesis, the simplest of the flagship architecture versions, consisting of only CLBs and Input-/Output-Blocks (IOBs), was selected. The flagship architecture uses uniform channel widths of 1.0 for all channels and Wilton switch blocks with $f_s = 3$. It uses only one type of wire segments of length 4 and fully populated connections to CB and SB, i.e. wires are connected to those elements whenever they intersect. Blocks shown in white are the logic generators (CLBs), and blocks with a dotted pattern are IOBs for

external connectivity. For more details about this architecture, refer to the VTR reference manual.

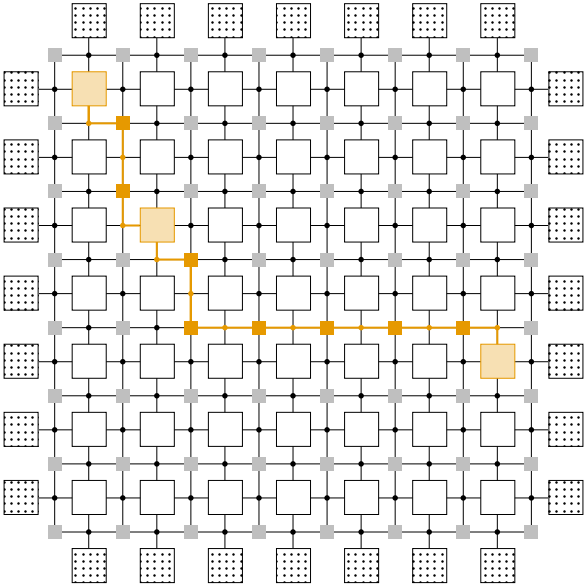


Figure 4.1: Top-level view of VTR's flagship *k6_frac_N10_40nm* architecture specified in the `timing/k6_frac_N10_40nm.xml` file shipped with VTR version 8. It consists of IOBs at the periphery and CLBs in all remaining locations. The corners do not contain any blocks.

A reduced excerpt of the `XML` architecture description is given in listing 4.1. Physical values depending on technology such as delays, capacitances, resistances and area have been removed in the excerpt for brevity. For all evaluations performed with VPR, the values have been kept as in the original file.

```
1  <architecture>
2    <layout>
3      <auto_layout aspect_ratio="1.0">
4        <perimeter type="io" priority="100"/>
5        <corners type="EMPTY" priority="101"/>
6        <fill type="clb" priority="10"/>
7      </auto_layout>
8    </layout>
9
```

```
10    <device>
11      <chan_width_distr>
12        <x distr="uniform" peak="1.000000"/>
13        <y distr="uniform" peak="1.000000"/>
14      </chan_width_distr>
15      <switch_block type="wilton" fs="3"/>
16      <connection_block input_switch_name="ipin_cblock"/>
17    </device>
18    <switchlist>...</switchlist>
19    <segmentlist>
20      <segment freq="1.000000" length="4" type="unidir">
21        <mux name="0"/>
22        <sb type="pattern">1 1 1 1 1</sb>
23        <cb type="pattern">1 1 1 1</cb>
24      </segment>
25    </segmentlist>
26
27    <complexblocklist>..</complexblocklist>
28    <power>..</power>
29    <clocks>..</clocks>
30  </architecture>
```

Listing 4.1: Top-level part of the XML architecture description used by the VTR framework and VPR.

The excerpt shows how architecture dimensions are defined and how specific blocks are assigned to locations, according to their `priority`. The `auto_layout` statement in line 3 instructs VPR to automatically size the device to fit the FPGA user application which is currently being placed. Lines 11 – 14 specify the channel width, which is specified as a relative fraction of the architecture channel width. This final architecture channel width is usually determined dynamically by VPR when placing the application: VPR starts with a small value and increases the channel-width whenever it can not route a design due to congestion.

The following lines specify SB and CB details. For the SB, only the switching pattern is specified here. For the CB, a switch is selected from the `switch_list`. This architecture uses only one SB and one CB type. Lines 19 – 25 specify the routing segments available in the global interconnect. This specifies the wire length as 4 and that the wires connect to all SBs and CBs. Line 28 species technological parameters for power analysis and line 29 specifies the clock, for which the reference architecture defines

only a single one. Line 27 describes the blocks in the architecture in more detail.

IOB Description IOBs will not be further examined in this thesis, but their architecture affects the placement of FPGA user applications in the evaluation. Therefore, a quick description of the blocks is given here, whereas the full IOB architecture description can be found in the appendix, see listing B.1 on page 349. IOBs in the architecture operate in one of two modes, but never in both at the same time: An IOB can either be in input, or in output mode. IOBs are clocked devices and therefore connect to the global device clock. On each IOB location, there can be up to 8 instances of IOBs. This means that each single location provides up to 8 inputs or outputs. For inputs, IOBs connect to 15 % of the wires in a channel, for outputs to 10 %. The IOB architecture model specifies pins on all sides allowing for a single model to be used for all FPGA sides. As only one of the IOB sides is ever connected to the global interconnect, this does not increase the number of available connections for the IOB.

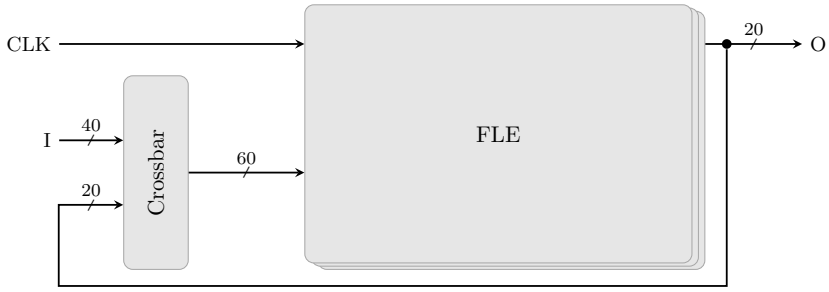


Figure 4.2: CLB in VTR's flagship *k6_frac_N10_40nm* architecture for VTR version 8, containing 10 Fracturable Logic Elements (FLEs).

CLB Description Figure 4.2 shows the structure of the CLB logic generator in the reference architecture. It is a cluster of $N = 10$ FLEs, where each FLE can be configured as either one $K = 6$ input LUT, or as two $K = 5$ input LUTs with identical inputs. The CLB connects to the global clock, which is forwarded to the FLE. To connect the inputs to the global interconnect, it contains a fully populated crossbar. The crossbar connects 40 inputs to the tracks in the adjacent channel through a CB. In addition, all 20 outputs of the FLE are fed back into the crossbar to realize a local interconnect. This allows to connect multiple FLEs in series without using global routing resources.

The crossbar then provides 60 inputs to the FLEs, 6 for each single one. As the crossbar is fully populated, all outputs can select any of the inputs. The CLB connects to 15 % of input wires in a channel and 10 % of outputs. An architecture description, reduced in the same way as the top architecture description, is shown below in listing 4.2.

```
<pb_type name="clb" area="53894">
  <input name="I" num_pins="40" equivalent="full"/>
  <output name="O" num_pins="20" equivalent="none"/>
  <clock name="clk" num_pins="1"/>

  <pb_type name="fle" num_pb="10">
    <input name="in" num_pins="6"/>
    <output name="out" num_pins="2"/>
    <clock name="clk" num_pins="1"/>

    <mode name="n2_lut5">...</mode>
    <mode name="n1_lut6">...</mode>
  </pb_type>
  <interconnect>
    <complete name="crossbar" input="clb.I fle[9:0].out"
        ↪ output="fle[9:0].in"></complete>
    <complete name="clks" input="clb.clk" output="fle[9:0].clk
        ↪ "></complete>
    <direct name="clbouts1" input="fle[9:0].out[0:0]" output="
        ↪ clb.O[9:0]"/>
    <direct name="clbouts2" input="fle[9:0].out[1:1]" output="
        ↪ clb.O[19:10]"/>
  </interconnect>

  <fc in_type="frac" in_val="0.15" out_type="frac" out_val="
      ↪ 0.10"/>
  <pinlocations pattern="spread"/>
</pb_type>
```

Listing 4.2: CLB part of the XML architecture description used by the VTR framework and VPR.

BLE Description A structural depiction of the FLE modes is shown in figure 4.3, where Figure 4.3a shows the 6 input LUT FLE mode. It contains a Basic Logic Element (BLE) with an additional FF and a MUX to select between the LUT output and the FF output. This way, the FLE can realize both combinational and sequential logic. The architecture description XML for this

mode is given in the appendix, see listing B.2 on page 350. figure 4.3b shows the second mode the FLE can operate in, as two 5-input LUTs. In this mode, only the first 5 of the 6 available inputs are used. The inputs are connected identically to both LUTs. LUTs are again combined with a FF and MUX in a BLE. This mode provides two outputs, which are both connected individually to the outputs of the CLB. This mode's architecture description `XML` can also be found in the appendix, see listing B.3 on page 351.

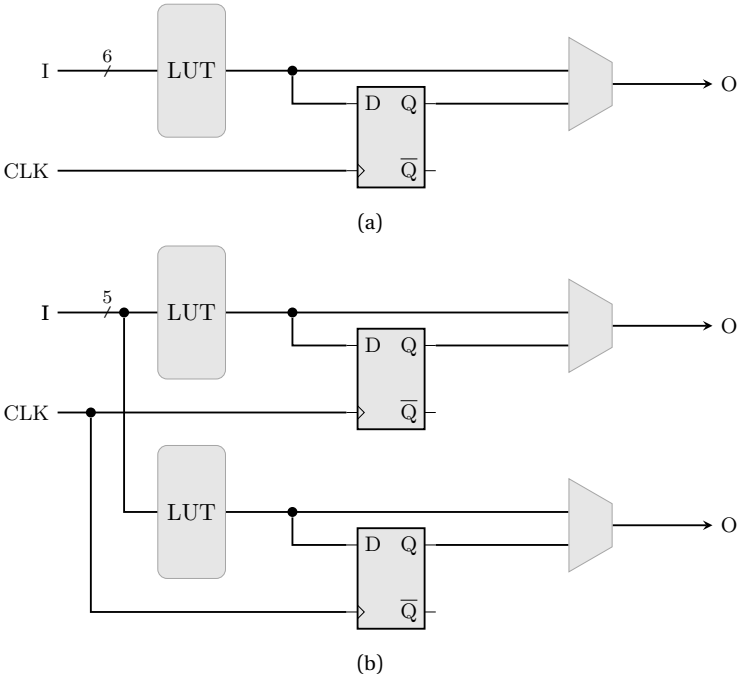


Figure 4.3: FLE used in the CLBs in VTR's flagship *k6_frac_N10_40nm* architecture. The FLE can be configured to operate in one of two modes: **(a)** LUT6 mode which implements a single 6-input LUT with one output. **(b)** Fractured mode, which implements two LUT5 with the same 5 inputs and provides two outputs in total.

RFET Hard IP In order to make use of the BG of RFETs to trade-off performance and static leakage, as introduced in section 2.2 on page 14, the transistors in the FPGA architecture have to be replaced with RFETs. One approach to achieve this is by replacing common CMOS cells with RFET standard cells. Chapter 5 on page 161 will therefore present an RFET standard

cell library. It will also describe an EDA flow as introduced in section 3.6 on page 101 to make use of them. Results will be compared with the state-of-the art works introduced in section 3.1 on page 69.

RFET Logic Generators Whereas RFETs can be used as drop-in replacements for standard silicon devices, such a usage neglects some of RFETs benefits, especially for reconfigurable LEs. This thesis will therefore analyze substitution of CLBs in the architecture with RFET based LEs, as introduced in section 2.6 on page 53. Details of this analysis will be presented in chapter 6 on page 177 and results will be compared to the state-of-the-art architectures of section 3.2 on page 74.

Due to different propagation delays of the cells, replacing these CLBs in figure 4.1 on page 108 and adjusting their BG voltages will not only affect the leakage current, but also the performance of the FPGA: In that figure, an example of a critical path is shown in orange. It depicts the path with the least available slack as introduced in section 2.3 on page 20, for a specific user application. Similarly to the example in that chapter, the total propagation delay t_{PD} is given as the sum of propagation delays in the wiring and in the combinational logic. And similarly to the introduction there, setup and hold conditions dictate the maximum clock frequency (equation (2.15) on page 25) and the minimal delay (equation (2.14) on page 25). A major difference is that in FPGAs, there are no logic gates to be considered. Delays are instead caused by the LEs (section 2.5 on page 45) and MUXs in the interconnect.

This dissertation has to answer the question how a change in BG voltage will affect these delays and how such a change can be made at application runtime without violating timing constraints. As shown in equation (2.3) on page 12, for normal CMOS devices, I_{D} is dependent on V_{th}, which is modulated in RFETs using the back gate:

$$\begin{aligned} I_D &= \frac{W}{L_{\mathrm{eff}} P_C} (V_{\mathrm{GS}} - V_{\mathrm{th}})^{\alpha} \\ &= C_{\alpha} \mu (V_{\mathrm{GS}} - V_{\mathrm{th}})^{\alpha} \end{aligned} \tag{4.1}$$

t_{PD} on the other hand depends on I_{D}, as introduced previously in equation (2.10) on page 23:

$$t_{\mathrm{PD}} = \left(\frac{1}{2} - \frac{1-\nu_{\mathrm{T}}}{1+\alpha} \right) t_{\mathrm{T}} + \frac{C_{\mathrm{L}} V_{\mathrm{DD}}}{2 I_{\mathrm{D}}} \tag{4.2}$$

This leads to a dependency of t_{PD} on the BG voltage. The propagation delay of a combined path, containing multiple transistors or gates, can be described as in equation (2.13) on page 23:

$$t_{PD,Path} \propto \frac{C_{tot} V_{DD}}{\mu (V_{GS} - V_{th})^{\alpha}} \tag{4.3}$$

The exact relation of t_{PD} and the BG voltage depends on the used technology. It can be simulated in SPICE simulations, if a suitable model for dynamic behavior is included in the PDK. As shown in section 3.1 on page 69, this is commonly not the case for RFET technologies. As an alternative, the Scarpato Model introduced in section 3.4 on page 85 will be adopted and parametrized for the considered target technologies. This parametrization will be detailed in the following sections in this chapter.

Possible change of t_{PD} in the critical path also needs to be considered during user application synthesis, which will be covered as part of chapter 7 on page 195. Furthermore, using different LEs requires changes in EDA tools, which will be discussed in chapter 6 on page 177 and compared to the state-of-the art solutions of section 3.6 on page 101.

Power can be described as static and dynamic power, as shown in equations (2.8) and (2.9) on page 22:

$$P_{total} = P_{dynamic} + P_{static} \tag{4.4}$$

$$P_{dynamic} = P_{switching} + P_{short\ circuit} \tag{4.5}$$

$$P_{static} = \left(I_{sub} + I_{gate} + I_{junct} + I_{cont}\right) V_{DD} \tag{4.6}$$

$$P_{switching} = \alpha C_L V_{DD}^2 f \tag{4.7}$$

As static leakage current is also dependent on I_D, there is a trade-off between performance and power usage that will be exploited in power management techniques similar to section 3.5 on page 96 and will be evaluated in chapter 9 on page 231.

4.2 FPGA Power Regions

Figure 4.4 shows the reference architecture of figure 4.1 with a slightly more complex user application, depicted in orange. As depicted in this example, utilization of FPGAs is commonly below 100 % in real world applications. Full

utilization usually reduces performance because of routing congestion or makes designs not routable in extreme cases. Other effects reducing the total utilization are limited amounts and fixed positions of other resources, such as DSP blocks or specific IOB.

As introduced in the previous section, it is now possible to scale the overall performance vs. leakage power of the FPGA, as long as the timing constraints for the critical path hold. However, as can also be seen, the utilization of CLBs can vary locally: This example design was placed into the bottom left corner of the device, adjacent to the used IOBs. On the other hand side, almost no CLBs in the top right corner are utilized. With a global power scaling approach as presented in section 3.5 on page 96, those unused cells still draw static power P_{static} due to leakage (see equation (4.6)). As a first estimate, it can be assumed that there is however no dynamic power use P_{dynamic}, as there is no switching, i.e. $f = 0$ (see equation (4.5)).

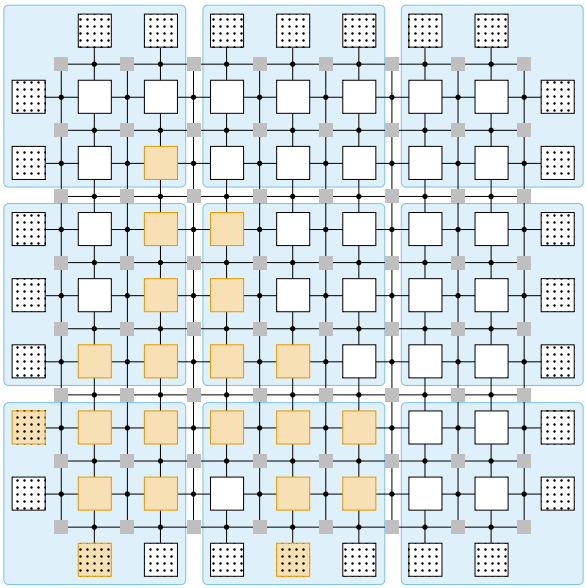


Figure 4.4: The *k6_frac_N10_40nm* architecture with a placed user application (orange) and the newly introduced region grid (blue). Region size was arbitrarily chosen as 3x3 blocks, including IOBs.

To avoid static power in unused CLBs, a region based power management strategy is introduced for the PARFAIT system. Regions are depicted in blue

in figure 4.4 and allow voltage scaling approaches to be used locally. The concepts introduced in the dissertation are largely independent of the power management technology used. The evaluation of the RFET system will use BG based threshold voltage adjustment, evaluation of SOI reference technology will use body biasing. The approaches and tools presented here can also be used for supply voltage scaling, as previously presented in literature. In that case, the limited output voltage swing of cells needs to be considered when driving cells in other regions. As the approaches used here do not scale the supply voltage, this problem is not relevant for the PARFAIT system.

With the exemplary region size and application of figure 4.4, four of nine regions could be turned off completely. But even for regions where this is not the case, more fine-grained power management is now possible: Timing requirements in form of the critical path can now be considered independently for the regions. Instead of focusing on the one global critical path, the critical path in each section needs to be considered. This becomes especially obvious when thinking of examples where the critical path does not cross one region at all. Nevertheless, timing constraints for paths in this region need to hold, and a local, critical path needs to be tracked. For this, the EDA process for user applications introduced in section 2.8 on page 63 needs to be modified: The critical path needs to be determined and stored for each single region. Special care must be taken for paths which cross region boundaries, as scaling the performance in one region affects the performance of the whole path. It might therefore preclude further scaling in other regions which contain this path as well.

In addition to critical path extraction, a region based architecture calls for further optimization in the EDA tools: If placement is made aware of regions by factoring in the cost of placing logic into currently unused regions, keeping regions empty can be incentivized. Another approach is to predefine regions with fixed performance/power trade-offs at FPGA manufacturing time. Such a static high performance / low power region approach reduces complexity and circuit overhead, as dynamic scaling after in-the-field programming is not required. EDA tool modifications presented here will also enable evaluation of such systems.

Details on power management region aspects of the PARFAIT system will be presented in chapter 7 on page 195 and a simulation framework to evaluate the whole system including regions support will be presented in chapter 9 on page 231. Final Design Space Evaluation (DSE) of parameters, such as the

region size or cost function weights, will then be presented in chapter 10 on page 253.

4.3 PVTA Compensation

Previous sections explained how voltage adjustment can be used to reduce performance where the slack of the critical paths allows it, and how to decrease the static leakage. In a first implementation, this can be realized as a static approach: The user application can be analyzed in an EDA tool and the slack can be derived in each region and then stored as part of the FPGA bitstream. When programming, the regions can be changed to the proper performance level and then don't change at runtime.

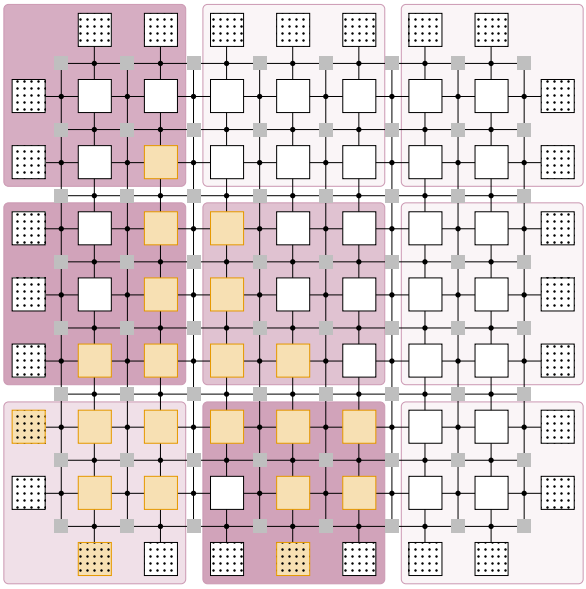
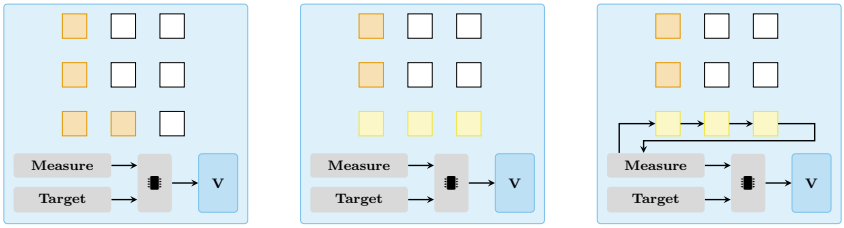


Figure 4.5: The *k6_frac_N10_40nm* architecture with the shade of red in each region representing local performance variations due to PVTA. Effects vary within each region as well, but have been limited to region scope here for readability.

Such a static approach however wastes some potential. As has been discussed in section 2.4 on page 27, manufactured FPGA ICs have local performance vari-

ations, which then cause pessimistic results in STA. This effect is depicted in figure 4.5, where different shades of red show different performance in the regions. Even when the application is analyzed during EDA, as the performance of each region is not known ahead of time, the performance adjustment needs to be performed conservatively.

To address this in the PARFAIT system, a dynamic, closed loop control system for performance in each region is introduced. The overall, high level concept is presented in figure 4.6.



(a) (b) (c)

Figure 4.6: Region detail and region controller concept. The figures show a zoomed in detail of the central region in figure 4.4. **(a)** Region controller in idle mode. **(b)** Row was reconfigured dynamically with measurement circuits. **(c)** Delay characterization is active.

Figure 4.6a shows a zoomed in detail of one region. In addition to the FPGA elements of the base architecture, it also contains the building blocks for the closed loop control system introduced in section 3.4 on page 85: Critical path identification is performed in EDA tools for the FPGA application. The determined, available slack is used to derive a slack factor, which describes the relative percentage of how much the delay in a region is allowed to increase. This is stored as the *Target* for each region.

The architecture does not make use of path replicas. As paths on FPGA are always made up of the base LEs anyway, it is sufficient to characterize these basic LEs. This work is performed in the *Measure* block. The PARFAIT architecture measures the real delays of real LEs, not artificial, additional logic. This also enables the architecture to effectively measure each single CLB and BLE on the FPGA. In order to keep resource overhead low, the architecture uses fine-grain dynamic reconfiguration (section 3.3 on page 77) to temporarily reprogram areas of the FPGA. As depicted in figure 4.6b, the architecture therefore transparently moved the user application down one row in the FPGA on a row-by-row basis. In order for this to work, one row of the FPGA needs

to stay unprogrammed. Furthermore, some special handling is needed to ensure that the user application can continue to be executed. Details about this are given in section 8.2 on page 206.

Once the original application logic has been moved, the row is reprogrammed with specific delay measurement circuits. This process will be called "Logic Invasion" in the future, as it replaces the user application logic. While most of the measurement logic is kept in reconfigurable cells, a central *Measure* controller orchestrates the measurements, as shown in figure 4.6c. The measured delay is then compared to the nominal delay and the target delay in a controller system. This controller appropriately adjusts the performance in the regions using a voltage controller. Although not directly visible in figure 4.6, this effectively forms a closed loop system: The change in voltage will change the cell delay, as introduced previously, closing the loop.

Whereas this enables more efficient power reduction for the user application, it also allows addressing PVTA effects: As these manifest in changed cell propagation delay, they will be picked up by the *Measure* characterization step and be compensated transparently. It should be noted that the high-level description here is simplified and the mentioned blocks do not necessarily all have dedicated hardware in each region. As will be shown later, all blocks, except for the voltage adjustment logic, will actually be centralized to reduce the hardware overhead.

Simulation and DSE Whereas previous sections have described the complete PARFAIT architecture, some more thoughts are necessary to efficiently simulate and evaluate the system. As previously described, simulation models are necessary to describe the actual influence of voltage adjustment on the propagation delays of cells. Furthermore, a similar model is needed to estimate the changes in P_{static} with voltage adjustment. For PVTA, two simulation models are required: First, it must be determined how those effects actually affect the propagation delay, t_{PD}:

$$t_{\text{PD}} = f(P, V, T, A) \tag{4.8}$$

And considering the possibility of another independent variable for performance adjustment, C:

$$t_{\text{PD}} = f(P, V, T, A, C) \tag{4.9}$$

These equations are heavily technology-dependent. For well-established technologies, an algebraic expression can be derived from equations given in chapter 2. For various novel technologies, including the RFET technology,

such detailed equations and understanding are however not yet available. Even for SOI and similar technology, using text-book equations may lead to larger modeling errors than using vendor-specific SPICE models shipped as part of PDKs. This dissertation will therefore use the generic Scarpato model presented in section 2.4 on page 27, fitting it to the technologies evaluated in this thesis.

With this model, the system can be simulated given the P, V, T, A and C inputs. C can be obtained through simulation of the voltage control loop, but for the other inputs, realistic test scenarios need to be given. The following sections will derive both the technology model and the scenarios used to derive PVTA parameters in detail.

4.4 FPGA Implementation

For comparison of the final results and as a basis for the PARFAIT FPGA, the *k6_frac_N10_40nm* architecture was implemented in VHDL. The VPR toolflow was extended with missing tools to enable a full Verilog-to-Bitstream flow for this architecture. Ultimately, benchmark applications can therefore be synthesized to a bitstream and the FPGA can be simulated in Questasim, including the application bitstream. This workflow enables prototyping of system-level FPGA features with quick functional verification.

Architecture Modifications Some minor details of the *k6_frac_N10_40nm* reference had to be changed for implementation: First, a fixed layout has been introduced instead of the originally used automated layout. With an automated layout, VPR dynamically sizes the FPGA, finding the smallest FPGA size which fits the user application to be placed. Although the bitstream generator and the FPGA implementation support parametrization to any size, there are benefits in fixing the size. For example, it allows generation of bitstreams for multiple applications, that can all be evaluated on the same FPGA instance. As a second step, FASM metadata has been added to the architecture to enable generation of bitstreams. This metadata does not affect the FPGA architecture in any way and will be discussed in detail in the next section.

In addition, there are actual changes of the functional architecture itself: The pin equivalence for CLBs has been changed to `none`. Without this setting, VPR introduces another stage in the final generated netlist which permutes inputs of CLBs. Although the VHDL code for the implemented architecture

supports this permutation, VPR does not emit FASM for those. Without this information, it is however not possible to generate correct bitstreams, so the permutation was disabled completely. In an additional change, the `capacity` of IO cells has been reduced to five instead of eight in the original architecture. As a consequence, there will be only five input and output pins per IOB instead of eight. This is not a real limitation, as the evaluated benchmarks require large FPGA sizes and the IO ports are therefore not a limiting factor. IOBs connect to channels on only one side, so five outputs in total match the CLB's five outputs on each side. Given that VPR selects the tracks to connect to depending on the outputs of blocks adjacent to a channel, using the same amount here greatly simplifies the connection pattern. With this change, this pattern becomes identical for all block outputs all over the FPGA, for both IOBs and CLBs. This greatly reduces complexity both in the VHDL FPGA implementation and in the bitstream generator. Block input mapping still varies between blocks, as well be explained later in this section.

The last change is related to the routing channels and Programmable Switch Matrixs (PSMs): In VPR, PSMs for FPGAs with unidirectional channels use modified Wilton switch patterns. These patterns are can only be tiled, i.e. are only identical for each PSM instance, if all tracks start or stop at a PSM. If a track instead passes through a PSM, VPR will use a custom pattern for each PSM. As such an irregularity largely increases complexity of the FPGA and the bitstream generator, pass-through tracks had to be avoided. To realize this, the segment length of tracks was set to one instead of the original four. As these architecture changes affect benchmark results, both the LUT based architecture used for comparison and the ULM RFET architecture implement all these changes consistently.

FPGA Interconnect An important aspect of FPGA implementations that has not been discussed so far is the interconnect system, responsible for routing of signals between logic blocks. This work focuses largely on changes of the CLB, but an FPGA implementation also has to deal with various interconnect aspects, which are explained in the following.

The interconnect system can be divided roughly into two categories: First, a local interconnect, which is used for internal connections within each logic block. In the PARFAIT FPGA, this is mainly the crossbar in CLBs, explained as part of figure 4.2 on page 110. The second category is the global interconnect, which consists of PSMs and the routing channels, which contain tracks connecting the PSMs. These tracks also connect to CBs, where CBs can be either part of CLB or of the top-level system. The PARFAIT FPGA realizes a generic CB with customizable patterns, which specify how to connect inputs

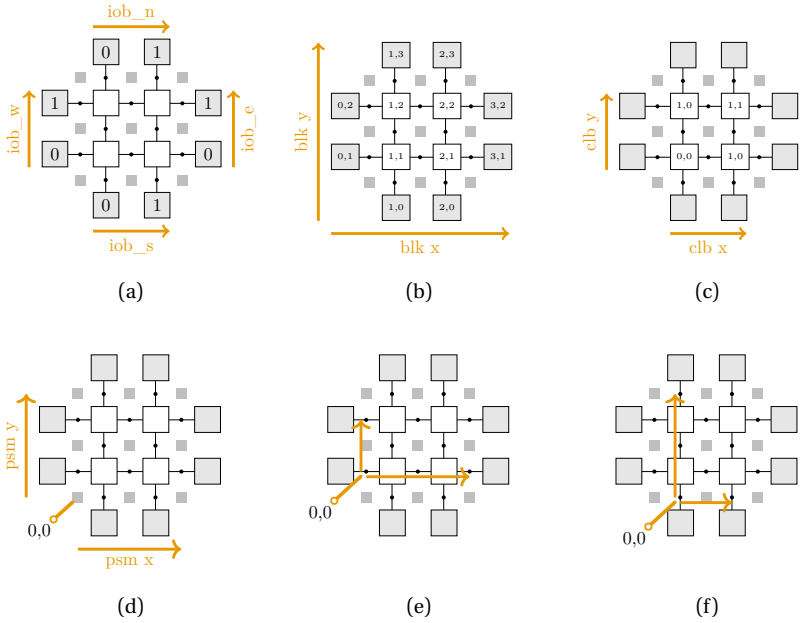


Figure 4.7: Various grids and numbering conventions used in the FPGA implementation. **(a)** Indices and names of IOs as used in VHDL. **(b)** Indices of all FPGA blocks as used in FASM. **(c)** Indices of CLBs as used in VHDL. **(d)** Indices of PSMs as used in FASM and VHDL. **(e)** Indices and names of Connection Box South-North (CBSN) as used in VHDL. **(f)** Indices and names of Connection Box East-West (CBEW) as used in VHDL.

and outputs to certain tracks. In this implementation, the CBs are therefore part of the global interconnect.

Figure 4.7 shows various grids and indexing that is used in the top-level design of the FPGA. As shown in **(a)**, IOBs are numbered in a one-dimensional scheme: They are grouped into north, east, south and west and then numbered from left to right or bottom to top. This indexing is primarily used in the VHDL code, as there is no IOB specific bitstream in the bitstream generator. Subfigure **(b)** shows the 2D grid used to number all blocks. This is the indexing scheme which is used in FASM and the generated bitstream, as it handles all block indices in a generic way. Subfigure **(c)** shows the 2D grid indexing scheme used to name CLBs in the VHDL code. The VHDL code gives names to blocks according to type.

Subfigure **(d)** shows the locations and indexing of PSMs. It should be noted that PSMs at edge and corner locations are technically not connected to all channels. Nevertheless, for simplicity, an identical implementation is used. For technical reasons, the VHDL code however has to actually connect unused channel outputs. Because of that, the unused signals are declared in the code, but they are not connected anywhere else. Subfigure **(e)** and **(f)** show CBs connecting channels to blocks south and north or east and west. The VHDL implementation of those CBs is the same for both variants. It should be noted that the size of all grid varies a bit, but is always derived from the total FPGA width and height.

The top-level file and interconnect are automatically generated using the custom `pgen` tool. This tool parses a mustache template file and then generates a VHDL file implementing an FPGA of the selected size. An excerpt of this template is shown in listing 4.3 as an example. As can be seen, this template system allows substitution of certain values and repeated instantiation of blocks. A single instance of generated code is shown in listing 4.4.

```
{{#IOB_N}}
iob_n{{NUM}}: entity vfpga.IOB
    port map (
        din => cb_ns_no({{CB_X}}, {{CB_Y}})(7 downto 0),
        dout => cb_ns_ni({{CB_X}}, {{CB_Y}})(7 downto 0),

        pad_i => iob_ni({{NUM}}),
        pad_o => iob_no({{NUM}})
    );
{{/IOB_N}}
```

Listing 4.3: Excerpt of the top-level FPGA mustache template showing the IOB instantiation code.

```
iob_n00: entity vfpga.IOB
    port map (
        din => cb_ns_no(0, 3)(7 downto 0),
        dout => cb_ns_ni(0, 3)(7 downto 0),

        pad_i => iob_ni(00),
        pad_o => iob_no(00)
    );
```

Listing 4.4: Excerpt of the top-level FPGA architecture generated from listing 4.3.

CB Implementation The used Connection Box consists of multiple configurable multiplexers, as shown in figure 4.8.

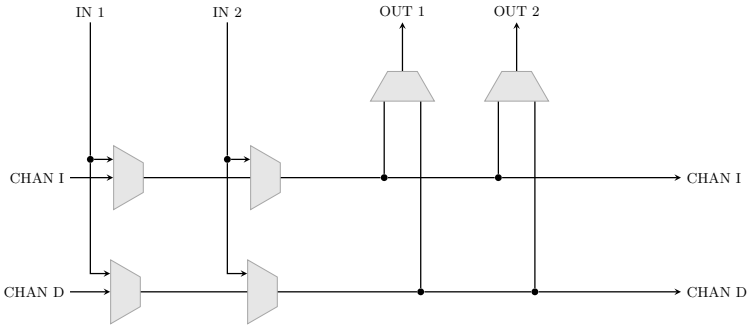


Figure 4.8: CB used in the PARFAIT FPGA. `CHAN I` and `CHAN D` are channel signals connecting to PSMs. As tracks are unidirectional, a CB connects to tracks in increasing and decreasing direction. `IN 1` and `OUT 1` are output and input signals from one CLB adjacent to the CB. `IN 2` and `OUT 2` are respective signals for the other CLB.

The exact number of tracks in all signal busses can be configured in the VHDL code. How a specific CB instance maps the output signals `OUT1` and `OUT2` to channel tracks can be specified using a `generic` parameter. The same is true for the input signals `IN 1` and `IN 2`. Which of the individual connected tracks is actually selected by the multiplexers is configurable through the FPGA bitstream. The CB instances in the PARFAIT architecture use pin maps extracted from VPR, to realize the connections following the VPR architecture description.

A requirement of VPR is that both connected CLBs must be able to drive inputs of the other one through the CB. Therefore, both CLB's inputs `IN 1` and `IN 2` are first multiplexed onto the channel bus. After that, both CLB outputs `OUT 1` and `OUT 2` are read from the now multiplexed channel. Signals are unidirectional, so each CLB connects to the channel busses for increasing and decreasing direction. The CLB signals must be able to drive and read signals in both directions, which is ensured by the realization in figure 4.8. VPR normally determines the required channel width on the fly, depending on the mapped application. To get more reproducible results, the channel width is always fixed to 80 tracks in this dissertation, giving 40 tracks in each direction.

Programming Programmable elements, which include the CLBs, PSM and the CBs contain internal configuration storage. This storage is realized as shift register with serial input and output, and a parallel output. This way, multiple elements can be chained. Data is transferred using a single serial data line, a clock signal and an enable signal. Such an approach reduces the number of required wires at the expense of longer configuration times, as long sequences of bits need to be shifted into the registers. Configuration time is not critical for the PARFAIT FPGA, so this approach was chosen.

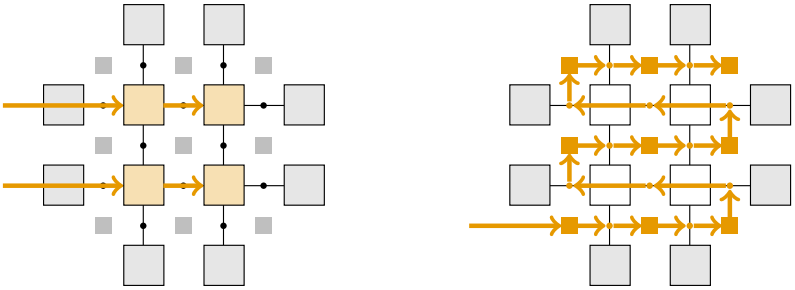


Figure 4.9: Programming chain connecting the configurable blocks in the PARFAIT FPGA. **(a)** Chain for the CLBs. Each row can be programmed individually. **(b)** Chain for interconnect elements. Programs all PSMs, CBSN and CBEW at once.

Figure 4.9 shows how the programmable elements are chained. The leftmost arrows depict external inputs, which are connected to the FPGA programming port driven by the `ProgController` code. Subfigure **(a)** depicts the configuration chains for the CLBs. CLB configuration includes the local interconnect, i.e. the crossbar. It does however not include any of the CBs, which are programmed as part of the interconnect bitstream instead. Each row gets an own CLB configuration chain. This enables configuration of single CLB rows while keeping other rows unaffected. Subfigure **(b)** shows the configuration chain for the global interconnect. As can be seen, this chain includes the PSMs, CBSNs and CBEWs.

Figure 4.10 shows the FSM describing the `PROG` controller. This controller can program a configuration chain of arbitrary length. It uses a simple interface to receive the data to be programmed in parallel, and to enable programming of the serial chain. As shown in the FSM, the controller initially resets the configured chain's storage registers. It then shifts all bits onto the con-

figuration chain. When it finished programming all bits, it asserts the done signal.

Figure 4.10: FSM describing the PROG component.

Figure 4.11 depicts how the PROG block is used in a ProgController to program all chains of figure 4.9. There are two instances of the PROG block: One directly drives the interconnect programming chain. The other one is used to program the CLB rows. As only one controller is used for the rows, a multiplexer connects the proper row configuration chain to the controller.

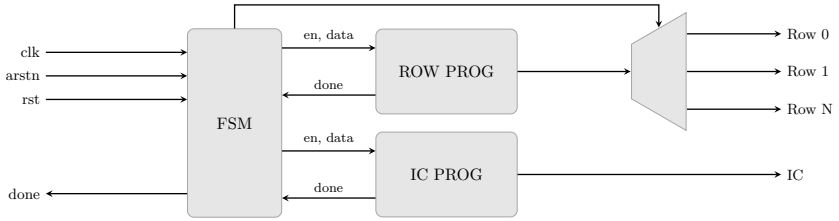


Figure 4.11: Block diagram describing the ProgController component. It consists of the control FSM two PROG controllers for the interconnect and for the CLB rows and one multiplexer to select the active CLB row.

The FSM shown in figure 4.12 is used to coordinate the programming process. It first programs the interconnect, then all the CLB rows individually. Once the FPGA is fully programmed, it asserts a done signal to inform the user that the FPGA application is now ready.

Figure 4.12: FSM describing the ProgController component.

The exact format of the bitstream will be introduced as part of the next section, as it is relevant for the logic invasion concept introduced in this thesis.

4.5 FPGA Toolflow

The VPR toolchain can place applications to FPGA architectures using only an architecture file. It can then gather statistics such as the number of used blocks, routing track utilization and more. When testing functional aspects such as logic invasion presented in section 8.2 on page 206, it is more promising to simulate the real implementation of the architecture presented in the previous chapter. However, for such an evaluation, the user application needs to be programmed to this custom FPGA architecture. This then requires a more complete toolflow than the one offered by VPR, as the placed and routed design also needs to be converted to a bitstream.

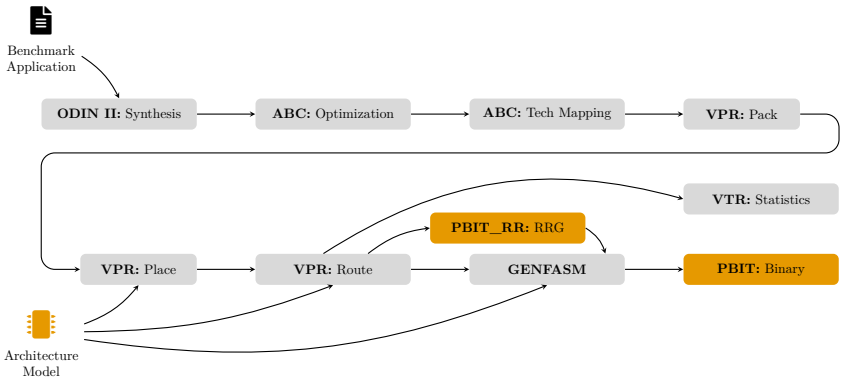


Figure 4.13: Verilog-to-Bitstream toolfow for the LUT based PARFAIT reference architecture. Parts marked in orange are custom parts that have been developed as part of this thesis.

Figure 4.13 shows the toolflow developed as part of this thesis. It is largely based on the original VPR toolflow, but extended with some newly written tools, which are shown in orange in the figure. The basic approach chosen for bitstream generation is based on FASM, a text based format for bitstream representation available in VPR [244].

```
1  BLK_X[001]Y[002]Z[000].CLB.FLE[8].5BLE[1].LUT5.INIT[31:0]=32'b
       ↪ 11110000000000001111000000000000
```

Listing 4.5: An example FASM feature for a 32 bit LUT.

FASM aims to be the analogon of software assembly formats for FPGA based hardware description. The format specification only specifies that “features”

are hierarchically grouped keys, which optionally can define a value. The exact content and meaning of these keys is specified by the architecture designer in the VPR architecture description, as metadata. An example FASM feature illustrating these elements is given in listing 4.5. Here, everything up to the equals sign is the feature key. The value is a Verilog-style binary string literal. The feature is hierarchical: It describes the `INIT` value of the second `LUT5` in a `5BLE` in the eight `FLE` of the `CLB` at block position `1, 2`.

Although FASM can be written manually for maximum control of placement and routing, it commonly is generated automatically from Verilog code instead. As shown in figure 4.13, the Verilog application is first fed into *ODIN II* for synthesis. After synthesis has transformed the Verilog application into a technology-independent netlist, this netlist is getting optimized in *ABC*. Then, also in *ABC*, the netlist is mapped to the target technology. For the reference architecture, *ABC* is instructed to map to 6-input LUTs. If less inputs are required, VPR will later automatically use the fractured 5-input LUT instead.

After synthesis, *VPR* is used for the implementation passes. First, it packs related FFs and LUTs together for use in BLEs. It then places all logic into BLEs and CLBs based on the architecture description. In the last implementation step, VPR routes the design. At this point, the placed and routed design is saved into multiple files: A `.place` file describing block locations, a `.route` file specifying the global routing, a `.net` file describing the intra-block routing and a `.blif` file, describing the netlist. All files need to be parsed manually to obtain the full information required to generate bitstreams. As this is a tedious and error-prone process, the *GENFASM* tool has been introduced. It can read the aforementioned files and emit FASM such as the one in listing 4.5. Given that FASM does not provide any semantic meaning in the specification, it however needs additional specifications using metadata in the architecture description. Furthermore, metadata needs to be added to the VPR Routing Resource Graph (RRG), which is realized by the custom written *PBIT_RR* tool. The generated `.fasm` code finally can be transformed to a binary bitstream using the custom *PBIT* tool. In the following pages, the newly introduced tools and steps will be explained in more detail.

Architecture Metadata VPR emits FASM whenever it creates instances of `pbs` specified in an architecture description. Listing 4.6 shows an example of such metadata added to the FPGA architecture description. In this case, it describes the `LUT5.INIT[31:0]` feature. VPR will automatically add the LUT table as the feature value. The prefix of the feature is also constructed automatically, by looking at the metadata of parent `pbs`.

```
<pb_type name="lut5" blif_model=".names" num_pb="1" class="lut
    ↪ ">
    <metadata>
        <meta name="fasm_type">LUT</meta>
        <meta name="fasm_lut">LUT5.INIT[31:0]</meta>
    </metadata>
```

Listing 4.6: Architecture metadata to generate FASM features for LUTs.

VPR's current FASM implementation does not support emitting block location indices. Instead, users have to specify metadata for each single CLB individually. As this is rather tedious, the FASM support was extended as part of this thesis to automatically add position information when a `$BLK_X_Y_Z` feature is processed. Indices in the block grid are defined according to figure 4.7b on page 122. Furthermore, VPR currently only supports emitting FASM for multiplexers, but not for crossbars. The reference `k6_frac_N10_mem32K_40nm` architecture however does use a crossbar in the CLB implementation. As the architecture should not be modified too much to keep results comparable, crossbar FASM support was added to VPR as well.

The features shown in listing 4.7 show all the features that are derived from the architecture description. Those include LUT data for the two 5-input LUTs in a BLE or for a single 6-input LUT, if the BLE is not in split mode. It further specifies whether the FF in the BLE is used or bypassed. The last feature shows how the CLB crossbar is modeled in the PARFAIT FASM description: It specifies that the third input of BLE 8 should be connected to the second input of the CLB, driven by the global interconnect. Alternatively, a value of `fle[0].O[0]` would specify internal feedback, i.e. connecting the first output of BLE 0 to this input.

```
BLK_X[001]Y[002]Z[000].CLB.FLE[8].5BLE[1].LUT5.INIT[31:0]=...
BLK_X[001]Y[002]Z[000].CLB.FLE[8].5BLE[1].FF.BYPASS
BLK_X[001]Y[002]Z[000].CLB.FLE[8].5BLE[0].LUT5.INIT[31:0]=...
BLK_X[001]Y[002]Z[000].CLB.FLE[8].5BLE[0].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[7].6BLE.LUT6.INIT[63:0]=...
BLK_X[000]Y[000]Z[000].CLB.FLE[7].6BLE.FF.ENABLE
BLK_X[001]Y[002]Z[000].CLB.FLE[8].CB.I[2]="clb[0].I[1]"
```

Listing 4.7: Summary of features that are derived from the architecture description.

PBIT_RR is used to parse the VPR RRG and add FASM metadata. Listing 4.8 shows an example of an RRG after it has been processed by `PBIT_RR`, where

the `metadata` tag has been added by the tool. The modified RRG is fed into the *GENFASM* tool, which will simply emit the feature string whenever it finds that a segment has been used in the routing.

```
<node id="20" type="OPIN" capacity="1">
    <loc xlow="0" ylow="1" xhigh="0" yhigh="1" side="RIGHT"
        ↪ ptc="1"/>
</node>
<node id="9552" type="CHANY" direction="INC_DIR" capacity="1">
    <loc xlow="0" ylow="1" xhigh="0" yhigh="1" ptc="0"/>
</node>
<!-- ... -->
<edge src_node="20" sink_node="9552" switch_id="2">
    <metadata>
        <meta name="fasm_features">BLK_X[000]Y[001]Z[000].CBW[
            ↪ E].ODST[00]=1</meta>
    </metadata>
</edge>
```

Listing 4.8: Example VPR RRG extended with FASM metadata by `PBIT_RR`.

In the RRG, nodes represent channels (connecting PSMs), or `IPIN`s and `OPIN`s, representing inputs and outputs of CLBs and other blocks. Edges on the other hand represent a possible connection between such nodes. The RRG contains all theoretically possible connections. It is therefore only dependent on the FPGA architecture description, but is independent of the application. The `PBIT_RR` tool analyzes all segments and derives FASM features for them. It then distinguishes two kinds of edges: If an edge connects two channel nodes, it is representing a connection in a PSM. If it connects a channel and an `IPIN` or a channel and an `OPIN`, it is a CB connection.

```
PSM_X[000]Y[001]Z[000].MUX[S].TRACK[79]="N"
BLK_X[001]Y[002]Z[000].CBW[W].ODST[79]=59
BLK_X[000]Y[001]Z[000].CBR[E].I[12]=79
```

Listing 4.9: Summary of features that are derived from the RRG.

Listing 4.9 gives examples of those generated features, with the first line describing a PSM connection. Apart from block coordinates, it contains the information that track 79 in the south output of the PSM needs to be driven by the input from north. Each PSM output only connects to one track in each direction, so this information is complete.

The second line shows information for the output part of a CB, driving routing channels from a CLB or IOB. This specific feature signifies that track 79 in the channel in the west needs to be driven by output 59 of the block. The tracks that can connect to a certain output, follow a fixed pattern, so not all numbers may be emitted here. The third line shows information for the input part of a CB, selecting tracks to route into a CLB or IOB. In this case, it specifies that input 12 should be driven by track 79 in the channel to the east of the logic block.

The generated FASM uses pin and track numbers as used by VPR, to enable easier debugging. These numbers do not directly correspond to multiplexer indices though: As only some connections are possible, there are less multiplexer addresses than pin and track numbers. The mapping is done by both *PBIT*, which maps track indices to multiplexer addresses, and by the VHDL code specifying the architecture. The latter has to undo the mapping to connect to the proper tracks for a certain multiplexer address. The connection patterns of pins to tracks are determined in VPR using an elaborate algorithm. Reimplementing this algorithm in custom tools would be error-prone, so a different solution has been chosen: The *PBIT_RR* tool will generate pin maps based on the RRG [245] to be used in the other tools. An example of such a pin map for inputs in the north of an IOB block is shown in listing 4.10. Here, the first line for example states that IOB pin 0 can connect to track pins 0, 1, 12, etc.

```
======================== IOB_N ========================
00 => [0, 1, 12, 13, 26, 27, 40, 41, 52, 53, 66, 67]
03 => [2, 3, 16, 17, 28, 29, 42, 43, 56, 57, 68, 69]
06 => [4, 5, 18, 19, 32, 33, 44, 45, 58, 59, 72, 73]
09 => [8, 9, 20, 21, 34, 35, 48, 49, 60, 61, 74, 75]
12 => [10, 11, 24, 25, 36, 37, 50, 51, 64, 65, 76, 77]
```

Listing 4.10: Pin map generated by `PBIT_RR` for the north inputs of an IOB.

PBIT reads FASM files, which might be either generated as previously explained, or manually written, and transforms them to binary configuration data. In this process, it first maps track indices and block pins to multiplexer addresses, using the pin maps generated by *PBIT_RR*. It then converts these addresses to binary data vectors. Vectors representing different blocks are then ultimately combined to fit the order of the configuration chains of figure 4.9 on page 125. Data can be directly emitted as a VHDL package, based on a given mustache template.

Bitstream formats are shown in tables 4.1 to 4.4. Table 4.1 shows the bitstream format used for PSMs. For all 40 tracks in all 4 directions, it encodes the direction a track is driven from. For example, the `N` entry in `TRACK_0` specifies where the track 0 output in the north of the PSM is driven from. Values are encoded as 2 bit in increasing order: from left, from center, from right (looking from output track into the PSM).

319								**7**			**0**
TRACK_39				TRACK_1				TRACK_0			
W	S	E	N	W	S	E	N	W	S	E	N

Table 4.1: Bitstream Format for the PSM

Table 4.1 describes the bitstream format used for BLEs. It consists of the LUT data, which might either be two times 32 bit in fractured LUT mode, or one time 64 bit for a 6-input LUT. Which mode is selected is specified by the `SPLIT` bit. `MUX0` and `MUX1` bits then drive the multiplexers selecting between LUT or FF outputs.

66	**65**	**64**	**32**	**0**
SPLIT	MUX_1	MUX_0	LUT_1	LUT_0

Table 4.2: Bitstream Format for the BLE

Table 4.3 describes the bitstream format for complete CLBs. It consists of 10 BLEs as specified in table 4.2. Furthermore, it contains the configuration for the crossbar in the CLB. Each entry is a 6 bit value, selecting one of 40 global inputs (0 – 39) or one of the feedback signals (40 – 59). Inputs are sorted, so that the six inputs `I[0]-I[5]` in category `MUX[0]` drive inputs 0 – 5 of BLE 0.

1029						**705**						**669**	**0**
MUX_9						MUX_0						BLE_9	BLE_0
I5	I4	I3	I2	I1	I0	I5	I4	I3	I2	I1	I0		

Table 4.3: Bitstream Format for the CLB

Table 4.4 describes the bitstream format for CBs. A CB can drive 32 tracks of a channel in both directions, from two inputs `A` and `B`. Each of the fields

DBD, etc., is a single bit which is high if the respective track is driven from the respective input. Otherwise, the track is passed through unmodified. As an example, a 1 bit in DBD of DRIVE_0 signifies that track 0 in decreasing channel direction should be driven by port B. Note that track index 0 is not the track index as in VPR and the pin index in B is not configurable. These values are fixed in the VHDL implementation according to the previously introduced pin maps. Values in the READ categories describe what channel track should drive an input to logic block A or B. Each value is a 4 bit vector, encoding values from 0 to 12. Again, the exact track numbers selected are determined according to the pin map. In addition, the numbers also encode the direction, as the multiplexer connects to both channel directions (refer to figure 4.8 on page 124). For example, RA = 4b'0010 in READ_9 signifies that input signal 9 of the block A should be driven by track 2.

207		**135**	**128**					**3**			**0**
READ_9		READ_0		DRIVE_31				DRIVE_0			
RA	RB	RA	RB	DAI	DAD	DBI	DBD	DAI	DAD	DBI	DBD

Table 4.4: Bitstream Format for the CB

4.6 Technology Modeling

To be able to simulate the PARFAIT architecture, a simulation model to predict the circuit delay under certain process conditions is required. Although any model can be used in the tools designed as part of this thesis, the evaluation here will use models based on Scarpato's work [3]. Refer to sections 2.3 and 2.4 on page 20 and on page 27 for an introduction to this model.

	ad	**as**	**nrd**	**nrs**	**pd**	**ps**	**W**
PMOS	1.68×10^{-12}	1.68×10^{-12}	0.08	0.08	7.96×10^{-6}	7.96×10^{-6}	$1u$
NMOS	2.4×10^{-13}	2.4×10^{-13}	0.54	0.54	1.96×10^{-6}	1.96×10^{-6}	$0.5u$

Table 4.5: Layout parameters for the MOSFETs used in the test circuit for t_{PD} analysis in *XT018*. Most parameters are identical to [Reu21], but the device width was recalculated to obtain symmetrical inverters at 1.8 V.

In general, the Scarpato model describes delay for a certain gate or a whole circuit. In this dissertation, the model was derived for specific test gates.

Based on this, it is assumed that FPGA circuits based on reconfigurable LEs show behavior proportional to the test gates. The propagation delay models will therefore be normalized to yield a 1.0 value at nominal condition. As a result, the final models will yield a factor that can be multiplied with the LE delay at nominal condition to describe performance increase and decrease.

SOI Reference Technology

The commercial *XT018* technology is used as a reference to compare the RFET results to. It is a 180 nm SOI technology, featuring a similar feature size as the PARFAIT RFET. As the technology features a complete, commercial PDK, required simulation models can be fitted to results of SPICE simulations. For these evaluations, a simple CMOS inverter as shown in figure 4.14 was designed in Cadence *Virtuoso*. General layout parameters were taken from Reuter et al. to match the investigation in [Reu21]. As the operating voltage used in this dissertation is 1.8 V, the width of the transistors had to be refitted though. Layout parameters are shown in table 4.5.

Test Circuit Figure 4.14 shows the test circuit used to extract the propagation delays. Delays are measured for the second inverter and in between the `I` and `O` points.

In analog simulations, the first inverter's input is driven by a simulated, variable voltage source. The source voltage starts at 0 V at $t = 0$, and starts rising linearly at $t = 40$ ps with a slope of 1.8 V/40 ps. For the nominal voltage of 1.8 V, this ensures that the input rises in 40 ps to the maximum input voltage. When using non-nominal supply voltage, this specification keeps the slope constant, changing the transition time. At $t = 500$ ps, the input falls linearly to 0 V, matching the same absolute slope as for the rising edge. This artificial signal generates an output at the first inverter, which is close to inverter output transients in real circuits. It drives the second inverter, which in turn drives four other inverters at its output. Outputs of the last inverter stage are kept unconnected. An implementation of this circuit is evaluated in Cadence *Virtuoso*, where it can be parametrized for the supply voltage, body biasing voltage and temperature.

Voltage and Temperature Following the process in [3], the model will first be fitted to the `V` and `T` data. Delays are extracted from the transient simulations shown in figures 4.15a and 4.15b on page 136, where the propagation delay is

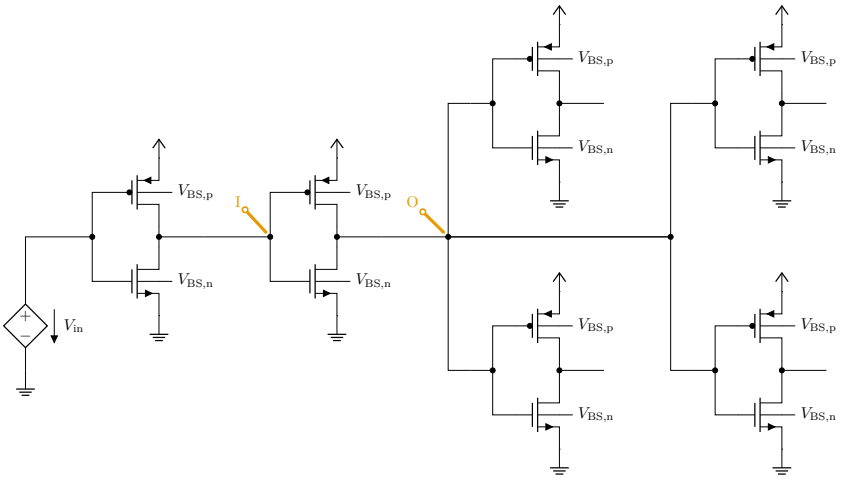
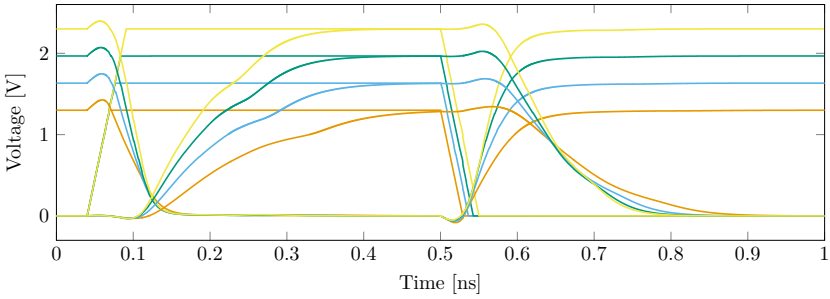


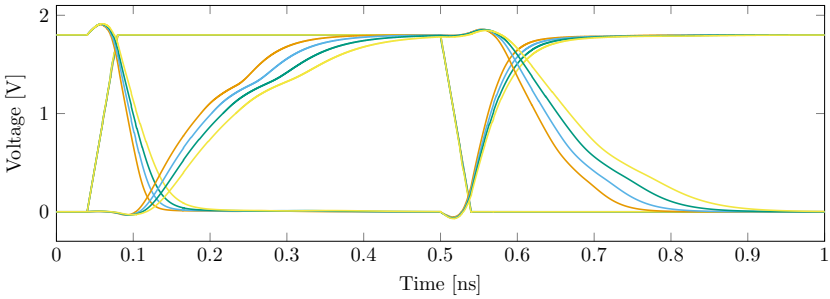
Figure 4.14: Test circuit to extract the FO4 delay of an inverter. The characterized inverter (second from left to right) is driven by another inverter and drives a load consisting of four inverters.

computed in Cadence *Virtuoso* using the `delayMeasure` function. Both the rise-fall t_{PD} and fall-rise t_{PD} haven been extracted, and it has been verified that they show identical behavior with respect to the tested parameters. The absolute values of these delays are not identical, due to small mismatches in the PMOS and NMOS networks. However, for the system level simulation, the model only needs to report a performance scaling factor. Because of this, the absolute values are not important, and the model developed here therefore will be parametrized using only fall-rise transitions. The model will then be normalized in the end.

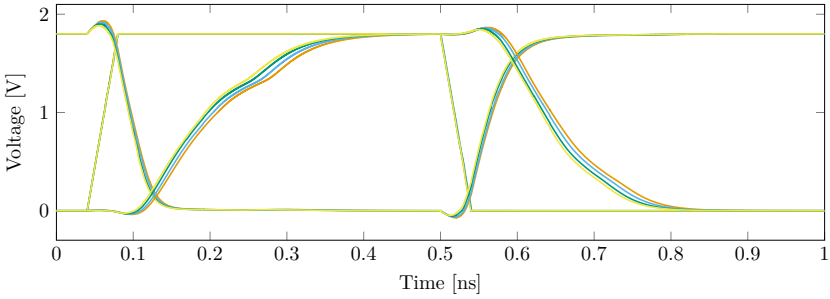
For supply voltage dependency modeling, the supply voltage was swept from 1.3 V to 2.3 V using steps of 50 mV, whereas other parameters have been fixed at nominal conditions. The simulated data, 21 points covering a range of ±500 mV centered on the nominal 1.8 V, was then exported. Data samples are shown in figure 4.16 on page 141 as dots in the left column. A similar approach was taken to extract the temperature dependency figure 4.15b. Temperature T was swept from −40 °C to 175 °C with step size 5 °C. The resulting 44 delay values were exported and then converted to Kelvin units. Resulting values are shown in figure 4.16 on page 141 as dots in the right column.



(a)



(b)



(c)

Figure 4.15: Transients obtained for the test circuit of figure 4.14 for XT018 technology, varying various parameters. All simulations represent the typical corner. Only four curves are shown for illustrative reasons. **(a)** Varying V from 1.3 V to 2.3 V. $T = 25\,°C$, $V_{BB} = 0\,V$. **(b)** Varying T from $-40\,°C$ to $175\,°C$. $V = 1.8\,V$, $V_{BB} = 0\,V$. **(c)** Varying V_{BB} from $-0.5\,V$ to $0.5\,V$. $T = 25\,°C$, $V = 1.8\,V$.

The data was fitted to the equation (2.22), which is repeated here in equation (4.10). Fitting used a least-squares optimization and the Python `scipy.curve_fit` function. This resulted in the parameters shown in table 4.6, resulting in the initial model to describe variation in delay based on temperature T and supply voltage VDD.

$$t_{\mathrm{PD}} = f(V,T) = p_\beta + (C_1 + k_1 T^{n_1}) \frac{V}{(V-(C_2 - k_2 T^{n_2}))^{p_\alpha}} \tag{4.10}$$

Parameter	Value
p_β	$3.228\,586\,39 \times 10^{-10}$
C_1	$-2.402\,611\,87 \times 10^{-2}$
k_1	$-1.794\,153\,24 \times 10^{-12}$
n_1	3.697 621 43
C_2	$-1.436\,105\,80 \times 10^{1}$
k_2	$5.799\,321\,14 \times 10^{-8}$
n_2	2.782 633 32
p_α	6.801 112 80

Table 4.6: Parameters obtained when least-squares fitting the delays obtained in simulations of the test circuit in the *XT018* technology to equation (4.10).

Process Variation As in [3], the next step in derivation of the delay model is characterization of the process variation dependency. Unfortunately, little information about process variation was available for the evaluated *XT018* technology. Variation of geometrical parameters (see section 2.4) could be simulated in *Virtuoso* through parametrization. However, the more interesting information is the absolute amount of variation to be expected, which could not be obtained. Therefore, instead of modifying the SPICE circuit simulation to estimate process variation, factory provided data for digital cells was used: For those cells, the PDK provides various corners (see section 2.8) for slow, typical and fast devices.

The test circuit of figure 4.14 has therefore been replicated in VHDL as shown in listing C.1 on page 353. The test uses inverter cells of type `INHDX0` from the `D_CELLS_HD` high density standard cell library shipped with the PDK. The VHDL test circuit has then been synthesized in Cadence *Genus* using all available corners. A timing analysis has been performed and the

top 100 critical paths have been extracted. Out of those paths, the worst rise-fall and fall-rise propagation times for the second inverter, `inv1`, have been extracted. Again, only the fall-rise transitions have been used for modelling.

The absolute delay of the custom designed cell in SPICE simulation is not identical to the one of the commercial cells in the PDK. Because of that, a scaling factor has been derived using the delays of both cells at nominal operating conditions. For the SPICE simulated results, the delay was obtained using the model in equation (4.10). For the cell from the standard cell library, the delay has been obtained through simulation in Cadence *Genus*. The obtained scaling factor was used to scale all delays obtained in the *Genus* simulations before any further processing.

Then, as in [3], all parameters except for C_2 and p_α were fixed to the values in table 4.6. As there were not many data points available, p_α could also be fixed in addition. Then, all 4 delays obtained for the fast corner, at various temperatures and VDD, were used to fit equation (4.10) with these fixed parameters. This process was repeated with the delays obtained for the slow corner. The resulting C_2 values are shown in table 4.7.

Parameter	**Value**
C_2 slow	−14.669 855 714 313 31
C_2 typical	−14.361 05
C_2 fast	−14.138 776 396 937 581

Table 4.7: Fitted C_2 for equation (4.10) and the Cadence *Genus* results for the *XT018* technology. Other parameters are kept fixed as in table 4.6.

As in [3], to obtain delays for continuous process variation instead of only for the corners, the C_2 variable has been interpolated linearly, fitting it to the following equation:

$$C_2(P) = C_{2\mathrm{C}} + C_{2\mathrm{m}} * P \tag{4.11}$$

Here, P describes the process as a number ranging from −1 (slow) to 1 (fast). Values were fitted as $C_{2\mathrm{C}} = -14.361\,068$ and $C_{2\mathrm{m}} = 0.265\,539\,66$ and the extended model with process variation is given as:

$$t_{\mathrm{PD}} = f(V,T,P) = p_\beta + (C_1 + k_1 T^{n_1}) \frac{V}{(V - (C_2(P) - k_2 T^{n_2}))^{p_\alpha}} \tag{4.12}$$

Figure 4.17 on page 142 shows examples of varying the process parameter P in the derived model, keeping other parameters at nominal values. Five

values are depicted, −1 (slow), −0.5, 0 (typical), 0.5 and 1.0 (fast). Dots in these figures depict the simulated values obtained from the Cadence *Genus*. Unfortunately, the available corners do not only differ in process variation, but also in supply voltage and temperature. Because of this, only a typical process corner is available at nominal temperature and supply voltage, so only one dot for the typical process is shown.

Body Biasing As a next step, the dependency of body biasing, used to control the performance of the device, is added to the model. Body biasing is not covered in [3], but the extension of the model is straightforward: Comparing equation (4.10) and equation (2.12) to the physical model of the drain current equation (2.3), shows the physical motivation of the Scarpato delay model, and that the threshold voltage V_{th} is modeled like this:

$$V_{\text{th}} \propto (C_2(P) - k_2 T^{n_2}) \tag{4.13}$$

Body biasing affects the threshold voltage V_{th} based on body effect factor γ and the Fermi potential Φ according to equation (2.1) on page 11. The original model has therefore been extended with additional terms affecting the threshold voltage following equation (2.1) in equation (4.15). The new complete model is given in equation (4.14):

$$\begin{aligned} t_{\text{PD}} &= f(V,T,P,C) \\ &= p_\beta + (C_1 + k_1 T^{n_1}) * \frac{V}{(V - (C_2(P) + C_3(C) - k_2 T^{n_2}))^{p_\alpha}} \end{aligned} \tag{4.14}$$

$$C_3(C) = C_\gamma * \left(\sqrt{|-C_\Phi - C|} - \sqrt{|-C_\Phi|}\right) \tag{4.15}$$

The control input C corresponds directly to the body bias voltage V_{BS} and C_γ and C_Φ are newly introduced fitting parameters.

The test circuit of figure 4.14 was evaluated for a body bias voltage sweep from −500 mV to 500 mV, where bulk voltages are defined as in the following:

$$V_{\text{BS,p}} = V_{\text{DD}} - V_{\text{BS}} \tag{4.16}$$

$$V_{\text{BS,n}} = 0\,\text{V} + V_{\text{BS}} \tag{4.17}$$

PMOS and NMOS devices are therefore biased symmetrically. Positive V_{BS} is defined as forward body biasing, where the delay of the circuit is reduced. The voltage has been increased in 50 mV increments and the resulting 21 data points are shown as dots in the left-hand side figures of figure 4.17. The newly introduced parameters were then fitted as $C_\gamma = 0.173\,299\,252\,539\,216\,82$ and

$C_\Phi = 0.758\,826\,970\,183\,017\,5$. Interpolated results, which were obtained using this model, are shown in figure 4.17.

Aging As a final step, the model was extended to model aging. Unfortunately, no detailed aging information was available in the used *XT018* PDK. This is an issue, as aging itself depends on temperature and voltage [3], which makes it difficult to model without sufficient data. However, as the model derived here will only be used as a realistic test case, it does not have to model the *XT018* technology exactly. As long as the model behaves in a way a real technology would, it is suitable for the architecture evaluation. Because of this, an existing aging model from [3] has been integrated into the SOI reference model used here. Following equation (3.10) in [3], aging is introduced as a shift in V_{th} with $\Delta V_{\text{th}} = A$, yielding the final model:

$$\begin{aligned} t_{\text{PD}} &= f(V,T,P,C,A) \\ &= p_\beta + (C_1 + k_1 T^{n_1}) * \frac{V}{(V - (C_2(P) + C_3(C) + A - k_2 T^{n_2}))^{p_\alpha}} \end{aligned} \tag{4.18}$$

The simplest of the aging models investigated in [3] has been chosen for the final propagation delay model:

$$A = f(t, V_{\text{A}}, T_{\text{A}}) = C * t^n * V_A^\gamma * e^{-900/T_A} \tag{4.19}$$

The parameters have been taken from table 3.5 in [3] and are reprinted in this thesis in table 4.8.

Parameter	Value
γ	4.88
C	2.5×10^{-3}
n	0.16

Table 4.8: Parameters for equation (4.19) taken from Table 3.5 in [3]. This reproduces the aging model for the technology evaluated in [3].

This approach is realistic, as this aging model represents an absolute shift of the threshold voltage. Even though the technologies are different, absolute values of threshold voltages are similar, which allows to obtain a realistic combined model. It should be noted that parameters V and T in equation (4.18) are instantaneous values at a certain time t. The values in equation (4.19) on the other hand are values which represent the aging process, e.g. devices age faster at higher temperature. V and V_{A}, and T and T_{A}, therefore have to be considered to be independent parameters.

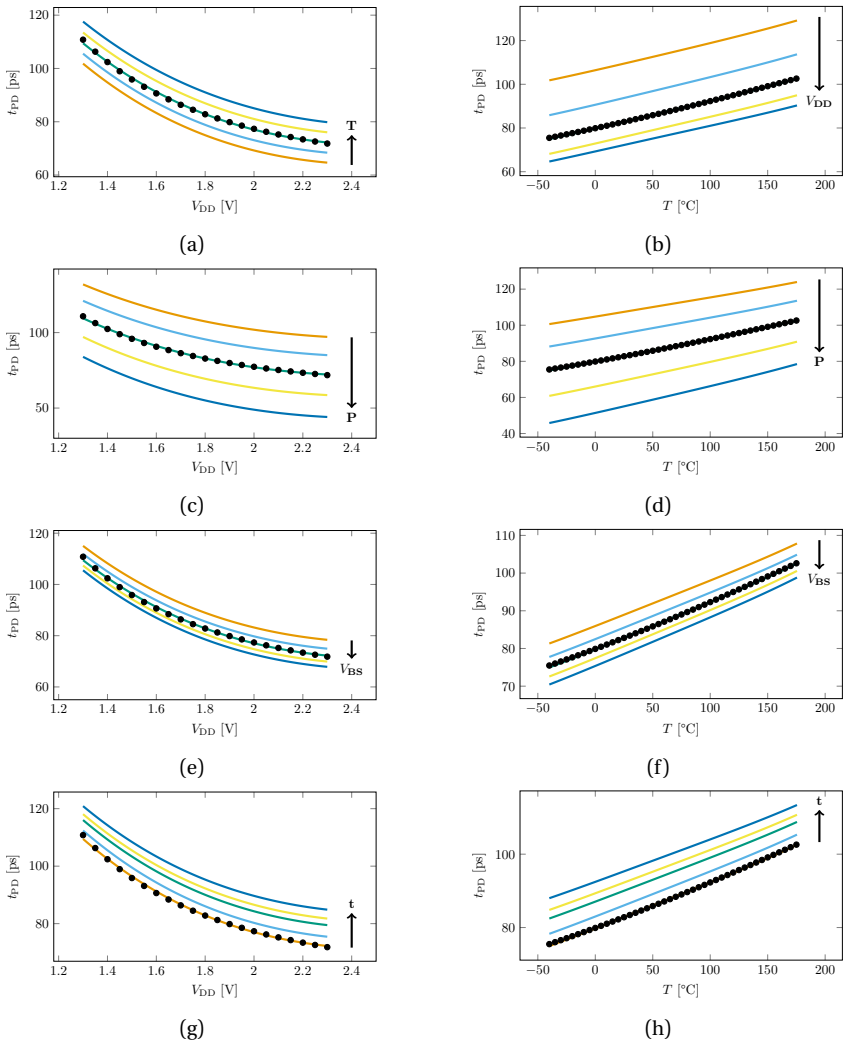


Figure 4.16: t_{PD} model (lines) and fitting data (dots) for the *XT018* technology. The left-hand side figures show propagation delay vs. supply voltage, parametrized on the four other parameters. The right-hand side shows the delay dependency on temperature. Parameters were chosen to yield five exemplary curves covering the full parameter range, where one curve matches the parameter value for the original fitting data.

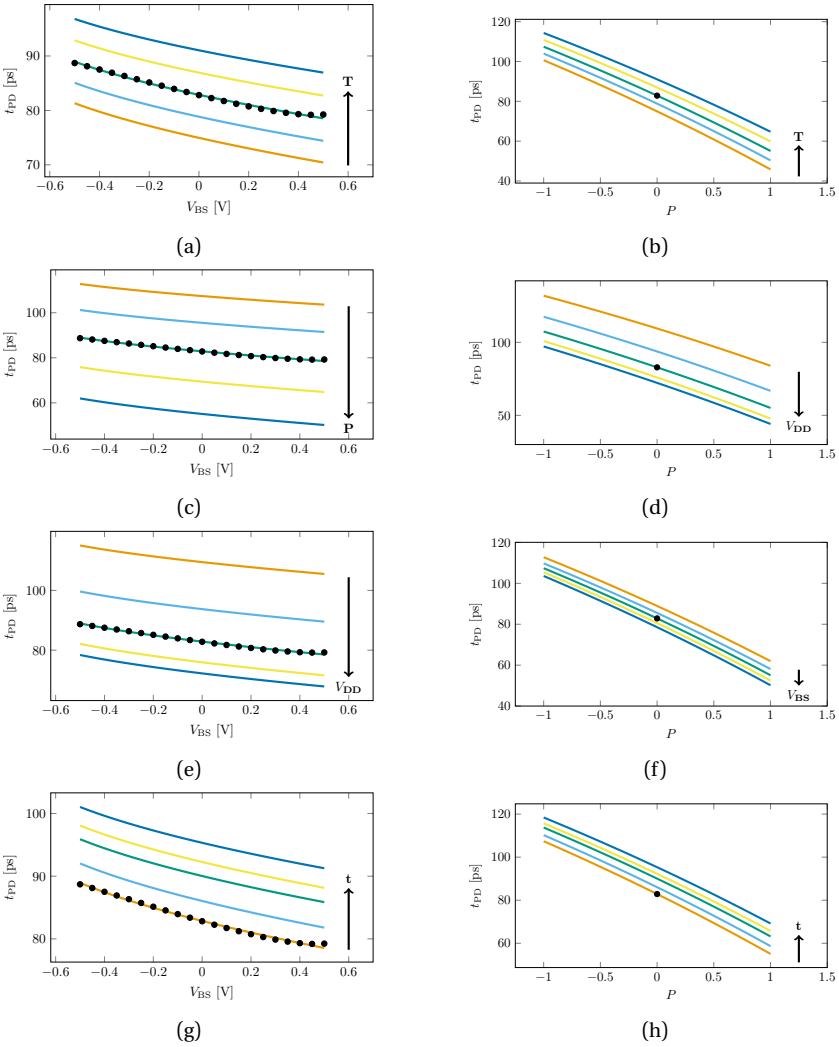


Figure 4.17: t_{PD} model (lines) and fitting data (dots) for the *XT018* technology. The left-hand side figures show propagation delay vs. body bias voltage, parametrized on the four other parameters. The right-hand side shows the delay dependency on process variation. Parameters were chosen to yield five exemplary curves covering the full parameter range, where one curve matches the parameter value for the original fitting data.

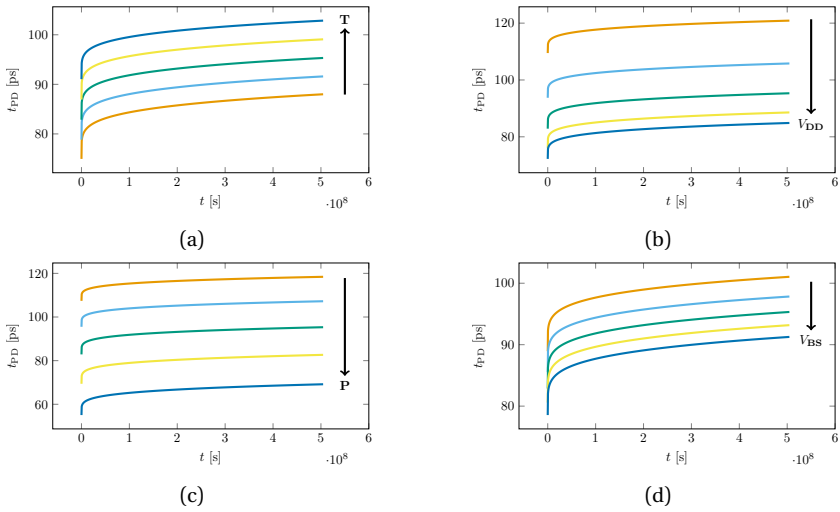


Figure 4.18: t_{PD} model (lines) and fitting data (dots) for the *XT018* technology. Figures show propagation delay vs. aging, parametrized on the four other parameters. Aging was simulated at a temperature of 120 °C and 1.8 V.

Normalization Ultimately, the model in equation (4.9) has been normalized to yield value 1.0 at nominal conditions, $P = 0$, $V = 1.8\,\text{V}$, $T = 25\,°\text{C}$, $A = 0\,\text{V}$ and $C = 0\,\text{V}$. As a result, the final model in equation (4.18) needs to be scaled by $C_{\text{norm}} = 1.206\,829\,605\,2 \times 10^{10}$.

Leakage Current In addition to the model for t_{PD}, to model power through static leakage (equation (2.8)), a model for the leakage current is required. Similar to the t_{PD} model, an inverter will be used as a representative CMOS circuit and a factor describing increase or decrease has been derived. The leakage current in a CMOS inverter is mostly determined by the I_D current of the transistor in off state. As such, two aspects are primarily important: The absolute value of the gate voltages, i.e. whether the driving gate achieves full output swing. For this dissertation, this aspect will be assumed to be given. The second aspect is the I_D when V_{GS} is close to 0 V.

Theoretically, this drain current can again depend on P, V, T and aging. For this thesis, the most important relation is the dependency on the control input or body bias voltage V_{BS}. As the derivation of a more complete model is out of scope of this thesis, only the V_{BS} dependency has been modeled. For this, a single inverter has been modeled in *Virtuoso*, again using the geometric

parameters of table 4.5. The inverter input has been biased to 0 V and the body bias voltage was again swept from 500 mV to 500 mV in 21 steps. The results of this simulation are shown as dots in figure 4.19.

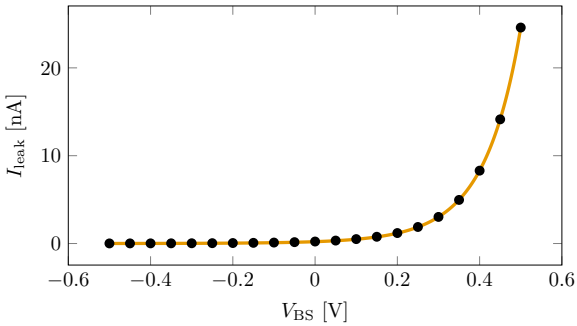


Figure 4.19: Leakage current in an inverter for the *XT018* technology. Dots depict values simulated in Cadence *Virtuoso* SPICE simulation, the line shows values obtained through equation (4.20).

This data was then fitted to a shifted power function:

$$I_{\text{leak}} = f(C) = C_{\text{ID}} * (1.8 - C)^{\alpha_{\text{ID}}} \tag{4.20}$$

The parameters for this model and the test data were calculated as $C_{\text{ID}} = 1.142\,930\,38 \times 10^{-6}$ and $\alpha_{\text{ID}} = -1.463\,373\,23 \times 10^{1}$.

RFET Technology

Now that the model for SOI technology has been derived, a similar t_{PD} model will be discussed for RFET technology: This work will primarily use Galderisi's *RGATE* [246] to demonstrate the feasibility of an FPGA based on ambipolar technology. To estimate the feasibility of the introduced PVTA compensation concepts, and to demonstrate the potential of RFETs, evaluations should ideally be performed using the *RGATE* technology.

Unfortunately, the *RGATE* and related technology were only introduced recently (2024 and 2022). Because of this, no propagation delay model, PDK or even SPICE model is available yet. However, extensive device characterization data for various parameters has been published in [42] and [2]. As a temporary solution until more exact models become available, the Scarpato model will be adapted and fitted to the device data in a best-effort approach. Certain

deviations and inaccuracies are expected: Physical effects determining the device performance are partially different for Schottky Barrier based RFETs, whereas the Scarpato model was derived for standard silicon technologies. Conclusions drawn using the adapted model should therefore be interpreted with caution, but this model at least enables a proof-of-concept evaluation. Concrete limitations of the model will be explained during the derivation and discussion of the model.

The general approach for t_{PD} modelling using device data was derived through observation of equation (2.11). Introducing the modeled parameters results in the following equation:

$$t_{\mathrm{PD}}(V,T,P,C,A) \propto \frac{C_{\mathrm{L}} V}{I_{\mathrm{D}}(V,T,P,C,A)} \tag{4.21}$$

With the parameters as in the SOI model: V as supply voltage VDD, T as temperature, P as process variable, C as control variable for the PG and A as an aging parameter. In the equation, it has been assumed that C_{L} is independent of the parameters. Whether this is the case needs to be confirmed through future device characteristics measurements and physical modeling, which are out of scope for this dissertation.

Equation (4.21) only provides a proportionality, so deriving absolute propagation delays requires fitting a model to measured or simulated delays. It is however not possible to perform these evaluations for the chosen RFET technology, as neither measured data nor simulation models are available. This dissertation however only needs information about relative effects in delay: If one of the parameters changes, how does the delay change in comparison to the original value? This normalization is used even for the SOI model, where absolute data is theoretically available: A model using absolute values is only valid for a single, specific gate with specific load conditions. Deriving relative changes instead enables generalization of the model for various gates, as will be shown later.

The relative delay can then be described as:

$$\frac{t_{\mathrm{PD}}(V,T,P,C,A)}{t_{\mathrm{PD}}(V_0,T_0,P_0,C_0,A_0)} = \frac{V \cdot I_{\mathrm{D}}(V_0,T_0,P_0,C_0,A_0)}{V_0 \cdot I_{\mathrm{D}}(V,T,P,C,A)} \tag{4.22}$$

It can be seen that equation (4.22) allows derivation of a propagation delay model when knowing only the drain current I_{D} and supply voltage VDD. These values are available from [2, 42]: I_{D} is the main quantity evaluated in device characterization and directly available. For VDD, the situation is more complicated: In a typical CMOS-like circuit, VDD determines both the

drain-source voltage of the device and the control gate voltage. How this voltage exactly determines the drain-source voltage in a circuit depends on the circuit and won't be discussed here in detail. For the model, measurements performed using a drain-source voltage of 1 V will be used, which is the highest voltage measured. The VDD parameter dependency will be extracted from the control gate voltage.

Another source of errors is inherent in this derivation process: Obtaining propagation delays from measured drain currents requires an inversion operation. Currents are however generally small, which causes larger relative measurement errors. When these values are inverted, such small errors lead to larger errors in propagation delay. It would therefore be preferable to directly measure delays, but respective data is not available. Measurement errors are partially reduced through the normalization step performed in equation (4.22), but this only cancels gain errors.

Voltage and Temperature Galderisi et al. provide characterizations for RFETs in low- and high-threshold configurations and for *n*, *p* and *ambipolar* modes [2]. For brevity, the model will be parametrized for only the *p* mode low-threshold configuration, although the *RGATE* uses both low- and high threshold configurations and *n* and *p* mode. In the model parameters and in the figures shown, the sign of all voltages will be flipped so that the final model parameters are positive numbers.

Parameter	Value
p_β	$7.387\,817\,12 \times 10^{9}$
C_1	$3.354\,015\,41 \times 10^{2}$
k_1	$-3.319\,076\,95 \times 10^{2}$
n_1	$2.169\,391\,71 \times 10^{-3}$
C_2	$-4.231\,161\,98 \times 10^{4}$
k_2	$-2.046\,606\,55 \times 10^{4}$
n_2	$6.828\,009\,25 \times 10^{-2}$
p_α	$-2.322\,585\,43$

Table 4.9: Parameters for RFET propagation delay model in nominal conditions. Data was derived from drain current measurements in [2].

For the voltage dependency, the data available from figure 1h in [2] has been used. The data was extracted for a drain voltage of -1 V, a program gate voltage of -3 V and a temperature of 25 °C. As for all measurements used, it

will be assumed that values correspond to the typical process. Control gate values were selected from −2.45 V to −3 V to ensure the current values are obtained in the saturation region.

Temperature data was obtained from figure 5c in [2], which again describes the p mode in low-threshold configuration. Program gate and drain voltage were again fixed at −3 V and −1 V, and the data was assumed to represent the typical process. Drain currents were extracted for a control gate voltage of −3 V and temperatures from 325 K to 425 K. In addition, a data point representing room temperature was taken from the voltage dependency dataset.

As in the SOI model, data was fitted to equation (2.22) using a least-squares fit and the Python `scipy.curve_fit` function:

$$t_{\mathrm{PD}} = f(V,T) = p_\beta + (C_1 + k_1 T^{n_1}) \frac{V}{(V-(C_2 - k_2 T^{n_2}))^{p_\alpha}} \tag{4.23}$$

This resulted in the parameters shown in table 4.9 for the initial model describing delay variation for temperature T and supply voltage VDD. Figure 4.20 on page 150 shows how the final model fits the extracted values. The model in equation (4.23) corresponds to the final model with $C = 3.0\,\mathrm{V}$, $P = 0$ and $t = 0$, i.e. no aging. Data in the figures has been normalized so that the delay at nominal condition ($T = 25\,°\mathrm{C}$, $V = 3.0\,\mathrm{V}$, $C = 3.0\,\mathrm{V}$, $P = 0$, $t = 0$) equals 1.0.

Process Variation Galderisi et al. provide some initial process variation data in [2, 42]. As this is currently a lab-scale process and scaling of the technology will likely have effects on process variation, this data was not used and the process variation data obtained for the industrial SOI process reference has been adapted instead: First, relative changes in delay for worst and best process have been evaluated for the SOI model, obtained at otherwise nominal conditions. This lead to a propagation delay increase of 30 % for the slow process and a decrease of 34 % for the fast process. These values were then used to scale the previously obtained temperature and voltage datasets. Like in the SOI model, all parameters except for C_2 were fixed to the values in table 4.9. The data for the slow and fast process was then again fitted to this model, yielding the C_2 values shown in table 4.10.

Following [3] the C_2 variable has been interpolated linearly again by fitting it to the following equation:

$$C_2(P) = C_{2\mathrm{C}} + C_{2\mathrm{m}} * P \tag{4.24}$$

Parameter	Value
C_2 slow	$-4.166\,208\,81\times10^4$
C_2 typical	$-4.231\,161\,98\times10^4$
C_2 fast	$-4.299\,515\,74\times10^4$

Table 4.10: Fitted C_2 for equation (4.23) and relative process variation matching the *RFET* technology. Other parameters are kept fixed as in table 4.9.

Here, P again describes the process as a number ranging from -1 (slow) to 1 (fast). Values were fitted as $C_{2C} = -4.231\,161\,98\times10^4$ and $C_{2m} = -666.535$ and the extended model with process variation is again given as:

$$t_{PD} = f(V,T,P) = p_\beta + (C_1 + k_1 T^{n_1}) \frac{V}{(V-(C_2(P)-k_2 T^{n_2}))^{p_\alpha}} \tag{4.25}$$

A comparison of the nominal operating point and the modelled values for various different process parameters is shown in figure 4.21 on page 151 in the right column. As can be seen in figure 4.20d, the model becomes unrealistic for high temperatures in the fast process, even leading to negative values. Evaluations in this parameter combination region should therefore be avoided.

Program Gate Voltage For the PG voltage, a different modeling approach had to be chosen compared to the SOI body biasing voltage: The physical principles are different and figure 1j in [2] suggests that the influence of the PG voltage is exponential. With the previously shown fitted parameters, p_β is comparatively large. This means that the whole second term can only have limited effects on the propagation delay value. Modeling the control parameter as part of this term, like in the SOI model, could therefore not represent this exponential influence.

$$\begin{aligned} t_{PD} &= f(V,T,P,C) \\ &= \left(p_\beta + (C_1 + k_1 T^{n_1}) \frac{V}{(V-(C_2(P)-k_2 T^{n_2}))^{p_\alpha}}\right) \cdot C_3(C) \end{aligned} \tag{4.26}$$

$$C_3(C) = C_\gamma \cdot e^{C_\phi \cdot C} \tag{4.27}$$

As a solution, the slightly modified version of the model above was introduced. Instead of modifying the second term, the whole function is scaled using an exponential function. This model is not physically motivated and

is therefore a pure mathematical model. It assumes that the PG voltage and other parameters are uncorrelated and can be modelled independently, which needs to be verified in future research and with more measurements. For the evaluation of the PARFAIT system, this assumption is however not limiting.

The data to be matched was extracted from figure 1j in [2] for $V = 3\,\mathrm{V}$. Drain voltage and temperature were 1 V and 25 °C respectively, and values belong to the typical process. The PG voltages for the extracted drain currents were between 0 V and 3 V. The exponential term was then first fitted independently to the normalized, inverted drain current values. The least-squares fit was modified with the additional constraint that the value for $V_{\mathrm{PG}} = 3\,\mathrm{V}$ matches the nominal model exactly, i.e. it yields exactly 1.0. This ensures high accuracy for the nominal PG voltage.

The newly introduced parameters were then fitted as $C_\gamma = 71\,310.4$ and $C_\Phi = -3.724\,932$. Interpolated results which were obtained using this model are shown in figure 4.21 on page 151 in the left column. Note that these results are plotted using a linear y axis, whereas other figures showing the PG voltage as a parameter are plotted on logarithmic y axis.

Aging As there is no aging data available for RFETs, the same model as used for the SOI reference has been integrated into the RFET model. For the same reasons as in the PG voltage modeling, aging can not easily be modeled as a shift in threshold voltage. A solution similar to the PG voltage model was used: The dependency was extracted from nominal values in the SOI model (SOI aging at 1.8 V and 120 °C). The obtained data was then fitted to a shifted root function and introduced in the final model:

$$\begin{aligned} t_{\mathrm{PD}} &= f(V,T,P,C,A) \\ &= \left(p_\beta + (C_1 + k_1 T^{n_1}) \frac{V}{(V - (C_2(P) - k_2 T^{n_2}))^{p_\alpha}}\right) \cdot C_3(C) \cdot A \end{aligned} \tag{4.28}$$

Where the aging parameter is defined as:

$$A = f(t) = 1 + C_A * t^{\alpha_A} \tag{4.29}$$

This models aging independently of temperature and voltage, which is sufficient for this thesis. Values were fitted to $C_A = 0.001\,922\,91$ and $\alpha_A = 0.199\,140\,02$. Examples of aging data obtained using the model are shown in figure 4.22 on page 152.

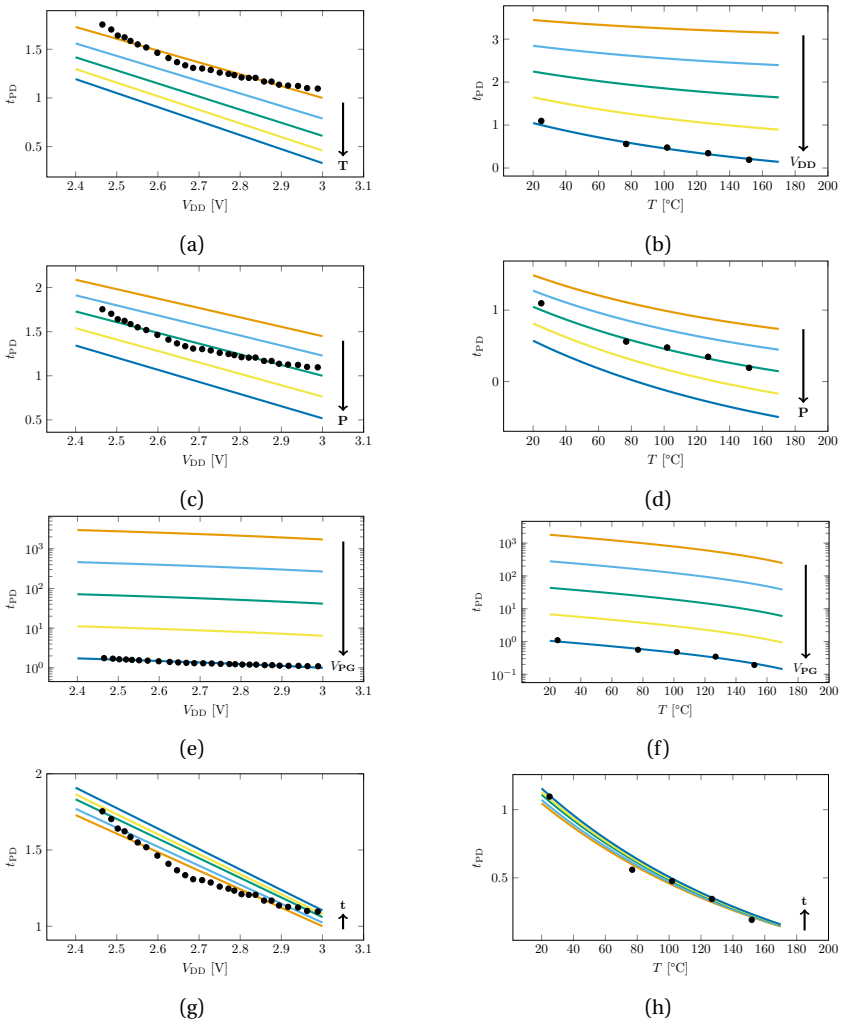


Figure 4.20: t_{PD} model (lines) and fitting data (dots) for the *RFET* technology. The left-hand side figures show propagation delay vs. supply voltage, parametrized on the four other parameters. The right-hand side shows the delay dependency on temperature. Parameters were chosen to yield five exemplary curves covering the full parameter range, where one curve matches the parameter value for the original fitting data.

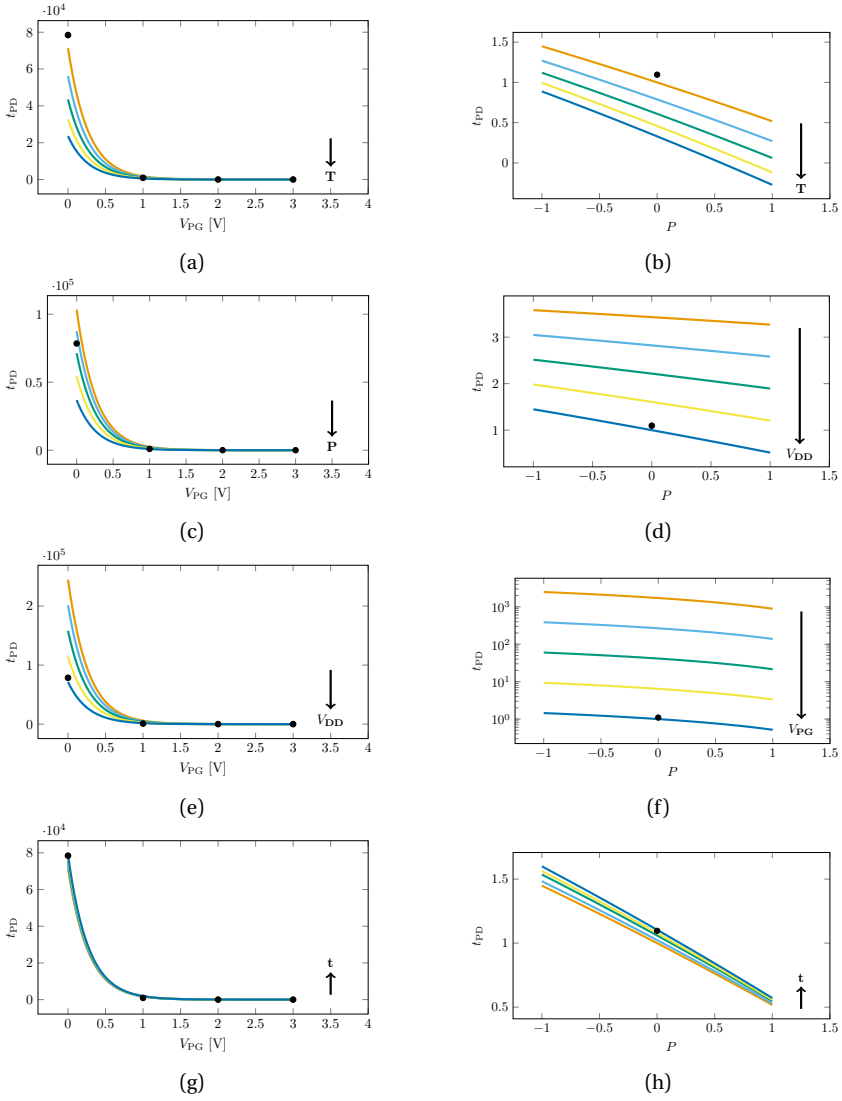


Figure 4.21: t_{PD} model (lines) and fitting data (dots) for the *RFET* technology. The left-hand side figures show propagation delay vs. program gate voltage, parametrized on the four other parameters. The right-hand side shows the delay dependency on process variation. Parameters were chosen to yield five exemplary curves covering the full parameter range, where one curve matches the parameter value for the original fitting data.

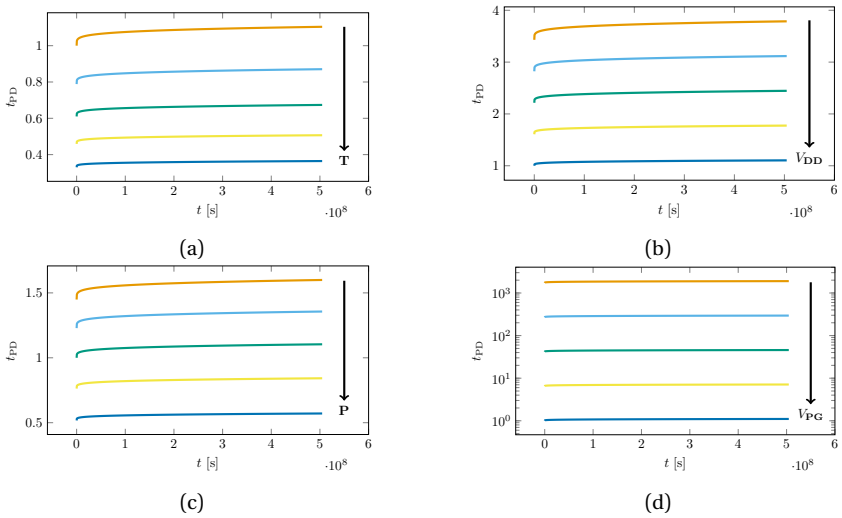


Figure 4.22: t_{PD} model (lines) and fitting data (dots) for the *RFET* technology. Figures show propagation delay vs. aging, parametrized on the four other parameters. Aging was matched to SOI aging at 120 °C and 1.8 V.

Leakage Current Leakage data was extracted from drain currents for a control gate voltage of 0 V from figure 1k in [2]. Program gate voltages were selected from 0 V to 3 V. Data had to be extracted from the *n* mode graphs, as data on *p* mode were not discernible at 0 V. Figure 4.23 shows the measured data as dots and the final model as a line.

For the final model, the measured data was fitted to the following exponential function:

$$I_{leak} = f(C) = C_{ID} * e^{\alpha_{ID} \cdot C} \tag{4.30}$$

The parameters for this model and the test data were calculated as $C_{ID} = 2.813\,599\,16 \times 10^{-14}$ and $\alpha_{ID} = 3.554\,172\,32 \times 10^{-1}$.

4.7 PVTA Scenario Modeling

Previously introduced delay models for SOI and RFET provide the delay for a circuit under certain PVTA conditions and body bias. To simulate the complete FPGA architecture, it is necessary to derive PVTA conditions on the chip as inputs for the delay models. In the following, specific PVTA scenarios used

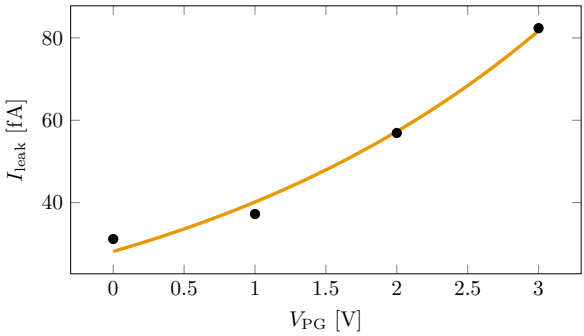


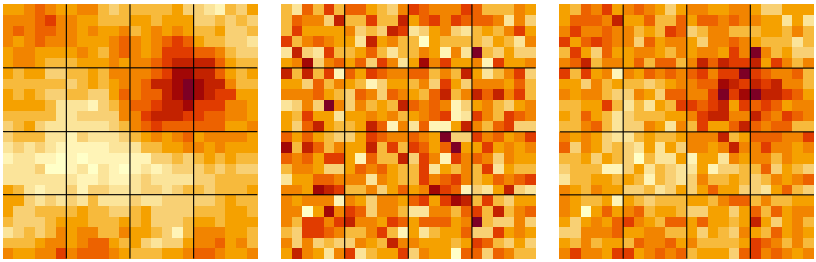
Figure 4.23: Leakage current in transistor for the *RFET* technology. Dots depict values extracted from [2], the line shows simulated values obtained through the model in equation (4.30).

for evaluation in this thesis will be discussed, but the propagation delay models and simulation framework enable simulation of any other PVTA scenario as well. The model's C input used for body biasing will not be discussed here: It is not an external scenario influence, but will be directly controlled by the FPGA architecture instead.

Process Variation As introduced in section 2.4 on page 27, models for process variation are usually technology dependent. As the required technology parameters are not always available, this can make adaption of some models difficult. Some of them also require detailed modeling of the circuit for which process variation is analyzed, e.g. in SPICE. As a primary aim of the framework developed in this thesis is quick evaluation of different architectures, such extensive modelling approaches are not feasible and a simpler model is needed.

The previously introduced VARIUS process variation model [53] fits these requirements: It has few parameters and provides exemplary values, which match the empirical study in [56]. It will therefore be used for all evaluation scenarios in this thesis. The simulation framework derived here however also supports more advanced models such as [60, 61, 247, 248], when required parameter values are available.

The VARIUS model considers two main factors in process variation, threshold voltage V_{th} and the effective gate length L_{eff}. For the discussion here and for the derivation of the P parameter, only the V_{th} part will be used. Following



(a) (b) (c)

Figure 4.24: Simulated intra-die process variation showing the transistor threshold voltage V_{th} for a 24x24 grid. The spatially correlated part is shown in **(a)**, the uncorrelated part in **(b)** and **(c)** depicts the resulting total variation. Brighter color indicates smaller V_{th}.

equations (2.19) and (2.20), 2D process variation maps will be derived as random samples of the stochastic distribution determined by the correlation between values:

$$corr(P_{\vec{x}}, P_{\vec{y}}) = \rho(r) \qquad r = |\vec{x} - \vec{y}| \tag{4.31}$$

$$\rho = \begin{cases} 1 - \frac{3r}{2\Phi} + \frac{r^3}{2\Phi^3} & (r \leq \Phi) \\ 0 & \text{otherwise} \end{cases} \tag{4.32}$$

Values are then normalized to be in range $[-1,1]$ to match the P process parameter. For the simulation, x and y are chosen to be coordinates on the finest FPGA grid, the block grid. This way, each CLB gets a distinct process variation factor. The P parameter is finally given as:

$$P = f(x,y) = \text{map}_i(x,y), \tag{4.33}$$

where map_i is one randomly sampled 2D map of the VARIUS model.

In the evaluation, a single map will be considered. Given the stochastic nature of the variation map, results obtained with a single instance are representative. The parameter values $\Phi = 0.5$, $\sigma/\mu = 0.063$ for random and systemic V_{th} presented in the original VARIUS paper will be used for process variation modeling throughout this thesis. An exemplary 24x24 variation grid for V_{th} derived in this way is shown in figure 4.24.

Voltage Variation As motivated in section 2.4, supply voltage variation affects the propagation delay on FPGAs. In ASIC design, voltage variation is

analyzed deterministically for each circuit, simulating switching activity and power supply wire density to determine IR drop. The information obtained in simulation is then commonly used to increase metal width are wire density locally. This technique can not be used in FPGA, as the local resource utilization and switching activity is not known at manufacturing time of the IC and only becomes known after application placement.

As deterministic analysis is effective, statistical models describing voltage variation for artificial circuits are not commonly available. Such models could be derived through analysis of multiple real circuits, but this is not necessary for this thesis: In a simpler approach, a deterministic, application specific voltage drop model is used for each user application. An IR drop analysis as sophisticated as in ASIC design is however out of scope for this thesis: It requires detailed knowledge of the power supply network of the modeled FPGA architecture and modeling of the metal layers of the used technology. In high-level, early design stage DSE of FPGA architectures, this information may not be readily available.

In this thesis, it is therefore assumed that the power grid in the FPGA architecture is regular: For all locations on the FPGA, an identical utilization percentage and switching activity will lead to the same IR drop. IR drop then does not only depend on a single unit such as one CLB, but on all devices in the vicinity as well, as they share at least parts of the supply. Furthermore, it is assumed that all LEs within a power region have identical supply voltage: This assumption requires that the local power connections within a region are dense enough that there is no voltage drop within the region. Each region then get assigned a voltage drop factor, describing the percentage of supply voltage reduction. To simplify the model, it is also assumed that power usage in regions does not interfere with other regions, so the voltage drop within each region depends only on the utilization within this region:

$$LE(R) = \{e \in R \,|\, \text{type}(e) = LE\} \tag{4.34}$$

$$LE_{\text{used}}(R) = \{le \in LE(R) \,|\, \text{used}(le)\} \tag{4.35}$$

$$\text{K}(R) = \epsilon * \frac{|LE_{\text{used}}(R)|}{|LE(R)|} \tag{4.36}$$

$$V = f(x,y) = (1 - \text{K}(reg(x,y))) * V_0 \tag{4.37}$$

In this notation, $reg(x,y)$ returns the set of all elements in the region which covers position (x,y). $\text{K}(R)$ is the voltage reduction factor for a region R and VDD reduction is obtained according to equation (4.37).

Temperature Variation For temperature variation, similar remarks apply as for voltage variation. There are no commonly used statistical models as their usefulness would be limited. Instead, ASICs are again designed with temperature analysis specific to each single circuit and again this approach is not suitable for FPGA. This thesis will therefore use a high level model for temperature modeling. Like in the voltage variation case, the FPGA is assumed to be regular and whether heating occurs is mainly determined by the utilization in the vicinity. However, unlike voltage variation, temperature variation is modelled as an IC-global effect: Even when a LE is at a physically large distance from the analyzed LE, it is assumed that it still adds to overall heating and produces a temperature increase in the analyzed LE. To model this, the distance between two LEs is first given as the geometrical distance between their coordinates:

$$d(x_1,y_1,x_2,y_2) = \sqrt{(x_1-x_2)^2+(y_1-y_2)^2} \tag{4.38}$$

The heating of a LE at position x,y is then determined according to the following equation:

$$\Lambda(x,y) = \delta_0 * \sum_{(i,j)\in U} e^{-\delta_1 * d(x,y,i,j)} \tag{4.39}$$

Here it is assumed that LEs have exponentially declining impact on the analyzed LE's temperature with increasing distance. The equation then sums up contributions of all LE which are used by the user application, denoted as set U. The rate of the decline can be adjusted using parameter δ_1, whereas parameter δ_0 is used to scale the overall contribution to the ambient temperature. It can be derived when finding the maximum $\Lambda(x,y)$ for an application, then fixing δ_0 to yield the desired maximum temperature increase ΔT:

$$\max_{(x,y)\in \mathrm{U}} \Lambda(x,y) * T_0 = \Delta T \tag{4.40}$$

This definition is dependent on the placed user application. If an absolute reference is desired, δ_0 can be calculated in the same way, but assuming that all LE are used. In this case, ΔT will be the maximum theoretically possible temperature increase if the FPGA is fully utilized.

Using the local temperature increase $\Lambda(x,y)$, the final temperature is derived based on the ambient temperature T_0 as:

$$T = f(x,y) = (1+\Lambda(x,y)) * T_0 \tag{4.41}$$

Previous analysis has only considered static scenarios, where the temperature stays constant over time. As the architectures discussed in this thesis cover

dynamic effects, dynamic heating scenarios will be considered as well. For this, temperature rise in local hotspots will be derived as an exponentially falling temperature gradient, relative to a center position (x_0, y_0). In addition, temperature will be exponentially increasing over time. These requirements lead to the model shown below:

$$\Lambda_t(x,y,t) = \delta_0 * e^{-\delta_1 * d(x,y,x_0,y_0)} * (1 - e^{-\delta_2 t}) \tag{4.42}$$

Here, δ_0 is a parameter for the total temperature rise at the center point, δ_1 models the distance covered by the hotspot and δ_2 determines how long it takes for the rising temperature to reach the final value. The final temperature for the dynamic model is given below:

$$T = f(x,y) = (1 + \Lambda_t(x,y,t)) * T_0 \tag{4.43}$$

This thesis will consider both local hotspot heating over time and global chip heating, which can be caused e.g. due to insufficient cooling of the IC. The same formula will be used for both cases, only the hotspot position and δ parameters will be changed.

Aging For aging, two scenarios can be considered: The first is global aging, which assumes the same conditions on the whole IC. In this mode, the aging parameter A will be derived from equation (4.19) using constant V_A and T_A values. This allows artificial aging simulations for predefined environment temperatures and voltage changes.

The second scenario is slightly more realistic and calculates V_A and T_A based on the temperature and voltage profiles defined before. The aging parameter will be then dependent on x and y as well. The model for this scenario is given below:

$$A = f(x,y,t) = A(t, V(x,y), T(x,y)), \tag{4.44}$$

with $A(t, V_A, T_A)$ from equation (4.19), $V(x,y)$ from equation (4.37) and $T(x,y)$ from equation (4.41). Like the aging model in general, this scenario assumes that V and T distributions calculated at the beginning are constant over time during the aging process.

4.8 Upcoming Aspects

This chapter has presented a high-level overview of the PARFAIT architecture derived in this thesis. Whereas this top-down view provides the overall

idea and picture, individual subtopics will be addressed in more detail in the following chapters. In addition, some aspects regarding simulation and evaluation of the architecture will also be presented.

Chapter 5 will derive a standard cell library for RFET devices and demonstrate a synthesis flow using these. This will enable the development of digital circuits in RFET technology using state-of-the-art commercial EDA tools. In the PARFAIT architecture, this could be used to realize the non-reconfigurable hard blocks, such as DSP blocks or others, in RFET technology. Although RFET technology can be combined with classical MOSFETs on one die, using RFETs for such circuits enables fine-grain performance scaling using the BG. For evaluation of the library, the dissertation will present basic arithmetic circuits and one large circuit, an accelerator for a symmetric cryptography cipher.

Chapter 6 will focus on the main element of a FPGAs, the reconfigurable LE. It will introduce the RFET based logic element used in the PARFAIT architecture and describe integration in the base FPGA architecture as described here. To allow for fairer comparison between the reference architecture and the PARFAIT one, the top-level architecture will be kept unmodified, as described in section 4.1. In order to also keep the FPGA device size similar for both approaches, the RFET based CLB needs to provide similar expressiveness as the LUT based one. Apart from discussing this in detail and evaluating the expressiveness, the chapter also focuses on changes needed in EDA tools to map user applications to such RFET based logic cells.

Chapter 7 will introduce power regions in more detail. As the architecture of power regions is simple, the chapter instead focuses on the changes in EDA tools necessary to support those regions. This includes support to model regions as part of the FPGA architecture description in VPR. Both pre-determined low-power / high performance modes for regions (Static Assignment) and runtime configurable region (Dynamic Assignment) are considered.

Chapter 8 will introduce the PVTA compensation system in detail. It first describes how the required performance for each region is determined by extracting the local critical path in VPR. It then describes the Ring Oscillator (RO) based approach to determine the momentary performance in each region. Based on this information, it continues with a description of the Logic Invasion (LI) system, which dynamically reconfigures parts of the FPGA as RO circuits. This allows to transparently characterize the whole FPGA without impacting the user application. After introduction of these building blocks, the power management controller will be introduced. The chapter concludes with

an overall summary of the PVTA compensation and performance adjustment system.

Chapter 9 concludes the main chapters. It will present methodology for simulation and evaluation of the PARFAIT architecture. For that, it will give an introduction to VFPGAs, which enable prototyping of FPGA architectures on commercial FPGAs. It then continues by describing the static power analysis used in VPR for power evaluation. In order to also model dynamic effects caused by the PVTA compensation system, a QuestaSim based runtime simulation framework is derived. Whereas this can provide a functional simulation of the FPGA architecture and the power management controller, it can not simulate the propagation delay due to PVTA and voltage scaling. The following section in the chapter therefore introduces a co-simulation system. This system combines the PVTA models described in section 4.7 as a software implementation with the VHDL hardware implementation simulated in QuestaSim. The chapter concludes with an introduction of the chosen user application benchmarks for the final DSE and descriptions of the concrete PVTA scenarios which will be evaluated.

The final chapters will conclude the thesis: Chapter 10 will provide DSE results and results for the previously introduced test scenarios. Results will be presented individually for the RFET standard cells, the RFET LE, power regions, and the overall PARFAIT power management system, combining those aspects. Chapter 11 will then summarize results and insights obtained in this thesis and will give an outlook on possible future research.

This page intentionally left blank

Chapter 5

Ambipolar Standard Cells

To analyze application of RFET in non-reconfigurable parts of FPGAs, a standard cell based approach has been taken. The following sections will first explain the derivation of an RFET standard cell library, based on characterization data obtained in the PARFAIT project by Reuter [Reu21]. Based on this cell library, this thesis will investigate application in simple arithmetic units, like they are used as static logic extensions in LEs. To also verify applicability of RFET cells in large circuits and to test mixing of RFET and standard silicon CMOS, the last section in this chapter presents an RFET based cryptographic accelerator.

5.1 Standard Cell Library

Characterization and derivation of an RFET based standard cell library has originally been described in [Reu21]. This chapter will first introduce the RFET device used. It will then summarize the work by Reuter, which consists of modeling and analyzation of the individual cells. The section will conclude with a derivation of a standard library as done by this thesis' author in a contribution to [Reu21].

RFET Device After early works of Reuter had analyzed static device behavior such as Voltage-Transfer-Characteristics (VTCs) [Reu20, Reu19], [Reu21] discusses dynamic characteristics. As no SPICE transient model was available for the RFET technology, logic cells have been modelled and analyzed in TCAD. Figure 5.1 shows the RFET device which was used for the design of the standard cell library. It is based on the work by Krauss [1], but device size has been optimized by Reuter [Reu21]. Providing both BG and TG, the device enables independent biasing of those gates. With a 90 nm FG length but 210 nm BG length, fair comparisons should compare the device to SOI

technology with 210 nm. Reuter therefore compares results to a commercial 180 nm SOI technology, where the transistors have been manually scaled to a gate length of 210 nm. As modification of the commercial standard cell library was not possible, this comparison was only possible for analysis of single cell characteristics. Comparison of the standard cell library was therefore based on the commercial 180 nm standard cells.

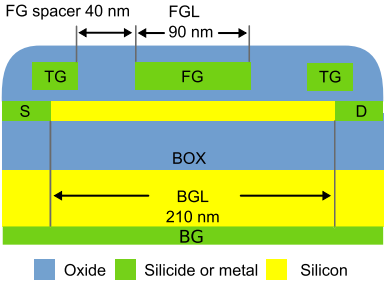


Figure 5.1: The RFET device used for the standard cell characterization [Reu21]. Possessing a 90 nm FG length and a 210 nm BG length, the device can be compared to 210 nm SOI devices.

The cell supply voltage VDD was fixed at 1.6 V to match the commercial SOI technology. In addition, the device has been tuned for symmetric behavior in p- and n- configuration through adjustment of gate work functions. As the device behaves symmetrically, one device can be used for pull-up and pull-down networks, so transistor sizing to compensate drive current is not necessary. After tuning, drive currents I_D were measured at 64.3 µA in n-mode and 83.4 µA in p-mode.

Cell Characterization For development of the standard cells, the devices operate as described in figure 2.6b: The BG of all devices are connected to the devices' source terminals. TG terminals of each device are combined and used as the configuration gate or secondary input. Use of the configuration gate to configure the device as NMOS or PMOS device is shown in figure 5.2a on the left: For NMOS operation, the TG is connected to VDD, for PMOS it's connected to VSS. The FG is used as the primary gate of a transistor.

The drive current of all cells has been normalized to match the drive current of the inverter. *INV*, *NAND* and *NOR* cells do not make use of reconfiguration and use statically configured RFETs as shown in figure 5.2b. The *XOR* cell on the other hand uses the reconfiguration gate as a logic input. The

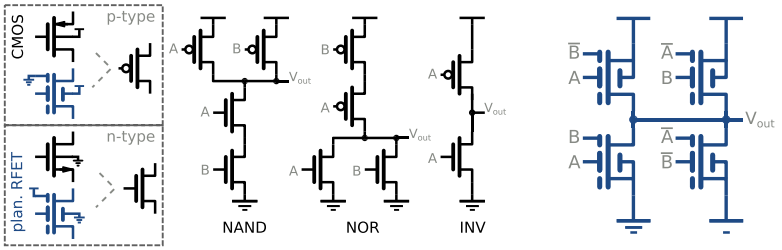


Figure 5.2: Schematics for the RFET based standard cells as designed by Reuter [Reu21]. **(a)** Static biasing of RFET devices to obtain *NAND*, *NOR* and *INV* gates equivalent to conventional CMOS designs. **(b)** *XOR* gate realized using RFET reconfiguration. As described in [Reu20], this device can be realized in 8 transistors, including required input inverters.

cell design shown in figure 5.2b is based on the design in [Reu20] and realizes the *XOR* gate using fewer transistors than is possible in standard CMOS technology.

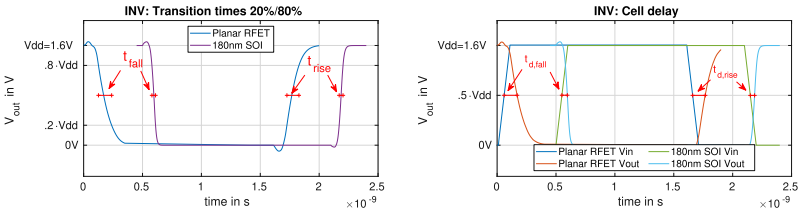


Figure 5.3: Characterization of dynamic cell behavior as described by Reuter for an *INV* cell [Reu21]. **(a)** Characterization of rise t_r and fall times t_f, here called t_{rise} and t_{fall}. **(b)** Characterization of propagation delay t_{PD}, here called t_d. Delays are individually characterized for possible ARCs and edges.

Figure 5.3 shows an exemplary extraction of cell characteristics for the *INV* gate. The characteristics are obtained in the Non-Linear Delay Model (NLDM). In this model, cell characteristics such as propagation delay t_{PD} and rise t_r and fall times t_f are modeled as a lookup table. The table is parametrized by the input transition time and the output capacitance. Furthermore, EDA tools interpolate between table entries. For the output capacitance, Reuter uses fixed values between 3 fF and 15 fF. Compared to other common models like FO4, dynamic capacitance effects are therefore not included. This

trade-off is necessary, as the number of transistors that can be simulated in reasonable time in TCAD simulation is limited. Reuter then performs transient simulations as shown in figure 5.3a for rise and fall times and in figure 5.3b for propagation delay. Similar simulations are executed for the modified 180 nm reference technology to obtain values for comparison. Characterization values are extracted from these simulations using custom scripts. Results show that the reference technology achieves higher drive currents, which also affects the slew rate. For the same output capacitance, depending on the gate type, propagation delay has been 160 % to 638 % slower for the RFET cells. On the other hand, RFET standard cells possess 86 % to 64 % less input capacitance C_{in}. Further details on cell characterization can be found in the original publication [Reu21]. Based on the work by Reuter described in the previous two paragraphs, this thesis' author contributed a cell library for synthesis in Cadence Genus to [Reu21], which will partially be reproduced and summarized in the following sections.

Wire Load Estimation Before considering a library description of the cells itself, a characterization of connections between the cells, wires, is needed. Such wires add parasitic resistance and capacitance to the load of their driving cell. To ensure that the results of RFET synthesis can be compared to synthesis results of CMOS technologies, such wire load effects need to be modeled. Cadence Genus in version 17.11.00 was used for synthesis in both reference SOI and RFET technologies. Genus supports four methods to estimate wire load effects: Wireload models, Physical Layout Estimation (PLE), spatial synthesis and physical synthesis. Wireload models are a statistical solution, providing tables of absolute capacitance, resistance, wire length and wire area. These tables are provided as part of the timing `.lib` file and parametrized on the number of gates in the circuit and the fanout of the respective cell. Additionally, these tables can be parametrized on process corner, supply voltage and temperature, using common liberty file constructs. Wire load models therefore essentially estimate values by averaging over representative circuits of a certain size. For fanout values which are not present in the table, Genus uses linearly interpolated values. PLE uses physical information from `.lef` files and capacitance table files, aiming to obtain more accurate estimations. Physical information most notably includes resistance and capacitance per wire length values. Genus then uses a proprietary algorithm to derive the wire lengths for various connections and calculate the capacitance and resistance for all nets. To provide more exact results, unlike wireload models, PLE calculates the length for each net individually. The third category of wire load estimation includes spatial and physical estimation. Both use the physical information used in PLE and additionally take a user-supplied floorplan into

account. In spatial mode, only a quick initial placement is used. In physical mode, an intermediate step invokes a full layout step to take into account cell placement, routing and congestion effects.

Circuit	Gates	Technique	R [Ω]	C [fF]
fa	5	Wireload	8.0	0.3
		PLE	6.0	1.2
		Physical	39.0	3.8
		Implementation	20.7	2.3
chacha_reg	16029	Wireload	5.0	0.2
		PLE	4.0	0.8
		Physical	13.0	2.1
		Implementation	10.1	1.3

Table 5.1: Comparison of the wire load estimation techniques supported in Cadence Genus, using the SOI reference technology. Values shown are averages over all nets in the circuit. Statistical models show larger deviations for small circuits, as those do not match the average circuit assumed in the model.

Table 5.1 shows estimated values in the reference SOI technology to estimate quality of results for different estimation techniques. Results have been evaluated for the small *fa* and the large *chacha_reg* circuit, which will be used for all analysis in this chapter. In addition, wire loads have been extracted after implementation in Cadence *Innovus* 17.11.000 to evaluate the accuracy of the estimations. For both physical synthesis and implementation, the same simple quadratic floorplan targeting a utilization ratio of 70 % has been used. As can be seen in table 5.1, the wireload model and PLE underestimate capacitance and resistance for the test circuits. The physical flow overestimates values for the small circuit but is more accurate for the large circuit.

Liberty File Derivation Based on the device characterization by Reuter and the wire load discussion, a `.lib` file has been derived to enable synthesis of larger circuits. This file is reprinted in appendix D for reference. Apart from the main definitions, it consists of the library name, the type of delay model used, metadata and the definition of units for all values. Power related units are defined for completeness, although the RFET liberty file currently focuses on timing information, and does not contain power or area information. For the wire load model, PLE or the physical flows cannot be used, as full layout and area information for cells is not available in the current RFET PDK.

Wireload models on the other hand can be included in the RFET standard cell library, enabling fairer comparison to the SOI reference technology. As one of planar RFET technology's main advantages is integration with existing CMOS technologies, repurposing an existing wireload model is possible. As the device size is comparable, and the metal layers will be identical, the reference SOI wireload model will be reused for the RFET library as well. Listing 5.1 shows how this wireload model is defined in the library. Concrete values where taken from SOI liberty files, but have been replaced with dummy values to conform to NDA rules. Line 1 describes one of the tables, defining absolute area, capacitance, wire length and resistance for various fanout values. Synthesis tools choose the respective entry within this wire load table according to the fanout, interpolating the value if necessary. The wire load selection `wload_sel` in line 9 maps the tables is to the area of the synthesized circuit. The final statement then defines the default wire load selection to be used in the library file.

```
1  wire_load_table (wload_1) {
2    fanout_area (1, 1);
3    fanout_capacitance (1, 1);
4    fanout_length (1, 1);
5    fanout_resistance (1, 1);
6    // ...
7  }
8  // ...
9  wire_load_selection (wload_sel) {
10   wire_load_from_area (0, 100, wload_1);
11   wire_load_from_area (100, 500, wload_5);
12   // ...
13 }
14
15 default_wire_load_selection : wload_sel;
```

Listing 5.1: Structure of wireload descriptions for RFET *.lib* file taken from reference SOI technology. Values have been randomized to avoid infringing NDAs.

An example for a cell definition is shown in listing 5.2 on the next page, a reduced version of the RFET inverter definition. The cell definition consists of pin definitions, including power pins (not shown here) and the input pin `A` and output pin `Q`. Each pin section includes the pin direction. For inputs, the input capacitance of the pin on a rising transition and on a falling transition are defined according to the values obtained by Reuter. Outputs contain a logical de-

```
cell (INV) {
  // ...
  pin (A) {
    direction : input;
    rise_capacitance : 0.004;
    fall_capacitance : 0.003;
  }
  pin (Q) {
    direction : output;
    function : "!(A)";

    timing () {
      timing_sense : negative_unate;
      related_pin : A;
      rise_transition (delay_3x3) {
        index_1 ("0.06, 0.30, 0.54");
        index_2 ("0.003, 0.0065, 0.010");
        values ( \
          "0.10, 0.14, 0.19", \
          "0.16, 0.20, 0.25", \
          "0.22, 0.26, 0.31");
      }
      fall_transition (delay_3x3) {
        index_1 ("0.06, 0.30, 0.54");
        index_2 ("0.003, 0.0065, 0.010");
        values ( \
          "0.11, 0.15, 0.19", \
          "0.17, 0.21, 0.25", \
          "0.24, 0.28, 0.31");
      }
      // ...
    }
  }
}
```

Listing 5.2: Shortened version of the RFET inverter cell section in the *.lib* file. Values have been truncated, *cell_rise* and *cell_fall* sections are not shown and power pin related constructs have been removed.

scription of the output as a function of the cell inputs and where modeled similarly to the SOI reference technology. Furthermore, output pins contain delay model tables for `rise` and `fall`, as well as for `cell_rise` and `cell_fall` (not shown here). The `rise` and `fall` tables model the transition times t_r and t_f of the output pin. `cell_rise` and `cell_fall` tables on the other hand model the cell propagation delay t_{PD} from input edge to output edge. Delay tables are indexed by the input slew rate and the total capacitance connected to the output pin. The slew rate is defined to be the duration in which the input edge is between 20 % and 80 % of VDD.

As FF cells are required to implement sequential circuits and the RFET PDK does not provide such a cell yet, the reference technology SOI cell was used in the RFET library. This is justified, as the technology allows mixing RFET and other cells. However, it should be noted that the limited drive current of RFET cells and the comparatively large input capacitance of the FF SOI cell will lead to increased delays. Furthermore, as the RFET cell consists of only five cells, a comparison to a normal standard cell library with hundreds of cells would not be fair: Synthesis tools could choose cell variants optimized for a certain fanout, or they could use advanced cells, which combine multiple basic functions in one cell. In order to allow for fairer comparison, a reduced library has been derived for the SOI technology. This library consists of only those cell types, which are also available in the RFET library.

5.2 Application in Arithmetic Units

After analysis of the dynamic characteristics of the standard cells and the extraction of properties such as propagation delays into a cell timing library, the library was evaluated in demonstration circuits. The selected circuits make use of *XOR* cells, which to demonstrate one main benefit of RFET technology. As shown, *XOR* cells can be implemented using fewer transistors than in the normal CMOS implementation. For a complete evaluation, two methods were examined to map the circuit to the cells in the respective timing library: In the first test with combinational circuits, cells are mapped manually to ensure the netlist of planar RFET and SOI reference technology coincide. In the next section, a normal synthesis approach will be used and will perform individual optimizations on the circuit, depending on the provided timing library. All syntheses are performed for both, the planar RFET technology and the SOI reference. Comparability between planar RFET and SOI reference technology is limited, due to the fact that the SOI reference cell library is based on 180 nm

digital devices. Upscaling of the SOI devices to match channel lengths, as performed by Reuter when comparing the cell performance, is not possible for a digital cell library. The device performance is expected to decrease with upscaling of the channel length, which benefits the presented timing characteristics of the reference technology. The interpretation of the results of the comparison between the two technologies therefore have to take into account a significant boost for the reference technology.

For the direct comparison of identical netlists, a set of simple test applications has been chosen. Those applications resemble circuits that are commonly used in FPGAs and implemented in non-reconfigurable logic. Here, full adders are commonly included in LEs to allow for faster carry-ripple adder implementations. Figure 5.4 introduces the combinational test circuits: A 32 bit carry ripple adder made entirely out of the full-adder cells of figure 5.4a, a 4 bit parity checking adder, as initially proposed for nanowire RFETs in [249], complemented carry generation for the checked adder in figure 5.4b and the two-rail checker for the checked adder in figure 5.4c. Circuits have been transformed to NAND and NOR logic, as they will be directly mapped to gates according to these schematics.

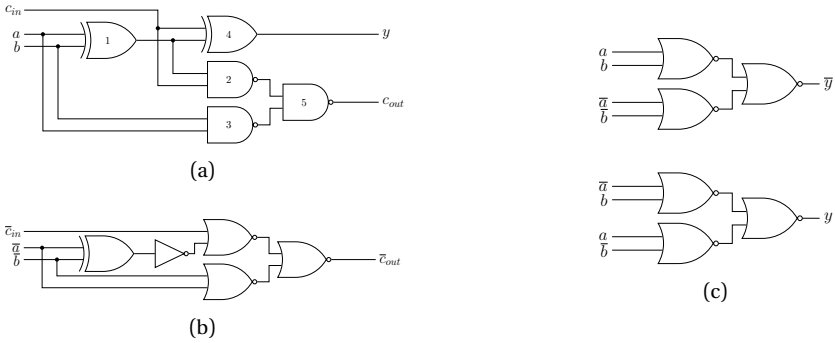


Figure 5.4: Arithmetic, combinational test circuits for logic synthesis. **(a)** Full Adder cell. **(b)** Complemented carry generation for Checked Adder [249]. **(c)** 2RC Checker for Checked Adder [249].

For these small circuits, each single gate was analyzed in detail. It was therefore necessary, that the final netlist is structured exactly as shown in figure 5.4. To ensure this, an essentially pre-mapped, albeit hierarchical netlist was used as input to the Genus synthesis tool: Circuits are modeled in structural VHDL, where gates are modelled as small entities wrapping the respective technology's cell. An example for the *xor2* cell is shown in listing 5.3.

```
architecture parfait of xor2 is
begin
    cellx: entity work.EO2
    port map (
        A => a,
        B => b,
        Q => y
    );
end;
```

Listing 5.3: VHDL description to model the *xor2* cell for synthesis.

To prevent Genus from analyzing the cell functions and mapping to different gates, the `dont_touch` attribute was used for all cells instantiated in the circuits. The `syn_map` phase was then skipped.

5.3 Application in Cryptographic Accelerators

To analyze the use of the RFET lib file in a more realistic way using standard synthesis and mapping, a larger test circuit is needed. To make use of RFET benefits, an application which makes heavy use of *XOR* cells was selected. An accelerator for the ChaCha cipher fits these requirements and will be described in the following. The accelerator was originally designed and evaluated for FPGAs by this thesis' author in [Pfa19].

The ChaCha Cipher

ChaCha is a symmetric stream cipher originally published by Bernstein in 2008 and is now commonly used in various communication protocols [250]. As a stream cipher, the various ChaCha*N* variants first generate a stream of key data, the `keystream`. To encrypt data, the bytes of the `keystream` are combined with the bytes of the `datastream` by an XOR operation, resulting in the `cipherstream`. As decryption is symmetrical, both the en-

and decrypting parties have to generate the same `keystream`. The algorithm for encryption and decryption is therefore identical and shown below:

$$\text{cipherstream} = \text{keystream} \oplus \text{datastream} \tag{5.1}$$
$$\text{datastream} = \text{keystream} \oplus \text{cipherstream} \tag{5.2}$$

In order to generate the `keystream`, ChaCha performs various operations on a matrix consisting of 32 bit unsigned integers. The initial matrix M is formed as follows:

$$M = \begin{pmatrix} 61707865 & 3320646e & 79622d32 & 6b206574 \\ key_0 & key_1 & key_2 & key_3 \\ key_4 & key_5 & key_6 & key_7 \\ counter_0 & counter_1 & nonce_0 & nonce_1 \end{pmatrix} \tag{5.3}$$

The values in the first row are the 16 constant bytes `expand 32-byte k` in hexadecimal notation, and are followed by the symmetric key. The counter is used to provide the current position in the keystream. For the first block, i.e. the first 64 bytes in the keystream, it is zero. For the next block it is one, etc. ChaCha allows for random access to the keystream: It is possible to calculate blocks at any stream offset without calculating any previous block. The last entry, is a number unique to each keystream, the Number-used-Once (nonce).

To process the matrix M, ChaCha$N$ performs $N = 8$, 12 or 20 rounds of operations on the matrix, as shown in listing 5.4:

```
1  foreach i in (0 .. N-1)
2      in = odd(i) ? diags(M) : cols(M)
3      out[0..4] = qround(in[0..4])
4      odd(i) ? diags(M) = out : cols(M) = out
```

Listing 5.4: The $rounds_N(M)$ Operation

Quarter-rounds consist of addition `+`, xor `^=` and rotate `<<<=` operations as shown in listing 5.5:

```
1  a += b; d ^= a; d <<<= 16;
2  c += d; b ^= c; b <<<= 12;
3  a += b; d ^= a; d <<<= 8;
4  c += d; b ^= c; b <<<= 7;
```

Listing 5.5: The $qround(a,b,c,d)$ Operation

Figure 5.5 shows how even rounds operate on the four columns of the matrix (column rounds) and odd rounds on four diagonal vectors (diagonal rounds). Each round then performs the quarter-round sub-operations *qround* once per input vector.

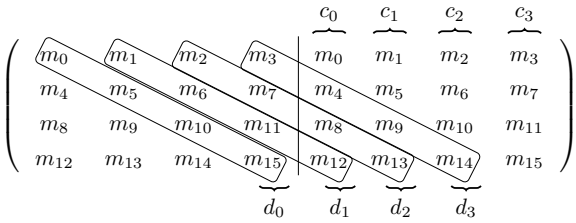


Figure 5.5: Definition of columns c_i and diagonals d_i for the *qround* input data.

After matrix M has been processed through N rounds, the last processing step for the ciphertext block $keystream(counter)$ is to add the processed matrix to the initial matrix:

$$keystream(counter) := M + rounds_N(M) \tag{5.4}$$

To obtain the next 64 B of the keystream, the counter field in the initial matrix M is incremented by one and the process in equation (5.4) is repeated for the new matrix.

ChaCha in Digital Hardware

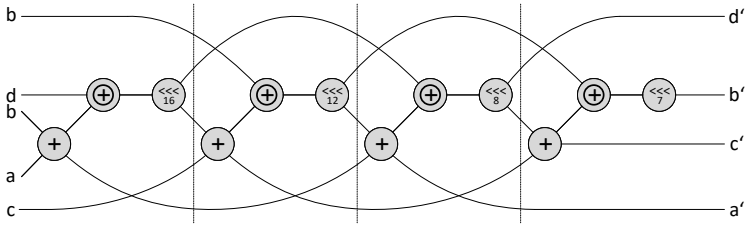


Figure 5.6: Quarter-round operation depicted as a Data Flow Graph (DFG).

Figure 5.6 shows the quarter-round operation of listing 5.5 as a DFG, depicting both the operations used and the data flow of the algorithm. As can be seen, each output a, b, c and d is dependent on each input and on intermediate

results, showing that parallelization of this operation is not easily possible. Figure 5.7 shows a slightly modified form of the first two sections of the graph, suggesting that the whole operation can be built out of four basis cells. These Add Rotate XOR (ARX) cells form the base operation used in the ChaCha cipher, but outputs need to be permuted in order to directly chain these cells: The outputs of the first cell, a', b', c' and d' are updated input variables as described in line one of listing 5.5. Line two of listing 5.5 then applies the same operations on the updated input variables. As it maps the inputs differently to the operations, the basis cell has to perform that permutation. If four basis cells are connected serially, the final result will be in the same order as the input variables.

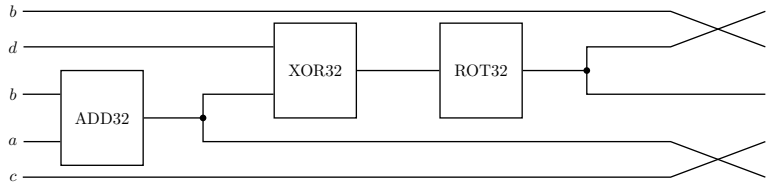


Figure 5.7: ARX cell with permuted outputs to realize the *qround* operation.

As shown in the graph, the rotation distance is different for each stage. To handle this in the basis cell, the distance can be required to be a constant parameter. The rotation operation will then be implemented as a simple wire permutation by synthesis tools, introducing no additional logic delay. As such a cell can be used for only one stage in the quarter-round, four physical copies of these cells with different rotation distances will be required. An alternative is to make the rotation distance changeable as a runtime input: In that case, the rotation operation will be synthesized into a 4-input multiplexer structure, requiring additional hardware resources and introducing logic delay. As explained in more detail in [Pfa19], it is possible to pipeline this ARX basis cell.

Quarter-Round and Rounds Two different options to combine ARX cells have been evaluated. One option is to employ a pipeline structure as depicted by the sections in figure 5.6. In this implementation, the quarter-round is implemented as one pipeline consisting of four physical ARX cells. This pipeline does the complete round processing for a quarter of the matrix and each operation can be mapped to one physical ARX cell. The rotation distance for each cell is constant, reducing logic delay. Another benefit of this structure is the possibility of deep pipelining: As the ARX basis cell is pipelined, an in-series

chain of these cells should not pose further restrictions on the critical path and overall performance.

In order to utilize such a deep-pipeline completely, all stages have to be filled with independent data vectors, i.e. any vector in the pipeline may not in any way depend on the final *qround* result for any other vector in the pipeline. As can be seen from figure 5.5, and in listing 5.4, the four quarter-rounds in each round are independent. Their inputs depend on the results of the previous round, but not on any result of other quarter-rounds in this same round. Problems however arise after the vectors of one round have been processed: The operations of the next round have data dependencies on all data vectors modified in the previous round [Pfa19]. One solution is to start processing the next round only when previous columns have been completely processed, but this reduces the performance of the implementation. As individual keystream blocks in ChaCha are independent, another solution is to prepare the initial matrix M_{n+1} for the next keystream block and process this matrix' rounds interleaved with the M_n rounds. This idea can be generalized to any number of matrices, depending on the pipeline depth. In general, processing then alternates between the rounds of multiple matrices.

Alternatively, a more software like approach where at least one ARX cell is used with a configurable rotation distance was evaluated. This way, the four stages of a quarter-round can be processed consecutively and the result of every step will be written back to the processing matrix. Four ARX cells were still used in parallel to enhance throughput, similar to Single Instruction Multiple Data (SIMD) in software optimizations. In this implementation, four parallel ARX cells will process the whole matrix at once instead of processing vectors consecutively. As one cycle always yields a complete matrix, there are no pipelining complications when starting to process the next round. It is therefore not necessary to alternate processing between multiple matrices. The main drawback is that this approach requires a runtime-adjustable rotation distance, as the distance will be different for each processing cycle.

After the matrix has been processed through the $rounds_N$ implementation, the final addition of equation (5.4) and the encryption or decryption operations equations (5.1) and (5.2) need to be performed. As the final operation is an addition followed by an `XOR` operation, it is possible to reuse the ARX cell for this operation.

ChaCha Accelerator System Architecture Based on the previously introduced building blocks, three different system architectures have been evalu-

ated. The pipeline implementation consists of four parallel quarter-round cells, calculating one round completely in parallel. The individual Round blocks are then chained to form a deep pipeline, taking care to permute signals connecting two rounds. If the output `datastream` is not ready or the input `datastream` is not valid, a FSM accordingly stops the Round blocks to pause keystream generation.

The Block Memory and the Register implementation share most modules. The core blocks which calculate the complete $rounds_N$ however differ between the two implementations. In either case, processing a matrix in a Core block takes a certain amount of cycles. The top level module therefore allows to use a configurable number of cores in parallel, interleaving their outputs and enhancing throughput. As the outputs are interleaved, the maximum number of cores is reached when every cycle yields 16 words of data. Any further parallelization then requires duplicating the top-level architecture.

For the Block Memory Implementation, one pipelined quarter-round is used. The four output words are then saved to four parallel Block Rams at certain indices: Each row of the matrix is kept in one Block Ram, which then allows to read all four inputs for a column round or for a diagonal round in parallel using proper addressing. The benefit of this implementation is that all operations in the quarter-round are placed into one physical pipeline implementation with constant rotation values.

For the register based implementation, no complex address calculation needs to be done for memory access. Instead, after the initial matrix has been loaded using a multiplexer, the complete matrix is kept in register storage internal to a parallel quarter-round implementation. The outputs are then looped back to the inputs. The block operates on the whole matrix, processing one fourth of a round per cycle.

RFET Implementation

After the cryptographic accelerator has been designed and evaluated on FPGA, this thesis' author used it for RFET standard cell evaluation in [Reu21]. Out of the ChaCha implementations, only the Register variant shown in figure 5.8 was used for RFET synthesis. This variant is smaller than the Pipeline variant and does not need block memory, which is not available in the minimal standard library. Instead of analyzing every single gate as done for small circuits, the purpose of this larger circuit is mainly to verify proper synthesis

and to evaluate wire load estimation approaches on large circuits. Because of this, standard behavioral VHDL coding techniques have been used and RFET `.lib` file with the *INV*, *NAND*, *NOR* and *XOR* cells has been supplied to Genus, instead of manually mapping cells. A similarly reduced library with the same types of cells was used for the SOI technology, to allow for direct comparison. As technology mapping is done entirely by the synthesis tool in usual digital standard cell flows, this approach is closer to common practice.

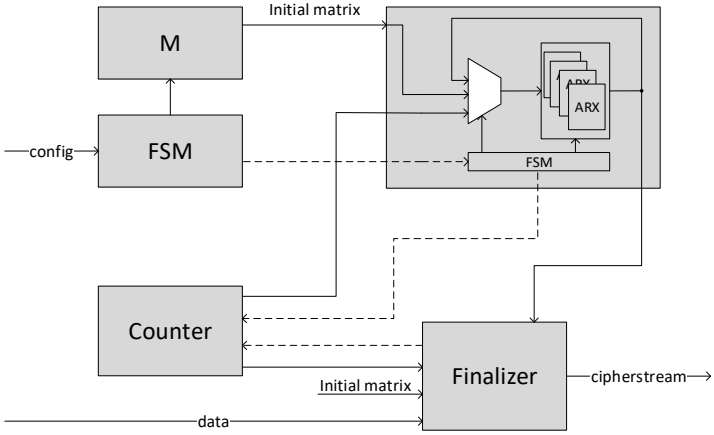


Figure 5.8: Simplified system architecture of the Register ChaCha accelerator variant [Reu21].

The timing library of the planar RFET is limited to combinational cells and therefore only allows for synthesis of combinational circuits. To be able to evaluate sequential circuits, the `.lib` a D-FF was added to the library. The deployed D-FF comes from the SOI reference technology `.lib` file as explained previously. The input capacitance of the D-FF is approximately half the capacitance of the RFET inverter input pin and the timing tables support output loads of up to 80 times the maximum input capacitance of all pins in the RFET library. The cells are therefore compatible for use in the timing analysis.

Chapter 6

Ambipolar Reconfigurable Cells

As applicability of RFET in non-reconfigurable digital logic was evaluated in the previous chapter, this chapter now focuses on the application of reconfigurable cells. A summary of available reconfigurable cells was given in section 2.6 on page 53, whereas reconfigurable cells in the PARFAIT RFET technology have only become available during writing of this thesis. This chapter therefore focuses on use of ULM cells in FPGA in general, independent of the technology used and of the concrete cell design. The content of this chapter has been previously published in [Pfa20], but it has been edited and extended for this thesis.

6.1 Basic Logic Cells

Various existing ambipolar technology reconfigurable cells were considered for an FPGA architecture which realizes the opportunities offered by ambipolar transistor technology. As a result, the *CNT-DR8F* cell presented by Liu et al. [128] was selected as a base for this work: Compared to simpler 2-input cells with only two selectable functions, this eight-function cell offers more choices for the mapping and packing algorithms. Unlike some larger proposed cells, this cell does however not provide the complete function set of two variables. It is therefore not a 1:1 replacement for LUTs: A cell which allows to represent all possible functions of N input variable can be treated like a LUT in most of the EDA tool flow. It therefore puts fewer restrictions on the EDA tools than cells with a reduced function set. For specialized RFET reconfigurable cells, it is often not easily possible to influence the realized function set. Because of that, it is important to evaluate the EDA tool flow and the architecture in regard to a real ambipolar base cell. This will take into account restrictions which may arise due to the limited set of functions in the used cells. *CNT-DR8F* offers the function set shown in table 6.1. The table also

shows the names of the functions as they are used in the rest of this chapter, the EDA tools and the FPGA architecture. Simple permutation of cell inputs can easily be done in the FPGA interconnect and does not need to be implemented explicitly in the reconfigurable cell. The function pairs ($\overline{A}\cdot B$, $A\cdot\overline{B}$) and ($A+\overline{B}$, $\overline{A}+B$) supported by the original cell have therefore been combined into single `ulm_and2n` and `ulm_or2n` functions.

$V_{\mathbf{bA}}$	$V_{\mathbf{bB}}$	$V_{\mathbf{bC}}$	**Y**	**Name**
+V	+V	+V	$\overline{A+B}$	ulm_nor2
+V	+V	-V	$A+B$	ulm_or2
-V	-V	+V	$A\cdot B$	ulm_and2
-V	-V	-V	$\overline{A\cdot B}$	ulm_nand2
+V	-V	+V	$\overline{A}\cdot B$	ulm_and2n
+V	-V	-V	$A+\overline{B}$	ulm_or2n
-V	+V	+V	$A\cdot\overline{B}$	ulm_and2n
-V	+V	-V	$B+\overline{A}$	ulm_or2n

Table 6.1: The set of functions supported by the *CNT-DR8F* ULM cell. The cell operates with two configuration voltage levels and is using in dynamic logic. Adapted from [128].

Logic Generator Table 6.1 also shows the configuration values to obtain a certain output function for reference. For the following discussions, the exact ($V_{\mathrm{bA}}, V_{\mathrm{bB}}, V_{\mathrm{bC}}$) combination used is however not relevant. If the physical view considering the voltages is mapped to a logical view considering configuration bits of SRAM cells, the combination determines essentially the bitstream encoding used. An important point is the number and type of used reconfiguration inputs. Some reconfigurable cells use three values for the configuration input: A positive, a negative and a zero voltage. In those cells, the positive and negative voltages are used to configure n-channel or p-channel behavior. The zero voltage is used to put the transistor in an off, high-impedance state. Such three-level configuration can however not directly be represented in SRAM, so a decoder network is needed for these cells. The selected *CNT-DR8F* cell uses two input values and three inputs. The configuration storage for such an element can therefore be kept entirely in three SRAM cells. If the chosen ambipolar technology supports to store the configuration as charges of the respective configuration gates [167], external SRAM storage is not needed. In this case, logic and memory are combined, so the extra transistors otherwise used for configuration memory can be saved. The total transistor count for

the *CNT-DR8F* cell, including configuration storage, then reduces to seven transistors [128].

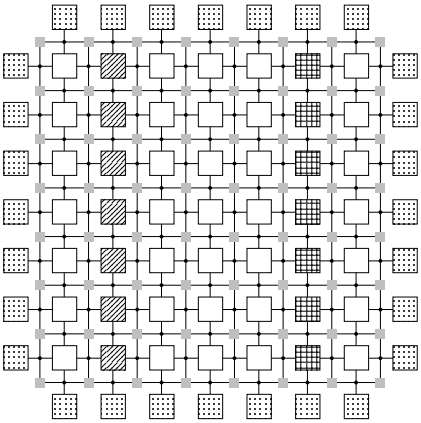


Figure 6.1: Top Level FPGA Architecture. Dotted: IO blocks; white: Configurable logic blocks; diagonal: Memory blocks; grid: Multiplier blocks. Multiplier and memory blocks repeat every 8th column. Wire length of the global interconnect is 4.

FPGA Architecture The RFET FPGA architecture is based on the VPR reference architecture `k6_frac_N10_mem32K_40nm` shown in figure 6.1. In addition to being the base for the RFET architecture, this architecture will also be used as a reference to compare the results to. Certain differences between the RFET reconfigurable cell and LUTs have to be considered when designing the RFET FPGA: As *CNT-DR8F* is a cell using only two inputs, it has less expressiveness than the 6-input LUT commonly used as LEs in FPGAs. Using it directly as a configurable logic block in the FPGA grid of the reference architecture therefore leads lots of wires having to be routed using the global interconnect. Such routing leads to increased wire delay, negatively affecting the critical path and severely limiting the maximum frequency achievable for user designs in the FPGA architecture. It may also lead to routing congestion for some user applications. As the interconnect in modern FPGAs already makes up most of the total area, using even wider interconnect channels is not an option. The RFET based architecture evaluated here will therefore keep identical channel width as the reference architecture which is used for comparison. To ensure the global interconnect can be kept unchanged, the RFET-based CLB replacement will be designed to possess a similar expressiveness as the reference architecture CLB. The discussion in this chapter will be

kept on system level: For example, cell area largely depends on the technology and cell used. In a more detailed analysis, cell area however does have an effect on the application performance and on interconnect design. Future, circuit-level investigation will therefore require a more detailed analysis of the logic cell and a complete circuit level reimplementation of the cell in the target technology, including a cell layout. Because of this, circuit-level analysis is deferred to future work.

Fracturable Logic Cell Aiming to stay close to the design of the VPR `k6_frac_N10_mem32K_40nm` reference architecture, the fracturable 6-input cell structure shown in figure 6.2a has been derived. It combines 5 ULM base cells in a tree-like structure, providing three outputs. In this base structure, the outputs are not independent and can only be used in parallel if an intermediate output is required. Analysis of the packed netlist will show whether such situations are common enough in the netlists of real applications to be beneficial. The maximum depth of the cell is three ULM cells deep, but it is not a fully populated tree. Depths of one ULM or two ULMs (full tree) are available at the other outputs. Such a cell does however not yet allow to implement all required functions on an FPGA: As an example, the case where an IO input is directly connected to a register cannot be represented in this architecture, as all signals have to be routed through a logic blocks before being fed into a register. LUT based architectures are not affected by this problem, as they always allow to implement the identity function, a simple pass-through. To make the ULM based architecture universal, a pass-through mode has to been added to the ULM element, as will be explained in the following sections.

6.2 Electronic Design Automation

To evaluate various parameters and compare the designed logic cells to LUT based FPGA, an evaluation methodology is needed. Like most FPGA research, this work will be carried out in an empirical way, analyzing various representative benchmarks applications. Evaluation and benchmarking of the different architectures will be performed in the most recent VTR release, version 8.0 [142]. The VTR tool suite has originally been designed for LUT based FPGAs, which means various changes are necessary to use it with fixed function ULM cells. An approach to handle this was originally presented in [Pfa20] and will be summarized here. In addition, a novel, more advanced EDA approach

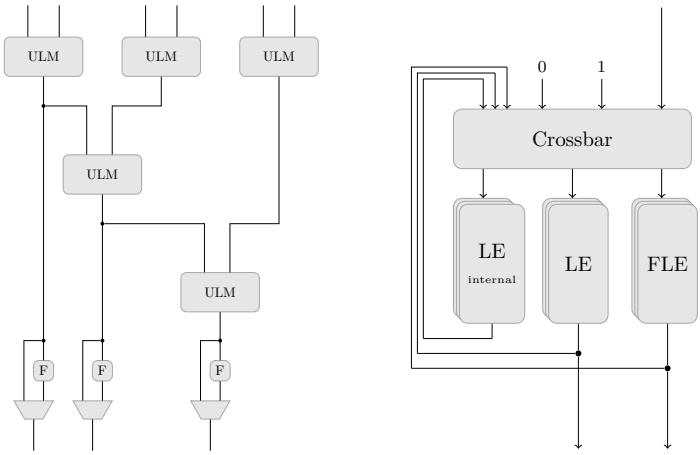


Figure 6.2: Basic structure of the ULM based LE and logic cluster. **(a)** ULM based 6-input FLEs. **(b)** Configurable logic block cluster of LEs and FLEs.

yielding better results for fracturable cells will be presented in the second part of this section.

Basic EDA Flow Figure 6.3 on the next page shows the most basic modifications of the VTR flow that are needed to synthesize for ULMs. Whereas tools depicted in gray and black are unchanged from the original VTR flow, those in orange are newly introduced or modified. User applications, in case of VTR the benchmark applications, are first passed to ODIN 2 for synthesis. This step is largely technology independent, except for the direct use of black boxes and other IP cores in the Verilog code. Use of special mathematical or memory operations may also need a DSP or block memory element in an architecture description for ODIN. These descriptions do not differ between ULM and LUT based FPGA and are therefore not further described. After synthesis, ABC performs technology independent optimizations. This step is again unchanged. The primary change is in the next step, technology mapping in ABC. Technology mapping is guided by a synthesis script, which is automatically generated by the VTR tool flow. The generated script however assumes mapping to LUTs and cannot be used for ULM. This script was adapted to perform a standard cell mapping, using a custom `genlib` technology file. The `genlib` library usually contains available cells in a standard cell library. In this case, the different ULM modes have been specified as individual gates.

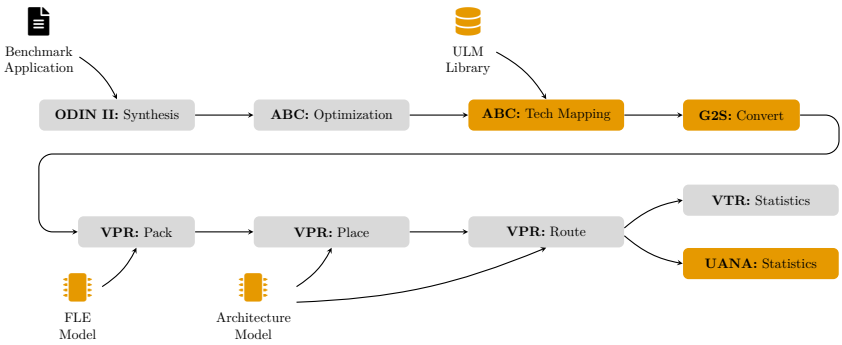


Figure 6.3: Custom EDA flow to map VTR benchmarks to ULMs. Steps in orange have been newly added using custom written tools. Libraries and models in orange have been newly derived for the specific ULM used.

As a result, ABC will map the user application logic to those functions. As a slight complication, VTR sometimes performs multiple iterations of reading, optimizing and writing the mapped netlist. This is supported by ABC when using LUTs as the target technology, as the representation of logic in unmapped netlists happens to be the same as for LUTs. For standard cell synthesis, this is not the case and care must be taken to perform initial ABC steps using LUT mapping, only switching to standard cell mapping in the final iteration.

The mapped netlist generated by ABC in cannot be directly used in the pack and place phase of VPR: The generated Berkeley Logic Interchange Format (BLIF) file uses `.gate` directives, which are not supported in VPR. Therefore, the custom G2S script is used to transform the `.gate` directives to `.subckt` directives. Those directives are supported by VPR and are usually used to specify black boxes and more complex predefined blocks in the FPGA architecture. As the VPR packer was designed to be flexible, it can be used to pack the ULMs functions into the FLEs. For this, a custom FLE model is provided to VPR. It is modeled to consist of ULMs, which are modeled as logic block with different modes. Each mode corresponds to one function of the ULM cell and to one entry in the ABC gate library. The configuration of each individual ULM can therefore be obtained by simply extracting the used mode from VPR results. A major drawback of this simple approach is extended tool runtime, as an unusually large amount of cells has to be processed by the packer. Furthermore, some logic functions cannot be legally packed onto some ULMs in a FLEs due to routing restrictions. This will lead to a lot of

backtracking in the packer. As in the LUT case, the packer is also used to pack FLEs into logic clusters.

Place and route steps are largely unmodified. VPR needs an architecture description which declares the location and amount of logic clusters. Here, no complex modifications are necessary and this thesis will adapt a reference architecture from VTR. To evaluate results, statistics will be extracted as will be described in the remainder of this chapter. Various top-level statistics such as FPGA size and channel width can be obtained directly from VTR. Other interesting statistics, such as the utilization of FLE in logic clusters, the modes ULM operate in etc. are not available from VTR's statistics. The information is however available in the mapped netlist generated by VPR. Therefore, the custom tool UANA was written to extract the relevant metrics.

Flow for Fracturable Cells As will be analyzed in the evaluation, the simple EDA flow presented in [Pfa20] has significant limitations. As previously described, it slows down the packing phase a lot. Furthermore, mapping to FLE has shown to be inefficient. Although in general VPR is able to pack multiple ULM into a FLE, output and input utilization of those FLE is in average underwhelming. As the problem is mainly in the packing step, a modified flow as shown in figure 6.4 on the following page has been implemented. Elements shown in blue have been modified compared to the previously described basic flow.

Instead of modeling the base primitives in the technology mapping, a more exhaustive technology library was derived. This library contains every complex N-input logic function which can be represented by the FLE. It includes both functions which use the whole FLE and those, which only use a part of the FLE and can be used in fracturable mode. Here, fitting more complex functions in a FLE makes less use of the local and global interconnect than mapping to multiple, simple functions. This should be reflected in the generated library to incentivize the synthesis tool to use the larger functions. Adjusting the cost of functions this way has been implemented by ensuring that larger functions are virtually assigned a smaller cost, i.e. area.

The G2S step now becomes slightly more complicated: It still adapts `.gate` directives into `.subckt` directives, but in addition, it also transforms the netlist from the complex, modeled function to ULM FLE mode. A FLE mode is a configuration, which differs in more than only the ULM function used. For example, a FLE can be in fractured mode realizing a two-input and a four-input function. Or it might only realize a single 6-input function. Two,

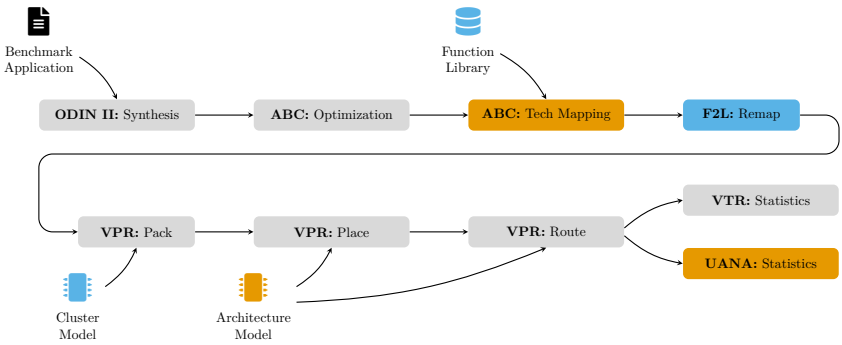


Figure 6.4: Advanced custom EDA flow to map VTR benchmarks efficiently to ULM based FLEs. Tools, libraries and models in orange have been newly added compared to the standard VTR flow. Components highlighted in blue have been modified or extended compared to the simpler approach of figure 6.3.

four and six-input are therefore modes, but a two-input *NAND* and a two-input *NOR* would be mapped to the same mode, as both can be realized by the same FLE topology and only need different ULM configurations. When this mapping step is performed in the F2L tool, it should be noted that the information about which function exactly is implemented in each directive is lost. While removing this information is the main step to simplify packing, it needs to be restored in bitstream generation. For packing, FLEs are now simply described as the different modes. ULMs do not need to be described in VTR anymore at all. The task of the packing tool is therefore reduced to its original task, packing multiple logic functions into one FLE. Packing FLEs into logic clusters then works as usual and the remaining EDA toolflow operates in the same way as in the basic flow.

6.3 Design Methodology

As the expressiveness of such a 6-input cell is still limited, the ULM based FLE cell has been embedded in a cluster, as shown in figure 6.2b on page 181. This is also consistent with the VPR reference architecture, which uses 10 6-input LUTs in each logic cluster. Like in the reference architecture, the inputs of the cluster are connected to the inputs of the FLEs using a full

crossbar and the outputs of the cell are fed back into the crossbar. This allows to route connections between multiple of these cells locally, without using global interconnect. The logic element is supposed to have the same expressiveness as the one of the reference architecture, which means the FPGA device size and global interconnect usage should be the same for a set of benchmark circuits. The exact amount and type of FLEs in a complex cluster will therefore be determined according to this goal in the following sections.

Benchmark-Driven Design The actual benchmarking of the architecture is performed using VTR's benchmark set, whereas for evaluation of intermediate architecture results, only a reduced set of benchmarks is used to reduce tool runtime. To evaluate the results and guide architecture design, the statistic results offered directly by VPR are useful, but not sufficient: These statistics are mostly centered on the top-level view of the FPGA architecture, including the FPGA dimensions in blocks of the top level grid, the block type which dominates the device size, channel width and congestion for global routing, and similar data points. The main conclusion that can be drawn from these measurements is whether the objective of performing similarly to a LUT based architecture, with respect to the global architecture, has been fulfilled. Here, a comparison between the VTR reference architecture and the same architecture with the LUT based logic block replaced by ULM logic clusters will be carried out.

Custom Analysis In the design phase of a replacement logic cell which is supposed to integrate in an unmodified global architecture, evaluating these global aspects is of limited use. Bottlenecks such as reduced logic expressiveness caused by too few logic elements in a cluster, by restricted local routing or because of underutilized logic elements can not be found in these top-level statistics. To remedy this, a custom tool for analysis and statistics collection for the logic cluster blocks has been designed. Parsing VPR's structured, packed netlist output, it is possible to gain interesting information about the sub-blocks in the hierarchy instead of only top-level information. Quantities which will be analyzed include the utilization of input and output ports of the logic clusters, the utilization of the available FLE cells in a cluster, utilization of inputs and outputs of the FLE cells, ULM configurations used in the FLE cells and similar metrics. Statistics are the generated for aggregated quantities, such as the average and median number of inputs used, and similar values. To make informed decisions on the architecture, histograms are generated in addition. Those can for example be used to gauge how likely outputs, inputs or FFs in a FLE are used. In addition to aggregated statistics, statistics for

individual outputs and cells are generated. This allows to answer questions such as how often a specific output or cell is used.

6.4 Logic Clusters

Changing certain parameters in the FLE and logic cluster can largely affect the expressiveness of the logic cluster. Cell design therefore needs to be guided through a set of measurable variables, which will be obtained through analysis of user application benchmarks. The following quantities have been selected to evaluate the expressiveness of the overall logic cluster:

1. Total FPGA size: The width and height of the FPGA in complex blocks. This measurement is only relevant if the logic blocks (LUT- or ULM-Cluster) determine the device size. Both points can be determined from the statistics file generated by VPR.

2. Percentage of FLEs used in logic clusters: This measurement gives an overview of how well device logic resources are utilized and allows drawing indirect conclusions on congestion issues of the global and local interconnect. Non-fully utilized FLEs (if the device size is limited through logic cells), hints that the cells cannot be connected accordingly. This may be caused by congestion on global or local interconnects, as well as by not enough available inputs or outputs.

3. Logic cluster input and output utilization: A high utilization of inputs or outputs with little FLE utilization suggests that excessive amounts of signals have to be routed using the global interconnect. Improving connectivity of the local interconnect can unburden the global interconnect and allow for better FLE utilization.

4. FF utilization: The amount of FFs which are actually used. This measurement is particularly sensitive to the set of benchmarks used. If the ULM based logic cluster is designed to have the same expressiveness as a LUT based cluster, it is also expected to see the same amount of FFs in such a cluster as in the reference architecture.

In addition to those quantities for logic cluster design, additional quantities can be used to evaluate the FLE. The following quantities are therefore also evaluated:

1. FLE input usage and distribution: Information about the average number of inputs used can suggest whether a FLE with more or less inputs may lead to a better utilization of ULM cells. A low number of used inputs suggests that the input circuits do not map well to the ULM topology, whereas a low number of distinct inputs suggests that some ULM inputs may be combined into a single input, to reduce size of the logic cluster crossbar. The distribution of inputs can also be used to gain certain insights: It allows drawing conclusions which part of the fracturable logic is most often used.

2. ULM usage and distribution: This is another measurement to determine which part of the fracturable logic is most used. It further provides direct feedback whether the FLE cell successfully matches the input logic functions, or whether the topology of ULMs does not allow nets to be mapped efficiently.

3. FLE output usage and distribution: This is the primary way to determine whether a FLE design is working efficiently. A low number in the average amount of used outputs suggests that the cell can not drive multiple outputs at the same time, likely caused by the cell topology. A low utilization of a certain, specific output can hint that this sub-part of the FLE topology is not frequently used and that the output may be removed from the architecture without reduction of expressiveness.

LE Inputs Figure 6.5a on the following page shows the LE utilization for a very simple, naïve FLE which consists of only one ULM. This LE is instantiated ten times in the cluster, according to figure 6.2b on page 181. The resulting architecture has been modelled in VPR and the VTR MCNC benchmarks have been evaluated for it. As can be seen in figure 6.5, the results motivate the need for a more complex cell. In the architecture, VPR makes use of only 9.64 inputs and 3.5 outputs in average, although the ULM utilization in figure 6.5a is at 96.1 %. This clearly shows that even when all ULM are fully utilized, not all inputs and outputs can be used. In such a situation, either the number of inputs and outputs needs to be reduced, or the LE in the cluster needs to be changed. Furthermore, the amount of complex logic blocks used in the benchmarks increased by 241.4 % compared to the reference architecture. Caused by fewer used inputs and outputs, the minimum channel width is also reduced to 77.0 %. As the FPGA top level architecture is supposed to be kept the same, this clearly indicates that a 2-input ULM is not expressive enough as a basic logic element.

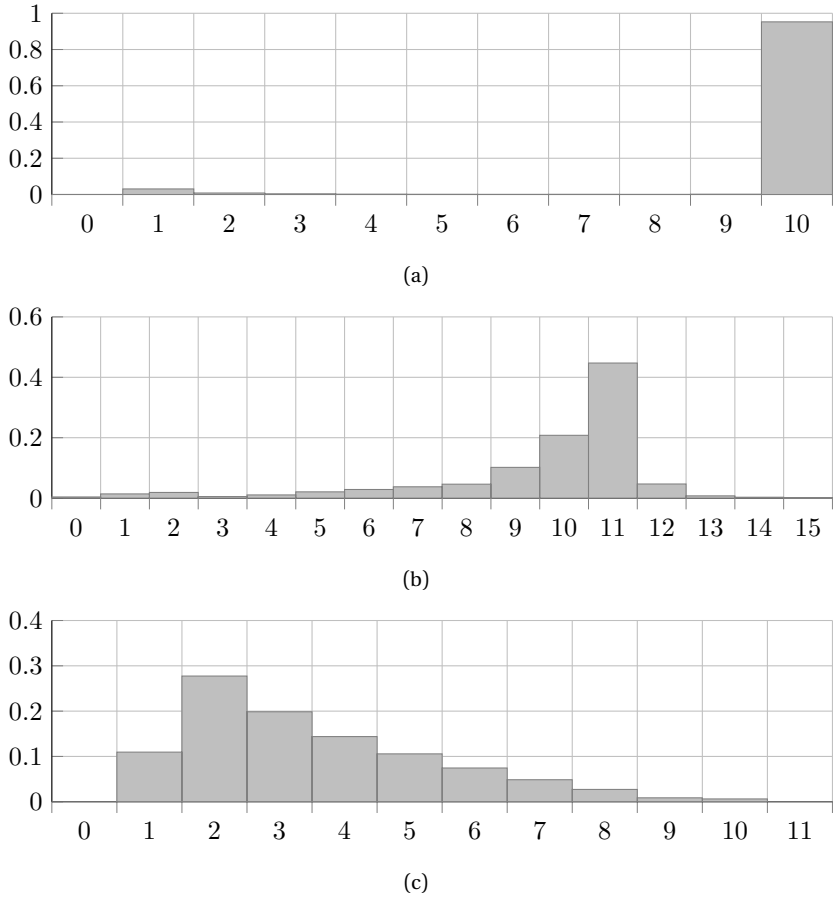


Figure 6.5: Logic cluster utilization when using a simple LE consisting of one ULM. **(a)** Utilization of LEs in the cluster. **(b)** Utilization of cluster inputs. Unused values not shown, in total 40 inputs are available. **(c)** Utilization of cluster outputs. Unused values not shown, in total 40 outputs are available.

Fracturable LEs As an initial improvement, the combined cell in figure 6.2a on page 181 was derived: Like the 6-LUT building block in the reference architecture, it can take up to six different inputs signals. This decision has been made to initially have similar routing requirements in the local interconnect of the logic cluster as in the reference architecture. This is supposed to yield higher input utilization for clusters even with an unchanged input crossbars. In addition, the cells have been arranged in a way to allow the system to be fracturable: If ULM 4 in the tree is configured to pass through its right-hand input, the 6-input FLE decomposes into a two-input and a four-input logic element. Similarly, the cell can be decomposed into three two-input logic elements by further putting ULM 3 into bypass mode, forwarding only its right input. As previously explained, multiple outputs can still be derived from all six inputs, if the outputs are related and their combinational logic functions can be mapped to the topology shown in figure 6.2a. Reevaluating the changed architecture yields the results in figure 6.6, where it can be seen in **(c)** that only one output is heavily used. Further analysis shows some functions mapped to this output actually are two-input functions, which forces other ULM cells into bypass mode and essentially implements simple functions in overly complex cells. The main problem lies within the overly simple technology mapping and packing approach: Directly mapping onto ULMs in synthesis and combining multiple such mapped functions into more complex logic blocks in the packing stage prevents the synthesis tool from transforming functions directly for those larger cells. Another drawback of the simple FLEs as a simple set of ULMs also increases calculation effort in the packing stage, which will result in increased tool runtime. This limitation is not present in the advance toolflow presented later on.

Internal LEs In order to reduce utilization of the global interconnect, logic clusters usually provide a local interconnect with direct feedback paths from the logic output to the logic input. If an output is fed back in the local interconnect, unless it is also required as an input for another logic function, it is not routed on the global interconnect. This means that larger utilization of the local interconnect will lead to less utilization of the logic cluster outputs. In order to avoid wasting resources, there are two possible solutions: One is to reduce the number of outputs. This however has consequences for the global interconnect and the overall FPGA architecture. As the original top-level architecture should be kept close to the reference architecture, the number of outputs in a logic cluster should be kept the same. To still achieve higher output utilization, internal-only cells have been tested: The output of these cells are only connected to the local interconnect and to other logic cells' inputs in the same cluster. They are not connected to outputs of the

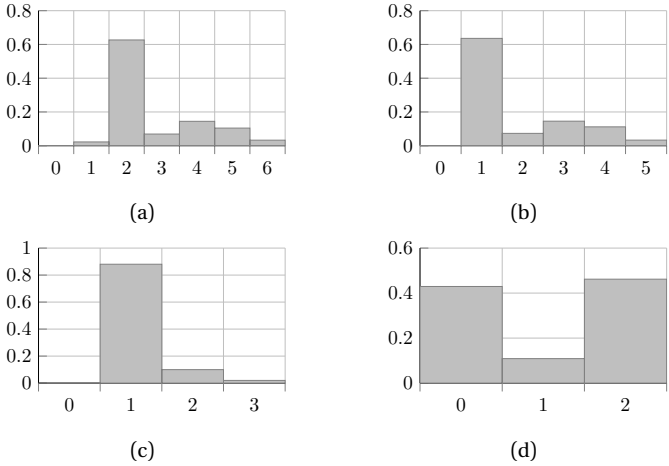


Figure 6.6: Internal utilization of elements in the 6-input FLE as well as input and output utilization. **(a)** Amount of FLE inputs used. **(b)** Amount of internal ULMs used in the FLE. **(c)** Amount of FLE outputs used. **(d)** Distribution of the used FLE outputs.

logic cluster and therefore also not connected to the global interconnect. This logic cluster architecture including internal cells was presented in figure 6.2b on page 181.

Simple LEs For internal cells, questions regarding number of inputs, the amount of basic ULMs chained and the overall topology of the LE arise in the same way they do for the non-internal cells. If internal cells do not contain FFs, they need to be part of a deeper logic function. In this case, the depth of the internal cell itself should be reduced to make sure that total depth of one internal-only cell chained with one non-internal-only cell is not too large, which would prevent mapping of logic functions. If FFs are used in internal cells, the internal cells can be used to terminate a combinational net. The output must however still be mapped to a non-internal cell and if the FF in a cell is optional and bypassed, the considerations regarding path depth still hold. For this internal cell purpose, a non-fracturable version of the FLE as shown in figure 6.7a was evaluated. Compared to simple logic cells consisting of one ULM and one FF that can be bypassed, these cells do not show large advantages in experimental benchmark statistics. The simple cell consisting of one ULM is sufficient to increase output utilization. The final

evaluated architecture therefore uses 5 such simple LE combined with 15 FLEs. As this adds up to 20 outputs in total, which matches the logic cluster output count, all cells outputs are exposed and internal-only cells are not used.

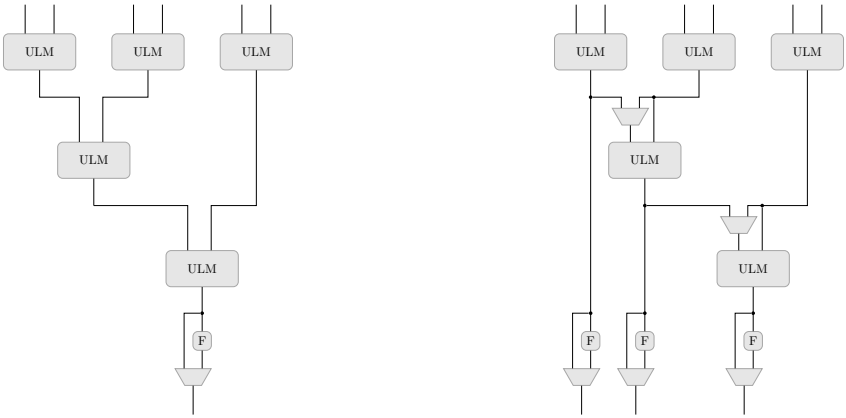


Figure 6.7: Modified ULM architectures for further evaluation. **(a)** A non-fracturable, reduced version of the LE. **(b)** A version introducing bypass paths.

Cell Functions Apart from the functions of table 6.1 on page 178 which are directly provided by the ambipolar base cell, EDA tools used require certain other, artificial modes: The ABC tool for synthesis expects a buffer cell, an inverter cell and constant zero and one generator cells. The buffer cell is unnecessary (or implicit) in an FPGA architecture and can simply be removed from the netlist. The inverter cell can be represented in the *CNT-DR8F* cell, when both inputs are connected to the same input variable and a configuration such as `ulm_nor2` is chosen. Connecting inputs in this way is possible for those ULM cells which are directly connected to the input crossbar, i.e. simple LE and the first level of cells in the FLEs. For other ULMs, an inverter function can not be realized without additional hardware. Alternatively, better EDA algorithms could reduce use of the inverter function, as it can be largely absorbed into complex modes with inverted inputs. For constant inputs, LUT based FPGA architectures can simply adjust the lookup tables to adjust for the constant inputs. An ULM based architecture on the other hand's side has to provide these constant values as possible inputs to the logic cells. The constant zero and one generator cells have therefore been

implemented as two additional inputs to the logic cluster input crossbar, as shown in figure 6.2b on page 181. Having local constants avoids routing of these constant nets, reducing congestion on the interconnects while still creating minimal resource usage in a logic cluster.

Cell Bypass In some cases, ULMs need a bypass mode which simply forwards one of the inputs to the output. This function equivalent to the buffer function which is required by synthesis tools, but the buffer function is removed from the netlist before packing. The main difference is the reason such a function is required: Whereas the buffer mode is required because of limitations of EDA tools not adapted completely to an ULM workflow, the bypass mode is used to increase flexibility of the fracturable logic cells: To realize simple two-input functions in the proposed FLE, some ULMs need to be bypassed. In the same way as the inverter function, the bypass function can be implemented with no overhead if both inputs of the bypassed ULM can be connected to the same input. This is again the case for those ULMs which directly connected to the input crossbar, as these can be configured in `ulm_and2` mode to forward the input. For other ULMs, additional multiplexers are required to either bypass the cell, or connect both inputs to the same value. Connecting the inputs has the benefit of the bypassed ULM still serving as an electrical buffer and has been implemented in figure 6.7b. This approach has the drawback of requiring additional transistors for the multiplexers, as well as an additional bit of configuration storage.

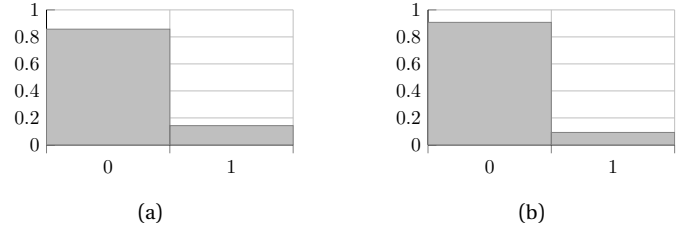


Figure 6.8: Utilization statistics for the FF in the logic cluster. **(a)** Utilization of FFs in FLE cells. **(b)** Utilization of FFs in simple cells.

FF Amount When there are more FLEs in the RFET logic cluster than in the reference architecture LUT cluster, it can be questioned whether each FLE needs to contain one FF. Gathering the FF usage statistics from the VPR benchmarks shown in figure 6.8 suggests that only 12.2 % of the FFs in FLEs and 9.2 % of the FFs in simple logic elements are used. Adjustments to the

architecture and further benchmark statistics suggest that a total of 10 FFs per logic cluster yields almost complete utilization of all FFs. At the same time, the total FPGA area did not increase in these benchmarks. This effect would occur if there are too few FF in a logic cluster and more clusters need to be instantiated. A reasonable trade-off therefore seems to be to reduce the amount of FFs in the cluster to 10: The final tested architecture therefore uses 5 simple logic elements with FFs, 5 FLEs with FFs and 10 FLEs without FFs.

This page intentionally left blank

Chapter 7

Power Management Regions

The following chapter provides details for the implementation of the region concept presented in section 4.2 on page 114 for VPR. The implementation presented enables different parts of the FPGA architecture to operate under different PVTA conditions and in different performance levels. An example of a supported region grid is shown in figure 7.1, reprinted from figure 4.4.

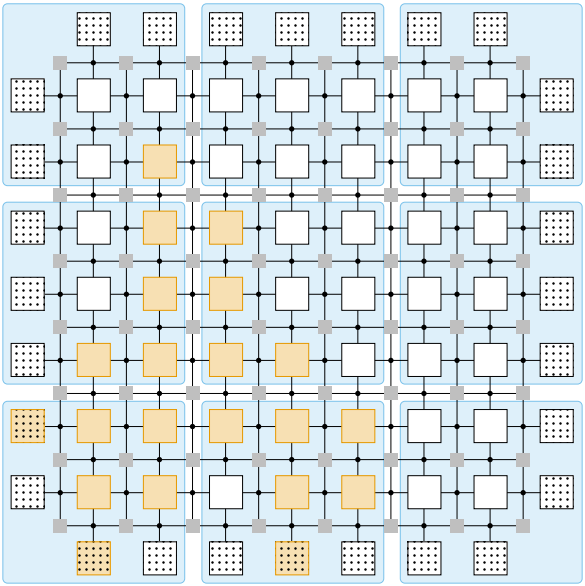


Figure 7.1: An example of the *k6_frac_N10_40nm* architecture with a placed user application (orange) and region grid (blue). Region size was arbitrarily chosen as 3x3 blocks, including IOBs.

Apart from the introduction of the region feature, region assignment will be discussed. Region assignment determines how FPGA resources are grouped into regions and which operating conditions are used for each region. Two variations of region assignment will be presented: Static region assignment maps the operating conditions to each region statically, according to the FPGA architecture description. This mapping is therefore fixed for all user applications and all FPGA devices of this architecture. Dynamic region assignment on the other hand is used for architectures, where the operating conditions of a region can change in the field, during application use. The final PARFAIT PVTA system will use only dynamic region assignment, but static assignment can provide a less complex alternative which needs fewer resources to realize.

The region description format was kept abstract, to enable modelling of not only DVS systems, but also other varying parameters. Parts of this chapter were originally published in [Pfa23b] and this chapter has been extended with further details. In addition, the region model support has been integrated and evaluated in a more thorough design space exploration, which will be presented in the final chapters of this dissertation.

7.1 Region Modelling in VPR

To support locally varying operating conditions, two new concepts are introduced: Power regions and region modes. A power region is a physical area on an FPGA containing one or multiple primitive blocks, usually CLBs. Such a region is the smallest entity for which operating conditions can be changed individually.

```
<region sizex="2" sizey="2" default="vdd_1V">
  <mode name="vdd_1V"/>
  <mode name="vdd_0.95V"/>
  <layout>
    <fill type="vdd_1V" priority="10"/>
    <pattern type="vdd_0.95V" priority="20" ... />
  </layout>
</region>
```

Listing 7.1: Example showing a region specification with two modes, `vdd_1V` and `vdd_0.95V`, with static region assignment.

What operating conditions can be changed within a region is irrelevant for the modeling scheme: It could be *VDD*, body biasing, PG voltage or any other quantity. A region mode is a specific instance of these operating conditions. For example, if *VDD* is varied, operating modes could be two different voltages, 1.0 V and 0.95 V, as shown in listing 7.1. Regions are usually placed as rectangles in grid form, but the introduced extensions for the architecture description allow any shape supported by VPR. In the simplest case, a VPR architecture description is extended with region width and height information and the grid will be created automatically. In listing 7.1, a `pattern` description enables static assignment of region modes.

LUT and therefore CLB delays are the most critical components to consider in mode adjustments, e.g. when reducing operating voltage [251]. Primitive blocks, switches and other elements in these blocks will exhibit different propagation delays and power usage, depending on the operating conditions. For the region models in this thesis, clocking and routing networks are assumed to have an independent power supply and are not included in the regions. The VPR architecture description format was accordingly extended to support different delay values depending on region mode. An example for such a mode-based timing specification is shown in listing 7.2.

```
1 <delay_constant max="85e-12" in_port="l.o" out_port="b.o">
2   <region mode="vdd_1V" max="85e-12"/>
3   <region mode="vdd_0.95V" max="84.27e-12"/>
4 </delay_constant>
```

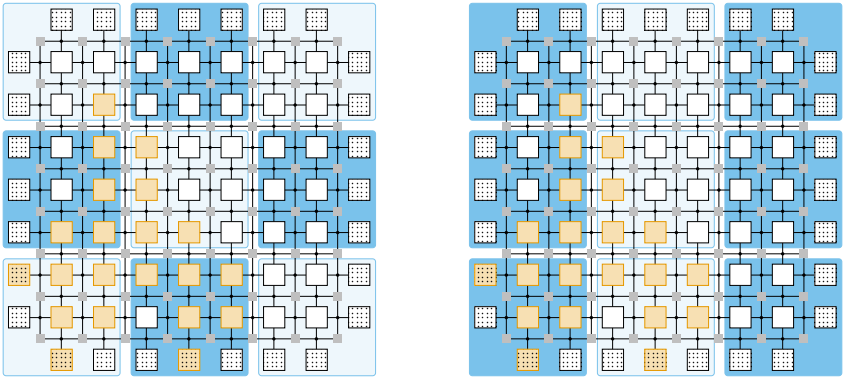
Listing 7.2: Example showing the description of timing specifications which vary depending on the region mode.

In addition, two changes were made in the VPR power estimator: To enable parametrization on modes, absolute values given in the architecture file such as in `absolute` or `pin-toggle` power models, are changed to tables. For more complex transistor level power modeling, e.g. as part of the `auto-size` model, VPR obtains per transistor data from a `.tech` file. This functionality was also extended to support loading different `.tech` files depending on each region's current mode.

For logic block delay calculation, the VPR algorithms were changed to use the delay from the architecture file according to the region mode. In the placement phase, it is ensured that the cost function also considers the now region-specific, adjusted delays. The power estimation was also changed to estimate each region's power usage independently, based on its current mode.

7.2 Static Mode Assignment

In static region assignment, modes are pre-assigned to power regions in the FPGA architecture description. VPR will not change modes of the regions, and will simply calculate the initial placement according to this grid instead. Figure 7.2 shows two simple examples of such static assignments, a checkerboard and a column pattern.



(a) (b)

Figure 7.2: Two example grids specified using static assignment. Two different modes are indicated by the shade of blue. **(a)** Checkerboard pattern. **(b)** Column pattern.

The main expected benefit of static assignment is reduced resource overhead in implementation: As each region operates in only one predetermined mode, the implementation only has to provide one operating condition, e.g. one supply voltage, for each region. Compared to a dynamic scheme, this however trades off resource overhead with reduced flexibility: Modes can not be adjusted according to the placed application. Instead, the placement logic will be modified to place the application accordingly to the predefined grid. Another major limitation that follows from this limitation is that a system with static assignment can not adjust modes at runtime. It can therefore not implement the PVTA compensation scheme.

Figure 7.3 shows modifications in the VPR toolflow to support static region assignment. Firstly, a new region layout stage is inserted in the VPR flow after packing, building the grid according to the architecture file. This step has to be performed after packing, as the FPGA size is not available before. This

limitation is grounded in VPR's support for application-specific FPGA size determination, which is performed in the packing phase. Secondly, the newly built region grid (*R-Grid*) is made available for the placement phase. In this phase, VPR uses the new information to place logic in low-power or high performance regions. This is accomplished primarily through consideration of timing delays according to the respective modes.

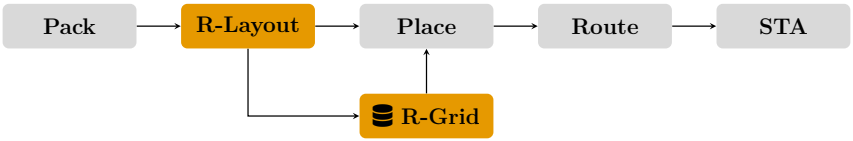


Figure 7.3: VPR extensions to support mapping applications to regions with statically assigned modes. The *R-Layout* pass and the *R-Grid* model shown in orange have been newly added to the original VPR flow.

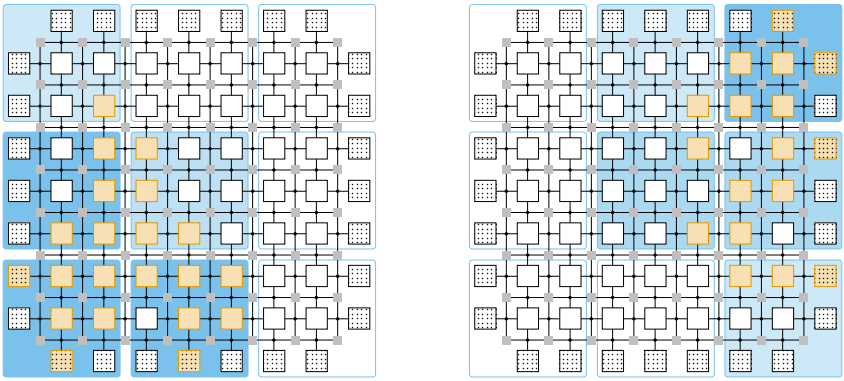
As an implementation peculiarity, VPR caches certain block-internal delay values, assuming those are independent of the exact location of the block. As this is not true if regions exhibit different performance characteristics, the implementation was changed to update cached delays whenever a logic block moves between regions.

7.3 Dynamic Mode Assignment

As static assignment can not be used for PVTA compensation, a dynamic mode assignment scheme was implemented as well. In this approach, modes are assigned to regions after the design has been placed as usual by the standard VPR flow. For the placement phase, there are two options to consider: All regions may be considered to be in low power mode when the design is placed. After placement, the voltages of regions containing paths failing the timing constraints are increased, until the paths meet the timing constraints. Alternatively, the design may be placed assuming all regions are in high-performance mode. After the initial placement, the voltage for each region is reduced until the design fails to achieve timing closure. Experiments show that the latter approach yields better results.

Figure 7.4 shows examples of dynamically assigned modes for the grid introduced in figure 7.2. Two different applications were placed to the FPGA, as indicated by the orange blocks. Regions which do not contain any logic used by the application can be switched off completely, as can be seen in

figure 7.4a in the top-right and in figure 7.4b in the bottom left. The modes selected by VPR map to the initial or expected conditions: Based on the propagation delays for the default process, VPR calculates the available slack. If PVTA conditions are considered, the real available slack in each region may change. Usually, the delays used during placement are the worst case expected delays: This ensures that the placed application works on every FPGA IC.



(a) (b)

Figure 7.4: Example mode assignments for two applications using dynamic assignment. Different modes are indicated by the shade of blue. **(a)** Application 1. **(b)** Application 2.

Figure 7.5 shows the implementation of dynamic region mode assignment in VPR. The implementation of the algorithm largely follows the variant proposed by [237] for dual power supply FPGAs: Two new processing phases are introduced in figure 7.5: Slack budget calculation and dynamic region assignment. In slack budget calculation, all paths going through a region are enumerated and their timing slacks are collected. The minimum slack is then assigned as the slack budget of that region. After determining the budgets of all regions, budgets are sorted and the dynamic region allocation phase is executed: VPR iterates through all regions, starting with those with the highest slack budget. It assigns a lower voltage to the region and recalculates the overall timing. If timing closure is still achieved, the change is accepted and processing continues with the next region. Otherwise, the change is reverted before continuing with the next region. When all regions are processed, the algorithm concludes with a final timing analysis.

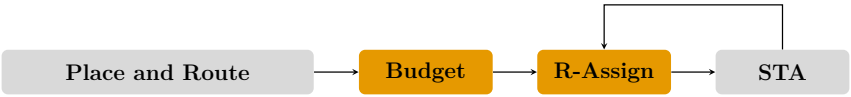


Figure 7.5: VPR extensions to support mapping applications to regions with dynamically assigned modes. The *Budget* slack estimation pass and the *R-Assign* mode assignment pass shown in orange have been newly added to the original VPR flow.

The approach introduced here uses information about the delays in modes to select modes during placement. The next chapter will discuss an alternative approach, where the slacks in each region will simply be recorded. Together with the critical path length in each region, this information will be used to derive a k_{slack} factor: This factor represents a relative measurement stating how much relative increase in path delay can be accepted, before timing violations occur.

This page intentionally left blank

Chapter 8

PVT- and Aging Compensation

With RFET-based LEs in place and power region support in EDA tools introduced, the final steps for power management are determining the acceptable performance degradation in regions and measuring a region's current performance. These two aspects are first covered in this chapter. The chapter then concludes with a summary describing the PARFAIT power controller and FPGA architecture.

8.1 Performance Requirement Determination

The PVTA compensation approach employed in the PARFAIT FPGA does not have direct feedback of whether an application achieves timing closure: Unlike the Razor schemes introduced in section 3.4 on page 85, the scheme used here does not detect timing violations in the configured application. Instead of that, the measurement approach introduced in the next section determines the current propagation delays for all LEs. The impact on user application paths is however not directly known, as the PVTA controller does not know how LEs are connected by paths in the application.

Nevertheless, the controller approach needs a target variable for each region. This dissertation therefore introduces the derivation of a slack factor for each region and each application. This factor is calculated in the EDA flow in each region, based on the critical path and its available slack. As the EDA timing analysis uses the nominal or typical LE delay, the factor describes how much LE delay can differ from nominal delay to still ensure timing closure. Referring to section 2.3 on page 20, t_{slack} is the difference between required and actual arrival time. This means that the path delay, as characterized by VPR when placing the user application, could increase by up to t_{slack} for the circuit to still work correctly:

$$t_{\text{PD,max}} = t_{\text{PD,typ}} + t_{\text{slack}} \tag{8.1}$$

The slack factor, k_{slack}, is then defined as:

$$t_{\text{PD,max}} = k_{\text{slack}} * t_{\text{PD,typ}} \tag{8.2}$$

$$k_{\text{slack}} = \frac{t_{\text{PD,max}}}{t_{\text{PD,typ}}} \tag{8.3}$$

$$= \frac{t_{\text{PD,typ}} + t_{\text{slack}}}{t_{\text{PD,typ}}} \tag{8.4}$$

Equation (8.4) can be easily obtained from the VPR tool with some minor code changes. Defining the k_{slack} factor provides a relative definition independent of the absolute slack and propagation delay, which both depend on the path length. When assuming that a path delay is a linear combination of LE delays and that all LEs are affected by a change in delay in the same way, the k_{slack} factor can be compared to relative change in LE performance: If the delay of each LE in a path increases by less than k_{slack}, the total path delay will not exceed $t_{\text{PD,max}}$.

When a path passes through multiple regions, the path element delays could theoretically have different relative delay increases. To properly handle this case, the VPR code calculating k_{slack} considers all paths passing through each region and selects the smallest factor. A path spanning multiple regions might therefore determine the factor in most of those regions. If the factor is different for one region, as per the definition, it must be smaller. So even in this case, $t_{\text{PD,max}}$ still won't be exceeded. Such a case however provides room for future optimization: If a part of a path is known to never be operating at its slowest possible performance, as other paths in that region require higher performance, the delay in other regions might be safely increased further than the averaged k_{slack} factor suggests.

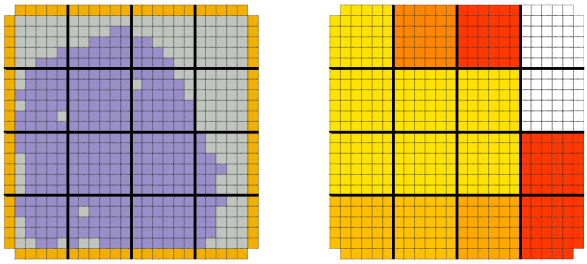
Listing 8.1 shows pseudocode implementing the k_{slack} calculation in VPR. In general, the calculation code builds on the region definition code of section 7.1 to load regions from the architecture definition. VPR then places and routes the user application or loads a pre-defined placement and routing. The pseudocode then uses VPR's `TimingPathCollector` to iterate over all paths and determines whether the path uses any CLB in a specific region. Ultimately, it selects the path with the smallest slack in a region to calculate the region's factor.

```
1 auto paths = path_collector.get_all_paths();
2
3 for (auto path: paths) {
4   auto slack = get_slack(path);
```

```
5     auto delay = get_delay(path);
6
7     for (auto elem: get_elements(path)) {
8       auto loc = get_loc(elem);
9       auto region = get_region(loc);
10
11      if (result[region].slack > slack) {
12        result[region].slack = slack;
13        result[region].k = (delay + slack) / delay;
14      }
15    }
16  }
```

Listing 8.1: Pseudocode implementing the newly introduced k_{slack} calculation in VPR.

Figure 8.1b shows a graphical demonstration of obtained k_{slack} for a benchmark application. Here, the `diffeq1` benchmark has been placed to a 24x24 FPGA, with 4x4 power regions. Figure 8.1a shows the user application placement, Figure 8.1b the k_{slack} factors. In this example, the critical path spans 7 regions, which therefore all have the same slack factor of 1.17. The remaining regions have varying factors from 1.39 to 4.73.



(a) (b)

Figure 8.1: Example application placement and available slack in each region. **(a)** shows how the benchmark application has been placed onto the FPGA and into regions. **(b)** depicts the slack factor for each region. Brighter color indicates a lower factor, i.e. less available slack in the critical path. Unused regions are depicted in white color.

8.2 Transparent Logic Invasion

As explained in section 4.1, a special logic invasion scheme has been designed as part of this thesis to characterize the performance of CLBs. In this scheme, the user application is transparently relocated, with the application output being unaffected. This relocation process is specifically designed to always keep a single row of the FPGA unused. Over time, each single row of the FPGA becomes temporarily unused, allowing characterization of all CLBs with few changes in FPGA architecture and little hardware overhead. The performance measurement system is described in detail in the next section. This section describes the invasion process itself, which realizes the logic relocation and the programming of arbitrary bitstreams onto unused CLBs.

Concept The overall process of logic invasion is depicted in figure 8.2 by means of a small example. Subfigure **(a)** shows the initial state after the user application has been programmed, i.e. the `DONE` state has been reached in figure 4.12 on page 126. In this example application, a bitstream has been programmed to the four upper CLBs, the lowest row being unused. Numbers in the CLBs denote the individual CLB bitstreams programmed onto those, whereas blocks in blue show unused CLBs. The global interconnect, including PSMs and CBs, is not affected by the logic invasion scheme. After the user application has been programmed initially, this configuration is never modified. It will therefore not be mentioned explicitly in the following discussion and figures.

Subfigure **(b)** shows the FPGA after the first logic invasion: The middle row has been invaded and is now effectively unused. It could be programmed with any arbitrary bitstream at this point without affecting the user application. To achieve this, the CLB configuration and bitstream of this row has been duplicated to the previously unused row below. In the following, this process will be called relocation of logic from the invaded row to the backup row. As mentioned before, the interconnect is not modified, so inputs for the relocated CLBs, which are now in the lower row, are still provided by the CBs in the invaded row. The architecture has been modified accordingly to allow redirection of the inputs of one CLB to the CLB directly below. Similarly, outputs can be redirected from the relocated-to row back to the invaded row. Active extra connections are shown in orange in the figures and make up the main changes and resource overhead of the proposed logic invasion scheme.

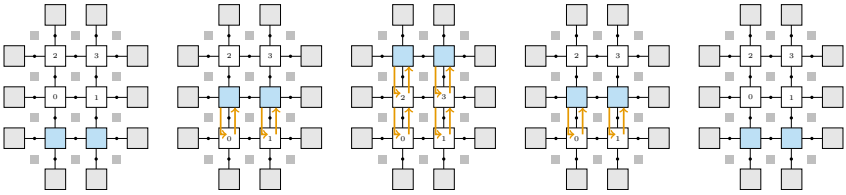


Figure 8.2: Overview of the logic invasion process, numbers depict specific application CLB bitstreams. **(a)** Initial state. **(b)** Row 1 invaded. **(c)** Row 2 invaded, final up state. **(d)** Row 1 invaded. **(e)** Row 0 invaded, final down state.

This strategy using additional connections has been chosen, as the alternative, dynamic reprogramming of the global interconnect, essentially requires expensive re-routing of the user application: It is not possible to simply move PSM configurations downwards. Even in this simple architecture there may be signals which still need to connect to specific IOBs in the invaded row. PSMs in the invaded row could therefore not simply be reprogrammed to forward between the upper and lower row. As an example, consider the leftmost CLB in the middle row would connect to the left IOB in the same row. If the CLB moves down, it still needs to connect to the IOB one row above. This connection requires introducing another track, which might not be available. For such a simple architecture, an elaborate solution could be implemented in the EDA toolflow: If the EDA tools ensure that in these cases, CBs always use a wire in upwards direction, this wire could be forwarded to the PSM above. Such an approach however may be difficult to scale to more complex architectures. It is mentioned here though as a possible optimization for future research, which could further reduce hardware overhead in logic invasion.

Subfigure **(c)** shows the final invasion in upward direction. In this case, the top-most row has been invaded, where the general concept is the same as for all other rows. The invaded, or unused, row has now fully moved from the bottom to the top of the FPGA, covering each single CLB once. As there is now no unprogrammed row in the bottom that can be used for relocation, the invasion process can not simply start again from the beginning. Instead, the invasion process is now rolled back step-by step until the original state is reached again.

Subfigure **(d)** shows the first invasion in downwards direction. In this case,

logic of CLB 2 and 3 is moved back to its original location in the top-most row. In the following, this process will be called relocation of logic from the invaded row back to the original row. As the logic moved back to the original location, no further connections are necessary. Previously active bypass connections shown in orange will just have to be deactivated again. Subfigure **(e)** shows the final invasion in the downwards direction. At this point, the bottom-most row is invaded and the FPGA state is identical to the state in the beginning in subfigure **(a)**. The invasion process can now start again from the beginning, allowing for unlimited, continuous operation.

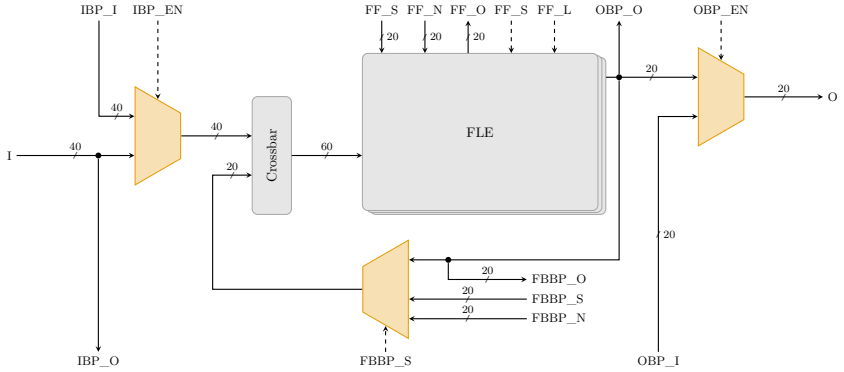


Figure 8.3: Modifications of the CLB of figure 4.2 on page 110 to enable logic invasion. Newly added components are marked in orange.

Architecture Changes As was explained in the overview, changes to the architecture are necessary to support the relocation process. All changes are contained in the CLB implementation and are shown in figure 8.3, which is a modified version of figure 4.2 on page 110. For the logic invasion implementation, the BLE is treated largely as a black box, enabling any logic generator to be used as the BLE. The concept here is therefore not limited to LUTs. The only needed change in BLEs is using the FF implementations with an asynchronous load option.

The main changes in figure 8.3 are the introduction of new multiplexers marked in orange and the corresponding in- and output signals. The multiplexer on the left side is added to support redirection of the inputs. It uses the `IBP_I` signal to gain access to the global interconnect inputs of the CLB in the row above. At the same time, it also forwards its input to the CLB in the row below using the `IBP_O` output. Similarly, the output multiplexer can

connect the global interconnect output to the output of the BLEs in the row below, the `OBP_I` signal. It also forwards its own BLE outputs to the CLB in the row above using the `OBP_O` signal.

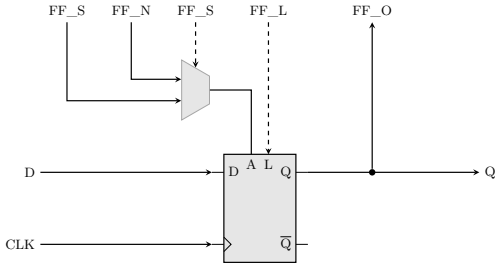


Figure 8.4: Modification of the FFs in the BLEs for logic invasion. A direct output provides the FF output independently of the MUX state in the BLE. In addition, the FF supports asynchronous loading from two selectable inputs.

Further modifications concern internal signals: During reconfiguration, internal feedback loops may not be initialized: Some BLE might recursively depend on internal feedback of some other BLE, rendering its output initially undefined. As the BLEs are not connected to the global interconnect yet, this is not immediately an issue, but ultimately, the output needs to be defined for relocation to be successful. As an example, think of a BLE acting as a simple latch, passing one input directly to the output. If the latch output is fed back to the input, this results in a stable `0` or `1` at the output. Which one it is after programming of the BLE is however undefined. The new architecture therefore introduces the feedback multiplexer and the `FBBP` signals: These allow to temporarily drive the feedback inputs for the crossbar from either the CLB above, or the CLB below. Note that in this case, connections to both above and below CLBs are required. For the global interconnect CB inputs and outputs, this was not necessary: In the downwards move, connections are just restored to the original connections. But for feedback, the relocated-to row has to get feedback from the active row, which is the row below in the downward relocation case.

Similar observations apply to the FFs in the BLEs. When relocating logic, the state of FFs needs to be preserved. FFs in the relocated-to row therefore need to load their values from the relocated-from row. Again, for the same reasons as for the feedback signals, the CLB must be able to load these from both the CLBs in the rows above and below. In figure 8.3, FF loading is realized by the `FF` signals. The modified FF architecture itself is shown in figure 8.4:

It consists of a multiplexer to select the load input from one of two options, north and south, and the FF itself. The FF has an additional asynchronous load input, `L`. As long as this input is high, the FF asynchronously loads the input `A` as its stored value. In this mode, the output `Q` therefore follows the input `A` transparently.

Toolflow Modifications As depicted in figure 8.2 on page 207, the introduced invasion scheme requires that the bottom row is not used in the initial state. This means that the application bitstream must not use the CLBs in this row. Other interconnect resources and IOBs can be used as usual. To ensure that VPR does not place any logic in these CLBs, they were defined as special blocks in the FPGA architecture. Marking them simply as empty has important side effects: VPR changes the pin connection pattern of other CLBs next to these empty blocks. This complicates bitstream generation and in order should be avoided. The blocks have therefore not been defined as empty, but as special black box blocks. These blocks have the same number of global interconnect inputs and outputs as CLBs, and therefore the same connection pattern. They however do not contain any BLEs, LUTs or any other logic blocks, so VPR does not attempt to place application logic in these locations.

In addition, these changes in the architecture may increase the worst-case propagation delays through CLBs, as additional signal routing is introduced. The EDA tools have to use these new worst-case delay values for CLBs in timing analysis.

Invasion Steps Figure 8.5 shows the new state machine introduced in the `ProgController` block. It hooks into the state machine in figure 4.12 and replaces the `DONE` state: Instead of stopping operations after the initial programming, the controller jumps to the new `WAIT` state in the logic invasion logic.

1. The initial `WAIT` state is used to pause logic invasion. It waits for the `invasion_en` signal and only continues to the next state if this signal is asserted.

2. In the `INIT` state, the FSM sets the row select signal for the row where logic will be relocated to. It also disables the clock of that row. All operations carried out during logic invasion operate on one specific row, so control signals are used in a special way: There is one `row_select` signal, selecting the row which is being operated on. All other control signals are provided in parallel to all rows. They are gated accordingly

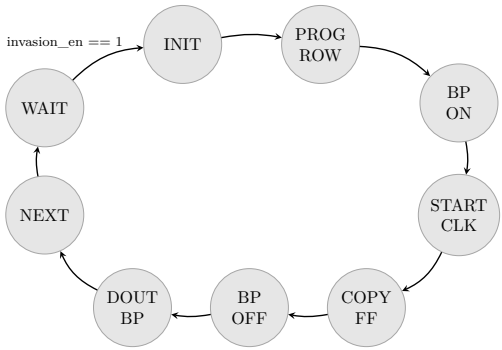


Figure 8.5: FSM which summarizes the steps in the invasion process. After invasion finishes in the `DOUT BP` state, the CLBs in the original row are unused and can be reprogrammed with an arbitrary bitstream.

to ensure only the selected row is affected. Operations always affect all CLBs in the selected row in the same way.

3. In `PROG ROW`, the FSM configures the row programmer of figure 4.11 to program the relocated-to row with the bitstream of the relocated-from row. In practice, the bitstream is not read from the relocated-from row, as the configuration system only supports writing the configuration. The FSM computes the index of the row in the original bitstream instead, depending on the reconfiguration direction. It then configures the row programmer to use that bitstream, activates the programmer and waits for the row to be programmed. Once programming is finished, it transitions to the next state.

4. Depending on reconfiguration direction, the `BP ON` state activates or deactivates the CLB input bypass `IBP_EN`: When moving up, logic is moved down and the bypass is enabled in the relocated-to row. Otherwise, the bypass in the relocated-to row is deactivated to fetch the inputs from the interconnect, as in the initial configuration. Furthermore, this state selects the `FFBP_N` input in upward relocation, `FFBP_S` in downward relocation. All these actions are performed in the same cycle, and the FSM immediately proceeds to the next state.

5. The `START CLK` state simply enables the clock for the newly configured, relocated-to row. The feedback bypass is still kept active and the FF have not been initialized yet.

6. In `COPY FF`, the FSM enables the load signals for the FFs. When relocating upwards, it selects `FF_N`, otherwise `FF_S`.
7. In the next state, `BP OFF`, the feedback bypass signals `FFBP_S` of `FFBP_N` and the FF load signal `FF_L` are deasserted. These signals are deasserted at exactly the same time, ensuring that the FFs values and the feedback signal are synchronized. After this step, the relocated-to CLBs are now fully using their internal feedbacks. They get interconnect inputs matching the relocated-from CLBs, and compute the same outputs.
8. The `DOUT BP` state configures the `OBP_EN` switch of the original row: It's either enabled, when relocating up, or disabled again, when relocating down. After this step, the relocated-from CLBs have been disconnected entirely from the global IC, and can be reconfigured arbitrarily.
9. The final state, `NEXT`, prepares invasion of the next row. It adjusts the internal row counter accordingly and changes direction if necessary. It then proceeds to the `WAIT` state again, which allows pausing the invasion process using the `invasion_en` signal.

Figure 8.6 shows two CLBs of two adjacent rows after an upwards relocation. Here, the input multiplexer in the lower row was configured to select the signal from above. The output multiplexer of the upper row selects the output signal from the CLB below. It can be seen that the lower CLB could still pass down its original input to the next CLB below. It could also still connect its interconnect output to the CLB below, even it is itself acting as a relocated-to row for the one above. This feature is required to ensure that all rows can be relocated consecutively.

The arrows marked in orange depict the complete signal path for a relocated CLB. The blue path denotes that the internal feedback is also completely contained within the relocated-to CLB. It does not depend on the relocated-from CLB in any way.

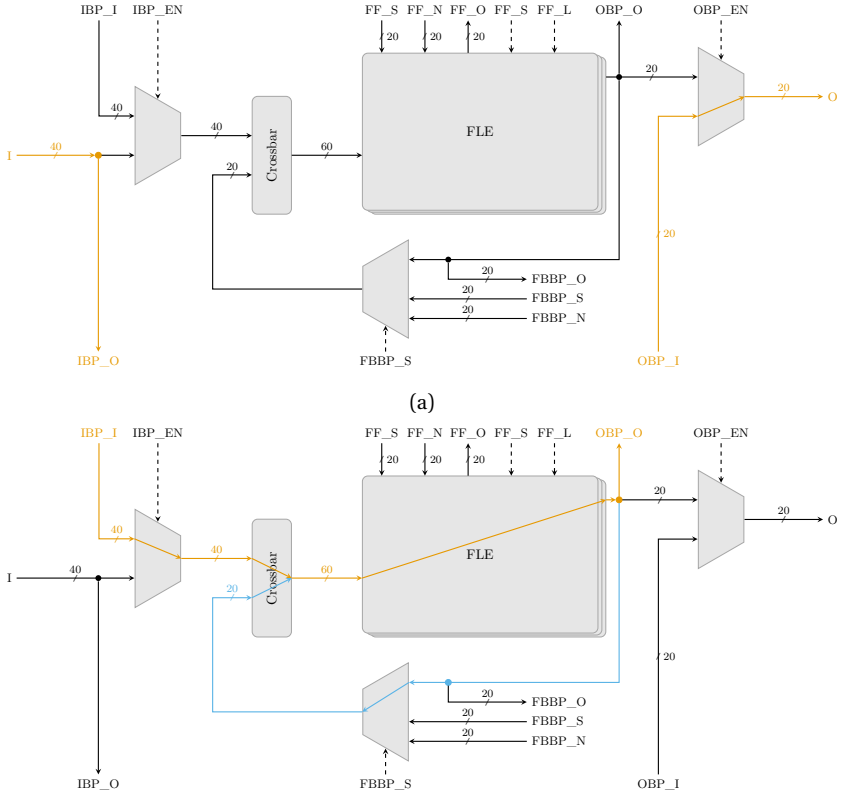


Figure 8.6: Example showing how the modifications in figure 8.3 are used to relocate a CLB. Used bypass connections are shown in orange. When the relocation is complete, the relocated-to CLB uses internal feedback, shown in blue. The original CLB is unused. **(a)** CLB in upper row, relocated from. **(b)** CLB in lower row, relocated to.

8.3 Chip Performance Characterization

With the changes introduced in the previous chapter, it is possible to program arbitrary bitstreams to invaded CLBs, without affecting the user application. In this section, this feature will be used to continuously perform performance characterizations of all CLBs on the FPGA. Here, the performance is measured as the propagation delay t_{PD} of the LEs in each CLB. As explained in section 4.3 on page 117, information about the performance of each specific CLB, which changes depending on location and time, will then be used to compensate PVTA effects.

Overview The primary goal of the performance measurement system in the PARFAIT FPGA is to achieve a fine-grain characterization of all CLBs with few additional hardware resources. This requirement and the specific architecture of FPGAs compared to ASICs makes some characterization approaches introduced in section 3.4 on page 85 less viable: Critical path extraction and replica are less useful in FPGA. First, replica paths require additional hardware resources. In addition, critical paths in FPGAs are not known during chip manufacturing. Only when the user application is considered as well, a critical path can be determined as explained in section 8.1. A replica would therefore have to be reconfigurable as well. Spending additional resources for reconfigurable replica which are not used for application logic, however seems wasteful.

The alternative approach, directly measuring performance of the individual building blocks in a path, is more viable for FPGAs: Paths are made up of LEs, which to a large part determine the delay of a path. In addition, the routing delay caused by both parasitic effects and interconnect multiplexers can be significant in FPGAs. As a first estimate, it will be assumed that performance increase or decrease in routing multiplexers follows the relative changes in speed of nearby logic elements. This work will therefore focus on the delay of logic elements, determining the delay of individual LEs using ring oscillators and counters: A ring oscillator forms an inverting, back-coupled delay path through certain LEs. As a result, when measuring a signal connecting any two LEs in the path, an oscillating rectangle wave can be obtained. The frequency of this rectangle wave is determined by the propagation delay, as will be explained in detail soon. Determining the propagation delay therefore becomes a problem equivalent to determining the oscillation frequency. As previously shown in state-of-the-art works, this can easily be achieved when counting the number of oscillations in a certain, predetermined time frame.

Whereas previous works have often used such measurements in a coarse-grain way and often measured temperature or voltage, this work will directly measure propagation delay. As all the PVTA effects affect this propagation delay, isolating one effect requires compensation of the other effects, and can therefore be difficult. For the closed loop PVTA compensation system designed here, this is not necessary: A change in propagation delay can be compensated independently of the original cause of the change. This measurement approach is therefore expected to yield better results than when used to obtain temperature or voltage measurements.

In order to achieve more fine-grain characterization than state-of-the-art works, each single CLB will be characterized. To avoid heavy resource overhead, configurable FPGA resources will be reused as much as possible through the previously introduced invasion technique: Ring oscillators and counters will be realized as soft logic using the BLEs of the FPGA architecture. Apart from saving resources, this provides one other, major benefit: Characterization is actually performed on those LEs, which also implement the user application circuit. Compared to path replica, the proposed approach therefore provides a better estimate of propagation delay in LEs. The following paragraphs describe this characterization system in more detail.

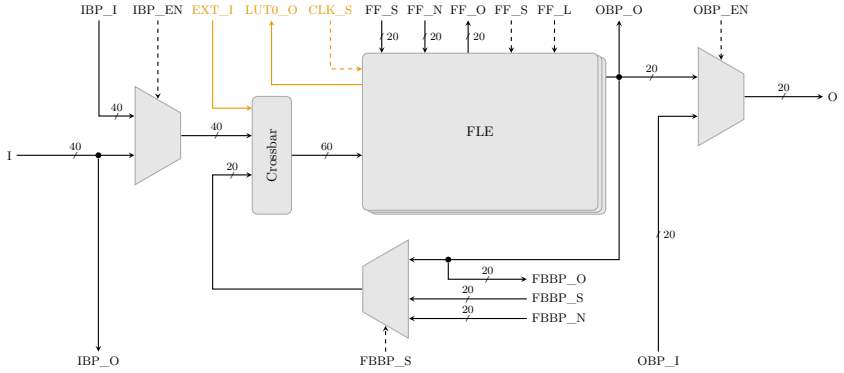


Figure 8.7: Modifications for the CLB of figure 8.3 on page 208 to enable measurement of propagation delay in the CLB. Newly added components and connections are marked in orange.

Architecture Changes Even though the measurement approach presented here largely reuses existing hardware in soft logic, three further modifications

to the CLBs are required. Figure 8.7 shows the CLB for logic invasion from figure 8.3 on page 208 with newly introduced signals marked in orange. The `EXT_I` signal is an additional input provided to the crossbar. It can be passed to any of the crossbar outputs, which in turn allows passing an external value to any of the BLEs and LEs. Changes to the bitstream structure are not required: The crossbar's 6 bit multiplexer select signals already allow selection from 64 inputs, leaving 4 inputs unused. The FASM assembler for the architecture has been extended to support this external input option for each BLE, generating the matching bitstream. VPR based logic synthesis does not use this input and therefore does not have to be changed. All soft-logic bitstreams for characterization will be manually designed in FASM instead of Verilog for full control of placement and routing. Supporting this signal in the assembler is therefore sufficient.

The second change is the introduction of the 1 bit `LUT0_O` signal. It passes the output of LE 0 in BLE 0 to the `ProgController`, which orchestrates the performance characterization. Such a single output allows for transmission of arbitrary data when using a serial protocol. It is deliberately connected to the LE output directly instead of the BLE output: As shown in figure 4.3b on page 112, this essentially means the signal always carries the LE output signal, regardless of how the output multiplexer is configured. This primarily allows configuration of this multiplexer to forward the FF state of the respective BLE, while still making the LE output accessible externally. As will be explained later, this is not strictly required, but it simplifies reading of register values via this output.

The last change is the introduction of a clock switch signal, `CLK_S`. Counting the ring oscillator pulses will be realized using a synchronous counter: The pulse signal to be counted will be provided as clock signal to all counter FFs synchronously. Compared to asynchronous counters, this requires only a small change in the clocking network: Instead of using an external clock signal, all FFs will be configured to switch to an internally provided clock signal. In the implementation, a single clock multiplexer in the CLB allows selection of the `LUT_0` signal as clock for all FFs. As this connection provides access to the signal output of LE 0, it can be used as a tap into the ring oscillator. In commercial FPGA architectures, clock switching architectures are commonly used to provide CLB with different clocks. The introduction of another clock switch is therefore not an intrusive change.

Measurement Steps Based on the discussed architecture changes, measurements can be implemented in soft logic. Like invasion, the measurement process is implemented in `ProgController`. An overview of all states for

measurement is given in figure 8.8. As an additional change, a new MEASURE state is inserted in figure 8.5 between the DOUT BP and NEXT states. In this state, the relocation of a row has just been finished and CLBs in the relocated-from row are ready to be configured with arbitrary bitstreams. The measurement states introduced here are sub-states of the MEASURE state in the invasion FSM. In the following, a quick overview of the measurement FSM states will be given, before the most important aspects will be discussed in detail.

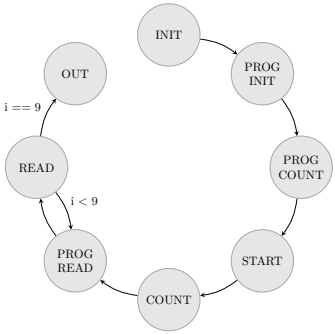


Figure 8.8: FSM realizing the measurement of propagation delay in the invaded CLBs.

1. The INIT state selects the relocated-from row for invasion and measurement.

2. PROG INIT programs the invaded row with a bitstream that initializes all FFs which are used in the counter. It also disables the external clock input for the row and the configuration reset in the row programmer: Usually, the row programmer resets FF contents before programming, to guarantee a defined reset state. The measurement process however requires that FF values are kept between reconfigurations, so the programmer is configured not to issue such reset requests.

3. The PROG COUNT state programs the bitstream that realizes the ring oscillator and the counter in the CLBs.

4. The START state starts the measurement. For this, it initializes the Prog-Controller time counter to 0 and sets the EXT_I signal to 1, enabling the ring oscillator. It also asserts the CLK_S signal to ensure that the FFs in the CLBs are clocked from the LE 0 output, i.e. the ring oscillator.

5. The COUNT state counts up the reference counter using the ProgController reference clock. When the count reaches a predefined value, it deasserts the CLK_S signal to stop clocking of the CLB FFs. This essentially switches back to the external clock, which is still disabled for the row. It ensures the FFs keep their values, which represent the final counter state. The EXT_I signal is furthermore set to 0 to disable the soft-logic oscillator, saving energy.
6. PROG READ programs the bitstream that outputs the registers using the LUT0_O signal.
7. In the READ state, the ProgController reads the values from all CLBs' LUT0_O outputs in the measured row. This essentially reads one register, i.e. one bit of the counter state, for each CLB. The process needs to be repeated for all 9 registers with a slightly modified bitstream, to select the registers properly. ProgController uses an internal counter for this and returns to the PROG READ state until all FFs have been read out. After reading the last FF, the FSM proceeds to the OUT state.
8. In OUT, the ProgController simply provides the measurement values of all CLBs in the row to an external controller. It uses a simple bus consisting of VALID, CLB_X, CLB_Y and VALUE signals. VALID is asserted for one cycle for each CLB. If it is asserted, CLB_X and CLB_Y provide the position of the CLB. VALUE provides the counter value describing the propagation delay.

Invasion Bitstreams Some measurement steps mentioned require programming of the measured CLBs with special bitstreams. In the following, these special bitstreams will be demonstrated for a single CLB. All CLBs in an invaded row are configured with the same bitstream and perform the same operation. Therefore, the bitstream for only a single CLB is stored in memory, and the ProgController duplicates this bitstream for all CLBs in the row.

The first special bitstream initializes the counter registers in the PROG INIT state. Although register initialization could be handled using explicit architecture modifications, this is not necessary. The required functionality can be realized without any additional logic using the existing CLB structure. Figure 8.9 shows the CLB configuration used. All BLEs are configured in split mode, providing two LEs and FFs. The multiplexers selecting between FF or LE output are not shown in the figure. Similarly, unused FFs are not shown and connections between FFs and LEs which pass through the crossbar, are shown as direct connections. Elements which had to be added to the CLB

only for measurement and which are used in this configuration are shown in orange. The blue box shows one example of how the depicted LEs and FFs map to the BLEs: The upper row shows the LE 0 elements of the CLB and no FFs, as the FF 0 elements are not used. The second row shows the LE 1 and FF 1 elements. One LE in the top row and the LE and register below therefore belong to one BLE.

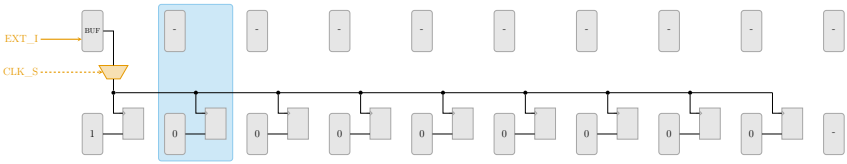


Figure 8.9: Initialization of registers used to realize the counter in a CLB. Required newly added components and connections are marked in orange. The blue box shows which LEs and FFs are contained in a single BLE (figure 4.3b).

Initializing the registers consists of setting the first register to logic value 1, all others to 0. This is required due to a peculiarity of the chosen counter: The Linear Feedback Shift Register (LFSR) connects feedback of registers through an `XOR` operation to the first shift register input. However, if the initial state is all 0, the shift register is stuck in this state. To initialize the LFSR, the LEs connected to the registers are configured to output a constant value, 0 or 1. All FFs then need to be clocked once, to ensure that the input is stored in the FF. As the measurement logic has no control of the user-provided application clock (it might be unpredictably slow or gated off), the CLB is configured to use the LE 0 output as a clock. This LE 0 is configured to pass through its only input, acting as a transparent buffer. The crossbar then connects this input to the newly added `EXT_I` input, which is driven by the `ProgController`. It can therefore directly drive the clocks and issue a clock edge as needed by driving this input pin. An excerpt of the FASM used to realize this configuration is shown in listing 8.2, the full FASM is available in the appendix, see listing E.1 on page 359.

```
# Pass through FLE0 LUT0 input 0 from external input
FLE[0]
 .CB.I[0]="ext[0].I"
 .5BLE[0].FF.ENABLE
 .5BLE[0].LUT5.INIT[31:0]=32'b10101010101010101010101010101010

 # Configure BLE0 LUT 1 as constant 1
 .5BLE[1].FF.ENABLE
 .5BLE[1].LUT5.INIT[31:0]=32'b11111111111111111111111111111111
```

```
10
11 # All other LUTs as constant 0
12 FLE[1]
13  .5BLE[0].FF.ENABLE
14  .5BLE[0].LUT5.INIT[31:0]=32'b00000000000000000000000000000000
15  .5BLE[1].FF.ENABLE
16  .5BLE[1].LUT5.INIT[31:0]=32'b00000000000000000000000000000000
```

Listing 8.2: FASM excerpt representing the register initialization for measurement. A full version can be found in listing E.1 on page 359.

The configuration for the second bitstream, used during measurement, is shown in figure 8.10, with the same conventions as in the previous figure. This bitstream realizes a ring oscillator using 11 LEs, 10 in the top row and the rightmost one in the bottom row. It also realizes a LFSR based counter using the first 9 FFs in the second row. The measurement system uses the newly added `CLK_S` signal to drive the FFs clocks using the LE 0 output. Placing the logic appropriately therefore enables the oscillation signal to be used as a clock for the counter. This setup ensures that the path used for propagation delay measurement passes through all 10 BLE elements and the additional eleventh LE is used to further reduce the oscillation frequency. When using a classical oscillator consisting of only inverters, the frequency directly represents the propagation delay of one gate. This is however not practical here, as the oscillation frequency is too high to clock the FFs in the counter: As the feedback is also realized using one LE, it will have a propagation delay of half the clock period in this case. To obtain higher margins for timing closure and to reduce the total number of clock periods counted, this setup instead uses a single inverter paired with 10 buffers. The frequency can then be obtained as in the following equation:

$$t_{\mathrm{PD,OSC}} = 11 * t_{\mathrm{PD,LE}} \tag{8.5}$$

$$T_{\mathrm{OSC}} = 2 * t_{\mathrm{PD,OSC}} \tag{8.6}$$

$$f_{\mathrm{OSC}} = \frac{1}{T_{\mathrm{OSC}}} \tag{8.7}$$

Instead of using a simple inverter in the first gate, it is replaced by a *NAND* gate combining the feedback path and the `EXT_IN` input. This input can then be driven low to stop oscillation, or high to enable the oscillator. Driving the output low for some time also ensures a defined state after programming: As the initial LE output values are unknown after programming, it is not guaranteed that all outputs initially have the same value. The measured oscillation signal could therefore be unpredictable. When driving `EXT_IN` low

until the LE outputs have propagated through the whole chain, all outputs will be predictably 0.

The LFSR used has the characteristic polynomial $x^9 + x^5 + 1$. This LFSR is of maximum length, with a period of $2^n - 1$ [252]. It therefore counts $C_{max} = 511$ distinct values, before it starts repeating values. The main benefit of this counter is, that it requires little logic, only a single *XOR* gate. The drawback of LFSR counters is, that decoding the LFSR state into the count number is non-trivial. As the amount of distinct states is small in this specific counter, decoding can however be efficiently implemented using a precomputed lookup table.

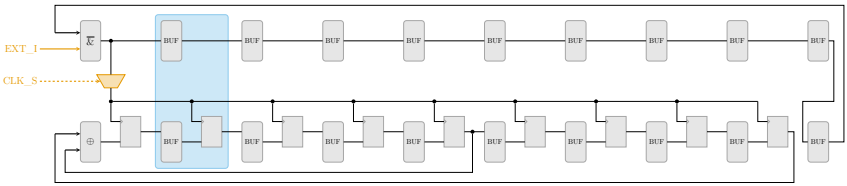


Figure 8.10: Ring oscillator and counter realized in a single CLB for propagation delay characterization. Required newly added components and connections are marked in orange.

Considering that the LE delay of the VPR reference architecture is 235 ps, 11 LEs cause an oscillation period of $T_{OSC} = 5.17$ ns. When using a 100 MHz reference clock in the `ProgController` and counting for 100 cycles, the measure time T_{meas} is 1000 ns, resulting in a count value of 194 for the nominal frequency. It is also possible to calculate the minimum and maximum LE delay before the counter will overflow or underflow:

$$T_{OSC,min} = \frac{T_{meas}}{C_{max}} \tag{8.8}$$

$$T_{OSC,max} = \frac{T_{meas}}{1} \tag{8.9}$$

$$t_{PD,LE,min} = \frac{T_{OSC,min}}{2 * 11} \tag{8.10}$$

$$t_{PD,LE,max} = \frac{T_{OSC,max}}{2 * 11} \tag{8.11}$$

For the values chosen here, this leads to a minimum measurable LE propagation delay of 89 ps. The circuit can therefore be up to 2.6 times faster than the nominal case, before the measurement circuit counter rolls over. Similarly, the calculated maximum propagation delay of 45 ns means the circuit can

slow down by factor 190. It should be noted, that the relative resolution of values becomes lower for lower count values:

$$\frac{t_{\mathrm{PD,LE,1}}}{t_{\mathrm{PD,LE,2}}} = \frac{T_{\mathrm{OSC,1}}}{T_{\mathrm{OSC,2}}} \tag{8.12}$$

$$= \frac{C_2}{C_1} \tag{8.13}$$

$$= \frac{C_1 + 1}{C_1} \tag{8.14}$$

For smaller counts, the factor is up to 100 %, whereas for counts close to the maximum, the factor is 0.2 %. At the nominal count of 194, the factor is 0.5 %, showing the minimal change relative to the nominal propagation delay that can be measured. Further optimization of these limits can be achieved through variation of the measure time: A longer measure time will yield higher counts and resolution, whereas a shorter measure time enables measuring faster signals without overflow. In addition, when the signal is known to be in a certain range, roll-over events of the counter can be taken into account to enable even longer measure times. Knowing the typical propagation delay of the LE, a relative increase k_{clb} in the delay can be calculated from the count value. This relative increase can then directly be compared to the slack factor k_{slack} determined in equation (8.4). As long as the measured increase factor is less than k_{slack}, user application paths are guaranteed to meet timing closure.

Listing 8.3 shows an excerpt of the FASM used to implement the counter and oscillator. The full FASM is available in the appendix, see listing E.2 on page 361.

```
# Invert fle[9].O[1] if ext[0].I is 1
FLE[0]
 .CB.I[0]="fle[9].O[1]"
 .CB.I[3]="ext[0].I"
 .5BLE[0].FF.BYPASS
 .5BLE[0].LUT5.INIT[31:0]=32'b01010101000000000101010100000000
 # x9 + x5 + 1. Use I1 and I2 for an XOR
 .CB.I[1]="fle[8].O[1]"
 .CB.I[2]="fle[4].O[1]"
 .5BLE[1].FF.ENABLE
 .5BLE[1].LUT5.INIT[31:0]=32'b00111100001111000011110000111100

# LUTs as pass-through, oscillator ones without registers
FLE[1]
 .CB.I[0]="fle[0].O[0]"
```

```
16   .CB.I[1]="fle[0].O[1]"
17   .5BLE[0].FF.BYPASS
18   .5BLE[0].LUT5.INIT[31:0]=32'b10101010101010101010101010101010
19   .5BLE[1].FF.ENABLE
20   .5BLE[1].LUT5.INIT[31:0]=32'b11001100110011001100110011001100
```

Listing 8.3: FASM excerpt representing the counter and oscillator for measurement. A full version can be found in listing E.2 on page 361.

The third special bitstream is used in the `PROG READ` state to receive the count value stored in the CLB registers. For this, all clocks applied to the registers are stopped after counting, so the values are stable. The readout then utilizes only the existing interconnect and the single, newly added `LUT0_O` output to read all 9 bit. Figure 8.11 shows the general idea for this: LE 0 is programmed in buffer mode, simply passing its input to its output. This output is connected to `LUT0_O` and can be read externally. The `ProgController` now modifies this original bitstream to obtain nine slightly different versions: The crossbar configuration is modified to pass other signals to the LE 0 input. This way, there is one modified bitstream for each of the nine FFs, routing its value to this LE and ultimately to the output. Programming all nine modified bitstreams one after the other, the `ProgController` can read all bits.

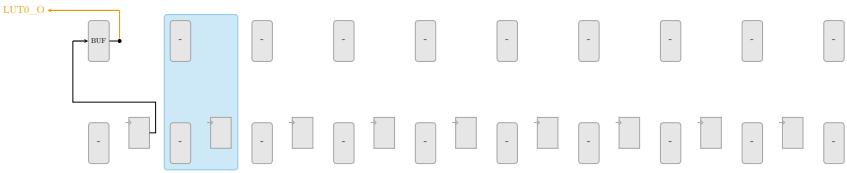


Figure 8.11: Configuration of the CLB used to output counter register 0 to the `ProgController`. Required newly added components and connections are marked in orange. To read other registers, the CLB crossbar is reprogrammed to connect these registers' outputs to the LUT 0 input.

Listing 8.4 shows an excerpt of the FASM used to implement the readout logic. The full FASM is available in the appendix, see listing E.3 on page 363.

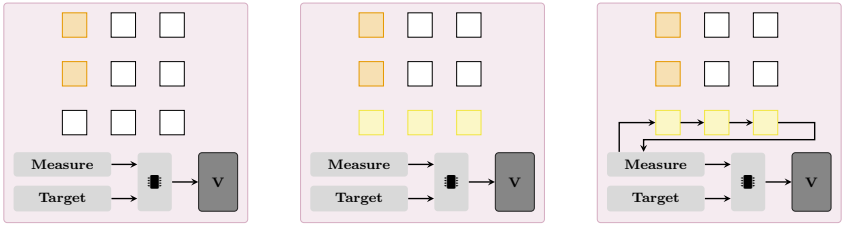
```
1  FLE[0]
2   # This selects the register routed to the first LUT
3   .CB.I[0]="fle[0].O[0]"
4
5   # LUT0 just forwards its input 0
```

```
6   .5BLE[0].LUT5.INIT[31:0]=32'b10101010101010101010101010101010
7   .5BLE[0].FF.ENABLE
8   .5BLE[1].FF.ENABLE
9
10  # Enable all FFs
11  FLE[1]
12   .5BLE[0].FF.ENABLE
13   .5BLE[1].FF.ENABLE
```

Listing 8.4: FASM excerpt representing the FF readout configuration for measurement. A full version can be found in listing E.3 on page 363.

8.4 Power Management Controller

With the performance requirement k_{slack} for a region, and the current measured performance k_{clb} of each CLB, the power management controller can now be derived in this chapter. Referring back to the concept in figure 4.6 on page 118, the region controller is reprinted here in figure 8.12.



(a) (b) (c)

Figure 8.12: Region detail and region controller concept. The figures show a zoomed in detail of the central region in figure 4.4. **(a)** Region controller in idle mode. **(b)** Row was reconfigured dynamically with measurement circuits. **(c)** Delay characterization is active.

The k_{clb} factor is measured for each CLB, but the k_{slack} factor is only determined once for a whole power region. This factor could easily be calculated for each CLB, but the performance and voltage adjustment techniques used operate on whole regions. In addition, as the k_{slack} values need to be stored as part of the bitstream, a single factor per region reduces the configuration

storage requirements. The configuration for the k_{slack} factors themselves is inserted in the interconnect configuration chain. As these factors are constant and do not need to change during logic invasion, they can be stored as part of the static configuration.

Unlike shown in the simplified concept in figure 8.12, the k_{clb} measurement is not performed locally by the region controller. It is rather orchestrated globally for the whole FPGA in the `ProgController's` logic invasion of section 8.2. The measured values therefore have to be passed from this central controller to the individual regions' controllers. Figure 8.13 shows this final FPGA power compensation scheme, with the global and the per-region controllers added, including control connections.

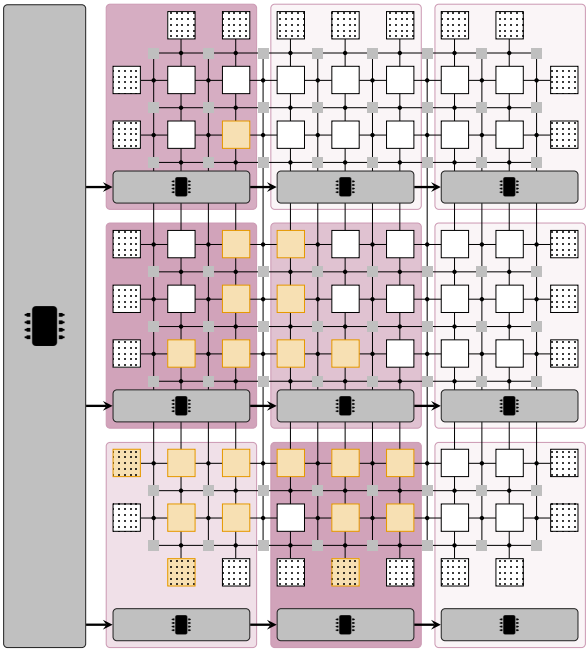


Figure 8.13: The PARFAIT FPGA architecture with the shade of red in each region representing locally adjusted performance using PVTA compensation. Also shown is how region controllers connect to the `ProgController` which orchestrates logic invasion and measurement.

All region controllers connect to the same measurement output port of the central controller. They then assess the `X` and `Y` signals of that port to deter-

mine whether a measured value belongs to the current region. The controllers detect when the factor for the last CLB in the region's last row has been received and then perform one update cycle. As controllers only consider the slowest CLB, they only keep the largest k_{clb} factor for the region. It then calculates a single new control value using a control algorithm and adjusts the voltage in the region. The next adjustment will then be performed when a new measurement is available.

Because of the slow invasion process, the controller operates in closed-loop configuration with a slow update rate. As the PVTA changes observed are generally slow effects, this is not an issue. For the initial characterization, which compensates primarily the process variation, this might seem counterintuitive: However, during the EDA flow, the application timing analysis was performed for nominal delay. Therefore, there shouldn't be any timing violations, as long as the CLBs do not have higher delays than the nominal values. Voltage changes in this initial case will in turn only reduce the voltage and performance, reducing energy consumption in the device. Adjustments to actually increase performance are only required once aging slows down the circuit further.

Listing 8.5 shows the VHDL code used to implement the control algorithm within the region controller. For the evaluations in this thesis, a basic proportional controller is used and investigation of more elaborate control algorithms is left for future work. In addition, the current implementation is only meant to be used during simulation and therefore was not designed to be synthesizable.

```
1  architecture sim of RegionPCTRL is
2    signal bg_voltage_buf: real := 1.0;
3    constant MAX_DELTA: real := 0.001;
4  begin
5    bg_voltage <= bg_voltage_buf;
6    impl: process(clk)
7      variable bg_new: real;
8    begin
9      if rising_edge(clk) then
10       if target_delay_factor > 10.0 then
11         bg_voltage_buf <= VAL_MIN;
12       else
13         bg_new := bg_voltage_buf - VAL_P * (
               ↪ target_delay_factor - current_delay_factor);
14         if bg_new < VAL_MIN then
15           bg_new := VAL_MIN;
16         elsif bg_new > VAL_MAX then
```

```
17            bg_new := VAL_MAX;
18          end if;
19          bg_voltage_buf <= bg_new;
20        end if;
21      end if;
22    end process;
23  end;
```

Listing 8.5: VHDL excerpt showing a simple proportional controller used to calculate the control voltage from k_{slack} and k_{clb} in each region. `MAX_DELTA`, `VAL_MIN` and `P` are `generic` parameters.

8.5 Power-Aware FPGA Architecture

This section will quickly summarize how the previously discussed, individual aspects are combined in the PARFAIT FPGA. An overview of the final architecture has been shown in figure 8.13 on page 225. Implementation of non-reconfigurable logic uses the ambipolar standard cells introduced in chapter 5. Apart from implementing standard logic, this provides an additional BG allowing for V_{th} voltage scaling.

For the LE in the PARFAIT FPGA, reconfigurable ambipolar cells as introduced in chapter 6 are used. However, for the final evaluation, the RFET technology characterized in [42] was used. This technology was developed by PARFAIT project partner NaMLab and was characterized in the ways necessary for the PARFAIT FPGA. The original characterization in [42] and the temperature characterization in [2] enabled the derivation of the RFET delay model in section 4.6 on page 144. During writing of this thesis, a novel LE based on this technology has been published in [246] as an early access publication. This allows the PARFAIT FPGA to actually make use of an ULM realized in exactly the technology used for the delay model characterization. For more consistent results, the final PARFAIT FPGA therefore uses this cell instead of the *CNT-DR8F* cell introduced in section 6.1 on page 177. As the set of supported functions is mostly overlapping for both cells, the approach presented in section 6.1 does not change. Figure 8.14 shows the LE based on the *RGATE*. Unlike the *CNT-DR8F* based LE in chapter 6, this cell has not yet been optimized to match the LUT expressiveness, as will be explained in the results chapter. For synthesis, the PARFAIT architecture only uses the combined `genlib` synthesis approach introduced in section 6.2.

Power management regions are used in the PARFAIT FPGA as introduced in chapter 7. The PARFAIT architecture exclusively uses dynamic power region assignment: As all regions are characterized repeatedly using the invasion scheme, all regions also enable threshold voltage adjustment. The benefits of higher power reduction in dynamic assignment are therefore achievable with little resource overhead, which makes it the preferred solution.

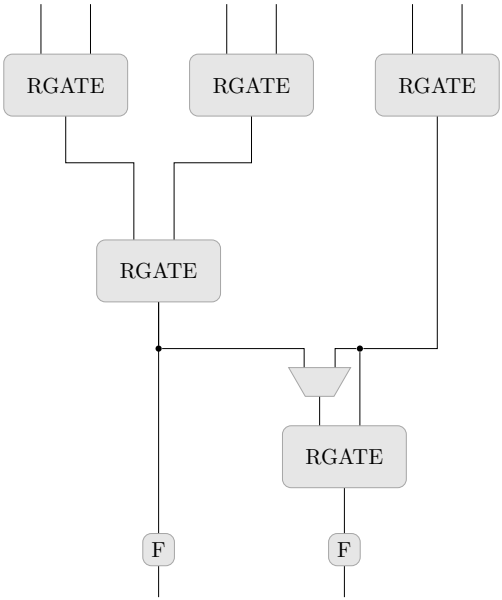


Figure 8.14: Final FLE used for the PARFAIT FPGA, using the *RGATE* as published in [246] based on the technology of [2].

For logic invasion itself, the scheme introduced in section 8.2 works identically for ambipolar cells as well as for LUTs. For measurement in section 8.3, some slight adjustments are necessary: Whereas the approach works just as before, the LUT values in the bitstream have to be replaced with configuration for the new LEs. The used measurement bitstreams demand some requirements from the LE feature set: For register initialization, the LEs must be able to output constant 0 or 1 values on both outputs. This can be achieved by routing some constant values to LE inputs in the crossbar and passing through or inverting in the LE. In addition, output 0 must be able to pass through an input to enable clocking of the registers, which is supported as well.

For the measurement bitstream, output 0 of a BLE must be able to realize a *NAND* function. The minimal LE based on *RGATE*s from [246] does not support this directly: Although the first *RGATE* supports the *NAND*, the second stage can not be configured in pass-through mode and will always invert. As a solution, output 0 of the second BLE in the oscillator chain can be configured as an inverter instead of a buffer. This is possible, as the first stage *RGATE* does support pass-through mode. The counter only requires pass through mode on the BLE 1 output, which is also supported. For FF readout, again only a pass-through for output 0 of LE 0 is needed. Therefore, both the *CNT-DR8F* ULM and the *RGATE* based ULM support all measurement operations.

This page intentionally left blank

Chapter 9

System Simulation and Evaluation Methodology

Previous chapters have introduced the PVTA compensation system, the PARFAIT architecture and the delay models. This chapter will describe how those parts are combined for the final evaluation. Two approaches will be covered: First, a hardware evaluation based on the VFPGA system will be described. This approach can only be used for prototyping of the digital aspects, so PVTA changes can not be assessed with such a system. For faster validation, a functional simulation of the PARFAIT system is introduced. To evaluate PVTA aspects, this simulation will be coupled with the technology models (section 4.6) and the scenario models (section 4.7). The introduced co-simulation system will also integrate VPR to enable power analysis.

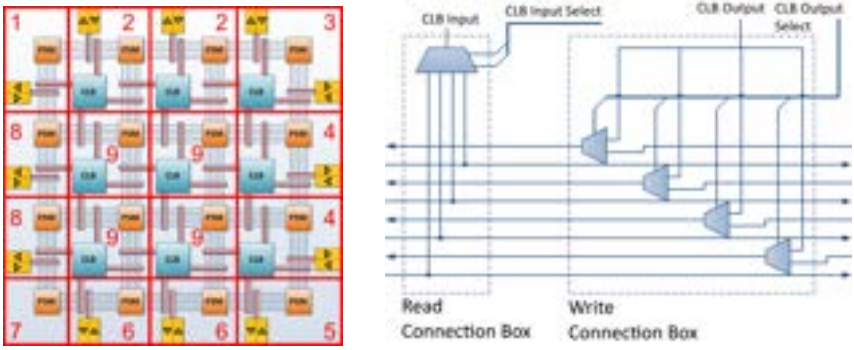
9.1 Virtual FPGA Evaluation

Evaluation of custom FPGA architectures on commercial FPGAs can be achieved using VFPGAs: VFPGAs are a small layer on-top of commercial FPGAs implementing custom FPGAs. The PARFAIT implementation presented in chapter 4 on page 107 can essentially be used as a VFPGA. When mapping to commercial FPGAs, there is however one issue: FPGAs need to ensure uniform delays for all their logic blocks. This then enables VPR architectures to describe the propagation delay characteristics for basic blocks, and to calculate the final delays. Such uniform delays can however not be guaranteed when mapping an FPGA to another FPGA. Fine-grain timing constraints can ensure that a guaranteed maximum delay is never exceeded, but this is not very practical: Ideally the whole FPGA should operate at the maximum possible performance, not at a user derived arbitrary frequency.

To solve this issue, manual placement was investigated in [Pfa21] and [Pfa22]. The primary idea here is that the custom FPGA architecture is uniform, as is the commercial FPGA it is mapped to. Using scripts, it is therefore possible to manually fix the placement, instead of leaving placement up to vendor tools. A detailed evaluation of this approach can be found in the original publication, but the following paragraphs will quickly reprint and summarize these publications.

VFPGA Architecture

Figure 9.1 shows the VFPGA architecture and the arrangement into the common types (1-9) of tiles. In its simplest configuration, the VFPGA consists of 9 different tile types, which are distinguished by orientation and contained elements. A single tile can contain all elements (like type 2 and 4), all but the IOB (type 9, i.e. the central tiles), PSM and IOB (type 1, 5, 6, 8), CLB, PSM and two IOBs (type 3), or only the PSM (type 1). Even for tiles which have the same types of elements, their differences in orientation — and therefore layout — will require them to be placed differently.



(a) (b)

Figure 9.1: VFPGA architecture details. **(a)** Tile Distribution and top-level architecture. **(b)** CBR and CBW implementation and connection to routing channels.

The Configuration Units (CUs) are not shown in figure 9.1a, as it is an implementation detail of the VFPGA. They store and provide the configuration for the CLB, PSM and IOB in their respective tile and enable

dynamic reconfiguration of the VFPGA. In the used VFPGA implementation, the CU is implemented essentially as a shift register with parallel output. It will have to be mapped to the host FPGA in addition to the other components.

Figure 9.1a also shows the wires corresponding to relevant delays for the final VPR architecture model: Apart from intra-block delays such as delays within the CLB, these consist of nets in the global routing channels. Figure 9.1b shows how the CLB in a tile connects to the global routing channels. Connection points are realized using Read Connection Boxes (CBRs) and Write Connection Boxes (CBWs). Those consist of multiplexers, which either connect multiple wires to one CLB input, or the CLB output to one of the channel wires.

To realize this connection, the CBW consists of one multiplexer for each wire in the channel. The multiplexer either forwards the signal of the channel wire or writes the CLB output to this wire. Structurally, these multiplexers will be mapped to host FPGA LUTs. Therefore, on the host FPGA, a VFPGA wire from PSM to PSM will actually consist of at least two host wire segments and the LUT. Similar effects are also caused by IOB connections.

Another peculiarity of the original VFPGA implementation concerns the implementation of its bidirectional wiring: Logically, the VFPGA architecture uses bidirectional wiring, but the implementation on the host FPGA can only make use of unidirectional wiring. Figure 9.1b shows how the CBR and CBW connect to different host wires, leading in different directions. In order to drive a logical VFPGA wire in both directions, the PSMs will loop back the right-to-left signal in left-to-right direction. Due to that, in the final PARFAIT architecture, only unidirectional wiring is used.

All these effects cause two implications for this work: First, when constraining the design, the VFPGA wire can not be constrained as one unit. Instead, all wire segments on the host FPGA have to be constrained individually. Furthermore, when modelling the VFPGA architecture in VPR, the delay for the complete net is needed. For this reason, the data extraction script extracts the individual segments. But for architecture modeling and for assessment of placement results in this publication, these have to be summed.

Uniformity Metrics

Uniformity is a measurement of local delay variation across the VFPGA structure. Placing every tile in the same way, the VFPGA is a uniform structure, which in theory could be placed uniformly on the host FPGA. As explained previously, not all tiles have exactly the same internal structure and non-uniformity of the host FPGA architecture will further degrade the uniformity. To address this, the introduced definition of uniformity divides the VFPGA into N_C sets, where each set represents one column C. Nets are grouped into classes, so that similar nets in different tiles are within a single class. The following classes have been introduced:

1. **PSM Left:** Horizonal nets, starting at PSM left output multiplexers and ending at PSM right input multiplexers.
2. **PSM Right:** Horizontal nets, starting at PSM right output multiplexers and ending at PSM left input multiplexers.
3. **PSM Top:** Vertical nets, starting at PSM top output multiplexers and ending at PSM bottom input multiplexers.
4. **PSM Bottom:** Vertical nets, starting at PSM left output multiplexers and ending at PSM right input multiplexers.
5. **PSM Internal:** Internal nets within the PSM, realizing the Wilton switch pattern.
6. **CLB Input:** Nets starting at the output of the CBR and ending at the input of the LUT.
7. **CLB Output:** Nets starting at the LUT output and ending at the input of the CBW.

The definition of uniformity then essentially measures differences between rows within a set. This definition is then formalized in the following equations:

$$\mu_{c,n} = \frac{1}{N_R} \sum_{r=1}^{N_R} t_{c,r,n} \tag{9.1}$$

$$\sigma_{c,n}^2 = \frac{1}{N_R} \sum_{r=1}^{N_R} \left(t_{c,r,n} - \mu_{c,n}\right)^2 \tag{9.2}$$

$$\overline{\sigma} = \frac{1}{N_C N_N} \sum_{c=1}^{N_C} \sum_{n=1}^{N_N} \sqrt{\sigma_{c,n}^2} \tag{9.3}$$

$$\overline{c_v} = \frac{1}{N_C N_N} \sum_{c=1}^{N_C} \sum_{n=1}^{N_N} \frac{\sqrt{\sigma_{c,n}^2}}{\mu_{c,n}} \tag{9.4}$$

Here, $t_{c,r,n}$ is the delay of a net in class n, column c and row r. Equation (9.1) provides the arithmetic mean $\mu_{c,n}$ of the delays, calculated over the N_R VFPGA rows. $\sigma_{c,n}^2$ then calculates the variance for a net class in a certain column over the rows. This is further used in $\overline{\sigma}$ to calculate the arithmetic mean of the standard deviations of all net classes in all columns. $\overline{c_v}$ provides the arithmetic mean over the coefficient of variation of all net classes in all columns. Whereas the standard deviation is an absolute value and therefore depends on the mean of the delays, the coefficient of variation provides a relative measurement. As the delays in the host FPGA are largely discrete (e.g. fixed delays in LUTs), it is expected that relative delays can not be reduced further at some point. Because of this, $\overline{\sigma}$ is used to guide the design of the placement strategies and for evaluation of practically achievable uniformity. $\overline{c_v}$ is used to judge the quality of results for VFPGA: As a smaller delay τ allows to put more logic elements in a path at the same frequency for VFPGA applications, a constant standard deviation leads to reduced certainty of the number of VFPGA logic elements in the path. A constant relative value $\overline{c_v}$ signifies unchanged conditions for the VFPGA application synthesis.

Building Blocks

After a VFPGA design has been synthesized, the custom placement strategies are applied. The strategies will be described in detail in the next section, but all of them are based on the following constraints:

Timing Constraints As *Vivado* analyzes every possible path in the design, it will also consider configurations of PSM multiplexers that can create combinational loops. It is therefore not easily possible to constrain the timing of the design by simple definition of the final clock period, as *Vivado* will break the loops at arbitrary points. This generates long paths through different numbers of CLBs and PSMs, making it further impossible to constrain a path just between two specific PSMs. To solve this problem, these paths are broken manually.

Two variants of constraints are used: In the variant with fine-grain constraints, all individual atomic nets have their delay constrained using the `set_max_delay` timing exception, ensuring that the design still meets timing constraints and forcing the timing driven optimization to operate. These constraints

will lead to path segmentation, which is the desired outcome. In addition, it will add false path constraints on the original long paths automatically. Path segmentation can affect logic placement and timing results, so special care needs to be taken when examining the *Vivado* timing reports. Because of this, custom scripts are used to evaluate the delays of relevant nets. As was shown in [Pfa22], these fine-grain constraints are necessary to force *Vivado* to optimize the routing for the manually placed design. The drawback of this approach is limited scalability, as large designs which introduce many of these constraints cause excessive memory usage and runtime in the *Vivado* toolflow. Due to this, placement strategies without the fine-grain constraints were evaluated as well.

Placement Constraints Placement constraints are used to perform floorplanning through definition of pin placement and absolute, or relative, placement of cells. It guides and controls where the place-and-route tools may put FPGA design elements. *Vivado* supports various placement constraints, ranging from just constraining a group of logic in a certain area to exact placement of single cells to a certain logic element. The following placement constraints were used in this work:

1. **LUTNM and HLUTNM:** Used to place two combinational functions into the same LUT.
2. **PROHIBIT:** When the only requirement is to avoid placing any logic at a specific site, this is achieved using this constraint.
3. **LOC and BEL:** To place a logical element in a specific location, the *place_cell* command is used. This command translates into LOC and BEL constraints, where LOC links the element from the netlist to a slice and BEL places it to a specific LUT or flip-flop within the slice.
4. **PBlock:** A PBlock is a collection of cells in one or more rectangular regions that specify the device resources contained by the block. It is more restrictive than no placement constraints, but less constraining than LOC and BEL.

Placement Strategies

The following paragraphs introduce the manual placement strategies in detail. The uniformity metric was used to guide development of the strategies. Critical path delay and LUT overhead will not be evaluated here, but were assessed in [Pfa22].

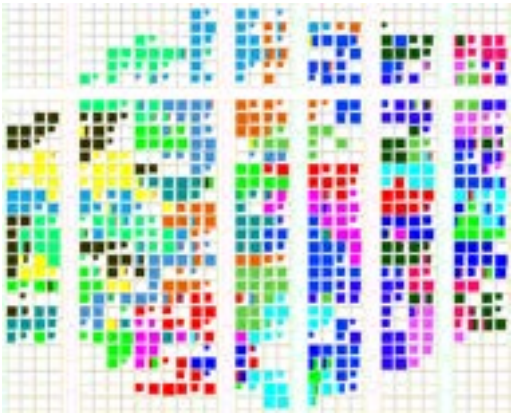


Figure 9.2: VFPGA placed using standard *Vivado* placement. Host-FPGA CLBs belonging to same VFPGA tile are shown in the same color. As can be seen, some tiles are compact, whereas some are scattered across wider area. It can also be seen that *Vivado* packs tightly and does not keep empty sites to preserve overall structure.

Standard Vivado Placement Figure 9.2 shows placement result of the default *Vivado* strategy (*Vivado* Synthesis Defaults, *Vivado* Implementation Defaults, *Vivado* 2019.1.1). The figure illustrates the arguments given previously in the motivation of this work: Automated placement does not make explicit use of the structural regularity of the VFPGA, which results in tiles being implemented in slightly different ways. Some are more distributed, others more localized, leading to varying net delays and reducing uniformity of the VFPGA architecture. This effect is even more apparent in larger designs, where placement algorithms have to deal with an overall larger amount of nets and cells.

Basic PBlock Strategy In the basic PBlock strategy, each tile is contained in a single Partition Block (PBlock): After a block with suitable size is created in quadratic or rectangular form, the `add_cells_to_pblock` TCL command is used to add all cells of a tile to the block. When the tile size has been determined, the host FPGA location and target area are fixed. Finally, all VFPGA cells are fixed to the PBlocks belonging to their tile using the PBlock constraints, completing the basic PBlock placement.

Nested PBlock Strategy In addition to the PBlocks used in the first strategy, this strategy introduces up to two additional PBlocks within each tile. Logic

belonging to the VFPGA CLBs and IOBs is mapped to these nested PBlocks accordingly: When defining the PBlocks, all assigned logic cells are forced into the blocks, but this does not prevent placing any additional unassigned cells into them. Based on this idea, two more variants are introduced in addition to the rectangular vs. quadratic layout distinction: In the partially nested strategy, the outer PBlock is used for the tile and nested blocks are used for IOB and CLB, but the PSM is only constrained by the outer PBlock. This gives *Vivado* the freedom to place the PSM in the remaining outer PBlock area, or place part of it inside the nested PBlocks. In the fully nested strategy, *Vivado* is forced to not place any PSM logic in the nested PBlocks, prohibiting usage of remaining logic cells in them. Figure 9.3 demonstrates the concept for a 5x5 CLB VFPGA.

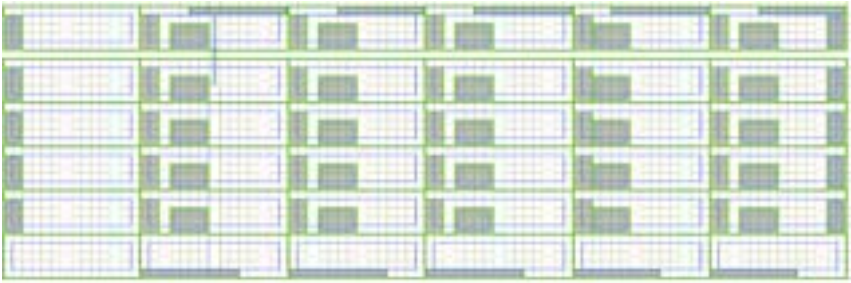


Figure 9.3: 5x5 CLB VFPGA floorplan with nested PBlocks. The nested CLB PBlock is divided into two pieces to ensure the minimum possible area is used. The top right corner tile has an extra nested PBlock for its second IOB unit. No internal PBlock was used at all in the bottom left corner tile, as it only contains a PSM.

The placement script is extended with the following steps to create the nested PBlocks:

1. The internal PBlocks can consist of multiple rectangles. The CLB PBlock is placed in the bottom left corner with height at most equal to the height of the tile minus one. This guarantees some freedom to IOB PBlock and to ensures distribution of the PSM unit over the tile PBlock.
2. The IOB PBlock is placed within the tile PBlock. The side is determined according to the tile type.

Fine-Grain Manual Placement Strategy This strategy further constrains logic, directly mapping the relevant LUTs and flip-flops to specific LUTs or

flip-flops in the 7 series host CLB. As there are numerous ways to place the logic within a tile, a manually derived layout is chosen instead of trying to find a fully automated one. The strategy is then made generic to support different VFPGA parameters, but the layout is fixed to the VFPGA and therefore cannot be reused for completely different applications. Evaluation of different manual layouts led to a placement as was presented in figure 9.1a: The PSM is located in the upper right corner and the CLB is placed in the lower part of the tile. Figure 9.4 shows the device view in *Vivado* after the manual placement strategy has been applied.

Figure 9.4: VFPGA tile (type 9) placed using the manual placement strategy. Multiplexers of the PSM's top, right, bottom and left side are marked red (1), purple (2), yellow (3) and blue (4). The CLB is located at the bottom with the LUT, two internal multiplexers and D-flip-flop colored in black (5). Sky blue color represents the configuration units of the tile (6). Yellow blocks at the bottom (7) depict Write Connection Boxes, whereas Read Connection Boxes are marked green and turquoise (8) and make up remaining logic distributed around the LUT.

The implementation of this strategy operates on two lists for each tile PBlock, an instruction list and a list of the free host FPGA LUTs. The instruction list contains simple VFPGA logic element place instructions, interleaved with sorting instructions. It is processed element by element, either placing logic elements or resorting the list of free resources. When an element placement instruction is processed, the logic elements are mapped sequentially to the elements in the sorted list of free resources, starting at a specified offset. When a resorting instruction is found, the resorting algorithm sorts the list of remaining available host LUTs. It sorts horizontally or vertically and uses ascending or descending sorting order, depending on the instruction. As an example, the `sort_xy_dd` instruction sorts first based on the x location, and if the x value is the same for some CLBs, it uses y as secondary criteria. Descending sorting is applied in both cases. This specific instruction is used to sort the list

of available logic elements before placing the right and left multiplexers of the PSM, as they need to be placed vertically from the top right corner. Sorting is always done on the list of free resources, so the length of this list decreases as the placement process proceeds. This makes it possible to reach every single CLB in the PBlock, not just the ones at the borders.

9.2 Static Power Analysis

Section 2.3 on page 20 introduced the sources for power consumption in digital circuits. Power consumption was split into two categories, dynamic power and static power. Where dynamic power is largely determined by switching power, static power is largely determined by leakage effects. Switching power was given in equation (2.9) as:

$$P_{\text{switching}} = \alpha C_{\text{L}} V_{\text{DD}}^2 f \tag{9.5}$$

As can be seen, it is dependent on the switching frequency, which is application specific.

Power analysis for custom FPGA architectures for both dynamic and static power can be performed in VPR. For the dynamic power estimation, VPR can estimate the switching activity in a placed application. To realize power estimation, it uses `.tech` files which represent power for a specific technology. VPR ships such files as examples for some technologies and provides a workflow to derive files for other technology from SPICE models. The VPR power estimator has been extended as explained in chapter 7 to support various operating modes and regions. However, obtaining the `.tech` models for RFET technology is difficult: As there are no SPICE models available, those would have to be derived manually. This is difficult again, as only selected current measurements are available for RFETs, but power measurements are not available.

As an alternative, higher level power modelling in VPR does not require `.tech` files: In this model, *RC* values or absolute power for components are specified in the architecture. This model was also extended to support regions and multiple modes. Still, obtaining power values for RFETs is difficult due to the reasons mentioned before.

For the evaluation in this thesis, a different approach was chosen because of these difficulties with the VPR power estimation: The PVTA system introduced and the voltage scaling methods used primarily effect static power:

Switching power in equation (9.5) is derived from charging curves of load capacitances and therefore only depends on the voltage the gates are charged to or from. This voltage is the supply voltage VDD, so threshold voltage scaling or body biasing do not affect the switching power. For dynamic power, these approaches can affect the short circuit power, but this effect is not further discussed here.

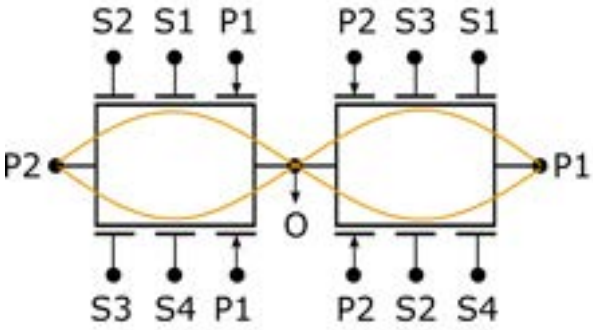


Figure 9.5: Potential current leakage paths in the *RGATE* between $P1$ and $P2$. Current direction depends on the program voltages $P1$ and $P2$. Which transistor is "off" and contributes to the total leakage, depends on $P1$ and $P2$ as well as the input variable state. Figure adapted from [2].

Static power and the subthreshold leakage on the other hand are directly influenced by threshold voltage scaling and body biasing. As figure 9.5 shows, leakage current is also a concern in the CMOS-like arrangement of transistors in the *RGATE*. With the models for the leakage current parametrized on control variables introduced in section 4.6, static power can be estimated. Whereas these models do not allow for absolute power estimation, a relative estimation can be used to evaluate how much the leakage current is reduced: First, the total leakage current in absence of voltage scaling for all CLBs in an FPGA can be described as:

$$\begin{aligned} I_{\text{tot0}} &= \sum_{i=1}^{N_{\text{CLB}}} \alpha \cdot I_{\text{Leak}}(V_{\text{C0}}) \\ &= \alpha \cdot N_{\text{CLB}} \cdot I_{\text{Leak}}(V_{\text{C0}}) \end{aligned} \tag{9.6}$$

Where α is a proportionality constant representing the number of transistors forming potential leakage paths and V_{C0} is the control parameter at the nominal value. Generalizing these formulas for arbitrary control parameters

then yields the general formulas. The current with body biasing or threshold voltage scaling in regions is then given as:

$$\begin{aligned} I_{\text{tot}} &= \sum_{i=1}^{N_{\text{Region}}} \alpha \cdot I_{\text{Leak}}(V_{\text{C,i}}) \\ &= \alpha \sum_{i=1}^{N_{\text{Region}}} I_{\text{Leak}}(V_{\text{C,i}}) \end{aligned} \tag{9.7}$$

The relative current consumption then is given as:

$$i = \frac{I_{\text{tot}}}{I_{\text{tot0}}} = \frac{\sum_{i=1}^{N_{\text{Region}}} I_{\text{Leak}}(V_{\text{C,i}})}{N_{\text{CLB}} \cdot I_{\text{Leak}}(V_{\text{C0}})} \tag{9.8}$$

Given that $P = U \cdot I$, VDD being the relevant voltage and VDD being unaffected by the control parameter, the relative power is equal to the relative current:

$$p = \frac{P_{\text{tot}}}{P_{\text{tot0}}} = \frac{V_{\text{DD}} \cdot I_{\text{tot}}}{V_{\text{DD}} \cdot I_{\text{tot0}}} = i \tag{9.9}$$

The derivation given here is only valid for body biasing and threshold voltage scaling. When scaling the supply voltage, formulas need to be adjusted accordingly. Furthermore, scaling the supply voltage also affects the dynamic power with a quadratic influence. Using different voltages however requires careful matching between the regions and is not considered in this thesis.

Power analysis will be mostly important for performance adjustments in an application: When the application is run at nominal conditions, the power can be reduced as the control voltage will be reduced in various sections. PVTA compensation uses the same control mechanism, but will not necessarily lead to leakage reduction: The PVTA effects increase the delay and therefore require adjusting the control voltage in ways which will increase leakage currents. When the process variation in a given IC leads to reduced propagation delay compared to the nominal delay, reductions in leakage currents may be observed though. When FPGA applications are placed for worst-case conditions, this will commonly be the case and process variation compensation can reduce the leakage.

If PVTA effects are analyzed over time, the energy may be more meaningful than the instant power. Relative energy can easily be obtained through integration of relative power over time.

9.3 Functional Runtime Simulation

To validate the logic invasion and characterization concept of section 8.2, a testing framework has been set up in QuestaSim. First, various small unit tests have been derived for the individual parts of the FPGA. These are also used as regression tests, to quickly find broken changes during development. In addition, a large integration test has been devised to test the system as a whole with all parts included.

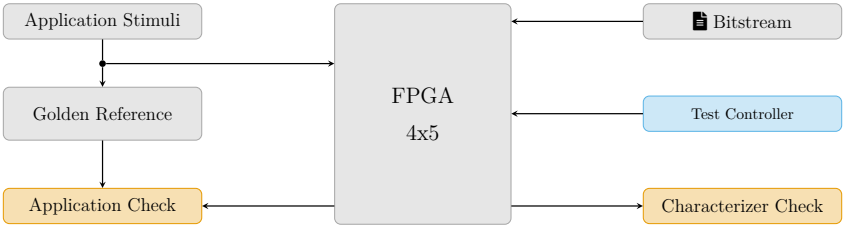


Figure 9.6: Test setup to validate logic invasion and logic characterization functions. A 4x5 FPGA is programmed with a sample application bitstream. The application is provided with stimuli and its outputs are compared to a golden reference. In parallel, the FPGA transparently invades the logic and provides characterization results, which are validated as well.

The structure of this integration test is shown in figure 9.6. It consists of the 4x5 FPGA to be tested and various helper logic. 4x5 is the smallest configuration where the logic invasion can be tested appropriately, as it contains two usable rows apart from the reserved one for logic invation. A simple test application was then synthesized for this FPGA, using 3 input signals to generate 3 output signals. The bitstream is embedded in the simulation and after initial programming, the application virtually runs on the FPGA in the QuestaSim simulation.

The *Application Stimuli* generator periodically repeats all possible input combinations. The stimuli are then provided to the FPGA on the IOB ports used by the test application. In parallel, stimuli are also provided to the golden reference model. This model calculates the expected outputs independently, reimplementing the test application's logic operations in VHDL code. The output of this golden reference and the test application running on the PARFAIT FPGA are continuously compared and checked for equality. If values differ at any point in time, the test fails.

After the test application has been programmed and some time has been spent verifying the application output, the *Test Controller* enables logic characterization in the FPGA. At this point, the `ProgController` embedded in the FPGA will start transparent logic invasion. Application stimuli testing continues in parallel, to verify that logic invasion does not upset the test application operation.

The measured delays are verified in the *Characterizer Check*. This component verifies the locations and delay values produced by the `ProgController` in the FPGA. To simulate the LUT delay, VHDL transport delays were modeled for the LUT as shown in listing 9.1.

```
1 if (split = '1') then
2   q0 <= transport table(sel and "011111") after 235 ps;
3   q1 <= transport table(sel or "100000")  after 235 ps;
4 else
5   q0 <= transport table(sel) after 235 ps;
6   q1 <= 'U';
7 end if;
```

Listing 9.1: VHDL implementation of a LUT with delay information added. Type casts have been omitted to increase readability.

With LUT delays of 235 ps, the period of the ring oscillator is $2 \cdot 11 \cdot 235\,\text{ps} = 5.17\,\text{ns}$. The `ProgController` is clocked at a period of 10 ns and measures for 100 clock cycles, yielding a measuring time of 1000 ns. Comparing the oscillator period to that, the expected count is 194. Converting this to the LFSR value yields a value of 241, which is the expected output of the characterizer in the test case. As the functional simulation does not model any difference in propagation delays in different LUTs, the expected value is the same for all tested CLBs. The *Characterizer Check* also validates that values are emitted for all CLBs in the FPGA.

In addition to the automated test cases, the measuring time has also been verified manually through observation of signal waveforms. This ensures that the measuring time was implemented correctly, which is crucial for the system. Validation of the system with *RGATE* or other ULMs can be performed in the same way. As the logic invasion system is independent of the logic cell used, results are however expected to match the LUT case.

9.4 Co-Simulating DVS

The previously introduced functional simulation shows the working principle of the PVTA invasion. It can however not directly answer the question, how much power can be saved or how well the controller reacts to PVTA changes. This information can not be evaluated in the pure digital simulation, as the effects cannot be modeled there. As a solution, the VHDL FPGA implementation was combined with the propagation delay and PVTA models from chapter 4. This section introduces this QuestaSim based co-simulation framework. Apart from the architecture and models, it will also include VPR for timing and power evaluation. Parts of this section were originally published in [Pfa23b], but it has been extended with further details.

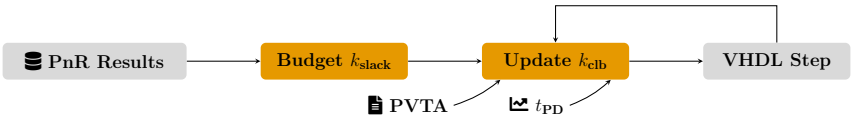


Figure 9.7: Process flow for the PVTA compensation co-simulation. The simulation loads results previously obtained using the VPR place and route flow. It then calculates the initial k_{slack} factors once before starting the main operation loop. In this loop, the simulation repeatedly updates the VHDL controller state and recalculates the k_{CLB} factors based on the control variable and PVTA conditions.

Figure 9.7 shows the general workflow of the co-simulation: First, applications are placed and routed using VPR as explained in chapter 6 for ULMs and in section 4.5 for LUT based architectures. The place and route files are then loaded into the co-simulation, which is implemented as a QuestaSim plugin. It uses an embedded version of VPR to calculate the initial slack budgets k_{slack} once for all regions. After this initial step, it repeatedly updates the k_{CLB} factors using the propagation delay model and the scenario models. The control input for the propagation delay model is directly obtained from the VHDL architecture implementation. This VHDL code is then given the k_{slack} and k_{CLB} factors and one time-step of the DVS controller is simulated. After this, updated k_{CLB} values are calculated and the process repeats.

As an optimization, the logic invasion based measurement is not used here and k_{CLB} factors are directly provided to the DVS controllers in the tiles instead. The main reason for this is that the simulation models produce relative delays, which directly correspond to these factors. They could be used to derive absolute LUT or ULM delays through scaling of a typical value, but there is

little benefit in doing this. It would significantly increase simulation time and provide no new information, as the logic invasion measurement was already evaluated independently.

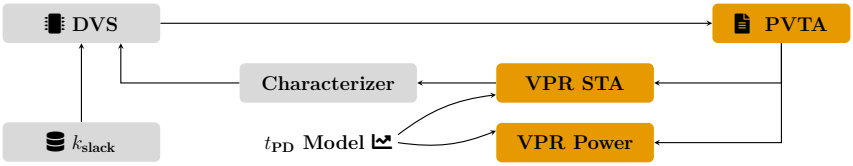


Figure 9.8: QuestaSim and VPR based co-simulation framework to simulate region based adaptive voltage scaling FPGA architectures. Grey blocks are implemented in HDL as part of the FPGA architecture and simulated using QuestaSim. Blocks in orange are either embedding VPR C++ code or a Lua interpreter, evaluating PVTA models as `.lua` scripts.

Figure 9.8 shows a structural view of the co-simulation extensions. As shown in figure 9.7, the overall simulation setup realizes a closed feedback loop. This loop can be used to evaluate either the *DVS* controller, user provided scenarios of external influences *PVTA*, or the t_{PD} *Model*. The left-hand side of the figure is part of the simulated FPGA architecture and is modeled in HDL using QuestaSim. It consists of the *DVS* controller, which adjusts the performance of regions dynamically, memory which stores k_{slack} characterization values, and the *Characterizer*, which assesses the current performance within a region. The simulation framework does not dictate any concrete implementation of those components, giving maximum flexibility to user supplied models.

The right-hand side of figure 9.8 is part of the simulation environment, implemented in C++, and interfaces to the HDL code using QuestaSim's Foreign Language Interface (FLI). It takes the requested control voltage from the architecture's *DVS* controller and passes it to user-supplied *PVTA* scenario models written in LUA. Scenario model results are passed to other modules to derive the k_{CLB} timing degradation factors through the t_{PD} models. Using this information, a static timing analysis with updated delays can be performed in VPR. The STA pass allows to assess the target application's timing requirements under changing operating conditions. The results are also forwarded to the *Characterizer*, which estimates the current delay in each region. To obtain a region's total performance, the characterizer uses the worst factor of all CLBs within a region. The interface integrating C++ and VHDL code is shown in figure 9.9.

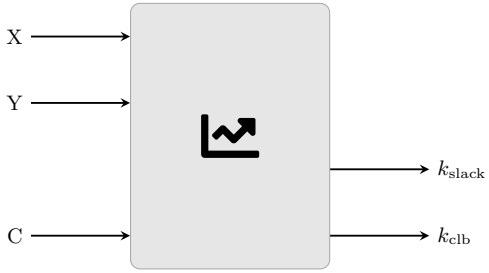
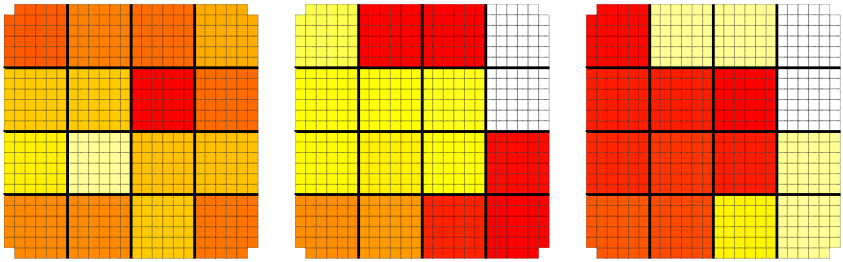


Figure 9.9: Logic interface block which provides the interface between software and VHDL code in each region. The interface receives its position in the grid as well as the control input. As output, it provides the current k_{CLB} factor as well as the initial k_{slack} factor for the region.

For power analysis, model values are forwarded to the *VPR Power* estimator, most notably the current voltage for a region. The VPR power analysis region extensions can then be used to calculate per-region power for all primitive blocks. The support of different `.tech` files for each region and point in time allows full flexibility for simulation of DVS systems. For the evaluation presented in this thesis though, the simpler analysis method introduced in section 9.2 on page 240 will be used.

Figure 9.10 shows a graphical depiction of an example benchmark evaluation. In this figure, subfigures **(a)** and **(b)** show the k_{CLB} factors at the beginning and at the end of the simulation. It can be seen that in the beginning, the measured delay largely conforms to the example process variation introduced in figure 4.24c on page 154. At time t_1, the FPGA architecture's DVS controller has adjusted the power supply in each region and made the measured delay variations match the applications slack factors, k_{slack}. The adjustment in the control voltage needed to achieve this, is shown in figure 9.10c. This result is mostly a combination of the process variation map and the slack factor, as the voltage adjustment has to both cancel the process variation and adapt the local performance to application requirements.

The simulation framework also enables validation of timing paths under voltage scaling: The slack values can be assessed through additional STA for the target application at different time steps in the simulation. The results for this demonstration are shown in figure 9.11, where regions are numbered from bottom-left to top-right. Analysis of the minimal slack in each region shows that all paths still achieve timing closure and most critical paths have



(a) t_0:Delay Factor (b) t_1:Delay Factor (c) t_1:Voltage

Figure 9.10: Delay factors k_{CLB} and region voltage at start and end of simulation. Unused regions are depicted in white color. **(a)** and **(b)** show the current delay factors at the respective point of time. Brighter color indicates a smaller factor, i.e. delays in the region are smaller compared to the nominal case. **(c)** shows the voltage factors used to achieve the delay factors in **(b)**. Brighter color indicates a lower factor, i.e. the threshold voltage in that region has been increased. At the initial time t_0, all region factors are 1.0.

reduced slacks. This also applies for the averages in most regions, showing that all paths are affected similarly.

An important observation is that some regions even provide larger slack values. This is caused by the initial measurement k_{slack} being based on the initial VPR

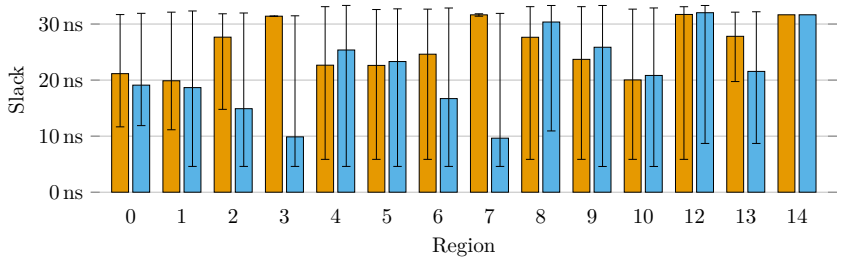


Figure 9.11: Available slacks for each a region. Bars depict the mean over the slacks of all paths in a region and error bars show the minimum and maximum slack values. The nominal initial distribution after placement, ignoring the device specific variation model, is shown as white bars. Black bars show the final distribution with adjusted voltage factors at t_1.

placement, which does not include device specific variation. In some cases, the device specific variation might provide locally better performance in some regions than the worst case corner model, that was used for k_{slack}. Another observation that can be made, is that the slack values of critical paths have not been reduced to zero. As explained previously, this happens when paths traverse other critical regions: For example, the worst path in region 3 could be scaled by factor 4.73 and the analysis assumes it is scaled by this factor in all regions it traverses. However, the path also passes through region 2, which contains another, more critical, path, and can only be scaled by 1.59. The effect is also further explained by the paths used for characterization not being identical to the real application paths.

9.5 Benchmark Applications

Results in FPGA research also depend on the benchmark user applications used to evaluate the architecture, so a representative standard set of benchmarks had to be chosen. Out of the benchmarks offered by VTR, the *MCNC20* benchmarks have been discarded as being too small: The VTR documentation discourages their use, as results obtained using those are no longer representative for real-world applications.

The *Titan* set of benchmarks provides large-scale benchmarks for FPGA design. These benchmarks are however only available as pre-synthesized netlists. Such netlists are already mapped to a certain logic generator primitive, usually a specific LUT type. They can therefore not be used to target ULM based FPGAs or FPGAs with other custom LUT types. This set of benchmarks has therefore also been discarded.

Newly available *Koios 2.0* benchmarks could theoretically be used in future work. The stable VTR distribution used for the evaluation of this thesis does however not yet support these benchmarks. Synthesizing the benchmarks with the old ODIN version included in stable VTR is not successful, as some Verilog features are not supported. As the benchmarks were designed for machine learning use case evaluation, which is not relevant here, they have not been used. Similarly, the *Symbiflow* benchmarks have not been used as they are meant to be used only as regression test suite for some specific architectures. *NOC* benchmarks have also not been used, as the PARFAIT architecture does not provide NoC support.

Instead of these, the *VTR Benchmark* standard set has been used. Those medium-sized benchmarks are directly supported in VTR, and can be mapped to any logic generator. To enable evaluation of all benchmarks, the PARFAIT architecture had to be extended with hard memory blocks. The evaluation was then performed using the *run_vtr_flow* tools: At first, all benchmarks were mapped using an `auto_layout` version of the architecture. Then, the maximum FPGA size to realize the benchmarks was determined. After that, all benchmarks were mapped to an appropriate `fixed_layout` of fixed size for all applications.

Part III

Final Remarks

This page intentionally left blank

Chapter 10

Evaluation

The following sections will present various evaluation results for the PARFAIT FPGA. First sections evaluate selected individual aspects, whereas the final section presents power saving and PVTA compensation results for the whole system.

10.1 Ambipolar Standard Cell Application

Non-reconfigurable logic can be implemented in RFET using the custom standard cell library introduced in chapter 5. In this section, implementations of test circuits using these cells will be evaluated. At first, simple combinational circuits will be analyzed. Then, a more complex cryptographic accelerator will be evaluated to demonstrate sequential circuits.

Combinational Cells

To compare the results of synthesis of combinational circuits, the full adder cell was synthesized with both planar RFET and SOI reference timing libraries. A manual mapping was performed to ensure the netlists in both technologies are equal. These results have previously been published as a part of [Reu21].

The timing report of the full adder circuit, presented in table 10.3 on page 257, lists the internal gates (1, 2 and 3) in the full adder, referring to gate numbers in the full adder schematic in figure 5.4a on page 169. Gates which are directly connected to an output do not drive any loads, as ideal high impedance circuit outputs were used. Because of this, the timing of internal gates, which do have loads, will be analyzed.

Although the *NAND* gates (index 2 and 3) show an increased cell delay, the overall critical path of the planar RFET circuit (1130 ps) is lower than for the SOI reference technology (1331 ps). The reason for this can be found when comparing the *XOR* gate (index 1) delay: While the planar RFET *XOR* gate has a cell delay of 336 ps, the cell delay of the *XOR* gate provided by the SOI reference is, with 493 ps, significantly higher. A substantial difference between the timing of the planar RFET and the reference technology is that the critical path is different: For the planar RFET, it runs from input a to the carry output c_{out}. For the SOI technology however, the critical path leads from a to output y. This observation reinforces the assumption, that the planar RFET boosts performance of largely XOR based circuits when compared to the SOI reference technology.

Table 10.4 on page 258 shows reduced versions of critical paths in **(a)** a 32 bit carry ripple adder, **(b)** a 4 bit checked adder and **(c)** the ARX cell. The circuits are based on the building blocks in figure 5.4 on page 169. They are completely combinational and therefore synthesized to exclusively RFET cells. Using fanout information and the input capacitance of the cells and the wire loads, Cadence Genus determines the output load of all cells. It then calculates the delay according to the cell type, arc (input pin to output pin used by the analyzed timing path), the input slew rate (not shown), the input transition edge (rise or fall) and the output load. Delays match what a manual analysis suggests, which ecourages the view that the extracted timing information has been properly transformed into the `.lib` file. Further extraction and validation of the `capacitance_max_rise` and `capacitance_max_fall` values has been performed, to ensure that rising and falling transitions are both handled correctly.

Cryptographic Accelerator

In the following, the various system architectures for the cryptographic accelerator will be presented first, followed by a mapping to RFET. Results for the system architecture have been presented previously in [Pfa19], results for the RFET mapping in [Reu21].

FPGA Evaluation Xilinx *Vivado* 2018.3 was used for evaluation. Retiming was explicitly enabled, and the VC707 board and Virtex 7 XC7VX485T-2FFG1761C FPGA were targeted for implementation. Architectures can be parametrized on core count and the maximum reachable clock frequency reduces with higher count. This can be explained by increased stress on placement and

routing. Throughput depends on both the number of cores and the clock frequency, so figure 10.1 compares different implementations' throughput vs. the required resources. The Pipeline implementation is shown as points, as it does not provide a Core count parameter to balance resources vs throughput. $d =$ denotes whether DSP blocks have been used for the ARX cell (1) or not (0) and $r =$ gives the number of introduced pipeline registers. Table 10.5 on page 259 shows that these implementations surpass all state of the art ChaCha implementations, except for the low-resource optimized implementation by At et al., when comparing bitrate per slice. The Block Memory and Pipeline implementation also surpass Advanced Encryption Standard (AES) state of the art, the Pipeline implementation even by a factor of 8. Results shown are for ChaCha8. To calculate numbers for ChaCha12/20, divide throughput by 1.5 and 2.5 for the Register and Memory implementations. For the Pipeline implementation, resource requirements are increased by these factors and the maximum clock frequency and therefore throughput may also be affected.

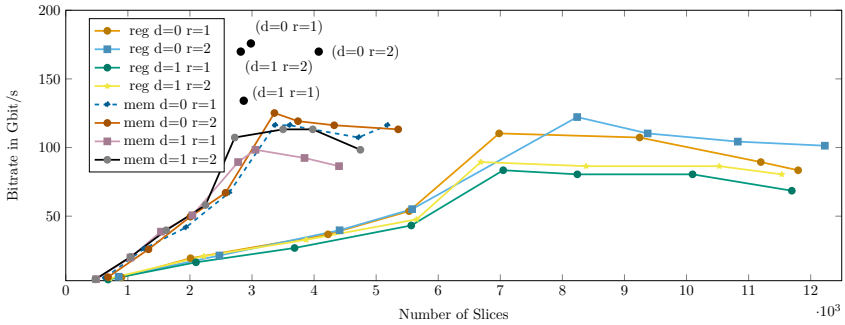


Figure 10.1: Comparing throughput vs. number of slices for various ChaCha accelerator system architectures.

RFET Evaluation To evaluate the RFET *.lib* file, the ChaCha accelerator circuit was synthesized in Cadence Genus. For this synthesis run, pure logic synthesis based on the *.lib* file was performed. Physical information from *.lef* files or capacitance tables are omitted. The synthesis makes use of the SOI D FF and is therefore not completely synthesized in RFET technology. Unlike for the small circuits, the `dont_touch` attribute is not used, permitting optimization during synthesis. As the cells have only been characterized for up to 15 fF output capacitance, synthesis tools insert buffer cells if deemed appropriate. This however increases the path lengths and reduces achievable target frequencies.

Table 10.1 shows the critical path in the ChaCha accelerator for the planar RFET timing library. The critical path length of the SOI reference technology is 58 % of the critical path length of the planar RFET, leading to an advantage of 42 % in speed for the reference technology. Inspecting the critical path of the planar RFET synthesis shows that 63 out of 96 gates in the critical paths are *NAND* gates. The comparison of cell timing characteristics between planar RFET and SOI reference cells in [Reu21] states that the relative performance of the planar RFET technology is worst for the *NAND* cell. The overall worse performance of the planar RFET implementation for the ChaCha accelerator can therefore be partly traced back to the worse relative performance of the proposed planar RFET cells. On the other hand, the missing derating of the SOI reference, which would compensate the device specific characteristics like channel length and threshold voltage, benefits the reference as well [Reu21].

Gate	**Arc**	**Edge**	**Fanout**	**Load** [fF]	**Delay** [ps]
DFF	C->Q	R	10	52.3	1150
INV	A->Q	F	2	9.1	367
NAND	B->Q	F	1	5.5	582
NAND	B->Q	R	1	10.2	416
XOR	B->Q	F	1	5.6	222
NAND	B->Q	R	1	6.7	364
NAND	B->Q	F	1	3.3	552
DFF	-	F	1	0	0

Table 10.1: Excerpt of critical path analysis in ChaCha accelerator for RFET standard library.

Table 10.2 shows the area summary for the ChaCha accelerator as reported by Cadence Genus. As can be seen, the cell area is largely zero. This is expected, as the *.lib* file is currently missing area definitions for the RFET cells and only SOI D FF cells provide area information. Although area reports are currently of limited use because of this, they can be used to verify whether wire load models are working correctly.

Instance	Cells	Cell Area	Net Area	Wire Load
chacha_reg	36386	0	33952	wload_30k
core[0].instance	12219	0	10143	wload_05k
arx[3].instance	2096	0	1652	wload_01k
pipe	519	0	350	wload_500
addi	1004	0	811	wload_500
roti	417	0	323	wload_100
xori	156	0	71	wload_100

Table 10.2: Genus area report for ChaCha accelerator for RFET standard library.

Technology	N	Type	Wire Load		Total	
			R [Ω]	**C** [fF]	**C** [fF]	**T** [ps]
Planar RFET 1130 *ps*	1	XOR	8	0.3	13.6	336
	2	NAND	3	0.1	5.5	429
	3	NAND	3	0.1	6.8	158
SOI 1331 *ps*	1	XOR	8	0.3	9.1	493
	2	NAND	3	0.1	3.1	373
	3	NAND	3	0.1	3.7	146

Table 10.3: Delay analysis for the full adder in RFET technology in comparison to SOI technology.

Gate	Arc	Edge	Fanout	Load [fF]	Delay [ps]
NAND	B->Q	F	1	5.7	373
NAND	A->Q	R	2	13.9	346
NAND	B->Q	R	2	13.9	453
XOR	B->Q	F	1	0.3	696

(a)

Gate	Arc	Edge	Fanout	Load [fF]	Delay [ps]
NAND	B->Q	F	1	5.7	373
NOR	B->Q	F	1	5.8	280
NOR	A->Q	R	1	7.2	329
XOR	A->Q	F	1	0.2	643

(b)

Gate	Arc	Edge	Fanout	Load [fF]	Delay [ps]
NAND	B->Q	F	1	5.7	373
NAND	A->Q	R	2	13.9	346
XOR	B->Q	F	2	7.9	771
XOR	A->Q	F	1	0.3	610

(c)

Table 10.4: Genus critical path analysis in combinational circuits for RFET technology. **(a)** 32 bit carry ripple adder: Reference is 78 % faster. **(b)** 4 bit checked adder: Reference is 26 % faster. **(c)** ARX cell: Reference is 42 % faster.

Variant	Cores	DSP	Depth	f_{max} MHz	LUT	FF	BRAM18	DSP	Slices	Bitrate Gbit s^{-1}	Bitr./Slice (Mbit/s)/slice
chacha_reg	1	yes	1	256.3	1693	1502	0	20	682	3.82	5.74
			2	312.5	1738	2025	0	20	713	4.66	6.69
		no	1	362.5	2369	2152	0	0	896	5.40	6.17
			2	393.7	2392	2873	0	0	852	5.87	7.06
chacha_mem	1	yes	1	275.0	862	1162	4	8	487	4.10	8.62
			2	268.7	871	1260	4	8	476	4.00	8.61
		no	1	343.8	1366	1831	4	0	633	5.12	8.25
			2	375.0	1387	2210	4	0	680	5.59	8.42
chacha_pipe	-	yes	1	281.2	4556	13484	0	144	2867	134.09	47.89
			2	356.3	5633	15707	0	144	2819	169.87	61.71
		no	1	368.7	9138	18004	0	0	2982	175.82	60.38
			2	356.3	10101	25680	0	0	4075	169.87	42.69
Xiphera (ChaCha20) [253] (Xilinx UltraScale+)	-				< 4000					0.50	> 0.26
At et al. [254] (Xilinx Virtex-6 XC6VLX75T-2)	-			356.3			2		49	0.60	12.54
Strömbergson [255] (Xilinx Artix-7 XC7A200T-3FBG484)	-			100.0	3837	1949			1076	5.96	5.67
Silitonga et al. (AES, HLS)[256] (Xilinx Zynq-7000)	-				11704	8512				43.90	7.68
Soltani et al. (AES)[257] (Xilinx Virtex-6 XC6VLX240T)	-								35328	260.15	7.54

Table 10.5: Resource and clock frequency comparison for selected implementation and configurations.

10.2 Ambipolar Reconfigurable Cells

For the evaluation of the logic cell, all parameters were tuned as explained in chapter 6: The logic cluster has 40 inputs that feed an input crossbar. It combines 5 simple, two-input logic elements with one FF each, 5 FLEs with one FF each and 10 FLE without FF. Each of the previously mentioned cells provides one output of the 20 outputs of the logic cluster and there are no internal-only cells in the final architecture. The evaluation used a FLE with the topology of figure 6.7a, as results did not show large benefits when using fracturable outputs. As will be shown, benefits of fracturable cells can only be realized when the more advanced combined `genlib` EDA approach is used. Results currently do not include an absolute timing analysis for the FPGA architecture, as this requires dynamic characterization of at least the ULM in the RFET technology. Preferable, all FPGA components should be characterized in the target technology to get detailed results, but due to incomplete PDKs, such a detailed analysis is currently not possible. As timing and delays also influence FPGA size due timing driven routing algorithms, timing driven routing and packing was disabled in VPR for this evaluation.

Simple EDA Flow Figure 10.2 on the next page show results obtained through the VTR benchmark flow. Data has been collected from benchmarks and averaged accordingly. Figure 10.2a shows the input utilization of the logic cluster. The architecture uses up to 32 inputs, which is the maximum channel width in the routed design. As VTR automatically increases the channel width in benchmark mode, this shows that a larger channel width was not required. One limiting factor of the architecture is therefore likely the amount of LEs in each cluster. This conclusion is assured by the output utilization of the logic cluster shown in figure 10.2b. It can be seen there, that out of 20 available outputs, an average of only 6.9 are used. Both FLE and simple LE are almost fully utilized, as can be seen in figures 10.2c and 10.2d. This further reinforces the conclusion that the main limiting factor of logic expressiveness in this architecture is still the LE. Introducing more LE however increases the size of the input crossbar, which increases area and makes comparison to the reference architecture more difficult. As will be shown in this section, even though this architecture does not yield full utilization of logic cluster inputs and outputs, in most benchmarks the final FPGA size is close to the reference architecture. A problem can be seen in figure 10.2e, which shows the utilization of FLE inputs. It can be seen that without the fracturable LE, the LE is fully utilized in only a few cases. In the most common case, simple 2-input functions are mapped to the LE and most inputs remain unused. As

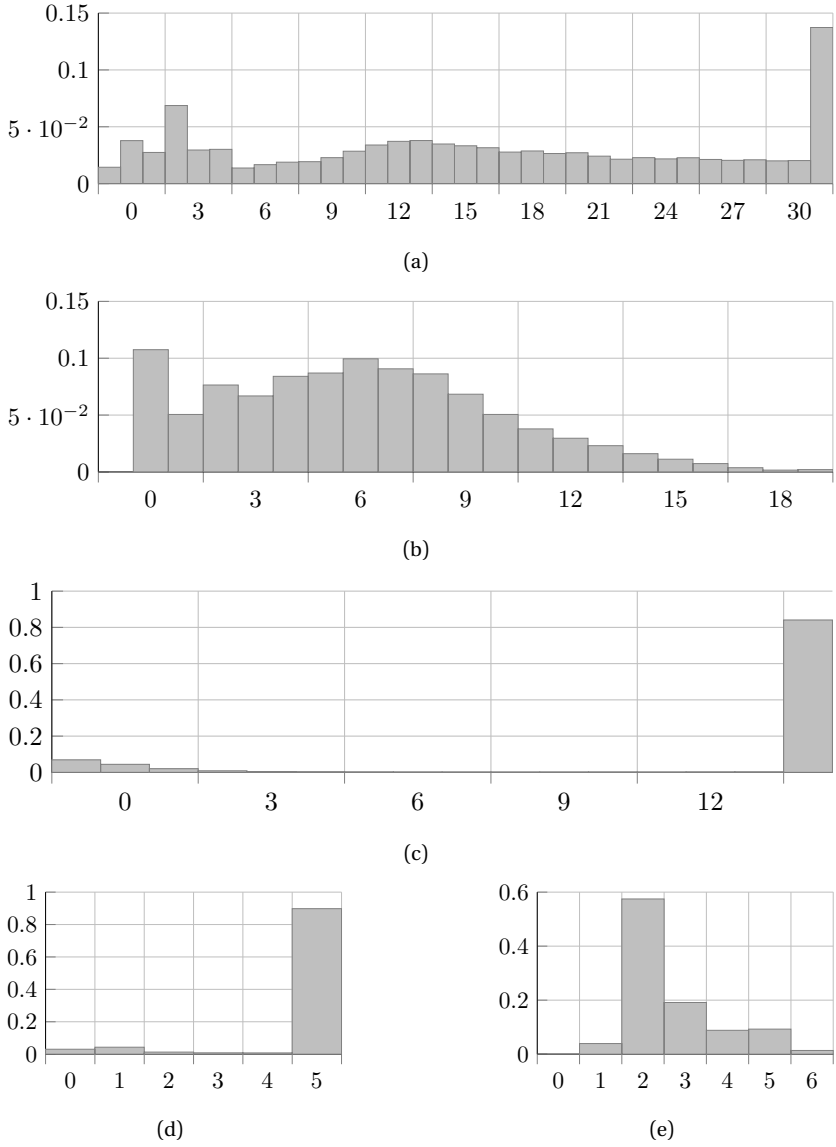


Figure 10.2: Utilization statistics for the proposed logic cluster. **(a)** Cluster input utilization. Mean = 16.8. **(b)** Cluster output utilization. Mean = 6.9. **(c)** Utilization of FLEs in cluster. Mean = 12.8. **(d)** Utilization of simple LE in cluster. **(e)** Utilization of the FLE inputs.

some complex functions are successfully mapped to the FLE, it can however be seen that the approach in general is working correctly. The issue with underutilized FLE inputs will be further discussed for the results using the advanced EDA flow.

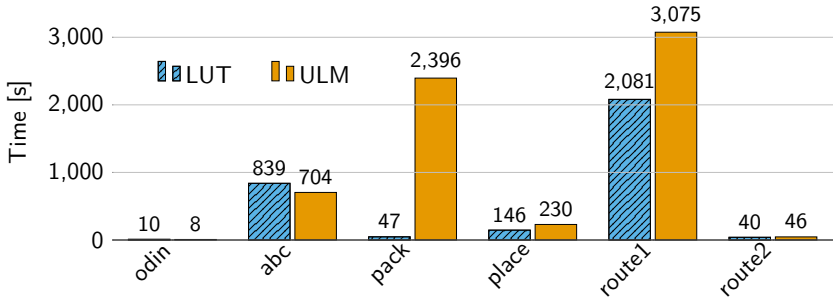


Figure 10.3: Comparison of EDA metrics for ULM clusters and LUT. This figure shows the EDA tool runtime, averaged over all benchmark circuits.

Figure 10.3 shows the average tool runtime compared to the runtime in the LUT based FPGA case for the evaluated benchmarks in the relevant tool steps. As can be seen, runtime is largely similar for most steps. For the packing phase however, the runtime is increased by almost two decades. An increase in packing time is expected, as the packer has to pack all ULMs into the FLE in the simple EDA flow. As there are numerous unused ULMs due to their limited expressiveness, more packing steps are necessary than in a LUT based architecture. In LUT-based FPGA architectures on the other hand, packing is often used only to combine FF and LUT or multiple LUTs into one cluster. Nevertheless, this increase in packing runtime is excessive and a severe limitation of the ULM synthesis toolflow. As will be explained in the next section, this overhead can fortunately be reduced using the advanced EDA flow. In addition, there is a slight increase in routing runtime. This increase can be explained by the mapped ULM benchmarks being slightly larger than the LUT baselines.

Figure 10.4 evaluates the FPGA sizes for the used VTR benchmark circuits. FPGA size is evaluated in blocks, counting logic and memory blocks but excluding the peripheral blocks. It can be seen that the ULM based architecture uses less than 10 % more blocks in most cases. For the `LU*PEEng` and the `mcml` circuits, the ULM based FPGA was however up to 50 % larger. This suggests that further architecture tuning may be beneficial for some circuits and that

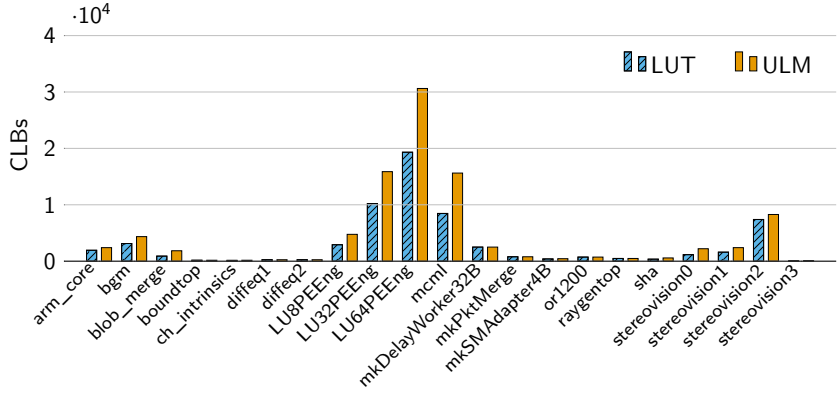


Figure 10.4: Comparison of EDA metrics for ULM clusters and LUT. This figure depicts the FPGA device size in blocks for the various benchmarks.

the FLE expressiveness does not yet fully match the LUT baseline. The issue can be addressed by introduction of more FLEs in the logic cluster, at the expense of increasing the input crossbar size.

Advanced EDA Flow To address underutilized inputs of LE cells and non-utilized outputs of the fracturable cell, results have been reevaluated with the advanced tool flow. As this flow models combined functions to be realized by multiple ULMs in one FLE cell, it reduces the amount of work in the packing step. The packing algorithm is then only used to pack one or multiple combined functions into one FLE and operates in the same way as for fracturable LUTs. Figure 10.5 shows an evaluation of the fracturable LE using the advanced flow. As can be seen in figure 10.5a, the EDA flow now successfully makes use of multiple inputs in the FLE. Functions with one input are still used, as those are needed to realize inverter or buffer functions in some cases. Two or three inputs are never used for the tested benchmark sets. This clearly shows that in cases where a 2 input function is mapped to a FLE, it gets fractured and another function is additionally mapped. More than half of the cells have fully utilized their inputs, a significant difference to the simple EDA results of figure 10.2e. Figure 10.5b shows the utilization of FLE outputs. It can clearly be seen that multiple outputs are now actively used. Whereas cells with only one mapped output suggest that the whole cell has been mapped to one function, two and three outputs can be intermediate results or be fractured cells implementing multiple functions.

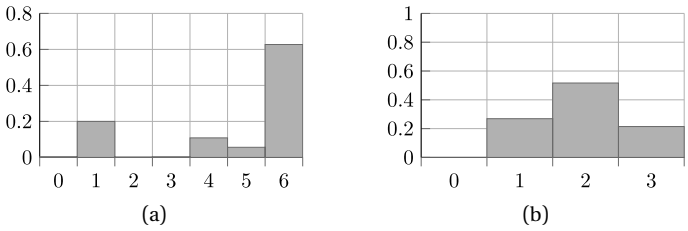


Figure 10.5: Utilization of the 6-input FLE when using the advanced, combined function EDA flow. **(a)** Amount of FLE inputs used. **(b)** Amount of FLEs outputs used.

10.3 Power Management Regions

To demonstrate the use of power management regions and mode assignment approaches, a simple architecture supporting two power modes is evaluated. Parts of this evaluation were originally published in [Pfa23b], but this version has been extended with a more thorough evaluation of modified FPGA architectures. For the static mode assignment, modes have been assigned in an alternating pattern. The architecture used for evaluation is based on the 40 nm `k6_frac_N10_40nm` architecture shipped with VTR. It contains only basic logic elements and no memory, allowing easier comparison. To extend the architecture, delay values for various supply voltages were obtained in a similar 45 nm PPDK using COFFEE 2 [258] and HSPICE. Power and delay values obtained with 1 V supply voltage were used for the high performance power regions. For the low power regions, varying power and delay values were evaluated. To evaluate the architecture, MCNC benchmarks shipped with VPR were used. As a system-level measurement for power reduction, the amount of used CLBs which are in low-power mode are compared to the baseline architecture. All result figures then show averages for the evaluated benchmarks.

Figure 10.6 depicts results for the static assignment strategy, showing that between 30 % and 40 % of the CLBs have been placed into low power regions. Larger region sizes, which need less hardware resources in implementation, do not strongly affect the amount of low-power CLBs in this evaluation. As the target frequency was not set (best-effort), there's a decrease of the maximum frequency of factor 1.1 at 0.95 V up to 1.8 at 0.7 V. When actively using static assignment, it should be assessed whether setting a fixed target frequency is necessary.

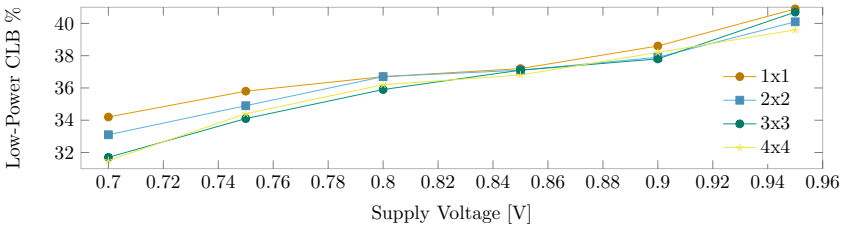


Figure 10.6: Static region assignment: Fraction of CLBs in low power mode for different region sizes and low power supply levels. The supply voltage in high-performance regions is 1 V.

Unlike static assignment, a dynamic assignment will not affect maximum application frequencies: The algorithm essentially modifies the distribution of timing slacks, reducing the slack of non-critical paths. For an exemplary evaluation, the architecture is modified to specify only one region type with two possible voltage/power/delay combinations. The high performance mode continues to operate at 1 V and the low power mode uses varying voltage, power and delay. Figure 10.7 shows the average amount of cells in low-power

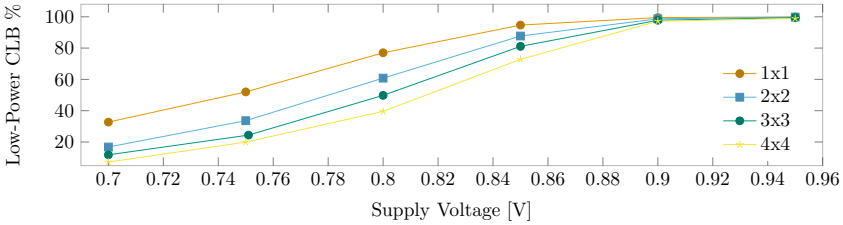


Figure 10.7: Dynamic region assignment: Fraction of CLBs in low power mode for different region sizes and low power supply levels. The supply voltage in high-performance regions is 1 V.

for the dynamic strategy. It illustrates that a large amount of CLBs can be operated in low power mode, but results depend heavily on the parameters chosen: A low-power supply voltage of 0.9 V will enable almost all CLBs to be in low-power mode. At 0.75 V for most region sizes, less than 25 % of the CLBs are placed in this mode. The other factor largely influencing the results is region size: A finer granularity in voltage adjustments allows for better results. For practical FPGA design, such fine-grain voltage selection will have to balanced with the need for excessive additional resources.

10.4 Power Management and Compensation

The following section will discuss various tests and evaluations using the final PARFAIT architecture. Evaluations will focus on PVTA compensation and power reduction using the co-simulation environment. The logic invasion scheme has been abstracted in the co-simulation to decrease simulation times. Apart from that, the evaluated architecture follows the description in section 8.5 on page 227. This evaluation section explains how to read the result graphs and provides an overview of aggregated results. Additional raw evaluations for eight selected benchmarks are available in appendix F on page 365 and will be referenced in this evaluation section.

General As previously mentioned, this final evaluation does not use the *CNT-DR8F* based LE presented in section 10.2, but the *RGATE* based LE from figure 8.14. The primary reason for this is that the *RGATE* was realized in a technology, for which characterization data is available for various PVTA parameters. This allowed the RFET propagation delay model in section 4.6 to be fitted for multiple parameters, which wasn't possible for the *CNT-DR8F* gate. Using *RGATE* therefore allows for more realistic evaluation, as the delay model matches the technology of the actual cell.

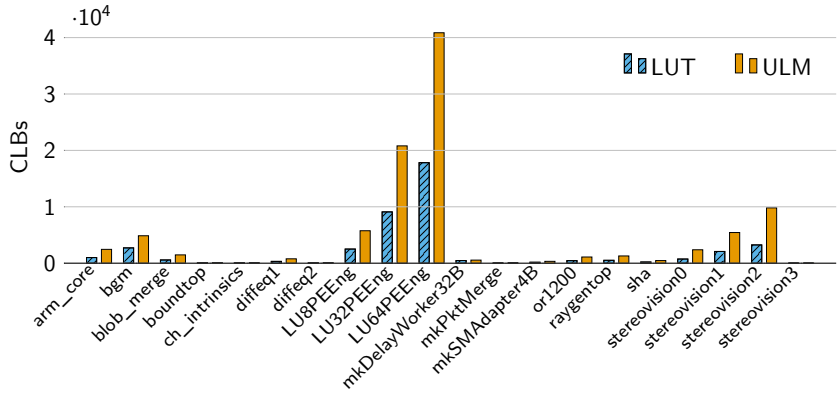


Figure 10.8: Number of ULM-based CLBs in the PARFAIT FPGA vs. number of LUT-based CLBs in the reference FPGA for the evaluated benchmarks.

Figure 10.8 provides an overview of CLB usage in the reference architecture vs. the modified architecture with *RGATE* cells. As can be seen, the *RGATE* based FLE has not been optimized for expressiveness, as was done for the

CNT-DR8F based LE. Due to that, in the worst case there is an increase of up to 3.31 in CLB usage. It should be noted that the *RGATE* LE shown in figure 8.14 needs only 10 bit for storage, plus one bit to select split mode. This is 54 bit less than the LUT used for comparison in the reference architecture.

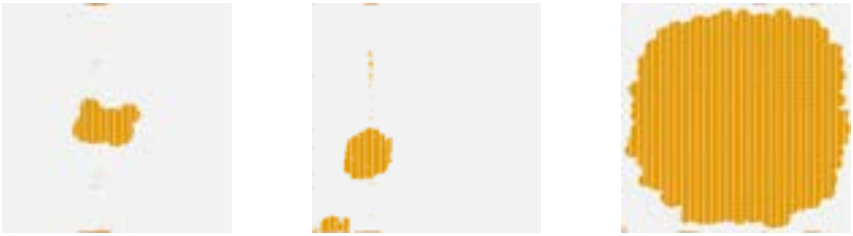
A 3.31x overhead would therefore still reduce the needed configuration storage. The total area however is also determined by the interconnect size, which increases by factor 3.31x as well. A naive approach could simply increase the number of LEs in a CLB, therefore avoiding increase in CLBs and global interconnect. This approach would however increase the size of the crossbar used for local interconnect in the CLB, as well as the number of output pins. A more advanced approach could consider that few of the *RGATE* LEs are actually used in split mode in the benchmarks. Split mode could therefore be removed, and the second output could be driven by an identical implementation of the LE. This essentially enables doubling the logic density without increasing the number of output pins or the crossbar, but input pins need to be shared between two instances of the *RGATE* logic. Further evaluation is necessary to determine whether such a system can be fully utilized in benchmarks. Alternatively, different topologies for the ULM, with e.g. more *RGATEs*, could be considered.

(a) (b) (c)

Figure 10.9: Process variation map used for the PARFAIT FPGA evaluation in this section. The map was generated as explained in section 4.7 and contains 250x250 points, one for each location on the FPGA. **(a)** Pure random variation. **(b)** Spatially correlated variation. **(c)** Final process variation model.

Figure 10.9 shows the process variation map which will be used to simulate process variation for the final results. All benchmarks in this analysis were mapped to a device of size 250x250, as this fits the largest benchmark. Because of this, the process variation map was also generated at a resolution of 250x250

according to section 4.7. This then enables each CLB in the architecture to get an individual process variation factor, as opposed to one factor for a whole region. Subfigure **(a)** shows the pure random variation, which does not have any spatial correlation. Subfigure **(b)** shows the spatially correlated part and subfigure **(c)** shows the combined process variation, which is used in the co-simulation. The variation maps were generated to describe V_{th}, as explained in the introduction of the process variation model. Values are then scaled to be in the range of $[-1,1]$ to fit the P parameter range of the propagation delay models. Darker colors in the figure depict a larger P value, i.e. a CLB with less propagation delay. As this process variation map is representative for process variation, it is used in all the following evaluations. Although the co-simulation easily allows simulation with different process variation maps to assess the effects of those, such an additional evaluation was out of scope of this thesis.



(a) (b) (c)

Figure 10.10: Placements for three exemplary benchmarks which are evaluated in this section on the PARFAIT FPGA architecture. Placements were obtained using the ULM VPR flow introduced in section 6.2 on page 180. **(a)** `arm_core` benchmark. **(b)** `stereovision0` benchmark. **(c)** `LU64PEEng` benchmark.

Figure 10.10 shows placement maps of three selected example benchmarks. Orange color depicts used CLB blocks, red color at the border shows used IOBs and blue color shows used memory blocks. Gray color in general depicts unused blocks and gray stripes are caused by columns of memory blocks being unused. Figure F.1 on page 366 in the appendix shows the placement maps of all evaluated benchmarks. In general, due to the device size being fixed to 250x250 blocks and the benchmarks having widely varying sizes, the results are quite different. For example, figure 10.10a shows an average size benchmark, figure 10.10b a similarly sized one with two clusters of placed logic and figure 10.10c shows the benchmark with the largest utilization. This evaluation section will only provide averaged statistics and explain the

general structure of evaluation graphics using examples. Due to the different placements of benchmarks, individual results may be interesting and are provided in appendix F for all benchmarks and most parameter combinations. The placement of benchmarks was fixed for all evaluations, the co-simulation was configured to always load the pre-placed results shown here.

Power Reduction Figure 10.12 provides an exemplary overview of all evaluations for the `arm_core` benchmark, with RFET delay and power models and no PVTA. Columns show different regions sizes, whereas the rows show different parameter evaluations. The first row shows target delay factors k_{slack}, as obtained from VPR. Values have been color coded according to figure 10.11, where unused blocks are shown in gray. For larger region sizes, the worst case value for all CLBs within a region is used. As can be seen in e.g. figure 10.12c and as expected, this causes larger regions to require the worst case slack value, even if only a single CLB within the region is utilized. k_{slack} values are independent of PVTA and are therefore shown only once. Factors for all benchmarks can be found in figures F.2 to F.5.

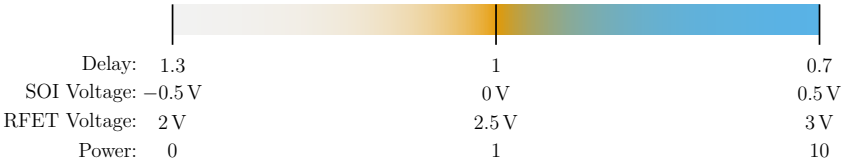


Figure 10.11: This legend shows the color coding for the various heatmaps. Values in the center depict typical values, the left side shows slow values and the right side fast values. Color gradients are scaled exponentially, so that smaller changes in the typical region cause larger variation in colors.

The second row shows the achieved delay factor, k_{CLB}. As in k_{slack} figures, a blue value shows a smaller target factor and depicts smaller propagation delay and therefore faster than nominal circuits. Colors in all graphs have been normalized to common values according to figure 10.11, enabling comparisons between multiple figures.

The third row shows control voltage C. Graphs have been normalized for the range of $[2,3]$ volts, with the minimum value used by the power controller being 1 V. Values outside the color range will be clamped to the range limits. Color normalization comes at the drawback of reduces color resolution for some regions. For example, a control voltage of 2.0 V can not easily be distinguished from a value of 2.05 V. Because of this, some control voltage

and power heatmaps appear to have uniform color, although there actually are differences in the obtained values. The last row in figure 10.12 shows the relative current or power, with blue color depicting higher power. For the absolute color values, the same remarks as for the control voltage apply.

Figure 10.13 shows relative power for the benchmarks. Simulations were performed using 100 simulation time steps, i.e. 100 invocations of the control loop. The value for each benchmark was obtained by first finding the control voltage in each CLB at the last simulated time point. These values were then used with RFET and SOI current models of section 4.6 to obtain the normalized leakage current for all CLBs. The result is assumed to also model the relative change in power, as explained in section 9.2. Finally, the average of these values is calculated to obtain an aggregate value for each benchmark. The results for SOI are shown in figure 10.13a, for RFET in figure 10.13b.

It can be seen that for the SOI model, all benchmarks applications at all region sizes have a relative power of 2.76×10^{-2}. This is caused by the characteristics of body bias control in this SOI technology: The application benchmarks have been placed assuming the worst-case process variation, $P = -1$. This co-simulation on the other hand does not yet include the process variation map and therefore determines the current delay factors k_{CLB} using nominal process variation, $P = 0$. Due to this, each CLB is already 30.2 % faster than what was assumed during application placement. The controller accordingly modifies the body bias to increase delay and save power. The slope $\delta t_{\mathrm{PD}}/\delta V_{\mathrm{BS}}$ is comparatively small, as was shown in figure 4.17 on page 142. This leads to the controller quickly setting the bias voltage to the minimum value, $-0.5\,\mathrm{V}$. Essentially, the controllers configure all region in the slowest low-power mode available, reducing power as far as possible. The power saving is still significant, due to the larger slope of $\delta I_{\mathrm{leak}}/\delta V_{\mathrm{BS}}$ as shown in figure 4.19 on page 144. In this case, region size does not have any influence, as all regions always use the same body bias, regardless of region size.

The RFET simulation in figure 10.13b shows different behavior: The slope $\delta t_{\mathrm{PD}}/\delta V_{\mathrm{PG}}$ is steeper and the behavior of that dependency is exponential, as was shown in figure 4.21 on page 151. This means possible change by controllers for PG voltage are limited, before the delay is increased too much. In addition, $\delta I_{\mathrm{leak}}/\delta V_{\mathrm{PG}}$ is comparatively small, as shown in figure 4.23 on page 153. It should be noted that this leakage current model may however vary a lot with changes in RFET technology and that the measurements used for modelling have a high uncertainty due to difficulty of measuring these

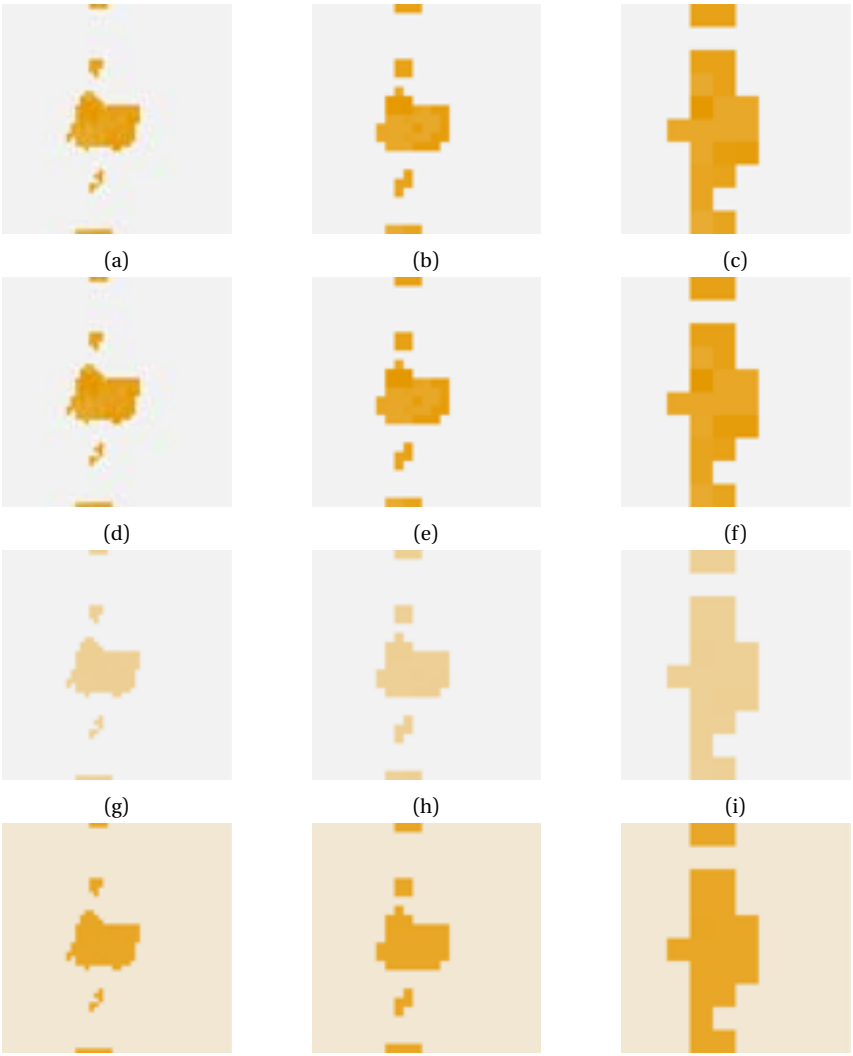
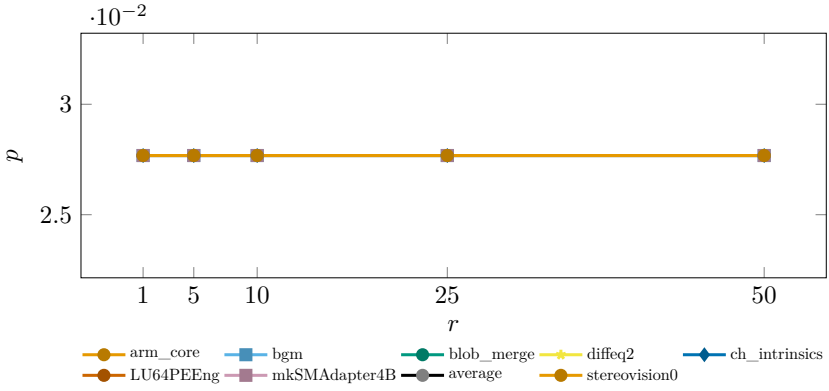
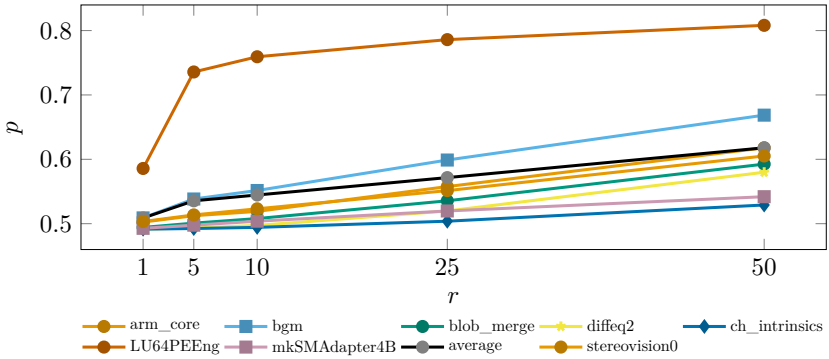


Figure 10.12: Power reduction and delay for the `arm_core` benchmark. Evaluated using the RFET model and without process variation. Rows show from top to bottom: Target delay, achieved delay, control voltage and relative power. Columns show different regions sizes. From left to right: 5x5, 10x10 and 25x25.

values. Refer to [2] for details. It should also be noted that while this effect limits the total energy saving possible using V_{th} scaling in RFET technology, it also means large compensation in performance can be achieved with small changes in program voltage.



(a)



(b)

Figure 10.13: Normalized static power p for the benchmarks versus power region size, simulated without process variation. **(a)** SOI model. **(b)** RFET model.

As the region controllers do not reach limits when scaling control voltages for RFET, region size and resource utilization effects can now be seen. In general, unused resources will still be scaled to the minimum control voltage and power consumption. Higher utilization reduces the number of regions in this low power mode and therefore leads to higher power consumption

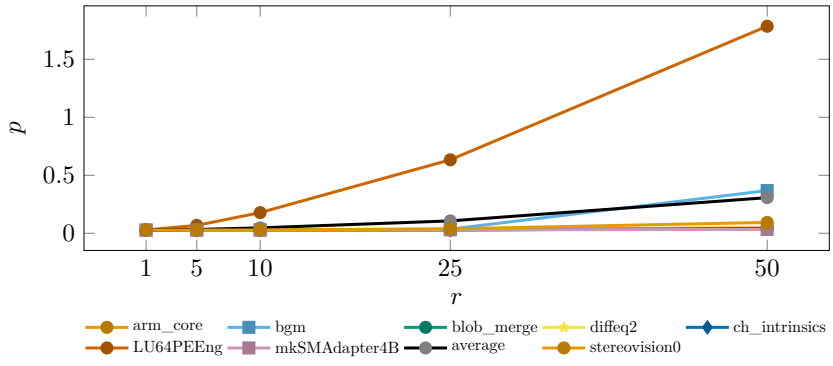
for all regions sizes. This can for example be seen easily in the largest benchmark, `LU64PEEng` having the highest power consumption and the second-largest, `bgm`, following. Region size effects can also be seen: A single CLB with higher performance requirements in a region causes the whole region to consume more power. Therefore, larger regions are expected to increase power consumption. On the other hand, placement of logic is often clustered, which affects the severity of this effect. Figure 10.13b shows that for benchmarks with limited resource utilization, the difference between fine-grain 1x1 regions and large 50x50 regions is less than 20 % and that benchmarks with less resource utilization are less affected by region size. The effects for 25x25 regions are already less severe, so a trade-off between additional logic for voltage scaling and power saving can be made based on these figures.

Process Variation Figure 10.16 shows similar FPGA maps, but this time for SOI technology and including process variation. Target delay factors k_{slack} are shown again in the first row to easily compare them to the graphs in the rows below. The figure also shows achieved delay, control voltage and relative power using the same conventions as the previous figures. It can be seen that even for SOI technology, with process variation not all regions can be operated in the lowest power mode. Higher power regions are required according to a combination of the placement map and the process variation map: Locations where the target slack factor is smaller, or the process variation leads to worse performance, require higher body bias voltages.

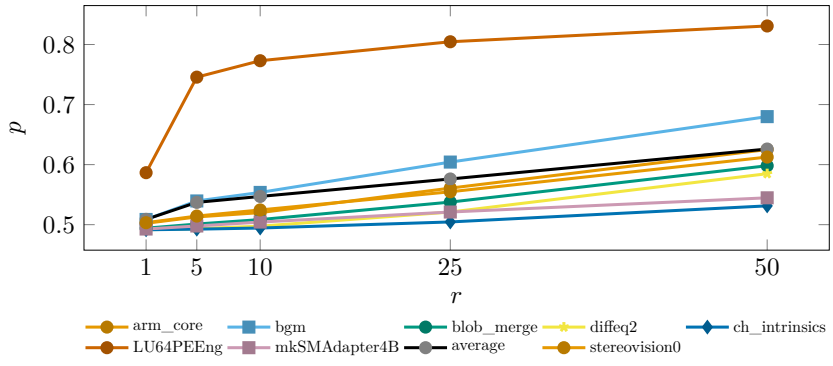
Figure 10.17 shows the effect for the RFET technology. Due to the previously explained differences in RFET and SOI models, the difference between off-regions and regions where active logic is placed is larger for RFET technology, which means the placement locations can be easily recognized in the second, third and fourth row. When looking closely, process variation can still be seen in the background of the current delay figures. The control parameters also vary slightly, but due to the large $\delta t_{\text{PD}}/\delta V_{\text{PG}}$, this slight variation is not visible in the figures.

Figure 10.14 shows aggregate values, obtained in the same way as the ones in figure 10.13. There is little change compared to the data without process variation for the RFET technology, as changes in the control voltage are small due to the large slope of the delay models. Results for SOI are more interesting: Here, it can be seen that with process variation, not all regions are in low-power mode and differences between regions sizes and benchmark resource utilization start to show. It can again be seen, that larger benchmarks cause higher relative power consumption for all regions. Also, larger regions lead to

more current consumption for all benchmarks and affect larger benchmarks more.



(a)

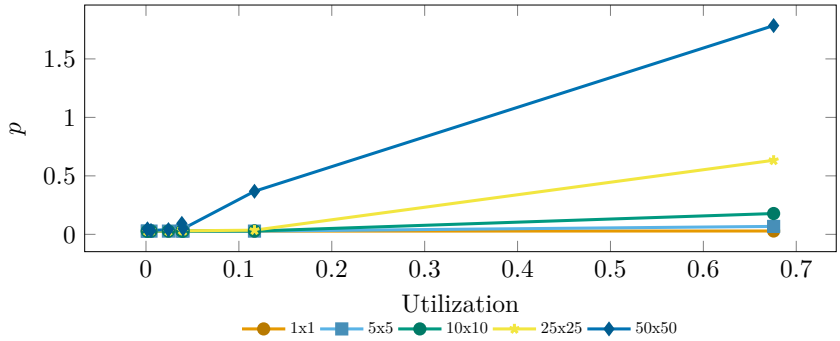


(b)

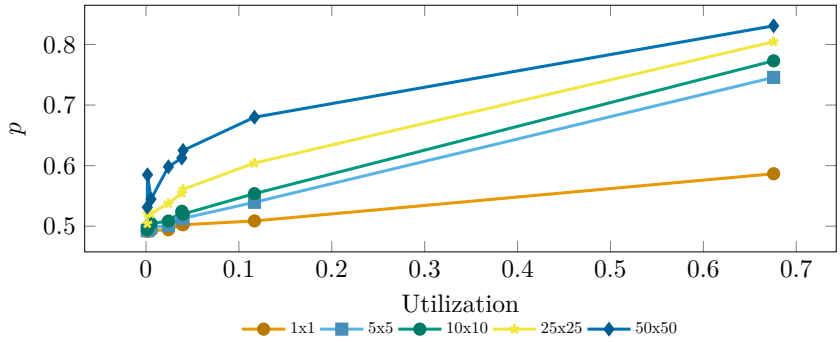
Figure 10.14: Normalized static power p for the benchmarks versus power region size, simulated with process variation. **(a)** SOI model. **(b)** RFET model.

One interesting observation is the largest benchmark requiring more power than nominal power in the 50x50 region case. This can be explained in the following way: Due to the simple proportional control algorithm used for the region controllers, the control voltage for some regions did not settle on a final value yet. The regions therefore are in higher performance mode than necessary and use more power. This is especially severe when large regions are used, as the affected area compared to total area becomes larger. In the 50x50 case, the effect is severe enough that more than nominal power

is required. Even with the simple algorithm, the problem would be solved by simulating more than 100 time steps. If a faster settling time is demanded, a more elaborate control algorithm should be chosen.



(a)



(b)

Figure 10.15: Normalized static power p for the benchmarks versus utilization, simulated with process variation. **(a)** SOI model. **(b)** RFET model.

Figure 10.15 evaluates the same data in a slightly different way: Here, the y-axis still depicts relative power, but over an x-axis depicting logic utilization. The tested utilization range is limited according to the available benchmarks. In this figure, it can be clearly seen how a larger utilization leads to higher power consumption. As expected, there's therefore more potential for power saving when there's less resource utilization. Additionally, it can be seen how larger region sizes reduce the obtainable power reduction. The previously mentioned limitation in the 50x50 test can also be seen here.

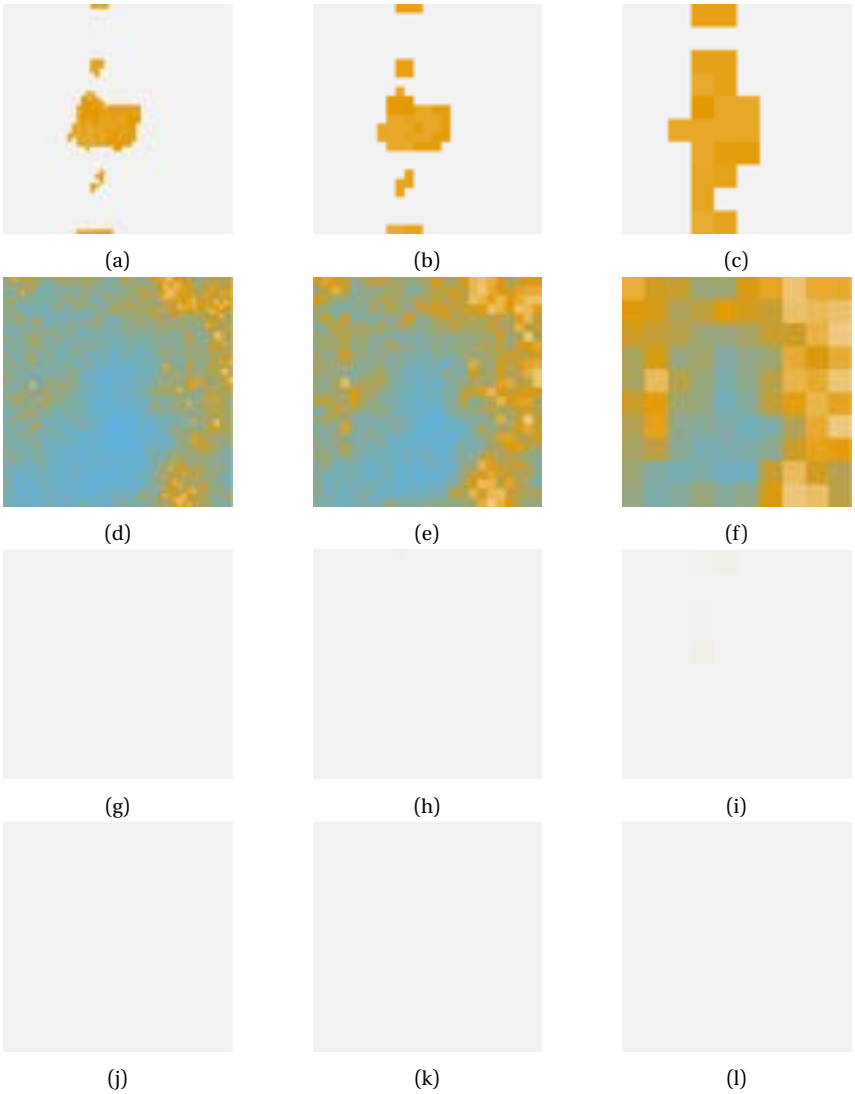


Figure 10.16: Process variation and delay for the `arm_core` benchmark, evaluated using the SOI model. Rows show from top to bottom: Target delay, achieved delay, control voltage and relative power. Columns show different regions sizes. From left to right: 5x5, 10x10 and 25x25.

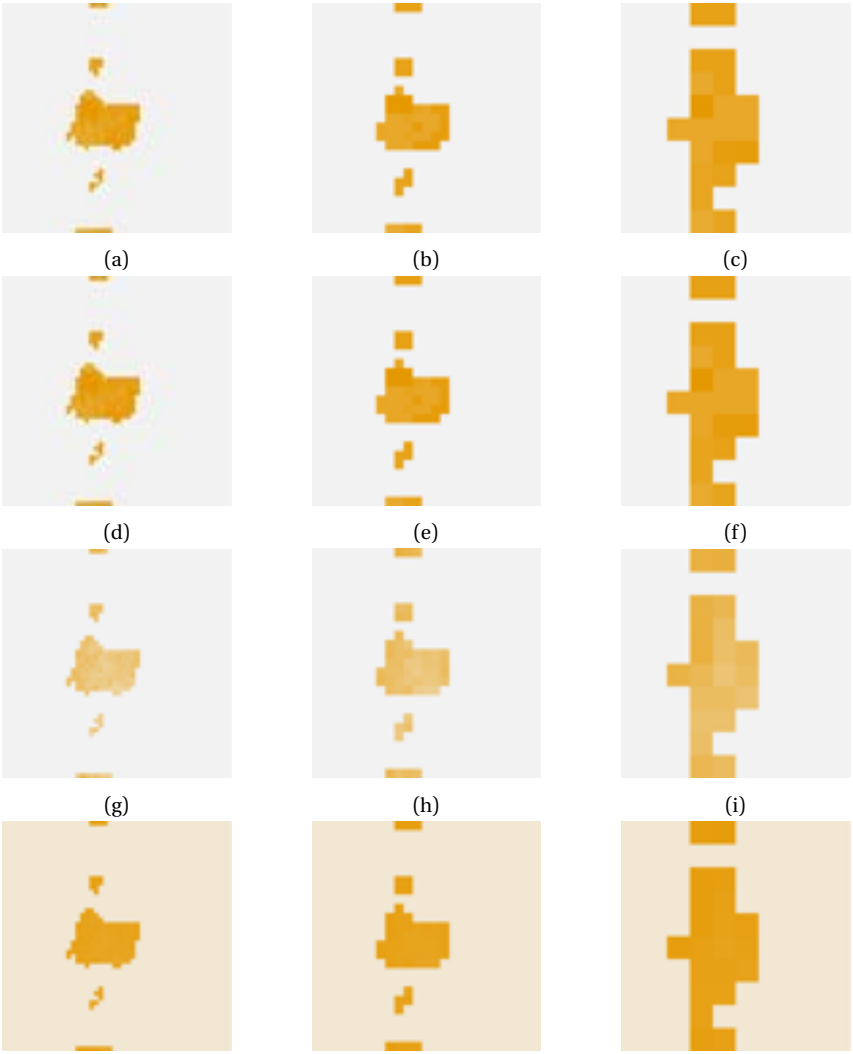


Figure 10.17: Process variation and delay for the `arm_core` benchmark, evaluated using the RFET model. Rows show from top to bottom: Target delay, achieved delay, control voltage and relative power. Columns show different regions sizes. From left to right: 5x5, 10x10 and 25x25.

Voltage Variation Voltage variation has been evaluated similarly, using the voltage variation scenario described in section 4.7. For this evaluation, it has been assumed that the power grid is using 5x5 tiles, independent of the power regions implemented for power management. The figures in figure 10.18 have been derived using this grid and variation scenario based on the local utilization in each region. Examples for three benchmarks are given here, whereas the voltage variation maps for the remaining benchmarks can be found in figure F.18 on page 383. The voltage variation maps shown here are independent of the ϵ value, as it scales all voltage drop values in the same way. Figure 10.19 shows the achieved delays for the `arm_core` benchmark and $\epsilon = 0.1$ in the RFET model. Heatmaps for the SOI model are similar and are therefore not explicitly shown in this chapter. The voltage variation simulations shown here and in the appendix additionally include process variation as explained in the previous section. Graphs for other values of ϵ and for SOI are shown in appendix F.

(a) (b) (c)

Figure 10.18: Voltage variation maps for example applications. **(a)** `arm_core` benchmark. **(b)** `stereovision0` benchmark. **(c)** `LU64PEEng` benchmark.

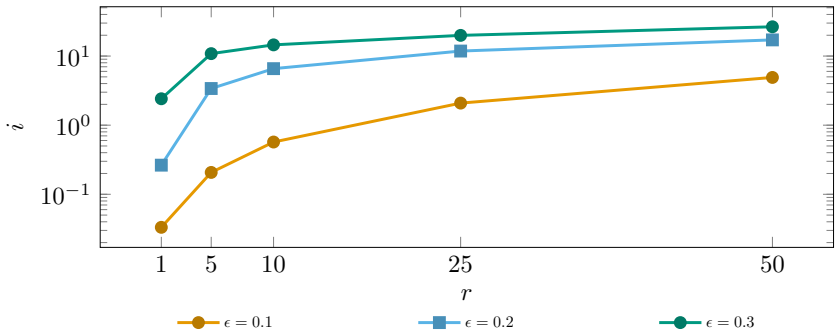
Figure 10.20 shows aggregate statistics for the voltage variation case, similarly to the previously shown aggregate statistics for process variation. Unlike in previous graphs, the graphs here only show relative current instead of relative power: Due to the voltage variations, VDD can no longer assumed to be constant and power can no longer be derived from current using a single factor. The overall trend of larger region causing higher leakage currents can be seen for both technologies. The effect seems to be independent of the ϵ value, i.e. the magnitude of the voltage drops. This can be explained by voltage drops occurring in regions with active logic independently of values of ϵ or region size.

For the SOI model, differences between ϵ values are clearly visible. Because of the large range of current values, the y-axis was plotted logarithmically.

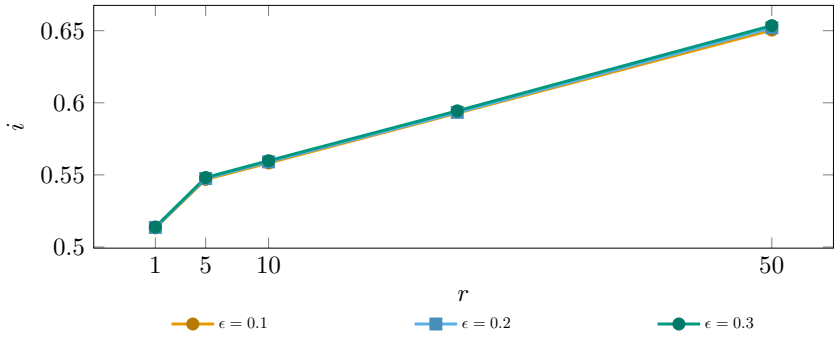
Values below 1 show operation parameters under which the system is still able to reduce current. For larger *epsilon* or larger region sizes, this is not the case and higher currents are accepted when boosting the performance of regions.

The influence of VDD in the RFET model is less significant and results for all ϵ values are similar. Due to the large influence of the program gate control voltage on the propagation delay, the compensation system is able to compensate completely. Because of this, the effect of different ϵ values on the total leakage current is also limited.

Figure 10.19: Voltage variation compensation for the `arm_core` benchmark, evaluated using the RFET model and $\epsilon = 0.1$. Rows show from top to bottom: Achieved delay, control voltage and relative power. Columns show different regions sizes. From left to right: 5x5, 10x10 and 25x25.



(a)



(b)

Figure 10.20: Normalized leakage current i for the benchmarks versus power region size and voltage variation. **(a)** SOI model. **(b)** RFET model.

Temperature Variation Temperature variation results are given in figure 10.21, showing again achieved delay, control voltage and relative power for one exemplary benchmark. Additionally, aggregated statistics in figure 10.23 show achieved static power reduction with the hotspot simulations. The local hotspot model introduced in section 4.7 used for the evaluation is visualized in figure 10.22. It can be seen that the hotspot appears over time, the evaluation figures however show the results at time step 100, when the final value has been settled. Simulation over time is supported in the co-simulator, but the data has not been evaluated here.

Figures 10.23a and 10.23b show little effect of the local hotspot. Those graphs closely resemble the average graphs for the process variation evaluation,

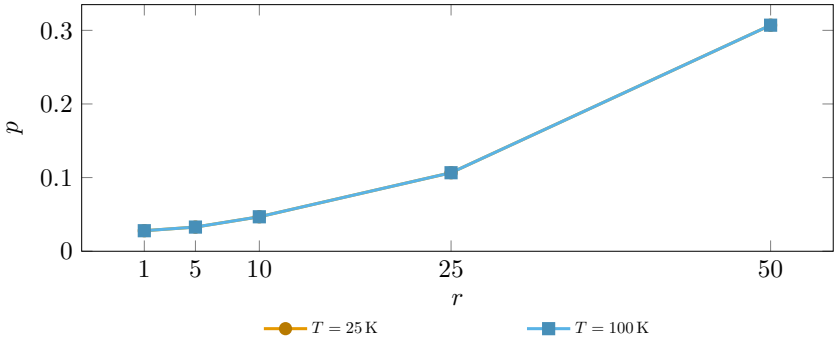
which is expected as the temperature hotspot scenario was simulated with process variation. The power usage is independent of the hotspot for both technologies. In the SOI model, the hotspot leads to slightly increased local delay. The effect is compensated by the PVTA scheme, but due to the small number of affected regions, the leakage power reduction does not change significantly.

In the RFET model, higher temperature leads to reduced delays. This effect allows reducing the control parameter locally, but this does not lead to visible changes in leakage power either.

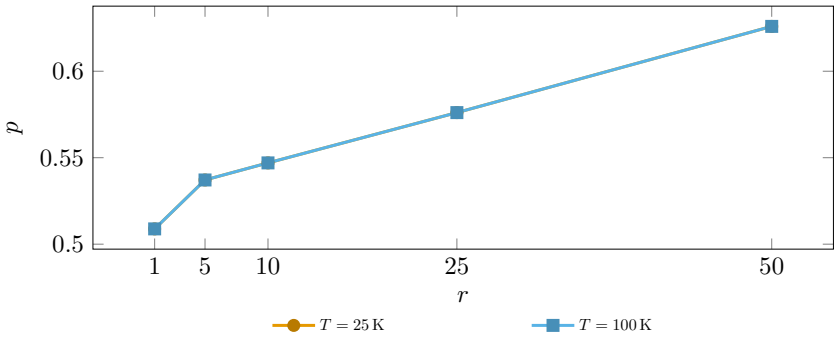
Figure 10.21: Temperature variation compensation for the `arm_core` benchmark, evaluated using the RFET model and $T = 100$. Rows show from top to bottom: Achieved delay, control voltage and relative power. Columns show different regions sizes. From left to right: 5x5, 10x10 and 25x25.

(a) (b) (c) (d)

Figure 10.22: Local hotspot simulation for various points of time in the simulation. **(a)** $t = 1$. **(b)** $t = 5$. **(c)** $t = 10$. **(d)** $t = 20$.



(a)



(b)

Figure 10.23: Normalized static power p for the benchmarks versus power region size and temperature variation. **(a)** SOI model. **(b)** RFET model.

Aging Achieved delay, control voltage and relative power for one exemplary benchmark in the aging scenario are shown in figure 10.24. Figure 10.25 again shows aggregated static power statistics and additional graphs are again available in appendix F.

Aging was modeled according to section 4.7 on page 152 and as all other simulations, was simulated with process variation. For SOI, the parameters used to derive that aging parameter A were chosen as 120 °C and 1.8 V. As was explained section 4.6, the RFET aging model was already matched to produce results matching this aging parameter in the SOI technology.

With aging, it can be seen that there is again no large influence on the RFET model, but the SOI power consumption changes after longer aging. This is again related to the delay sensitivity to control voltage changes in RFET and SOI models. Again, due to the large sensitivity of the current model in SOI, relative power increases up to an order of magnitude. The region size again has a large influence on the additional power used. For RFET, high sensitivity of propagation delay and low sensitivity of current on the control voltage again result in little changes over time.

The RFET technology therefore seems to be relatively stable against any PVTA influences, as they can be compensated more effectively than in SOI technology. On the other hand, power saving benefits in RFET result primarily from completely unutilized regions: Due to small changes in control voltage and little sensitivity of the leakage current to this control voltage, the effects are limited. This can be beneficial, as increasing relative performance is easier, and detrimental, as saving power is more difficult. Whether this different behavior is beneficial therefore depends on the absolute performance reachable in each technology. Such performance results are however largely dependent of the maturity of the technology and could not be evaluated in this thesis. It should also be remembered that the RFET model partially uses data from the SOI model, such as aging and process variation. When production ready manufacturing for RFET is introduced, those parts need to be re-modeled to match the technology. Nevertheless, this initial analysis provided certain insights in the differences of the technologies used and showed viability of a power region based FPGA with PVTA compensation at system level, for both technologies. The introduced co-simulation and the evaluation methodology lay the foundations to evaluate improved models and region controller control strategies in future work.

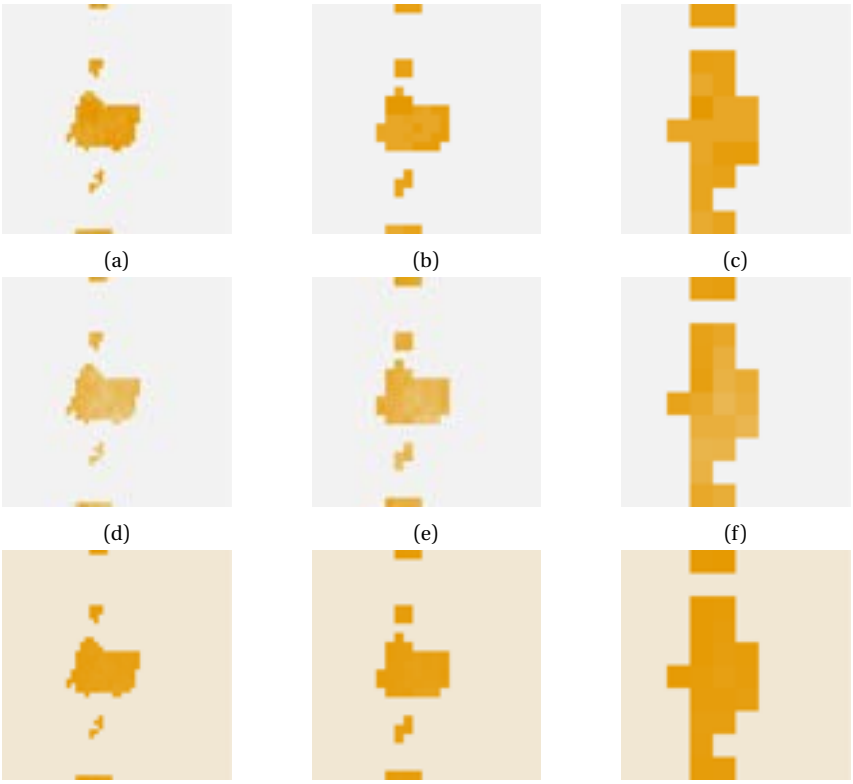
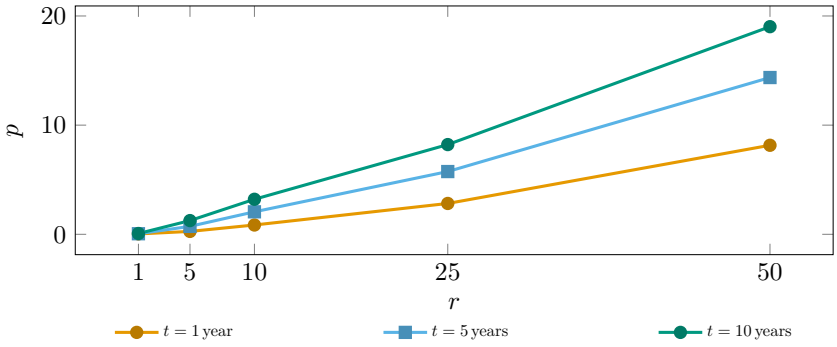
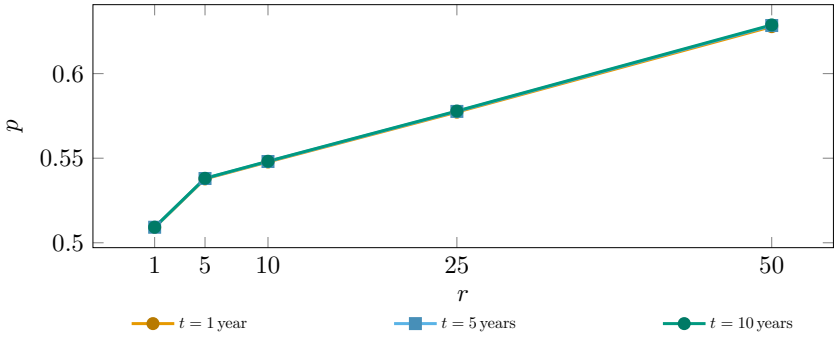


Figure 10.24: Aging compensation for the `arm_core` benchmark, evaluated using the RFET model and $t = 10y$. Rows show from top to bottom: Achieved delay, control voltage and relative power. Columns show different regions sizes. From left to right: 5x5, 10x10 and 25x25.



(a)



(b)

Figure 10.25: Normalized static power p for the benchmarks versus power region size and aging. **(a)** SOI model. **(b)** RFET model.

This page intentionally left blank

Chapter 11

Conclusion and Outlook

The following sections will quickly summarize the topics and results of this thesis. The final section will then discuss aspects that could not be addressed fully in this thesis and should be addressed in future work.

Summary

The overall goal of this thesis was to introduce novel ideas to reduce power consumption in FPGA. Here, the thesis focused largely on static leakage currents, which could be reduced down to 2.76 % in the most extreme cases, when the whole FPGA could be put to low-power mode with the evaluated SOI technology. Other, related topics were addressed in this document as well: For PVTA compensation, simulation models and a co-simulation system were introduced. For example, this simulator has shown that for region sizes up to 10x10, voltage drops of up to 10 % could be compensated in the RFET technology without additional current draw. Similarly, it was shown that small, local temperature hotspots have little effect on overall leakage current with the simulated transistor technologies. It was also shown the process variation can lead to propagation delay variation of up to 30 % in the evaluated technologies. Making use of the fact that FPGA applications are commonly checked against worst case delays in STA, the power reduction to 2.76 % could be realized. It was also shown that for some benchmarks, depending on region size, the improvement can be less. The results therefore can serve as guidance when selecting the region size for such a power-aware FPGA architecture.

The thesis also introduced RFET based logic generators for FPGAs, evaluating the power savings achievable with RFET technology. It introduced a delay and a power model for this technology, that was used in the final evaluation to

estimate the power savings. Furthermore, it was shown that the RFET technology used has a high sensitivity of propagation delay in regard to the program gate voltage, and little sensitivity of leakage current depending on this voltage. The high sensitivity in the propagation delay enables more efficient PVTA compensation compared to the tested SOI technology. On the other hand, low sensitivity in the leakage current limits the achievable power reduction compared to SOI. For example, the final evaluation showed a best-case power reduction to 49.1 %, whereas the SOI model could reduce power to 2.76 %. The results of this thesis suggest that future technology research could focus more on influence of program gate bias on off-currents. On the other hand, it should be noted that the RFET technology already offers absolute values for off currents, that are much smaller than the ones in other technologies [2]. To demonstrate all those benefits, various FPGA aspects have been analyzed with respect to RFET technology.

Simulation Models For the final evaluation, system-level simulation models for RFET were needed. This thesis therefore analyzed a way to derive high level propagation delays when only knowing the device currents, as available in device research papers. It further described how to take this propagation delay information for a single cell to determine delays in a whole FPGA. Those delays were then modeled as relative changes, as absolute values were not obtainable with raw device data. The models were then designed to describe PVTA dependencies and control parameters for program gate voltage scaling. In addition to RFET models based on device measurements, a SOI model was introduced to describe a commercial technology for comparison. For this comparison model, propagation delays could directly be obtained from SPICE simulation. As some PVTA dependencies were not known for the RFET technology, they were transferred from the SOI model to the RFET model, obtaining a plausible, but not real, estimated technology.

Building Circuits In order to build large circuits using RFET technology, the design and use of an RFET standard cell library was investigated. Whereas simulation of individual cells was performed by Reuter et al. [Reu21], the derivation of the standard cell library was part of this thesis. The library was then used for STA of large circuits, focusing on a cryptographic accelerator for the ChaCha cipher. Especially for FPGAs, application of RFETs in reconfigurable logic was evaluated. Whereas small reconfigurable cells are available in literature, they can not directly be used in FPGA due to their limited expressiveness. Because of that, more complex ULMs have been derived based on these reconfigurable cells to replace the LUTs in FPGA. Then, efficient strategies to combine these ULMs in clusters were evaluated. This thesis also

developed tools and methodology to evaluate the expressiveness of these cells and compare them to reference LUTs. In addition, a toolflow that can map arbitrary circuits to such ULM-based FPGA instead of LUT-based FPGA has been introduced.

Building FPGAs To use these RFET ULMs in FPGA, system level changes in the FPGA architecture were evaluated: Power management regions were introduced to enable local trade-offs between performance and power. To map applications to power regions, two approaches to assign region modes have been introduced: Static assignment and dynamic assignment. In static assignment body biasing, or program gate voltage scaling are determined during user application implementation. The dynamic assignment, which is used in the final PARFAIT architecture, allows changing these control voltages at runtime. For this, modifications in the EDA toolflow have been implemented to characterize application paths and derive slack factors for each region. Whereas this gives a metric for the required performance in a region, a mechanism to measure the actual performance in each region is also required. Here, the novel logic invasion scheme allows measuring the propagation delay of all CLBs on an FPGA with little hardware overhead. Logic invasion uses special auto-generated, partial bitstreams, to dynamically reprogram parts of the FPGA with delay measurement logic, i.e. ring oscillators and counters. This invasion occurs transparently, not interfering with the user application. It is performed repeatedly to enable measurement of dynamic effects over time. Target slack factors and the measured delays are then compared in a region controller. This controller can adjust the body bias or program gate voltage in a closed-loop manner, as it receives feedback from the delay measurement system.

Methodology and Evaluation To evaluate possible power savings, the derived current and delay models for SOI and RFET technology were used in simulation. For this simulation, a co-simulation framework has been designed to combine VHDL simulation for the FPGA architecture, Lua models for PVTA influence and VPR for target slack factor estimation. The resulting simulator is fully flexible in regard to the used technology models and the simulation scenarios. Whereas this thesis provided some example evaluations, further evaluations can be realized easily. For example, heating evaluation could consider larger hotspots, faster dynamic effects, multiple hot spots etc. In addition to this simulation, the VFPGA was extended to allow for some hardware simulation of the architecture.

Outlook

This thesis has set the foundations for research into power aware FPGA architectures. In some of the addressed topics, there are however opportunities for further research:

Evaluation results depend heavily on the PVTA models. Whereas the SOI model was derived from mature SPICE models, the RFET model was directly extracted from measurement data. This data is not as mature as final SPICE models, so the RFET model could be improved with better source data. For scenario evaluations, more evaluations are possible. In addition, DSE could be performed for more parameters: In this thesis, it was assumed that control voltages can be set to arbitrary values. Further research could evaluate the effect of discrete power levels and the number of levels required for efficient power saving.

The *RGATE* ULM presented in this work can be further improved as well. Such improvements were shown in detail for the *CNT-DR8F* ULM, but could not be implemented for the *RGATE* one because of time restrictions. The *RGATE* cell has various benefits over *CNT-DR8F*: It is a static logic cell, similar to common CMOS cells, and it is implemented in a technology for which detailed characterization data is available. It also reduces the number of required configuration inputs, which is crucial to reduce the FPGA configuration storage. For example, due to the missing optimization, FPGAs needed three to four times more CLBs to realize the evaluated benchmarks, than compared to the LUT reference architecture. However, the *RGATE* ULM consists of only 5 *RGATE*s with 2 bit configuration each, requiring 10 bit configuration storage in total. The reference LUT on the other hand uses 64 bit of data storage. The *RGATE* FPGA therefore needed less configuration storage, even though it was larger. Increasing CLB numbers however also increase the global interconnect area, which reduces the benefits of smaller storage. Future work therefore should focus on introduction of more logic in CLBs, e.g. by doubling of the number of *RGATE* ULMs in a FLE. For example, instead of using fracturable cells, both outputs could be connected to full logic trees. This would enable doubling of logic in a CLB without affecting the input or output interconnects. Also not considered in this thesis was the topic of dynamic power, evaluation focused largely on leakage current induced power loss.

Nevertheless, this thesis demonstrated the potential of power aware FPGA architectures, more clever FPGA architectures that perform active power management. Application of logic invasion has shown that fine-grain performance measurement in FPGAs is possible with little additional resource

usage. In addition, an interdisciplinary approach has been used to evaluate these power saving on novel RFET technology, even before detailed circuit simulation models are available. The propagation delay models introduced for RFET and SOI in this thesis are thoroughly described and can be easily adapted for other circuit evaluations. Similarly, the developed simulation approach can be used to simulate arbitrary PVTA scenarios. All in all, this thesis demonstrated that RFET based FPGA are feasible and the required changes in toolflow and architecture are limited. RFET technology has been shown to provide better compensation of PVTA effects due to the strong dependency of device on-current on program gate voltage. On the other hand, the comparison SOI technology showed better improvements in leakage power reduction, as the sensitivity of off-current on program gate voltage was limited for RFET. For larger power savings, RFET technology research would have to focus on improving this parameter. After writing of this thesis, improved RFET device measurements with better control of I_{off} have been researched in the PARFAIT project. Updated evaluation results are intended to be published in late 2024. Besides that, the power management and PVTA compensation system was shown to be technology independent, through evaluation in SOI and RFET technologies. Such a power aware FPGA architecture could therefore be implemented in any technology that allows for power and performance trade-offs in some way.

This page intentionally left blank

Bibliography

[1] Krauss, Tillmann A.: "Planare elektrostatisch dotierte rekonfigurierbare Schottky-Barriere FDSOI Feldeffekttransistor Strukturen". Dissertation. Darmstadt: Technische Universität Darmstadt, 2019 (cit. on pp. 3, 14–16, 161).

[2] Galderisi, Giulio; Beyer, Christoph; Mikolajick, Thomas and Trommer, Jens: "Insights into the Temperature Dependent Switching Behaviour of Three–Gated Reconfigurable Field Effect Transistors". In: *physica status solidi (a)* (2023). doi: 10.1002/pssa.202300019 (cit. on pp. 3, 6, 144–149, 152, 153, 227, 228, 241, 272, 288).

[3] Altieri Scarpato, Mauricio: "Digital circuit performance estimation under PVT and aging effects". Thesis. Université Grenoble Alpes, 2017. url: https://theses.hal.science/tel-01773745 (cit. on pp. 6, 12, 23, 32–35, 38–43, 133, 134, 137–140, 147).

[4] Clerc, Sylvain; Di Gilio, Thierry and Cathelin, Andreia, eds.: The Fourth Terminal: Benefits of Body-Biasing Techniques for FDSOI Circuits and Systems. 1st ed. 2020. Integrated Circuits and Systems. Cham: Springer International Publishing and Imprint Springer, 2020 (cit. on p. 11).

[5] Ranica, R.; Planes, N.; Weber, O.; Thomas, O.; Haendler, S.; Noblet, D.; Croain, D.; Gardin, C. and Arnaud, F.: "FDSOI process/design full solutions for ultra low leakage, high speed and low voltage SRAMs". In: *2013 Symposium on VLSI Circuits* (2013) (cit. on p. 12).

[6] Hartmann, Joel: "FD-SOI Technology Development and Key Devices Characteristics for Fast, Power Efficient, Low Voltage SoCs". In: *2014 IEEE Compound Semiconductor Integrated Circuit Symposium (CSICS)*. IEEE, 2014, pp. 1–4. doi: 10.1109/CSICS.2014.6978554 (cit. on p. 12).

[7] Skotnicki, Thomas et al.: "Innovative Materials, Devices, and CMOS Technologies for Low-Power Mobile Multimedia". In: *IEEE Transactions on Electron Devices* 55.1 (2008), pp. 96–130. doi: 10.1109/TED.2007.911338 (cit. on p. 12).

[8] Sakurai, T. and Newton, A. R.: "Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas". In: *IEEE Journal of Solid-State Circuits* 25.2 (1990), pp. 584–594. doi: 10.1109/4.52187 (cit. on pp. 12, 23).

[9] Dasdan, Ali and Hom, Ivan: "Handling inverted temperature dependence in static timing analysis". In: *ACM Transactions on Design Automation of Electronic Systems* 11.2 (2006), pp. 306–324. doi: 10.1145/1142155.1142158 (cit. on pp. 12, 13).

[10] Ghoneim, H.; Knoch, J.; Riel, H.; Webb, D.; Bjork, M. T.; Karg, S.; Lortscher, E.; Schmid, H. and Riess, W.: "Interface engineering for the suppression of ambipolar behavior in Schottky-barrier MOSFETs". In: *2009 10th International Conference on Ultimate Integration of Silicon*. IEEE, 2009, pp. 69–72. DOI: 10.1109/ULIS.2009.4897541 (cit. on p. 14).

[11] Ghoneim, H.; Knoch, J.; Riel, H.; Webb, D.; Björk, M. T.; Karg, S.; Lörtscher, E.; Schmid, H. and Riess, W.: "Suppression of ambipolar behavior in metallic source/drain metal-oxide-semiconductor field-effect transistors". In: *Applied Physics Letters* 95.21 (2009), p. 213504. DOI: 10.1063/1.3266526 (cit. on p. 14).

[12] Reena Monica, P.: "Seven Strategies to Suppress the Ambipolar Behaviour in CNTFETs: a Review". In: *Silicon* 14.16 (2022), pp. 10199–10216. DOI: 10.1007/s12633-022-01813-5 (cit. on p. 14).

[13] Lin, Y.-M.; Appenzeller, J.; Knoch, J. and Avouris, P.: "High-Performance Carbon Nanotube Field-Effect Transistor With Tunable Polarities". In: *IEEE Transactions on Nanotechnology* 4.5 (2005), pp. 481–489. DOI: 10.1109/TNANO.2005.851427 (cit. on pp. 14, 17).

[14] Krauss, Tillmann; Wessely, Frank and Schwalke, Udo: "Fabrication and simulation of electrically reconfigurable dual metal-gate planar field-effect transistors for dopant-free CMOS". In: *2017 12th International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS)*. IEEE, 2017, pp. 1–6. DOI: 10.1109/DTIS.2017.7930155 (cit. on p. 15).

[15] Krauss, Tillmann; Wessely, Frank and Schwalke, Udo: "Electrostatically Doped Planar Field-Effect Transistor for High Temperature Applications". In: *ECS Journal of Solid State Science and Technology* 4.5 (2015), Q46–Q50. DOI: 10.1149/2.0021507jss (cit. on pp. 15, 17).

[16] Hueting, Raymond J. E. and Gupta, Gaurav: "Electrostatic Doping and Devices". In: *Springer Handbook of Semiconductor Devices*. Ed. by Rudan, Massimo; Brunetti, Rossella and Reggiani, Susanna. Springer Handbooks. Cham: Springer International Publishing, 2023, pp. 371–389. DOI: 10.1007/978-3-030-79827-7_11 (cit. on p. 15).

[17] Carter, R. et al.: "22nm FDSOI technology for emerging mobile, Internet-of-Things, and RF applications". In: *2016 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2016, pp. 2.2.1–2.2.4. DOI: 10.1109/IEDM.2016.7838029 (cit. on pp. 15, 17).

[18] Mikolajick, T.; Heinzig, A.; Trommer, J.; Baldauf, T. and Weber, W. M.: "The RFET—a reconfigurable nanowire transistor and its application to novel electronic circuits and systems". In: *Semiconductor Science and Technology* 32.4 (2017), p. 043001. DOI: 10.1088/1361-6641/aa5581 (cit. on pp. 15, 17, 53, 54).

[19] Krauss, Tillmann; Wessely, Frank and Schwalke, Udo: "Reconfigurable electrostatically doped 2.5-gate planar field-effect transistors for dopant-free CMOS". In: *2018 13th International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS)*. IEEE, 2018, pp. 1–4. DOI: 10.1109/DTIS.2018.8368567 (cit. on p. 16).

[20] Weber, W. M.; Heinzig, A.; Trommer, J.; Martin, D.; Grube, M. and Mikolajick, T.: "Reconfigurable nanowire electronics – A review". In: *Solid-State Electronics* 102 (2014), pp. 12–24. doi: 10.1016/j.sse.2014.06.010 (cit. on p. 16).

[21] Fei, Wenwen; Trommer, Jens; Lemme, Max Christian; Mikolajick, Thomas and Heinzig, André: "Emerging reconfigurable electronic devices based on two–dimensional materials: A review". In: *InfoMat* 4.10 (2022). doi: 10.1002/inf2.12355 (cit. on p. 16).

[22] Heinzig, André; Slesazeck, Stefan; Kreupl, Franz; Mikolajick, Thomas and Weber, Walter M.: "Reconfigurable silicon nanowire transistors". In: *Nano letters* 12.1 (2012), pp. 119–124. doi: 10.1021/nl203094h (cit. on p. 17).

[23] Wessely, Frank; Krauss, Tillmann and Schwalke, Udo: "Reconfigurable CMOS with undoped silicon nanowire midgap Schottky-barrier FETs". In: *Microelectronics Journal* 44.12 (2013), pp. 1072–1076. doi: 10.1016/j.mejo.2012.08.004 (cit. on p. 17).

[24] Koo, Sang-Mo; Li, Qiliang; Edelstein, Monica D.; Richter, Curt A. and Vogel, Eric M.: "Enhanced channel modulation in dual-gated silicon nanowire transistors". In: *Nano letters* 5.12 (2005), pp. 2519–2523. doi: 10.1021/nl051855i (cit. on p. 17).

[25] Colli, Alan; Tahraoui, Abbes; Fasoli, Andrea; Kivioja, Jani M.; Milne, William I. and Ferrari, Andrea C.: "Top-gated silicon nanowire transistors in a single fabrication step". In: *ACS nano* 3.6 (2009), pp. 1587–1593. doi: 10.1021/nn900284b (cit. on p. 17).

[26] Trommer, Jens; Heinzig, Andre; Baldauf, Tim; Slesazeck, Stefan; Mikolajick, Thomas and Weber, Walter M.: "Functionality-Enhanced Logic Gate Design Enabled by Symmetrical Reconfigurable Silicon Nanowire Transistors". In: *IEEE Transactions on Nanotechnology* 14.4 (2015), pp. 689–698. doi: 10.1109/TNANO.2015.2429893 (cit. on pp. 17, 54, 55).

[27] Trommer, J.; Heinzig, A.; Slesazeck, S.; Muhle, U.; Loffler, M.; Walter, D.; Mayr, C.; Mikolajick, T. and Weber, W. M.: "Reconfigurable germanium transistors with low source-drain leakage for secure and energy-efficient doping-free complementary circuits". In: *2017 75th Annual Device Research Conference (DRC)*. IEEE, 2017, pp. 1–2. doi: 10.1109/DRC.2017.7999426 (cit. on p. 17).

[28] Trommer, Jens et al.: "Enabling Energy Efficiency and Polarity Control in Germanium Nanowire Transistors by Individually Gated Nanojunctions". In: *ACS nano* 11.2 (2017), pp. 1704–1711. doi: 10.1021/acsnano.6b07531 (cit. on p. 17).

[29] Nakaharai, Shu; Iijima, Tomohiko; Ogawa, Shinich; Suzuki, Shingo; Tsukagoshi, Kazuhito; Sato, Shintaro and Yokoyama, Naoki: "Electrostatically-reversible polarity of dual-gated graphene transistors with He ion irradiated channel: Toward reconfigurable CMOS applications". In: *2012 International Electron Devices Meeting*. IEEE, 2012, pp. 4.2.1–4.2.4. doi: 10.1109/IEDM.2012.6478976 (cit. on p. 17).

[30] Nakaharai, Shu; Yamamoto, Mahito; Ueno, Keiji; Lin, Yen-Fu; Li, Song-Lin and Tsukagoshi, Kazuhito: "Electrostatically Reversible Polarity of Ambipolar

α-MoTe2 Transistors". In: *ACS nano* 9.6 (2015), pp. 5976–5983. DOI: 10.1021/acsnano.5b00736 (cit. on p. 17).

[31] LARENTIS, Stefano; FALLAHAZAD, Babak; MOVVA, Hema C. P.; KIM, Kyounghwan; RAI, Amritesh; TANIGUCHI, Takashi; WATANABE, Kenji; BANERJEE, Sanjay K. and TUTUC, Emanuel: "Reconfigurable Complementary Monolayer MoTe2 Field-Effect Transistors for Integrated Circuits". In: *ACS nano* 11.5 (2017), pp. 4832–4839. DOI: 10.1021/acsnano.7b01306 (cit. on p. 17).

[32] RESTA, Giovanni V.; BALAJI, Yashwanth; LIN, Dennis; RADU, Iuliana P.; CATTHOOR, Francky; GAILLARDON, Pierre-Emmanuel and MICHELI, Giovanni de: "Doping-free complementary inverter enabled by 2D WSe2 electrostatically-doped reconfigurable transistors". In: *2018 76th Device Research Conference (DRC)*. IEEE, 2018, pp. 1–2. DOI: 10.1109/DRC.2018.8442152 (cit. on p. 17).

[33] PANG, Chin-Sheng and CHEN, Zhihong: "First Demonstration of WSe2 CMOS Inverter with Modulable Noise Margin by Electrostatic Doping". In: *2018 76th Device Research Conference (DRC)*. IEEE, 2018, pp. 1–2. DOI: 10.1109/DRC.2018.8442258 (cit. on p. 17).

[34] WU, Peng; AMEEN, Tarek; ZHANG, Huairuo; BENDERSKY, Leonid A.; ILATIKHAMENEH, Hesameddin; KLIMECK, Gerhard; RAHMAN, Rajib; DAVYDOV, Albert V. and APPENZELLER, Joerg: "Complementary Black Phosphorus Tunneling Field-Effect Transistors". In: *ACS nano* 13.1 (2019), pp. 377–385. DOI: 10.1021/acsnano.8b06441 (cit. on p. 17).

[35] KOLODINSKI, S. et al.: "IPCEI subcontracts contributing to 22-FDX Add-On Functionalities at GF". In: *ESSDERC 2019 - 49th European Solid-State Device Research Conference (ESSDERC)*. IEEE, 2019, pp. 74–77. DOI: 10.1109/ESSDERC.2019.8901736 (cit. on p. 17).

[36] SIMON, Maik; MULAOSMANOVIC, Halid; SESSI, Violetta; DRESCHER, Maximilian; BHATTACHARJEE, Niladri; SLESAZECK, Stefan; WIATR, Maciej; MIKOLAJICK, Thomas and TROMMER, Jens: "Three-to-one analog signal modulation with a single back-bias-controlled reconfigurable transistor". In: *Nature communications* 13.1 (2022), p. 7042. DOI: 10.1038/s41467-022-34533-w (cit. on p. 17).

[37] ZHANG, Jian; TANG, Xifan; GAILLARDON, Pierre-Emmanuel and MICHELI, Giovanni de: "Configurable Circuits Featuring Dual-Threshold-Voltage Design With Three-Independent-Gate Silicon Nanowire FETs". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 61.10 (2014), pp. 2851–2861. DOI: 10.1109/TCSI.2014.2333675 (cit. on pp. 17, 18).

[38] GAILLARDON, Pierre-Emmanuel; BEIGNE, Edith; LESECQ, Suzanne and MICHELI, Giovanni de: "A Survey on Low-Power Techniques with Emerging Technologies". In: *ACM Journal on Emerging Technologies in Computing Systems* 12.2 (2015), pp. 1–26. DOI: 10.1145/2714566 (cit. on pp. 17, 18).

[39] MARCHI, M. de; SACCHETTO, D.; FRACHE, S.; ZHANG, J.; GAILLARDON, P.-E.; LEBLEBICI, Y. and MICHELI, G. de: "Polarity control in double-gate, gate-all-around vertically stacked silicon nanowire FETs". In: *2012 International Elec-*

tron Devices Meeting. IEEE, 2012, pp. 8.4.1–8.4.4. DOI: 10.1109/IEDM.2012.6479004 (cit. on p. 17).

[40] WESSELY, F.; KRAUSS, T. and SCHWALKE, U.: "Virtually dopant-free CMOS: Midgap Schottky-barrier nanowire field-effect-transistors for high temperature applications". In: *Solid-State Electronics* 74 (2012), pp. 91–96. DOI: 10.1016/j.sse.2012.04.017 (cit. on p. 17).

[41] NI, Wangze; ZHANG, Yichi; HUANG, Bairun and CHEN, Zhuojun: "The Impact of Temperature on Reconfigurable Field-Effect Transistor and Its Applications". In: *2021 9th International Symposium on Next Generation Electronics (ISNE)*. IEEE, 2021, pp. 1–4. DOI: 10.1109/ISNE48910.2021.9493616 (cit. on p. 17).

[42] GALDERISI, Giulio; MIKOLAJICK, Thomas and TROMMER, Jens: "Robust Reconfigurable Field Effect Transistors Process Route Enabling Multi-V T Devices Fabrication for Hardware Security Applications". In: *2022 Device Research Conference (DRC)*. IEEE, 2022, pp. 1–2. DOI: 10.1109/DRC55272.2022.9855805 (cit. on pp. 18, 144, 145, 147, 227).

[43] ZHANG, Jian; MARCHI, Michele de; SACCHETTO, Davide; GAILLARDON, Pierre-Emmanuel; LEBLEBICI, Yusuf and MICHELI, Giovanni de: "Polarity-Controllable Silicon Nanowire Transistors With Dual Threshold Voltages". In: *IEEE Transactions on Electron Devices* 61.11 (2014), pp. 3654–3660. DOI: 10.1109/TED.2014.2359112 (cit. on p. 18).

[44] BAKER, Russel Jacob: CMOS circuit design, layout, and simulation. Fourth edition. Vol. 22. IEEE Press series on microelectronic systems. Piscataway, NJ and Hoboken, New Jersey: IEEE Press and Wiley, 2019 (cit. on p. 20).

[45] WESTE, Neil H. E. and HARRIS, David Money: CMOS VLSI design: A circuits and systems perspective. 4. ed. Boston, Mass.: Addison-Wesley, 2011 (cit. on pp. 22, 23).

[46] KAHNG, Andrew B.; LIENIG, Jens; MARKOV, Igor L. and HU, Jin: VLSI Physical Design: From Graph Partitioning to Timing Closure. Cham: Springer International Publishing, 2022. DOI: 10.1007/978-3-030-96415-3 (cit. on pp. 24–26, 65, 66).

[47] HUFF, Michael: "Review—Important Considerations Regarding Device Parameter Process Variations in Semiconductor-Based Manufacturing". In: *ECS Journal of Solid State Science and Technology* 10.6 (2021), p. 064002. DOI: 10.1149/2162-8777/ac02a4 (cit. on pp. 28, 30).

[48] MAY, Gary S. and SPANOS, Costas J.: Fundamentals of semiconductor manufacturing and process control. Hoboken, NJ: Wiley-Interscience, 2006 (cit. on p. 28).

[49] VAN ZANT, Peter: Microchip fabrication: A practical guide to semiconductor processing. 6. ed. New York, NY: McGraw-Hill, 2014 (cit. on p. 28).

[50] GENG, Hwaiyu: Semiconductor Manufacturing Handbook, Second Edition. 2nd edition. New York, N.Y.: McGraw-Hill Education and McGraw Hill, 2017 (cit. on p. 28).

[51] BLAAUW, D.; CHOPRA, K.; SRIVASTAVA, A. and SCHEFFER, L.: "Statistical Timing Analysis: From Basic Principles to State of the Art". In: *IEEE Transactions on*

Computer-Aided Design of Integrated Circuits and Systems 27.4 (2008), pp. 589–607. DOI: 10.1109/TCAD.2007.907047 (cit. on pp. 29, 85).

[52] Champac, Victor and Garcia Gervacio, Jose: Timing Performance of Nanometer Digital Circuits Under Process Variations. Vol. 39. Cham: Springer International Publishing, 2018. DOI: 10.1007/978-3-319-75465-9 (cit. on pp. 29–31).

[53] Sarangi, Smruti R.; Greskamp, Brian; Teodorescu, Radu; Nakano, Jun; Tiwari, Abhishek and Torrellas, Josep: "VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects". In: *IEEE Transactions on Semiconductor Manufacturing* 21.1 (2008), pp. 3–13. DOI: 10.1109/TSM.2007.913186 (cit. on pp. 31, 153).

[54] Agarwal, A.; Blaauw, D. and Zolotov, V.: "Statistical timing analysis for intra-die process variations with spatial correlations". In: *ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No.03CH37486)*. IEEE, 2003, pp. 900–907. DOI: 10.1109/ICCAD.2003.159781 (cit. on p. 31).

[55] Agarwal, A.; Blaauw, D.; Zolotov, V.; Sundareswaran, S.; Zhao, Min; Gala, K. and Panda, R.: "Statistical delay computation considering spatial correlations". In: *Proceedings of the ASP-DAC Asia and South Pacific Design Automation Conference, 2003*. IEEE, 2003, pp. 271–276. DOI: 10.1109/ASPDAC.2003.1195028 (cit. on p. 31).

[56] Keshavarzi, Ali et al.: "Measurements and modeling of intrinsic fluctuations in MOSFET threshold voltage". In: *Proceedings of the 2005 international symposium on Low power electronics and design - ISLPED '05*. Ed. by Roy, Kaushik and Tiwari, Vivek. New York, New York, USA: ACM Press, 2005, p. 26. DOI: 10.1145/1077603.1077611 (cit. on pp. 31, 32, 153).

[57] Friedberg, P.; Cao, Yu; Cain, J.; Wang, Ruth; Rabaey, Jan and Spanos, C.: "Modeling Within-Die Spatial Correlation Effects for Process-Design Co-Optimization". In: *Sixth International Symposium on Quality of Electronic Design (ISQED'05)*. IEEE, 2005, pp. 516–521. DOI: 10.1109/ISQED.2005.82 (cit. on p. 31).

[58] Pang, Liang-Teck and Nikolic, Borivoje: "Measurements and Analysis of Process Variability in 90 nm CMOS". In: *IEEE Journal of Solid-State Circuits* 44.5 (2009), pp. 1655–1663. DOI: 10.1109/JSSC.2009.2015789 (cit. on p. 31).

[59] Chang, Hongliang and Sapatnekar, S. S.: "Statistical timing analysis under spatial correlations". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24.9 (2005), pp. 1467–1482. DOI: 10.1109/TCAD.2005.850834 (cit. on p. 33).

[60] Ghasemzadeh Mohammadi, Hassan; Gaillardon, Pierre-Emmanuel and Micheli, Giovanni de: "Efficient Statistical Parameter Selection for Nonlinear Modeling of Process/Performance Variation". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35.12 (2016), pp. 1995–2007. DOI: 10.1109/TCAD.2016.2547908 (cit. on pp. 33, 153).

[61] Lange, Andre; Sohrmann, Christoph; Jancke, Roland; Haase, Joachim; Cheng, Binjie; Asenov, Asen and Schlichtmann, Ulf: "Multivariate Modeling

of Variability Supporting Non-Gaussian and Correlated Parameters". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35.2 (2016), pp. 197–210. DOI: 10.1109/TCAD.2015.2459042 (cit. on pp. 33, 153).

[62] LANGE, Andre; SOHRMANN, Christoph; JANCKE, Roland; HAASE, Joachim; LORENZ, Ingolf and SCHLICHTMANN, Ulf: "Probabilistic standard cell modeling considering non-Gaussian parameters and correlations". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014*. New Jersey: IEEE Conference Publications, 2014, pp. 1–4. DOI: 10.7873/DATE.2014.243 (cit. on p. 33).

[63] KISHORE, Sajja Krishna; PATNALA, Tulasi Radhika; TIGADI, Arun S. and JAMSHED, Aatif: "An On-chip Analysis of the VLSI designs under Process Variations". In: *2020 International Conference on Smart Electronics and Communication (ICOSEC)*. IEEE, 2020, pp. 1273–1277. DOI: 10.1109/ICOSEC49089.2020.9215244 (cit. on p. 33).

[64] SHAH, Nivana; HD, Nataraj Urs; GADHAWE, Akanksha and SAXENA, Ankit: "Study of the Impact of Variations on Standard Cells". In: *Indian Journal of Science and Technology* 12.36 (2019), pp. 1–5. DOI: 10.17485/ijst/2019/v12i36/147751 (cit. on p. 33).

[65] JIN, Leilei; FU, Wenjie; YAN, Hao and SHI, Longxing: "A Statistical Cell Delay Model for Estimating the 3ς Delay by Matching Kurtosis". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 69.6 (2022), pp. 2932–2936. DOI: 10.1109/TCSII.2022.3157981 (cit. on p. 33).

[66] WIRNSHOFER, Martin: Variation-aware adaptive voltage scaling for digital CMOS circuits. Vol. 41. Springer series in advanced microelectronics. Dordrecht: Springer, 2013. DOI: 10.1007/978-94-007-6196-4 (cit. on pp. 34, 35).

[67] BORKAR, Shekhar; KARNIK, Tanay; NARENDRA, Siva; TSCHANZ, Jim; KESHAVARZI, Ali and DE, Vivek: "Parameter variations and impact on circuits and microarchitecture". In: *Proceedings of the 40th annual Design Automation Conference*. Ed. by GETREU, Ian; FIX, Limor and LAVAGNO, Luciano. New York, NY, USA: ACM, 2003, pp. 338–342. DOI: 10.1145/775832.775920 (cit. on pp. 34, 36, 38).

[68] WONG, K. L.; RAHAL-ARABI, T.; MA, M. and TAYLOR, G.: "Enhancing Microprocessor Immunity to Power Supply Noise With Clock-Data Compensation". In: *IEEE Journal of Solid-State Circuits* 41.4 (2006), pp. 749–758. DOI: 10.1109/JSSC.2006.870925 (cit. on p. 34).

[69] GNAD, Dennis R. E.; OBORIL, Fabian; KIAMEHR, Saman and TAHOORI, Mehdi B.: "An Experimental Evaluation and Analysis of Transient Voltage Fluctuations in FPGAs". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.10 (2018), pp. 1817–1830. DOI: 10.1109/TVLSI.2018.2848460 (cit. on pp. 34, 90, 104).

[70] KRAUTTER, Jonas: "Analysis and Mitigation of Remote Side-Channel and Fault Attacks on the Electrical Level". PhD thesis. 2022. DOI: 10.5445/IR/1000144660 (cit. on p. 35).

[71] Larsson, Patrik: "di/dt Noise in CMOS Integrated Circuits". In: *Analog Design Issues in Digital VLSI Circuits and Systems*. Ed. by Becerra, Juan J. and Friedman, Eby G. Boston, MA: Springer US, 1997, pp. 113–129. doi: 10.1007/978-1-4615-6101-9_10 (cit. on p. 35).

[72] Gupta, Meeta S.; Oatley, Jarod L.; Joseph, Russ; Wei, Gu-Yeon and Brooks, David M.: "Understanding Voltage Variations in Chip Multiprocessors using a Distributed Power-Delivery Network". In: *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6. doi: 10.1109/DATE.2007.364663 (cit. on p. 35).

[73] Kiamehr, Saman; Ebrahimi, Mojtaba; Golanbari, Mohammad Saber and Tahoori, Mehdi B.: "Temperature-Aware Dynamic Voltage Scaling to Improve Energy Efficiency of Near-Threshold Computing". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.7 (2017), pp. 2017–2026. doi: 10.1109/TVLSI.2017.2669375 (cit. on p. 36).

[74] Mondal, S.; Mukherjee, R. and Memik, S. O.: "Fine-Grain Thermal Profiling and Sensor Insertion for FPGAs". In: *2006 IEEE International Symposium on Circuits and Systems*. IEEE, 2006, pp. 4387–4390. doi: 10.1109/ISCAS.2006.1693601 (cit. on p. 36).

[75] Sundararajan, Priya; Gayasen, Aman; Vijaykrishnan, N. and Tuan, T.: "Thermal characterization and optimization in platform FPGAs". In: *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design - ICCAD '06*. Ed. by Hassoun, Soha. New York, New York, USA: ACM Press, 2006, p. 443. doi: 10.1145/1233501.1233589 (cit. on p. 36).

[76] Amouri, Abdulazim; Amrouch, Hussam; Ebi, Thomas; Henkel, Jorg and Tahoori, Mehdi: "Accurate Thermal-Profile Estimation and Validation for FPGA-Mapped Circuits". In: *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2013, pp. 57–60. doi: 10.1109/FCCM.2013.48 (cit. on pp. 36, 37).

[77] Amouri, Abdulazim; Hepp, Jochen and Tahoori, Mehdi: "Built-In Self-Heating Thermal Testing of FPGAs". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35.9 (2016), pp. 1546–1556. doi: 10.1109/TCAD.2015.2512905 (cit. on p. 36).

[78] Amrouch, Hussam; Ebi, Thomas; Schneider, Josef; Parameswaran, Sridevan and Henkel, Jorg: "Analyzing the thermal hotspots in FPGA-based embedded systems". In: *2013 23rd International Conference on Field programmable Logic and Applications*. IEEE, 2013, pp. 1–4. doi: 10.1109/FPL.2013.6645567 (cit. on p. 36).

[79] Amouri, Abdulazim: "Degradation in FPGAs: Monitoring, Modeling and Mitigation". PhD thesis. 2015. doi: 10.5445/IR/1000051435 (cit. on p. 36).

[80] Lu, Weina; Hu, Yu; Ye, Jing and Li, Xiaowei: "TeSHoP: A Temperature Sensing based Hotspot-Driven Placement technique for FPGAs". In: *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016, pp. 1–4. doi: 10.1109/FPL.2016.7577304 (cit. on p. 37).

[81] Soleimani, S.; Afzali-Kusha, A. and Forouzandeh, B.: "Temperature dependence of propagation delay characteristic in FinFET circuits". In: *2008 International Conference on Microelectronics*. IEEE, 2008, pp. 276–279. doi: 10.1109/ICM.2008.5393513 (cit. on pp. 37, 38).

[82] Kumar, R. and Kursun, V.: "Impact of temperature fluctuations on circuit characteristics in 180nm and 65nm CMOS technologies". In: *2006 IEEE International Symposium on Circuits and Systems*. IEEE, 2006, p. 4. doi: 10.1109/ISCAS.2006.1693470 (cit. on p. 38).

[83] Kumar, R. and Kursun, V.: "Reversed Temperature-Dependent Propagation Delay Characteristics in Nanometer CMOS Circuits". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 53.10 (2006), pp. 1078–1082. doi: 10.1109/TCSII.2006.882218 (cit. on p. 38).

[84] Lange, André; Gonzalez, Fabio A. Velarde; Giering, Kay-Uwe; Vervantidis, Anastasios; Hahne, Lukas; Heinig, Andy and Jancke, Roland: "A general approach for degradation modeling to enable a widespread use of aging simulations in IC design". In: *Microelectronics Reliability* 137 (2022), p. 114775. doi: 10.1016/j.microrel.2022.114775 (cit. on pp. 38, 41).

[85] Wang, Xiaofei; Tang, Qianying; Jain, Pulkit; Jiao, Dong and Kim, Chris H.: "The Dependence of BTI and HCI-Induced Frequency Degradation on Interconnect Length and Its Circuit Level Implications". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23.2 (2015), pp. 280–291. doi: 10.1109/TVLSI.2014.2307589 (cit. on p. 39).

[86] Tyaginov, Stanislav; Jech, Markus; Franco, Jacopo; Sharma, Prateek; Kaczer, Ben and Grasser, Tibor: "Understanding and Modeling the Temperature Behavior of Hot-Carrier Degradation in SiON nMOSFETs". In: *IEEE Electron Device Letters* 37.1 (2016), pp. 84–87. doi: 10.1109/LED.2015.2503920 (cit. on p. 39).

[87] Khoshavi, Navid; Ashraf, Rizwan A.; DeMara, Ronald F.; Kiamehr, Saman; Oboril, Fabian and Tahoori, Mehdi B.: "Contemporary CMOS aging mitigation techniques: Survey, taxonomy, and methods". In: *Integration* 59 (2017), pp. 10–22. doi: 10.1016/j.vlsi.2017.03.013 (cit. on pp. 39, 41, 42, 91).

[88] Velamala, Jyothi Bhaskarr; Sutaria, Ketul; Sato, Takashi and Cao, Yu: "Physics matters: Statistical aging prediction under trapping/detrapping". In: *Proceedings of the 49th Annual Design Automation Conference*. Ed. by Groeneveld, Patrick; Sciuto, Donatella and Hassoun, Soha. New York, NY, USA: ACM, 2012, pp. 139–144. doi: 10.1145/2228360.2228388 (cit. on p. 39).

[89] Guo, Xinfei; Burleson, Wayne and Stan, Mircea: "Modeling and Experimental Demonstration of Accelerated Self-Healing Techniques". In: *Proceedings of the 51st Annual Design Automation Conference*. New York, NY, USA: ACM, 2014, pp. 1–6. doi: 10.1145/2593069.2593162 (cit. on p. 39).

[90] Ye, Wei; Alawieh, Mohamed Baker; Hsu, Che-Lun; Lin, Yibo and Pan, David Z.: "Dealing with Aging and Yield in Scaled Technologies". In: *Dependable Embedded Systems*. Ed. by Henkel, Jörg and Dutt, Nikil. Embedded Systems.

Cham: Springer International Publishing, 2021, pp. 409–429. DOI: 10.1007/978-3-030-52017-5_17 (cit. on p. 40).

[91] Arnaud, Lucile; Tartavel, G.; Berger, T.; Mariolle, D.; Gobil, Y. and Touet, I.: "Microstructure and electromigration in copper damascene lines". In: *Microelectronics Reliability* 40.1 (2000), pp. 77–86. DOI: 10.1016/S0026-2714(99)00209-7 (cit. on p. 40).

[92] Lange, Andre: Aging Models: The Basis For Predicting Circuit Reliability. 2018. URL: https://semiengineering.com/aging-models-the-basis-for-predicting-circuit-reliability/ (visited on 04/25/2023) (cit. on p. 41).

[93] Open Model Interface Provides Standard for Advanced SPICE Capabilities. URL: https://si2.org/open-model/ (visited on 04/25/2023) (cit. on p. 41).

[94] Naphade, T.; Goel, N.; Nair, P. R. and Mahapatra, S.: "Investigation of stochastic implementation of reaction diffusion (RD) models for NBTI related interface trap generation". In: *2013 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, 2013, XT.5.1–XT.5.11. DOI: 10.1109/IRPS.2013.6532120 (cit. on p. 41).

[95] Kaczer, B.; Grasser, T.; Roussel, Ph. J.; Franco, J.; Degraeve, R.; Ragnarsson, L.-A.; Simoen, E.; Groeseneken, G. and Reisinger, H.: "Origin of NBTI variability in deeply scaled pFETs". In: *2010 IEEE International Reliability Physics Symposium*. IEEE, 2010, pp. 26–32. DOI: 10.1109/IRPS.2010.5488856 (cit. on p. 41).

[96] Nunes, Cícero; Butzen, Paulo F.; Reis, André I. and Ribas, Renato P.: "BTI, HCI and TDDB aging impact in flip–flops". In: *Microelectronics Reliability* 53.9-11 (2013), pp. 1355–1359. DOI: 10.1016/j.microrel.2013.07.044 (cit. on p. 41).

[97] Jafari, Atousa; Raji, Mohsen and Ghavami, Behnam: "Impacts of Process Variations and Aging on Lifetime Reliability of Flip-Flops: A Comparative Analysis". In: *IEEE Transactions on Device and Materials Reliability* 19.3 (2019), pp. 551–562. DOI: 10.1109/TDMR.2019.2933998 (cit. on p. 41).

[98] Lorenz, Dominik; Barke, Martin and Schlichtmann, Ulf: "Aging analysis at gate and macro cell level". In: *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2010, pp. 77–84. DOI: 10.1109/ICCAD.2010.5654309 (cit. on p. 41).

[99] Lu, Yinghai; Shang, Li; Zhou, Hai; Zhu, Hengliang; Yang, Fan and Zeng, Xuan: "Statistical reliability analysis under process variation and aging effects". In: *Proceedings of the 46th Annual Design Automation Conference*. New York, NY, USA: ACM, 2009, pp. 514–519. DOI: 10.1145/1629911.1630044 (cit. on p. 41).

[100] Kiamehr, Saman; Weckx, Pieter; Tahoori, Mehdi; Kaczer, Ben; Kukner, Halil; Raghavan, Praveen; Groeseneken, Guido and Catthoor, Francky: "The impact of process variation and stochastic aging in nanoscale VLSI". In: *2016 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, 2016, CR-1-1-CR-1–6. DOI: 10.1109/IRPS.2016.7574590 (cit. on p. 41).

[101] Oboril, Fabian and Tahoori, Mehdi B.: "ExtraTime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level". In: *IEEE/IFIP*

International Conference on Dependable Systems and Networks (DSN 2012). IEEE, 2012, pp. 1–12. DOI: 10.1109/DSN.2012.6263957 (cit. on p. 42).

[102] Firouzi, Farshad; Kiamehr, Saman; Tahoori, Mehdi and Nassif, Sani: "Incorporating the Impacts of Workload-Dependent Runtime Variations into Timing Analysis". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*. New Jersey: IEEE Conference Publications, 2013, pp. 1022–1025. DOI: 10.7873/DATE.2013.213 (cit. on p. 42).

[103] van Santen, Victor M.; Amrouch, Hussam; Martin-Martinez, Javier; Nafria, Montserrat and Henkel, Jörg: "Designing guardbands for instantaneous aging effects". In: *Proceedings of the 53rd Annual Design Automation Conference*. New York, NY, USA: ACM, 2016, pp. 1–6. DOI: 10.1145/2897937.2898006 (cit. on p. 42).

[104] Brown, S. and Rose, J.: "FPGA and CPLD architectures: a tutorial". In: *IEEE Design & Test of Computers* 13.2 (1996), pp. 42–57. DOI: 10.1109/54.500200 (cit. on pp. 45, 46, 50, 51).

[105] Parandeh-Afshar, Hadi; Benbihi, Hind; Novo, David and Ienne, Paolo: "Rethinking FPGAs: elude the flexibility excess of LUTs with and-inverter cones". In: *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays - FPGA '12*. Ed. by Compton, Katherine and Hutchings, Brad. New York, New York, USA: ACM Press, 2012, p. 119. DOI: 10.1145/2145694.2145715 (cit. on pp. 46, 47, 75, 104).

[106] Zgheib, Grace; Yang, Liqun; Huang, Zhihong; Novo, David; Parandeh-Afshar, Hadi; Yang, Haigang and Ienne, Paolo: "Revisiting and-inverter cones". In: *Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays*. Ed. by Betz, Vaughn and Constantinides, George A. New York, NY, USA: ACM, 2014, pp. 45–54. DOI: 10.1145/2554688.2554791 (cit. on p. 47).

[107] Thummler, Martin; Rai, Shubham and Kumar, Akash: "Improving Technology Mapping for And-Inverter-Cones". In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 274–279. DOI: 10.23919/DATE54114.2022.9774544 (cit. on p. 47).

[108] Rai, Shubham; Nath, Pallab; Rupani, Ansh; Vishvakarma, Santosh Kumar and Kumar, Akash: "A Survey of FPGA Logic Cell Designs in the Light of Emerging Technologies". In: *IEEE Access* 9 (2021), pp. 91564–91574. DOI: 10.1109/ACCESS.2021.3092167 (cit. on pp. 47, 55).

[109] Zilic, Z. and Vranesic, Z. G.: "Using BDDs to Design ULMs for FPGAs". In: *Fourth International ACM Symposium on Field-Programmable Gate Arrays*. IEEE, 1996, pp. 24–30. DOI: 10.1109/FPGA.1996.242252 (cit. on pp. 47, 48).

[110] Preparata, Franco P. and Muller, David E.: "Generation of near-optimal universal Boolean functions". In: *Journal of Computer and System Sciences* 4.2 (1970), pp. 93–102. DOI: 10.1016/S0022-0000(70)80002-2 (cit. on p. 47).

[111] IIDA, Masahiro; AMAGASAKI, Motoki; OKAMOTO, Yasuhiro; ZHAO, Qian and SUEYOSHI, Toshinori: "COGRE: A Novel Compact Logic Cell Architecture for

Area Minimization". In: *IEICE Transactions on Information and Systems* E95-D.2 (2012), pp. 294–302. DOI: 10.1587/transinf.E95.D.294 (cit. on p. 48).

[112] THAKUR, S. and WONG, D. F.: "On Designing ULM-Based FPGA Logic Modules". In: (1995), pp. 3–9. DOI: 10.1109/FPGA.1995.241856 (cit. on p. 48).

[113] HUTTER, M.: "Designing universal logic modules". In: *9th International Conference on Electronics, Circuits and Systems*. IEEE, 2002, pp. 709–712. DOI: 10.1109/ICECS.2002.1046267 (cit. on p. 48).

[114] MICROSEMI: Axcelerator Family FPGAs. Revision 18. 2012. (Visited on 06/28/2023) (cit. on pp. 49, 50).

[115] KUON, Ian; TESSIER, Russell and ROSE, Jonathan: "FPGA Architecture: Survey and Challenges". In: *Foundations and Trends® in Electronic Design Automation* 2.2 (2007), pp. 135–253. DOI: 10.1561/1000000005 (cit. on p. 49).

[116] YANG, Haigang; ZHANG, Jia; SUN, Jiabin and LE YU: "Review of advanced FPGA architectures and technologies". In: *Journal of Electronics (China)* 31.5 (2014), pp. 371–393. DOI: 10.1007/s11767-014-4090-x (cit. on pp. 50, 51).

[117] CHIN, Stephen Alexander; LUU, Jason; HUDA, Safeen and ANDERSON, Jason H.: "Hybrid LUT/Multiplexer FPGA Logic Architectures". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.4 (2016), pp. 1280–1292. DOI: 10.1109/TVLSI.2015.2451658 (cit. on p. 50).

[118] DEHON, André and HAUCK, Scott: Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation. 1. Aufl. Systems on Silicon. s.l.: Elsevier professional, 2007 (cit. on pp. 50, 51, 57, 58, 64–66).

[119] CHIASSON, Charles and BETZ, Vaughn: "COFFE: Fully-automated transistor sizing for FPGAs". In: *2013 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2013, pp. 34–41. DOI: 10.1109/FPT.2013.6718327 (cit. on p. 51).

[120] AMANO, Hideharu, ed.: Principles and structures of FPGAs. Singapore: Springer, 2018 (cit. on pp. 51, 58–60).

[121] RODRÍGUEZ-ANDINA, Juan José; LA TORRE-ARNANZ, Eduardo de and VALDÉS PEÑA, María Dolores: FPGAs: Fundamentals, advanced features, and applications in industrial electronics. First issued in paperback. Boca Raton: CRC Press, 2020 (cit. on p. 51).

[122] CHENG, Kevin; LE BEUX, Sebastien and O'CONNOR, Ian: "Hybrid Topologies for Reconfigurable Matrices Based on Nano-Grain Cells". In: *2017 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2017, pp. 1–8. DOI: 10.1109/ICRC.2017.8123639 (cit. on pp. 53, 55).

[123] JABEUR, Kotb; YAKYMETS, Natalya; O'CONNOR, Ian and LE-BEUX, Sébastien: "Fine-grain reconfigurable logic cells based on double-gate CNTFETs". In: *Proceedings of the 21st edition of the great lakes symposium on Great lakes symposium on VLSI*. Ed. by ATIENZA, David; XIE, Yuan; AYALA, Jose L. and STEVENS, Ken. New York, NY: ACM, 2011, p. 19. DOI: 10.1145/1973009.1973014 (cit. on pp. 53–55).

[124] CHENG, Kevin; LE BEUX, Sebastien and O'CONNOR, Ian: "Am/IDG-FET based reconfigurable cells versus LUTs: Characteristics description and analysis".

In: *2013 25th International Conference on Microelectronics (ICM)*. IEEE, 2013, pp. 1–4. DOI: 10.1109/ICM.2013.6734987 (cit. on pp. 53, 55).

[125] GAILLARDON, Pierre-Emmanuel; AMARÙ, Luca Gaetano; BOBBA, Shashikanth; MARCHI, Michele de; SACCHETTO, Davide and MICHELI, Giovanni de: "Nanowire systems: technology and design". In: *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences* 372.2012 (2014), p. 20130102. DOI: 10.1098/rsta.2013.0102 (cit. on pp. 53, 55).

[126] RAI, Shubham; TROMMER, Jens; RAITZA, Michael; MIKOLAJICK, Thomas; WEBER, Walter M. and KUMAR, Akash: "Designing Efficient Circuits Based on Runtime-Reconfigurable Field-Effect Transistors". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.3 (2019), pp. 560–572. DOI: 10.1109/TVLSI.2018.2884646 (cit. on pp. 54, 71).

[127] O'CONNOR, Ian et al.: "CNTFET Modeling and Reconfigurable Logic-Circuit Design". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 54.11 (2007), pp. 2365–2379. DOI: 10.1109/TCSI.2007.907835 (cit. on p. 54).

[128] LIU, J.; O'CONNOR, I.; NAVARRO, D. and GAFFIOT, F.: "Design of a Novel CNTFET-based Reconfigurable Logic Gate". In: *IEEE Computer Society Annual Symposium on VLSI (ISVLSI '07)*. IEEE, 2007, pp. 285–290. DOI: 10.1109/ISVLSI.2007.39 (cit. on pp. 54, 177–179).

[129] KATO, Junki; WATANABE, Shigeyoshi; NINOMIYA, Hiroshi; KOBAYASHI, Manabu and MIURA, Yasuyuki: "Circuit design of reconfigurable dynamic logic based on double gate CNTFETs focusing on number of states of back gate voltages". In: *Contemporary Engineering Sciences* 7 (2014), pp. 39–52. DOI: 10.12988/ces.2014.3952 (cit. on p. 54).

[130] KUMAR, T. Nandha; ALMURIB, Haider A. F. and LOMBARDI, Fabrizio: "A novel design of a memristor-based look-up table (LUT) for FPGA". In: *2014 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. IEEE, 2014, pp. 703–706. DOI: 10.1109/APCCAS.2014.7032878 (cit. on p. 55).

[131] GUO, Yanwen; WANG, Xiaoping and ZENG, Zhigang: "A Compact Memristor-CMOS Hybrid Look-Up-Table Design and Potential Application in FPGA". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.12 (2017), pp. 2144–2148. DOI: 10.1109/TCAD.2017.2681079 (cit. on p. 55).

[132] NINOMIYA, Hiroshi; KOBAYASHI, Manabu and WATANABE, Shigeyoshi: "Reduced Reconfigurable Logic Circuit Design Based on Double Gate CNTFETs Using Ambipolar Binary Decision Diagram". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E96.A.1 (2013), pp. 356–359. DOI: 10.1587/transfun.E96.A.356 (cit. on p. 55).

[133] YAKYMETS, N.; JABEUR, K.; O'CONNOR, I. and LE BEUX, S.: "Interconnect topology for cell matrices based on low-power nanoscale devices". In: *2011 Faible Tension Faible Consommation (FTFC)*. IEEE, 2011, pp. 99–102. DOI: 10.1109/FTFC.2011.5948929 (cit. on p. 55).

[134] BABU, Praveenkumar and PARTHASARATHY, Eswaran: "Reconfigurable FPGA Architectures: A Survey and Applications". In: *Journal of The Institution of*

Engineers (India): Series B 102.1 (2021), pp. 143–156. DOI: 10.1007/s40031-020-00508-y (cit. on pp. 57, 59, 60).

[135] INTEL: Intel® Arria® 10 Core Fabric and General Purpose I/Os Handbook. 2023. URL: https://www.intel.com/content/www/us/en/docs/programmable/683461/ (visited on 07/22/2023) (cit. on p. 58).

[136] ZGHEIB, Grace and IENNE, Paolo: "Evaluating FPGA clusters under wide ranges of design parameters". In: *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8. DOI: 10.23919/FPL.2017.8056826 (cit. on p. 58).

[137] BOUTROS, Andrew and BETZ, Vaughn: "FPGA Architecture: Principles and Progression". In: *IEEE Circuits and Systems Magazine* 21.2 (2021), pp. 4–29. DOI: 10.1109/MCAS.2021.3071607 (cit. on p. 59).

[138] XILINX: UltraScale Architecture Configurable Logic Block: User Guide. 2017. URL: https://docs.xilinx.com/v/u/en-US/ug574-ultrascale-clb (visited on 07/22/2023) (cit. on p. 59).

[139] SMITH, Michael John Sebastian: Application-specific integrated circuits. VLSI systems series. Reading, Mass.: Addison-Wesley, 1997 (cit. on p. 61).

[140] DEL SOZZO, Emanuele; CONFICCONI, Davide; ZENI, Alberto; SALARIS, Mirko; SCIUTO, Donatella and SANTAMBROGIO, Marco D.: "Pushing the Level of Abstraction of Digital System Design: A Survey on How to Program FPGAs". In: *ACM Computing Surveys* 55.5 (2023), pp. 1–48. DOI: 10.1145/3532989 (cit. on p. 63).

[141] LYKE, James C.; CHRISTODOULOU, Christos G.; VERA, G. Alonzo and EDWARDS, Arthur H.: "An Introduction to Reconfigurable Systems". In: *Proceedings of the IEEE* 103.3 (2015), pp. 291–317. DOI: 10.1109/JPROC.2015.2397832 (cit. on p. 63).

[142] LUU, Jason et al.: "VTR 7.0". In: *ACM Transactions on Reconfigurable Technology and Systems* 7.2 (2014), pp. 1–30. DOI: 10.1145/2617593 (cit. on pp. 64, 180).

[143] SHAH, David; HUNG, Eddie; WOLF, Clifford; BAZANSKI, Serge; GISSELQUIST, Dan and MILANOVIC, Miodrag: "Yosys+nextpnr: An Open Source Framework from Verilog to Bitstream for Commercial FPGAs". In: *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 1–4. DOI: 10.1109/FCCM.2019.00010 (cit. on pp. 64, 66).

[144] BRAYTON, Robert and MISHCHENKO, Alan: "ABC: An Academic Industrial-Strength Verification Tool". In: *Computer Aided Verification*. Ed. by HUTCHISON, David et al. Vol. 6174. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 24–40. DOI: 10.1007/978-3-642-14295-6_5 (cit. on p. 64).

[145] SYNOPSYS: Liberty User Guides and Reference Manual Suite. 2017 (cit. on p. 64).

[146] BETZ, Vaughn and ROSE, Jonathan: "VPR: a new packing, placement and routing tool for FPGA research". In: 1304 (1997), pp. 213–222. DOI: 10.1007/3-540-63465-7_226. (Visited on 04/04/2019) (cit. on pp. 65, 66).

[147] Gerez, Sabih H.: Algorithms for VLSI design automation. Chichester and Weinheim: Wiley, 1999. URL: http://www.loc.gov/catdir/bios/wiley042/98039574.html (cit. on p. 66).

[148] F4PGA: FPGA Assembly (FASM) documentation. 2023. URL: https://fasm.readthedocs.io/en/latest/ (visited on 07/23/2023) (cit. on p. 66).

[149] VTR Developers: FPGA Assembly (FASM) Output Support. 2023. URL: https://docs.verilogtorouting.org/en/latest/utils/fasm/ (visited on 07/23/2023) (cit. on p. 67).

[150] SkyWater Technology: SkyWater Open Source PDK. 2020. URL: https://github.com/google/skywater-pdk (visited on 08/01/2023) (cit. on p. 69).

[151] GlobalFoundries: GlobalFoundries GF180MCU Open Source PDK. 2022. URL: https://github.com/google/gf180mcu-pdk (cit. on p. 69).

[152] Stine, James E. et al.: "FreePDK: An Open-Source Variation-Aware Design Kit". In: *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*. IEEE, 2007, pp. 173–174. DOI: 10.1109/MSE.2007.44 (cit. on p. 69).

[153] Bhanushali, Kirti and Davis, W. Rhett: "FreePDK15". In: *Proceedings of the 2015 Symposium on International Symposium on Physical Design*. Ed. by Davoodi, Azadeh and Young, Evangeline. New York, NY, USA: ACM, 2015, pp. 165–170. DOI: 10.1145/2717764.2717782 (cit. on p. 69).

[154] Martins, Mayler; Matos, Jody Maick; Ribas, Renato P.; Reis, André; Schlinker, Guilherme; Rech, Lucio and Michelsen, Jens: "Open Cell Library in 15nm FreePDK Technology". In: *Proceedings of the 2015 Symposium on International Symposium on Physical Design*. Ed. by Davoodi, Azadeh and Young, Evangeline. New York, NY, USA: ACM, 2015, pp. 171–178. DOI: 10.1145/2717764.2717783 (cit. on pp. 70, 71).

[155] Rai, Shubham; Raitza, Michael; Sahoo, Siva Satyendra and Kumar, Akash: "DiSCERN: Distilling Standard-Cells for Emerging Reconfigurable Nanotechnologies". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 674–677. DOI: 10.23919/DATE48585.2020.9116216 (cit. on p. 71).

[156] Krinke, Andreas; Rai, Shubham; Kumar, Akash and Lienig, Jens: "Exploring Physical Synthesis for Circuits based on Emerging Reconfigurable Nanotechnologies". In: *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9. DOI: 10.1109/ICCAD51958.2021.9643439 (cit. on p. 72).

[157] Ben-Jamaa, M. Haykel; Mohanram, Kartik and Micheli, Giovanni de: "An Efficient Gate Library for Ambipolar CNTFET Logic". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30.2 (2011), pp. 242–255. DOI: 10.1109/TCAD.2010.2085250 (cit. on pp. 72, 74).

[158] Rai, Shubham; Rupani, Ansh; Walter, Dennis; Raitza, Michael; Heinzig, Andre; Baldauf, Tim; Trommer, Jens; Mayr, Christian; Weber, Walter M. and Kumar, Akash: "A physical synthesis flow for early technology evaluation of silicon nanowire based reconfigurable FETs". In: *2018 Design, Automation &*

Test in Europe Conference & Exhibition (DATE). IEEE, 2018, pp. 605–608. DOI: 10.23919/DATE.2018.8342080 (cit. on pp. 72, 74).

[159] Gore, Ganesh; Cadareanu, Patsy; Giacomin, Edouard and Gaillardon, Pierre-Emmanuel: "A Predictive Process Design Kit for Three-Independent-Gate Field-Effect Transistors". In: *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2019, pp. 172–177. DOI: 10.1109/VLSI-SoC.2019.8920358 (cit. on pp. 73, 74).

[160] Gauchi, Roman; Snelgrove, Ashton and Gaillardon, Pierre-Emmanuel: "An Open-source Three-Independent-Gate FET Standard Cell Library for Mixed Logic Synthesis". In: *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2022, pp. 273–277. DOI: 10.1109/ISCAS48785.2022.9937590 (cit. on pp. 73, 74).

[161] Keyser, Michael; Gauchi, Roman and Gaillardon, Pierre-Emmanuel: "An Energy-Efficient Three-Independent-Gate FET Cell Library for Low-Power Edge Computing". In: *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2022, pp. 1–6. DOI: 10.1109/VLSI-SoC54400.2022.9939636 (cit. on pp. 73, 74).

[162] Quijada, Jorge Navarro; Baldauf, Tim; Rai, Shubham; Heinzig, Andre; Kumar, Akash; Weber, Walter M.; Mikolajick, Thomas and Trommer, Jens: "A Germanium Nanowire Reconfigurable Transistor Model for Predictive Technology Evaluation". In: *IEEE Transactions on Nanotechnology* (2022), pp. 1–8. DOI: 10.1109/TNANO.2022.3221836 (cit. on pp. 73, 74).

[163] Goncalves, O.; Prenat, G. and Dieny, B.: "Radiation Hardened MRAM-Based FPGA". In: *IEEE Transactions on Magnetics* 49.7 (2013), pp. 4355–4358. DOI: 10.1109/TMAG.2013.2247744 (cit. on pp. 75, 104).

[164] Ben Jamaa, M. Haykel; Gaillardon, Pierre-Emmanuel; Frégonèse, Sebastien; Marchi, Michele de; Micheli, Giovanni de; Zimmer, Thomas; O'Connor, Ian and Clermidy, Fabien: "FPGA Design with Double-Gate Carbon Nanotube Transistors". In: *ECS Transactions* 34.1 (2011), pp. 1005–1010. DOI: 10.1149/1.3567706 (cit. on pp. 75, 104).

[165] Gaillardon, Pierre-Emmanuel; Tang, Xifan; Kim, Gain and Micheli, Giovanni de: "A Novel FPGA Architecture Based on Ultrafine Grain Reconfigurable Logic Cells". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23.10 (2015), pp. 2187–2197. DOI: 10.1109/TVLSI.2014.2359385 (cit. on p. 76).

[166] Gaillardon, Pierre-Emmanuel; Ben-Jamaa, M. Haykel; Clermidy, Fabien and O'Connor, Ian: "Ultra-fine grain FPGAs: A granularity study". In: *2011 IEEE/ACM International Symposium on Nanoscale Architectures*. IEEE, 2011, pp. 9–15. DOI: 10.1109/NANOARCH.2011.5941477 (cit. on pp. 76, 104).

[167] Park, Jun-Mo; Bae, Jong-Ho; Eum, Jai-Ho; Jin, Sung Hun; Park, Byung-Gook and Lee, Jong-Ho: "High-Density Reconfigurable Devices With Programmable Bottom-Gate Array". In: *IEEE Electron Device Letters* 38.5 (2017), pp. 564–567. DOI: 10.1109/LED.2017.2679343 (cit. on pp. 76, 77, 178).

[168] Vipin, Kizheppatt and Fahmy, Suhaib A.: "FPGA Dynamic and Partial Reconfiguration". In: *ACM Computing Surveys* 51.4 (2019), pp. 1–39. doi: 10.1145/3193827 (cit. on pp. 77–79).

[169] Vassiliadis, Stamatis, ed.: Fine- and coarse-grain reconfigurable computing. Dordrecht: Springer, 2007 (cit. on p. 78).

[170] Cardona, Luis Andres and Ferrer, Carles: "AC_ICAP: A Flexible High Speed ICAP Controller". In: *International Journal of Reconfigurable Computing* 2015 (2015), pp. 1–15. doi: 10.1155/2015/314358 (cit. on p. 78).

[171] Steiger, C.; Walder, H. and Platzner, M.: "Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks". In: *IEEE Transactions on Computers* 53.11 (2004), pp. 1393–1407. doi: 10.1109/TC.2004.99 (cit. on pp. 80, 81).

[172] Diessel, Oliver and ElGindy, Hossam: "Run-time compaction of FPGA designs". In: *Field-Programmable Logic and Applications*. Ed. by Goos, Gerhard; Hartmanis, Juris; van Leeuwen, Jan; Luk, Wayne; Cheung, Peter Y. K. and Glesner, Manfred. Vol. 1304. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 131–140. doi: 10.1007/3-540-63465-7_218 (cit. on p. 80).

[173] Walder, H.; Steiger, C. and Platzner, M.: "Fast online task placement on FPGAs: free space partitioning and 2D-hashing". In: *Proceedings International Parallel and Distributed Processing Symposium*. IEEE Comput. Soc, 2003, p. 8. doi: 10.1109/IPDPS.2003.1213329 (cit. on p. 80).

[174] Sidiropoulos, Harry; Figuli, Peter; Siozios, Kostas; Soudris, Dimitrios and Becker, Jurgen: "A platform-independent runtime methodology for mapping multiple applications onto FPGAs through resource virtualization". In: *2013 23rd International Conference on Field programmable Logic and Applications*. IEEE, 2013, pp. 1–4. doi: 10.1109/FPL.2013.6645564 (cit. on p. 80).

[175] Diessel, O.; ElGindy, H.; Middendorf, M.; Schmeck, H. and Schmidt, B.: "Dynamic scheduling of tasks on partially reconfigurable FPGAs". In: *IEE Proceedings - Computers and Digital Techniques* 147.3 (2000), p. 181. doi: 10.1049/ip-cdt:20000485 (cit. on p. 81).

[176] Sterpone, Luca and Bozzoli, Ludovica: "Fast Partial Reconfiguration on SRAM-Based FPGAs: A Frame-Driven Routing Approach". In: *Applied Reconfigurable Computing. Architectures, Tools, and Applications*. Ed. by Voros, Nikolaos; Huebner, Michael; Keramidas, Georgios; Goehringer, Diana; Antonopoulos, Christos and Diniz, Pedro C. Vol. 10824. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 319–330. doi: 10.1007/978-3-319-78890-6_26 (cit. on p. 81).

[177] da Silva, Bruno; Braeken, An and Touhafi, Abdellah: "Probabilistic Performance Modelling when Using Partial Reconfiguration to Accelerate Streaming Applications with Non-deterministic Task Scheduling". In: 11444 (2019), pp. 81–95. doi: 10.1007/978-3-030-17227-5_7 (cit. on p. 81).

[178] Steiger, C.; Walder, H.; Platzner, M. and Thiele, L.: "Online scheduling and placement of real-time tasks to partially reconfigurable devices". In:

Proceedings. 2003 International Symposium on System-on-Chip (IEEE Cat. No.03EX748). IEEE Comput. Soc, 2003, pp. 224–225. DOI: 10.1109/REAL.2003.1253269 (cit. on p. 81).

[179] ISMAIL, Aws and SHANNON, Lesley: "FUSE: Front-End User Framework for O/S Abstraction of Hardware Accelerators". In: *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2011, pp. 170–177. DOI: 10.1109/FCCM.2011.48 (cit. on p. 81).

[180] JANSSEN, Benedikt; WINGENDER, Tim and HÜBNER, Michael: Hardware Accelerator Framework Approach for Dynamic Partial Reconfigurable Overlays on Xilinx PYNQ. 2017. DOI: 10.18420/IN2017_44 (cit. on p. 81).

[181] JANSSEN, Benedikt; KÄSTNER, Florian; WINGENDER, Tim and HUEBNER, Michael: "A Dynamic Partial Reconfigurable Overlay Framework for Python". In: *Applied Reconfigurable Computing. Architectures, Tools, and Applications*. Ed. by VOROS, Nikolaos; HUEBNER, Michael; KERAMIDAS, Georgios; GOEHRINGER, Diana; ANTONOPOULOS, Christos and DINIZ, Pedro C. Vol. 10824. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 331–342. DOI: 10.1007/978-3-319-78890-6_27 (cit. on p. 81).

[182] KORINTH, Jens; HOFMANN, Jaco; HEINZ, Carsten and KOCH, Andreas: "The TaPaSCo Open-Source Toolflow for the Automated Composition of Task-Based Parallel Reconfigurable Computing Systems". In: *Applied Reconfigurable Computing*. Ed. by HOCHBERGER, Christian; NELSON, Brent; KOCH, Andreas; WOODS, Roger and DINIZ, Pedro. Vol. 11444. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 214–229. DOI: 10.1007/978-3-030-17227-5_16 (cit. on p. 81).

[183] TRIMBERGER, S.; CARBERRY, D.; JOHNSON, A. and WONG, J.: A time-multiplexed FPGA. Los Alamitos Calif.: IEEE Computer Society Press, 1997. DOI: 10.1109/FPGA.1997.624601 (cit. on p. 82).

[184] LI, Z.; COMPTON, K. and HAUCK, S.: "Configuration caching management techniques for reconfigurable computing". In: *Proceedings 2000 IEEE Symposium on Field-Programmable Custom Computing Machines (Cat. No.PR00871)*. IEEE Comput. Soc, 2000, pp. 22–36. DOI: 10.1109/FPGA.2000.903390 (cit. on p. 82).

[185] COMPTON, K.; LI, Zhiyuan; COOLEY, J.; KNOL, S. and HAUCK, S.: "Configuration relocation and defragmentation for run-time reconfigurable computing". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 10.3 (2002), pp. 209–220. DOI: 10.1109/TVLSI.2002.1043324 (cit. on pp. 82, 104).

[186] BREBNER, Gordon and DIESSEL, Oliver: "Chip-Based Reconfigurable Task Management". In: *Field-Programmable Logic and Applications*. Ed. by GOOS, Gerhard; HARTMANIS, Juris; VAN LEEUWEN, Jan; BREBNER, Gordon and WOODS, Roger. Vol. 2147. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 182–191. DOI: 10.1007/3-540-44687-7_19 (cit. on pp. 83, 104).

[187] KOCH, D.; AHMADINIA, A.; BOBDA, C. and KALTE, H.: "FPGA architecture extensions for preemptive multitasking and hardware defragmentation". In: *Proceedings. 2004 IEEE International Conference on Field- Programmable Technology*

(IEEE Cat. No.04EX921). IEEE, 2004, pp. 433–436. DOI: 10.1109/FPT.2004.1393318 (cit. on p. 83).

[188] FIGULI, Peter; HUBNER, Michael; GIRARDEY, Romuald; BAPP, Falco; BRUCKSCHLOGL, Thomas; THOMA, Florian; HENKEL, Jorg and BECKER, Jurgen: "A heterogeneous SoC architecture with embedded virtual FPGA cores and runtime Core Fusion". In: *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, 2011, pp. 96–103. DOI: 10.1109/AHS.2011.5963922 (cit. on pp. 83, 84, 104).

[189] BOZZOLI, Ludovica and STERPONE, Luca: "ReM: A Reconfigurable Multipotent Cell for New Distributed Reconfigurable Architectures". In: 11444 (2019), pp. 295–304. DOI: 10.1007/978-3-030-17227-5_21 (cit. on p. 84).

[190] ADETOMI, Adewale; ENEMALI, Godwin and ARSLAN, Tughrul: "Enabling Dynamic Communication for Runtime Circuit Relocation". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.1 (2020), pp. 142–155. DOI: 10.1109/TVLSI.2019.2934927 (cit. on p. 84).

[191] DONGAONKAR, Sourabh; MUDANAI, Sivakumar P. and GILES, Martin D.: "From Process Corners to Statistical Circuit Design Methodology: Opportunities and Challenges". In: *IEEE Transactions on Electron Devices* 66.1 (2019), pp. 19–27. DOI: 10.1109/TED.2018.2860929 (cit. on p. 85).

[192] LIN, Yan; HUTTON, Mike and HE, Lei: "Placement and Timing for FPGAs Considering Variations". In: *2006 International Conference on Field Programmable Logic and Applications*. IEEE, 2006, pp. 1–7. DOI: 10.1109/FPL.2006.311192 (cit. on p. 86).

[193] KAENEL, V. von; MACKEN, P. and DEGRAUWE, M.G.R.: "A voltage reduction technique for battery-operated systems". In: *IEEE Journal of Solid-State Circuits* 25.5 (1990), pp. 1136–1140. DOI: 10.1109/4.62134 (cit. on pp. 87, 104).

[194] CHENG, Lerong; XIONG, Jinjun; HE, Lei and HUTTON, Mike: "FPGA Performance Optimization Via Chipwise Placement Considering Process Variations". In: *2006 International Conference on Field Programmable Logic and Applications*. IEEE, 2006, pp. 1–6. DOI: 10.1109/FPL.2006.311193 (cit. on pp. 87, 104).

[195] GHOSH, Swaroop; BHUNIA, Swarup and ROY, Kaushik: "CRISTA: A New Paradigm for Low-Power, Variation-Tolerant, and Adaptive Circuit Synthesis Using Critical Path Isolation". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26.11 (2007), pp. 1947–1956. DOI: 10.1109/TCAD.2007.896305 (cit. on pp. 87, 104).

[196] EBRAHIMI, Mohammad and NAVABI, Zainalabedin: "Selecting Representative Critical Paths for Sensor Placement Provides Early FPGA Aging Information". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.10 (2020), pp. 2976–2989. DOI: 10.1109/TCAD.2019.2953174 (cit. on p. 88).

[197] ELGEBALY, Mohamed and SACHDEV, Manoj: "Efficient adaptive voltage scaling system through on-chip critical path emulation". In: *Proceedings of the 2004 international symposium on Low power electronics and design*. Ed. by JOSHI,

Rajiv; Choi, Kiyoung; Tiwari, Vivek and Roy, Kaushik. New York, NY, USA: ACM, 2004, pp. 375–380. DOI: 10.1145/1013235.1013325 (cit. on p. 88).

[198] Fojtik, Matthew; Fick, David; Kim, Yejoong; Pinckney, Nathaniel; Harris, David Money; Blaauw, David and Sylvester, Dennis: "Bubble Razor: Eliminating Timing Margins in an ARM Cortex-M3 Processor in 45 nm CMOS Using Architecturally Independent Error Detection and Correction". In: *IEEE Journal of Solid-State Circuits* 48.1 (2013), pp. 66–81. DOI: 10.1109/JSSC.2012.2220912 (cit. on p. 88).

[199] Miro-Panades, Ivan; Beigne, Edith; Billoint, Olivier and Thonnart, Yvain: "In-situ Fmax/Vmin tracking for energy efficiency and reliability optimization". In: *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 96–99. DOI: 10.1109/IOLTS.2017.8046240 (cit. on p. 89).

[200] Chen, Poki; Shie, Mon-Chau; Zheng, Zhi-Yuan; Zheng, Zi-Fan and Chu, Chun-Yan: "A Fully Digital Time-Domain Smart Temperature Sensor Realized With 140 FPGA Logic Elements". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 54.12 (2007), pp. 2661–2668. DOI: 10.1109/TCSI.2007.906073 (cit. on p. 89).

[201] Yu, Haile; Xu, Qiang and Leong, Philip H.W.: "Fine-grained characterization of process variation in FPGAs". In: *2010 International Conference on Field-Programmable Technology*. IEEE, 2010, pp. 138–145. DOI: 10.1109/FPT.2010.5681770 (cit. on pp. 90, 104).

[202] Franco, John J. Leon; Boemo, Eduardo; Castillo, Encarnacion and Parrilla, Luis: "Ring oscillators as thermal sensors in FPGAs: Experiments in low voltage". In: *2010 VI Southern Programmable Logic Conference (SPL)*. IEEE, 2010, pp. 133–137. DOI: 10.1109/SPL.2010.5483027 (cit. on pp. 90, 104).

[203] Happe, Markus; Agne, Andreas and Plessl, Christian: "Measuring and Predicting Temperature Distributions on FPGAs at Run-Time". In: *2011 International Conference on Reconfigurable Computing and FPGAs*. IEEE, 2011, pp. 55–60. DOI: 10.1109/ReConFig.2011.59 (cit. on p. 90).

[204] Agarwal, Mridul; Paul, Bipul C.; Zhang, Ming and Mitra, Subhasish: "Circuit Failure Prediction and Its Application to Transistor Aging". In: *25th IEEE VLSI Test Symmposium (VTS'07)*. IEEE, 2007, pp. 277–286. DOI: 10.1109/VTS.2007.22 (cit. on p. 90).

[205] Huard, V.; Cacho, F.; Giner, F.; Saliva, M.; Benhassain, A.; Patel, D.; Torres, N.; Naudet, S.; Jain, A. and Parthasarathy, C.: "Adaptive Wearout Management with in-situ aging monitors". In: *2014 IEEE International Reliability Physics Symposium*. IEEE, 2014, 6B.4.1–6B.4.11. DOI: 10.1109/IRPS.2014.6861106 (cit. on p. 90).

[206] Sengupta, Deepashree and Sapatnekar, Sachin S.: "Estimating Circuit Aging Due to BTI and HCI Using Ring-Oscillator-Based Sensors". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.10 (2017), pp. 1688–1701. DOI: 10.1109/TCAD.2017.2648840 (cit. on p. 90).

[207] Zick, Kenneth M. and Hayes, John P.: "Low-cost sensing with ring oscillator arrays for healthier reconfigurable systems". In: *ACM Transactions on Reconfigurable Technology and Systems* 5.1 (2012), pp. 1–26. doi: 10.1145/2133352.2133353 (cit. on p. 90).

[208] Alam, M.: "Reliability- and process-variation aware design of integrated circuits". In: *Microelectronics Reliability* 48.8-9 (2008), pp. 1114–1122. doi: 10.1016/j.microrel.2008.07.039 (cit. on p. 91).

[209] Gupta, Meeta S.; Rivers, Jude A.; Bose, Pradip; Wei, Gu-Yeon and Brooks, David: "Tribeca: Design for PVT Variations with Local Recovery and Fine-grained Adaptation". In: *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. Ed. by Albonesi, David; Martonosi, Margaret; August, David and Martínez, José. New York, NY, USA: ACM, 2009, pp. 435–446. doi: 10.1145/1669112.1669168 (cit. on p. 91).

[210] Mittal, Sparsh: "A Survey of Architectural Techniques for Managing Process Variation". In: *ACM Computing Surveys* 48.4 (2016), pp. 1–29. doi: 10.1145/2871167 (cit. on p. 91).

[211] Rahimi, Abbas; Benini, Luca and Gupta, Rajesh K.: "Variability Mitigation in Nanometer CMOS Integrated Systems: A Survey of Techniques From Circuits to Software". In: *Proceedings of the IEEE* 104.7 (2016), pp. 1410–1448. doi: 10.1109/JPROC.2016.2518864 (cit. on p. 91).

[212] Tschanz, J. W.; Kao, J. T.; Narendra, S. G.; Nair, R.; Antoniadis, D. A.; Chandrakasan, A. P. and De, V.: "Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage". In: *IEEE Journal of Solid-State Circuits* 37.11 (2002), pp. 1396–1402. doi: 10.1109/JSSC.2002.803949 (cit. on p. 92).

[213] Teodorescu, Radu; Nakano, Jun; Tiwari, Abhishek and Torrellas, Josep: "Mitigating Parameter Variation with Dynamic Fine-Grain Body Biasing". In: *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 2007, pp. 27–42. doi: 10.1109/MICRO.2007.43 (cit. on p. 92).

[214] Mauricio, Joan and Moll, Francesc: "Local variations compensation with DLL-based Body Bias Generator for UTBB FD-SOI technology". In: *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*. IEEE, 2015, pp. 1–4. doi: 10.1109/NEWCAS.2015.7182005 (cit. on p. 93).

[215] Matsushita, Yusuke; Okuhara, Hayate; Masuyama, Koichiro; Fujita, Yu; Kawano, Ryuta and Amano, Hideharu: "Body bias grain size exploration for a coarse grained reconfigurable accelerator". In: *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016, pp. 1–4. doi: 10.1109/FPL.2016.7577346 (cit. on p. 93).

[216] Chow, C. T.; Tsui, L.S.M.; Leong, P.H.W.; Luk, W. and Wilton, S.J.E.: "Dynamic voltage scaling for commercial FPGAs". In: *Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005.* IEEE, 2005, pp. 173–180. doi: 10.1109/FPT.2005.1568543 (cit. on pp. 94, 99, 104).

[217] Nabaa, Georges; Azizi, Navid and Najm, Farid N.: An adaptive FPGA architecture with process variation compensation and reduced leakage. New York, NY and Piscataway, NJ: Association for Computing Machinery and IEEE Service Center, 2006. doi: 10.1145/1146909.1147069 (cit. on pp. 94, 95, 104).

[218] Hioki, Masakazu; Ma, Chao; Kawanami, Takashi; Ogasahara, Yasuhiro; Nakagawa, Tadashi; Sekigawa, Toshihiro; Tsutsumi, Toshiyuki and Koike, Hanpei: "SOTB Implementation of a Field Programmable Gate Array with Fine-Grained Vt Programmability". In: *Journal of Low Power Electronics and Applications* 4.3 (2014), pp. 188–200. doi: 10.3390/jlpea4030188 (cit. on pp. 95, 104).

[219] Burmester Campos, Pedro: "Variability-Aware Circuit Performance Optimisation Through Digital Reconfiguration". PhD thesis. 2015 (cit. on p. 95).

[220] Maragos, Konstantinos; Lentaris, George and Soudris, Dimitrios: "A PVT-Aware Voltage Scaling Method for Energy Efficient FPGAs". In: *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5. doi: 10.1109/ISCAS51556.2021.9401622 (cit. on pp. 96, 104).

[221] EETimes: Lattice unveils first FPGAs on FDSOI. 2019. url: https://www.eetimes.com/lattice-unveils-first-fpgas-on-fd-soi/ (visited on 08/07/2023) (cit. on p. 96).

[222] Akgun, Gokhan; Ali, Muhammad and Gohringer, Diana: "Power-Aware Computing Systems on FPGAs: A Survey". In: *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2021, pp. 45–51. doi: 10.1109/FPL53798.2021.00016 (cit. on pp. 96, 97).

[223] Sutter, G.; Todorovich, E.; Lopez-Buedo, S. and Boemo, E.: "Low-Power FSMs in FPGA: Encoding Alternatives". In: *Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation*. Ed. by Goos, Gerhard; Hartmanis, Juris; van Leeuwen, Jan; Hochet, Bertrand; Acosta, Antonio J. and Bellido, Manuel J. Vol. 2451. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 363–370. doi: 10.1007/3-540-45716-X_36 (cit. on p. 97).

[224] Singh, Amit; Parthasarathy, Ganapathy and Marek-Sadowska, Malgorzata: "Efficient circuit clustering for area and power reduction in FPGAs". In: *ACM Transactions on Design Automation of Electronic Systems* 7.4 (2002), pp. 643–663. doi: 10.1145/605440.605448 (cit. on p. 97).

[225] Gayasen, A.; Tsai, Y.; Vijaykrishnan, N.; Kandemir, M.; Irwin, M. J. and Tuan, T.: "Reducing leakage energy in FPGAs using region-constrained placement". In: *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*. Ed. by Tessier, Russ and Schmit, Herman. New York, NY, USA: ACM, 2004, pp. 51–58. doi: 10.1145/968280.968289 (cit. on p. 97).

[226] Tuan, Tim; Rahman, Arif; Das, Satyaki; Trimberger, Steve and Kao, Sean: "A 90-nm Low-Power FPGA for Battery-Powered Applications". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26.2 (2007), pp. 296–300. doi: 10.1109/TCAD.2006.885731 (cit. on p. 97).

[227] Bsoul, Assem A. M.; Wilton, Steven J. E.; Tsoi, Kuen Hung and Luk, Wayne: "An FPGA Architecture and CAD Flow Supporting Dynamically Controlled Power Gating". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.1 (2016), pp. 178–191. doi: 10.1109/TVLSI.2015.2393914 (cit. on p. 98).

[228] Bsoul, Assem A. M. and Wilton, Steven J. E.: "An FPGA architecture supporting dynamically controlled power gating". In: *2010 International Conference on Field-Programmable Technology*. IEEE, 2010, pp. 1–8. doi: 10.1109/FPT.2010.5681533 (cit. on p. 98).

[229] Seifoori, Zeinab; Asadi, Hossein and Stojilovic, Mirjana: "A Machine Learning Approach for Power Gating the FPGA Routing Network". In: *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 10–18. doi: 10.1109/ICFPT47387.2019.00010 (cit. on p. 98).

[230] Nabina, Atukem and Nunez-Yanez, Jose Luis: "Adaptive Voltage Scaling in a Dynamically Reconfigurable FPGA-Based Platform". In: *ACM Transactions on Reconfigurable Technology and Systems* 5.4 (2012), pp. 1–22. doi: 10.1145/2392616.2392618 (cit. on p. 99).

[231] Nunez-Yanez, Jose Luis: "Adaptive Voltage Scaling with In-Situ Detectors in Commercial FPGAs". In: *IEEE Transactions on Computers* 64.1 (2015), pp. 45–53. doi: 10.1109/TC.2014.2365963 (cit. on pp. 99, 100).

[232] Ahmed, Ibrahim; Zhao, Shuze; Trescases, Olivier and Betz, Vaughn: "Measure twice and cut once: Robust dynamic voltage scaling for FPGAs". In: *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016, pp. 1–11. doi: 10.1109/FPL.2016.7577342 (cit. on p. 99).

[233] Ahmed, Ibrahim; Zhao, Shuze; Trescases, Olivier and Betz, Vaughn: "Automatic Application-Specific Calibration to Enable Dynamic Voltage Scaling in FPGAs". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.12 (2018), pp. 3095–3108. doi: 10.1109/TCAD.2018.2801222 (cit. on p. 99).

[234] Levine, Joshua M.; Stott, Edward and Cheung, Peter Y.K.: "Dynamic voltage & frequency scaling with online slack measurement". In: *Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays*. Ed. by Betz, Vaughn and Constantinides, George A. New York, NY, USA: ACM, 2014, pp. 65–74. doi: 10.1145/2554688.2554784 (cit. on p. 99).

[235] Zhao, Shuze; Ahmed, Ibrahim; Lamoureux, Carl; Lotfi, Ashraf; Betz, Vaughn and Trescases, Olivier: "A universal self-calibrating Dynamic Voltage and Frequency Scaling (DVFS) scheme with thermal compensation for energy savings in FPGAs". In: *2016 IEEE Applied Power Electronics Conference and Exposition (APEC)*. IEEE, 2016, pp. 1882–1887. doi: 10.1109/APEC.2016.7468125 (cit. on p. 100).

[236] Taka, Endri; Lentaris, George and Soudris, Dimitrios: "Improving the performance of RISC-V softcores on FPGA by exploiting PVT variability and DVFS". In: *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2022, pp. 1595–1599. doi: 10.1109/ISCAS48785.2022.9937320 (cit. on p. 100).

[237] Gayasen, A.; Lee, K.; Vijaykrishnan, N.; Kandemir, M.; Irwin, M. J. and Tuan, T.: "A Dual-VDD Low Power FPGA Architecture". In: *Field Programmable Logic and Application*. Ed. by Hutchison, David et al. Vol. 3203. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 145–157. doi: 10.1007/978-3-540-30117-2_17 (cit. on pp. 100, 200).

[238] Zukoski, Andrew; Yang, Xuebei and Mohanram, Kartik: "Universal logic modules based on double-gate carbon nanotube transistors". In: *Proceedings of the 48th Design Automation Conference*. Ed. by Stok, Leon; Dutt, Nikil and Hassoun, Soha. New York, NY, USA: ACM, 2011, pp. 884–889. doi: 10.1145/2024724.2024921 (cit. on p. 101).

[239] Anderson, Jason H. and Wang, Qiang: "Area-efficient FPGA logic elements: Architecture and synthesis". In: *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*. IEEE, 2011, pp. 369–375. doi: 10.1109/ASPDAC.2011.5722215 (cit. on pp. 101, 102).

[240] Lin, Chih-Chang and Marek-Sadowska, M.: "On designing universal logic blocks and their application to FPGA design". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 16.5 (1997), pp. 519–527. doi: 10.1109/43.631214 (cit. on p. 101).

[241] Cong, Jason; Huang, Hui and Yuan, Xin: "Technology mapping and architecture evalution for k/m -macrocell-based FPGAs". In: *ACM Transactions on Design Automation of Electronic Systems* 10.1 (2005), pp. 3–23. doi: 10.1145/1044111.1044113 (cit. on p. 101).

[242] Hu, Yu; Das, Satyaki; Trimberger, Steve and He, Lei: "Design, Synthesis and Evaluation of Heterogeneous FPGA with Mixed LUTs and Macro-Gates". In: *Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design* (2007), pp. 188–193 (cit. on p. 102).

[243] Luo, Tao; Liang, Hao; Zhang, Wei; He, Bingsheng and Maskell, Douglas: "A Hybrid Logic Block Architecture in FPGA for Holistic Efficiency". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 64.1 (2017), pp. 71–75. doi: 10.1109/TCSII.2016.2551555 (cit. on pp. 102, 103).

[244] VTR Developers: VTR Documentation: FPGA Assembly (FASM) Output Support. 2024. url: https://docs.verilogtorouting.org/en/stable/utils/fasm/ (visited on 02/01/2024) (cit. on p. 127).

[245] VTR Developers: VTR Documentation: Routing Resource Graph File Format. 2024. url: https://docs.verilogtorouting.org/en/stable/vpr/file_formats/#routing-resource-graph-file-format-xml (visited on 02/01/2024) (cit. on p. 131).

[246] Galderisi, G.; Mikolajick, T. and Trommer, J.: "The RGATE: an 8-in-1 Polymorphic Logic Gate Built from Reconfigurable Field Effect Transistors". In: *IEEE Electron Device Letters* (2024). Early Access. doi: 10.1109/LED.2023.3347397 (cit. on pp. 144, 227–229).

[247] Freeley, Jennifer; Mishagli, Dmvtro; Brazil, Tom and Blokhina, Elena: "Statistical Simulations of Delay Propagation in Large Scale Circuits Using Graph Traversal and Kernel Function Decomposition". In: *2018 15th Interna-*

tional Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD). IEEE, 2018, pp. 213–219. DOI: 10.1109/SMACD.2018.8434901 (cit. on p. 153).

[248] MANFREDI, Paolo and TRINCHERO, Riccardo: "A Probabilistic Machine Learning Approach for the Uncertainty Quantification of Electronic Circuits Based on Gaussian Process Regression". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.8 (2022), pp. 2638–2651. DOI: 10.1109/TCAD.2021.3112138 (cit. on p. 153).

[249] TURKYILMAZ, Ogun; CLERMIDY, Fabien; AMARU, Luca Gaetano; GAILLARDON, Pierre-Emmanuel and MICHELI, Giovanni de: "Self-checking ripple-carry adder with Ambipolar Silicon NanoWire FET". In: *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*. IEEE, 2013, pp. 2127–2130. DOI: 10.1109/ISCAS.2013.6572294 (cit. on p. 169).

[250] BERNSTEIN, Daniel J.: ChaCha, a variant of Salsa20. 2008. URL: http://cr.yp.to/chacha/chacha-20080120.pdf (visited on 04/25/2023) (cit. on p. 170).

[251] AHMED, Ibrahim; SHEN, Linda L. and BETZ, Vaughn: "Optimizing FPGA Logic Circuitry for Variable Voltage Supplies". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.4 (2020), pp. 890–903. DOI: 10.1109/TVLSI.2019.2962501 (cit. on p. 197).

[252] GOLOMB, Solomon W.: Shift register sequences: [secure and limited-access code generators ; efficiency code generators ; prescribed property generators ; mathematical models]. Rev. ed. Laguna Hills, Calif.: Aegean Park Pr, 1982 (cit. on p. 221).

[253] XIPHERA LTD.: CHACHA20-POLY1305 PRODUCT BRIEF. 2019. URL: https://xiphera.com/product_brief/ChaCha20_Poly1305_MPSoC.pdf (visited on 04/16/2019) (cit. on p. 259).

[254] AT, Nuray; BEUCHAT, Jean-Luc; OKAMOTO, Eiji; SAN, Ismail and YAMAZAKI, Teppei: "Compact Hardware Implementations of ChaCha, BLAKE, Threefish, and Skein on FPGA". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 61.2 (2014), pp. 485–498. DOI: 10.1109/TCSI.2013.2278385 (cit. on p. 259).

[255] STRÖMBERGSON, Joachim: Verilog 2001 implementation of the ChaCha stream cipher. 2019. URL: https://github.com/secworks/chacha/ (visited on 04/16/2019) (cit. on p. 259).

[256] SILITONGA, Arthur; SCHADE, Florian; JIANG, Guanru and BECKER, Juergen: "HLS-Based Performance and Resource Optimization of Cryptographic Modules". In: *Proceedings of the 16th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA2018), Melbourne, Australia, 11th-13th December 2018*. IEEE, 2018, pp. 1009–1016. DOI: 10.1109/BDCloud.2018.00147 (cit. on p. 259).

[257] SOLTANI, Abolfazl and SHARIFIAN, Saeed: "An ultra-high throughput and fully pipelined implementation of AES algorithm on FPGA". In: *Microprocessors and Microsystems* 39.7 (2015), pp. 480–493. DOI: 10.1016/j.micpro.2015.07.005 (cit. on p. 259).

[258] Yazdanshenas, Sadegh and Betz, Vaughn: "COFFE 2: Automatic Modelling and Optimization of Complex and Heterogeneous FPGA Architectures". In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 12.1 (2019), p. 3. doi: 10.1145/3301298 (cit. on p. 264).

This page intentionally left blank

This page intentionally left blank

Publications

This section contains a complete list of own publications. Publications [Pfa18, Pfa19, Pfa20, Pfa21, Pfa22, Pfa23a, Pfa23b] have been authored by this thesis' author and are listed first. Out of those, [Pfa18] focuses on FPGA-based signal processing and [Pfa23a] on teaching of System-on-Chip (SoC) design. Both are not directly related to the content of this thesis. Those original publications are also available as open access preprints in the KITopen repository. The list continues with publications to which this thesis' author has contributed to. PARFAIT publications [Reu19, Reu20, Reu21, Reu22] contain contributions relevant to this thesis and are again listed first. All remaining publications are not related to the thesis' topic.

[Pfa18] Pfau, Johannes; Figuli, Shalina Percy Delicia; Bähr, Steffen and Becker, Jürgen: "Reconfigurable FPGA-Based Channelization Using Polyphase Filter Banks for Quantum Computing Systems". In: *Applied Reconfigurable Computing. Architectures, Tools, and Applications*. Ed. by Voros, Nikolaos; Huebner, Michael; Keramidas, Georgios; Goehringer, Diana; Antonopoulos, Christos and Diniz, Pedro C. Vol. 10824. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 615–626. doi: 10.1007/978-3-319-78890-6_49.

[Pfa19] Pfau, Johannes; Reuter, Maximilian; Harbaum, Tanja; Hofmann, Klaus and Becker, Jürgen: "A Hardware Perspective on the ChaCha Ciphers: Scalable Chacha8/12/20 Implementations Ranging from 476 Slices to Bitrates of 175 Gbit/s". In: *2019 32nd IEEE International System-on-Chip Conference (SOCC)*. IEEE, 2019, pp. 294–299. doi: 10.1109/socc46988.2019.1570548289 (cit. on pp. 170, 173, 174, 254).

[Pfa20] Pfau, Johannes; Reuter, Maximilian; Hofmann, Klaus and Becker, Jürgen: "Designing Universal Logic Module FPGA Architectures for Use With Ambipolar Transistor Technology". In: *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020, pp. 165–173. doi: 10.1109/icfpt51103.2020.00031 (cit. on pp. 177, 180, 183).

[Pfa21] Pfau, Johannes; Zaki, Peter Wagih and Becker, Jürgen: "Evaluation of Different Manual Placement Strategies to Ensure Uniformity of the V-FPGA". In: *Applied Reconfigurable Computing. Architectures, Tools, and Applications*. Ed. by Derrien, Steven; Hannig, Frank; Diniz, Pedro C. and Chillet, Daniel. Vol. 12700. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 35–49. doi: 10.1007/978-3-030-79025-7_3 (cit. on p. 232).

[Pfa22] Pfau, Johannes; Zaki, Peter Wagih and Becker, Jürgen: "V-FPGAs: Increasing Performance with Manual Placement, Timing Extraction and Extended Timing Modeling". In: *Journal of Signal Processing Systems* 94.9 (2022), pp. 865–882. doi: 10.1007/s11265-022-01786-z (cit. on pp. 232, 236).

[Pfa23a] Pfau, Johannes; Leys, Richard; Neu, Marc; Serdyuk, Alexey; Peric, Ivan and Becker, Jürgen: "A Unified SoC Lab Course: Combined Teaching of Mixed Signal Aspects, System Integration, Software Development and Documentation". In: *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2023, pp. 1–5. doi: 10.1109/ISCAS46773.2023.10181679.

[Pfa23b] Pfau, Johannes; Hernandez, Jiro; Reuter, Maximilian; Hofmann, Klaus and Becker, Jürgen: "Co-Simulating Region-Based Dynamic Voltage Scaling for FPGA Architecture Design". In: *2023 IEEE Nordic Circuits and Systems Conference (NorCAS)*. IEEE, 2023, pp. 1–7. doi: 10.1109/NorCAS58970.2023.10305486 (cit. on pp. 196, 245, 264).

[Reu19] Reuter, Maximilian; Krauss, Tillmann A.; Moradinasab, Mahdi; Pfau, Johannes; Schwalke, Udo; Becker, Jürgen and Hofmann, Klaus: "From MOSFETs to Ambipolar Transistors: A Static DeFET Inverter Cell for SOI". In: *2019 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. IEEE, 2019, pp. 113–116. doi: 10.1109/APCCAS47518.2019.8953083 (cit. on pp. 20, 21, 161).

[Reu20] Reuter, Maximilian; Pfau, Johannes; Krauss, Tillmann A.; Moradinasab, Mahdi; Schwalke, Udo; Becker, Jürgen and Hofmann, Klaus: "Towards Ambipolar Planar Devices: The DeFET Device in Area Constrained XOR Applications". In: *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*. IEEE, 2020, pp. 1–4. doi: 10.1109/lascas45839.2020.9069043 (cit. on pp. 3, 161, 163).

[Reu21] Reuter, Maximilian; Pfau, Johannes; Krauss, Tillmann A.; Becker, Jürgen and Hofmann, Klaus: "From MOSFETs to Ambipolar Transistors: Standard Cell Synthesis for the Planar RFET Technology". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 68.1 (2021), pp. 114–125. doi: 10.1109/TCSI.2020.3035889 (cit. on pp. 3, 133, 134, 161–164, 175, 176, 253, 254, 256, 288).

[Reu22] Reuter, Maximilian; Kramer, Andreas; Krauss, Tillmann; Pfau, Johannes; Becker, Jürgen and Hofmann, Klaus: "Reconfiguring an RFET Based Differential Amplifier". In: *2022 IEEE 40th Central Amer-*

ica and Panama Convention (CONCAPAN). IEEE, 2022, pp. 1–6. DOI: 10.1109/CONCAPAN48024.2022.9997726 (cit. on p. 18).

[Pis16] PISTORIUS, Felix; LAUBER, Andreas; PFAU, Johannes; KLIMM, Alexander and BECKER, Jürgen: "Development of a Latency Optimized Communication Device for WAVE and SAE Based V2X-Applications". In: *SAE Technical Paper Series*. SAE Technical Paper Series. SAE International400 Commonwealth Drive, Warrendale, PA, United States, 2016. DOI: 10.4271/2016-01-0150.

[Nus22] NUSS, Benjamin; GROESCHEL, Patrick; PFAU, Johannes; BECKER, Juergen; VOSSIEK, Martin and ZWICK, Thomas: "Broadband MIMO Testbed for the Development and Research on 6G". In: *European Wireless 2022; 27th European Wireless Conference*. 2022.

[Kar22] KARLE, Christian Maximilian; KREUTZER, Marius; PFAU, Johannes and BECKER, Jürgen: "A hardware/software co-design approach to prototype 6G mobile applications inside the GNU Radio SDR Ecosystem using FPGA hardware accelerators". In: *International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*. New York, NY, USA: ACM, 2022, pp. 33–41. DOI: 10.1145/3535044.3535049.

[Chu23] CHU, Anqi et al.: "LETSCOPE: Lifecycle Extensions Through Software-Defined Predictive Control of Power Electronics". In: *IEEE EUROCON 2023 - 20th International Conference on Smart Technologies*. IEEE, 2023, pp. 665–670. DOI: 10.1109/EUROCON56442.2023.10199076.

[Kar23] KARLE, Christian; NEU, Marc; PFAU, Johannes; SPERLING, Jan and BECKER, Jürgen: "ReLoDAQ: Resource-Efficient, Low-Overhead 200 Gbits^{-1} Data Acquisition System for 6G Prototyping". In: *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2023, p. 209. DOI: 10.1109/FCCM57271.2023.00037.

[Mar23] MARTEN, Johann Christian; YOUNIS, Marwan; KRIEGER, Gerhard; PFAU, Johannes; UNGER, Kai and BECKER, Jürgen: "Design and Implementation of Staggered-SAR Azimuth-Processing". In: *2023 24th International Radar Symposium (IRS)*. IEEE, 2023, pp. 1–10. DOI: 10.23919/IRS57608.2023.10172405.

[Kre23] KRESS, Fabian; PFAU, Johannes; KEMPF, Fabian; SCHMIDT, Patrick; HE, Zhuofan; HARBAUM, Tanja and BECKER, Jürgen: "Automated Replacement of State-Holding Flip-Flops to Enable Non-Volatile Checkpointing". In: *2023 IEEE Nordic Circuits and Systems Conference (NorCAS)*. IEEE, 2023, pp. 1–7. DOI: 10.1109/norcas58970.2023.10305469.

This page intentionally left blank

Student Theses

This section contains a list of student theses which have been supervised by this thesis' author. Entries are sorted by year and author name. For [Den21, Len21, He22b, Li23], this work's author was the secondary supervisor. Thesis [Cre19, Ste19, Len21] were external thesis, where this work's author was the institute's supervising contact.

[Cre19] Crell, Markus: "Implementation of a Data-Driven, Semi-Autonomous Control for the KIT Prosthetic Hand". Bachelor's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2019.

[Ste19] Steinhilper, Tobias: "Entwurf eines Interfaces als integrierte Hochspannungs-CMOS Schaltung für Kfz-Generatorspannungsregler mit programmierbarer Funktion". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2019.

[Lai20] Laicher, Gerit: "Entwurf einer dynamisch aktualisierbaren Filterarchitektur für Digitales Beamforming mit hohem Datendurchsatz". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2020.

[Wag20] Wagih Zaki Naam, Peter: "Design and Evaluation of Manual Placement Techniques for V-FPGA Tiles on FPGA". Bachelor's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2020.

[Zha20] Zhang, Haoliang: "Evaluation und Adaption von Open-Source FPGA-Architektur Frameworks". Bachelor's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2020.

[Che21] Cheng, Shuohan: "Design und Evaluierung von FPGA Architekturerweiterungen zur Realisierung von Unum Arithmetik". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2021.

[Den21] Deng, Zihao: "An Emulation Framework to Evaluate NVM-based Flip-flops in Processor Architectures". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2021.

[Her21] Hernandez, Victor Eugene Jiro: "Designing a Framework to Evaluate the Performance of Region-based FPGA Power Management Using VPR". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2021.

[Len21] Lenhardt, Dirk: "Performance Comparison Between the SXP and PCI Express Protocol with a Refinement Approach of SXP with Security". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2021.

[Mal21] Mallat, Mirna: "Power Management Techniques in FPGA Architectures". Seminar Paper. Karlsruhe: Karlsruhe Institute of Technology, 2021.

[Mar21b] Marten, Johann Christian: "Entwurf und Evaluation einer Azimut-Prozessierung für Spaceborne SAR". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2021.

[Mül21] Müller, Rasmus: "Design and Evaluation of Hard-Logic Adder Extensions for Virtual FPGAs". Bachelor's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2021.

[Ram21] Ramirez, Nicolas: "Ein Ko-Simulation Framework für dynamisches FPGA Power Management". Bachelor's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2021.

[He22a] He, Xinnan: "Design and Evaluation of RFET In-Memory-Computing Architectures and Programming Toolflow". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2022.

[He22b] He, Zhuofan: "Entwicklung einer FPGA Synthese-Toolchain zur automatisierten Integrierung hybrider Flip-Flops". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2022.

[Mor22] Mortazavi, Sayed Hadi: "Designing a RISCV based Audio Player System for the ITIV System on Chip Lab". Seminar Paper. Karlsruhe: Karlsruhe Institute of Technology, 2022.

[Pri22] Priester, Morris: "Open Source FPGAs und Design Tools Eine Übersicht". Seminar Paper. Karlsruhe: Karlsruhe Institute of Technology, 2022.

[Sil22] Silz, Johannes: "Entwurf und Evaluation eines effizienten LSTM Beschleunigers für den NEORV32 Prozessor". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2022.

[Li23] Li, Jingyi: "Design and Evaluation of a Data Generator for a SAR Satellite Ground Demonstrator". Master's Thesis. Karlsruhe: Karlsruhe Institute of Technology, 2023.

This page intentionally left blank

This page intentionally left blank

Figures

This page intentionally left blank

Tables

This page intentionally left blank

Listings

Acronyms

ABB	Adaptive Body Biasing
AES	Advanced Encryption Standard
AIC	AND-Inverter Cone
AIG	And-Inverter Graph
ALM	Adaptive Logic Module
API	Application Programming Interface
ARX	Add Rotate XOR
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Integrated Processor
BB	Body Biasing
BDD	Binary Decision Diagram
BEOL	Back End of Line
BG	Back Gate
BLE	Basic Logic Element
BLIF	Berkeley Logic Interchange Format
BOX	Buried Oxide
BTI	Bias Temperature Instability
CAD	Computer Aided Design
CB	Connection Box
CBEW	Connection Box East-West
CBR	Read Connection Box
CBSN	Connection Box South-North
CBW	Write Connection Box
CGRA	Coarse Grain Reconfigurable Array
CLB	Controllable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
CMP	Chemical Mechanical Polishing
CNTFET	Carbon Nanotube FET
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CU	Configuration Unit

CVD	Chemical Vapor Deposition
DAC	Digital to Analog Converter
DFG	Data Flow Graph
DGCNTFET	Dual Gate CNTFET
DOS	Denial Of Service
DRAM	Dynamic Random-Access Memory
DRC	Design-Rule Check
DSE	Design Space Evaluation
DSL	Domain Specific Language
DSP	Digital Signal Processing
DVFS	Dynamic Voltage and Frequency Scaling
DVS	Dynamic Voltage Scaling
EDA	Electronic Design Automation
EM	Electromigration
FASM	FPGA ASM
FDSOI	Fully Depleted SOI
FEOL	Front End of Line
FET	Field Effect Transistor
FF	Flip-Flop
FG	Front Gate
FIR	Finite Impulse Response
FLE	Fracturable Logic Element
FLI	Foreign Language Interface
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HCI	Hot Carrier Injection
HDL	Hardware Description Language
HLS	High Level Synthesis
IC	Integrated Circuit
IO	Input / Output
IOB	Input-/Output-Block
IP	Intellectual Property
ITD	Inverted Temperature Dependence
LAB	Logic Array Block
LE	Logic Element

LER	Line-Edge Roughness
LFSR	Linear Feedback Shift Register
LI	Logic Invasion
LUT	Lookup Table
LVS	Layout vs. Schematic
MIGFET	Multiple Independent Gate FET
MLAB	Memory LAB
MOS	Metal Oxide Semiconductor
MOSFET	Metal Oxide Semiconductor FET
MRAM	Magnetoresistive Random-Access Memory
MUX	Multiplexer
NBTI	Negative Bias Temperature Instability
NLDM	Non-Linear Delay Model
NMOS	N-Channel MOSFET
NoC	Network-on-Chip
nonce	Number-used-Once
NTVO	Near-Threshold Voltage Operation
OPE	Optical Proximity Effect
OS	Operating System
PAL	Programmable Array Logic
PARFAIT	Power Aware Reconfigurable FPGA Architecture
PBTI	Positive Bias Temperature Instability
PCB	Printed Circuit Board
PCIe	Peripheral Component Interconnect Express
PDK	Process Design Kit
PDR	Partial Dynamic Reconfiguration
PDSOI	Partially Depleted SOI
PEX	Parasitics Extraction
PG	Program Gate
PLA	Programmable Logic Array
PLE	Physical Layout Estimation
PMOS	P-Channel MOSFET
PPDK	Predictive PDK
PROM	Programmable Read Only Memory
PRR	Partially Reconfigurable Region
PSM	Programmable Switch Matrix
PVD	Physical Vapor Deposition

PVT	Process-, Voltage-, Temperature-Variation
PVTA	Process-, Voltage-, Temperature-Variation and Aging
RDF	Random Dopant Fluctuation
RFET	Reconfigurable (Ambipolar) FET
RO	Ring Oscillator
RRG	Routing Resource Graph
RTL	Register Transfer Level
SB	Switch Box
SBFET	Schottky Barrier FET
SIMD	Single Instruction Multiple Data
SiNW	Silicon Nanowire
SoC	System-on-Chip
SOI	Silicon on Insulator
SOP	Sum-of-Products
SOTB	Silicon on Thin BOX
SRAM	Static Random-Access Memory
SSTA	Statistical STA
STA	Static Timing Analysis
TCAD	Technology CAD
TDDB	Time Dependent Dielectric Breakdown
TG	Top Gate
TIGFET	Three-Independent-Gate FET
ULM	Universal Logic Module
VFPGA	Virtual FPGA
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integration
VPR	Versatile Place and Route
VTC	Voltage-Transfer-Characteristic
VTR	Verilog to Routing

Glossary

ABC	Open-Source logic optimization tool used in VTR.
ARC	A timing ARC is a component of a timing path, an indivisible path from one pin to another. Cell ARCs start at a cell input pin and end at an output pin of the same cell. For net ARCs, the path starts at one cell's output pin and ends at another cell's input pin.
BB	Body Biasing, applying a voltage potential other than the source potential to the body of a MOSFET.
BEOL	Back End of Line, refers to the manufacturing of metallization layers during IC fabrication.
BLIF	File format used to store netlists. The BLIF file format is commonly used by open source tools such as ODIN and ABC.
FASM	A textual format to represent FPGA bitstreams. Used in the VPR tools.
FEOL	Front End of Line, refers to the manufacturing of semiconductor layers during IC fabrication.
FinFET	FET where the gate electrode can be on multiple sides of the channel or even wrap around the channel completely.
FO4	Fanout-four model. In this model, the cell's output is connected to the inputs of four instances of the same gate type. This simulates output loads including dynamic capacitances similar to a circuit with a fanout of four.
genfasm	Open-Source tool to generate FASM from VPR output files. Part of VTR.
Genus	Commercial digital synthesis tool by Cadence used for ASICs.
Innovus	Commercial digital implementation tool by Cadence used for ASICs.

LEF This file format is commonly used in commercial EDA tools to describe the physical layout of gates.

LIB This file format is commonly used in commercial EDA tools to encode timing information of gates.

LI Logic Invasion, a term coined in this thesis. Reusing reconfigurable logic originally used for user applications for system management tasks.

LUA Scripting language designed to be embedded into programs. Used in this thesis to implement delay models.

LUT Lookup Table, a combination of memory and multiplexer that allows deriving output values from a table according to an input value. Most commonly used reconfigurable LE in FPGAs.

nonce In cryptography, a number which must be used only once. For example in many stream ciphers, if a counter value is reused, cryptographic guarantees no longer hold.

ODIN II Open-Source digital synthesis tool for FPGAs and ASICs used in VTR.

PARFAIT Research project where topics of this thesis originated from. Also: Term covering all FPGA architecture extensions introduced in this thesis.

pbit Tool to generate a bitstream for the PARFAIT architecture from FASM. Developed as part of this thesis.

pbit_rr Tool to modify VPR RRGs to generate FASM for the PARFAIT architecture. Developed as part of this thesis.

PDK Process Design Kit, a set of simulation models and library files describing a technology. Usually provided by the technology vendors. A PDK usually does not contain tools, but only configuration data for tools.

PPDK A PDK for a predictive technology, i.e. a technology that is not actually available for fabrication. Used for early evaluation of technologies not yet ready for large-scale production.

RRG File format used by VPR to describe all possible routing paths in an FPGA architecture.

SPICE Analog electronic simulator. The original version is Open Source with most EDA vendors providing custom, commercial spin-offs.

TCAD Technology CAD, EDA tools commonly used to model transistor devices using physical semiconductor models.

TECH File format used in VPR to model power requirements of FPGAs depending on the used technology.

ULM Reconfigurable LE usually realized using logic gates instead of memory as used in LUT. Strictly speaking, ULMs cover only circuits that can realize any function of N inputs. The term is however also commonly used to refer to circuits that can only realize a selected subset of functions.

Vivado Commercial digital synthesis, implementation and simulation tool for FPGAs by Xilinx.

VPR Open-Source tool realizing placement, packing and routing EDA steps. Part of the VTR tool suite.

VTR A suite of tools for FPGA architecture design. It combines of a set of benchmarks for evaluation with a full tool flow based on ODIN II, ABC and VPR.

XML Generic file format used to describe structured data. In this thesis, it is primarily used for VPR to describe FPGA architectures.

Yosys Open-Source digital synthesis tool for FPGAs and ASICs optionally usable with VTR.

This page intentionally left blank

Appendix A

Research Data Archive

Some data used in this thesis can not be published as Open-Source, as it uses parts of commercial PDK or was partially derived from source code from previous institute members with unclear licensing. Nevertheless, all code which was developed as part of this thesis as well as all the raw evaluation data has been stored in the Research Data Archive. The filename for the thesis data is `johannes_pfau_thesis.tar.zst` and the SHA256 checksum of the file is `bf8192cbd243af405b845f2aa5c95d856c37a0ae43cebdb2747b8f899e8fb728`.

This page intentionally left blank

Appendix B

FPGA Architecture Descriptions

```
<pb_type name="io" capacity="8" area="0">
  <input name="outpad" num_pins="1"/>
  <output name="inpad" num_pins="1"/>
  <clock name="clock" num_pins="1"/>

  <!-- IOs can operate as either inputs or outputs.
    -->
  <mode name="inpad">
    <pb_type name="inpad" blif_model=".input" num_pb="1">
      <output name="inpad" num_pins="1"/>
    </pb_type>
    <interconnect>
      <direct name="inpad" input="inpad.inpad" output="io.
          ↪ inpad">
        <delay_constant max="4.243e-11" in_port="inpad.inpad"
            ↪ out_port="io.inpad"/>
      </direct>
    </interconnect>
  </mode>

  <mode name="outpad">
    <pb_type name="outpad" blif_model=".output" num_pb="1">
      <input name="outpad" num_pins="1"/>
    </pb_type>
    <interconnect>
      <direct name="outpad" input="io.outpad" output="outpad.
          ↪ outpad">
        <delay_constant max="1.394e-11" in_port="io.outpad"
            ↪ out_port="outpad.outpad"/>
      </direct>
    </interconnect>
  </mode>

```

```
  <fc in_type="frac" in_val="0.15" out_type="frac" out_val="
      ↪ 0.10"/>

  <pinlocations pattern="custom">
    <loc side="left">io.outpad io.inpad io.clock</loc>
    <loc side="top">io.outpad io.inpad io.clock</loc>
    <loc side="right">io.outpad io.inpad io.clock</loc>
    <loc side="bottom">io.outpad io.inpad io.clock</loc>
  </pinlocations>

  <power method="ignore"/>
</pb_type>
```

Listing B.1: Full description of the *IOB* used in the *k6_frac_N10_40nm* reference architecture.

```
<mode name="n1_lut6">
  <pb_type name="ble6" num_pb="1">
    <input name="in" num_pins="6"/>
    <output name="out" num_pins="1"/>
    <clock name="clk" num_pins="1"/>

    <pb_type name="lut6" blif_model=".names" num_pb="1" class=
        ↪ "lut">
      <input name="in" num_pins="6" port_class="lut_in"/>
      <output name="out" num_pins="1" port_class="lut_out"/>
    </pb_type>

    <pb_type name="ff" blif_model=".latch" num_pb="1" class="
        ↪ flipflop">
      <input name="D" num_pins="1" port_class="D"/>
      <output name="Q" num_pins="1" port_class="Q"/>
      <clock name="clk" num_pins="1" port_class="clock"/>
      <T_setup value="66e-12" port="ff.D" clock="clk"/>
      <T_clock_to_Q max="124e-12" port="ff.Q" clock="clk"/>
    </pb_type>

    <interconnect>
      <direct name="direct1" input="ble6.in" output="lut6[0:0
          ↪ ].in"/>
      <direct name="direct2" input="lut6.out" output="ff.D">
        <pack_pattern name="ble6" in_port="lut6.out" out_port=
            ↪ "ff.D"/>
      </direct>
```

```
25        <direct name="direct3" input="ble6.clk" output="ff.clk"/
              ↪ >
26        <mux name="mux1" input="ff.Q lut6.out" output="ble6.out"
              ↪ ></mux>
27      </interconnect>
28    </pb_type>
29    <interconnect>
30      <direct name="direct1" input="fle.in" output="ble6.in"/>
31      <direct name="direct2" input="ble6.out" output="fle.out[0
            ↪ :0]"/>
32      <direct name="direct3" input="fle.clk" output="ble6.clk"/>
33    </interconnect>
34  </mode>
```

Listing B.2: LUT6 mode of the FLE XML architecture description used by the VTR framework and VPR.

```
1   <mode name="n2_lut5">
2     <pb_type name="lut5inter" num_pb="1">
3       <input name="in" num_pins="5"/>
4       <output name="out" num_pins="2"/>
5       <clock name="clk" num_pins="1"/>
6
7       <pb_type name="ble5" num_pb="2">
8         <input name="in" num_pins="5"/>
9         <output name="out" num_pins="1"/>
10        <clock name="clk" num_pins="1"/>
11
12        <pb_type name="lut5" blif_model=".names" num_pb="1"
              ↪ class="lut">
13          <input name="in" num_pins="5" port_class="lut_in"/>
14          <output name="out" num_pins="1" port_class="lut_out"/>
15        </pb_type>
16
17        <pb_type name="ff" blif_model=".latch" num_pb="1" class=
              ↪ "flipflop">
18          <input name="D" num_pins="1" port_class="D"/>
19          <output name="Q" num_pins="1" port_class="Q"/>
20          <clock name="clk" num_pins="1" port_class="clock"/>
21        </pb_type>
22
23        <interconnect>
24          <direct name="direct1" input="ble5.in[4:0]" output="
                ↪ lut5[0:0].in[4:0]"/>
```

```
25          <direct name="direct2" input="lut5[0:0].out" output="
                ↪ ff[0:0].D">
26            <pack_pattern name="ble5" in_port="lut5[0:0].out"
                  ↪ out_port="ff[0:0].D"/>
27          </direct>
28          <direct name="direct3" input="ble5.clk" output="ff[0:0
                ↪ ].clk"/>
29          <mux name="mux1" input="ff[0:0].Q lut5.out[0:0]"
                ↪ output="ble5.out[0:0]"></mux>
30        </interconnect>
31      </pb_type>
32      <interconnect>
33        <direct name="direct1" input="lut5inter.in" output="ble5
              ↪ [0:0].in"/>
34        <direct name="direct2" input="lut5inter.in" output="ble5
              ↪ [1:1].in"/>
35        <direct name="direct3" input="ble5[1:0].out" output="
              ↪ lut5inter.out"/>
36        <complete name="complete1" input="lut5inter.clk" output=
              ↪ "ble5[1:0].clk"/>
37      </interconnect>
38    </pb_type>
39
40    <interconnect>
41      <direct name="direct1" input="fle.in[4:0]" output="
            ↪ lut5inter.in"/>
42      <direct name="direct2" input="lut5inter.out" output="fle.
            ↪ out"/>
43      <direct name="direct3" input="fle.clk" output="lut5inter.
            ↪ clk"/>
44    </interconnect>
45  </mode>
```

Listing B.3: Dual LUT5 mode of the FLE XML architecture description used by the VTR framework and VPR.

Appendix C

Delay Model Extraction

```
library ieee;
use ieee.std_logic_1164.all;

entity tpd is
    port (
        a: in std_logic;
        y: out std_logic_vector(3 downto 0)
    );
end;

architecture xt018 of tpd is
    signal inv0_out, inv1_out: std_logic;
begin

    inv0: entity work.INHDX0
    port map (
        A => a,
        Q => inv0_out
    );

    inv1: entity work.INHDX0
    port map (
        A => inv0_out,
        Q => inv1_out
    );

    inv2_0: entity work.INHDX0
    port map (
        A => inv1_out,
        Q => y(0)
    );

    inv2_1: entity work.INHDX0
```

```
34      port map (
35          A => inv1_out,
36          Q => y(1)
37      );
38
39      inv2_2: entity work.INHDX0
40      port map (
41          A => inv1_out,
42          Q => y(2)
43      );
44
45      inv2_3: entity work.INHDX0
46      port map (
47          A => inv1_out,
48          Q => y(3)
49      );
50
51  end;
```

Listing C.1: VHDL code used in Cadence Genus synthesis with the *XT018* PDK to extract FO4 inverter delay. The model is used with different corner files to extract differences caused by process variation.

Appendix D

Standard Cell Library Excerpts

```
/*
 * cell_description : 2-Input NOR
 */

cell (NO2HDX1) {
  area : 10.0352;
  cell_footprint : NO2;

  pg_pin (gnd) {
    voltage_name : gndCommon;
    pg_type : primary_ground;
  }
  pg_pin (vdd) {
    voltage_name : vddInt;
    pg_type : primary_power;
  }

  pin (A) {
    direction : input;
    related_ground_pin : gnd;
    related_power_pin : vdd;
    rise_capacitance : 0.006461898799178912;
    fall_capacitance : 0.005618442917028880;
  }
  pin (B) {
    direction : input;
    related_ground_pin : gnd;
    related_power_pin : vdd;
    rise_capacitance : 0.006480533990037019;
    fall_capacitance : 0.005240246832871408;
  }
  pin (Q) {
    direction : output;
```

```
34      function : "!(A+B)";
35      max_capacitance: 0.015;
36      max_fanout : 10;
37      related_ground_pin : gnd;
38      related_power_pin : vdd;
39      power_down_function : "!vdd + gnd";
40
41      timing () {
42        timing_sense : negative_unate;
43        related_pin : A;
44        rise_transition (delay_template_3x3) {
45          index_1 ("0.060000, 0.300000, 0.540000");
46          index_2 ("0.003000, 0.009000, 0.015000");
47          values ( \
48          "0.205750, 0.241829, 0.277907", \
49          "0.329676, 0.356424, 0.383173", \
50          "0.453601, 0.471020, 0.488439");
51        }
52        fall_transition (delay_template_3x3) {
53          index_1 ("0.060000, 0.300000, 0.540000");
54          index_2 ("0.003000, 0.009000, 0.015000");
55          values ( \
56          "0.270651, 0.305638, 0.340626", \
57          "0.384688, 0.406597, 0.428506", \
58          "0.498726, 0.507556, 0.516386");
59        }
60        cell_rise (delay_template_3x3) {
61          index_1 ("0.060000, 0.300000, 0.540000");
62          index_2 ("0.003000, 0.009000, 0.015000");
63          values ( \
64          "0.250443, 0.271110, 0.291777", \
65          "0.339382, 0.364643, 0.389904", \
66          "0.428321, 0.458176, 0.488031");
67        }
68        cell_fall (delay_template_3x3) {
69          index_1 ("0.060000, 0.300000, 0.540000");
70          index_2 ("0.003000, 0.009000, 0.015000");
71          values ( \
72          "0.235192, 0.321403, 0.407613", \
73          "0.312413, 0.392071, 0.471729", \
74          "0.389633, 0.462739, 0.535844");
75        }
76      }
77      timing () {
```

```
78        timing_sense : negative_unate;
79        related_pin : B;
80        rise_transition (delay_template_3x3) {
81          index_1 ("0.060000, 0.300000, 0.540000");
82          index_2 ("0.003000, 0.009000, 0.015000");
83          values ( \
84          "0.205300, 0.252093, 0.298886", \
85          "0.329448, 0.366706, 0.403964", \
86          "0.453595, 0.481319, 0.509043");
87        }
88        fall_transition (delay_template_3x3) {
89          index_1 ("0.060000, 0.300000, 0.540000");
90          index_2 ("0.003000, 0.009000, 0.015000");
91          values ( \
92          "0.158357, 0.193715, 0.229073", \
93          "0.272944, 0.298892, 0.324839", \
94          "0.387532, 0.404069, 0.420605");
95        }
96        cell_rise (delay_template_3x3) {
97          index_1 ("0.060000, 0.300000, 0.540000");
98          index_2 ("0.003000, 0.009000, 0.015000");
99          values ( \
100         "0.183191, 0.207483, 0.231776", \
101         "0.272849, 0.306402, 0.339955", \
102         "0.362508, 0.405321, 0.448134");
103       }
104       cell_fall (delay_template_3x3) {
105         index_1 ("0.060000, 0.300000, 0.540000");
106         index_2 ("0.003000, 0.009000, 0.015000");
107         values ( \
108         "0.160941, 0.223859, 0.286777", \
109         "0.236255, 0.303522, 0.370789", \
110         "0.311569, 0.383185, 0.454801");
111       }
112     }
113   }
114 }
```

Listing D.1: Full description of the *NO2* gate in the RFET *.lib* file.

```
1 wire_load_table (0_1k) {
2   fanout_area (1, x.xx);
3   fanout_capacitance (1, x.xx);
4   fanout_length (1, x.xx);
```

```
    fanout_resistance (1, x.xx);
    fanout_area (5, x.xx);
    fanout_capacitance (5, x.xx);
    fanout_length (5, x.xx);
    fanout_resistance (5, x.xx);
    fanout_area (20, x.xx);
    fanout_capacitance (20, x.xx);
    fanout_length (20, x.xx);
    fanout_resistance (20, x.xx);
    fanout_area (10000, x.xx);
    fanout_capacitance (10000, x.xx);
    fanout_length (10000, x.xx);
    fanout_resistance (10000, x.xx);
}
```

Listing D.2: Wireload description in the RFET *.lib* file.

Appendix E

FASM for Logic Invasion

```
# This is the fasm file is used to initialize the LFSR counter . LUTs always set the registers to a constant value.
# To compile: ./src/sw/pbit --single CLB src/fasm/clb_counter_init.fasm

# Pass through FLE0 LUT0 input 0 from external input
BLK_X[000]Y[000]Z[000].CLB.FLE[0].CB.I[0]="ext[0].I"
BLK_X[000]Y[000]Z[000].CLB.FLE[0].5BLE[0].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[0].5BLE[0].LUT5.INIT[31:0]=32'b10101010101010101010101010101010

# Configure BLE0 LUT 1 as constant 1
BLK_X[000]Y[000]Z[000].CLB.FLE[0].5BLE[1].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[0].5BLE[1].LUT5.INIT[31:0]=32'b11111111111111111111111111111111

# All other LUTs as constant 0
BLK_X[000]Y[000]Z[000].CLB.FLE[1].5BLE[0].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[1].5BLE[0].LUT5.INIT[31:0]=32'b00000000000000000000000000000000
BLK_X[000]Y[000]Z[000].CLB.FLE[1].5BLE[1].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[1].5BLE[1].LUT5.INIT[31:0]=32'b00000000000000000000000000000000

BLK_X[000]Y[000]Z[000].CLB.FLE[2].5BLE[0].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[2].5BLE[0].LUT5.INIT[31:0]=32'b00000000000000000000000000000000
BLK_X[000]Y[000]Z[000].CLB.FLE[2].5BLE[1].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[2].5BLE[1].LUT5.INIT[31:0]=32'b00000000000000000000000000000000

BLK_X[000]Y[000]Z[000].CLB.FLE[3].5BLE[0].FF.ENABLE
```

```
BLK_X[000]Y[000]Z[000].CLB.FLE[3].5BLE[0].LUT5.INIT[31:0]=32'b
    ↪ 00000000000000000000000000000000
BLK_X[000]Y[000]Z[000].CLB.FLE[3].5BLE[1].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[3].5BLE[1].LUT5.INIT[31:0]=32'b
    ↪ 00000000000000000000000000000000

BLK_X[000]Y[000]Z[000].CLB.FLE[4].5BLE[0].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[4].5BLE[0].LUT5.INIT[31:0]=32'b
    ↪ 00000000000000000000000000000000
BLK_X[000]Y[000]Z[000].CLB.FLE[4].5BLE[1].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[4].5BLE[1].LUT5.INIT[31:0]=32'b
    ↪ 00000000000000000000000000000000

BLK_X[000]Y[000]Z[000].CLB.FLE[5].5BLE[0].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[5].5BLE[0].LUT5.INIT[31:0]=32'b
    ↪ 00000000000000000000000000000000
BLK_X[000]Y[000]Z[000].CLB.FLE[5].5BLE[1].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[5].5BLE[1].LUT5.INIT[31:0]=32'b
    ↪ 00000000000000000000000000000000

BLK_X[000]Y[000]Z[000].CLB.FLE[6].5BLE[0].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[6].5BLE[0].LUT5.INIT[31:0]=32'b
    ↪ 00000000000000000000000000000000
BLK_X[000]Y[000]Z[000].CLB.FLE[6].5BLE[1].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[6].5BLE[1].LUT5.INIT[31:0]=32'b
    ↪ 00000000000000000000000000000000

BLK_X[000]Y[000]Z[000].CLB.FLE[7].5BLE[0].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[7].5BLE[0].LUT5.INIT[31:0]=32'b
    ↪ 00000000000000000000000000000000
BLK_X[000]Y[000]Z[000].CLB.FLE[7].5BLE[1].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[7].5BLE[1].LUT5.INIT[31:0]=32'b
    ↪ 00000000000000000000000000000000

BLK_X[000]Y[000]Z[000].CLB.FLE[8].5BLE[0].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[8].5BLE[0].LUT5.INIT[31:0]=32'b
    ↪ 00000000000000000000000000000000
BLK_X[000]Y[000]Z[000].CLB.FLE[8].5BLE[1].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[8].5BLE[1].LUT5.INIT[31:0]=32'b
    ↪ 00000000000000000000000000000000

BLK_X[000]Y[000]Z[000].CLB.FLE[9].5BLE[0].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[9].5BLE[0].LUT5.INIT[31:0]=32'b
    ↪ 00000000000000000000000000000000
```

```
BLK_X[000]Y[000]Z[000].CLB.FLE[9].5BLE[1].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[9].5BLE[1].LUT5.INIT[31:0]=32'b
    ↪ 00000000000000000000000000000000
```

Listing E.1: FASM representing the register initialization for measurement in section 8.3 on page 214.

```
# This is the fasm file used to build an oscillator for logic
    ↪ invasion
# To compile: ./src/sw/pbit --single CLB src/fasm/clb_ring.
    ↪ fasm

# We build a ring oscillator in the LUT0s of BLE 0-9 and LUT1
    ↪ of BLE10
# Oscillator input is I0, so an inverter/NAND of I0 is used
# The counter / LFSR is implemented in LUT1s of BLE0-8.
# I1 is counter input, I2 optional second input when using an
    ↪ xor for the first tap.
# For non-first tap, use a pass-through of I1

# Externally gated inverter (invert fle[9].O[1] if ext[0].I is
    ↪  1)
BLK_X[000]Y[000]Z[000].CLB.FLE[0].CB.I[0]="fle[9].O[1]"
BLK_X[000]Y[000]Z[000].CLB.FLE[0].CB.I[3]="ext[0].I"
BLK_X[000]Y[000]Z[000].CLB.FLE[0].5BLE[0].FF.BYPASS
BLK_X[000]Y[000]Z[000].CLB.FLE[0].5BLE[0].LUT5.INIT[31:0]=32'b
    ↪ 01010101000000000101010100000000
# x9 + x5 + 1
BLK_X[000]Y[000]Z[000].CLB.FLE[0].CB.I[1]="fle[8].O[1]"
BLK_X[000]Y[000]Z[000].CLB.FLE[0].CB.I[2]="fle[4].O[1]"
BLK_X[000]Y[000]Z[000].CLB.FLE[0].5BLE[1].FF.ENABLE
# The XOR of I1 and I2
BLK_X[000]Y[000]Z[000].CLB.FLE[0].5BLE[1].LUT5.INIT[31:0]=32'b
    ↪ 00111100001111000011110000111100

# All following ring elements are buffers
BLK_X[000]Y[000]Z[000].CLB.FLE[1].CB.I[0]="fle[0].O[0]"
BLK_X[000]Y[000]Z[000].CLB.FLE[1].CB.I[1]="fle[0].O[1]"
BLK_X[000]Y[000]Z[000].CLB.FLE[1].5BLE[0].FF.BYPASS
BLK_X[000]Y[000]Z[000].CLB.FLE[1].5BLE[0].LUT5.INIT[31:0]=32'b
    ↪ 10101010101010101010101010101010
BLK_X[000]Y[000]Z[000].CLB.FLE[1].5BLE[1].FF.ENABLE
BLK_X[000]Y[000]Z[000].CLB.FLE[1].5BLE[1].LUT5.INIT[31:0]=32'b
    ↪ 11001100110011001100110011001100
```

```
29
30 BLK_X[000]Y[000]Z[000].CLB.FLE[2].CB.I[0]="fle[1].O[0]"
31 BLK_X[000]Y[000]Z[000].CLB.FLE[2].CB.I[1]="fle[1].O[1]"
32 BLK_X[000]Y[000]Z[000].CLB.FLE[2].5BLE[0].FF.BYPASS
33 BLK_X[000]Y[000]Z[000].CLB.FLE[2].5BLE[0].LUT5.INIT[31:0]=32'b
       ↪ 10101010101010101010101010101010
34 BLK_X[000]Y[000]Z[000].CLB.FLE[2].5BLE[1].FF.ENABLE
35 BLK_X[000]Y[000]Z[000].CLB.FLE[2].5BLE[1].LUT5.INIT[31:0]=32'b
       ↪ 11001100110011001100110011001100
36
37 BLK_X[000]Y[000]Z[000].CLB.FLE[3].CB.I[0]="fle[2].O[0]"
38 BLK_X[000]Y[000]Z[000].CLB.FLE[3].CB.I[1]="fle[2].O[1]"
39 BLK_X[000]Y[000]Z[000].CLB.FLE[3].5BLE[0].FF.BYPASS
40 BLK_X[000]Y[000]Z[000].CLB.FLE[3].5BLE[0].LUT5.INIT[31:0]=32'b
       ↪ 10101010101010101010101010101010
41 BLK_X[000]Y[000]Z[000].CLB.FLE[3].5BLE[1].FF.ENABLE
42 BLK_X[000]Y[000]Z[000].CLB.FLE[3].5BLE[1].LUT5.INIT[31:0]=32'b
       ↪ 11001100110011001100110011001100
43
44 BLK_X[000]Y[000]Z[000].CLB.FLE[4].CB.I[0]="fle[3].O[0]"
45 BLK_X[000]Y[000]Z[000].CLB.FLE[4].CB.I[1]="fle[3].O[1]"
46 BLK_X[000]Y[000]Z[000].CLB.FLE[4].5BLE[0].FF.BYPASS
47 BLK_X[000]Y[000]Z[000].CLB.FLE[4].5BLE[0].LUT5.INIT[31:0]=32'b
       ↪ 10101010101010101010101010101010
48 BLK_X[000]Y[000]Z[000].CLB.FLE[4].5BLE[1].FF.ENABLE
49 BLK_X[000]Y[000]Z[000].CLB.FLE[4].5BLE[1].LUT5.INIT[31:0]=32'b
       ↪ 11001100110011001100110011001100
50
51 BLK_X[000]Y[000]Z[000].CLB.FLE[5].CB.I[0]="fle[4].O[0]"
52 BLK_X[000]Y[000]Z[000].CLB.FLE[5].CB.I[1]="fle[4].O[1]"
53 BLK_X[000]Y[000]Z[000].CLB.FLE[5].5BLE[0].FF.BYPASS
54 BLK_X[000]Y[000]Z[000].CLB.FLE[5].5BLE[0].LUT5.INIT[31:0]=32'b
       ↪ 10101010101010101010101010101010
55 BLK_X[000]Y[000]Z[000].CLB.FLE[5].5BLE[1].FF.ENABLE
56 BLK_X[000]Y[000]Z[000].CLB.FLE[5].5BLE[1].LUT5.INIT[31:0]=32'b
       ↪ 11001100110011001100110011001100
57
58 BLK_X[000]Y[000]Z[000].CLB.FLE[6].CB.I[0]="fle[5].O[0]"
59 BLK_X[000]Y[000]Z[000].CLB.FLE[6].CB.I[1]="fle[5].O[1]"
60 BLK_X[000]Y[000]Z[000].CLB.FLE[6].5BLE[0].FF.BYPASS
61 BLK_X[000]Y[000]Z[000].CLB.FLE[6].5BLE[0].LUT5.INIT[31:0]=32'b
       ↪ 10101010101010101010101010101010
62 BLK_X[000]Y[000]Z[000].CLB.FLE[6].5BLE[1].FF.ENABLE
63 BLK_X[000]Y[000]Z[000].CLB.FLE[6].5BLE[1].LUT5.INIT[31:0]=32'b
```

```
    ↪ 1100110011001100110011001100110011001100
64
65  BLK_X[000]Y[000]Z[000].CLB.FLE[7].CB.I[0]="fle[6].O[0]"
66  BLK_X[000]Y[000]Z[000].CLB.FLE[7].CB.I[1]="fle[6].O[1]"
67  BLK_X[000]Y[000]Z[000].CLB.FLE[7].5BLE[0].FF.BYPASS
68  BLK_X[000]Y[000]Z[000].CLB.FLE[7].5BLE[0].LUT5.INIT[31:0]=32'b
    ↪ 10101010101010101010101010101010
69  BLK_X[000]Y[000]Z[000].CLB.FLE[7].5BLE[1].FF.ENABLE
70  BLK_X[000]Y[000]Z[000].CLB.FLE[7].5BLE[1].LUT5.INIT[31:0]=32'b
    ↪ 1100110011001100110011001100110011001100
71
72  BLK_X[000]Y[000]Z[000].CLB.FLE[8].CB.I[0]="fle[7].O[0]"
73  BLK_X[000]Y[000]Z[000].CLB.FLE[8].CB.I[1]="fle[7].O[1]"
74  BLK_X[000]Y[000]Z[000].CLB.FLE[8].5BLE[0].FF.BYPASS
75  BLK_X[000]Y[000]Z[000].CLB.FLE[8].5BLE[0].LUT5.INIT[31:0]=32'b
    ↪ 10101010101010101010101010101010
76  BLK_X[000]Y[000]Z[000].CLB.FLE[8].5BLE[1].FF.ENABLE
77  BLK_X[000]Y[000]Z[000].CLB.FLE[8].5BLE[1].LUT5.INIT[31:0]=32'b
    ↪ 1100110011001100110011001100110011001100
78
79  BLK_X[000]Y[000]Z[000].CLB.FLE[9].CB.I[0]="fle[8].O[0]"
80  BLK_X[000]Y[000]Z[000].CLB.FLE[9].CB.I[1]="fle[9].O[0]"
81  BLK_X[000]Y[000]Z[000].CLB.FLE[9].5BLE[0].FF.BYPASS
82  BLK_X[000]Y[000]Z[000].CLB.FLE[9].5BLE[0].LUT5.INIT[31:0]=32'b
    ↪ 10101010101010101010101010101010
83  BLK_X[000]Y[000]Z[000].CLB.FLE[9].5BLE[1].FF.BYPASS
84  # Pass-Through I1
85  BLK_X[000]Y[000]Z[000].CLB.FLE[9].5BLE[1].LUT5.INIT[31:0]=32'b
    ↪ 1100110011001100110011001100110011001100
```

Listing E.2: FASM representing the counter and oscillator for measurement in section 8.3 on page 214.

```
1  # This is the fasm file is used to output the register values
       ↪ to the LUT0 output.
2  # All registers are on, so they are available on the CB
3  # To compile: ./src/sw/pbit --single CLB src/fasm/
       ↪ clb_counter_init.fasm
4
5  # This selects the register routed to the 0 input of the first
       ↪  LUT,
6  # which is then emitted to LUT0. This is just a placeholder,
       ↪ the actual
7  # value is substituted in gen_clb_read_bitstream
```

```
8  BLK_X[000]Y[000]Z[000].CLB.FLE[0].CB.I[0]="fle[0].O[0]"
9
10 # LUT0 just forwards its input 0. The CB must route the wanted
       ↪ register to this input.
11 BLK_X[000]Y[000]Z[000].CLB.FLE[0].5BLE[0].LUT5.INIT[31:0]=32'b
       ↪ 10101010101010101010101010101010
12 BLK_X[000]Y[000]Z[000].CLB.FLE[0].5BLE[0].FF.ENABLE
13 BLK_X[000]Y[000]Z[000].CLB.FLE[0].5BLE[1].FF.ENABLE
14
15 # Enable all FFs
16 BLK_X[000]Y[000]Z[000].CLB.FLE[1].5BLE[0].FF.ENABLE
17 BLK_X[000]Y[000]Z[000].CLB.FLE[1].5BLE[1].FF.ENABLE
18
19 BLK_X[000]Y[000]Z[000].CLB.FLE[2].5BLE[0].FF.ENABLE
20 BLK_X[000]Y[000]Z[000].CLB.FLE[2].5BLE[1].FF.ENABLE
21
22 BLK_X[000]Y[000]Z[000].CLB.FLE[3].5BLE[0].FF.ENABLE
23 BLK_X[000]Y[000]Z[000].CLB.FLE[3].5BLE[1].FF.ENABLE
24
25 BLK_X[000]Y[000]Z[000].CLB.FLE[4].5BLE[0].FF.ENABLE
26 BLK_X[000]Y[000]Z[000].CLB.FLE[4].5BLE[1].FF.ENABLE
27
28 BLK_X[000]Y[000]Z[000].CLB.FLE[5].5BLE[0].FF.ENABLE
29 BLK_X[000]Y[000]Z[000].CLB.FLE[5].5BLE[1].FF.ENABLE
30
31 BLK_X[000]Y[000]Z[000].CLB.FLE[6].5BLE[0].FF.ENABLE
32 BLK_X[000]Y[000]Z[000].CLB.FLE[6].5BLE[1].FF.ENABLE
33
34 BLK_X[000]Y[000]Z[000].CLB.FLE[7].5BLE[0].FF.ENABLE
35 BLK_X[000]Y[000]Z[000].CLB.FLE[7].5BLE[1].FF.ENABLE
36
37 BLK_X[000]Y[000]Z[000].CLB.FLE[8].5BLE[0].FF.ENABLE
38 BLK_X[000]Y[000]Z[000].CLB.FLE[8].5BLE[1].FF.ENABLE
39
40 BLK_X[000]Y[000]Z[000].CLB.FLE[9].5BLE[0].FF.ENABLE
41 BLK_X[000]Y[000]Z[000].CLB.FLE[9].5BLE[1].FF.ENABLE
```

Listing E.3: FASM representing the FF readout configuration for measurement in section 8.3 on page 214.

Appendix F

PARFAIT FPGA Evaluation Results

This chapter contains additional figures for the evaluation in chapter 10 on page 253. The evaluation chapter contains detailed instructions on how these figures can be interpreted. It also provides examples and interpretations for selected benchmarks. The figures in this appendix cover eight of the evaluated benchmarks. In addition, not all RFET and SOI evaluations are shown, due to space limitations.

This appendix first shows power evaluation figures, followed by process variation, voltage variation, temperature variation and aging figures. Aggregated statistics, which summarize average values, are given in the evaluation chapter. Appendix figure figure F.1 on the following page provides placements for all FPGA. Figure F.2 on page 367 to figure F.9 on page 374 provide evaluations for power management without PVTA. Figure F.10 on page 375 to figure F.17 on page 382 show the simulations with process variation. Figure F.18 on page 383 gives an overview of all voltage variation maps used in the voltage evaluation. The voltage variation evaluation itself is given in figure F.19 on page 384 to figure F.32 on page 397. Temperature evaluation maps follow in figure F.33 on page 398 to figure F.36 on page 401. The appendix concludes with aging evaluations in figure F.37 on page 402 to figure F.44 on page 409.

There's one aspect to be noted in the achieved delay graphs: Even for regions in low-power modes, such as unused regions, the achieved delay can be lower than the nominal delay. This is caused by process variation causing some areas to have better than typical delay and results in graphs having a mostly blue shade. For larger region sizes and the same benchmark, the amount of such high-performance regions can seem to be smaller: The reason for this being that the current delay is showing the delay as it is known by the compensation algorithm. This algorithm always uses the worst value in a region, causing the whole area to show higher delay. The real delay can however be lower for most CLBs in the region.

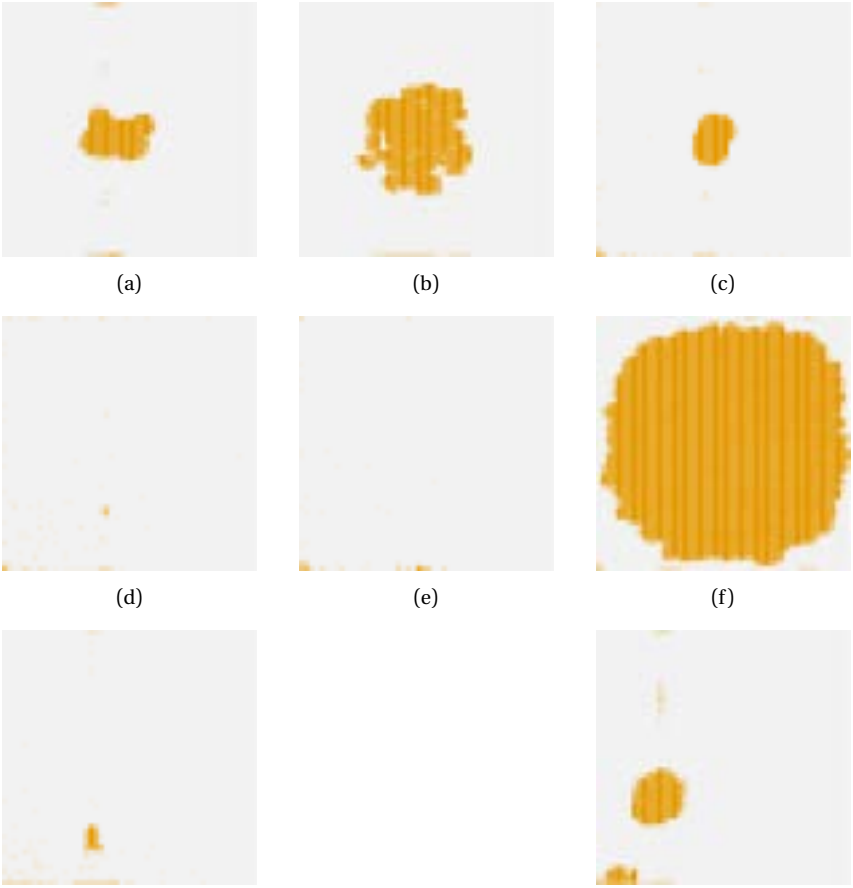


Figure F.1: Placements for the benchmarks which are evaluated on the PARFAIT FPGA architecture. Placements were obtained using the ULM VPR flow introduced in section 6.2 on page 180. **(a)** to **(h)**: Benchmarks `arm_core`, `bgm`, `blob_merge`, `diffeq2`, `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B` and `stereovision0`.

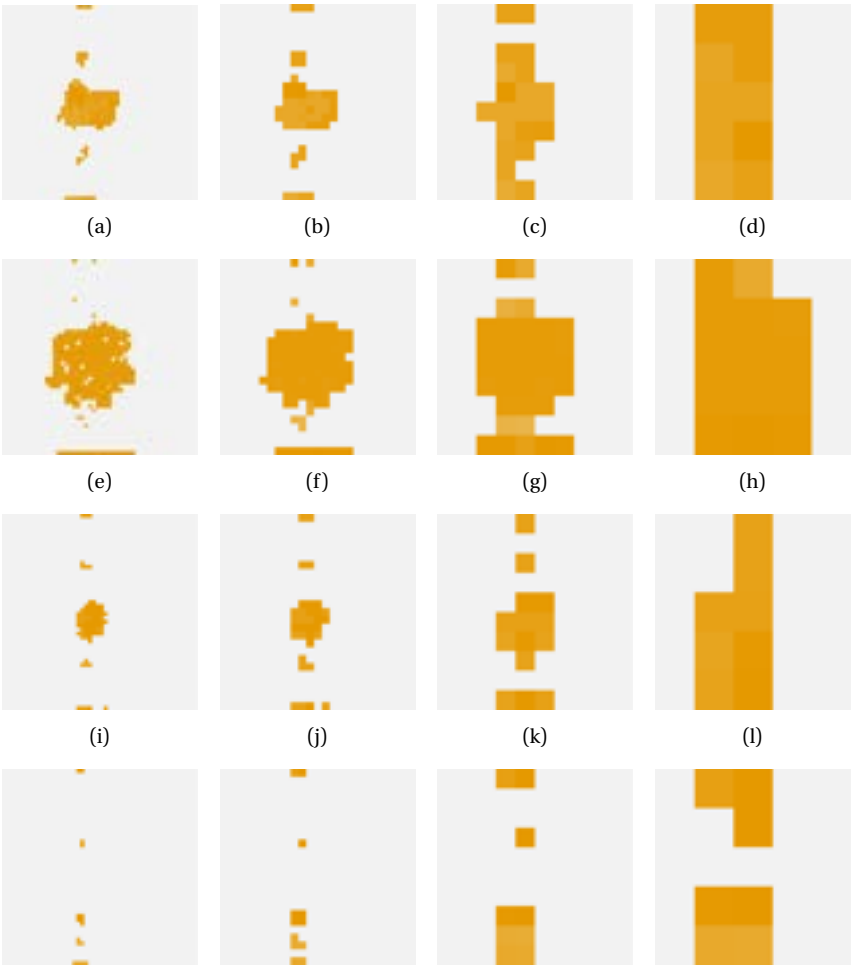


Figure F.2: PARFAIT target factor maps evaluated using the SOI delay model introduced in section 4.6. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

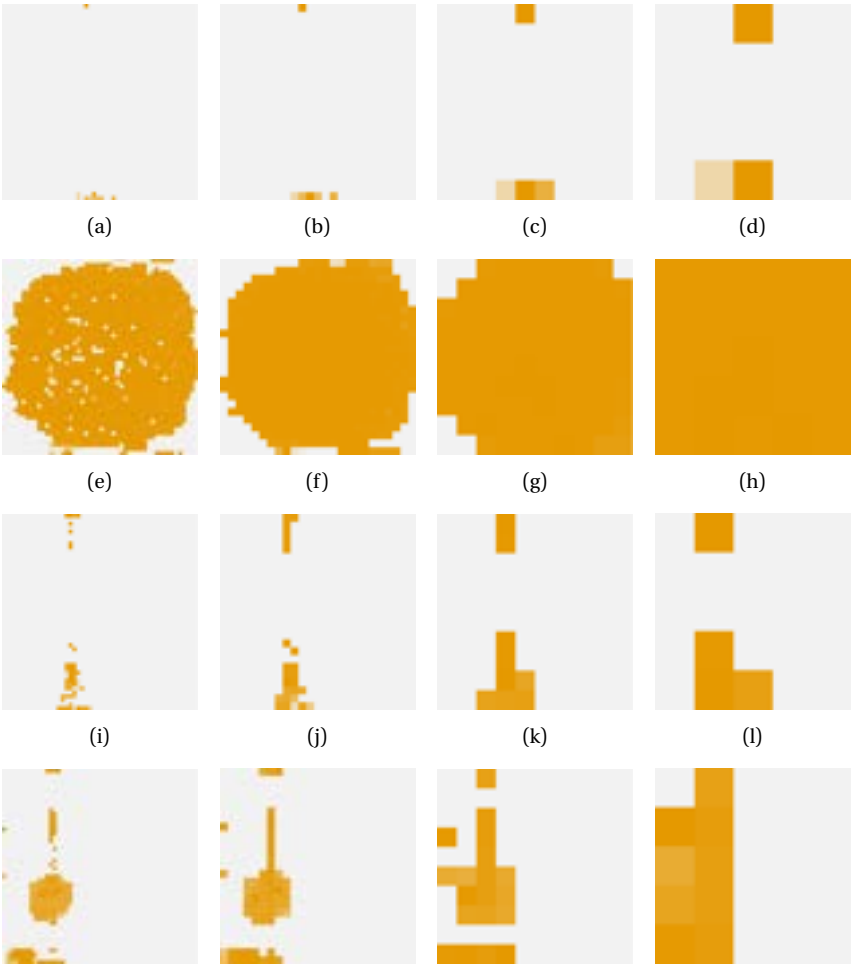


Figure F.3: PARFAIT target factor maps evaluated using the SOI delay model introduced in section 4.6. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

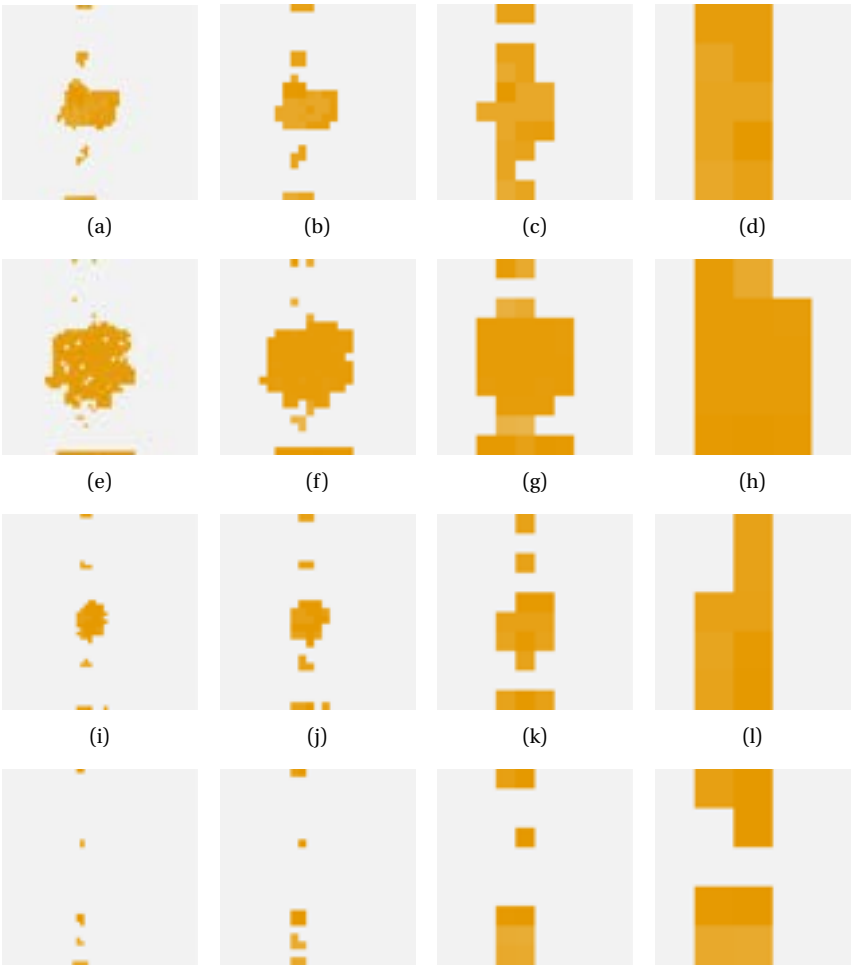


Figure F.4: PARFAIT target factor maps evaluated using the RFET delay model introduced in section 4.6. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

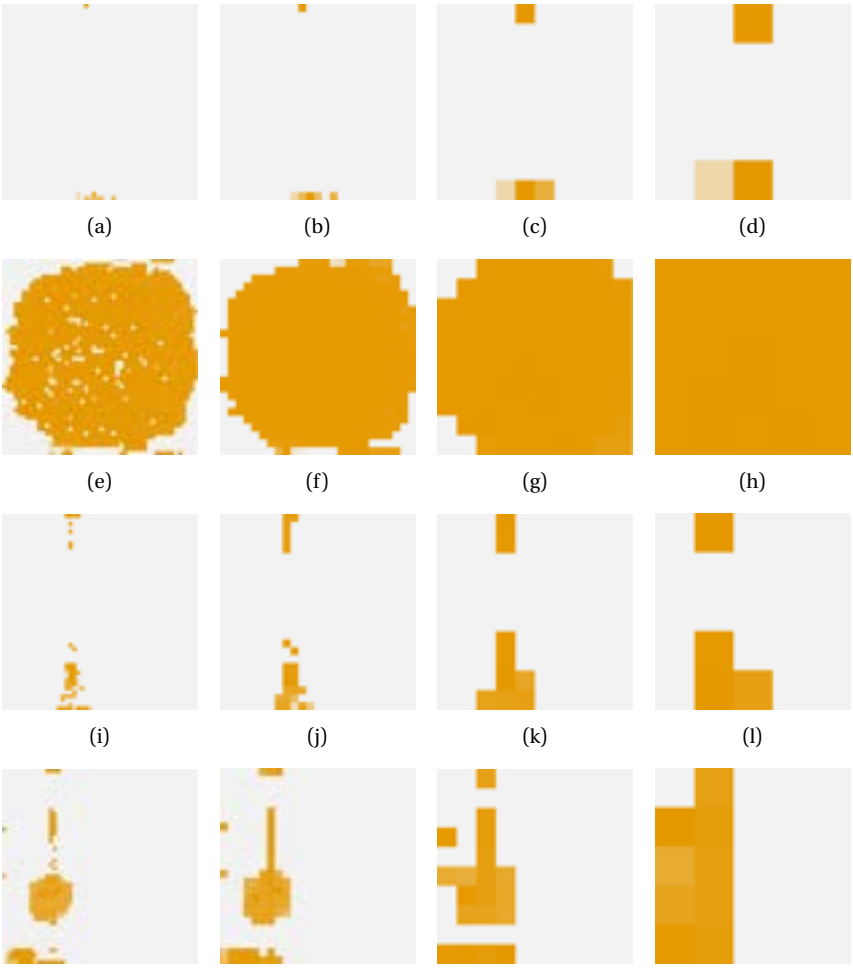


Figure F.5: PARFAIT target factor maps evaluated using the RFET delay model introduced in section 4.6. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

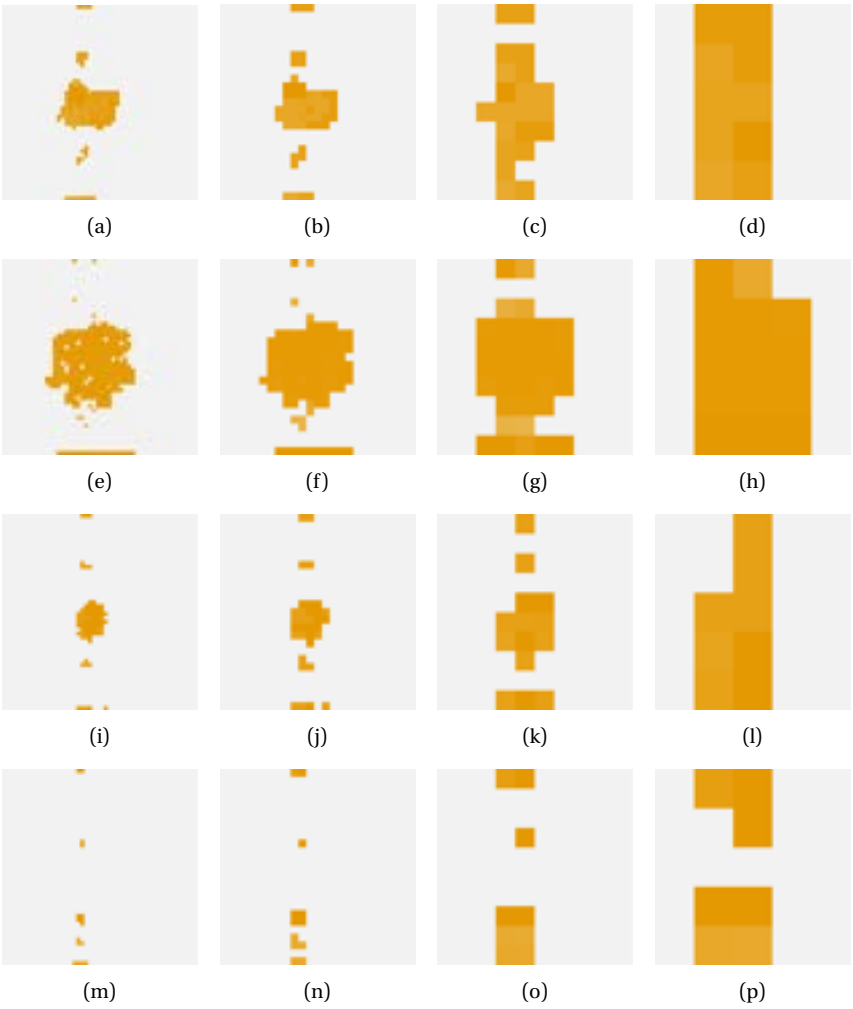


Figure F.6: PARFAIT delay factor maps evaluated using the RFET delay model introduced in section 4.6. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

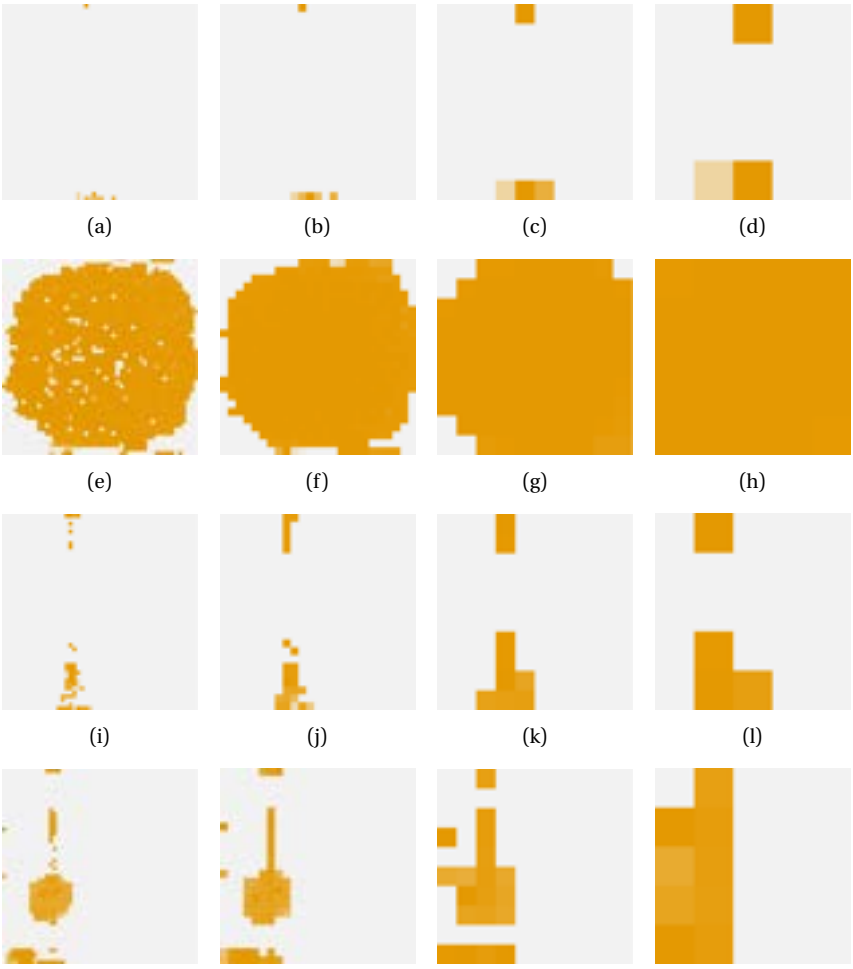


Figure F.7: PARFAIT delay factor maps evaluated using the RFET delay model introduced in section 4.6. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

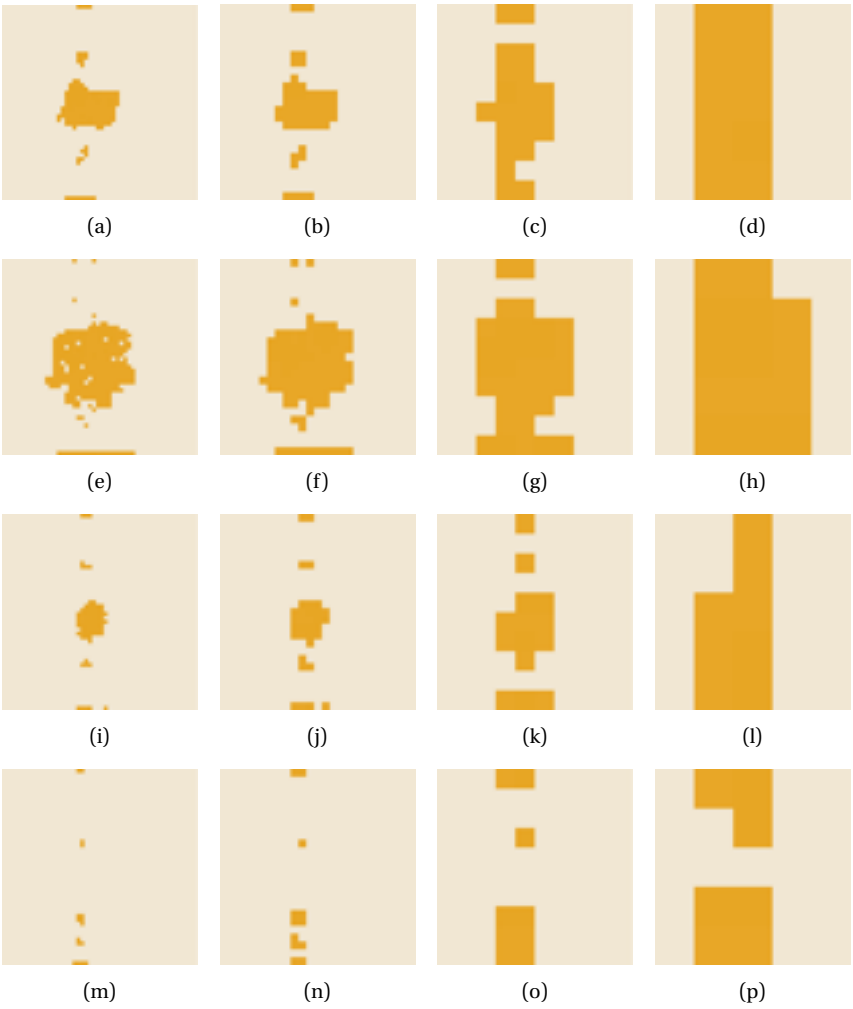


Figure F.8: PARFAIT power maps evaluated using the RFET delay model introduced in section 4.6. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

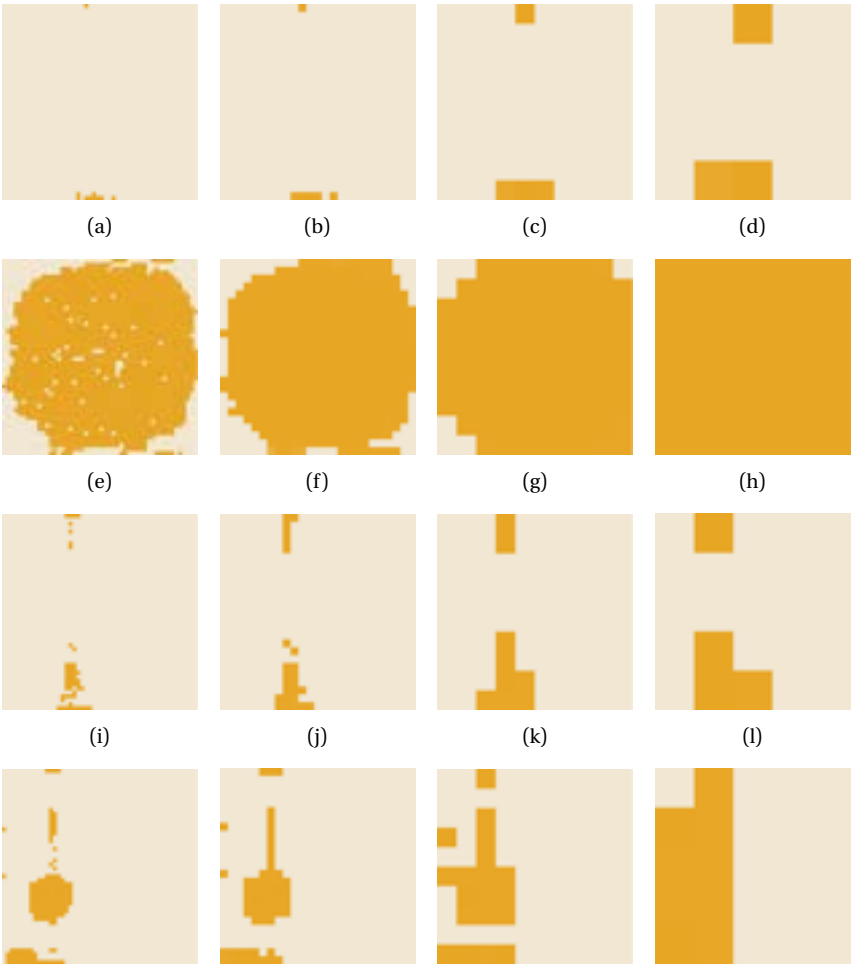


Figure F.9: PARFAIT power maps evaluated using the RFET delay model introduced in section 4.6. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

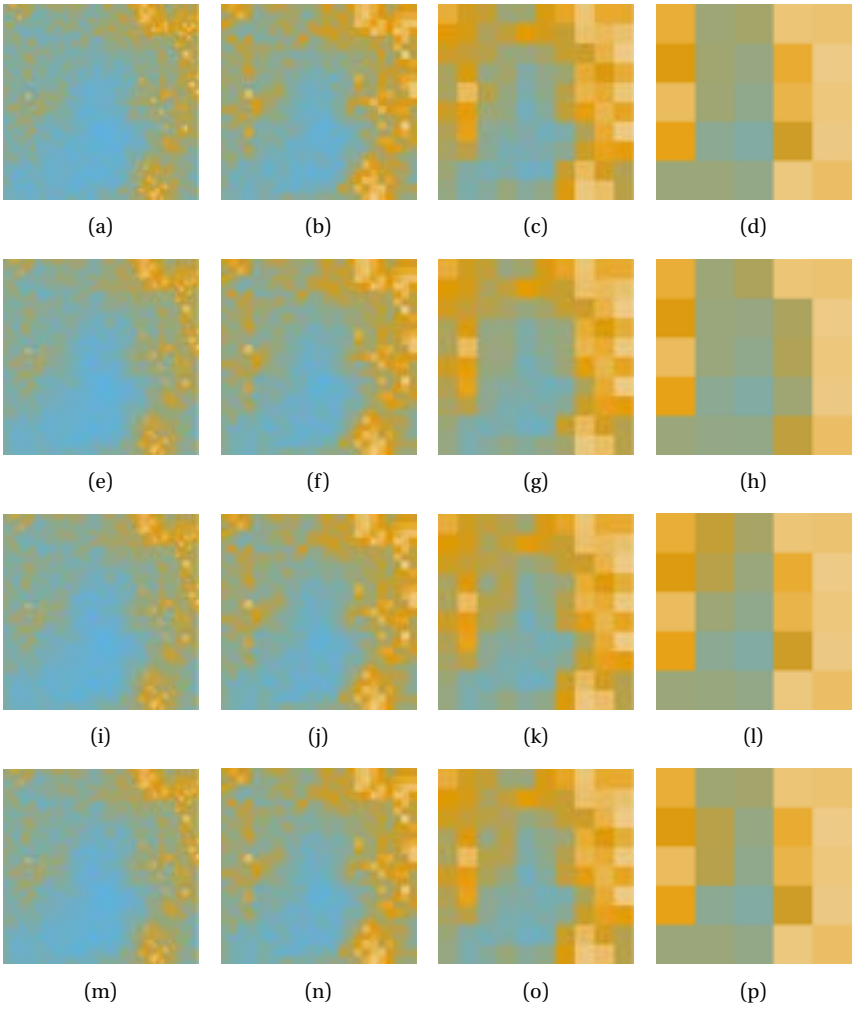


Figure F.10: PARFAIT delay factor maps evaluated using the SOI delay model introduced in section 4.6 with process variation. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

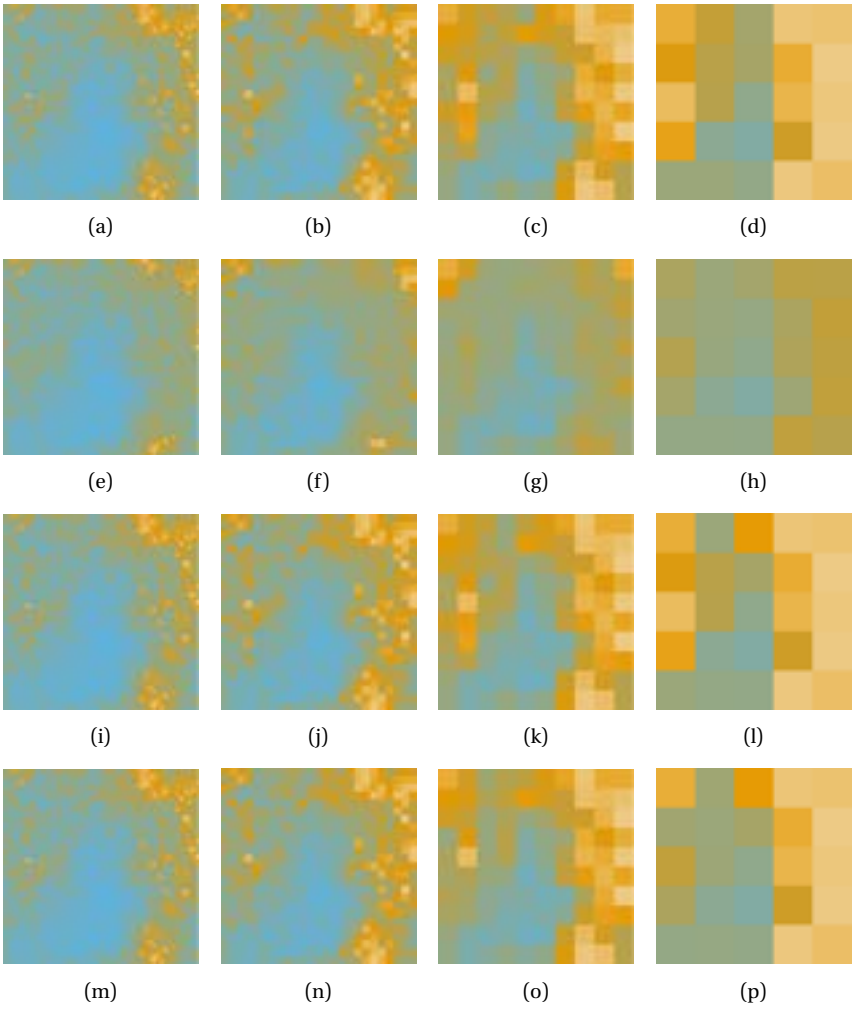


Figure F.11: PARFAIT delay factor maps evaluated using the SOI delay model introduced in section 4.6 with process variation. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

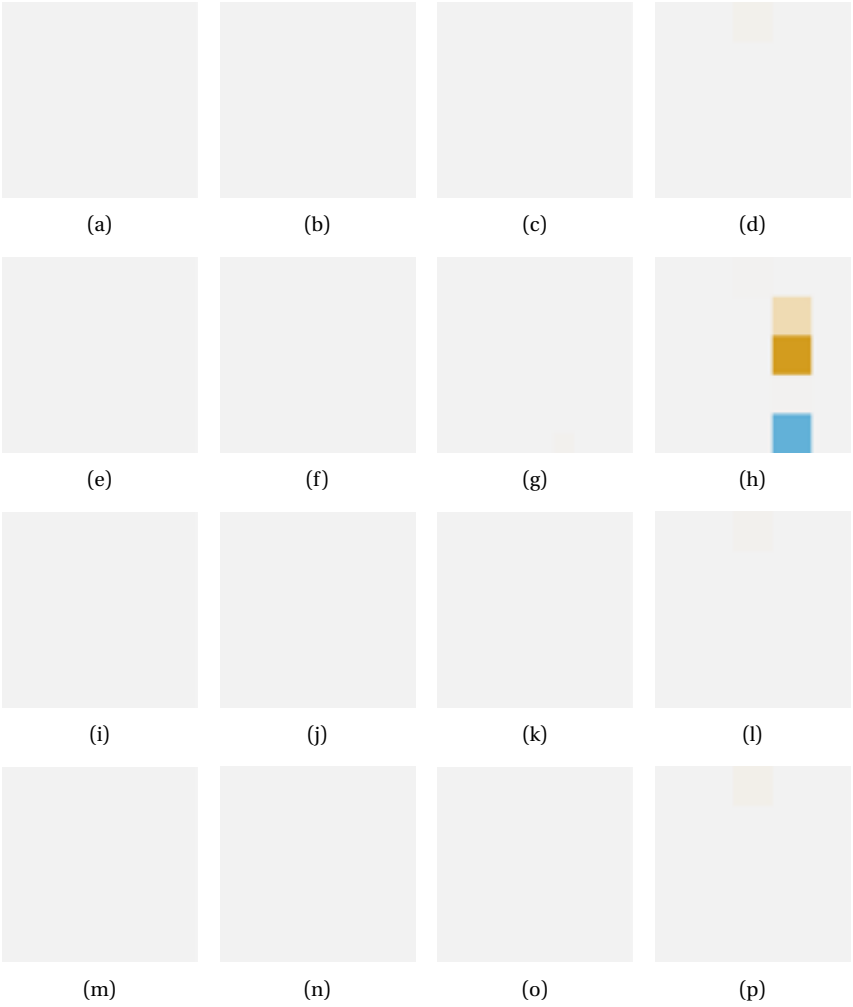


Figure F.12: PARFAIT power maps evaluated using the SOI delay model introduced in section 4.6 with process variation. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

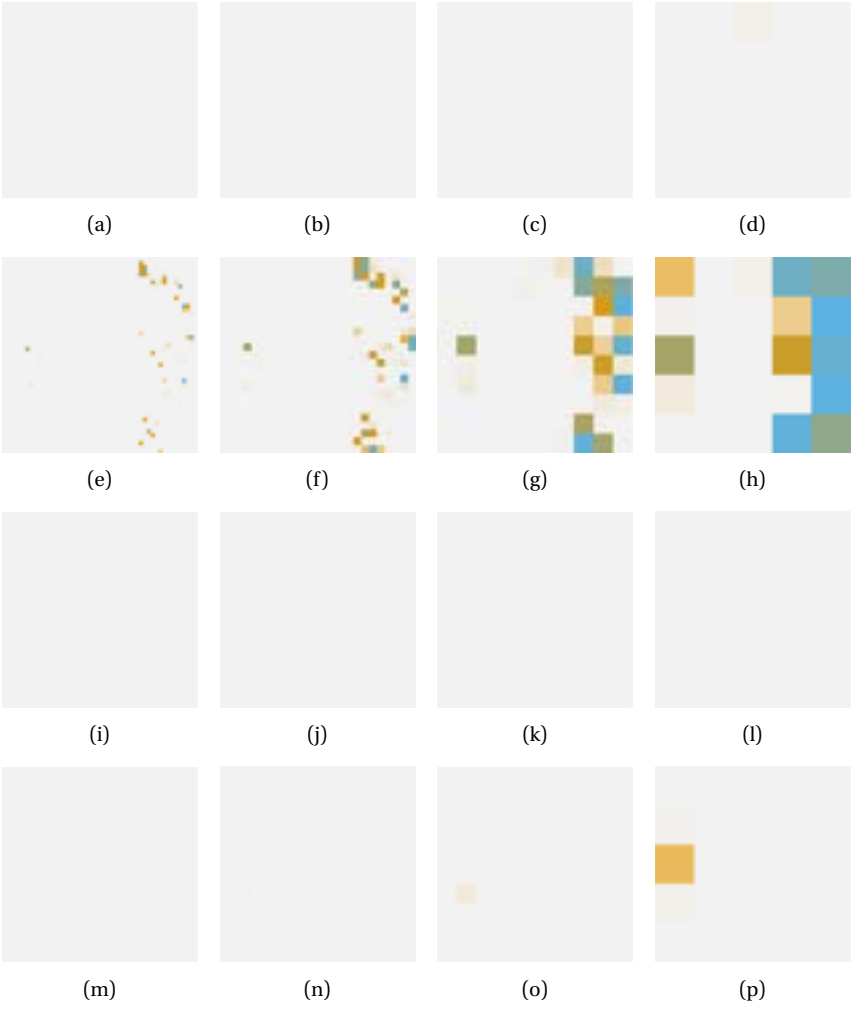


Figure F.13: PARFAIT power maps evaluated using the SOI delay model introduced in section 4.6 with process variation. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

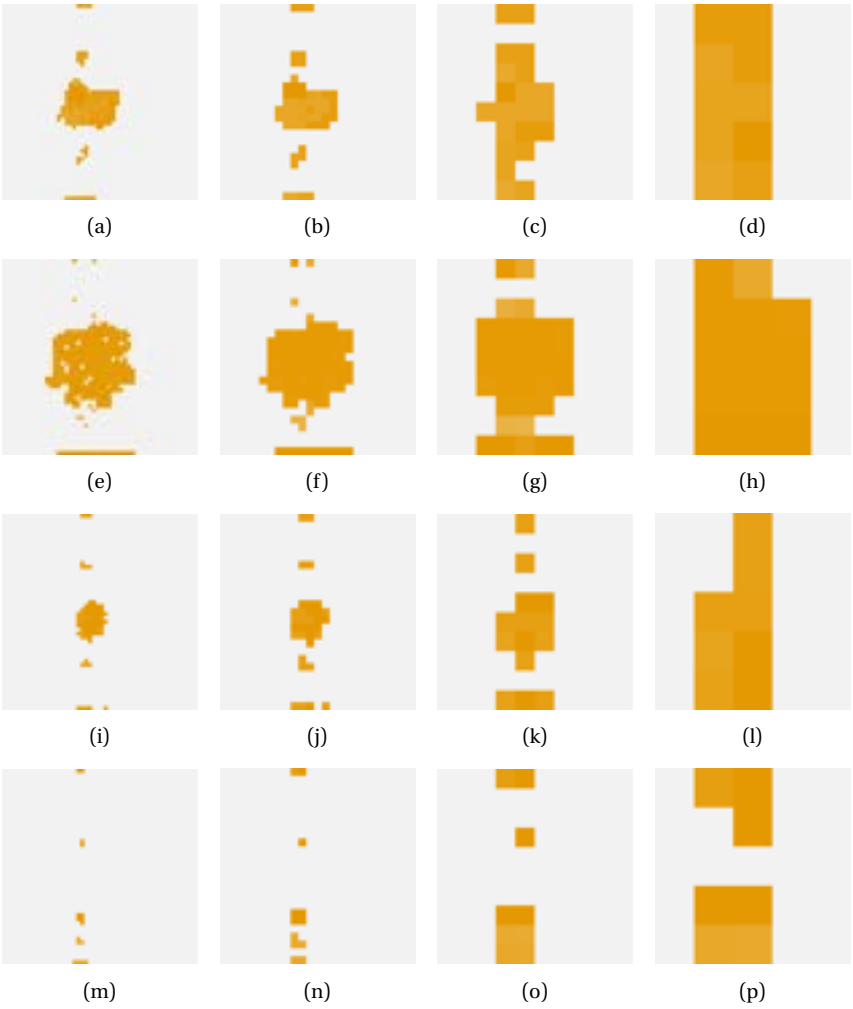


Figure F.14: PARFAIT delay factor maps evaluated using the RFET delay model introduced in section 4.6 with process variation. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

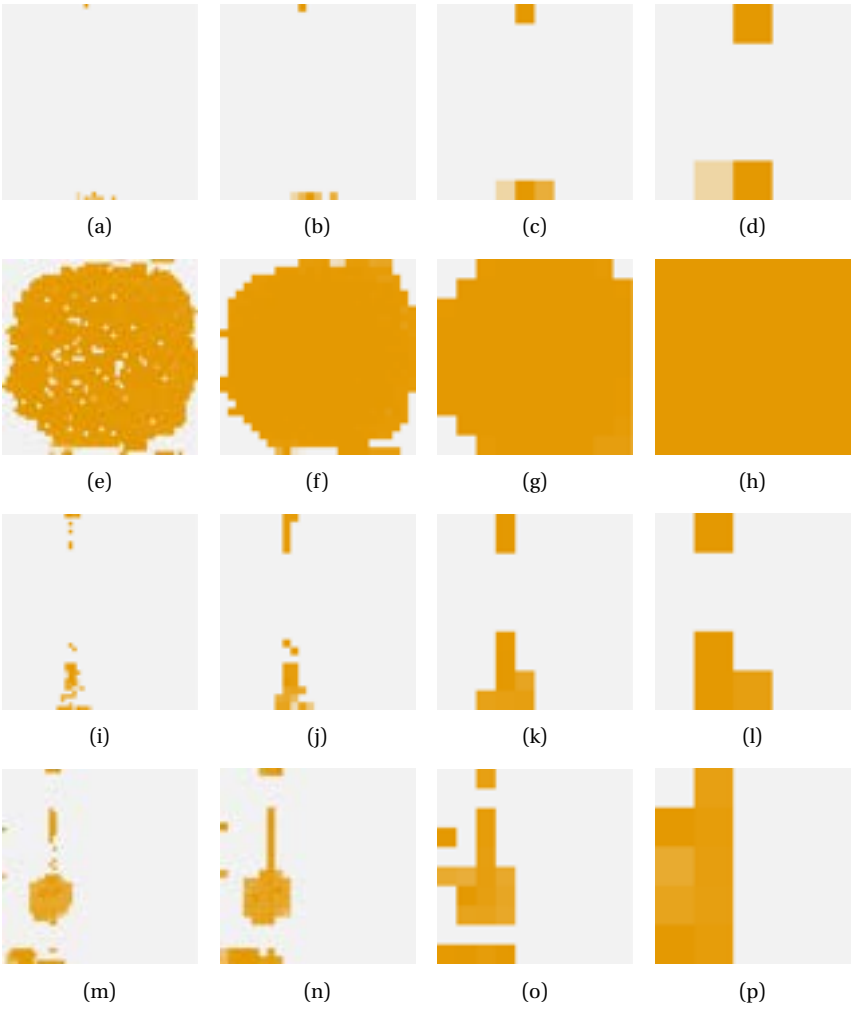


Figure F.15: PARFAIT delay factor maps evaluated using the RFET delay model introduced in section 4.6 with process variation. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

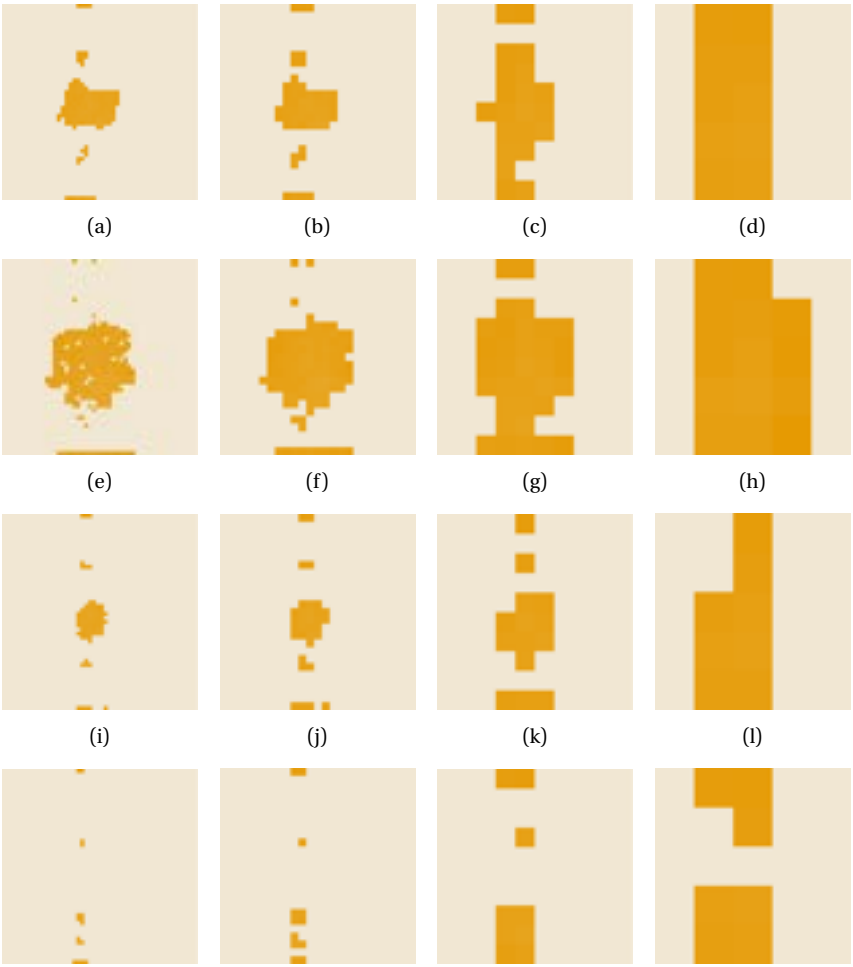


Figure F.16: PARFAIT power maps evaluated using the RFET delay model introduced in section 4.6 with process variation. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

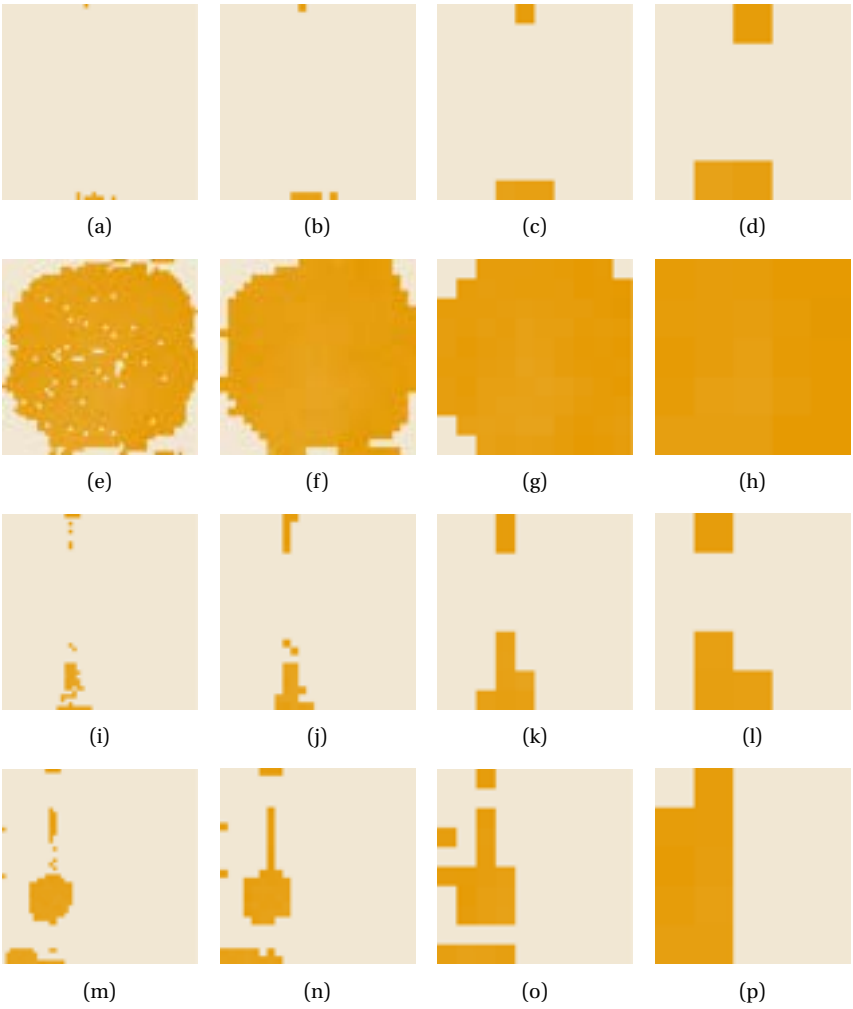


Figure F.17: PARFAIT power maps evaluated using the RFET delay model introduced in section 4.6 with process variation. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50.

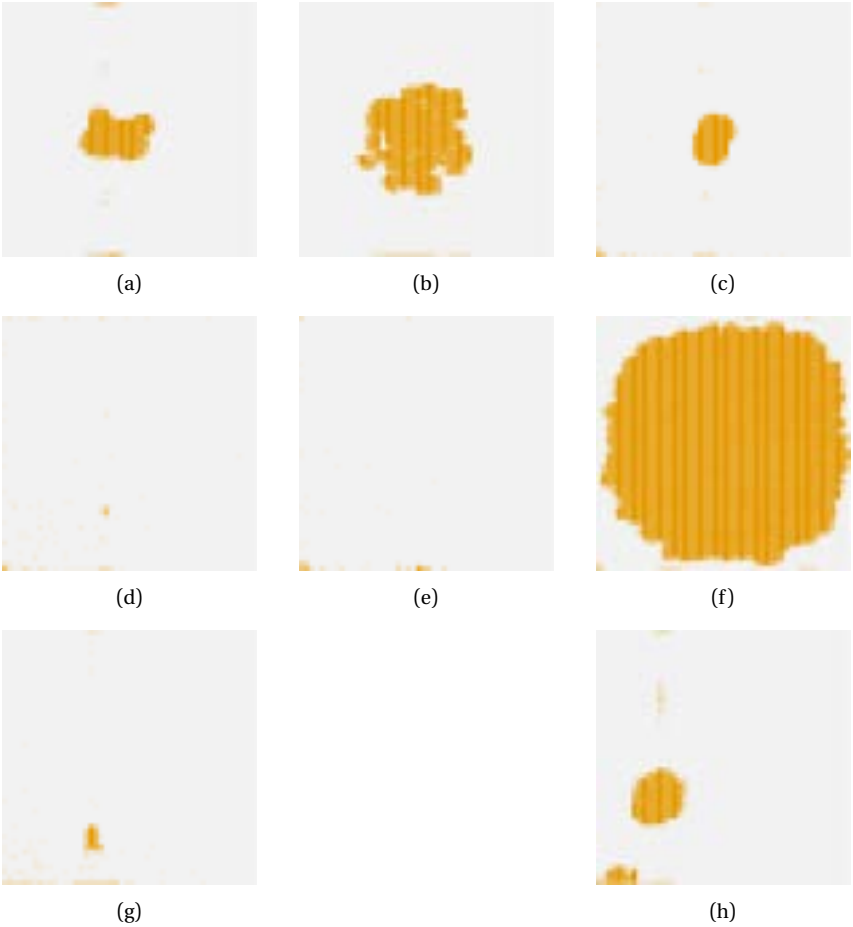


Figure F.18: Voltage Variation Maps for the benchmarks which are evaluated on the PARFAIT FPGA architecture. Placements were obtained using the ULM VPR flow introduced in section 6.2 on page 180. **(a)** to **(h)**: Benchmarks `arm_core`, `bgm`, `blob_merge`, `diffeq2`, `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B` and `stereovision0`.

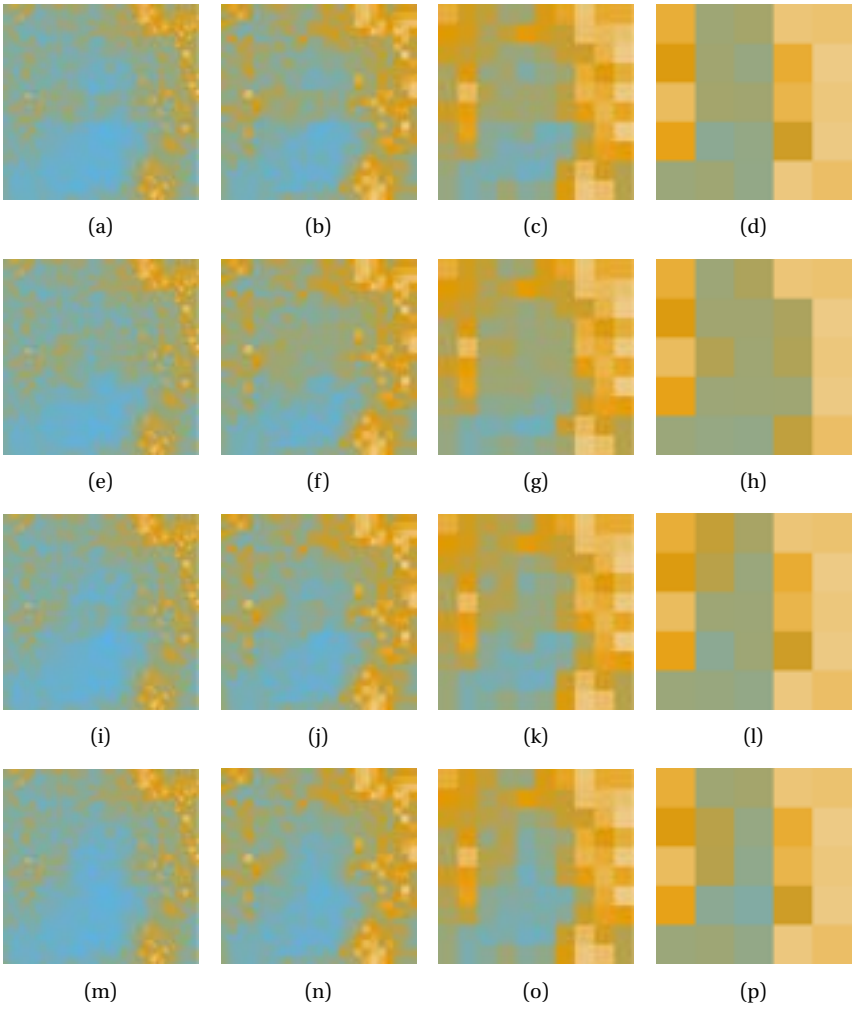


Figure F.19: PARFAIT delay factor maps evaluated using the SOI delay model introduced in section 4.6 with voltage variation. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $\epsilon = 0.1$.

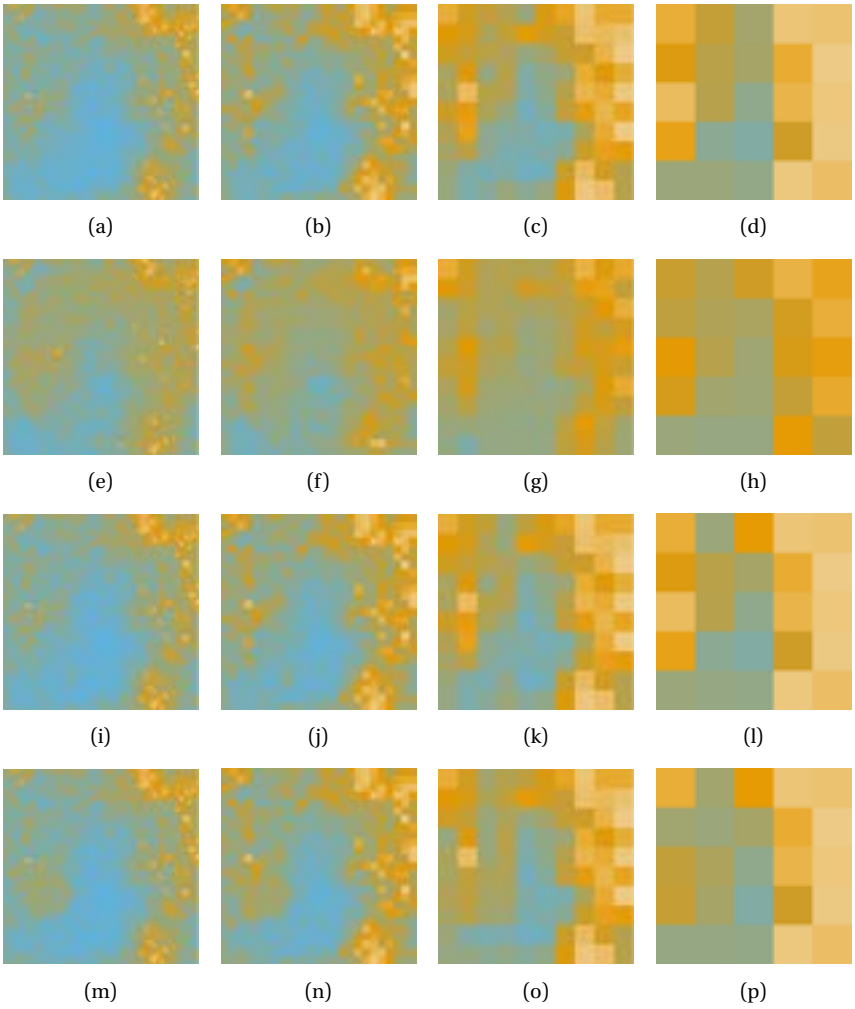


Figure F.20: PARFAIT delay factor maps evaluated using the SOI delay model introduced in section 4.6 with voltage variation. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $\epsilon = 0.1$.

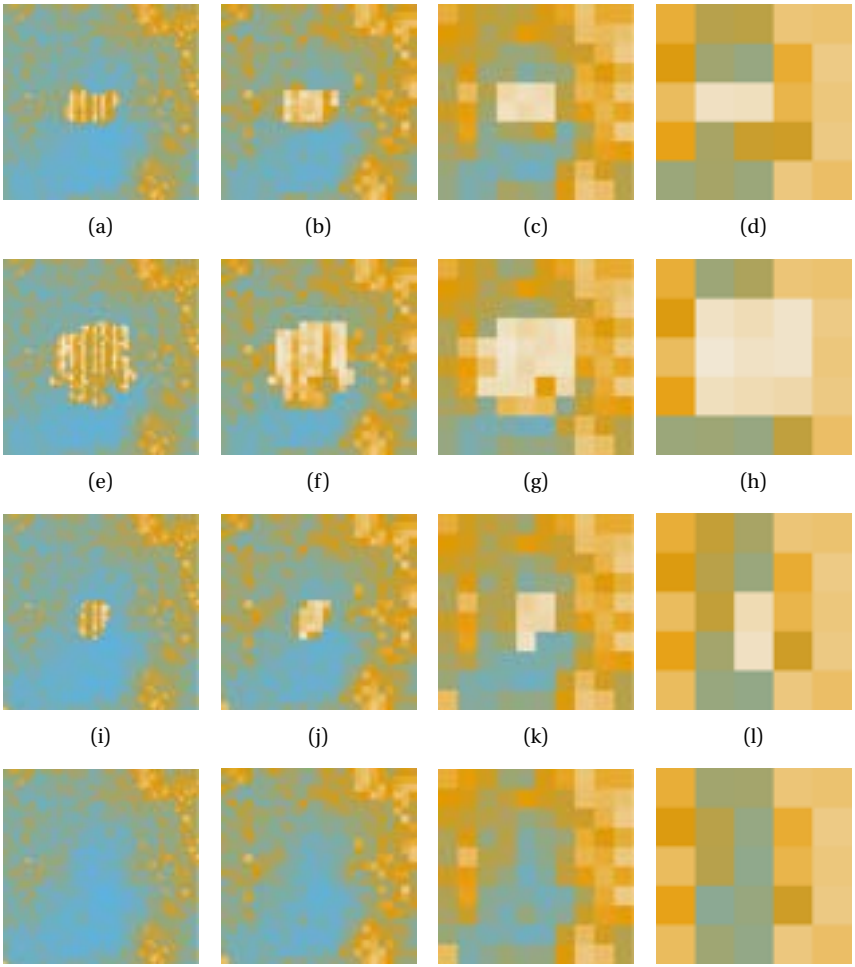


Figure F.21: PARFAIT delay factor maps evaluated using the SOI delay model introduced in section 4.6 with voltage variation. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $\epsilon = 0.3$.

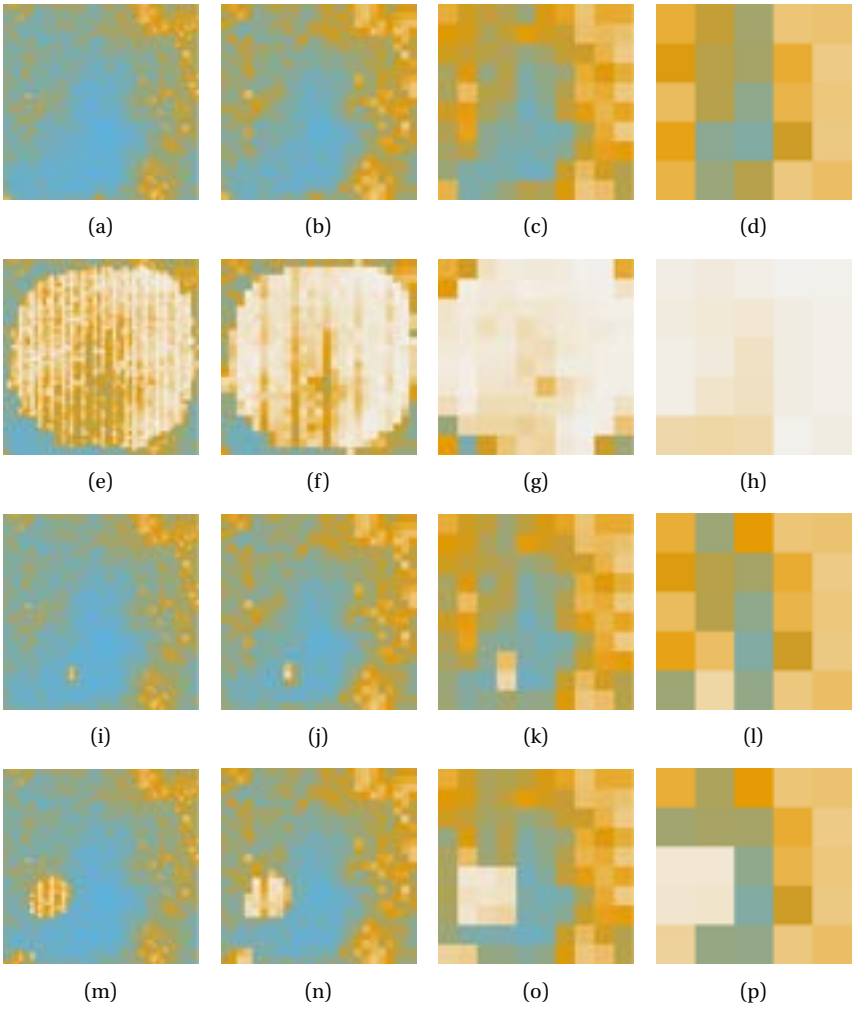


Figure F.22: PARFAIT delay factor maps evaluated using the SOI delay model introduced in section 4.6 with voltage variation. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $\epsilon = 0.3$.

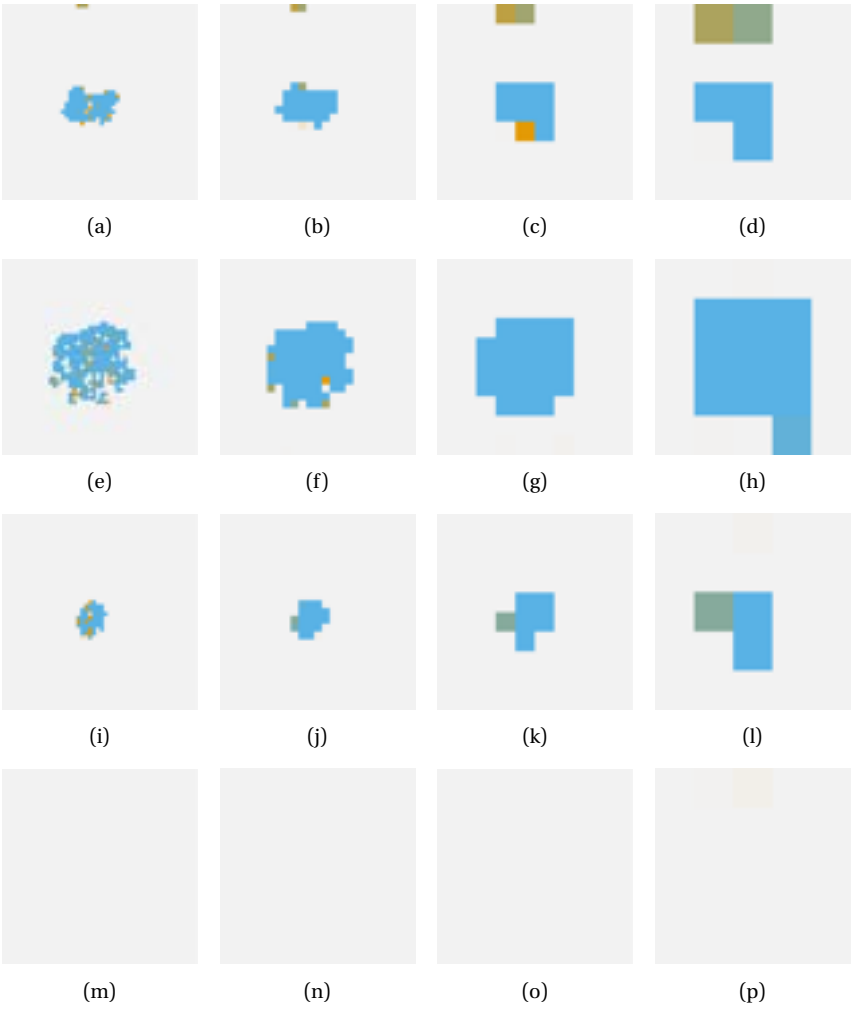


Figure F.23: PARFAIT power maps evaluated using the SOI delay model introduced in section 4.6 with voltage variation. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $\epsilon = 0.3$.

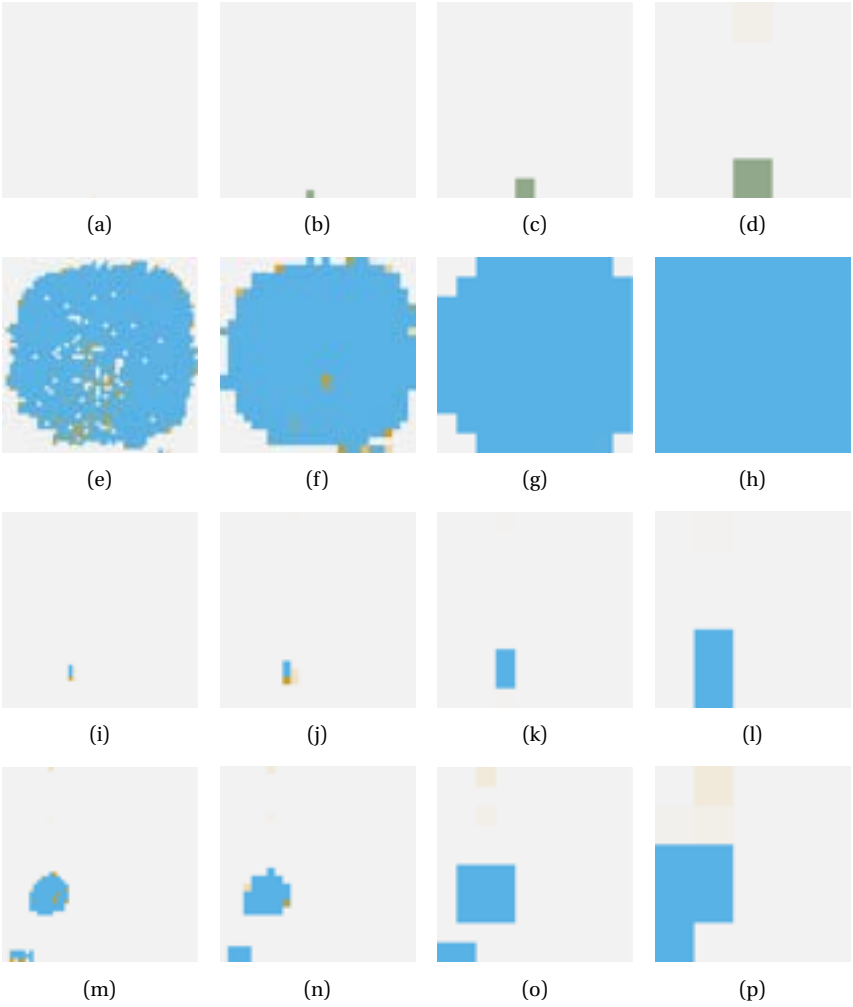


Figure F.24: PARFAIT power maps evaluated using the SOI delay model introduced in section 4.6 with voltage variation. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $\epsilon = 0.3$.

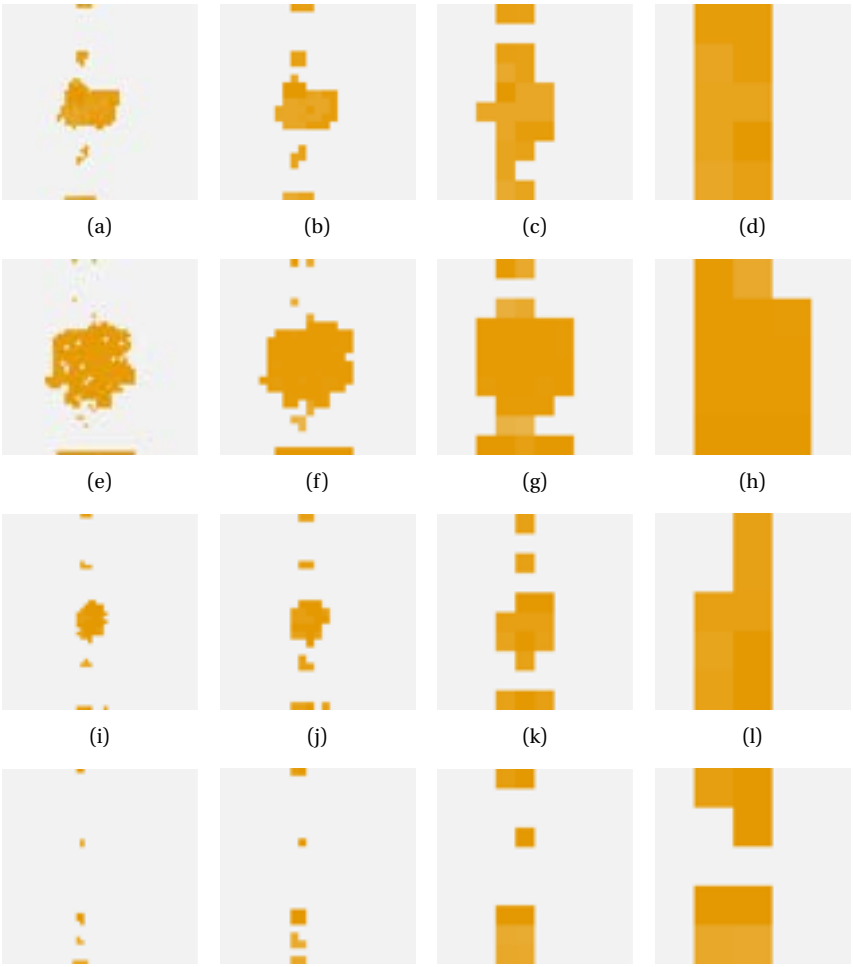


Figure F.25: PARFAIT delay factor maps evaluated using the RFET delay model introduced in section 4.6 with voltage variation. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $\epsilon = 0.1$.

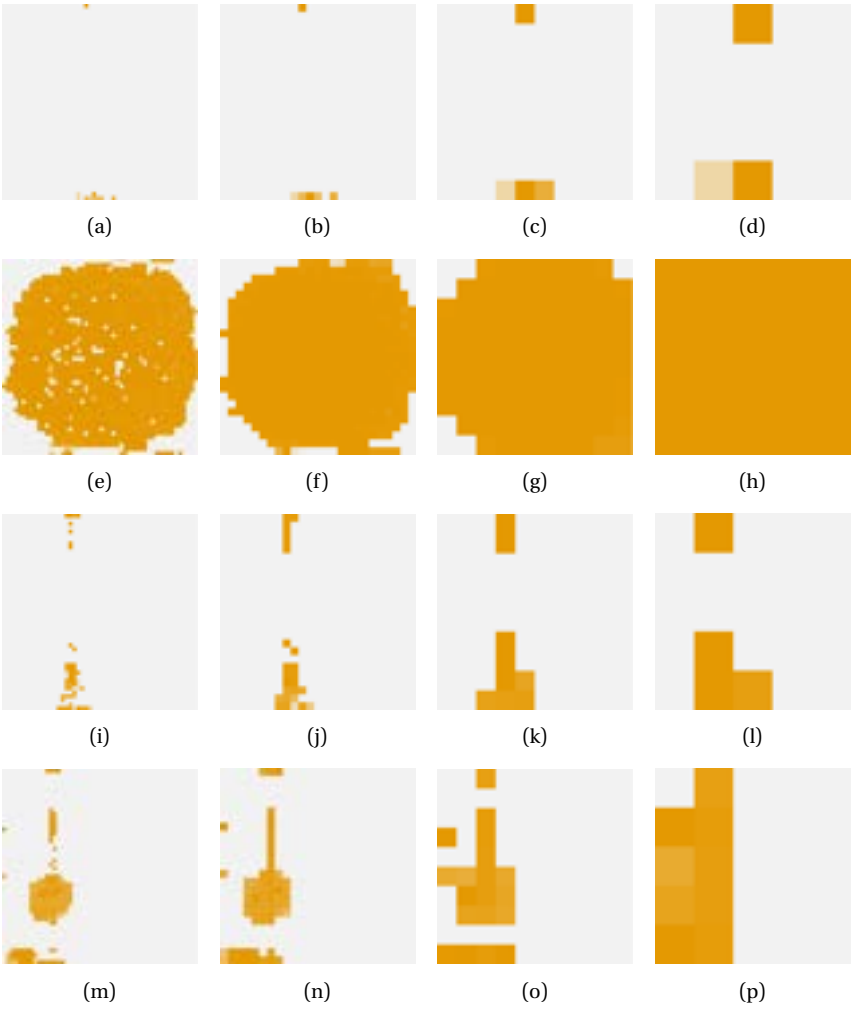


Figure F.26: PARFAIT delay factor maps evaluated using the RFET delay model introduced in section 4.6 with voltage variation. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $\epsilon = 0.1$.

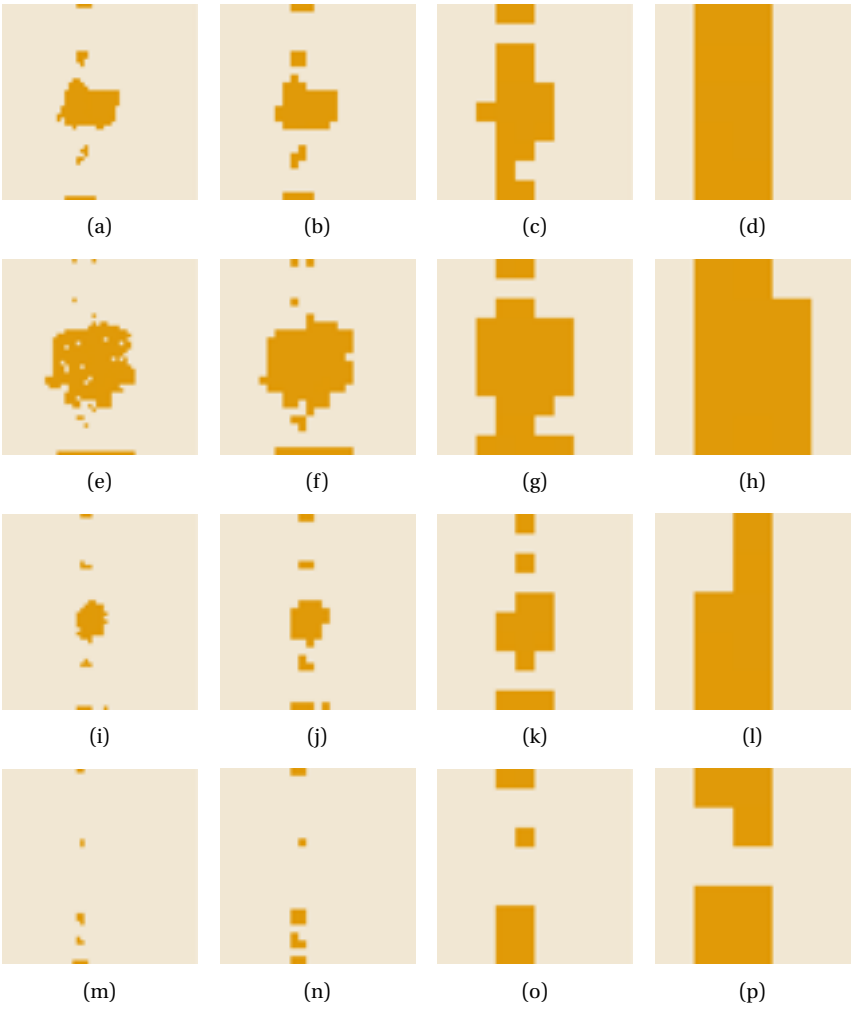


Figure F.27: PARFAIT power maps evaluated using the RFET delay model introduced in section 4.6 with voltage variation. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $\epsilon = 0.1$.

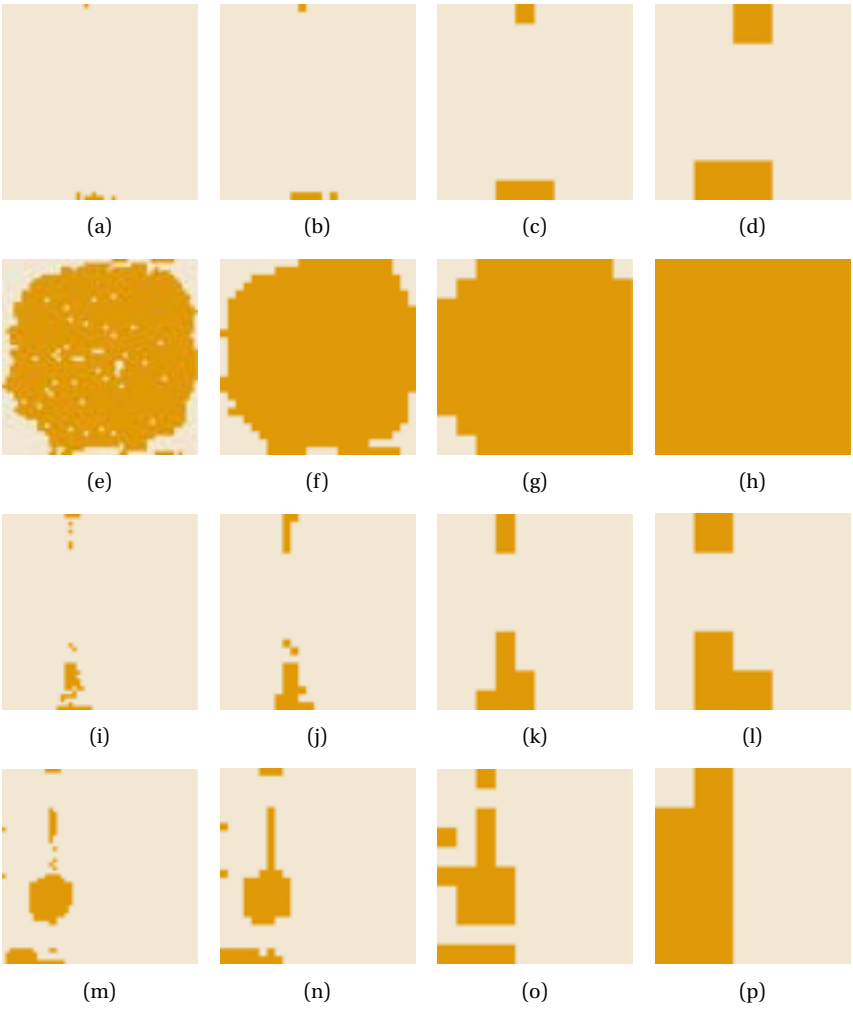


Figure F.28: PARFAIT power maps evaluated using the RFET delay model introduced in section 4.6 with voltage variation. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $\epsilon = 0.1$.

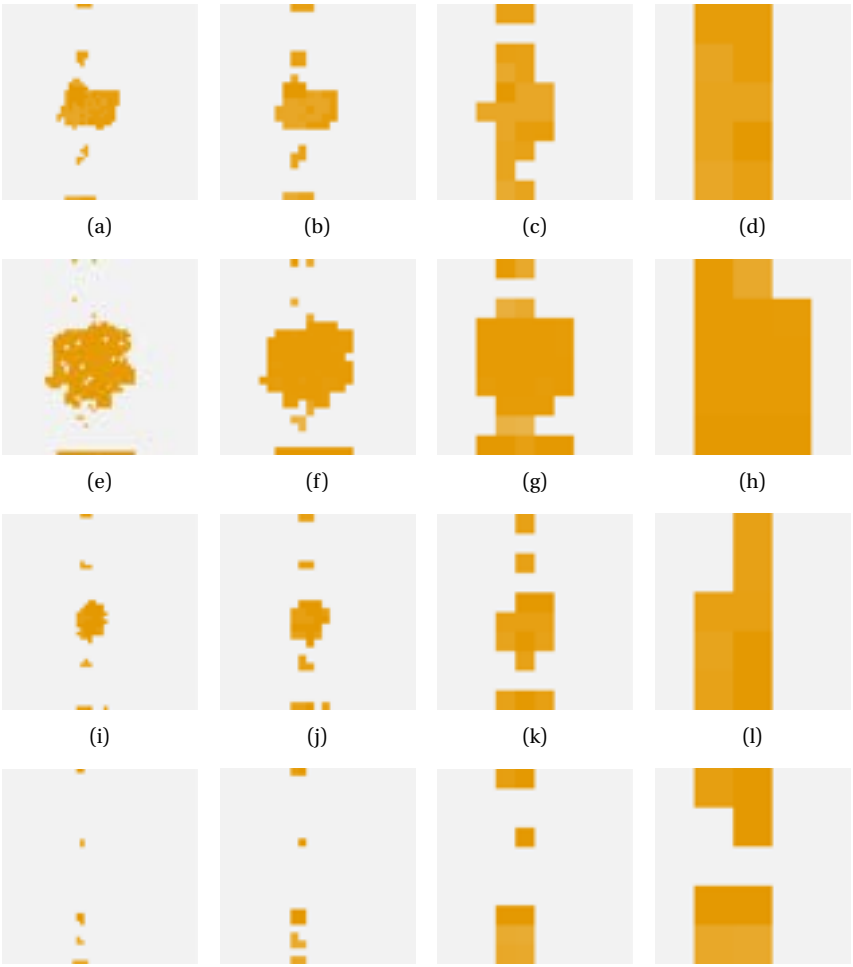


Figure F.29: PARFAIT delay factor maps evaluated using the RFET delay model introduced in section 4.6 with voltage variation. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $\epsilon = 0.3$.

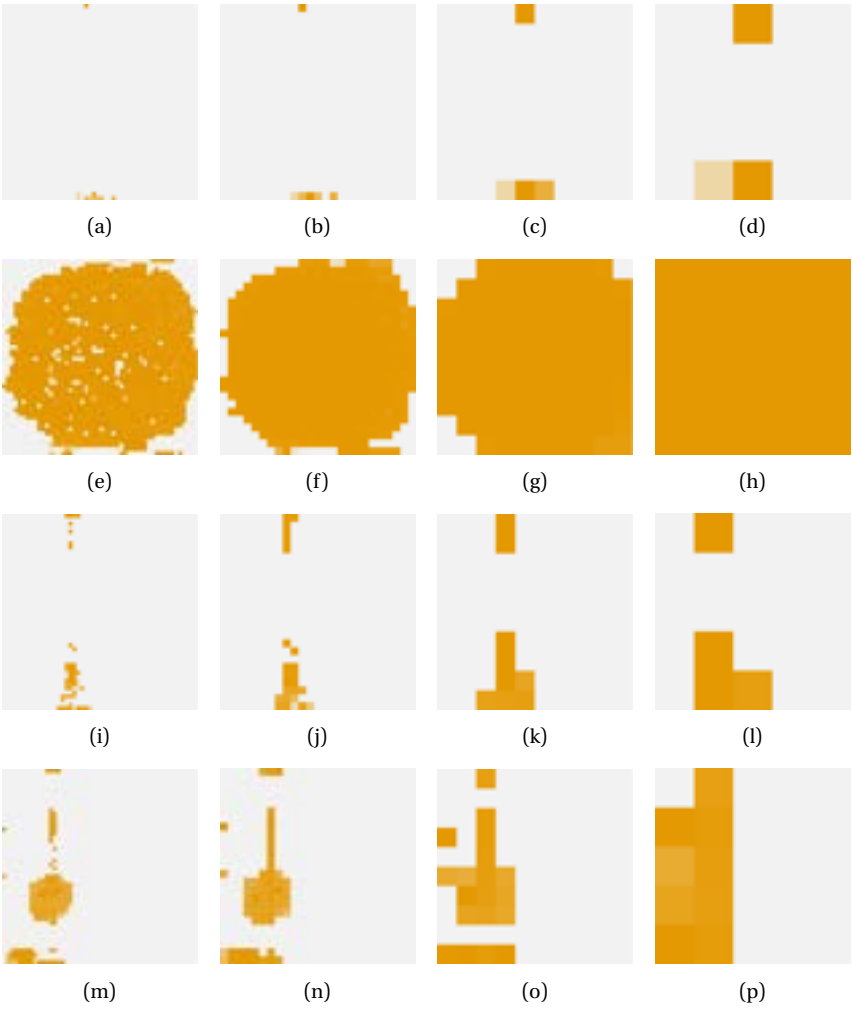


Figure F.30: PARFAIT delay factor maps evaluated using the RFET delay model introduced in section 4.6 with voltage variation. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $\epsilon = 0.3$.

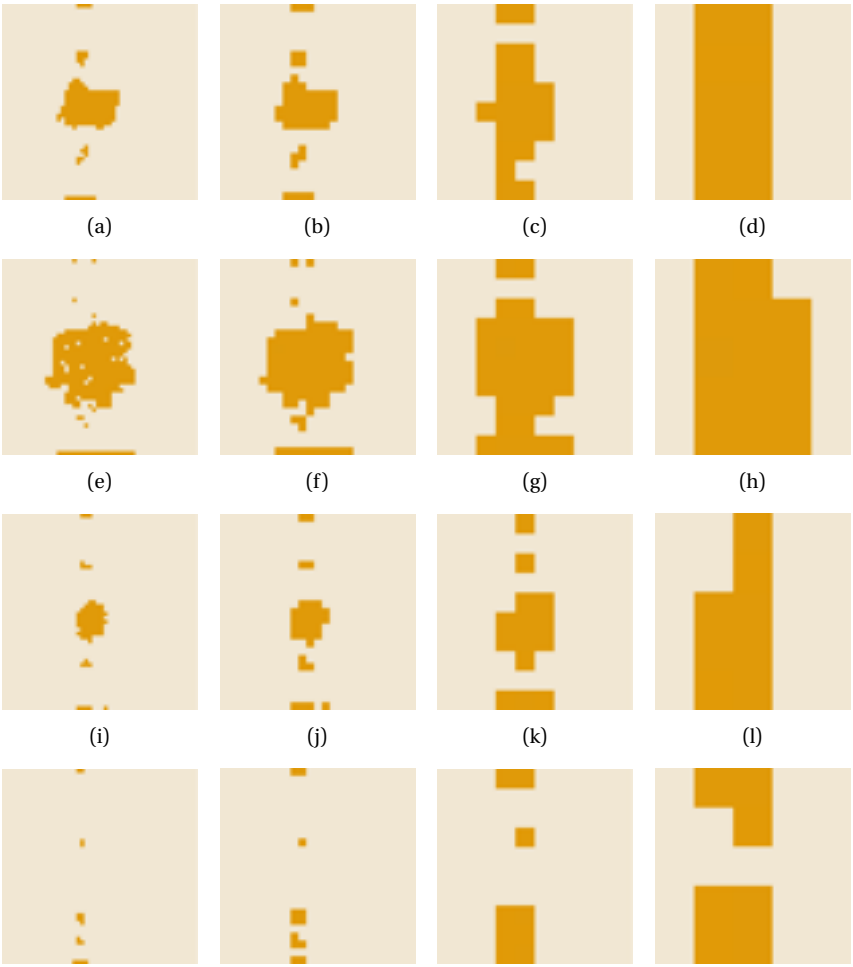


Figure F.31: PARFAIT power maps evaluated using the RFET delay model introduced in section 4.6 with voltage variation. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $\epsilon = 0.3$.

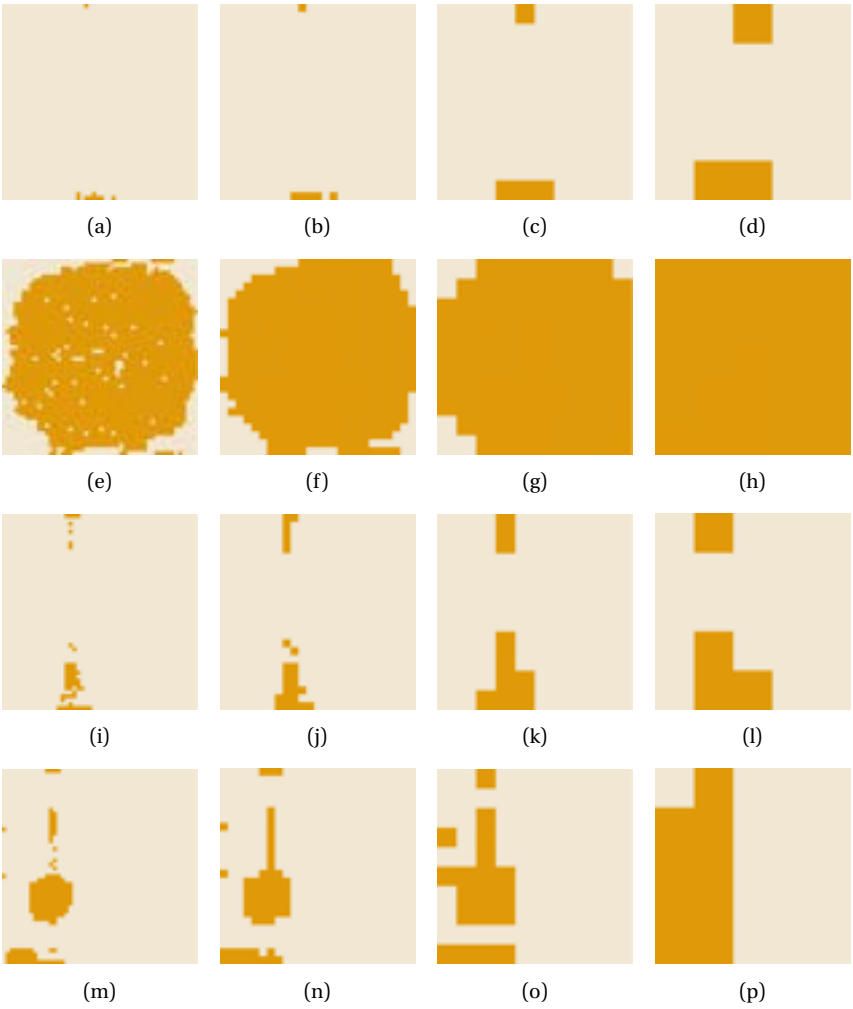


Figure F.32: PARFAIT power maps evaluated using the RFET delay model introduced in section 4.6 with voltage variation. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $\epsilon = 0.3$.

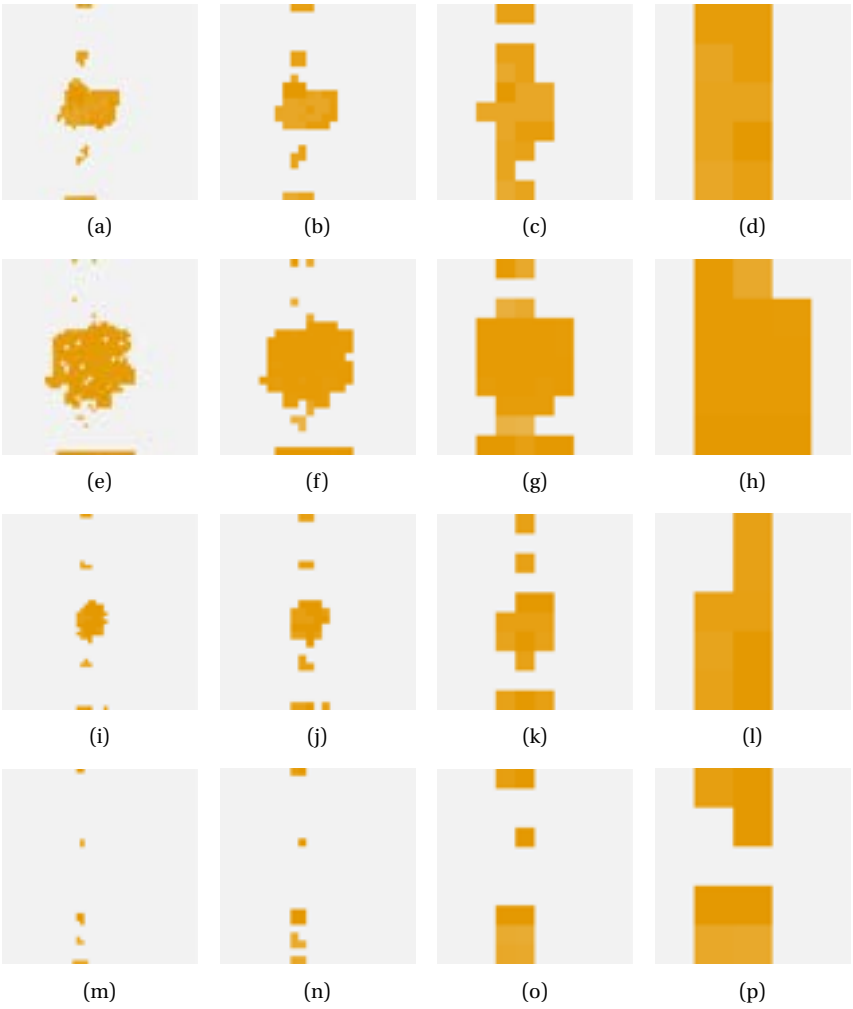


Figure F.33: PARFAIT delay factor maps evaluated using the RFET delay model introduced in section 4.6 with temperature variation. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $T = 100\,\mathrm{K}$.

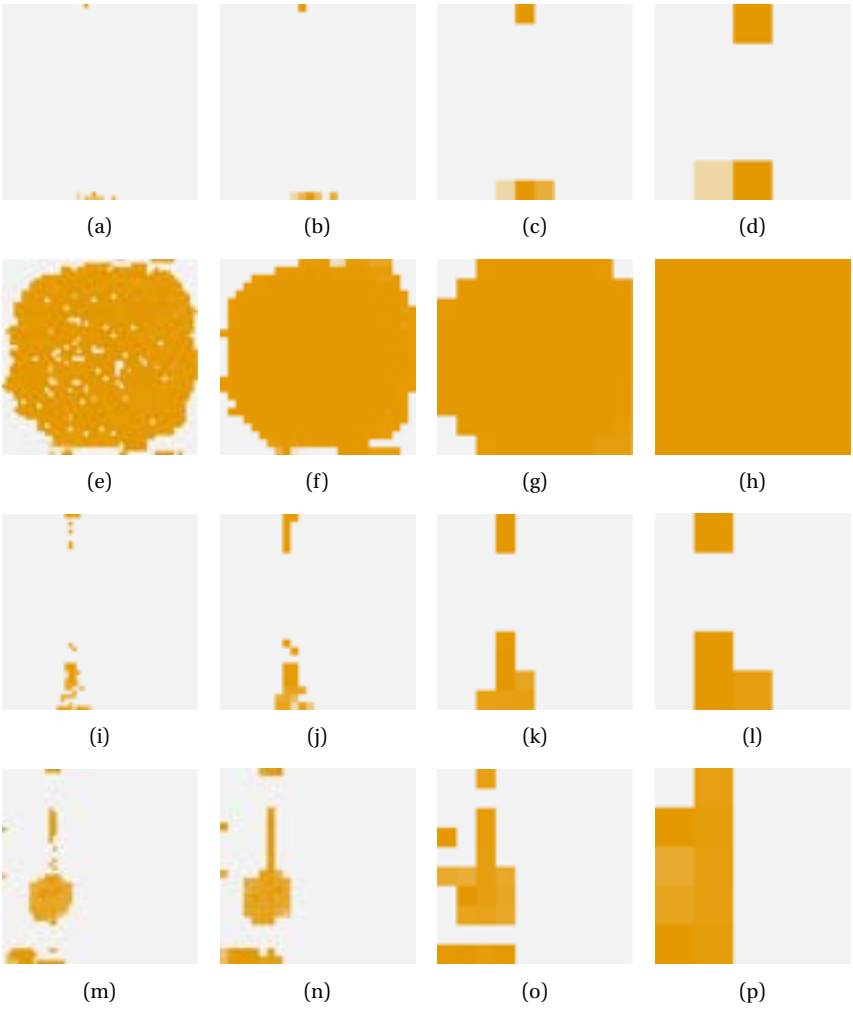


Figure F.34: PARFAIT delay factor maps evaluated using the RFET delay model introduced in section 4.6 with temperature variation. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $T = 100\,\mathrm{K}$.

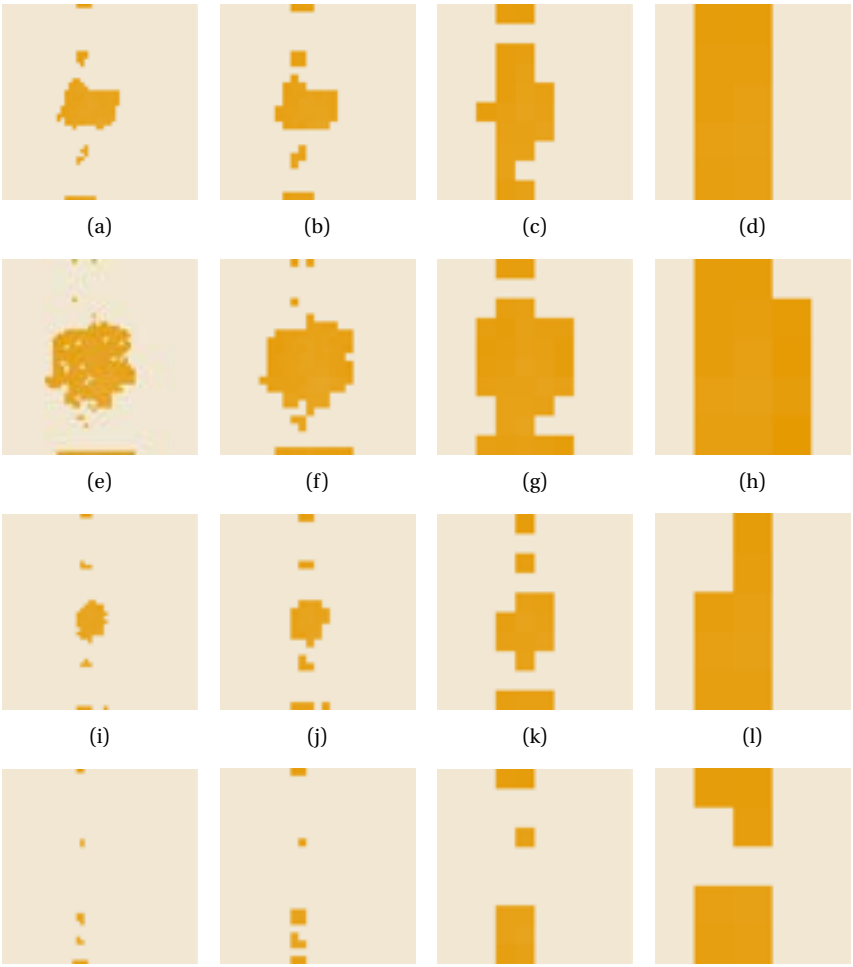


Figure F.35: PARFAIT power maps evaluated using the RFET delay model introduced in section 4.6 with temperature variation. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $T = 100\,\mathrm{K}$.

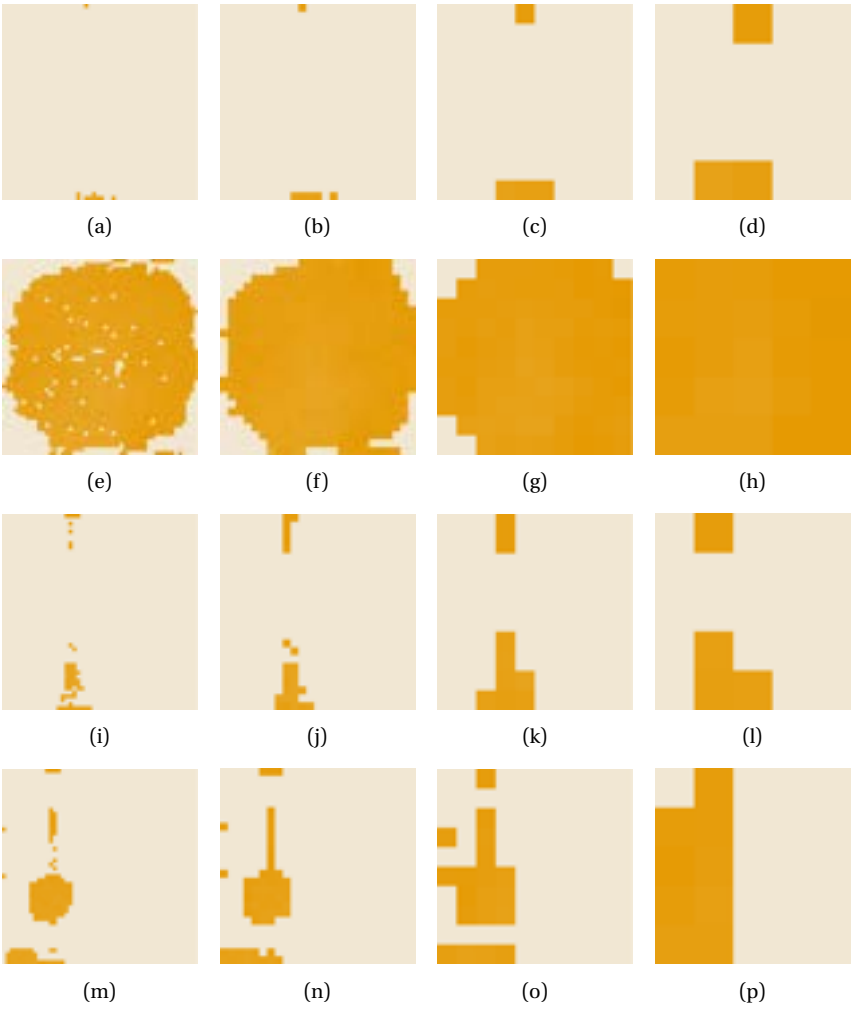


Figure F.36: PARFAIT power maps evaluated using the RFET delay model introduced in section 4.6 with temperature variation. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $T = 100\,\mathrm{K}$.

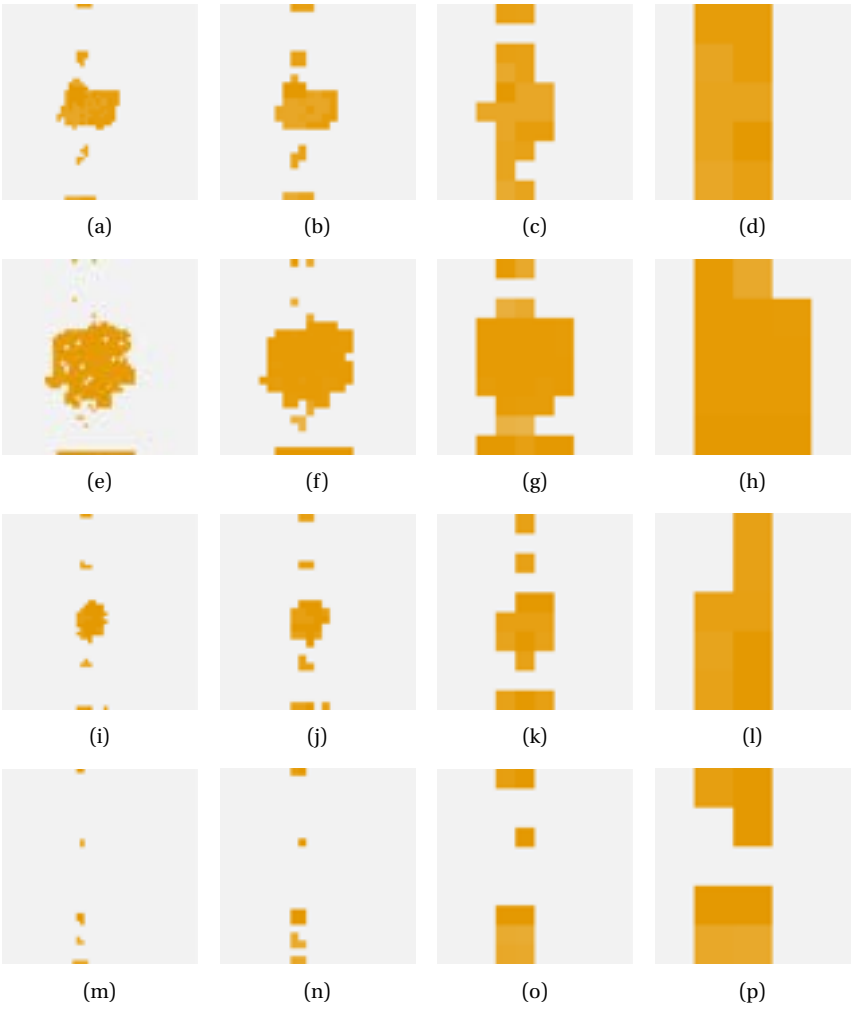


Figure F.37: PARFAIT delay factor maps evaluated using the RFET delay model introduced in section 4.6 with aging. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $t = 1$ year.

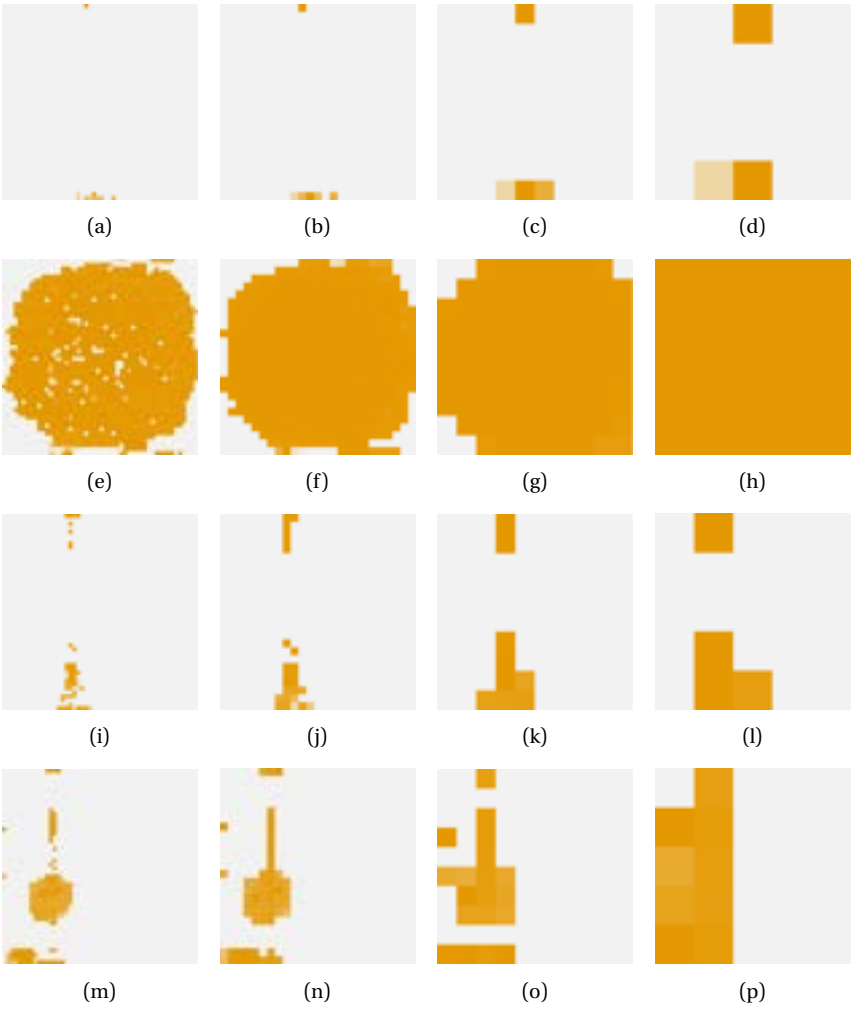


Figure F.38: PARFAIT delay factor maps evaluated using the RFET delay model introduced in section 4.6 with aging. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $t = 1$ year.

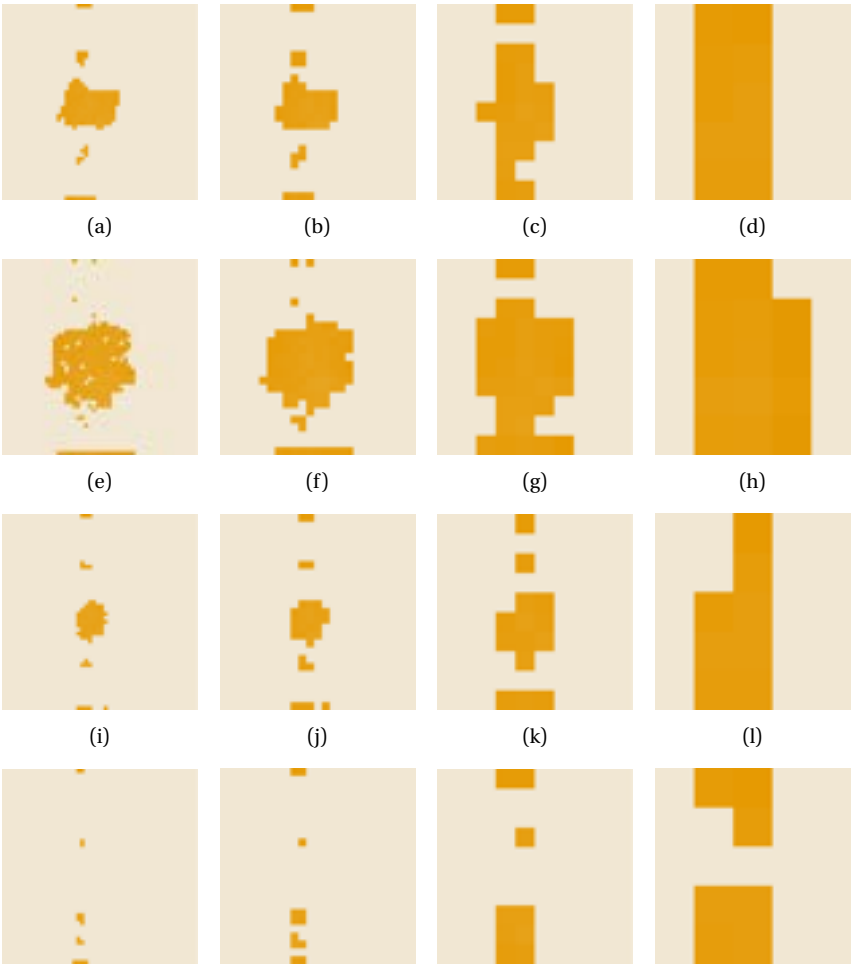


Figure F.39: PARFAIT power maps evaluated using the RFET delay model introduced in section 4.6 with aging. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $t = 1$ year.

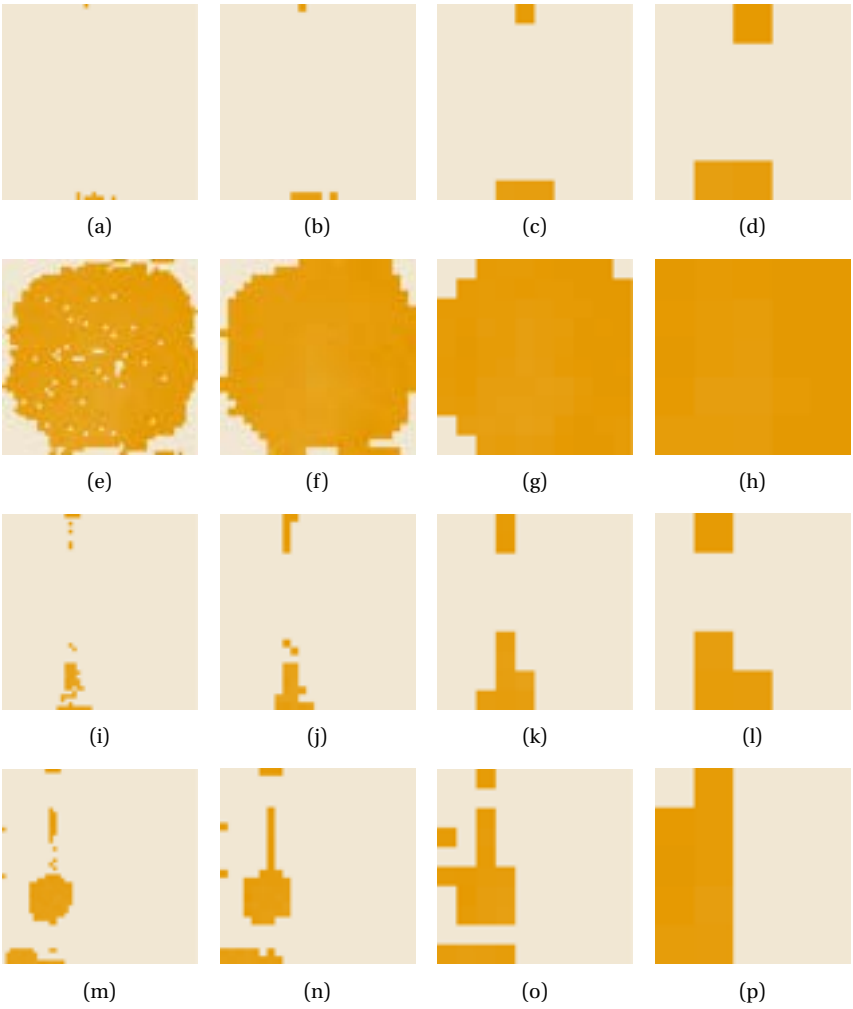


Figure F.40: PARFAIT power maps evaluated using the RFET delay model introduced in section 4.6 with aging. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $t = 1$ year.

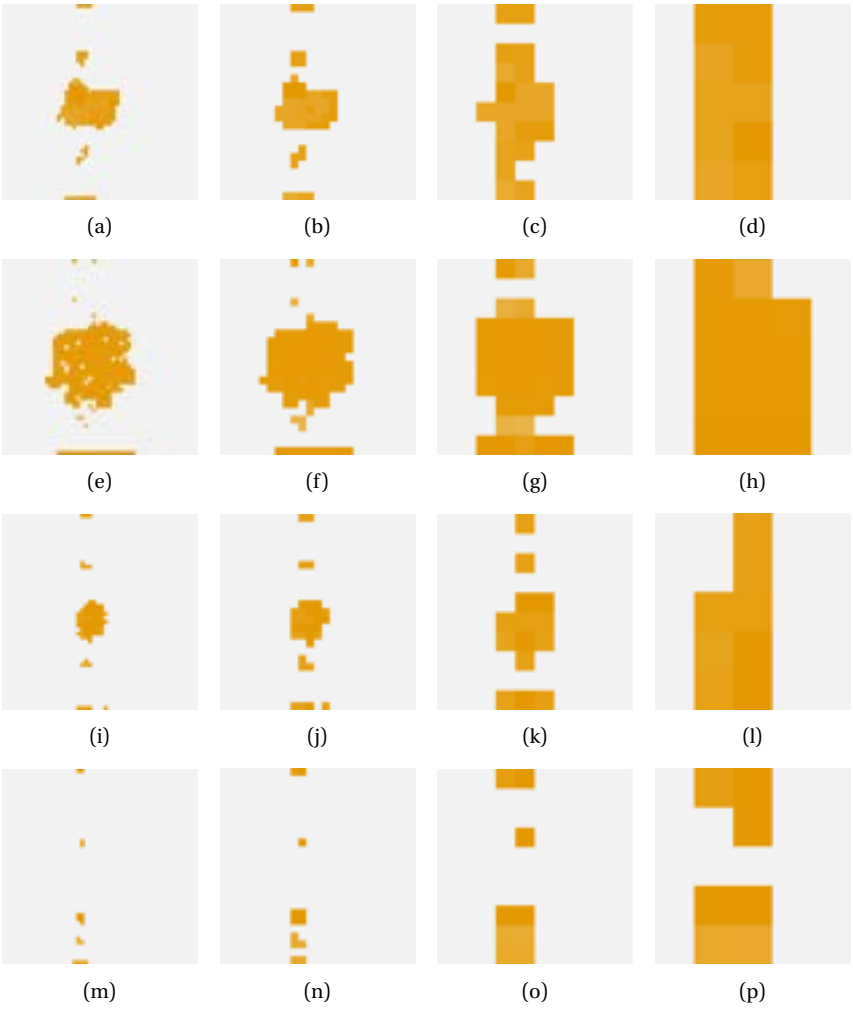


Figure F.41: PARFAIT delay factor maps evaluated using the RFET delay model introduced in section 4.6 with aging. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $t = 10$ years.

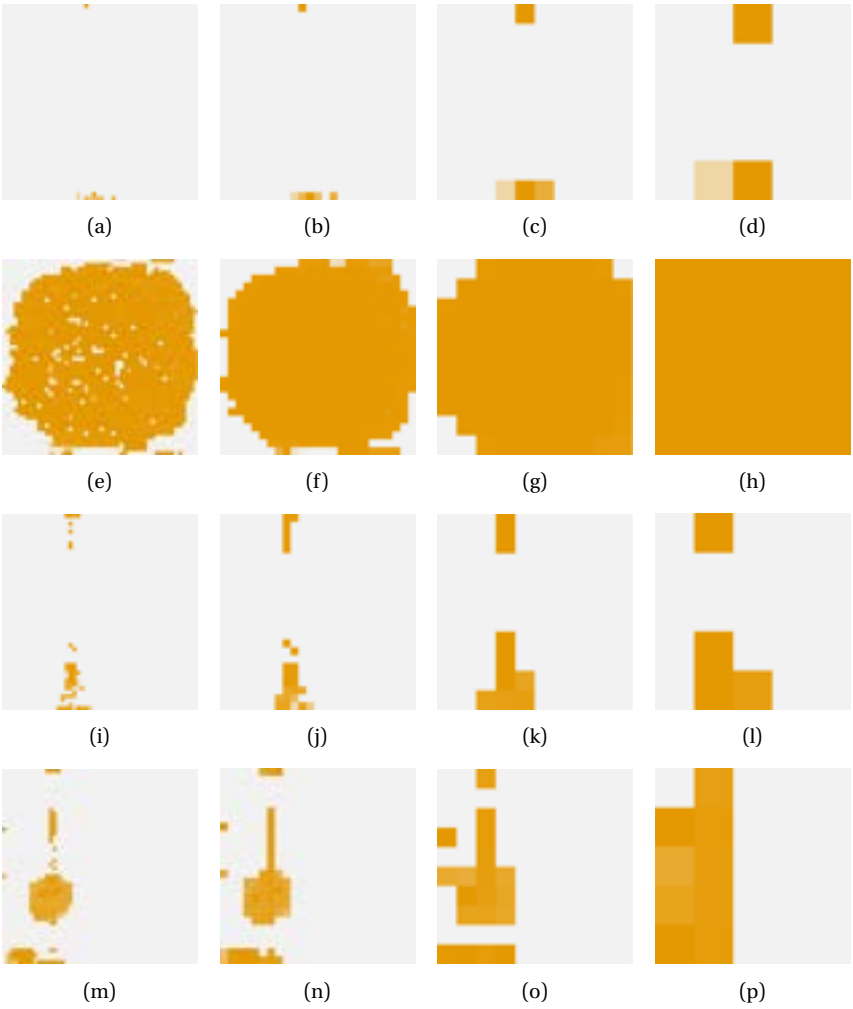


Figure F.42: PARFAIT delay factor maps evaluated using the RFET delay model introduced in section 4.6 with aging. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $t = 10$ years.

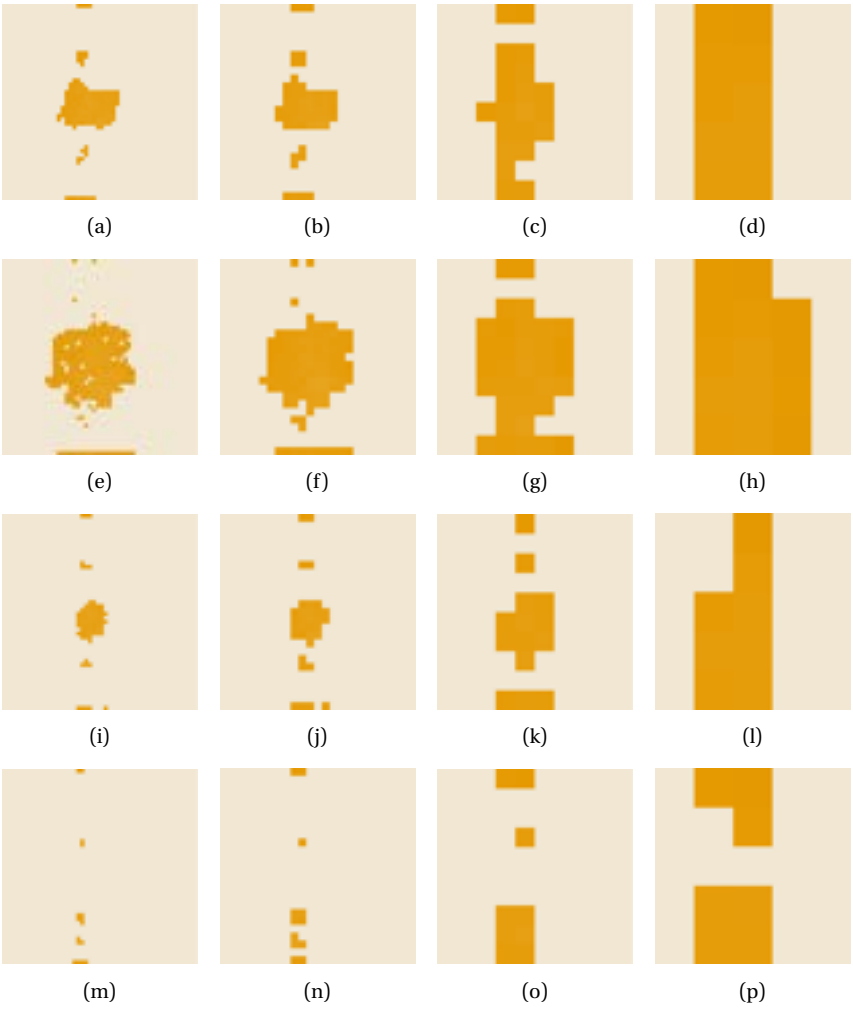


Figure F.43: PARFAIT power maps evaluated using the RFET delay model introduced in section 4.6 with aging. Benchmarks top to bottom: `arm_core`, `bgm`, `blob_merge`, `diffeq2`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $t = 10$ years.

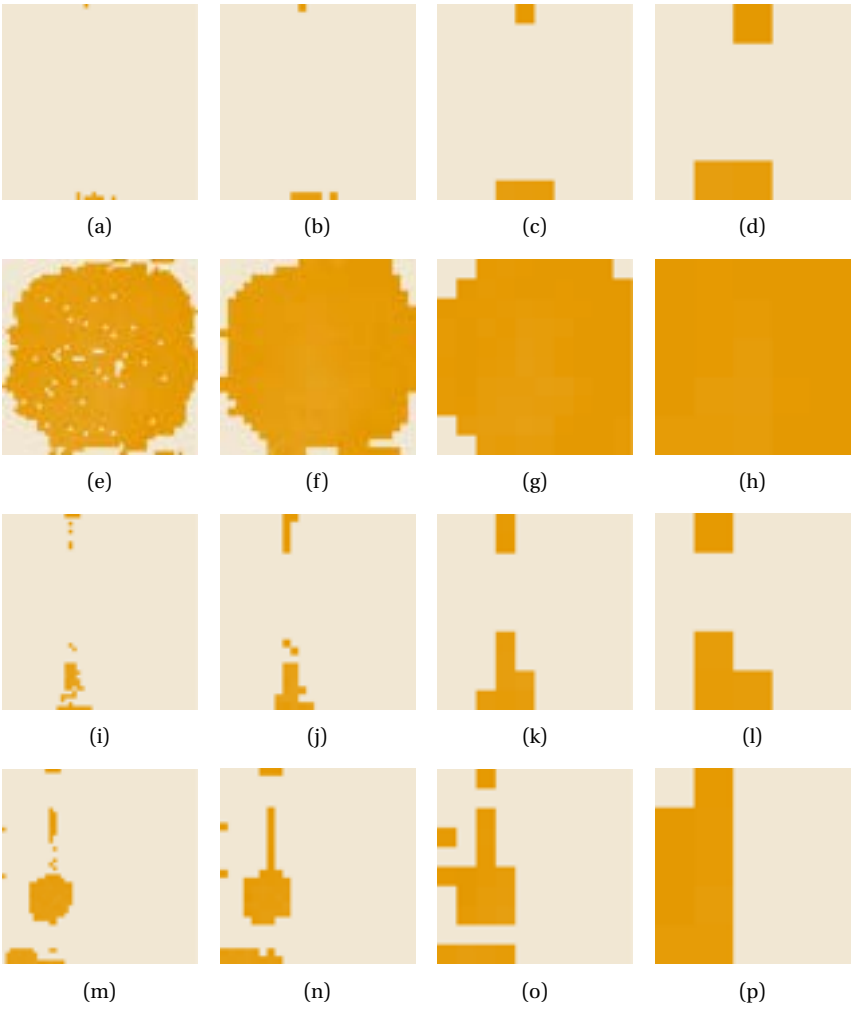


Figure F.44: PARFAIT power maps evaluated using the RFET delay model introduced in section 4.6 with aging. Benchmarks top to bottom: `ch_intrinsics`, `LU64PEEng`, `mkSMAdapter4B`, `stereovision0`. Region size left to right: 5x5, 10x10, 25x25, 50x50. $t = 10$ years.

This page intentionally left blank

Index

– R –

– S –

– T –

– U –

– V –